UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE ELECTRICIDAD

TEMA:

IMPLEMENTACIÓN DE UN SISTEMA INTELIGENTE DE MEDICIÓN DE CONSUMO ELÉCTRICO RESIDENCIAL MEDIANTE HOME ASSISTANT.

Trabajo de Grado previo a la obtención del título de Ingeniero Eléctrico

Línea de investigación: Biotecnología, energía y recursos naturales renovables

AUTOR:

Erick Santiago Condo Nevárez

DIRECTOR:

Ing. Francisco Roberto Naranjo Cobo MSc.

Ibarra, 2024

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	2350560245		
APELLIDOS Y NOMBRES:	Condo Nevárez Erick Santiago		
DIRECCIÓN:	Joaquín Balaguer y Pablo Neruda, Santo Domingo de los Colorados, Santo Domingo de los Tsáchilas, Ecuador.		
EMAIL:	escondon@utn.edu.ec		
TELÉFONO FIJO:	2747994	**TELÉFONO MÓVIL:**	0994796445

DATOS DE LA OBRA	
TÍTULO:	Implementación de un Sistema Inteligente de Medición de Consumo Eléctrico Residencial mediante Home Assistant.
AUTOR (ES):	Condo Nevárez Erick Santiago
FECHA DE APROBACIÓN: DD/MM/AAAA	18/06/2024
PROGRAMA:	■ **PREGRADO** ☐ **POSGRADO**
TITULO POR EL QUE OPTA:	Ingeniero(a) Eléctrico(a)
ASESOR /DIRECTOR:	Ing. Francisco Roberto Naranjo Cobo MSc

2. CONSTANCIAS

El autor (es) manifiesta (n) que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es (son) el (los) titular (es) de los derechos patrimoniales, por lo que asume (n) la responsabilidad sobre el contenido de esta y saldrá (n) en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 18 días del mes de junio de 2024

EL AUTOR:

.....

Nombre: Erick Santiago Condo Nevárez

CERTIFICADO DEL DIRECTOR DE TRABAJO DE INTEGRACIÓN CURRICULAR

Yo, Francisco Roberto Naranjo Cobo, en calidad de director del señor estudiante Erick Santiago Condo Nevárez certifico que ha culminado con las normas establecidas en la elaboración del Trabajo de Integración Curricular con el tema: Implementación de un Sistema Inteligente de Medición de Consumo Eléctrico Residencial mediante Home Assistant.

Para la obtención del título de Ingeniero Eléctrico, aprobado la defensa, impresión y empastado.

Ing. Francisco Naranjo Roberto Cobo MSc.
DIRECTOR DE TRABAJO DE INTEGRACIÓN CURRICULAR

DEDICATORIA

Este trabajo de grado está dedicado a las figuras más fundamentales en mi vida: a mi mamá y a mi papá. Ellos han sido el verdadero motor de mi esfuerzo, inspirándome a dar siempre lo mejor de mí gracias a su amor incondicional, su sacrificio sin límites y un apoyo tan firme como inquebrantable.

Extiendo esta dedicación a mis hermanos y amigos, quienes han estado a mi lado tanto en los buenos como en los malos momentos. Desde el inicio de mi aventura universitaria hasta el final, su presencia constante ha sido el ancla que me ha proporcionado la estabilidad emocional y mental necesaria para sobresalir en mis estudios.

Dedico este logro a cada individuo que ha formado parte de este viaje, de maneras grandes y pequeñas. Cada uno de ustedes ha dejado una huella indeleble en mi camino hacia este momento.

AGRADECIMIENTO

Agradezco a todos los miembros de mi familia que siempre me llenaron de buenos deseos en lo que a mi carrera universitaria se refiere.

También, expreso mi sincera gratitud a la Universidad Técnica del Norte y a los distinguidos docentes de la Carrera de Electricidad. Sus enseñanzas me han dado las herramientas necesarias para lograr mis objetivos profesionales.

Agradezco al director de este Trabajo de Grado, el Ing. Francisco Naranjo, MSc. Su papel no solo como guía sino también como mentor ha sido fundamental en el desarrollo de este proyecto. Su búsqueda constante de excelencia a través de sus orientaciones ha sido un pilar clave en mi desarrollo académico y profesional.

A cada persona que fue parte del proceso, muchas gracias.

ÍNDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

RESUMEN

Este trabajo describe la creación de un sistema de medición de consumo eléctrico inalámbrico mediante Home Assistant. Se emplearon siete tecnologías de comunicación para evaluar la eficiencia en la transmisión de potencia. Además, se diseñaron dispositivos capaces de medir en tiempo real variables eléctricas como voltaje, corriente, frecuencia, y potencias activa, reactiva, y aparente, así como el factor de potencia. Estos dispositivos también pueden controlar cargas y forman una red de sensores que miden temperatura, humedad, luz ambiental, y niveles de humo, gas, y ruido. A través de la red local de Home Assistant, se pueden visualizar y controlar las mediciones y cargas conectadas.

Los componentes seleccionados cumplen con los estándares eléctricos para asegurar la compatibilidad con los electrodomésticos, soportando hasta 15 A a 125VAC 60 Hz. La elección de tecnologías de comunicación se basó en las normativas ecuatorianas de frecuencias.

Los resultados mostraron un error de medición de energía de solo 0.12% con un factor de potencia de 1, aumentando a 4.51% con un factor de 0.5. Entre las tecnologías de comunicación, LoRa destacó por su óptima relación calidad-precio, logrando una cobertura de hasta 100 metros en interiores, superando a Zigbee por 40 metros.

Mediante Home Assistant, se desarrollaron automatizaciones que incluyen alarmas para situaciones como la desconexión de un nodo, y la detección de humo o gas. También se estableció el control automático de las cargas basado en diferentes intervalos del día, destinado a minimizar el consumo energético en modo stand-by.

Palabras clave: Medidores eléctricos, Tecnologías de Comunicación Inalámbrica , Home Assistant, Red de Sensores, Automatización.

ABSTRACT

This work describes the development of a wireless electrical consumption measurement system using Home Assistant. Seven communication technologies were employed to assess power transmission efficiency. In addition, devices capable of real-time measurement of electrical variables such as voltage, current, frequency, and active, reactive, and apparent powers, as well as power factor, were designed. These devices can also control loads and form a network of sensors measuring temperature, humidity, ambient light, and levels of smoke, gas, and noise. Through Home Assistant's local network, it is possible to visualize and control the measurements and connected loads.

The selected components comply with electrical standards to ensure compatibility with household appliances, supporting up to 15 A at 125VAC 60 Hz. The choice of communication technologies was based on Ecuadorian frequency regulations.

The results showed an energy measurement error of only 0.12% with a power factor of 1, increasing to 4.51% with a power factor of 0.5. Among the communication technologies, LoRa stood out for its optimal cost-quality ratio, achieving coverage of up to 100 meters indoors, surpassing Zigbee by 40 meters.

Using Home Assistant, automations were developed that include alarms for situations such as disconnection of a node, and detection of smoke or gas. Automatic control of loads based on different times of the day was also established, aimed at minimizing energy consumption in stand-by mode.

Keywords: Electric Meters, Wireless Communication Technologies, Home Assistant, Sensor Network, Automation.

CAPÍTULO I
INTRODUCCIÓN

Tema

Implementación de un Sistema Inteligente de Medición de Consumo Eléctrico Residencial mediante Home Assistant.

1.1 Problemática a investigar

El avance tecnológico ha implicado que el uso de ciertos dispositivos electrónicos sea indispensable en el diario vivir de la población en general, como consecuencia, estos son usados de forma inconsciente a lo que el ahorro energético se refiere, trayendo consigo consumos altos, inclusive cuando éstos se mantienen en un estado de suspensión o reposo.

Los medidores de energía eléctrica tradicionales están diseñados con el propósito de registrar el consumo general del circuito total de una residencial, que posteriormente es registrado por la empresa distribuidora de energía eléctrica de la localidad. Este sistema no es amigable para el consumidor, puesto que no entrega información relevante como el consumo individual de un electrodoméstico, su curva de demanda u otras estadísticas que podrían ser concientizadoras del modo de uso de la energía en una residencia.

Los medidores son dispositivos sencillos que sólo registran la variable necesaria para el cobro de electricidad, no están pensados en optimización o ahorro energético, por lo tanto, no consideran fenómenos comunes en una red residencial como el consumo en Standby, el cual es la cantidad de energía que un dispositivo consume mientras está en modo de espera, es decir, cuando está conectado a la fuente de alimentación, pero no está en uso activo. Este consumo puede representar una cantidad significativa de energía.

El incremento de la demanda eléctrica en Ecuador ha llevado a un aumento en la construcción de centrales hidroeléctricas. Sin embargo, es importante tener en cuenta que la

construcción de estas centrales hidroeléctricas puede tener un impacto ambiental significativo, especialmente en áreas donde se construyen grandes presas. La alteración del ecosistema fluvial y la pérdida de hábitat pueden tener consecuencias a largo plazo para la biodiversidad y la calidad del agua.

¿Cómo implementar un sistema inteligente de medición del consumo eléctrico residencial utilizando Home Assistant?

1.2 Objetivos

Se establecieron objetivos generales y específicos a cumplir durante la realización del trabajo de grado. Cada objetivo específico corresponde a cierto fragmento o capítulo del documento presente.

Objetivo General

Implementar un sistema inteligente de medición de consumo eléctrico residencial que permita el control y automatización de las cargas, mediante comunicación inalámbrica y Home Assistant.

Objetivos Específicos

1. Describir los tipos de medidores y protocolos de comunicación inalámbrica.
2. Diseñar un Sistema Inteligente de Medición de Consumo Eléctrico.
3. Implementar un Sistema Inteligente de Medición de Consumo Eléctrico en una red residencial.

1.3 Alcance y delimitación

Este proyecto consiste en la construcción de un entorno inalámbrico de recopilación de datos de consumo eléctrico residencial, empleando dispositivos ESP32 y Raspberry PI mediante protocolos de comunicación como Xbee, Z-wave, WiFi, NRF42L, LoRa, BLE, GSM o LTE, incluyendo la integración a dispositivos del mercado convencionales como Amazon Alexa y Google Assistant.

Se utilizará Home Assistant para acceder a los datos obtenidos por medio de los dispositivos inalámbricos individuales y revisar la información adquirida a través de smartphones de sistemas operativos como Android o iOS. Además, proporcionará la posibilidad de controlar y automatizar los electrodomésticos conectados.

El Sistema Inteligente de Medición de Consumo Eléctrico contará con 7 medidores independientes que a la vez funcionarán de actuadores, éstos se conectarán a los tomacorrientes del circuito de fuerza de la red residencial. Cada uno de los 7 dispositivos independientes se comunicarán inalámbricamente con el controlador principal por medio de cada protocolo de comunicación. El controlador principal recibirá los datos recopilados de los dispositivos independientes y será quien procese la información y emita las órdenes para el control de las cargas. El smartphone se conectará con el controlador principal para visualizar los datos y controlar las cargas manualmente en caso de que se requiriera.

Se considerará un sistema inteligente debido que el controlador ejecutará un algoritmo que comparará los valores medidos en un solo dispositivo, para así, detectar cuando éste se encuentre consumiendo electricidad en un estado de Standby o reposo y decidir si es factible o no desconectarlo de la red, por lo tanto, será capaz de detectar otros estados del dispositivo en base a su consumo.

El Sistema Inteligente de Medición de Consumo Eléctrico recopilará variables eléctricas como voltaje, corriente, potencia y factor de potencia. Funcionará en redes monofásicas de 120V a 60 Hz. Los medidores/actuadores independientes soportarán una corriente de funcionamiento nominal de hasta 15 A.

1.4 Justificación

Según Manzano (2022) el sector eléctrico ecuatoriano enfrenta importantes desafíos para desarrollar su potencial energético, tales como las elevadas inversiones requeridas y los sistemas limitados para estimar la disponibilidad de los recursos de manera eficiente. Esto conlleva a que no estemos preparados para la construcción de más centrales hidroeléctricas en caso de requerir incrementar la capacidad del sistema eléctrico del país. Además, Moreno (2022) afirma que las centrales hidroeléctricas, a pesar de anteriormente haberse considerado como fuentes de energía limpia, tienen un fuerte impacto ambiental en la salud del agua, del suelo, del aire y en la fauna y flora de los ecosistemas, incluyendo a las poblaciones aledañas.

En cuanto al consumo eléctrico de nuestros dispositivos del hogar, hay que considerar que existen consumos de los que el cliente no tiene la responsabilidad de forma directa, si no, de la poco eficientes que son algunos dispositivos. Meier (2018) afirma que, a nivel mundial, el consumo de electricidad en Standby por dispositivo es pequeño, a menudo de menos 1 W, pero miles de millones de dispositivos consumen energía en modo de espera.

