

**Departamento de Electrotecnia Facultad de Ingeniería**

**Universidad Nacional del Comahue**

**Medición y seguimiento de temperatura de transformadores de distribución eléctrica**

CM-ING-2129

**Dirigida por el Ing. Carlos Eduardo Canal**

**Codirigida por Rodolfo Rafael Medina**

**Ingeniería Electrónica Neuquén, 2 de M de 2025**

# Resumen

El trabajo a continuación abarca el análisis y estudio de la importancia, utilidad e impacto de la necesidad de monitorear la temperatura de trabajo de los transformadores de distribución eléctrica. Basado en las normas de la Comisión Electrotécnica Internacional (IEC) se plantea el desarrollo con la utilización de una placa micropocesada ESP32 S3. Se presentan los distintos diseños de plataformas para llevar al usuario la accesibilidad de los datos en tiempo real. Tras análisis y ensayos se planea una posible estrategia de implementación inicial en el campo.

Este proyecto se enmarca en la necesidad de monitorear la temperatura de los transformadores de la zona.

# Abstract

**Índice**

Contenido

[Resumen 2](#_Toc212639991)

[Abstract 4](#_Toc212639992)

[Capítulo1 Introducciónyobjetivos 6](#_Toc212639993)

[1.1. Contextoyfundamentación 6](#_Toc212639994)

[1.2. Objetivos 7](#_Toc212639995)

[1.3. Organización deltrabajo 8](#_Toc212639996)

[Capítulo 2 Teoría y definiciones 9](#_Toc212639997)

[2.1. Transformadores de distribución 9](#_Toc212639998)

[2.1.1. Transformadores 9](#_Toc212639999)

[2.1.2. Temperatura según IEC 10](#_Toc212640000)

[2.2 Hardware utilizado 11](#_Toc212640001)

[2.2.1 ESP32 12](#_Toc212640002)

[2.2.2 Sensor de Temperatura PT100 13](#_Toc212640003)

[2.2.3 Sensor de Temperatura DS18B20 14](#_Toc212640004)

[2.2.4 RS485 14](#_Toc212640005)

[2.2.5 Modulo GPRS 14](#_Toc212640006)

[2.3 Conceptos IoT 15](#_Toc212640007)

[Capítulo 3 Programación y desarrollo 20](#_Toc212640008)

[3.1.1. Método Things board 20](#_Toc212640009)

[3.1.2. Método a través de wifi ESP32 a IoT 27](#_Toc212640010)

[3.2. Método Modbus con ESP32 29](#_Toc212640011)

3.3 Método B.L.E. (Bluetooth Low Energy)

[Capítulo 4 Ensayos y resultados 35](#_Toc212640012)

[4.1. Pruebaspreliminares 35](#_Toc212640013)

[4.2. Propuestadeabordajeencampo 39](#_Toc212640014)

[4.2.1. Requisitosmínimos 39](#_Toc212640015)

[Capítulo 5 Conclusiones 41](#_Toc212640016)

[5.1. Resumendeltrabajorealizado 41](#_Toc212640017)

[5.2. Conclusiones 42](#_Toc212640018)

[5.3. Cierredeltrabajoypropuestasdemejora 43](#_Toc212640019)

[Bibliografía 46](#_Toc212640020)

# Capítulo 1 - Introducción y objetivos

El presente capítulo tiene por objetivo establecer el contexto y la línea de trabajo del proyecto integrador. La primera sección detalla los fundamentos que lideraron a la elaboración de este trabajo. Luego, la segunda sección plantea los objetivos del proyecto integrador, y la tercera describe la estructura que se seguirá a lo largo de todo el escrito para su desarrollo.

## Contexto y fundamentación

Las normas de la Comisión Electrotécnica Internacional (IEC), contemplan documentos para asegurar la seguridad, calidad, fiabilidad y compatibilidad de los productos eléctricos, electrónicos y tecnologías relacionadas a nivel mundial. Estas normas establecen requisitos para el diseño, fabricación, prueba y certificación de aparatos, ayudando a garantizar que funcionen de manera segura y eficiente. Entre ellas la norma IEC 60076 especifica los límites máximos de temperatura para los devanados y el aceite de los transformadores de potencia sumergidos en líquido. Define la temperatura máxima alcanzada cuando el transformador opera a su potencia asignada para garantizar un funcionamiento seguro y una vida útil prolongada.

Las protecciones comunes para transformadores de estaciones eléctricas suelen ser fusibles o cortacircuitos en la entrada, pararrayos para proteger contra sobretensiones, válvulas de alivio de presión (en transformadores en aceite) e interruptores térmicos o sensores de temperatura.

Este proyecto integrador de basa en la norma antes mencionada y en la necesidad del Ente Provincial de Energía de Neuquén (EPEN) para monitorear la temperatura interna de un transformador de distribución a modo de prevenir rupturas en los equipos en funcionamiento. Dada la naturaleza de las instalaciones y o condiciones de las instalaciones de los transformadores de distribución en la zona, no se cuenta con protecciones ni formas de monitorear la temperatura. Por lo que se idearon distintas las aplicaciones que se adaptan a los entornos donde puedan estas ubicados estos equipos.

* Entorno con disponibilidad de WIFI
* Entorno con disponibilidad de red interna
* Entorno sin redes disponibles

## Objetivos

Los objetivos que se plantearon para este trabajo son los siguientes:

* Analizar el uso de tecnología para obtener una medición significativa y precisa de la temperatura de un transformador de distribución.
* Desarrollar un acceso ameno al usuario para un seguimiento y control, acorde a las situaciones reales a las que se tienen los equipos instalados en la zona.
* Diseñar e implementar un prototipo de medición interna de la temperatura en un transformador para Distribución Eléctrica según las consideraciones y especificaciones indicadas en la norma IEC 60076. La cual especifica los requisitos para la temperatura de operación y sobre elevación permitida para transformadores de Potencia (SET).
* Contemplar la lectura de la temperatura de la carcasa o temperatura ambiente del transformador.

## Organización del trabajo

La organización del proyecto integrador es la siguiente: en el Capítulo[2](#_bookmark7), se plantean las bases teóricas para introducir al lector al funcionamiento de los algoritmos de los distintos métodos desarrollados, describiendo hardware, telemetrías y plataformas utilizadas. Estos son:

* IoT “Internet of Things” en un entorno con acceso a Internet mediante WIFI.
* Protocolo Modbus TCP/IPpara los casos de acceso a una línea internet tipo WIFI.
* Protocolo Modbus RTU Remote Terminal Unit (Unidad Terminal Remota) en entornos donde no se posee líneas o redes propietarias de comunicaciones disponibles.
* Comunicación inalámbrica de corto alcance Bluetooth BLE

A continuación, en el Capítulo [3](#_bookmark19), se procederá a describir todo lo relacionado con el desarrollo de los programas y plataformas del prototipo PMTT (Prototipo de Medición y Telemetría de Temperatura), basados en los objetivos planteados en la sección anterior. Esto cubrirá detalles significativos de la programación en como las claves importantes de las plataformas para la visualización de los datos y controles al usuario.

Luego, en el Capítulo [4](#_bookmark33), se expondrán los ensayos y resultados obtenidos según lo propuesto en el Capítulo [3](#_bookmark19), y se detallarán las bases de lo que sería la propuesta de abordaje en campo, en función de los mismos resultados.

Finalmente, en el Capítulo [5](#_bookmark55)se termina de resumir el trabajo realizado y qué conclusiones se obtuvieron, dando cierre al trabajo con las propuestas de trabajos a futuro.

# Capítulo 2 - Teoría y definiciones

El capítulo que se desarrolla a continuación establece un marco teórico para comprender los conceptos claves utilizados a lo largo del trabajo. La primera parte introduce definiciones básicas relacionadas a la relación temperatura y transformadores. La segunda parte describe el hardware que se utiliza, los algoritmos y plataformas con sus características fundamentales.

## Transformadores de distribución

En esta sección se concientizará sobre los puntos relevantes de los transformadores y su relación con la temperatura según las normas de la IEC.

### Transformadores

Los transformadores son nodos críticos en la infraestructura eléctrica, ya que sin ellos la transmisión a larga distancia sería económicamente inviable debido a las elevadas pérdidas energéticas. Su alta eficiencia (superior al 98%) y fiabilidad los convierten en el corazón del sistema de energía eléctrica, garantizando que la electricidad generada llegue de manera segura y controlada a los puntos de consumo. A lo largo de todo el trayecto de generación y distribución se tienen distintos tipos según sus funciones, de potencia, distribución, medición, aislamiento, etc

Las protecciones eléctricas son fundamentales para garantizar la seguridad, fiabilidad y continuidad del servicio en la industria energética, minimizando daños en los equipos por anomalías en la red. Pero no todos estos tipos poseen las protecciones necesarias instaladas en la práctica de la zona.

Un transformador de distribución eléctrica es un equipo electromecánico que se utiliza para reducir el voltaje de la energía eléctrica desde los niveles de transmisión (alta tensión) a niveles más bajos adecuados para su distribución y consumo en hogares, industrias y comercios.

La **eficiencia de un transformador** está influenciada directamente por la **temperatura**, ya que el calor afecta tanto a las **pérdidas eléctricas** como al **aislamiento** interno. La temperatura puede aumentar las perdidas resistivas (efecto joule), acelerar el envejecimiento del aislamiento, aumentar las perdidas en el núcleo (perdidas por histéresis y corrientes parasitas). Aunque la peor consecuencia es la indicada sobre la aislación del equipo, ya que el calor excesivo degrada el aislamiento dieléctrico del transformador lo que acorta la vida útil del equipo y puede llevar a fallos si no se controla. Aunque esto no afecta directamente la eficiencia instantánea, reduce la eficiencia a largo plazo por mantenimiento o reemplazo.

Un transformador puede aumentar su temperatura de trabajo ya sea por factores externos, como ambientales, como por inconvenientes en su funcionamiento. Este último siendo el más influyente en la práctica, puede ocasionarse por un aumento desmesurado en su devanado secundario, ya sea por exceder el consumo límite de carga de corriente o como una falla que ocasione un cortocircuito. Aunque estos incrementos de corriente pueden no dañar por si mismas al equipo, conllevan un aumento de temperatura en forma de histéresis y así degradar la integridad del transformador.

De acuerdo con las normas de IEC, el punto caliente (PC) de un transformador es la temperatura más elevada que se encuentra en el aislamiento de los devanados. No se trata de una temperatura uniforme en todo el transformador, sino que se concentra en zonas específicas. Por la accesibilidad que se tiene a estos puntos en la práctica, se utilizará como punto de medición para este trabajo.

### Temperatura según IEC

La norma IEC 60076 es un estándar internacional de la Comisión Electrotécnica Internacional (IEC) que establece las directrices para el diseño, fabricación, pruebas y puesta en servicio de los transformadores de potencia. Una de las adiciones más importantes en las ediciones más recientes de la norma (a partir de la versión de 2011) fue la inclusión de un límite para la temperatura del "punto caliente" del devanado, que es la temperatura más alta que se alcanza en un punto del bobinado del transformador. Esto permite una evaluación más precisa del envejecimiento del aislamiento.

La norma IEC 60076-2 es la segunda parte del estándar y se centra en el calentamiento de los transformadores sumergidos en líquido. Define los límites máximos de elevación de la temperatura permitidos para los devanados y el líquido (aceite) del transformador. Estos límites están diseñados para proteger el aislamiento del equipo, que es el componente más sensible al calor. Por otro lado, la norma IEC 60076-7 sirve como guía de carga para transformadores de potencia sumergidos en aceite mineral. Su propósito principal es describir cómo las diferentes condiciones de temperatura ambiente y de carga afectan la vida útil del transformador, ofreciendo recomendaciones para operarlos de forma segura y efectiva.

Con estas consideraciones podemos resumir que los límites de temperatura máxima del punto caliente es la temperatura absoluta del punto más caliente del devanado, y es la norma a seguir para la operación segura bajo carga y sobrecarga. Este punto para los transformadores sumergidos en aceite ronda los 100ºC según la clase térmica del aislamiento del transformador. Para conocer el valor de temperatura máxima según la IEC 60076, se debe consultar el diseño específico del transformador, ya que la norma establece los parámetros y el constructor determina el límite final basado en la clase de aislamiento.

Además, la norma establece además límites para la temperatura ambiente, ya que esta es un factor crítico en el diseño y la operación segura de los transformadores. Los límites de temperatura ambiente determinan el punto de partida para el aumento de temperatura del transformador y son esenciales para calcular la temperatura máxima total que alcanzarán sus bobinados y su aislamiento. Un aumento de la temperatura ambiente por encima de los límites nominales reducirá la capacidad del transformador para disipar el calor de forma eficaz, lo que elevará la temperatura de su aislamiento, acelerará su degradación y acortará la vida útil del equipo.

El límite máximo de temperatura ambiente que se aplican a los transformadores de potencia inmersos en líquido, según la norma, es +40 °C en cualquier momento. Por el lado de la temperatura mínima -25 °C para transformadores de exterior, pero al no ser un inconveniente en la práctica de la zona no se abordará este caso.

En Argentina la Norma IRAM 2018 define los procedimientos para los ensayos de calentamiento en transformadores de potencia inmersos en líquido. Se alinea estrechamente con la norma IEC 76-2 con lo cual, establece las bases para verificar si un transformador puede operar dentro de los límites de temperatura seguros bajo condiciones de carga nominal.

### Hardware del sistema de adquisición de datos

En esta sección se detallará el hardware utilizado que se ilustra en la figura 2.1, en el cual se visualiza en la división lo desarrollado o abarcado en este proyecto.

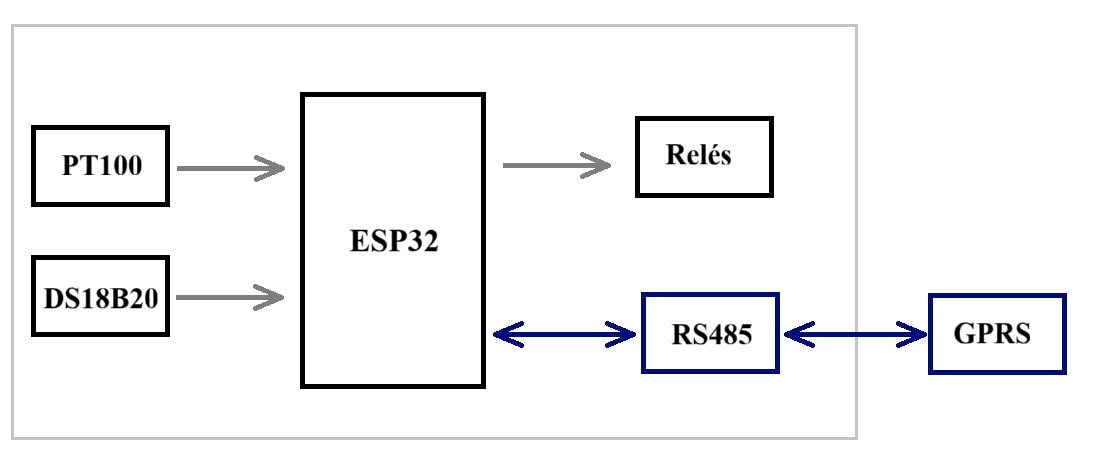


Figura 2‑1

Figura2.1: Hardware utilizado.