Otra razón por la cual es importante una solución para monitorear el consumo eléctrico residencial es que puede ayudar a los usuarios a identificar el estado de trabajo del dispositivo y como también posibles problemas de funcionamiento. Por ejemplo, si la aplicación muestra un aumento inesperado en el consumo de energía, esto podría indicar que hay un problema con un electrodoméstico o con el sistema eléctrico en general.

Este proyecto beneficia a los consumidores de energía eléctrica, puesto que evitaría que se haga uso de la electricidad de forma descontrolada, sin ser conscientes del consumo provocado por sus electrodomésticos de uso diario. El ahorro energético satisface económicamente al consumidor del sistema eléctrico.

CAPÍTULO II
MARCO TEÓRICO

En los últimos años, el avance tecnológico ha revolucionado diversos aspectos de nuestra vida cotidiana, y el ámbito de la energía eléctrica no ha sido la excepción. En este contexto, los medidores eléctricos domiciliarios y la domótica han emergido como dos áreas clave para mejorar el consumo energético en el hogar. Estas innovaciones tecnológicas permiten un control más preciso y personalizado del consumo eléctrico, así como la automatización de diversos dispositivos y sistemas dentro de nuestras viviendas.

Las herramientas ofrecidas por softwares de código abierto como HomeAssistant, facilitan el diseño de sistemas domóticos inalámbricos capaces de adaptarse a dispositivos de uso cotidiano, por lo que es necesario explicar algunos conceptos necesarios para la implementación de un proyecto de este tipo.

2.1. Medidores eléctricos

En un sistema eléctrico, tener a disposición todas las variables que intervienen en su funcionamiento es sustancial para la comercialización y el análisis de la calidad energética. Existen varios sectores de la industria eléctrica donde es necesario realizar mediciones, por ende, su diseño y modo de operación es distinto en base a la aplicación por la que fueron pensados. [1] mencionan que todas las actividades de diseño, operación y control de equipos en ingeniería se basan en la adecuada medición y registro de parámetros físicos, químicos, mecánicos, ópticos y de muchos otros tipos.

2.1.1. Medidores estáticos

También conocidos como de estado sólido, son medidores embebidos en el resto del circuito, por ende, pueden considerarse del tipo “intrusivo”. [2] define un medidor estático como “contador en el que la corriente y el voltaje actúan sobre los elementos (electrónicos) de estado sólido para producir una salida proporcional a la energía a medirse”.

Los medidores estáticos utilizan multiplicadores analógicos con el fin de obtener la potencia instantánea en base a los valores instantáneos de tensión y corriente. En el caso de los que aplican señales digitales, éstos utilizan técnicas de procesamiento como circuitos integrados del tipo ADC. [3] afirman que una configuración común en este tipo de medidores para una instalación monofásica está respaldada por dos ADC sincronizados, que convierten en paralelo las señales de voltaje y corriente.

2.1.2. Medidores de inducción

Un medidor de inducción utiliza principios de electromagnetismo para registrar la potencia en base al movimiento de un disco provocado por dos electroimanes, que, al energizarse, provocan el giro del disco cuya velocidad es proporcional a la potencia activa instantánea que alimenta el electroimán. Estos medidores poseen un mecanismo que registra el número de vueltas con relación al tiempo, obteniendo así la energía consumida.

Este principio fue tradicionalmente utilizado en los medidores proporcionados por las empresas de distribución eléctrica. Sin embargo, posee algunos inconvenientes que los hacen susceptibles a manipulaciones ocasionadas por armónicos. [3] afirman que los armónicos del tipo 2 y 4 provocan fuerzas adicionales (momentos) que aceleran el medidor, mientras que los armónicos del tipo 3 y 5 provocan leves frenados.

2.1.3. Medidores inteligentes

Un medidor inteligente, además de medir todos los parámetros eléctricos posibles, utiliza redes de comunicación para almacenar y gestionar los datos recopilados en su funcionamiento. Algunos de estos medidores buscan adaptarse e intervenir en la actividad del sistema eléctrico.

La comunicación bidireccional alude a la comunicación entre los proveedores de electricidad y los clientes con la ayuda de comunicaciones por cable o inalámbricas. Es

una de las características vitales que separa a los medidores inteligentes de los medidores convencionales [4].

2.2. Home Assistant

Home Assistant es una herramienta de código abierto para la gestión de dispositivos de domótica desarrollada en Python. Está orientado al hogar, y es capaz de adaptarse a una gran cantidad de dispositivos y utilidades de varios desarrolladores.

La compatibilidad con más de 1400 dispositivos significa que probablemente casi todos los tipos de dispositivos Smart Home son integrables, como Hue, Lifx, Google Home, Alexa, Ecobee, Z-Wave, WeMo, IKEA Trådfri y tantos dispositivos DIY que usan aplicaciones de Python o MQTT [5].

Este tipo de plataformas son el enlace que existe entre los distintos actuadores y sensores con el usuario dentro de un sistema domótico. Autores han mencionado Home Assistant como una opción fiable en la implementación de redes de comunicación:

Cualquier dispositivo IoT está compuesto por uno o más sensores con posibilidad de conexión a la red a través de cable o Wi-Fi. Tales dispositivos electrónicos inteligentes pueden comunicarse entre sí usando complementos de Home Assistant [6].

Hoy en día, Home Assistant cuenta con 2490 complementos disponibles en la sección de Integraciones en su página web. Estos pueden ser dedicados a facilitar o expandir las posibilidades en aplicaciones de automatización, impresión 3D, geolocalización, salud, etc. Además, [7] afirma que la plataforma Home Assistant está respaldada por una gran cantidad de usuarios, por lo que se publican frecuentemente actualizaciones que aumenta el número de complementos compatibles.

2.3. Consumo en Standby

Según [8], el consumo en standby es la electricidad consumida por los aparatos y dispositivos mientras esperan realizar sus funciones principales. Esta categoría de consumo

ocurre en casi todos los productos electrónicos de consumo y en otros dispositivos equipados con pantallas digitales, controles remotos y conexiones de red. Incluso cuando estos dispositivos no se usan activamente, aún usan energía para permanecer en modo de espera, lo que puede representar una parte significativa de su consumo total de energía.

Esta problemática ya ha sido estudiada y analizada, por lo cual se han realizado diálogos e iniciativas donde los fabricantes han considerado reducirlo con distintas técnicas, sin embargo, los equipos electrónicos siguen evolucionando, y tecnologías modernas como las redes de comunicación han provocado que existan más desencadenantes de consumo en standby en un electrodoméstico común. El coste energético de mantener continuamente una conexión de red puede superar los 2,5 W [8]. Como consecuencia, el decrecimiento del consumo en standby ha sido compensado con el incremento del uso de dispositivos electrónicos en una residencia.

Según investigaciones realizadas, el consumo de energía en espera representa el 1 al 2 % del uso de electricidad global y al menos el 10 % del uso de electricidad residencial en los países desarrollados [8].

2.4. Redes en Malla

Las Redes en Malla o Mesh se definen como un tipo de red compuesta por un Router también llamada estación base y los puntos de acceso, los cuales se encargan de comunicarse entre sí, de esa forma el usuario puede trasladarse por todos los lugares hasta donde llegue el rango de cobertura de la red con la característica principal que estarán conectados siempre a una única red Wi-fi con el mismo ID y contraseña [9].

Los clientes pueden ser estáticos o móviles. Tienen las funciones necesarias para trabajar en una red mesh y pueden trabajar como enrutadores, aunque sin las funciones de puente y puerta de enlace. Disponen de una única interface de red, por lo que el número de

dispositivos que puede actuar de cliente en una red mesh es mayor que el de dispositivos que pueden actuar de routers mesh [10].

Según [11], un dispositivo fuera del rango de cobertura es capaz de unirse a algún nodo móvil dentro del rango de la red de malla y usarlo como punto de acceso. Ésta es una de las principales ventajas del uso de redes en malla en un sistema inalámbrico.

2.5. Machine to Machine

Machine to Machine (M2M) o Máquina a Máquina es, según [12], el intercambio de información entre dos máquinas de forma remota ya sea de forma cableada o inalámbrica y es la configuración de red mayormente utilizada en sistemas IoT. Algunos autores destacan las ventajas y las aplicaciones comunes con las redes M2M:

> M2M se utiliza ampliamente en energía, transporte, control mecánico, comercio, salud, agua, seguridad y en diferentes empresas. M2M es fundamental por ser sencillo, barato y listo para trabajar sin mantenimiento durante períodos de tiempo prolongados y para impartir sobre el territorio remoto [13].

Según [14], Una red M2M consta de 3 dominios principales, el dominio del dispositivo, la puerta de enlace, el dominio de la red y el dominio de la aplicación, donde el dominio del dispositivo se encarga de proporcionar conectividad entre la puerta de enlace M2M y los dispositivos. La puerta de enlace detecta y transmite los datos de los dispositivos M2M a la red de comunicación de forma inteligente y eficiente en patrones de uno o múltiples saltos. El dominio de la red abarca la comunicación entre la aplicación y la puerta de enlace, y la aplicación finalmente, contiene la capa de middleware por donde pasan los datos.

2.6. Tecnologías de Comunicación Inalámbrica

Los protocolos de comunicación son conjuntos de reglas y estándares que definen cómo se establece, mantiene y finaliza la comunicación entre sistemas de computadoras. Estos

protocolos son fundamentales para permitir que diferentes dispositivos y sistemas se comuniquen de manera efectiva y confiable.

2.6.1. Wi-Fi

Wi-Fi es actualmente el protocolo de comunicación más utilizado por fabricantes y usuarios, su señal inalámbrica de corto alcance está basada en los estándares IEEE 802.11 propuestos en 1997. En 2022, [15] estimaba que habría casi 18 mil millones de dispositivos Wi-Fi en uso, y más de 4,4 mil millones serían lanzados ese mismo año.

En [16] mencionan que, con el tiempo, el estándar 802.11 ha evolucionado y mejorado sus parámetros técnicos, como el ancho de banda y las tasas de datos y la banda de frecuencia de uso. La primera versión del estándar, denominada 802.11a, utilizaba modulación OFDM en la banda de 5 GHz y una tasa de datos máxima de 54 Mbps.

La versión 802.11b tiene especificaciones similares, con una tasa de datos más baja, pero un mayor alcance de señal debido que la modulación DSSS ocasionaba interferencias con otras ondas de frecuencia. La versión 802.11n fue un gran salto tecnológico debido que utilizaba la tecnología MIMO 3x3, capaz de transmitir y recibir simultáneamente múltiples frecuencias, logrando una tasa de transferencia de datos de 600 Mbps [16].

La versión más utilizada de los dispositivos inalámbricos actuales es 802.11ac. Esta versión ya es capaz de velocidades de datos en gigabits y solo funciona en 5GHz.

En cuanto a seguridad, Wi-Fi ha realizado avances a medida que se lanzaban nuevas versiones del protocolo. [17] menciona que con el nuevo estándar de seguridad para Wi-Fi, Wireless Protected Acces 3 (WPA 3), las redes inalámbricas se volvieron más seguras. Ataques pasivos como el “eavesdropping” (Técnica de hacking que consistía en capturar toda la información enviada a la red) ya no son posibles en las nuevas versiones.

2.6.2. Bluetooth Low Energy

BLE (Bluetooth Low Energy) es una variación del protocolo Bluetooth BR/EDR diseñada para reducir el consumo energético manteniendo especificaciones técnicas similares a la versión estándar.

Según [22], los dispositivos BLE pueden clasificarse en dos tipos, modo único y dual. El modo dual tiene soporte para Bluetooth Classic y BLE, por lo tanto, es capaz de conectarse a dispositivos de ambos tipos, mientras que el modo único solo puede conectarse a BLE. Además, menciona que al igual que su versión anterior, el diseño de BLE se compone de tres bloques: Aplicación, Hospedador y Controlador, y forman una arquitectura en capas.

Tabla 1

Comparación entre Bluetooth Low Energy y Bluetooth Classic.

Especificaciones	**Bluetooth Low Energy**	**Bluetooth Classic**
Rango	Mayor a 100 m	100 m
Velocidad de datos	125 kbitps – 1 Mbps – 2 Mbps	1-3 Mbps
Rendimiento de la aplicación	0.7-2.1 Mbps	0.27 Mbps
Esclavos activos	7	No definido
Frecuencia	2.4 GHz	2.4 GHz
Seguridad	AES con Modo Contador CBC-MAC de 128-bit	56/128-bit
Robustez	CRC de 24-bit, Revisión de Integridad del Mensaje de 32-bit	Salto de Frecuencia Rápida Adaptativa. FEC, Fast ACK
Latencia	6 ms	100 ms
Tiempo de retraso	3 ms	100 ms

Capacidad de voz	No	Yes
Topología de la red	Estrella	Estrella
Consumo energético	0.01 W – 0.50 W	1 W
Consumo de corriente pico	Menor a 15 mA	Menor a 30 mA

Nota. Adaptado de A Survey on Internet of Things for Smart Health Technologies [18].