### ESP32

ESP32 es un circuito integrado creado por la empresa Espressif Systems, de bajo costo y bajo consumo de energía, que integra un microcontrolador y conectividad inalámbrica, principalmente Wi-Fi y Bluetooth. Dentro de sus prestaciones se resumen a grandes rasgos en la figura 1.1

El ESP32 se programa con tres lenguajes principales: C, C++ y Python, el utilizado en este trabajo es el modelo ESP32-S3. Para esta placa se utilizó el marco de desarrollo ESP-IDF (Espressif IoT Development Framework), el cual es el kit de desarrollo de software (SDK) oficial de Espressif Systems para desarrollar software en sus microcontroladores, utilizando C como lenguaje de programación.

En el caso de periféricos esta placa ofrece una gran cantidad de pines de propósito general (GPIO), que pueden ser configurados para diferentes funciones. Entre los periféricos más comunes y sus funciones se encuentran:

* + ADC/DAC: Permite la conversión de señales analógicas a digitales y viceversa, lo que facilita la interacción con sensores analógicos.
  + I2C, SPI, UART: Para la comunicación con otros chips y módulos.
  + PWM: Para controlar motores, LED y otros componentes.
  + Capacidades táctiles: Algunos pines pueden detectar toques capacitivos.

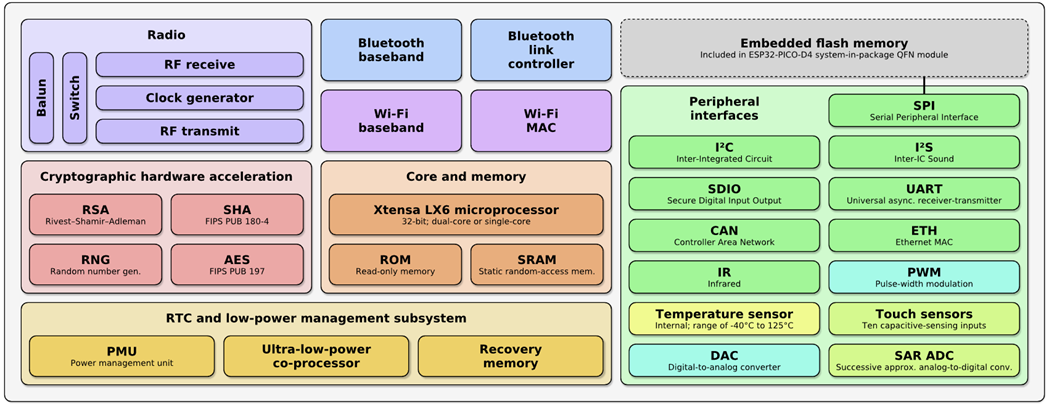


Figura 2.1: ESP32 Diagrama de funciones en bloque.

Por el lado del código de programación de esta placa en el siguiente capitulo se explicarán las líneas de código destacables a la aplicación correspondiente. Como dispositivos externos que se utilizan en este proyecto podemos nombrar los sensores de temperatura, descritos en la siguiente sección una placa de relés, la cual se utiliza en el caso que el usuario quiera conectar o desconectar elementos o equipos extras, y una placa que transforma la comunicación digital seria tipo RS485 (seleccionada para este trabajo por sus prestaciones descritas en este capítulo) para su correcta transmisión.

### Sensor de Temperatura PT100

Para la recolección de datos de temperatura se tiene en el mercado sensores como el PT100, este es un tipo de sensor de temperatura de resistencia que utiliza platino como material conductor, su resistencia es de 100 ohmios a 0 °C y aumenta al aumentar su temperatura. Se destaca por su alta precisión y estabilidad a lo largo del tiempo, lo que lo hace ideal para aplicaciones industriales donde se requiere una medición precisa y confiable de la temperatura. En la siguiente figura se ilustra el comportamiento de este tipo de sensor utilizado en este proyecto, su selección se basa en la practicidad de utilizarlo para la medición del PC del transformador, ya que debe estar embebido en aceite.

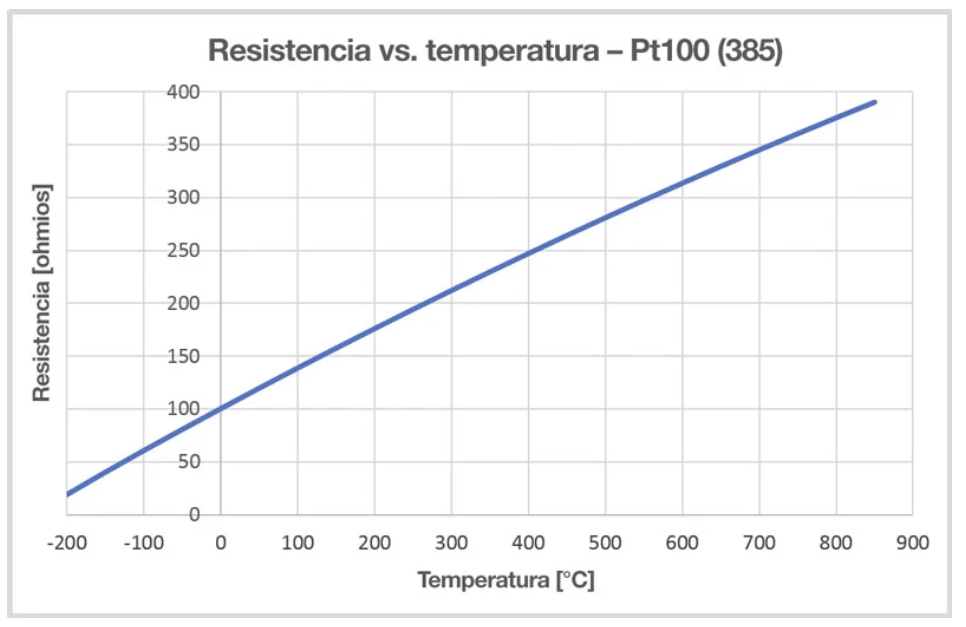


Figura2.2: Resistencia vs. Temperatura PT100 ejemplo

Por otro lado, el MAX31865 es un circuito integrado de alta precisión diseñado para convertir la resistencia de un termómetro de resistencia de platino (RTD), como un PT100 o un PT1000, en una señal digital. Se comunica con la placa ESP32 a través de una interfaz SPI, lo que simplifica la integración en este proyecto

### Sensor de Temperatura DS18B20

El DS18B20 es un sensor digital de temperatura muy popular por ser preciso, versátil y fácil de conectar. Mide temperaturas desde -55 °C hasta 125 °C y se comunica mediante el protocolo 1-Wire, que requiere solo un pin de datos para enviar y recibir información.  
Este sensor se seleccionó para medir la temperatura ambiente del transformador.

### Placa RS485

Un módulo MAX485 es un conversor para adaptar señales de lógica del microcontrolador a señales del estándar de comunicación RS485. Esto permite que microcontroladores como Arduino, ESP32 o Raspberry Pi se comuniquen a largas distancias y en entornos con mucho ruido eléctrico, donde la comunicación convencional no funcionaría. Este modulo solo se utilizará en el caso de requerir la comunicación serial, como con los métodos Modbus.

### Modulo GPRS

Un módulo GPRS permite a un dispositivo (como un microcontrolador, Arduino o ESP32) conectarse a Internet a través de la red celular, utilizando el servicio General Packet Radio Service (GPRS). Esencialmente, le da a un proyecto de electrónica la capacidad de un teléfono móvil para enviar y recibir datos. Este módulo es externo a lo desarrollado en este proyecto pero necesario para la comunicación remota. Poner mas cosas??????????

### Conceptos IoT

En un medio donde se tiene acceso a una red de internet tipo wifi se propone como medio de comunicación de la transmisión de datos el uso de IoT (Internet of things), el cual es una red de objetos físicos que pueden estar equipados con sensores, software u otras tecnologías para conectarse e intercambiar datos a través de Internet. Esto permite que objetos cotidianos, como electrodomésticos o equipos industriales, se vuelvan "inteligentes", recopilen y compartan información, y ejecuten programas sin intervención humana. Para desarrollar el sistema en este proyecto se optó por utilizar MQTT (Message Queuing Telemetry Transport) que es un protocolo de mensajería ligero para la comunicación entre dispositivos con recursos limitados, como lo es el mencionado IoT.

MQTT se basa en un modelo de publicación-suscripción con un [agente de mensajes](https://www.google.com/search?q=agente+de+mensajes&sca_esv=7c4197be7832523f&source=hp&ei=r_ffaNbHPMyc5OUP9o2BqAs&iflsig=AOw8s4IAAAAAaOAFwL6mNoI1T2MHYELBSSeMc8Cfgj_H&ved=2ahUKEwi99N7Mt4iQAxV_KLkGHa7UOMkQgK4QegYIAQgAEAQ&uact=5&oq=mqtt&gs_lp=Egdnd3Mtd2l6GgIYAyIEbXF0dDIQEAAYgAQYsQMYQxiDARiKBTIKEAAYgAQYQxiKBTIFEAAYgAQyChAAGIAEGEMYigUyChAAGIAEGEMYigUyCBAAGIAEGIsDMggQABiABBiLAzIIEAAYgAQYiwMyCBAAGIAEGIsDMggQABiABBiLA0jlEFAAWL8HcAB4AJABAJgBZ6AB_gKqAQMzLjG4AQPIAQD4AQGYAgSgAosDwgINEAAYgAQYQxiKBRiLA8ICGhAuGIAEGLEDGNEDGNIDGIMBGMcBGKgDGIsDwgITEAAYgAQYsQMYQxiDARiKBRiLA8ICFxAuGIAEGLEDGNEDGNIDGMcBGKgDGIsDwgIZEC4YgAQY0QMY0gMYQxjHARioAxiKBRiLA8ICCxAAGIAEGLEDGIsDwgIOEAAYgAQYsQMYgwEYiwPCAg0QABiABBixAxhDGIoFmAMAkgcDMy4xoAfKGLIHAzMuMbgHiwPCBwUwLjMuMcgHCQ&sclient=gws-wiz), llamado [broker](https://www.google.com/search?q=broker&sca_esv=7c4197be7832523f&source=hp&ei=r_ffaNbHPMyc5OUP9o2BqAs&iflsig=AOw8s4IAAAAAaOAFwL6mNoI1T2MHYELBSSeMc8Cfgj_H&ved=2ahUKEwi99N7Mt4iQAxV_KLkGHa7UOMkQgK4QegYIAQgAEAU&uact=5&oq=mqtt&gs_lp=Egdnd3Mtd2l6GgIYAyIEbXF0dDIQEAAYgAQYsQMYQxiDARiKBTIKEAAYgAQYQxiKBTIFEAAYgAQyChAAGIAEGEMYigUyChAAGIAEGEMYigUyCBAAGIAEGIsDMggQABiABBiLAzIIEAAYgAQYiwMyCBAAGIAEGIsDMggQABiABBiLA0jlEFAAWL8HcAB4AJABAJgBZ6AB_gKqAQMzLjG4AQPIAQD4AQGYAgSgAosDwgINEAAYgAQYQxiKBRiLA8ICGhAuGIAEGLEDGNEDGNIDGIMBGMcBGKgDGIsDwgITEAAYgAQYsQMYQxiDARiKBRiLA8ICFxAuGIAEGLEDGNEDGNIDGMcBGKgDGIsDwgIZEC4YgAQY0QMY0gMYQxjHARioAxiKBRiLA8ICCxAAGIAEGLEDGIsDwgIOEAAYgAQYsQMYgwEYiwPCAg0QABiABBixAxhDGIoFmAMAkgcDMy4xoAfKGLIHAzMuMbgHiwPCBwUwLjMuMcgHCQ&sclient=gws-wiz), que recibe mensajes de los [editores](https://www.google.com/search?q=editores&sca_esv=7c4197be7832523f&source=hp&ei=r_ffaNbHPMyc5OUP9o2BqAs&iflsig=AOw8s4IAAAAAaOAFwL6mNoI1T2MHYELBSSeMc8Cfgj_H&ved=2ahUKEwi99N7Mt4iQAxV_KLkGHa7UOMkQgK4QegYIAQgAEAY&uact=5&oq=mqtt&gs_lp=Egdnd3Mtd2l6GgIYAyIEbXF0dDIQEAAYgAQYsQMYQxiDARiKBTIKEAAYgAQYQxiKBTIFEAAYgAQyChAAGIAEGEMYigUyChAAGIAEGEMYigUyCBAAGIAEGIsDMggQABiABBiLAzIIEAAYgAQYiwMyCBAAGIAEGIsDMggQABiABBiLA0jlEFAAWL8HcAB4AJABAJgBZ6AB_gKqAQMzLjG4AQPIAQD4AQGYAgSgAosDwgINEAAYgAQYQxiKBRiLA8ICGhAuGIAEGLEDGNEDGNIDGIMBGMcBGKgDGIsDwgITEAAYgAQYsQMYQxiDARiKBRiLA8ICFxAuGIAEGLEDGNEDGNIDGMcBGKgDGIsDwgIZEC4YgAQY0QMY0gMYQxjHARioAxiKBRiLA8ICCxAAGIAEGLEDGIsDwgIOEAAYgAQYsQMYgwEYiwPCAg0QABiABBixAxhDGIoFmAMAkgcDMy4xoAfKGLIHAzMuMbgHiwPCBwUwLjMuMcgHCQ&sclient=gws-wiz) y los envía a los [suscriptores](https://www.google.com/search?q=suscriptores&sca_esv=7c4197be7832523f&source=hp&ei=r_ffaNbHPMyc5OUP9o2BqAs&iflsig=AOw8s4IAAAAAaOAFwL6mNoI1T2MHYELBSSeMc8Cfgj_H&ved=2ahUKEwi99N7Mt4iQAxV_KLkGHa7UOMkQgK4QegYIAQgAEAc&uact=5&oq=mqtt&gs_lp=Egdnd3Mtd2l6GgIYAyIEbXF0dDIQEAAYgAQYsQMYQxiDARiKBTIKEAAYgAQYQxiKBTIFEAAYgAQyChAAGIAEGEMYigUyChAAGIAEGEMYigUyCBAAGIAEGIsDMggQABiABBiLAzIIEAAYgAQYiwMyCBAAGIAEGIsDMggQABiABBiLA0jlEFAAWL8HcAB4AJABAJgBZ6AB_gKqAQMzLjG4AQPIAQD4AQGYAgSgAosDwgINEAAYgAQYQxiKBRiLA8ICGhAuGIAEGLEDGNEDGNIDGIMBGMcBGKgDGIsDwgITEAAYgAQYsQMYQxiDARiKBRiLA8ICFxAuGIAEGLEDGNEDGNIDGMcBGKgDGIsDwgIZEC4YgAQY0QMY0gMYQxjHARioAxiKBRiLA8ICCxAAGIAEGLEDGIsDwgIOEAAYgAQYsQMYgwEYiwPCAg0QABiABBixAxhDGIoFmAMAkgcDMy4xoAfKGLIHAzMuMbgHiwPCBwUwLjMuMcgHCQ&sclient=gws-wiz) interesados en esos temas. Sus ventajas incluyen baja cantidad de memoria de código, optimización del ancho de banda, soporte para redes inestables y niveles de calidad de servicio (QoS). Dentro de los distintos brokers se encuentra Thingsboard, el cual permite realizar un tablero para visualizar al usuario los distintos controles y variables.