2.6.3. GSM

Global System for Mobile Communication, abreviado como GSM, es una red celular que opera en rangos de frecuencia de 850 MHz, 900 MHz, 1800 MHz y 1900 MHz para transmitir datos. Es ampliamente utilizado por clientes de telefonía móvil de todo el mundo. Ésta viene siendo la segunda generación (2G) de redes celulares existentes. Las redes celulares sufrieron varios cambios evolutivos en un lapso relativamente corto de tiempo, siendo 4G LTE (Long-Term Evolution) una actualización mucho más moderna y completa a su predecesora, 3G y ante predecesora GSM.

[19] menciona que LTE implementa un sistema de red basado en Internet Protocol (IP) de extremo a extremo, mejorando la calidad de servicio (QoS), las tasas de transmisión de datos y reduciendo el coste de los recursos a comparación de 3G. Otros autores destacan lo siguiente:

> LTE utiliza multiplexación por división de frecuencia ortogonal (OFDM) y sistemas MIMO, que permiten múltiples antenas en los receptores y transmisores, para permitir un mayor ancho de banda de datos, aumentar el rendimiento y la resistencia a interferencias y reflejos [20].

2.6.4. Zigbee

ZigBee se basa en el estándar IEEE 802.15.4 y es uno de los principales protocolos de comunicación utilizados en sistemas de automatización residencial. [21] mencionan que, ZigBee se desarrolló con el objetivo de implementarse en aplicaciones que requieren comunicaciones confiables y seguras con una baja tasa transporte de datos, por lo cual es una

opción ideal para redes en malla y de sensores que necesiten admitir un gran número de dispositivos.

En un sistema comunicado por ZigBee, este define la aplicación, seguridad y las capas de red, mientras que el estándar IEEE 8020.15.4 establece las capas físicas y el MAC.

> La capa física es la más cercana al hardware y mediante la cual, se controla la comunicación con el transceptor radio. Permite, por lo tanto, la administración del acceso al hardware ZigBee, incluyendo la inicialización de este o la selección del canal según la previa estimación de la calidad de señal en los canales disponibles. Puede trabajar en tres bandas de frecuencias distintas: 868 MHz (para Europa), 915 MHz (en Norteamérica y Australia) y 2.4 GHz (disponible en todo el mundo) [22].

[21] mencionan que, ZigBee puede conectarse a 2 tipos de dispositivos con funcionalidades y características distintas entre sí. Los dispositivos de función completa pueden actuar como enrutadores en la red y, además, funcionar como coordinador de la red de área personal y también pueden operar con funciones reducidas, siendo capaces de conectarse a la red Zigbee a su enrutador padre como nodos finales.

2.6.5. Z-Wave

Z-wave es, al igual que Zigbee, un protocolo de comunicación dedicado a la automatización del hogar [23] menciona que Z-wave utiliza ondas de baja energía para comunicarse con otros dispositivos tanto local como fuera de la red.

> Se trata de un sistema fiable de comunicación basado en una red en malla que utiliza ondas de radio de baja energía para comunicarse de un aparato a otro con un alcance medio de comunicación entre dos nodos es de aproximadamente 30 metros en interiores y 100 metros en exteriores [24].

El protocolo Z-wave es utilizado en una gran gama de productos distribuidos por los mismos desarrolladores del protocolo, pero también es posible aprovecharse en aplicaciones independientes. [24] menciona que una de las principales ventajas del protocolo es que trabaja en una frecuencia de 900 MHz, distinta a protocolos populares como WiFi y Bluetooth, lo que evita interferencias.

Tabla 2

Comparación entre Z-Wave y ZigBee.

Especificaciones	**ZigBee**	**Z-Wave**
Rango	10-20 m	30-65 m
Velocidad de datos	250kb/s	40kb/s
Frecuencia	2.4 GHz	908 MHz
Consumo de corriente pico	40 mA	2.5 mA
Fabricante	Múltiples fabricantes	Exclusivamente comercializado por Silicon Labs
Proceso de certificación	Proceso variable	Proceso estricto
Nodos	Soporta hasta 65,000 nodos finales	Soporta hasta 232 nodos finales
Dificultad de Configuración	Más dificultad	Más amigable al usuario
Costo	Mejorar calidad-precio	Más costoso que ZigBee

Nota. Adaptado de ZigBee vs. Z-Wave: What's the Difference? [25].

2.6.6. LoRa

LoRa, proveniente de "Long-Range", es un protocolo de comunicación capaz de habilitar transmisiones de largo alcance de hasta 10 km con una eficiencia mayor a la de cualquier otro protocolo de comunicación estandarizado. [26] afirma que, una sola puerta de enlace o estación base puede cubrir una ciudad y países enteros con una infraestructura mínima.

Según expertos en la materia de telecomunicaciones, LoRa es una tecnología ideal para soportar la gran ola de dispositivos y aplicaciones IoT que emergerán en un futuro cercano. En 2021, [27], en una comparación de LoRaWaN con otros protocolos de comunicación se mencionó lo siguiente:

> Aunque algunas soluciones tradicionales como Bluetooth, Wi-Fi, ZigBee, WLAN, Z-wave, y redes celulares como GSM y LTE pueden proporcionar una conexión inalámbrica de los dispositivos IoT en una red, estas soluciones demandan alto costo, alto consumo de energía y su implementación es de alta complejidad.

2.6.7. nRF24

NRF24 es un módulo transceptor de RF de 2,4 GHz, lo que significa que cada módulo puede enviar y recibir datos. Este transceptor de un solo chip tiene un motor de protocolo de banda base integrado para aplicaciones inalámbricas de potencia ultra baja y funciona a 3,3 V. [28] menciona que un solo módulo puede comunicarse hasta con seis módulos más simultáneamente.

NRF24 opera a través del motor de protocolo de banda base embebido, o más conocido como Enhanced ShockBurst, un protocolo de comunicación inalámbrico de corto alcance desarrollado exclusivamente para los chips. Según [29], ESB es capaz de operar en un modo de consumo de ultra baja potencia, lo que lo convierte en una alternativa a considerar en implementaciones de comunicación inalámbrica con microcontroladores.

> La estructura original de ShockBurst constaba de los campos Preámbulo, Dirección, Carga útil y Comprobación de redundancia cíclica (CRC). ShockBurst mejorado trajo una mayor funcionalidad para comunicaciones mejoradas permitiendo cargas útiles de longitud variable con un especificador de longitud de carga útil, lo que significa que las cargas útiles pueden variar de 1 a 32 bytes [30].

2.7. Comunicación Serial

La comunicación serial es una interfaz de comunicaciones de datos digitales, frecuentemente utilizada por computadores y periféricos, donde la información es transmitida bit a bit enviando un solo bit a la vez [31].

Según [37], un método sencillo para comunicar un CPU con otro sistema electrónico externo es mediante una interfaz serial, debido a que son sencillos de implementar.

2.7.1. Interfaz SPI

La Interfaz Periférica Serial es una de las interfaces más utilizadas para comunicar microcontroladores con circuitos integrados periféricos, como sensores, ADC, DAC, registros de desplazamiento, SRAM, etc.

La Interfaz Periférica Serial es una interfaz síncrona basada en un subnodo principal de dúplex completo, esto quiere decir que los datos del nodo principal se sincronizan en el flanco ascendente o descendente del reloj [32].

Como se aprecia en la Figura 6, para la comunicación se utiliza una señal SCLK necesaria para la sincronización por medio de bits enviados en cada pulso de reloj. Las señales MOSI (Master Output Slave Input) y MISO (Master Input Slave Output) se refieren al esclavo y al maestro actuando como entrada y/o salida. La señal SS o Select alterna el subnodo en caso de existir varios.

2.7.2. UART

El estándar UART se basa en una comunicación serial que permite usar un transmisor/ receptor estándar serial universal asíncrono que esta embebido en la mayoría de los microcontroladores modernos [33].

Algunos autores definen el estándar UART de la siguiente manera:

Es un periférico de propósito general capaz de entablar comunicación serie a velocidades programables. Por lo general, este periférico se utiliza para fines de comunicación y depuración entre el circuito integrado y una computadora disponible para el usuario final [34].

Según [33], UART puede ser del tipo simplex, full-duplex o half-duplex, dependiendo del hardware implementado en el módulo. Por ejemplo, el UART de PicoSoC es principalmente usado para aplicaciones de impresión, debido a que envía bytes del tipo char a una computadora para ser visualizados en una terminal mediante full-duplex.

2.7.3. I2C

I2C (Inter-Integrated Circuit Bus), es un bus de comunicación serial sincrónica desarrollado por Phillips Semiconductores a principios de los años 80's, con la principal intención de interconectar una cierta cantidad de dispositivos en sus equipos de radio y TV [35].

[35] destaca que el protocolo de comunicación I2C es una interface serial muy útil para comunicación con otros periféricos y dispositivos microcontroladores, y puede operar tanto en la modalidad de esclavo, máster y de máster/eslavo.

Según [36], I2C permite el intercambio de información entre muchos dispositivos a una velocidad aceptable, de unos 100 Kbits por segundo, aunque hay casos especiales en los que el reloj llega hasta los 3,4 MHz.

Se utilizan dos líneas de comunicaciones, una para los datos llamada SDA y otra para el reloj (SCL). Cada dispositivo que se conecta al bus es direccionable por software, a través de una única e irrepetible dirección dentro del bus. La misma es determinada a través de una combinación de Hardware/Software, donde el fabricante define los bits más significativos. Los bits de menor peso son alambrados en la placa que soporta al componente [35].

2.8. Raspberry Pi

Raspberry Pi es un computador monoplaca que trabaja con Raspbian Pi OS, su sistema operativo basado en Linux. Según la hoja de datos técnicos oficiales publicada por [37], los componentes que conforman esta placa varían dependiendo la versión, en el caso del procesador, todos usan un multinúcleo de la marca Broadcom de distintos modelos. La RAM del Raspberry Pi 4 Model B puede ser de 1, 2, 4 y hasta 8 GB LPDDR4 si se desea.

En conectividad, el computador monoplaca Raspberry Pi 4 Model B cuenta con módulos dedicados IEEE 802.11 b/g/n (WiFi), Bluetooth y BLE para comunicación inalámbrica, un puerto LAN, 2 puertos USB 2.0 y 2 USB 3.0 para comunicación alámbrica.

Además, posee características adicionales que aumenta las posibilidades de aplicación de la Raspberry Pi, como varios puertos de audio y video, entrada para expandir el almacenamiento mediante una tarjeta SD y 40 pines GPIO que es posible programar en base a las necesidades de la aplicación.

2.9. ESP32

ESP32 es un potente microcontrolador que, a pesar de su tamaño, cuenta con características especiales de conectividad siendo una opción viable para aplicaciones de IoT a pesar de su accesible costo.

> Según [38], el chip que posee el ESP32 está diseñado para ser escalable y adaptable, puesto que los dos núcleos del CPU pueden manipular su frecuencia de reloj en un rango de 80 MHz a 240 MHz de forma individual, además son capaces de entrar en modo de bajo consumo usando el coprocesador, en espera de pulsos o cruces de umbral.

Independientemente del modelo de ESP32, todos suelen contener un chip integrado, un cristal de cuarzo de 40MHz y una memoria flash. Además, en el PCB se encuentran integrados los botones de boot y reset, el puerto de alimentación USB y los pines soldados [46].

2.10. Sensores

Los sensores son dispositivos que convierten una señal física a otra de salida proporcional a la entrada. Los sensores comúnmente usados miden variables como temperatura, distancia, luminosidad, proximidad, humedad, etc.

Los sensores miden cierta muestra de energía, un indicador o detector, en definitiva, la energía detectada es convertida en señales eléctricas que son recibidas por los dispositivos de vigilancia. Esta adquisición de datos es llevada por los operadores lógicos o en otras ocasiones, analizada por un individuo [39].

2.11. Actuadores

Los actuadores son dispositivos capaces de transformar energía hidráulica, neumática o eléctrica en la activación de un proceso con el fin de generar un efecto dentro de un sistema [40]. Según [31], un actuador puede clasificarse por el tipo de energía utilizada, por lo que puede ser neumático, hidráulico o eléctrico, y por el tipo de movimiento generado, por lo que el actuador puede cambiar su estado lineal o rotatoriamente.

CAPÍTULO III
MATERIALES Y MÉTODOS

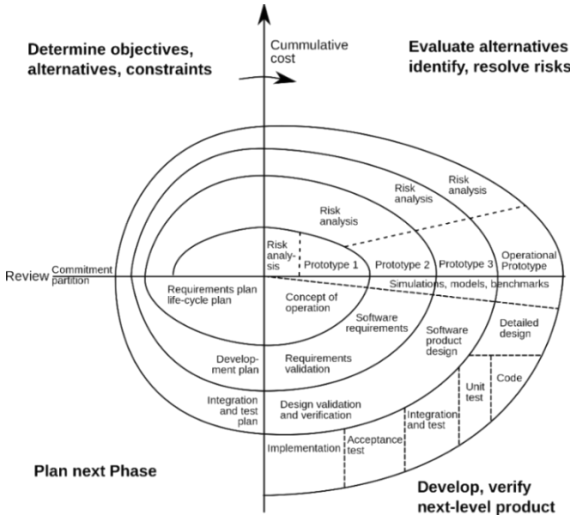
En este capítulo se detalla la metodología para la ejecución del proyecto. Se describen modelos matemáticos, componentes de software, hardware e instrumentos necesarios. Además, se justifica la elección de estos en función de sus características técnicas y grado de accesibilidad, proporcionando una base firme para una implementación adecuada.

3.1. Metodología

La metodología aplicada es la de Espiral Modificada, basada en el Espiral de Boehm, ya que, por su naturaleza, es un método eficiente para el desarrollo de proyectos de ingeniería que involucren el diseño de prototipos debido a su ciclo iterativo, que busca perfeccionar resultados y prevenir riesgos.

Figura 1

Método en Espiral de Boehm



Nota. Tomado de Software Engineering Economics [41].

Como se explica en la Figura 9, [42] propone cuatro etapas iterativas en su Método de Espiral: Planificación, Análisis de Riesgos, Ingeniería y Evaluación y Retroalimentación. Sin embargo, se simplificó a Análisis, Diseño, Implementación y Pruebas, manteniendo el ciclo iterativo, proceso investigativo que se basa en la experimentación. Esta readaptación se debe a que no existe riesgos de inversión económica en este proyecto, ya que no ha sido realizada con fines comerciales.

Figura 2

Método en Espiral Modificado

Nota. Método de Espiral adaptado a este proyecto.

El análisis ha sido realizado por medio del método documental, en el que se recopiló información sobre la estructura básica de un sistema domótico en el que está basado este proyecto, se consideraron normativas y regulaciones y, además, se realizaron entrevistas de consultoría al tutor del presente trabajo de grado, el Ing. Francisco Naranjo, como método de recolección de datos para el análisis de requerimientos y la planificación del diagrama de funcionamiento inicial.

Para el Diseño, se realizó la determinación de componentes mediante documentación otorgada por los fabricantes de los dispositivos utilizados, y se compararon sus características y aspectos técnicos entre sí. Además, se realizaron cálculos de grosor de trazo, cálculos de

protecciones y de fuente de alimentación. Luego, se realizaron los diagramas de flujo de los sistemas del proyecto para cada nodo, automatización y/o procedimiento.

En la Implementación, se aplicó el método experimental donde se aplicaron distintas formas de implementación del código de programación, verificando parámetros como los intervalos de actualización de las mediciones. Además, se imprimieron las placas de circuito impreso y los modelos 3D para las carcasas en base a cada esquemático.

Finalmente, en las Pruebas realizadas por el método experimental, se verificó la precisión de las mediciones eléctricas con otros instrumentos de medición en distintos tipos de carga, se hicieron pruebas de intensidad de señal en base a la distancia y se hizo un muestreo de las mediciones de la energía recolectada durante distintos intervalos de tiempo utilizando gráficos y tablas.

Figura 3

Diagrama de flujo para el análisis en la determinación de componentes.

Nota. Elaboración Propia.

3.2. Criterios de diseño eléctrico y de telecomunicaciones

Se utilizó normativa tanto nacional como internacional al considerarse distintos aspectos como el diseño de PCB, las bandas de frecuencia admitidas, criterios de calidad en medidores residenciales, etc.

3.2.1. Estándar NEMA 5-15P

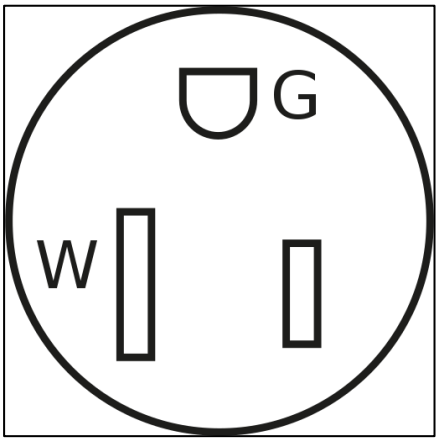
El Estándar NEMA 5-15P es el estándar eléctrico más común en los Estados Unidos y Canadá. Es un estándar de dos polos y tres cables con toma de tierra que se utiliza para un máximo de 15A a 125V [43] como se observa en la Figura 4.

Este Estándar está polarizado, lo que significa que mediante su diseño, asegura que la fase y el neutro se conecten de manera consistente para garantizar el correcto funcionamiento de los electrodomésticos. Este Estándar es el que se usa en gran parte de Latinoamérica en conectores monofásicos debido a la herencia eléctrica que existe en el continente y su uso garantiza la compatibilidad de equipos y dispositivos fabricados en esa región.

Además de sugerir un diseño mecánico resistente, NEMA 5-15P propone unos valores eléctricos de placa a los que los fabricantes de electrodomésticos en el continente se han reñido desde su aprobación.

Figura 4

Conector NEMA 5-15.



Nota. Tomado de *NEMA 5-15P* [44].

3.2.2. Regulaciones Nacionales e Internacionales de Radiofrecuencia

Es importante asegúrese de que las frecuencias utilizadas por las tecnologías de comunicación no convencionales estén permitidas y cumplan con las regulaciones locales. Por

lo cual, se consultó la documentación ofrecida por la Agencia de Regulación y Control de las Telecomunicaciones (ARCOTEL).

A continuación, se presenta una tabla donde se expone la tecnología de comunicación, la banda de frecuencia a la que opera y la regulación nacional que la aprueba.

Tabla 3

Tecnologías de comunicación con su regulación nacional.

Tecnología de Comunicación	Banda de Frecuencia (MHz)	Regulación	Descripción
Z-Wave	919.8 a 921.4	ARCOTEL Resolución 04-02-ARCOTEL-2021, Nota EQA.45	En las bandas 915 – 928 MHz, 2 400 – 2 483,5 MHz, 5 150 – 5 350 MHz, 5 470 – 5 725 MHz y 5 725 – 5 850 MHz y 24,05 – 24,25 GHz operan, a título secundario, sistemas que ocupan espectro radioeléctrico para Uso Determinado en Bandas Libres (UDBL), para los servicios fijo y móvil. [45]
LoRa	865 a 928		

Otras tecnologías implementadas en este proyecto, como Wi-Fi, BLE, GSM, nRF24L, Zigbee funcionan en bandas de frecuencia ISM, como 2.4 GHz, 5 GHz y 850 MHz para el caso de los servicios móviles.

Las bandas ISM (Industrial, Scientific and Medical) o bandas ICM (Industrial, Científico y Médico) son bandas para uso sin licencia del espectro radioeléctrico en aplicaciones de tipo industrial, científico y médico pero no para telecomunicaciones [46].

Las bandas libres reguladas por la Normativa Nacional son las mismas que forman parte de la ISM en la Región 2, la cuales contemplan a América. Estas son las frecuencias 902-928 MHz (frecuencia central de 915 MHz), 2400-2500 MHz (frecuencia central de 2450 MHz),

5725-5875 MHz (frecuencia central de 5800 MHz), y 24-24.25 GHz (frecuencia central de 24.125 GHz) [46].

3.2.3. Normativa de Control de Calidad de Medidores Residenciales

La Normativa de Fabricación y Ensayos NTE INEN-IEC62053-2 se basan en las normativas IEC 62052-11, IEC 62053, IEC 62056-21 y IEC 62056-1, los cuales se resumen en la siguiente tabla.

Tabla 4

Especificaciones Técnicas de Medidores Electrónicos Bifásicos.

Descripción	Especificación
Voltaje Nominal	2x120/240V
Rango de Voltaje de Funcionamiento Extendido	0,8 a 1,15 Vn
Corriente Nominal (Ib)	10A o menor
Corriente Máxima	100A
Frecuencia	60Hz
Clase de Exactitud	Clase 1 (error menor al 1%) y 2 (error menor al 2%)
Magnitudes a medir	Energía Activa Acumulada (kWh), Energía Reactiva Acumulada (kVAR), Voltajes, Corrientes, Demanda Máxima (kW) en periodos de 15 minutos (en bloque)
Visualizador o Registrador	LCD 60x20mm, mínimo 6 dígitos de 5mm de ancho y 10mm de alto.
Potencia Absorbida por cada elemento de voltaje a condiciones nominales	Máximo 1.0 W
Potencia Absorbida por cada elemento de corriente a condiciones nominales	Máximo 0,5 VA

Unidad de la constante del medidor (Imp/kWh)	1000 Imp/kWh

Nota. Adaptado de INEN [47].

Como podemos observar en la Tabla 5, se detalla que 2x120/240V es el Voltaje Nominal de estos medidores. Esto debido que los transformadores de distribución son del tipo monofásico de dos hilos, brindando a los consumidores la capacidad de utilizar dispositivos que funcionan con 240V.

Sin embargo, este proyecto fue diseñado con el propósito de que el dispositivo se conecte a los enchufes eléctricos de los domicilios, los cuales en su mayoría son del tipo NEMA 5-15P porque son monofásicos de 1 hilo y funcionan con 125V.

3.3. Criterios para la selección de componentes

Se plantearon criterios para la selección de los componentes electrónicos, en base a entrevistas de consultoría y a los estándares y regulaciones expuestos anteriormente.

3.3.1. Criterios para la selección del controlador principal

Los criterios tomados en consideración para la selección del controlador secundario fueron los siguientes.

- Disponibilidad en el país.
- Buena relación calidad/precio.
- Soporte continuo con Home Assistant
- Ser lo más pequeño posible para adecuarse a los estándares de diseño del mercado (asistentes del hogar inteligentes).
- Contar con el mayor soporte a tecnologías de comunicación (facilita la implementación de algunos puntos del proyecto).
- Buena cantidad de documentación y librerías disponibles en la web.

3.3.2. Criterios para la selección de los controladores secundarios

Los criterios tomados en consideración para la selección de los controladores secundarios fueron los siguientes.

- Disponibilidad en el mercado.
- Buena relación calidad/precio.
- Ser lo más pequeños posibles para adecuarse a los estándares de diseño del mercado (interruptores inteligentes).
- Contar con el mayor soporte a tecnologías de comunicación (optimiza el espacio en el diseño de las placas de circuito impreso).
- Contar con los pines de comunicación serial necesarios que permitan la inclusión de módulos adicionales de cualquier tipo.
- Accesibilidad a documentación y librería disponibles en la web.

3.3.3. Criterios para la selección del dispositivo de control de carga

Los criterios tomados en consideración para la selección del dispositivo de control de carga fueron los siguientes.

- Disponibilidad en el mercado.
- Valores nominales del relé igual o superior a 15A en 125V (siguiendo el estándar NEMA 5-15).
- Ser lo más pequeño posible para optimizar espacio en la PCB.
- Tener un voltaje de activación óptimo para operarse con un microcontrolador de forma sencilla.
- Tener implementado un sistema de optoacoplador para aislar el microcontrolador del bobinado y la carga conectada.
- En caso de activarse con 5V y el microcontrolador operar con 3.3V, tener implementado un conversor de nivel lógico.

3.3.4. Criterios para la selección de los módulos GSM

Los criterios tomados en consideración para la selección de los módulos GSM fueron los siguientes.

- Disponibilidad en el mercado.
- Buena relación calidad/precio.
- Ser lo más pequeño posible para adecuarse al diseño del resto de controladores secundarios.
- Ser capaz de conectarse al controlador principal y secundarios de forma práctica.
- Accesibilidad a documentación y librerías disponibles en la web.
- Funcionar en las bandas de frecuencia 700 MHz, 1900 MHz, 1700-2100 MHz y 2500 MHz disponibles en el país.

3.3.5. Criterios para la selección de los módulos LoRa

Los criterios tomados en consideración para la selección de los módulos LoRa son los siguientes.

- Disponibilidad en el mercado.
- Buena relación calidad/precio.
- Ser lo más pequeño posible para adecuarse al diseño del resto de controladores secundarios.
- Operar en las bandas de frecuencia 865 a 928 MHz disponibles en el país.
- Ser capaz de conectarse al controlador principal y secundarios de forma práctica.
- Tener una potencia de transmisión considerable.
- Accesibilidad a documentación y librerías disponibles en la web.

3.3.6. Criterios para la selección de los módulos Zigbee

Los criterios tomados en consideración para la selección de los módulos Zigbee fueron los siguientes.

- Disponibilidad en el mercado.
- Buena relación calidad/precio.
- Ser lo más pequeño posible para adecuarse al diseño del resto de controladores secundarios.
- Ser capaz de conectarse al controlador principal y secundarios de forma práctica.
- Tener una potencia de transmisión considerable.
- Accesibilidad a documentación y librerías disponibles en la web.

3.3.7. Criterios para la selección de los módulos Z-Wave

Los criterios tomados en consideración para la selección de los módulos Z-Wave fueron los siguientes.