Con estos conceptos podemos resumir en un diagrama la comunicación de lo implementado en este proyecto de la siguiente manera, cuyo funcionamiento se detalla en el capítulo 3.

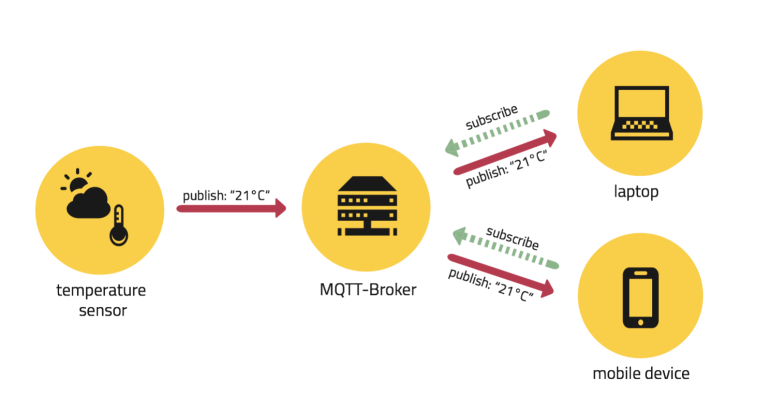


Figura2.3: Diagrama en bloques para el uso de IoT ejemplificando la toma de mediciones del PC hasta la visualización del usuario.

Para el enlace de datos entre el hardware y el software (Thingsboard) se crean en este último uno o más dispositivos. Cada dispositivo tiene un código alfanumérico identificatorio especifico llamado token. Un token en MQTT es una credencial de seguridad para autenticar y autorizar la vinculación entre los dispositivos y clientes, por lo que ese dato se utiliza en la programación de nuestra placa ESP32. Por si se pone desarrollo en el capítulo 4

### Conceptos de desarrollo en Modbus

Modbus es un protocolo de comunicación industrial abierto publicado originalmente por Modicon (ahora Schneider Electric) en 1979, basado en el modelo maestro/esclavo o cliente/servidor, que permite la transferencia de datos entre dispositivos electrónicos, como PLCs, sensores y actuadores. Su simplicidad y fiabilidad lo han convertido en el estándar para la supervisión y el control en la automatización industrial, con variantes para redes seriales (como RTU y ASCII) y para redes Ethernet ([Modbus/TCP](https://www.google.com/search?q=Modbus%2FTCP&sca_esv=7c4197be7832523f&ei=t_ffaP28Dv_Q5OUPrqnjyQw&ved=2ahUKEwjRm5bEvIiQAxUwjpUCHWXEOP0QgK4QegYIAAgAEAQ&uact=5&oq=modbus&gs_lp=-AUYqAMYigUYiwPCAg4QABiABBixAxiDARiLA8ICERAAGIAEGLEDGIMBGIoFGIsDwgIFEAAYgASYAwPxBYU36s-eIGiJiAYBkAYKugYECAEYB5IHAzQuNaAHszWyBwMxLjW4B4wFwgcFMC41LjTIBxk&sclient=gws-wiz-serp)).

Modbus RTU y Modbus TCP/IP son las dos variantes más comunes del protocolo de comunicación industrial Modbus. Aunque ambos comparten la estructura de mensajes central de Modbus, se diferencian fundamentalmente en el medio de comunicación que utilizan y en cómo empaquetan los datos.

Por otro lado, también puede utilizarse ESP Now que permite la comunicación directa entre dispositivos sin necesidad de un router o punto de acceso WiFi. Funciona usando el hardware de WiFi pero en una capa de enlace de datos simplificada, lo que resulta en baja latencia, bajo consumo de energía y una comunicación rápida para paquetes de datos de hasta 250 bytes. La comunicación se establece emparejando dispositivos a través de sus direcciones MAC y puede ser unidireccional o bidireccional.

* + 1. **Modbus RTU**

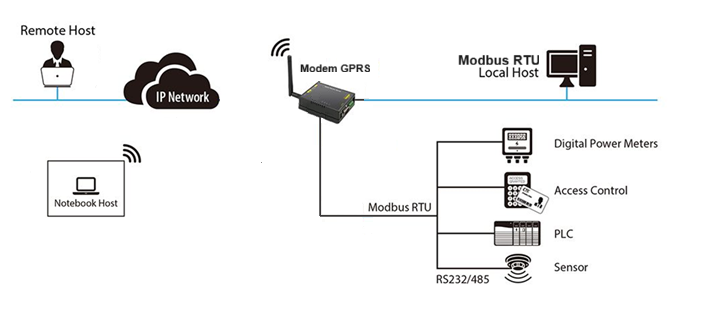
****

Figura2.4: Diagrama ejemplo de Modbus RTU.

Esta es la implementación más común disponible para Modbus. Se utiliza en la comunicación serie y hace uso de una representación binaria compacta de los datos para el [protocolo de comunicación](https://es.wikipedia.org/wiki/Protocolo_de_comunicaciones). El formato RTU sigue a los comandos/datos con una suma de [comprobación de redundancia cíclica](https://es.wikipedia.org/wiki/Verificaci%C3%B3n_de_redundancia_c%C3%ADclica) (CRC) como un mecanismo de comprobación de errores para garantizar la fiabilidad de los datos. Un mensaje Modbus RTU debe transmitirse continuamente sin vacilaciones entre caracteres. Los mensajes Modbus son entramados (separados) por períodos inactivos (silenciosos).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Start** |  | **Address** | **Function** | **Data** | **CRC** |  | **End** |
| ≥3.5 char |  | 1 Byte | 1 Byte | N\*2 Bytes | 2 Bytes |  | ≥3.5 char |

#### Donde:

#### Start y End son los espacios divisores entre cada trama.

#### Address la dirección de a quien se le comunica.

#### Function la función que se quiere realizar.

#### Data la información necesaria según la función utilizada.

#### CRC la corroboración de redundancia cíclica.

Como enlace o via para enviar estas señales pueden utilizarse normas de comunicación en serie como el RS-232 y el RS-485. El RS-232 es un protocolo simple, ideal para distancias cortas (hasta 15 metros) y conexiones punto a punto entre dos dispositivos, la comunicación no utiliza una señal de reloj o de sincronización compartida entre el emisor y el receptor. Utiliza niveles de voltaje positivos y negativos para representar los bits. Un 1 lógico (marca) se representa con un voltaje negativo (entre -3V y -25V), mientras que un 0 lógico (espacio) se representa con un voltaje positivo (entre +3V y +25V).

El RS-485 a diferencia del RS232, utiliza un sistema de señalización diferencial que lo hace ideal para largas distancias y redes con múltiples dispositivos. Es decir, transmite los datos mediante dos señales opuestas (A y B) sobre un par de cables trenzados. El receptor compara la diferencia de voltaje entre ambas, lo que elimina el ruido común que afecta a ambos cables, haciéndolo muy resistente a las interferencias electromagnéticas. los niveles de voltaje no se definen en relación con una tierra común, como en RS232, sino como una diferencia de voltaje entre los dos cables, el A y el B. En la figura se ilustra las líneas que utiliza cada método.

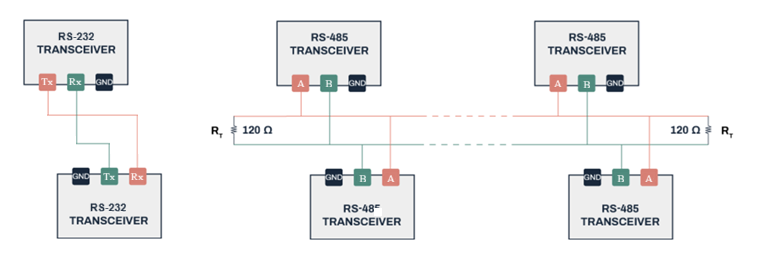


Figura2.5: Comunicación RS232 y RS485.

* + 1. **Modbus TCP/IP**

#### 

Figura 2.: Ejemplo de comunicación con Modbus TCP/IP.

Modbus TCP/IP (Protocolo de Control de Transmisión/Protocolo de Internet) es un estándar de automatización que utiliza el protocolo de red TCP/IP sobre Ethernet para transportar mensajes.  La principal diferencia con Modbus RTU es el uso de redes Ethernet/WIFI, ofreciendo mayor velocidad y escalabilidad aprovechando la infraestructura de red ya existente, tanto para una conexión cableada como para una red Wireless interna. Cada paquete se envía a su destino utilizando direcciones IP, y puede tomar rutas diferentes si es necesario. Opera en común con RTU con una arquitectura cliente/servidor aunque con distinta trama ya que la infraestructura de red TCP/IP proporciona la base de comunicación, encargándose del enrutamiento, la fragmentación y la verificación de la correcta recepción de los datos, con esto Modbus se encarga de definir el significado e interpretación de los datos.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Transaction Identifier** | **Protocol Identifier** | **Length Field** | **Address** | **Function** | **Data** |
| 2 Bytes | 2 Bytes | 2 Bytes | 1 Byte | 1 Byte | N\*1 Byte |

Donde:

* Transaction Identifier es un número que asocia una solicitud con su respuesta correspondiente, permitiendo múltiples transacciones simultáneas.
* Protocol Identifier es un campo fijo que para Modbus/TCP siempre es 0x0000.
* Length Field indica el número de bytes que siguen a este campo, es decir, la longitud total del resto del mensaje.
* Address la dirección de a quien se le comunica.
* Function la función que se quiere realizar.
* Data la información necesaria según la función utilizada.
  + 1. **Lab Windows/CVI**

LabWindows/CVI es un entorno de desarrollo de software para lenguaje ANSI C, Instituto Nacional Estadounidense de Estándares (ANSI), creado por National Instruments (NI). Está diseñado para crear aplicaciones de pruebas, mediciones y automatización que se centra en la programación basada en texto, siendo CVI el acrónimo de "C for Virtual Instrumentation" (C para Instrumentación Virtual).

Un instrumento virtual es un sistema que utiliza una computadora estándar equipada con software de aplicación potente y hardware de adquisición de datos para realizar las funciones de los instrumentos tradicionales.

Las características principales se basan en su capacidad para combinar la familiaridad y eficiencia del lenguaje C con herramientas específicas para la instrumentación virtual, incluyendo una extensa integración de hardware (GPIB, USB, PXI), bibliotecas integradas para análisis de datos y procesamiento de señales, y un constructor de interfaces gráficas de usuario (GUI) intuitivo, lo que permite crear aplicaciones robustas de pruebas y medición de forma rápida y eficiente.

Con este entorno se obtiene un ejecutable para ser instalado en cualquier computadora, de forma de contar en cualquier computadora con el programa desarrollado y así conectarse a la red para acceder a los datos correspondientes.

### Bluetooth BLE

Bluetooth es una tecnología de comunicación inalámbrica de corto alcance que permite el intercambio de datos y la conexión entre dispositivos electrónicos sin necesidad de cables, mientras que BLE (Bluetooth Low Energy) es una versión optimizada de esa tecnología, diseñada específicamente para un consumo de energía significativamente menor. Este último está diseñado para aplicaciones que envían pequeñas cantidades de datos de forma intermitente, como sensores de fitness, monitores de frecuencia cardíaca, dispositivos domóticos (IoT) y aplicaciones que se conectan instantáneamente. Por esto último fue seleccionado para desarrollar una aplicación con la cual conectarse fácilmente a un transformador y revisar su funcionamiento.

La comunicación BLE desarrollada en este proyecto se basa en un dispositivo (tranformador) que anuncia su presencia enviando paquetes de publicidad y acepta una conexión de otro dispositivo BLE, por otro lado la aplicación desarrollada para teléfonos móviles realizará un escaneo de esta publicidad y enviará la solicitud para iniciar la conexión entre ellos y comenzar el traspaso de datos de temperatura del equipo.

Para lograr esta conexión de datos se deben configurar términos importantes como los siguientes:

* GAP (Generic Access Profile) se encarga de la conexión entre dispositivos.
* GATT (Generic Attribute Profile) trata de la estructura de los datos una vez que dos dispositivos BLE están conectados. Define como se estructuran e intercambian datos.
* UUID (Universally Unique Identifier) se encarga de la identificación de los datos. Son identificadores únicos de 128 bits (aunque a menudo se abrevian a 16 o 32 bits para servicios y características estándar) que permiten a los dispositivos saber exactamente con qué tipo de datos están tratando.
* CCCD (Client Characteristic Configuration Descriptor) gestiona las notificaciones automáticas de datos, es decir que el dispositivo esclavo le envía al maestro notificaciones cuando sus variables sufren algún cambio.

# 

# Capítulo 3 - Programación y desarrollo

El presente capítulo describirá los puntos destacables o distintivos de los programas creados en sus respectivos lenguajes de programación y sus distintos formatos para lograr la comunicación de los parámetros a medir y tratar para la correcta visualización e interacción con el usuario final. Además, de las plataformas y sus componentes diseñados para una interacción amena con el usuario.

## Método IoT

Thingsboard como se mencionó en el capítulo anterior se trata de un bróker/servidor donde diseñamos un entorno para la visualización e interacción con el usuario, llamado Dashboard. En este se agregaron distintos widgets/herramientas para visualizar finalmente como se muestra en la figura, en primera instancia se pueden ver todos los dispositivos vinculados con sus respectivos datos de temperatura y sus coordenadas, con las cuales se visualiza la ubicación en un mapa a su derecha. Debajo se tiene una ventana donde emergerán todas las alarmas vinculadas a los dispositivos.



Figura3.1: Tablero inicial Thingsboard.

Luego al seleccionar cada transformador se obtiene un tablero con los datos exclusivos de ese dispositivo.

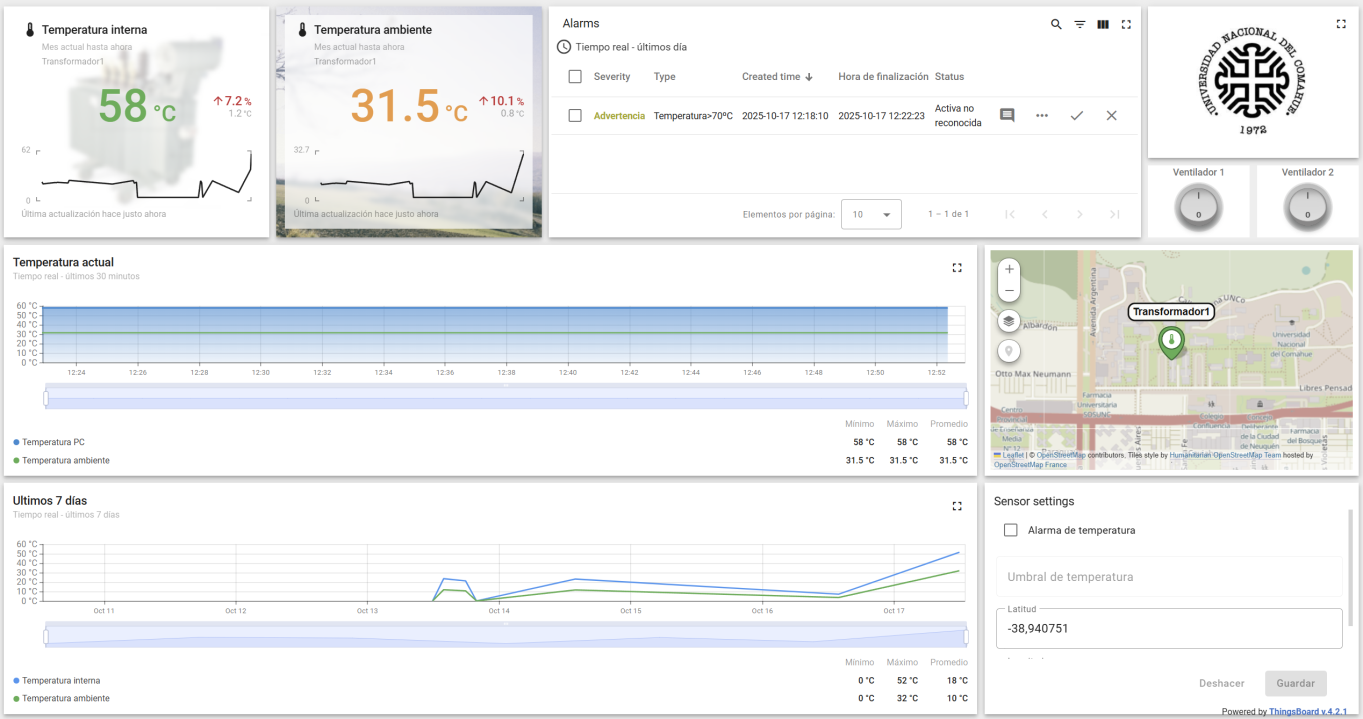


Figura3.2: Tablero de cada transformador.

De este tablero podemos destacar:

* Displays de temperatura: Se ajusta su imagen según lo analizado en la sección 2.1 para un reconocimiento rápido de las zonas de temperatura y se anidan respectivamente a los valores enviados, visualizando la temperatura PC y ambiente del transformador en análisis.



Figura3.3: Displays de temperatura.

* Alarmas: Para una visualización de un estado crítico de temperaturas se muestran por un lado la cantidad de alarmas activas y por otro una lista donde se puede analizar la criticidad de la alarma, su proveniencia, los tiempos en los cuales comenzó y terminó ese estado de alarma y la posibilidad de descartar estas alarmas.



Figura3.4: Sección de alertas de alarmas.

* Históricos: Se puede observar la temperatura a lo largo del tiempo primero en detalle de tiempo real de los últimos 30 minutos y el histórico del día en curso.



Figura3.5: Históricos de las temperaturas.

* Opciones de control a distancia de un seccionador/interruptor que podría operar de tener la posibilidad (ejemplo ventilador 1 y 2).



Figura3.6: Selectores de elementos externos.

* Seteo de ubicación y un umbral de alarma si se quisiera agregar.



Figura3.7: Mapa con las coordenadas del transformador y seteo de coordenadas y umbral extra.

**Desarrollo:**

Para iniciar con la programación de este tablero como primer punto se crea un perfil de dispositivos (llamado Temperatura), con el cual armamos la estructura de las características específicas a todos nuestros dispositivos, como por ejemplo alarmas de temperatura. Por otro lado, se crea un dispositivo por cada transformador

, cuya finalidad es ser el enlace de datos entre el hardware y el software (Thingsboard). Este dispositivo tiene un código alfanumérico identificatorio especifico llamado token. Un token en MQTT es una credencial de seguridad para autenticar y autorizar la vinculación entre los dispositivos y clientes, por lo que ese dato se utiliza en la programación de nuestra placa ESP32. Poner desarrollo en el capitulo 4?



Figura3.8: Ejemplo dispositivo.

Al crearlo y vincularlo con nuestro sistema de medición y transmisión de datos de la ESP32 mediante wifi, podemos visualizar las telemetrías que se reciben en la pestaña corroborando si se efectúa correctamente la transmisión de datos como se visualiza en la figura 3.9. En la siguiente sección se comentará el funcionamiento del programa de la ESP32 que transmite estos datos.



Figura3.9: Telemetría de dispositivo

Una vez vinculado nuestro dispositivo se crea un tablero o Dashboard del cual se indica la toma de datos del dispositivo en cuestión. La sección de displays de temperatura se anida con los valores de temperatura directos para tener visualización continua de su valor en tiempo real. En el caso de los gráficos se configuran las modalidades de tiempo y datos con las temperaturas asignadas, agregando un umbral según corresponda a las temperaturas peligro.

Para configurar las alarmas predefinidas se agregan al perfil de dispositivo, con lo cual todos los dispositivos que utilizarán este perfil tendrán estas alarmas incorporadas. Esto se ilustra en la siguiente figura

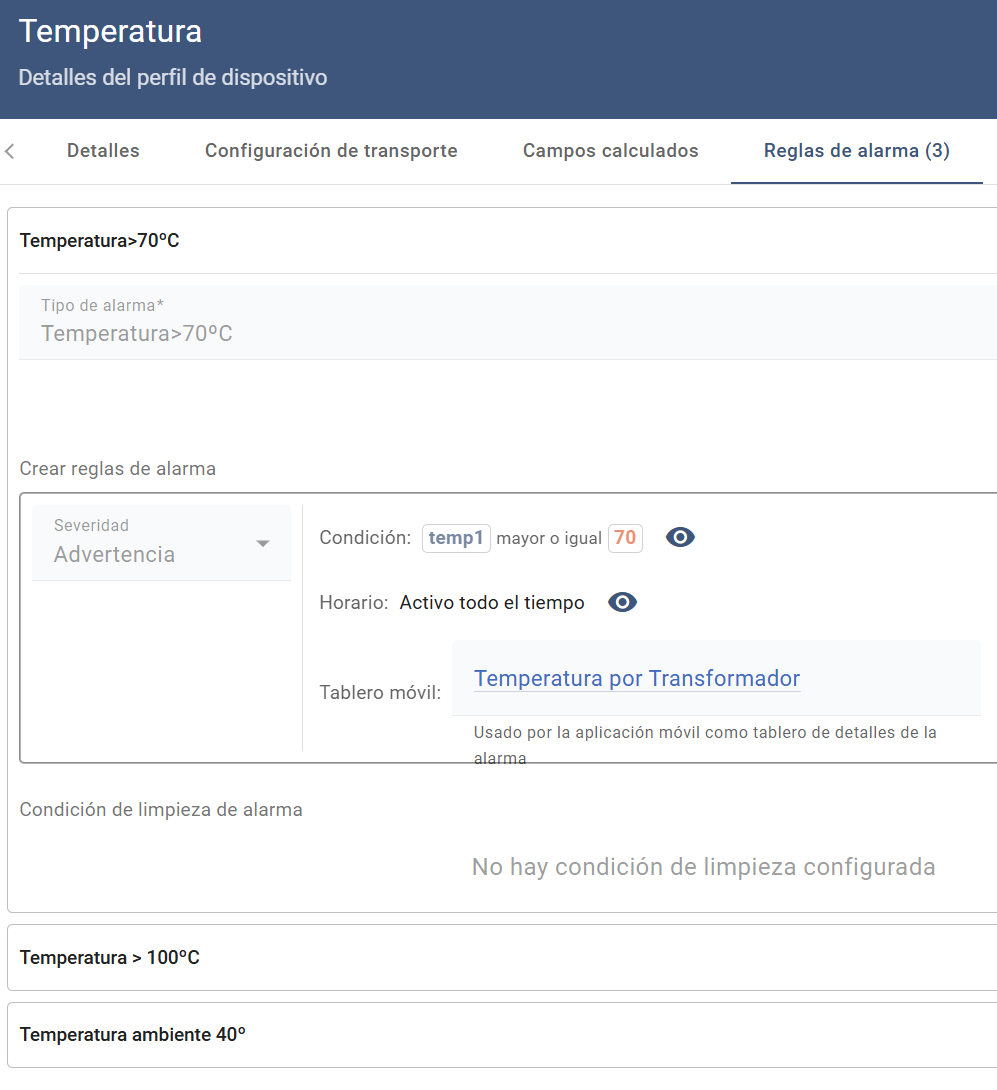


Figura3.10: Configuración de alarmas del perfil de dispositivo.

Una vez configurado esto se configuran widgets que ilustran las alarmas para visualizar de la manera deseada las alarmas categorizadas como críticas y advertencias entre críticas, advertencias, mayores, menores o indeterminadas, a las cuales se determina la visualización de activas despejadas, no reconocidas o reconocidas entre la opción alarmas activas, despejadas, no reconocidas o reconocidas para que el usuario tenga la opción de eliminar las alarmas ya revisadas. La figura 3.11 se visualiza el caso de las 3 alarmas por dispositivo activadas, alarma por temperatura del punto caliente mayor a 70 de calidad advertencia, alarma critica de temperatura de PC cercana a los 100ºC y por último una advertencia de que la temperatura ambiente se encuentra alcanzando los 40ºC.

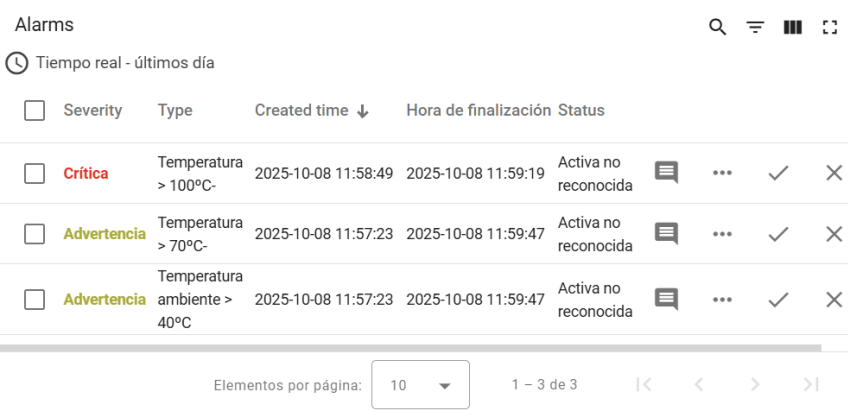


Figura3.11: Total de alarmas por dispositivo

En el caso del control de los ventiladores 1 y 2 se utiliza la funcionalidad RPC (Llamada a Procedimiento Remoto) permite enviar comandos entre la plataforma y los dispositivos conectados. Esta característica es fundamental para el control remoto en sistemas IoT y soporta dos tipos principales de comunicación: RPC iniciado por el servidor (Server-side RPC), en este caso la solicitud de comando se origina en la plataforma, y RPC iniciado por el cliente (Client-side RPC), donde la solicitud de comando se origina en el dispositivo y se envía a la plataforma. Ambos casos según la naturaleza del comando pueden requerir respuesta con algun tipo de informacion o solo seteos de parametros.

La trama que se utiliza ya sean de cliente o de servidor, se envían con una estructura JSON(JavaScript Object Notation), la cual se basa en dos elementos fundamentales: objetos y arrays, que permiten representar datos de manera organizada y legible. Con esto la trama para todas las comunicaciones será [“method”;”params”]

Donde:

* "method": El nombre del procedimiento que se quiere ejecutar (por ejemplo, "reboot" o "turnOn").
* "params": Parámetros necesarios para ejecutar el método.

## Método a través de wifi ESP32 a IoT

Para iniciar se declaran todas las librerías necesarias a utilizar, en este caso además se incluyen librerías para el uso de wifi y mqtt a fin de poder utilizar funciones y o variables especificas de la aplicación.

#include"wifi.h"

#include"mqtt.h"

#include"Tingsboard

.h"

Se debe declarar el nombre del wifi y su contraseña a utilizar como el token del dispositivo declarado en la plataforma Thingsboard

char ssid[MAX\_SSID\_LEN]= "WIFI";

char password[MAX\_PASS\_LEN]= "Contraseña";

#define THINGSBOARD\_ACCESS\_TOKEN "75831rprfjpof6q3vlc7" //<-- Token de acceso al dispositivo

Para iniciar y apuntar a la página de mqtt seleccionada como Thingsboard se configura de la siguiente manera:

esp\_mqtt\_client\_config\_t mqtt\_cfg = {

.broker.address.uri = "mqtt://demo.thingsboard.io",

.broker.address.port = 1883,

.session.protocol\_ver = MQTT\_PROTOCOL\_V\_3\_1\_1,

.credentials.username = THINGSBOARD\_ACCESS\_TOKEN,

};

En este método se toman muestras de las temperaturas y se envían sin necesitar algún pedido por parte del servidor, enviando la variable y su respectivo valor. En el caso de querer activar uno de los elementos (como ventilador 1 o 2) el servidor manda una señal, RPC, con el método de seteo y parámetro 1 o 0 para activar o desactivar al mismo, en consecuencia a esto la placa realiza la activación correspondiente. Por lo que el bucle principal del programa se basa en toma de muestras, envío de datos y un tiempo de espera que se muestra a continuación. En la figura 3.12 se visualiza a grandes rasgos el programa entero.

while (1){

sensor\_read();

if (s\_temperature < 100.0)//para evitar falsas lecturas

{printf("Temperatura exterior : %0.1fC\n", s\_temperature);

float cTemp2=31.5;//Temperatura ambiente

char json\_payload[200];//crear JSON y asignarle valores

snprintf(json\_payload, sizeof(json\_payload),

"{\"temp1\":%.2f, \"temp2\":%.2f}",

s\_temperature, cTemp2)

PublicarMQTT( json\_payload);//Enviar comandos

vTaskDelay(2000 / portTICK\_PERIOD\_MS);//espera 2s

}

}

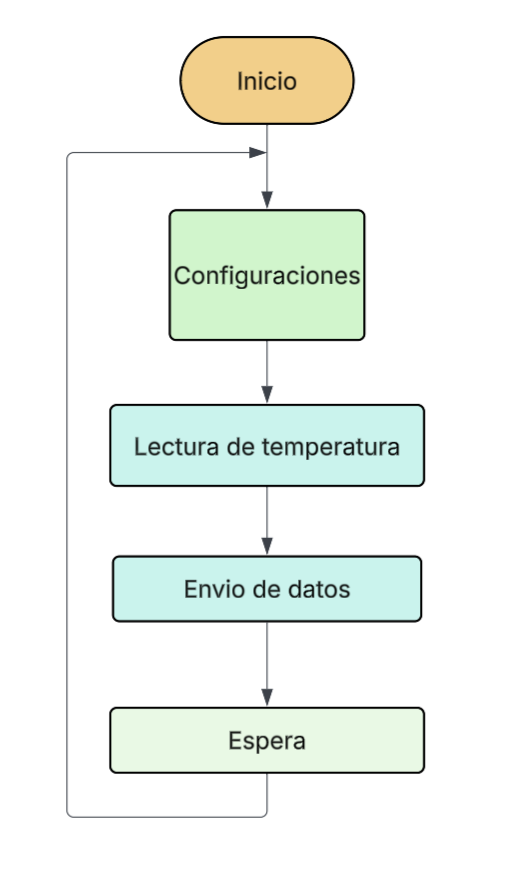


Figura 3.12: Diagrama de flujo del programa ESP para IOT.