- Disponibilidad en el mercado.
- Buena relación calidad/precio.
- Ser lo más pequeño posible para adecuarse al diseño del resto de controladores secundarios.
- Operar en las bandas de frecuencia 919.8 a 921.4 MHz.
- Ser capaz de conectarse al controlador principal y secundarios de forma práctica.
- Tener una potencia de transmisión considerable.
- Accesibilidad a documentación y librerías disponibles en la web.

3.3.8. Criterios para la selección de los módulos BLE

Los criterios tomados en consideración para la selección de los módulos BLE fueron los siguientes.

- Disponibilidad en el mercado.
- Buena relación calidad/precio.

- Ser lo más pequeño posible para adecuarse al diseño del resto de controladores secundarios.
- Ser capaz de conectarse al controlador principal y secundarios de forma práctica.
- Tener una potencia de transmisión considerable.
- Accesibilidad a documentación y librerías disponibles en la web.

3.3.9. Criterios para la selección del módulo Wi-Fi

Los criterios tomados en consideración para la selección del módulo Wi-Fi fueron los siguientes.

- Disponibilidad en el mercado.
- Buena relación calidad/precio.
- Ser lo más pequeño posible para adecuarse al diseño del resto de controladores secundarios.
- Ser capaz de conectarse al controlador principal y secundarios de forma práctica.
- Tener una potencia de transmisión considerable.
- Accesibilidad a documentación y librerías disponibles en la web.

3.3.10. Criterios para la selección del módulo nRF24L

Los criterios tomados en consideración para la selección del módulo nRF24L fueron los siguientes.

- Disponibilidad en el mercado.
- Buena relación calidad/precio.
- Ser lo más pequeño posible para adecuarse al diseño del resto de controladores secundarios.
- Ser capaz de conectarse al controlador principal y secundarios de forma práctica.
- Tener una potencia de transmisión considerable.
- Buena cantidad de documentación y librerías disponibles en la web.

3.3.11. Criterios para la selección del sensor de medición eléctrica

Los criterios tomados en consideración para la selección del sensor de medición eléctrica fueron los siguientes.

- Disponibilidad en el mercado.
- Buena relación calidad/precio.
- Ser lo más pequeño posible para adecuarse al diseño del resto de controladores secundarios.
- Ser capaz de conectarse al controlador principal y secundarios de forma práctica.
- Buena cantidad de documentación y librerías disponibles en la web.
- Medir la mayor cantidad de variables eléctricas posibles.

3.4. Análisis de los componentes considerados

Se plantearon componentes electrónicos potenciales para la implementación, contrastándolos con los criterios expuestos anteriormente.

3.4.1. Análisis de los controladores principales considerados

Home Assistant, en su documentación, establece ciertas características que los dispositivos donde se instalará el software deben cumplir, por lo tanto, también recomienda su instalación en ciertos dispositivos ya probados, expuestos en la siguiente tabla.

Tabla 5

Comparación entre dispositivos recomendados por Home Assistant.

Dispositivo (específico)	CPU	RAM (GB)	Tamaño (mm)	Comunicación Inalámbrica	Precio aproximado ($ USD)
Raspberry Pi	1.5 GHz	4 ”[a]	85 x 56 ”[a]	2. 4 GHz y 5. 0 GHz Wi-Fi	60 ”[e]

(4 model B)	4 núcleos ”a			Bluetooth 5.0 ”a	
ODROID (Hardkernel C4)	2 GHz 4 núcleos ”b	4 ”b	85 x 56 ”b	Infrarrojo ”b	70 ”e
ASUS Tinkerboard (Smartfly info Tinkerboard 2)	2 GHz 6 núcleos ”c	2 ”c	85 x 56 ”c	2. 4 GHz y 5. 0 GHz Wi-Fi Bluetooth 5.0 ”c	105 ”e
x86-64 Genérico (Lenovo ThinkCentre Premium M91P)	3.1 GHz 4 núcleos ”d	8 ”d	457.2 x 76 ”d	2. 4 GHz y 5. 0 GHz Wi-Fi ”d	80 ”e

Nota. Se tomó como referencia el dispositivo más económico para cada clase. ”a [43]. “b [48]. “c [49]. “d [50].

La placa de desarrollo Raspberry Pi 4 Model B destaca por su amplio soporte a tecnologías de comunicación y conectividad inalámbrica a pesar de su bajo precio y pequeño tamaño. Es el minicomputador más comercial del mercado, por lo que tiene una variada retroalimentación por usuarios que comparten soluciones en la web.

Según [51], las especificaciones técnicas recomendadas para Home Assistant son las siguientes.

- 2 GB RAM
- 32 o más GB de almacenamiento (idealmente SSDs de tipo A2)
- Fuente de Alimentación de 3.5 A

Hardkernel ODROID-C4 posee mejor procesador que el Raspberry Pi 4 model B, el cual podría ser una característica importante en algunos tipos de aplicaciones. Sin embargo, no cuenta con ninguna tecnología de comunicación inalámbrica útil en este proyecto.

Smartfly info Tinkerboard 2 es un dispositivo mucho más potente a un precio mayor. Cuenta con las características más modernas posibles en una placa de desarrollo de este tipo, sin embargo, su precio es muy distante al más económico de la Tabla 4.

El Lenovo ThinkCentre Premium M91P es un computador convencional. Tiene un buen procesador, relacionado directamente con su tamaño, el cual se aleja de los estándares de diseño que se buscan en esta implementación.

3.4.2. Análisis de los controladores secundarios considerados

Se buscó un microcontrolador de tipo placa de desarrollo para facilitar la implementación, se analizaron los fabricantes más conocidos del mercado y sus características más relevantes para el proyecto.

Tabla 6

Comparación entre los microcontroladores más populares.

Dispositivo (específico)	**CPU**	**SRAM (GB)**	**Tamaño (mm)**	**Comunicación Inalámbrica**	**Precio aproximado ($ USD)**
Espressif (ESP32-Devkit-C)	240 MHz 2 núcleos ”a	520 KB ”a	54.4 x 27.9 ”a	2. 4 GHz y 5. 0 GHz Wi-Fi Bluetooth 5.0 ”a	5 ”e
Arduino (Mini Nano V3)	16 MHz 1 núcleo ”b	2 KB ”b	45 x 18 ”b	Ninguna ”b	6 ”e

Raspberry Pi (Pico)	133 Hz 2 núcleos [c]	264 KB [c]	85 x 56 [c]	Ninguna [c]	12 [e]
STM32 (Shutao STM32F103C6T6)	72 MHz 1 núcleo [d]	20 KB [d]	93.2 x 66 [d]	Ninguna	3.5 [e]

Nota. Se tomó como referencia el dispositivo más económico para cada clase. [a] [45]. [b] [52].[c] [53]. [d] [54].

Los microcontroladores de Espressif son altamente usados en aplicaciones de IoT porque están diseñados para ofrecer conectividad inalámbrica integrada a bajo precio. Por otro lado, Arduino produce dispositivos de menor consumo energético, pero sacrificando funcionalidades.

Raspberry Pi Pico no cuenta con conectividad inalámbrica, sin embargo destaca por su amplia cantidad de puertos GPIO y de comunicación serial, además de tener soporte con MicroPython, la cual puede ser una opción factible para quienes lo prefieran.

Debido a su popularidad, se han realizado varias versiones alternas a los dispositivos lanzados por ESP32 donde se incluye soporte a tecnologías de comunicación adicionales, los cuales se presentan en la Tabla 7.

Tabla 7

Versiones alternativas de microcontroladores Espressif.

Dispositivo específico	CPU	SRAM (GB)	Tamaño (mm)	Comunicación Inalámbrica	Precio aproximado ($ USD)
Heltec ESP32 Lora-V3 (ESP32-S3)	240 MHz 2 núcleos ”a	512 KB ”a	50.2 x 25.5 x 10.2 mm ”a	2. 4 GHz y 5. 0 GHz Wi-Fi Bluetooth 5.0 / LoRa SX1262 ”a	17.89 ”e
ESP32-C6	160 MHz 1 núcleo ”b	512 KB ”b	51,8 x 20 x 3.1 mm”b	2. 4 GHz y 5. 0 GHz Wi-Fi Bluetooth 5.0 / Zigbee y Thread ”b	20 ”e

Nota. ”a [45]. “b [52]..

3.4.3. Análisis de los dispositivos de control considerados

Para el análisis y selección del dispositivo de control, se compararon distintos tipos de interruptores eléctricos destacando su corriente nominal, tamaño, precio y accesibilidad. Los modelos de relés de la siguiente tabla fueron elegidos en base a su disponibilidad en tiendas online.

Tabla 8

Comparación entre los relés candidatos

Modelo	Tipo de Relé	Voltaje de Activación	Corriente de Input (mA)	Carga Nominal	Tamaño (mm)
SRD-12VDC-SL-C ”a	Mecánico	12VDC	30	10A 125VAC 7A 250VAC 10A 28VDC	15.6 x 19.21 x 5.6 x 19.2

SRD-5VDC-SL-C [b]	Mecánico	5VDC	90	10A 125VAC 10A 250VAC 10A 28VDC	15.6 x 19.21 x 5.6 x 19.2
SRD-9VDC-SL-C [c]	Mecánico	9VDC	40	10A 125VAC 10A 240VAC 10A 28VDC	15.6 x 19.21 x 5.6 x 19.2
SLA-5VDC-SL-C [d]	Mecánico	5VDC	185	30A 250VAC 1HP 125VAC 30A 30VDC	31.8 x 19.8 x 27.4
G3MB-202P [e]	Estado Sólido	5VDC	20	0.1-2A 75-264VAC	33.5 x 25.4 x 13.5
JQC-3FF-S-Z [f]	Mecánico	5VDC	71.4	10A 250VAC 15A 125VAC 10A 28VDC	15.6 x 19.21 x 5.6 x 19.2
A5W-K 5VDC [g]	Estado Sólido	5VDC	0.01	0.5A 125VAC 1A 30VDC	9 x 14 x 5
MPA-S-112-C [e]	Mecánico	12VDC	-	10A 250VAC 15A 125VAC 7A 28VDC	18,7 x 15 x 15.2

Nota. El tamaño estándar es de 19.2 mm x 15.4 mm. . [a] [55]. [b] [56]. [c] [57]. [d] [58] . [e] [59]. [f] [60]. [g] [61] . [e] [62].

Según [51], el Estándar NEMA5-15 es el que define la forma y composición de los enchufes para dispositivos eléctricos en territorio americano y canadiense, sin embargo, es el que también utilizamos en Latinoamérica. Este conector está diseñado para soportar una corriente de 15 A a 125 V, valores a los que los fabricantes se rigen al diseñar electrodomésticos. Por ende, se tomó como referencia esta norma para definir la corriente nominal de los dispositivos de control que se diseñaron en este proyecto.

3.4.4. Análisis de los módulos GSM considerados

Según [63],se recomiendan estos dispositivos para integrar GSM a su sistema.

Tabla 9

Comparación entre dispositivos GSM recomendados por Home Assistant.

Dispositivo específico	**Tipo**	**Compatibilidad**	**Alimentación (V)**	**Precio aproximado ($ USD)**
SIM800C ”a	Módulo o Módem USB ”a	GSM/GPRS ”a	3.3-5 ”a	10 ”g
Huawei E3372-510 “b	Módem USB ”b	LTE/3G/2G ”b	5 ”b	30”g
Huawei E3531 “c	Módem USB ”c	3G/2G ”c	5 ”c	25 ”g
Huawei E3272 “d	Módem USB ”d	LTE/3G/2G ”d	5 ”d	40 ”g
ZTE K3565-Z “e	Módem USB ”e	3G/2G ”e	5 ”e	20 ”g
Lenovo F5521gw	Módulo mPCIe ”f	3G/2G/1X-EVDO ”f	5 ”f	30 ”g

Nota. Se tomó como referencia el dispositivo más económico para cada clase. ”a [64], “b [65], ”c [66], “d [67], “e [68].

Para los módulos de comunicación, los dispositivos de tipo Módem USB son prácticos al momento de conectarlos en dispositivos con entradas de este tipo, Home Assistant los detecta automáticamente en modo de Plug and Play. Sin embargo, para integrarlos a microcontroladores, el tipo de módulo con pines es mejor, ya que optimiza el espacio al

adherirse a una placa de circuito impreso. En algunas tecnologías, es necesario seleccionar un módulo distinto para el controlador principal y para el secundario.

Recibir actualizaciones de mediciones en los menores intervalos de tiempo posibles es ideal, sin embargo, no es rentable en redes móviles debido al costo del servicio. Por esto, es suficiente GSM para la implementación, puesto la velocidad de transferencia de datos no es un factor diferencial.

3.4.5. Análisis de los módulos LoRaWaN considerados

REYAX, una empresa especializada en la fabricación de módulos de comunicación ha desarrollado módulos compatibles con LoRaWAN, una tecnología de comunicación de baja potencia y largo alcance diseñada para aplicaciones IoT. Entre los transceptores más populares tenemos los siguientes:

Tabla 10

Comparación entre dispositivos LoRa de Reyax.