## Método Modbus con ESP32

En este método la placa ESP32 se encuentra en modo esclavo del pedido de toma y envío de muestras del servidor, por lo que se tiene el programa continuamente consultando si tiene algún pedido, y en ese caso realizar el muestreo y envío. Para esta aplicación se tienen dos interfaces para el usuario, utilizando Labwindows CVI, el cual puede conectarse tanto como a la aplicación Modbus RTU como a TCP/IP, y con BBBBB pagina

* + 1. Labwindows CVI

En esta interfaz desarrollada con el programa Labwindows, se programó en C de forma de añadir elementos como botones, indicadores, gráficos y distintos seteos en los cuales cada uno posee su código. De esta forma se creó una pantalla inicial, Figura 3.13, de la cual se pueden seleccionar distintas pestañas detalladas a continuación.



Figura 3.13: Inicio de aplicación por Labwindows.

Serial: Desde esta ventana se puede setear las configuraciones de comunicación como el puerto com y la velocidad de la transmisión serial, para luego abrir el puerto y confirmar la comunicación indicándose en “Indicador de Puerto”.

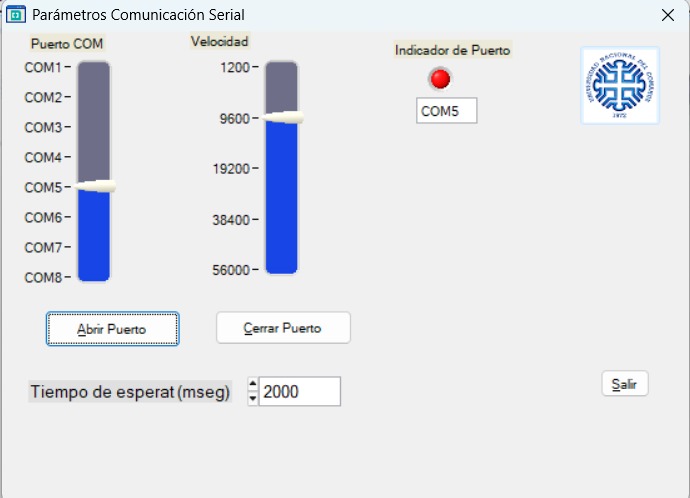


Figura 3.14: Parámetros de comunicación Serial

TCP/IP: Para visualizar datos desde la transmisión TCP/IP debe declararse la IP del dispositivo deseado y la IP propia y luego confirmar con el botón “Conectar Modbus TCP/IP”, el cual se ilumina ante este enlace.

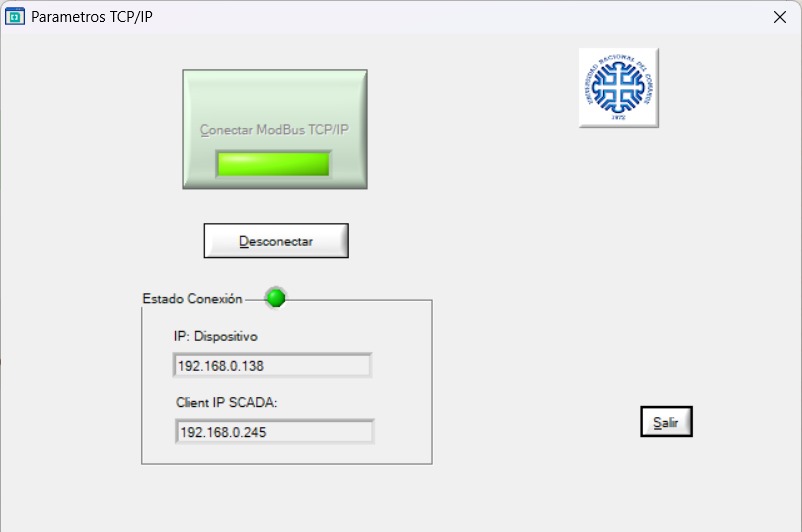


Figura 3.15: Parámetros para comunicación TCP/IP.

Lectura de Datos: Donde se tienen indicadores de las temperaturas, botones para la transferencia de datos de un único pedido o iniciar cada X minutos observando los datos en tiempo real, botones para la activación o desactivación de los equipos externos (ventilador 1 y 2), un botón de Almacenar con el cual se selecciona el directorio donde guardar los datos en un .txt, esta dirección se visualiza debajo para facilitar al usuario, y por último un botón de Ver Datos para abrir dicho txt.



Figura 3.16: Ventana de datos Labwindows.

La figura 3.17 muestra un ejemplo de los datos almacenados en el txt, el cual se utilizará para la opción de graficar las variables.

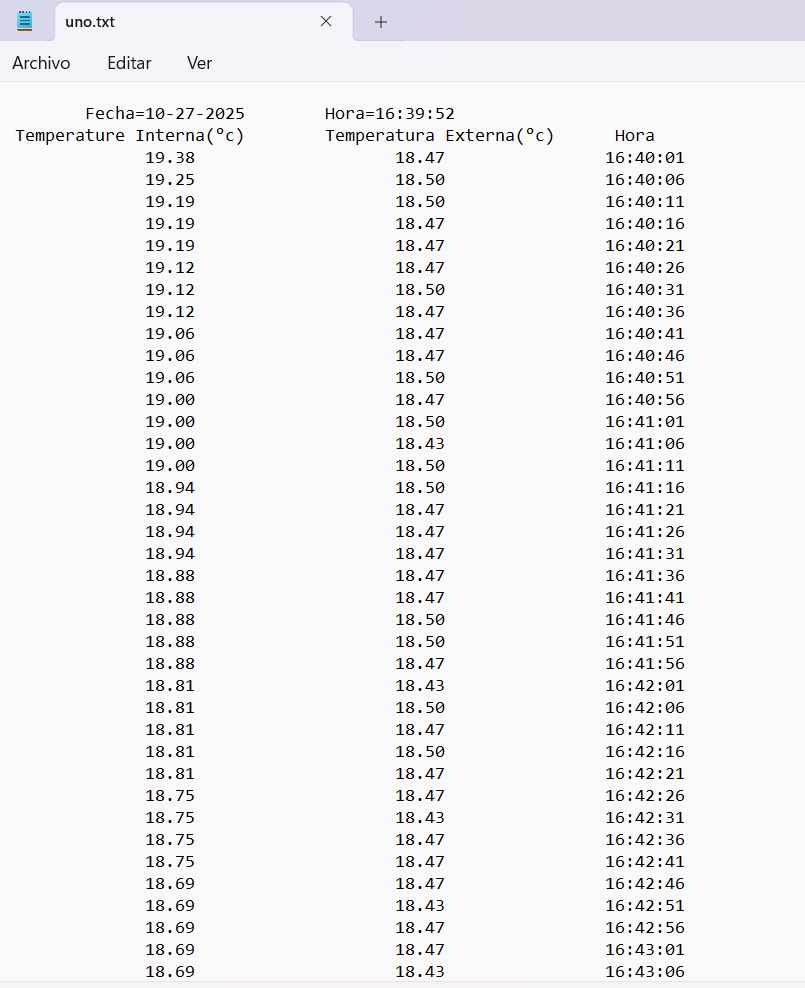


Figura 3.17: Ejemplo de txt de datos guardados.

Gráficos: Desde esta ventana se pueden visualizar el histórico guardado en los txt generador en la pestaña de Lectura de Datos de las temperaturas de un transformador. Además, puede utilizarse un cursor para tomar los datos de temperatura y tiempo más exactos.

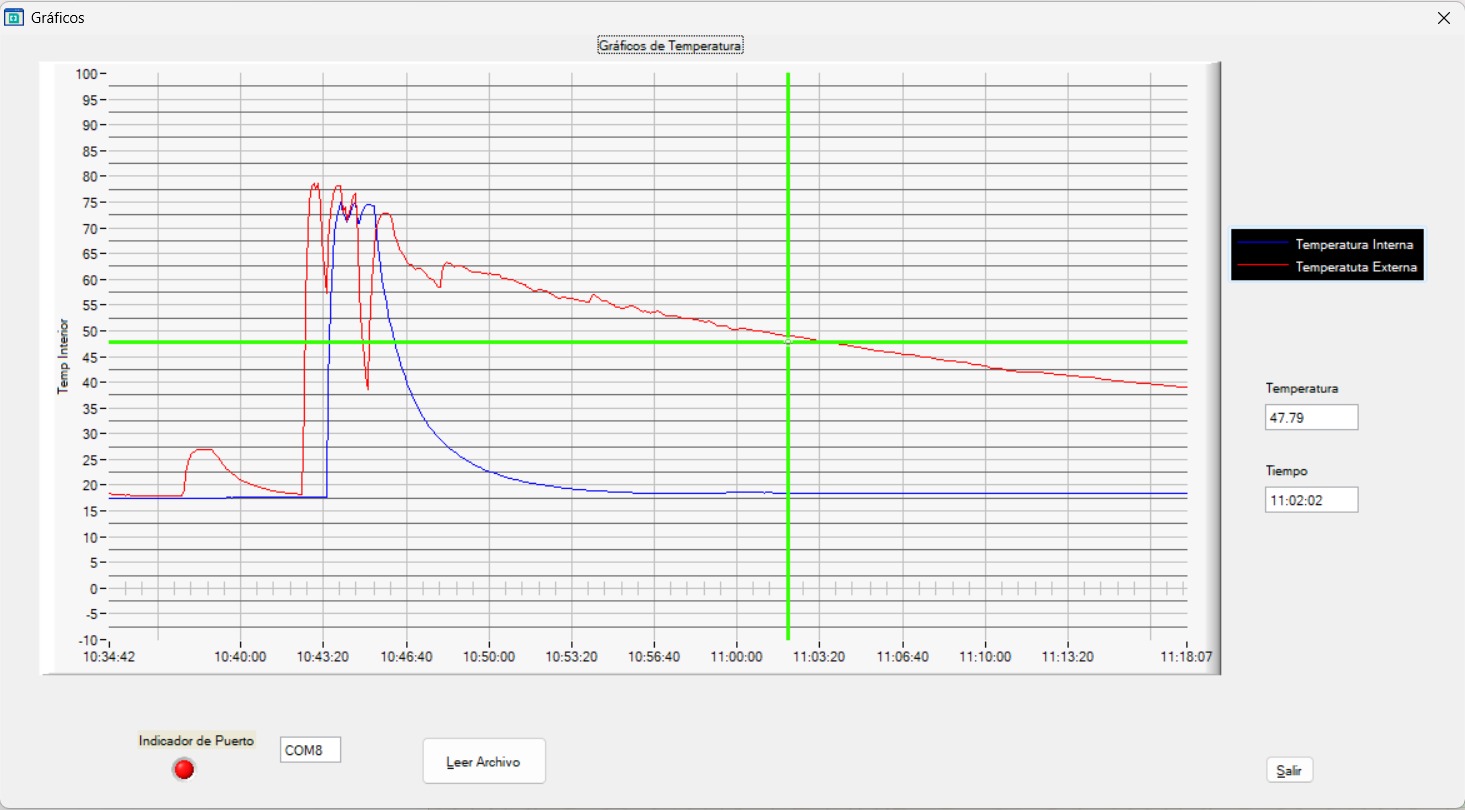


Figura 3.18: Grafico ejemplo de toma de datos Labwindows.

* + 1. Lenguaje ESP-IDF

Se tomó como referencia la biblioteca Espressif ESP-Modbus, la cual permite la comunicación Modbus en redes basadas en interfaces RS485, WiFi y Ethernet. Esta librería admite la creación de varias instancias de objetos maestro y esclavo Modbus. Como se mencionó anteriormente en este método el programa de la placa ESP32 está configurado en modo esclavo, por lo que se mantiene a la espera de un pedido del maestro. Estos comandos u ordenes pueden ser una lectura de temperatura (orden 03) o activación de los equipos externos llamado “Coil” en el entorno de programación (05), lo que se traduce en este proyecto al comando de relés en este caso. En siguiente diagrama en bloques se ilustra el programa con esta lógica mencionada.

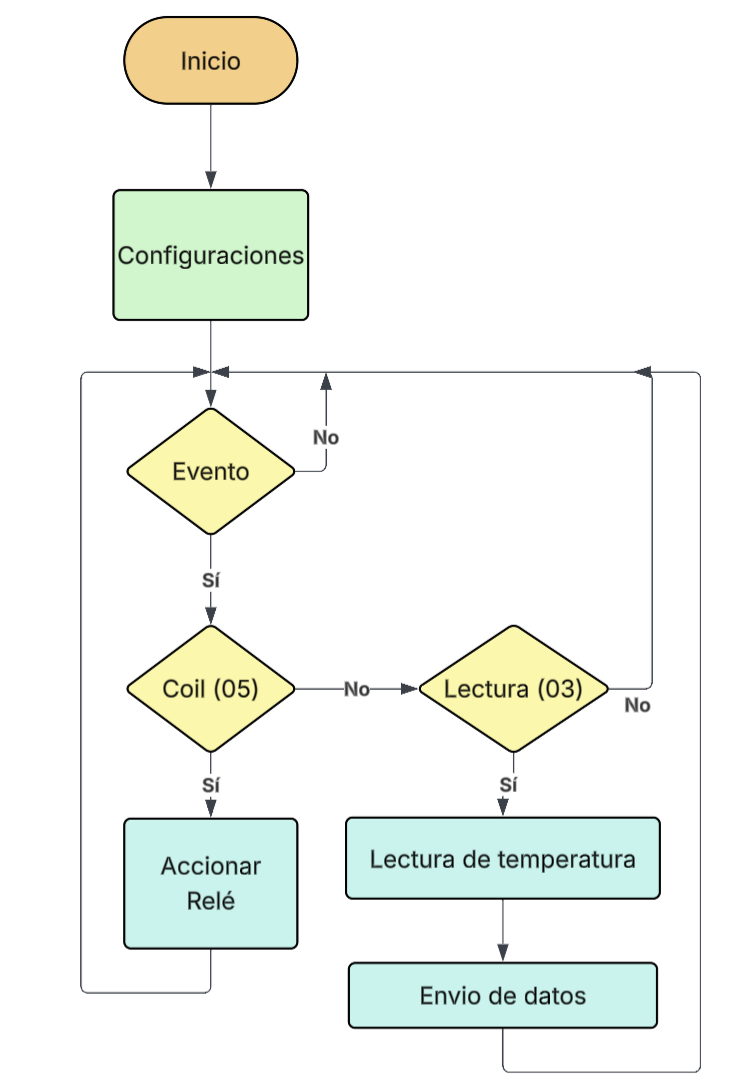


Figura 3.19: Diagrama de flujo del programa ESP para Modbus.