Dispositivo REYAX	Corriente de transmisión (mA)	Dimensiones (mm)	Potencia de Transmisión	Precio aproximado ($ USD)
RYLR689 ”[a]	107	67 x 13	22 dBm	5.5
RYLR998 ”[b]	140	32 x 14	22 dBm	14
RYLR993 ”[c]	140	13 x 13	22 dBm	11
RYLR890 ”[d]	29	13 x 11	15 dBm	9
RYLR896 ”[f]	15	42.4 x 0.72	15 dBm	19

Nota. Se tomó como referencia el dispositivo más económico para cada clase. ”[a][69], “[b] [70], ”[c] [69], “[d] [71].

Es importante destacar que de los módulos expuestos anteriormente en la Tabla 8, el único que cuenta con soporte con Helium es el RYLR993.

3.4.6. Análisis de los módulos Zigbee considerados

Según [72], se recomiendan estos dispositivos para integrar Zigbee a su sistema.

Tabla 11

Comparación entre dispositivos Zigbee para Home Assistant.

Dispositivo específico	Potencia de Transmisión	Dimensiones (mm)	Precio aproximado ($ USD)
Home Assistant SkyConnect ”[a]	+20 dBm ”[a]	40 x 17 x 2 mm	30 ”[g]
SONOFF Zigbee 3.0 USB Dongle ”[b]	+20 dBm ”[b]	111.2 x 10 x 2 mm	30 ”[g]
POPP ZB-STICK ”[c]	+13 dBm	28 x 16 x 8 mm	30 ”[g]
Elelabs Zigbee Raspberry Pi Shield “[d]	+13 dBm ”[e]	28 x18 x 10 mm	25 ”[g]

Nota. Se tomó como referencia el dispositivo más económico para cada clase. ”[a] [73], “[b] [74], ”[c] [75].

En el caso de SkyConnect, el dispositivo es desarrollado directamente por Home Assistant, sin embargo, su disponibilidad es poca en tiendas online populares. El USB Dongle de SONOFF, por otro lado, si se encuentra disponible, sin embargo su tamaño es mayor. ZB-Stick es un dispositivo que, además de ser escaso, su potencia de transmisión es menor. Finalmente, El módulo de Elelabs es un Shield, lo que en algunos tipos de aplicación podría ser eficiente. Sin embargo, esto podría ser contraproducente en proyectos donde se usan varios módulos más. Además su potencia de transmisión es menor.

3.4.7. Análisis de los módulos Z-Wave considerados

Según [76], se recomiendan estos dispositivos para integrar Z-Wave a su sistema.

Tabla 12

Comparación entre dispositivos Z-Wave recomendados por Home Assistant.

Dispositivo	Corriente de transmisión	Dimensiones (mm)	Precio aproximado ($ USD)
Aeotec Z-Stick Gen5+	98 mA	67 x 13 ”[a]	60 ”[g]
Nortek HUSBZB-1	80 mA	32 x 14	40
Zooz ZST10	29 mA	13 x 11	29.95
Z-WaveMe UZB	15 mA	16.5 x 8	9.75

Nota. Se tomó como referencia el dispositivo más económico para cada clase. ”[a] [77], “[b] [78], “[c] [79], “[e] [80].

Existen algunos shields que se adaptan a dispositivos como los Raspberry. Sin embargo, no es posible utilizarlos para este proyecto porque entrarían en conflicto con pines dedicados a otros propósitos (para los módulos dedicados al resto de tecnologías de comunicación).

3.5. Criterios de diseño de PCB

Es importante considerar estos criterios de diseño para obtener una placa acorde a los parámetros eléctricos de nuestro sistema, como la corriente, resistencia y ancho de trazo, y más aún cuando usamos corrientes altas en nuestra placa. Estas fórmulas fueron establecidas por el Asociación de Conexiones Electrónicas (IPC) en la norma IPC-2221b.

3.5.1. Área Transversal y Ancho de Trazo

En la fórmula del área transversal, I es la corriente máxima de nuestro circuito, A es el área transversal, Trise es el crecimiento de temperatura y la constante k.

$$A = \left(\frac{I}{k \cdot T_{rise}{}^{b}}\right)^{\frac{1}{c}} \quad (1)$$

[81]

La constantes k, b y c están determinadas por la ubicación del trazo. Si se encuentra entre capas, k = 0.024, b = 0.44 y c = 0.725. Si se encuentra en el exterior, k = 0.048, b = 0.048 y c = 0.725.

Para calcular el ancho del trazo, se utiliza la siguiente fórmula, donde t es el grosor del cobre, w es el ancho de trazo y A es el área transversal.

$$W = \frac{A}{t \cdot 1.378} \quad (1)$$

[81]

3.6. Software

El software utilizado se ha seleccionado con el fin de facilitar el desarrollo del proyecto, optimizando los procesos y cumpliendo los objetivos de funcionamiento. Se destaca el software Home Assistant que funciona como el cerebro del proyecto. Software de desarrollo como Arduino IDE, útil para la programación de varios tipos de microcontroladores, Mosquitto MQTT Broker utilizado como intermediario de los datos enviados y finalmente InfluxDB, que facilita la administración de base de datos.

3.6.1. Home Assistant

Home Assistant funciona como el sistema operativo. A través de él se accede de una forma sencilla e intuitiva a todos los nodos del sistema de control, donde se incluyen mediciones eléctricas, mediciones realizadas por los diversos sensores y la oportunidad de controlar las cargas de manera sencilla.

3.6.2. Arduino IDE

Arduino IDE es un software de código abierto que se utiliza principalmente para escribir y compilar el código en el módulo Arduino, Este entorno admite los lenguajes C y C++ [82].

Sin embargo, también es capaz de trabajar con módulos de ESPRESSIF cuando instalamos los complementos necesarios.

3.6.3. Mosquitto MQTT Broker

Mosquitto es un intermediario de mensajes de código abierto que implementa el protocolo MQTT. Es liviano y adecuado para su uso en todos los dispositivos, desde computadoras de placa única de bajo consumo hasta servidores completos [83].

3.6.4. InfluxDB

InfluxDB es una base de datos de series temporales optimizada para cargas elevadas de escritura y consultas que se puede utilizar como solución de almacenamiento de datos para cualquier caso de uso que involucre grandes volúmenes de datos con marca de tiempo [84].

3.6.5. Autodesk Fusion 360

Este es una plataforma de programa computacional que permite desarrollar diseño asistido por ordenador o CAD, manufactura asistida por ordenador o CAM, ingeniería asistida por ordenador o CAE y de circuitos modelados en tres dimensiones en la nube tanto para el diseño, así como para la manufactura de productos. [85]

3.6.6. EasyEDA

EasyEDA es un conjunto de herramientas EDA (Easy Data Access) basadas en la nube que permite a los usuarios de hardware diseñar, simular, compartir, de manera pública y privada, y discutir esquemas, simulaciones y tableros de circuitos impresos. [86]

3.7. Hardware

Se utilizaron equipos especializados para la realización de las pruebas, enfocados en la medición de magnitudes eléctricas. Los instrumentos empleados cuentan con las

certificaciones pertinentes que los convierten en puntos de referencia importantes para obtener resultados fiables de error.

3.7.1. Banco de prueba de medidores de ZERA STM4000

ZERA es una empresa que fabrica y comercializada dispositivos y sistemas para producir, medir, probar y calibrar magnitudes eléctricas. [87] El Banco de pruebas multiposición STM4000 de este fabricante cuenta con hasta 10 posiciones para la realización de pruebas simultáneamente, y está combinado con una fuente de poder MTS310 para la prueba de medidores trifásicos. Este medidor cuenta con una clase de precisión del 0.02%. [88]

3.7.2. Analizador de prueba METREL POWER Q4 PLUS

Es un analizador de calidad de energía de gama alta. Con sus 4 corrientes y 4 canales de voltaje, es adecuado para localizar, predecir y solucionar problemas en sistemas de distribución de energía trifásicos y monofásicos. [89] Este sistema cuenta con una clase de precisión del tipo A de acuerdo con la normativa IEC 61000-4-30. [89]

3.7.3. Medidor residencial DDS8888 de BLUE SKY HI-TECH

Este medidor electrónico de energía monofásica adopta el avanzado técnica microelectrónica y la tecnología de fabricación de SMT. Se utiliza para la medición monofásica AC energía electrónica activa cuya frecuencia nominal es de 50 Hz o 60Hz. Cuenta con una clase de precisión tipo 1.0 y 2.0 de acuerdo con la normativa IEC 62053-21 e IEC 62052-11. [90]

CAPÍTULO IV
Resultados

En este capítulo se respondió la pregunta de investigación, exponiendo los resultados obtenidos en la implementación del proyecto, incluyéndose la selección de componentes, cálculos de las fuentes DC, cálculo de anchos de trazo, diagramas eléctricos, diagramas de flujo de los sistemas diseñados, esquemáticos de PCB, código de programación y todo lo referente al procedimiento en Home Assistant. Además, se realizó la validación de los medidores comparando los valores obtenidos con las especificaciones técnicas requeridas en la normativa nacional referente a los medidores electrónicos bifásicos.

4.1. Diagrama de funcionamiento general del Sistema

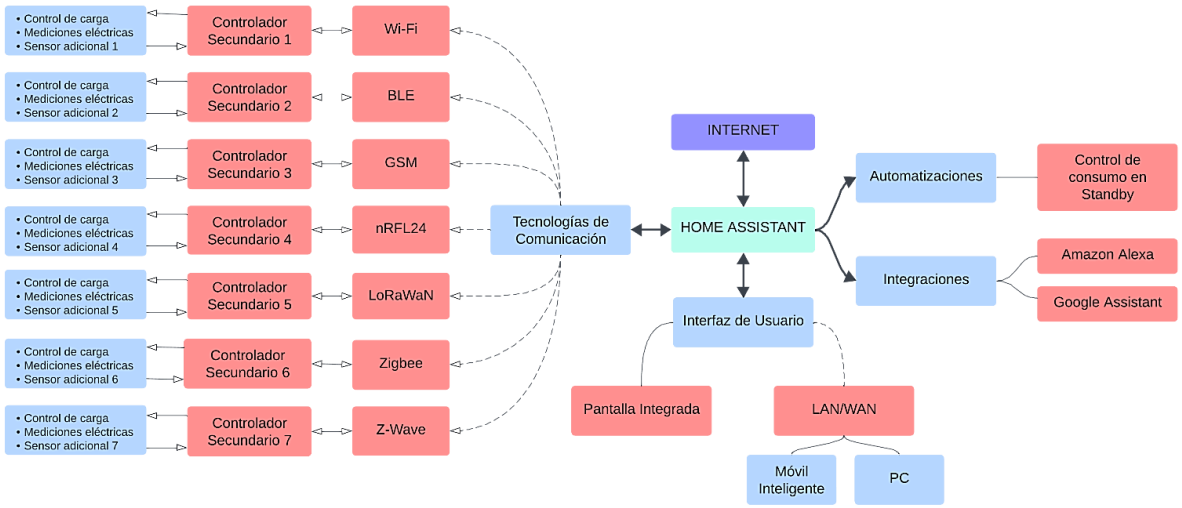
Como se muestra en la Figura 5, el Sistema Inteligente de Medición Inalámbrica funciona a través del software Home Assistant, el cual está instalado en un dispositivo compatible conectado a internet. Este dispositivo principal mantiene comunicación permanente con otros dispositivos secundarios que envían mediciones eléctricas, mediciones de sensores de distinto tipo y que reciben órdenes para control de carga desde Home Assistant. La comunicación de estos dispositivos es a través de distintas tecnologías (Wi-Fi, BLE, GSM, nRFL24, LoRa, Zigbee y Z-wave).

Es posible acceder a estos datos a través de un monitor integrado o desde otro dispositivo conectado a la red como un móvil inteligente o computador. Además se utiliza Amazon Alexa y Google Assistant para unificar estos medios con Home Assistant.

Home Assistant es utilizado para la programación de automatizaciones, que evitan el prolongado estado de Standby evitando un incremente de consumo constante en los electrodomésticos inactivos conectados en los nodos.

Figura 5

Diagrama de funcionamiento del Sistema Inteligente de Medición Inalámbrica.



Como se muestra en la Figura 5 cada nodo fue identificado con un número correspondiente. Además, los nodos fueron diseñados con el fin de medir variables eléctricas, controlar el relé de la carga y realizar una medición adicional.

4.2. Esquemáticos de conexión

A continuación, se presentaron los componentes de cada nodo previo a los criterios de selección del capítulo anterior junto a los esquemáticos de conexión utilizados.