Los puntos destacables del código abarcan la declaración de librerías específicas de modbus

#include "mbcontroller.h" // para mbcontroller definiciones y api

#include "modbus\_params.h" // para estructuras de parametros de modbus

Además de declaraciones comunes a todos los programas de funciones de lectura de las temperaturas, como lo es “sensor\_read” para la lectura de la temperatura del ds18b20 o

La espera por el evento del pedido del maestro y su flujo de decisiones realizado con sentencias IF anidadas.

ESP\_ERROR\_CHECK(mbc\_slave\_get\_param\_info(mbc\_slave\_handle, &reg\_info, MB\_PAR\_INFO\_GET\_TOUT));//espera por un evento

//

if(reg\_info.type &(MB\_EVENT\_HOLDING\_REG\_WR | MB\_EVENT\_HOLDING\_REG\_RD))// IF con Orden 03 para el pedido de datos

//

} else if (reg\_info.type &(MB\_EVENT\_COILS\_RD | MB\_EVENT\_COILS\_WR)) {//IF para activacion de Relés

Poner distintivo RTU yTCPIP

* + 1. TCP/IP
  1. Método B.L.E.

Otro posible desarrollo propuesto para las lecturas de las temperaturas interna y externa de un transformador de Distribución, es el uso de Bluetooth, una tecnología de comunicación inalámbrica de corto alcance que permite la transferencia de datos entre dispositivos electrónicos sin necesidad de cables. Lo que posibilita al operador del Sistema Eléctrico visualizar los registros de temperatura y la operación del equipo al pie de la plataforma en forma segura y confiable.

Acá también dividimos el informe del trabajo en dos partes, una en lo referente a la implementación del código para la ESP32 y otra parte lo que concierne al código e implementación para el dispositivo móvil.

* + 1. Código ESP32

Existen dos tipos de pilas o colecciones de protocolos que [Espressif Systems](https://documentation.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/bluetooth/index.html?title=ESP32%20%20%E8%8A%AF%E7%89%87%20ESP-IDF%20%E7%BC%96%E7%A8%8B%E6%8C%87%E5%8D%97,%20SDK%20latest" \t "_blank) ofrece para sus microcontroladores ESP32 dentro del entorno de desarrollo ESP-IDF, el ESP-Bluedroid y ESP-NimBLE. La principal diferencia entre ellos es la siguiente: ESP-NimBLE requiere menos memoria heap y memoria flash, pero solo funciona para B.L.E, mientras que Bluedroid es más robusta y soporta B.L.E. y Bluetooth Clásico. Se tomó la decisión de utilizar Bluedroid por permitir aplicaciones estándar para la lectura de datos de los sensores de temperatura.

Se tomo como base para el desarrollo, el código ejemplo de la tabla de servicios del servidor GATT para ESP32 de ESP-IDF(GATTS\_SERVER\_\_DEMO). Este ejemplo implementa un servidor de atributos genéricos (GATT) con Bluetooth de baja energía (BLE) utilizando una estructura de datos tabular para definir los servicios y características del servidor. Por lo tanto, facilita la manera de definir la funcionalidad de la estructura GATT, la cual es muy jerárquica y específica.

Lo siguiente, describe los UUID (Identificadores Universales Únicos) que usamos en nuestro proyecto

/\* Servicios \*/

static const uint16\_t GATTS\_SERVICE\_UUID\_TEST = 0x00FF;// UUID del Servicio

static const uint16\_t GATTS\_CHAR\_UUID\_TEST\_A = 0xFF01; //UUID Sensor temperatura DS18B20

static const uint16\_t GATTS\_CHAR\_UUID\_TEST\_B = 0xFF02; //UUID Sesnsor de Temperatura MAX31865

static const uint16\_t GATTS\_CHAR\_UUID\_TEST\_C = 0xFF03; //UIDD para efectuar encendido de Reles

A continuación, se muestra como ejemplo la tabla de Atributos del sensor de temperatura DS18B20 (denominado como TEST\_A), que es la base de datos del servidor GATT (GATTS), lo mismo aplica al sensor de temperatura MAX31865 y al control de los relés. En estas declaraciones se describe la estructura completa de los servicios y características.

/\* Descripción completa de la Base de datos - Se usa para adicionar atributos en la base de datos \*/

static const esp\_gatts\_attr\_db\_t gatt\_db[HRS\_IDX\_NB] =

{

// Declaración de Servicio Principal

[IDX\_SVC] = {{ESP\_GATT\_AUTO\_RSP}, {ESP\_UUID\_LEN\_16, (uint8\_t \*)&primary\_service\_uuid, ESP\_GATT\_PERM\_READ, sizeof(uint16\_t), sizeof(GATTS\_SERVICE\_UUID\_TEST), (uint8\_t \*)&GATTS\_SERVICE\_UUID\_TEST}},

/\* Declaracion de caracteristicas Temperatura externa\*/

[IDX\_CHAR\_A] = {{ESP\_GATT\_RSP\_BY\_APP}, {ESP\_UUID\_LEN\_16, (uint8\_t \*)&character\_declaration\_uuid, ESP\_GATT\_PERM\_READ, CHAR\_DECLARATION\_SIZE, CHAR\_DECLARATION\_SIZE, (uint8\_t \*)&char\_prop\_read\_write\_notify}},

/\* Caracteristicas del valor \*/

[IDX\_CHAR\_VAL\_A] = {{ESP\_GATT\_RSP\_BY\_APP}, {ESP\_UUID\_LEN\_16, (uint8\_t \*)&GATTS\_CHAR\_UUID\_TEST\_A, ESP\_GATT\_PERM\_READ | ESP\_GATT\_PERM\_WRITE,

GATTS\_DEMO\_CHAR\_VAL\_LEN\_MAX, sizeof(char\_value), (uint8\_t \*)char\_value}},

/\* Descriptor de Configuración del Cliente/Servidor CCCD \*/

[IDX\_CHAR\_CFG\_A] = {{ESP\_GATT\_RSP\_BY\_APP}, {ESP\_UUID\_LEN\_16, (uint8\_t \*)&character\_client\_config\_uuid, ESP\_GATT\_PERM\_READ | ESP\_GATT\_PERM\_WRITE,

sizeof(uint16\_t), sizeof(heart\_measurement\_ccc), (uint8\_t \*)heart\_measurement\_ccc}},

Y por último se incluye al informe partes de la función **static void gatts\_profile\_event\_handler**, la cual es una función de *callback* (devolución de llamada) muy importante en el proyecto. Su propósito principal es atender todos los eventos relacionados con el perfil GATT definido en la base datos del BLE.

case ESP\_GATTS\_WRITE\_EVT:

if (!param->write.is\_prep){

ESP\_LOGI(GATTS\_TABLE\_TAG, "GATT\_WRITE\_EVT, handle = %d, value len = %d, value :", param->write.handle, param->write.len);

ESP\_LOG\_BUFFER\_HEX(GATTS\_TABLE\_TAG, param->write.value, param->write.len);

// --- Lógica para controlar LEDsVerde (Característica C) ---

if (param->write.handle == heart\_rate\_handle\_table[IDX\_CHAR\_VAL\_C]) {

if (param->write.len == 1) {

uint8\_t led\_state = param->write.value[0]; // Leer el primer byte del valor

if (led\_state==1) clear\_led\_rojo(led\_state); // Llama a tu función (temporal o real en main.c)

if (led\_state==2) set\_led\_rojo(led\_state); // Llama a tu función (temporal o real en main.c)

if (led\_state==3) clear\_led\_verde(led\_state); // Llama a tu función (temporal o real en main.c)

if (led\_state==4) set\_led\_verde(led\_state); // Llama a tu función (temporal o real en main.c)

ESP\_LOGI(GATTS\_TABLE\_TAG, "LED Estado de los LEDS: %d", led\_state);

} }

Este case ESP\_GATTS\_WRITE\_EVT: sque se encuentra en gatts\_profile\_event\_handler, es responsable de manejar la situación en la que **un cliente BLE escribe datos en una característica**para luego ejecutar la lógica de aplicación.

case ESP\_GATTS\_READ\_EVT://

ESP\_LOGI(GATTS\_TABLE\_TAG, "ESP\_GATTS\_READ\_EVT");

{

ESP\_LOGI(GATTS\_TABLE\_TAG, "ESP\_GATTS\_READ\_EVT, handle = %d", param->read.handle);

// Si el handle leído coincide con la Característica A del proyecto estándar:

if (param->read.handle == heart\_rate\_handle\_table[IDX\_CHAR\_VAL\_A]) {

// Obtenemos la temperatura (usando tu función 'extern' de main.c)

sensor\_read();//Leo la temperatura del DS18B20

memcpy(rsp.attr\_value.value, &s\_temperature, sizeof(float));

rsp.attr\_value.len = sizeof(float);

}

if (param->read.handle == heart\_rate\_handle\_table[IDX\_CHAR\_VAL\_B]) {

// Obtenemos la temperatura (usando tu función 'extern' de main.c)

float temp\_celsius2 = get\_max31865\_temperature();

memcpy(rsp.attr\_value.value, &temp\_celsius2, sizeof(float));

rsp.attr\_value.len = sizeof(float);

}

esp\_ble\_gatts\_send\_response(gatts\_if, param->read.conn\_id, param->read.trans\_id, ESP\_GATT\_OK, &rsp);

break;

}

El case ESP\_GATTS\_READ\_EVT: dentro del manejador de eventos GATT cumple la función de responder cuando un cliente BLE solicita leer el valor actual de una característica, como ser en este proyecto las temperaturas Interna y Externa.

3.3.2 Código Móvil (Flutter)

La elección del lenguaje y plataforma de desarrollo para la hacer la aplicación que se ejcute en el móvil, requiero mucho tiempo y consultas previas, al no tener conocimiento en el uso de estas herramientas. Se analizó las prestaciones y disponibilidad de Android Studio, IDE oficial para desarrollo de APPs de Android; también se analizó IoT Blink" , es una plataforma de IoT de bajo código que permite a los usuarios crear aplicaciones y controlar dispositivos a través de Internet , pero se eligio trabajar con Flutter , que es herramienta de código abierto desarrollado por Google, que permite a los desarrolladores crear aplicaciones multiplataforma . Las razones de esta elección son que FLUTTER es multiplataforma (Android e IOS), más amigable, utiliza el lenguaje DART que es rápido y de fácil aprendizaje, mientras que a IoT Blink se lo descartó rápidamente al haber discontinuado el soporte para BLE.

Otra situación que se contempló es que Flutter y Dart utiliza VS Code . A diferencia de Android Studio que es un IDE complejo y pesado, si bien se utiliza su SDK de Android (nota)y sus herramientas de compilación.

Nota1: El **SDK de Android** (Software Development Kit o Kit de Desarrollo de Software) es un conjunto fundamental de herramientas, bibliotecas y documentación que permite a los programadores crear aplicaciones para el sistema operativo Android. Es la base que necesitan los desarrolladores para interactuar con todas las funcionalidades de un dispositivo Android.

Para que Flutter pueda utilizar la tecnología BLE  , se utiliza el componente flutter\_blue\_plus , que es un paquete más difundido y con muy buen soporte, ofrece una API (Nota2) completa para escanear, conectar, descubrir servicios, leer, escribir y suscribirse a características BLE. Es compatible con Android, iOS, macOS, y en desarrollo para Windows/Linux. La versión utilizada es flutter\_blue\_plus: ^2.0.2

Nota2: Una API (Interfaz de Programación de Aplicaciones) es un conjunto de reglas, protocolos y herramientas que permiten que diferentes programas de software se comuniquen entre sí, intercambien datos y realicen funciones.

El proyecto tiene dos archivo fundamentales, que son main.dart , es el primer archivo que se ejecuta cuando inicias la aplicación. Su función principal es inicializar el framework de Flutter y lanzar el primer widget que representará la interfaz de usuario. y ble\_service.dart , su propósito es **abstraer**la lógica BLE del resto de la interfaz de usuario, haciendo el código más limpio y fácil de mantener. Dentro podemos mencionar por su importancia a los siguientes códigos:

Future<void> \_startProcess() async { setState(() => \_status = "Solicitando permisos...");

// Solicitar todos los permisos necesarios

Map<Permission, PermissionStatus> statuses = await [

Permission.bluetoothScan,

Permission.bluetoothConnect,

Permission.location,

].request();

if (statuses[Permission.bluetoothScan]!.isGranted &&

statuses[Permission.bluetoothConnect]!.isGranted) {

setState(() => \_status = "Permisos OK. Conectando...");

try {

await \_bleService.connectToESP32();

setState(() => \_status = "✅ Conectado exitosamente.");

} catch (e) { setState(() => \_status = "❌ Falló la conexión: ${e.toString()}");

}

} else { setState(() => \_status = "❌ Permisos denegados.");

}

Este código se encuentra en main.dart y es responsable de conectarse a la placa ESP32 por BLE.

Future<void> connectToESP32() async {

if (!await FlutterBluePlus.isOn) {

throw Exception("Bluetooth no está activado.");

}

// --- Lógica de Escaneo ---

await FlutterBluePlus.startScan(timeout: Duration(seconds: 10));

var subscription = FlutterBluePlus.scanResults.listen((results) {

for (ScanResult r in results) {

if (r.device.platformName == DEVICE\_NAME) { device = r.device;

FlutterBluePlus.stopScan();

break;

} } });

await FlutterBluePlus.isScanning.where((val) => val == false).first;

await subscription.cancel();

if (device == null) { throw Exception("No se encontró el ESP32 con nombre $DEVICE\_NAME."); }

// --- Lógica de Conexión ---

await device!.connect(

license: License.free, // 'license' ya no es necesario en versiones recientes de fbp

timeout: Duration(seconds: 10),

autoConnect: false, );

// --- Lógica de Descubrimiento de Servicios y Características ---

List<BluetoothService> services = await device!.discoverServices(subscribeToServicesChanged: false);

bool foundService = false;

for (var service in services) { String serviceUuidNormalized = \_normalizeUuid(service.uuid.toString());

if (serviceUuidNormalized == SERVICE\_UUID.toUpperCase()) {

print(" ¡Coincidencia de Servicio 0x00FF! ");

foundService = true;

for (var characteristic in service.characteristics) {

String charUuidNormalized = \_normalizeUuid(characteristic.uuid.toString());

print(" - UUID de característica (Normalizado): $charUuidNormalized");

if (charUuidNormalized == \_normalizeUuid(TEMP\_CHAR\_1\_UUID)) {

\_tempChar1 = characteristic;

await \_readTemperature(\_tempChar1!, \_temp1Controller);

} else if (charUuidNormalized == \_normalizeUuid(TEMP\_CHAR\_2\_UUID)) {

\_tempChar2 = characteristic;

await \_readTemperature(\_tempChar2!, \_temp2Controller);

} else if (charUuidNormalized == \_normalizeUuid(LED\_CONTROL\_UUID)) {

\_ledControlChar = characteristic;

} } } }

if (!foundService || \_ledControlChar == null || \_tempChar1 == null || \_tempChar2 == null) {

throw Exception("No se encontraron todos los servicios/características GATT esperados.");

}

print("Conexión y descubrimiento completados con éxito."); }

Obs: los códigos completos del firmware correspondiente a la placa ESP32 y las aplicaciones para el servidor o movil, se encuentran alojados en GitHub,, xxxxxx

# Capítulo 4 - Ensayos y resultados

Este capítulo expondrá los ensayos y resultados obtenidos en función de las metodologías planteadas durante el capítulo anterior. Primero se mostrará las comunicaciones de las pruebas preliminares, donde se corrobora el buen funcionamiento de los algoritmos de las distintas aplicaciones. Luego se enseñarán los resultados pertenecientes a las pruebas con medición de recursos consumidos, que son los experimentos base para analizar la viabilidad de aplicación de uno o más diseños.