4.2.1. Nodo maestro con Raspberry Pi 4B

El nodo maestro funciona utilizando un Raspberry Pi 4B como cerebro. Siendo energizado por un regulador con salida de 5VDC 3A. El Raspberry puede ser conectado a la red mediante Ethernet o a través de Wi-Fi, dependiendo de las posibilidades de conexión en el domicilio. El sistema operativo y la configuración se almacena en una Micro SD con unas altas velocidades de lectura y escritura. El USB Hub permitió aumentar los puertos disponibles originalmente en el Raspberry, y la posibilidad de energizarlos con una fuente externa,

necesarios para el correcto funcionamiento de los USB Stick y dispositivos encargados de recibir los datos de las tecnologías de comunicación utilizadas (Wi-Fi, BLE, GSM, nRF24, LoRa, Zigbee y Z-Wave)

Tabla 13

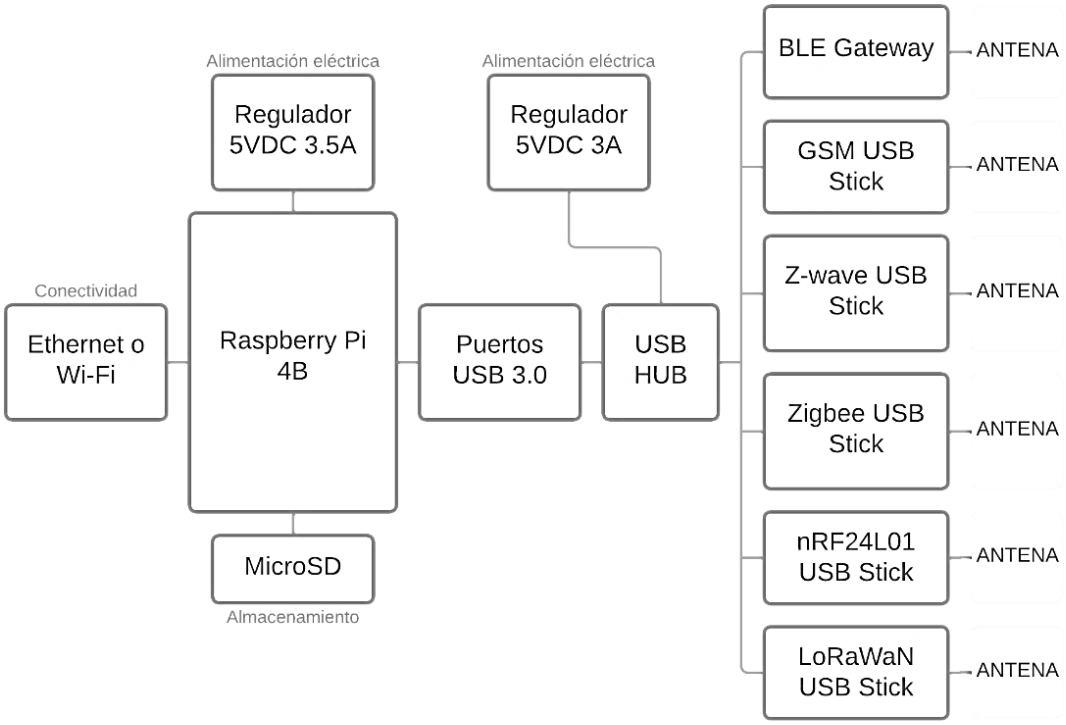
Principales componentes utilizados en el Nodo Maestro.

Dispositivo	Descripción
Raspberry Pi 4B 8GB	Minicomputador
CanaKit Fuente de Alimentación USB-C	Regulador 5VDC 3.5 A
REYAX RYLR998	LoRa USB Stick
ESP32-Devkit-C	BLE Gateway
nRF24L01	nRF24L01 USB Stick
TTGO ESP32 SIM800L	Microcontrolador con módulo GSM
Xbee S2C	Zigbee USB Stick
Zooz Serie 7 Z-wave	Z-wave USB Stick
Atolla Hub USB 3.0	USB Hub con fuente de alimentación externa

La conexión de los módulos al Raspberry se realizó mediante puertos USB, debido que facilita la comunicación con Home Assistant. En algunos nodos se utilizaron integraciones específicamente diseñadas para leer los USB Stick, mientras que en otros, se leyó el puerto serial para la obtención de los datos recibidos.

Figura 6

Esquemático del Nodo Maestro.



Como se observa en la Figura 6, cada uno de los dispositivos contó con su antena independiente con el fin de evitar las interferencias por frecuencia e incompatibilidades de esta.

4.2.2. Nodo 1 comunicado mediante Wi Fi

El primer nodo implementado fue el Nodo 1 comunicado a través de Wi-Fi. La implementación de este nodo se la realizó en base a los dispositivos seleccionados y el método de comunicación utilizado. Ambos factores se complementaron para la decisión del diseño estructural y de software.

Los componentes seleccionados en esta implementación son los enlistados en la siguiente tabla.

Tabla 14

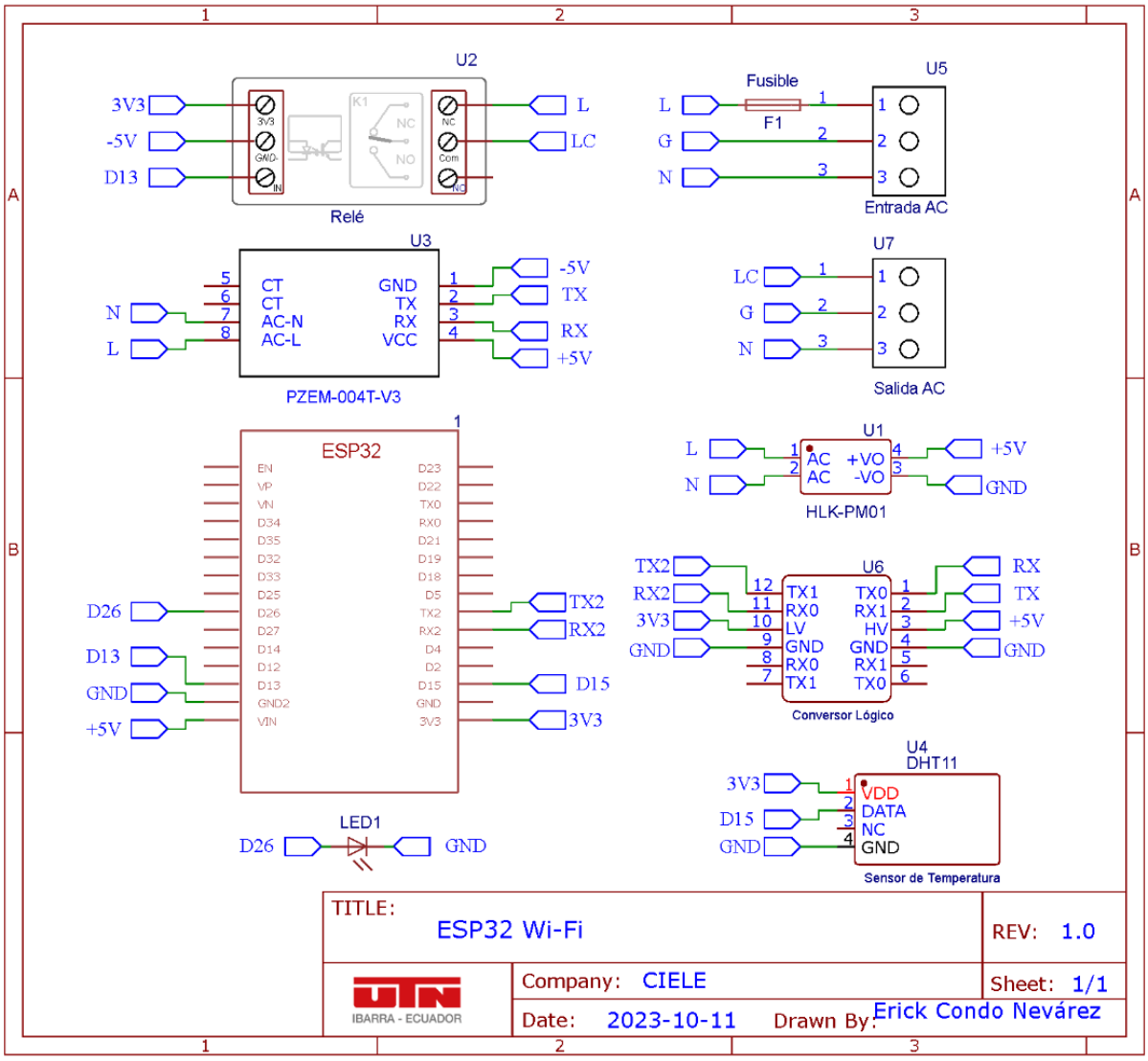
Principales componentes utilizados en el Nodo 1 con Wi Fi

Dispositivo	Descripción
ESP32-Devkit-C	Microcontrolador
PZEM 004T V3	Módulo Medidor AC
Módulo Relé AEDIKO con JQC-T78-DC05V-C	Módulo Relé
HLK-PM01	Fuente de Alimentación 5V
Conversor de nivel lógico	Módulo de 4 canales
DHT11	Sensor de Temperatura
Fusible 15A	Protección

El esquemático de conexión utilizado en el Nodo 1 es presentado a continuación utilizando los componentes de la tabla anterior. El circuito se divide en dos, el circuito de fuerza, que conforma las borneras, fusibles y los pines COM y NC/NO del relé, y el circuito de control, que conecta el microcontrolador ESP32 como cerebro principal del nodo.

Figura 7

Esquemático del Nodo 1 implementado con Wi-Fi



Se observa en la Figura 7 que, en la Entrada AC, se agregó un fusible como protección al circuito de control. Como sensor adicional, se utilizó un DHT11, el cual es capaz de entregar mediciones de temperatura y humedad.

4.2.3. Nodo 2 comunicado mediante BLE

Como se puede observar en la Tabla 15, los dispositivos son similares, esto debido a que se utilizó el soporte nativo BLE del ESP32, por lo cual no hubo cambios importantes en los componentes principales en este Nodo.

Tabla 15

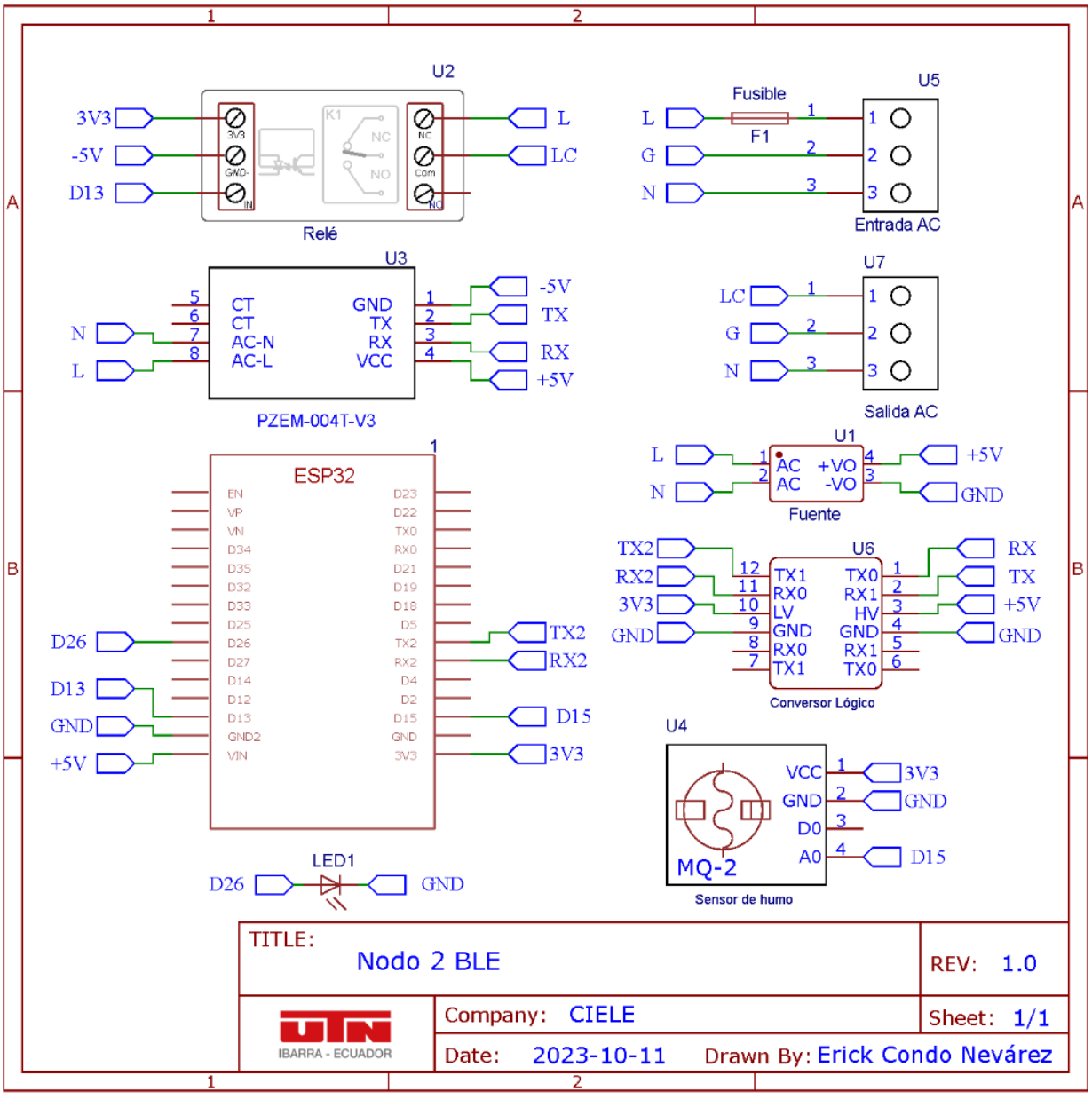
Principales componentes utilizados en el Nodo con BLE

Dispositivo	Descripción
ESP32-Devkit-C	Microcontrolador
PZEM 004T V3	Módulo Medidor AC
Módulo Relé AEDIKO con JQC-T78-DC05V	Módulo Relé
HLK-PM01	Fuente de Alimentación 5V
Conversor de nivel lógico	Módulo de 4 canales
MQ-2	Sensor de humo, etc
Fusible 15A	Protección

Sin embargo, como sensor adicional se utilizó un MQ-2 capaz de detectar presencia de humo, gas LP, i-butano, propano, metano, alcohol e hidrógeno en el domicilio, pero no se realizaron cambios importantes en el esquemático de la Figura 8, puesto utiliza los mismos pines de conexión con los dispositivos base del nodo anterior.