En paralelo, el análisis practico y la ejecución de una puesta en marcha abarcando la instalación del prototipo y sus condiciones reales.

## Pruebas preliminares

En el caso del desarrollo de Thingsboard como primera instancia se utilizó el bróker mosquitto con el cual se apuntó a un dispositivo en uso para visualizar y enviar comandos desde el tablero diseñado.

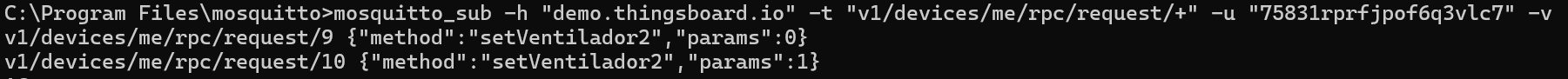


Figura 4.1: Suscripción y mensajes RCP.

En la imagen podemos ver la suscripción al dispositivo, es decir un comando para visualizar los mensajes de RCP que ocurran a la dirección de nuestro dispositivo, donde se observan los mensajes con el formato método;parámetro. Con esta herramienta se realizaron pruebas a fin de concretar la correcta comunicación entre el tablero de Thingsboard y las temperaturas medidas.

Para el método de Modbus se utilizó una herramienta llamada Modbus Poll la cual fue utilizada de forma análoga a mosquitto para Thingsboard. Con ella se pudieron enviar comandos y visualizar los mensajes que emitía la placa en respuesta a ellos, dada la función de esclavo de la ESP32. Para lograrlo se debe iniciar con la configuración, como se ilustra en la siguiente figura, conexión tipo serial, el puerto y la velocidad, tipo de mensajes RTU, el tiempo de espera por una respuesta entre otros.

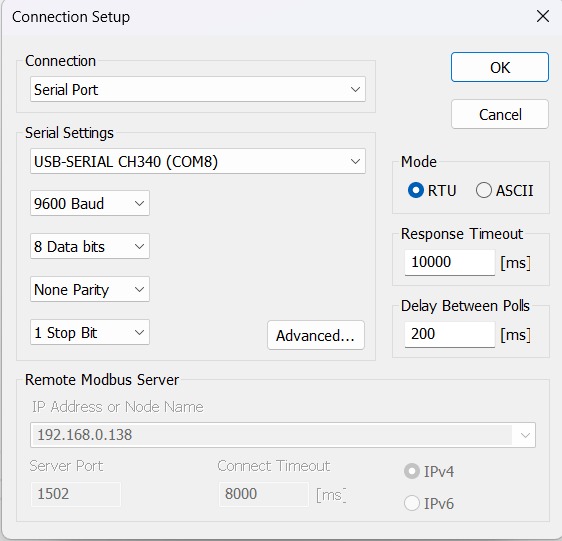


Figura 4.2: Configuraciones de Modbus Poll.

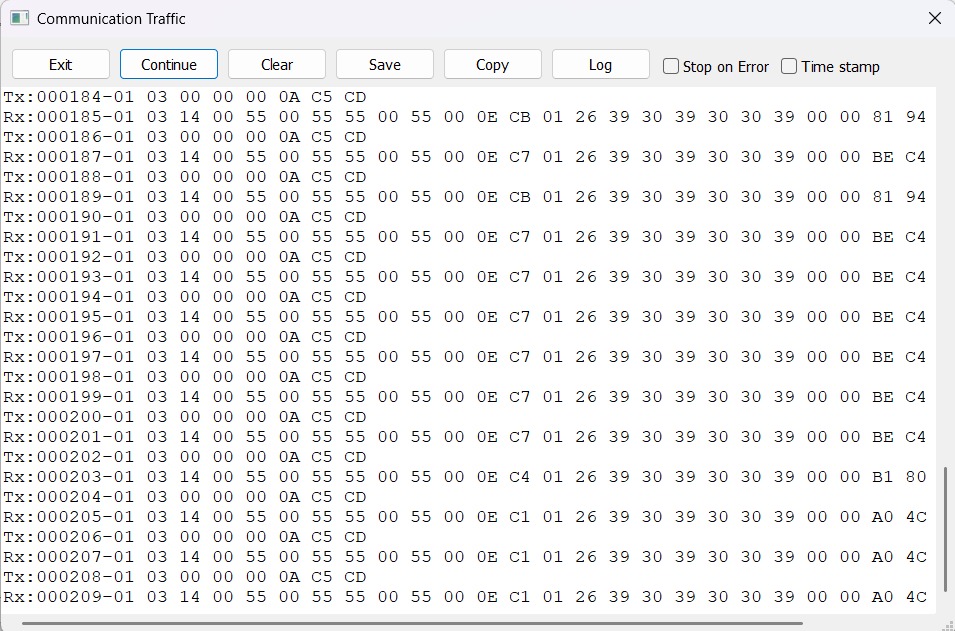


Figura 4.3: Cadena de mensajes al dispositivo 01.

De estos mensajes podemos notar que indican la dirección Transmisor Tx o Receptor Rx, el número de mensaje y la trama correspondiente a la comunicación tipo Modbus RTU, como por ejemplo se ilustra en la siguiente imagen el análisis de una trama de transmisión.

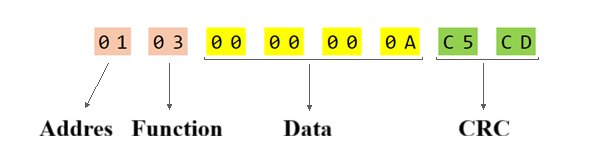


Figura 4.4: Análisis de trama Modbus RTU

En la cual “01” es el número de dispositivo, “03” la función lectura de registro “00 00 00 0A” el numero o cantidad de registros (10 registros) y por último “C5 CD” la comprobación de redundancia cíclica.

Por otro lado, con Modbus Poll podemos observar el resultado de la transmisión de datos, figura 4.5, donde en este caso se obtuvieron 1847 mensajes sin errores y los últimos valores de los 10 registros pedidos, en este caso por ejemplo los datos de temperatura situados en el 4º y 5º lugar.

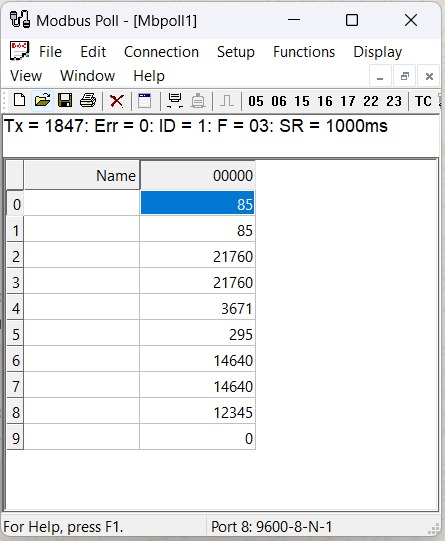


Figura 4.5: Resultado de comunicación Modbus RTU con Modbus Poll.

## Bluetooth BLE

## 

Los pasos que se siguieron fueron los siguiente: entender el funcionamiento de la pila (stack) de Bluedroid, incorporar los componentes de los sensores DS18B20 y MAX31865 para el lenguaje esp-idf y finalmente adicionar la lógica de operación del Rele1 y 2 , depurar el código y programar la placa ESP32. Luego de la concreción de estas tareas la dificultad se presento en comprobar que las lecturas de las temperaturas y el accionar de los reles funcionen correctamente, para ello fue de mucha ayuda el uso de la aplicación de nRF Connect for Moviles, que permite explorar, interactuar y analizar dispositivos BLE .

## 

En la figura se observa que nRf Connect detectó el dispositivo BLE, denominado TESIS\_TRAFO

## 

En la figura se puede observar los valores crudos de la temperatura interna y externa, la representación de estos valores flotantes están en un formato de little indian de 32bits.

## 

En la figura se visualiza la operación de escritura sobre el UUID 0xFF03, con la finalidad de operar los Reles de la placa.

## La aplicación para el móvil , denominada “Transformadores” , como ya fue mencionado en el capitulo anterior se desarrolló con la plataforma Flutter, inicialmente se genera la aplicación “app-release.apk” mediante la seguidilla de comandos desde el terminal de VSCode , primero un comando de limpieza de archivos precompilados ,ejecutables y otros :

## PS C:\Flutter\Ejemplo\Tesis\_BLE\_F> flutter clean

## Deleting build... 2.550ms

## Deleting .dart\_tool... 103ms

## Deleting ephemeral... 5ms

## Deleting Generated.xcconfig... 15ms

## Deleting flutter\_export\_environment.sh... 1ms

## Deleting ephemeral... 1ms

## Deleting ephemeral... 2ms

## Deleting ephemeral... 1ms

## Deleting .flutter-plugins-dependencies... 0ms

## Y luego se genera el ejecutable:

## PS C:\Flutter\Ejemplo\Tesis\_BLE\_F> flutter build apk

## Resolving dependencies...

## Downloading packages...

## Got dependencies!

## 10 packages have newer versions incompatible with dependency constraints.

## Try `flutter pub outdated` for more information.

## Font asset "MaterialIcons-Regular.otf" was tree-shaken, reducing it from 1645184 to 1312 bytes (99.9% reduction). Tree-shaking can be disabled by providing the --no-tree-shake-icons flag when building your app.

## Running Gradle task 'assembleRelease'... 162,9s

## √ Built build\app\outputs\flutter-apk\app-release.apk (41.2MB)

## 

## 

## En el móvil se vería según figura:

## 

## O esta

## Al inicial la aplicación Transformadores se visualiza la siguiente pantalla:

## 

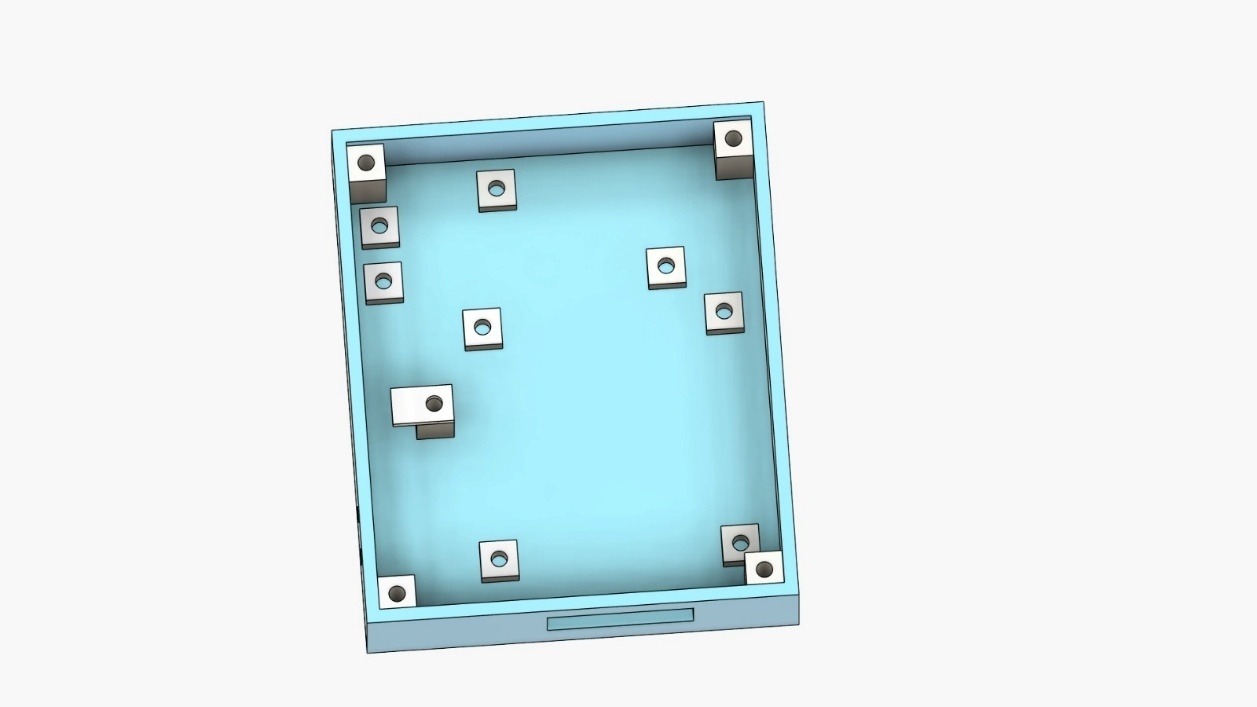
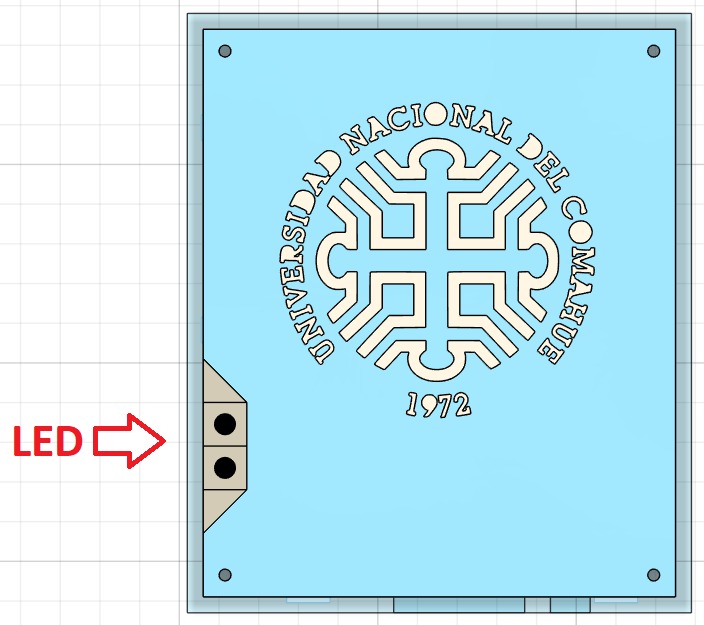
## Se observa que la aplicación inicialmente, no se encuentra conectada con la Placa ESP32 , según mensaje “Listo para conectar manualmente”, la conexión se realiza mediante el botón “Conectar a ESP32 Manualmente”. Si la verificación de los permisos correspondiente al Perfil de Acceso Generico (GAP) que es la encargada de definir como los dispositivos de descubren y conectan, mas los permisos de aceso a los atributos definidos por el Perfil de Atributos Generico (GATT), es exitoso , la pantalla de la aplicación se ve de la siguiente manera:

## 

## Las lecturas de las temperaturas Interna (Temp 1)y Externa (Temp 2) del Transformador se obtiene con el botón “Refrescar Temperaturas Manualmente”, para abrir o cerrar los Reles 1 y 2 se incluyen los pulsadores correspondiente para cada Rele y operación (Abrir o Cerrar).

## Propuesta de abordaje en campo o costos

Para la implementación en la práctica se desarrolló una caja o carcaza para almacenar y proteger todas las placas pero al mismo tiempo la posibilidad de conexión usb, indicaciones luminosas de funcionamiento y encendido. Este fue desarrollado en PROGRAMA para realizar su desarrollo en una impresora tipo 3D



Tras haber presentado los resultados de los ensayos en la sección anterior, se está en condiciones de plantear una propuesta de cómo se podría abordar la implementación del CCM- SLAMencampo.Lapresenteseccióndetallarácuálesseríanlosrequisitosmínimosacumplir para que esto sea factible, y cómo debería ser la disposición del agente, servidor y conexiones, considerando un transformador de distribución típico.

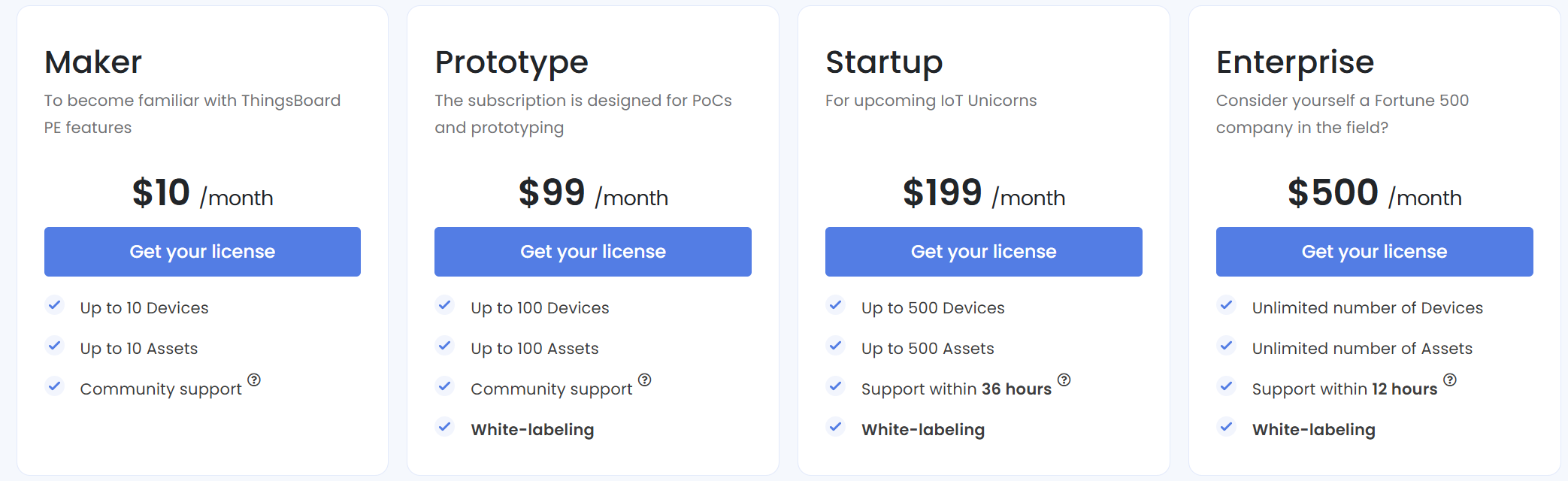
Dentro de los costos asociados a cada aplicación y desarrollo se engloba el hardware donde cabe destacar la siguiente planilla, con esto se podría estimar la viabilidad de la implementación de los métodos.

Para la

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **Equipo** | **Descripcion** | **Valor unitario** | **Link** |
| **1** | ESP32 S3 | Placa microporcesada con BLE, WiFI y GPIOS | 26000S | <https://surl.li/sklrbj> |
| **2** | RS485 | Driver de comunicaciones RS485 | 3000$ | https://surli.cc/dfkrju |
| **3** | MAX31865 y PT100 | Placa senor PT100 | 65000$ | https://surl.li/xgnxcw |
| **4** | DS18B20 | Sensor temperatura | 5000$ | https://surl.li/wxxvsw |
| **5** | Placa Base ESP32 S3 | Placa interconexion para ESP32 S3 y perifericos | 25000$ | https://surl.li/jgwejf |
| **6** | Modulo dos reles | Placa don dos reles independientes | 5000$ | https://surl.li/ykiivs |
| **7** | Gabinete |  |  |  |

Tabla X.X: Tabla de valores de Hardware Poner en anexo?

Por el lado de Thingsboard este trabajo se realizó con la versión demo gratuita, que, aunque posee limitaciones se adapta a la idea principal planteada. En la página oficial de esta plataforma se tienen distintos planes con sus elementos disponibles, así como cantidad de dispositivos a conectar, assets y soporte informático.



Poner?

Con Labwindows…

Se estima una pérdida de un transformador y medio por mes por sobrecalentamiento, ya sea por deficiencias en su uso o inconvenientes en la línea, con los problemas que esto conlleva, como tiempo no productivo o reclamos por falta de servicio.

## Resultados

Como resultados cabe destacar la implementación real de los dispositivos propuestos

Para poner a prueba los distintos métodos, como se presentó en el capítulo 4, fue necesario primero, hacer pruebas para entender cómo funcionaba un bróker como los mosquitto y luego Thingsboard y cómo se llevarían a cabo las mediciones y uso de las plataformas. La segunda parte consistió en los ensayos sobre placas de hardware embebido, que fueron las tres ESP32-S3. Tanto paralasmedicionescomoparalaspruebassobrelasplacasseespecificarondiferentes programas y formas de comunicación.

Dentro de lo desarrollado en base a IoT, se tiene una aplicación dinámica y agradable para el usuario en el cual pueden desarrollarse y verificarse desde cualquier punto con acceso a internet. Aunque es determinante la necesidad de esto último, tanto para el equipo como para el usuario.

Por otro lado, se obtuvo el desarrollo de la comunicación con Modbus RTU, un protocolo de los más utilizados y difundidosen la industria, entre plataformas como SCADA.

Nodo recolector en cada una de las localidades donde se u¡implementa el sistema o concentrar la implementación en una subestacion eléctrica

No se puede acceder fácilmente en forma remota o hay que replicar en cada uno de los lugare que se quiere tener las mediciones

En el caso del desarrollo de Modbus TCP/IP es necesario tener una red interna inalámbrica o en forma cableada que permite la comunicación de los dispositivos, además con lo desarrollado en XXXX se necesita de un servidor en dicha red para lograr la comodidad de la accesibilidad de cualquier equipo dentro de esta intranet de la empresa.

Tcpip comodidad de la aplicación se encuentra corriendo en un servidor peromitieno acceder desde cualquier otro equipo dentro de la intranet de la empresa

Malo servidor funcionando

El desarrollo de la interfaz en Labwindows permite al usuario utilizar tanto un equipo programado con Modbur RTU como con TCP/IP, lo que la hace más versátil. Al mismo tiempo es una aplicación instalable en cualquier computadora dándole mayor versatilidad a la misma y agrega la posibilidad de exportar los datos en forma sencilla en un .txt. Aunque los datos no se grafiquen en tiempo real se tienen en pantalla a fin de realizar su monitoreo.

Aunque la implementación real escapa a las posibilidades inmediatas, ya que para la implementación de estos equipos debe sacarse de servicio el transformador completo con el servicio que esto conlleva, se ha implementado simulando o haciendo una analogía a las condiciones de uso resultando en un funcionamiento satisfactorio.

# Capítulo 5 - Conclusiones

El presente capítulo tiene como fin, en primera instancia, resumir el desarrollo presentado a lo largo de todo el trabajo. Luego de esto, la siguiente sección presentará las conclusiones obtenidas en función de los objetivos planteados en la Sección[1.2](#_bookmark5) y de los resultados redactados en el Capítulo [4](#_bookmark33). Para finalizar, se dará cierre al trabajo y se propondrán posibles acciones o mejoras a seguir.

## Resumen del trabajo realizado

Este trabajo comenzó en el marco de la iniciativa de solventar una falta de monitoreo o control preventivo sobre la temperatura de un transformador a modo de evitar posibles fallas sobre el mismo. Durante este trabajo, fue necesario investigar sobre las situaciones reales que se encuentran los transformadores de distribución tanto en la zona como lugares mas remotos dentro de la provincia de Neuquén, de forma de poder aplicar los distintos métodos en el entorno deseado. Así como normas internacionales y distintas metodologías de desarrollo, requiriendo la comprensión de los conceptos teóricos que se introdujeron durante el Capítulo [2](#_bookmark7).

## Conclusiones

Luego de lo presentado a lo largo del Capítulo [4](#_bookmark33), se puede concluir que se logró exitosamente la implementación y análisis de la medición de temperatura de un transformador de distribución. En el estudio sobre esta aplicación se evaluó un gran abanico de posibles configuraciones o dificultades que conlleva llevara la práctica este tipo de aplicaciones, además de qué ventajas y desventajas implica cada una de ellas.

En conclusión, se lograron tres variantes dentro de los objetivos propuestos con vasto desarrollo, complementación con practica y analizados desde la comodidad del usuario a fin. De estos modelos puede seleccionarse el que mejor se adapte a las características del entorno de trabajo de los transformadores a medir, teniendo como opción adicional la posibilidad de conectar un dispositivo externo o accionar sobre el mismo.

Ya que en comparación con instrumentos similares de venta libre dentro de la industria se tiene una mayor injerencia en los instrumentos o accesorios aplicados, dándole a este proyecto la ventaja de ser completamente ideado a fines de las necesidades planteadas en un principio. Además de una posible comparativa en costos fijos o variables.

## Cierre del trabajo y propuestas de mejora

Luego de analizar los resultados y las variantes de este trabajo, en conclusión, se obtuvo el prototipo que cumple con los objetivos planteados aun teniendo lugar a diversas mejoras. Entre estas propuestas de mejora podemos nombrar el desarrollo de la capacidad de ESP-Now de las placas ESP32, con el cual puede realizarse una red inalámbrica entre distintos equipos teniendo en cuenta la distancia entre si con cualquier condición de trabajo. Ya que no necesita un router o punto de acceso Wi-Fi para que los dispositivos se comuniquen entre sí, opera en la misma banda de frecuencia que Wi-Fi, pero sin requerir una conexión de red y sin interferir con ella. Permite configuraciones de uno a muchos, muchos a uno o muchos a muchos y es ideal para esta aplicación donde los mensajes son cortos.

Por otro lado, se puede mejorar la seguridad de la plataforma online realizada con Thingsboard. La seguridad de los datos es una prioridad en cualquier sistema de monitoreo. Como mejora futura, se propone implementar un sistema de autenticación multifactor y protocolos de cifrado más avanzados para garantizar la privacidad y la seguridad de la información almacenada y transmitida. Al mismo tiempo agregar la posibilidad de exportar los datos de las mediciones, al tener acceso a uno de los planes de Thingsboard se tiene el fácil acceso a esta posibilidad.

Otra mejora sería analizar la posibilidad y capacidad de recolectar todos los programas realizados en ESP-IDF en un mismo archivo compilado con la posibilidad de configurar el método elegido por medio de Bluetooth. Lo que facilitaría la estandarización de la programación de la placa en lugar de programar según la aplicación que se necesite.

Finalmente, otro posible trabajo futuro sería agregar una medición en los cables de salida de cada fase a fin de monitorear la distribución de carga realizada y tener la información necesaria para reacomodar dicha carga y evitar malos funcionamientos de forma preventiva. Con lo que se enriquece la funcionabilidad a un bajo costo, ya que el hardware seleccionado podría incluir estos sensores extra sin inconvenientes. Además, podría monitorearse las vibraciones de los transformadores para darle un seguimiento de forma de anticipar deficiencias o mal funcionamiento del mismo agregando en comunicación con la placa un acelerómetro en contacto con el equipo. La detección de dichas vibraciones permite prevenir fallas, programar tareas de mantenimiento preventivo y realizar ajustes en partes mecánicas para asegurar un funcionamiento sin inconvenientes.

Otra línea de trabajo es equipar al dispositivo PMTT con un dispositivo de comunicación con tecnología LoRa (Long Range), esta tecnología inalámbrica es muy recomendable, de uso incipiente pero con mucho futuro, ya que permite integrar dentro de una red con requerimientos y características de IOT, todos los dispositivos de operación, protección y medición de una red Eléctrica de Distribución, como los medidores domiciliarios y comerciales, tanto directos como indirectos.

Otra línea de trabajo muy importante y aplicable a la medición inteligente de energía en las empresas de Distribución de Energía, es el uso del protocolo IEC 62056 (DLMS/COSEM), ya que se puede asignar un OBIS (Object Identification System) especifico a las mediciones de temperatura PMTT y así vincularlos con los software de medición comerciales. Es decir ….

implementa el protocolo MQTT mediante la programación de código en lenguaje Script en dispositivos compatibles. Esta implementación permite una comunicación eficiente y estandarizada entre equipos, facilitando así la integración y el intercambio de datos en entornos industriales.

Anexos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Item | Equipo | Descripcion | Valor unitario | Link |
| 1 | ESP32 S3 | Placa microporcesada con BLE, WiFI y GPIOS | 26000S | <https://surl.li/sklrbj> |
| 2 | RS485 | Driver de comunicaciones RS485 | 3000$ | <https://surli.cc/dfkrju> |
| 3 | MAX31865 y PT100 | Placa senor PT100 | 65000$ | <https://surl.li/xgnxcw> |
| 4 | DS18B20 | Sensor temperatura | 5000$ | <https://surl.li/wxxvsw> |
| 5 | Placa Base ESP32 S3 | Placa interconexion para ESP32 S3 y perifericos | 25000$ | <https://surl.li/jgwejf> |
| 6 | Modulo dos reles | Placa don dos reles independientes | 5000$ | <https://surl.li/ykiivs> |
| 7 | Gabinete |  |  |  |

# Bibliografía

1. Pedro Avelino Perez. *Transformadores de distribución, Teoria, calulo , construcción y pruebas* 3ª edición 2008 Mexico DF ISBN 978-84-291-9358-9
2. INTRO TO BLUETOOTH LOW ENERGY by Mohammad Afaneh
3. Página oficial de International Electrotechnical Commission I.E.C
4. Página oficial Analog Devices, Hojas de datos del integrado MAX31865.
5. Página oficial Analog Devices, Hojas de datos del integrado MAX485.
6. Página oficial Espressif , hojas de datos la placa ESP32 S3.
7. Página oficial ModBus Organization, manual del protocolo ModBus.
8. Página oficial Analog Devices, Hojas de datos del integrado DS18B20.
9. Página oficial MQTT Organization, manual del protocolo MQTT.