Figura 8

Esquemático del Nodo 2 implementado con BLE.



Varios de los sensores adicionales utilizados necesitan estar expuestos al exterior, por lo cual se diseñó el circuito con el fin de agregar estos módulos fuera de la PCB utilizando

pines y cables de conexión. El esquemático también fue conectado con el fin de utilizar los valores analógicos del sensor, esto para poder recibir mediciones más específicas del módulo que se esté utilizando.

4.2.4. Nodo 3 comunicado mediante GSM

Como se ve en la Tabla 16, los cambios más notables de este nodo fueron los dispositivos adicionales correspondientes a la fuente de voltaje, puesto que el módulo SIM800C funciona correctamente con un voltaje de 4.2 VDC a 2 A.

Tabla 16

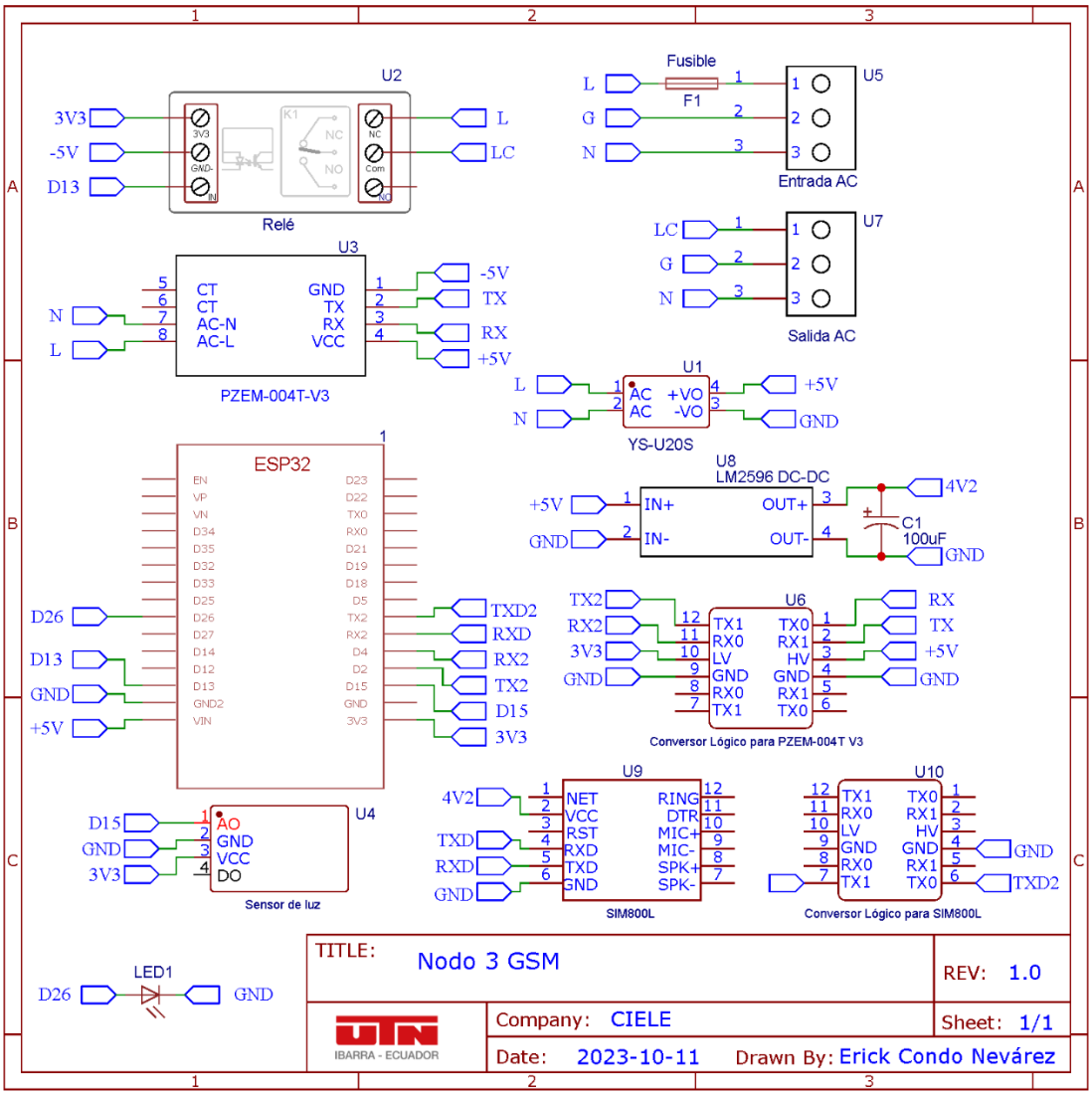
Principales componentes utilizados en el Nodo 3 con GSM

Dispositivo	Descripción
ESP32-Devkit-C	Microcontrolador
SIM800L	Módulo de comunicación GSM
PZEM 004T V3	Módulo Medidor AC
Módulo Relé AEDIKO con JQC-T78-DC05V	Módulo Relé AEDIKO con JQC-T78-DC05V
YS-U20S	Fuente de Alimentación 5V 3A
LM2596 DC-DC	Módulo regulador de voltaje DC
Conversor de nivel lógico	Módulo de 4 canales
Sensor luz fotodiodo	Sensor de luz
Fusible 15A	Protección

En el circuito correspondiente al módulo GSM SIM800L, se añadió un módulo Buck reductor de voltaje LM2596 DC-DC para obtener una salida de 4.2 V a 2A y alimentar el módulo de comunicación SIM800L. Se utilizó un capacitor de 3.3uF para mantener un voltaje estable en la entrada de voltaje al módulo de comunicación.

Figura 9

Esquemático del Nodo 3 implementado con GSM.



Como se observa en la Figura 9, se utilizó el conversor lógico adicional al del PZEM-004t V3 en la comunicación serial del SIM800L con el fin de tener una comunicación más estable y con menos ruido por las diferencias de voltaje presentes.

4.2.5. Nodo 4 comunicado mediante nRF24L01

En este nodo se utilizó la misma estructura base de los nodos 1 y 2, sin embargo, se añadió el módulo de comunicación correspondiente al nRF24L01, el cual se conecta al ESP32 mediante comunicación SPI.

Tabla 17

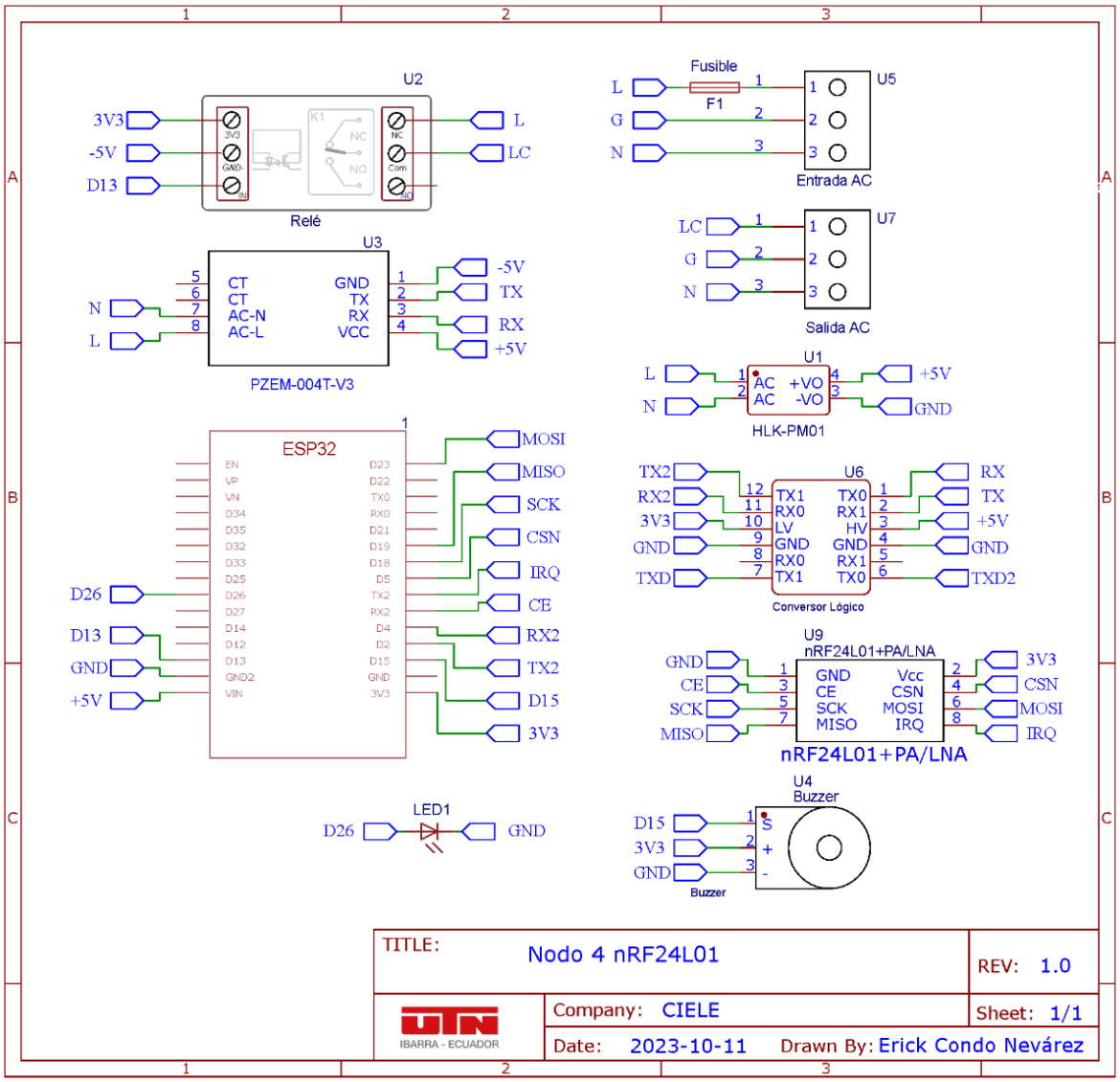
Principales componentes utilizados en el Nodo 4 con nRF24L01

Dispositivo	Descripción
ESP32-Devkit-C	Microcontrolador
nRF24L01 + regulador 3.3 V	Módulo de comunicación nRF24
PZEM 004T V3	Módulo Medidor AC
Módulo Relé AEDIKO con JQC-T78-DC05V	Módulo Relé AEDIKO con JQC-T78-DC05V
HLK-PM01	Fuente de Alimentación 5V 0.5 A
Conversor de nivel lógico	Módulo de 4 canales
Buzzer	Altavoz
Fusible 15A	Protección

Como se observa en la Figura 17, se utilizaron dos tipos de comunicación serial para este dispositivo, SPI y UART, por lo cual se utilizó todo el lado izquierdo del microcontrolador para el conexionado con los módulos.

Figura 10

Esquemático del Nodo 4 implementado con nRF24L01.



Se utilizó un actuador adicional en este módulo, el cual corresponde a un Buzzer. Esto fue ideado con el fin de realizar alarmas utilizando el resto de los nodos junto a las automatizaciones de Home Assistant. Por lo tanto, es posible emitir un sonido cuando tal dispositivo de otro nodo entra en Standby, cuando se detecta presencia, cuando el consumo de energía supera cierto límite, etc.

4.2.6. Nodo 5 comunicado mediante LoRa

En este nodo se utilizó una versión distinta del ESP32, correspondiente al ESP32-S3, pero con un módulo LoRa agregado al microcontrolador. El cual nos permitió ahorrar espacio en el esquemático y simplificar las conexiones.

Tabla 18

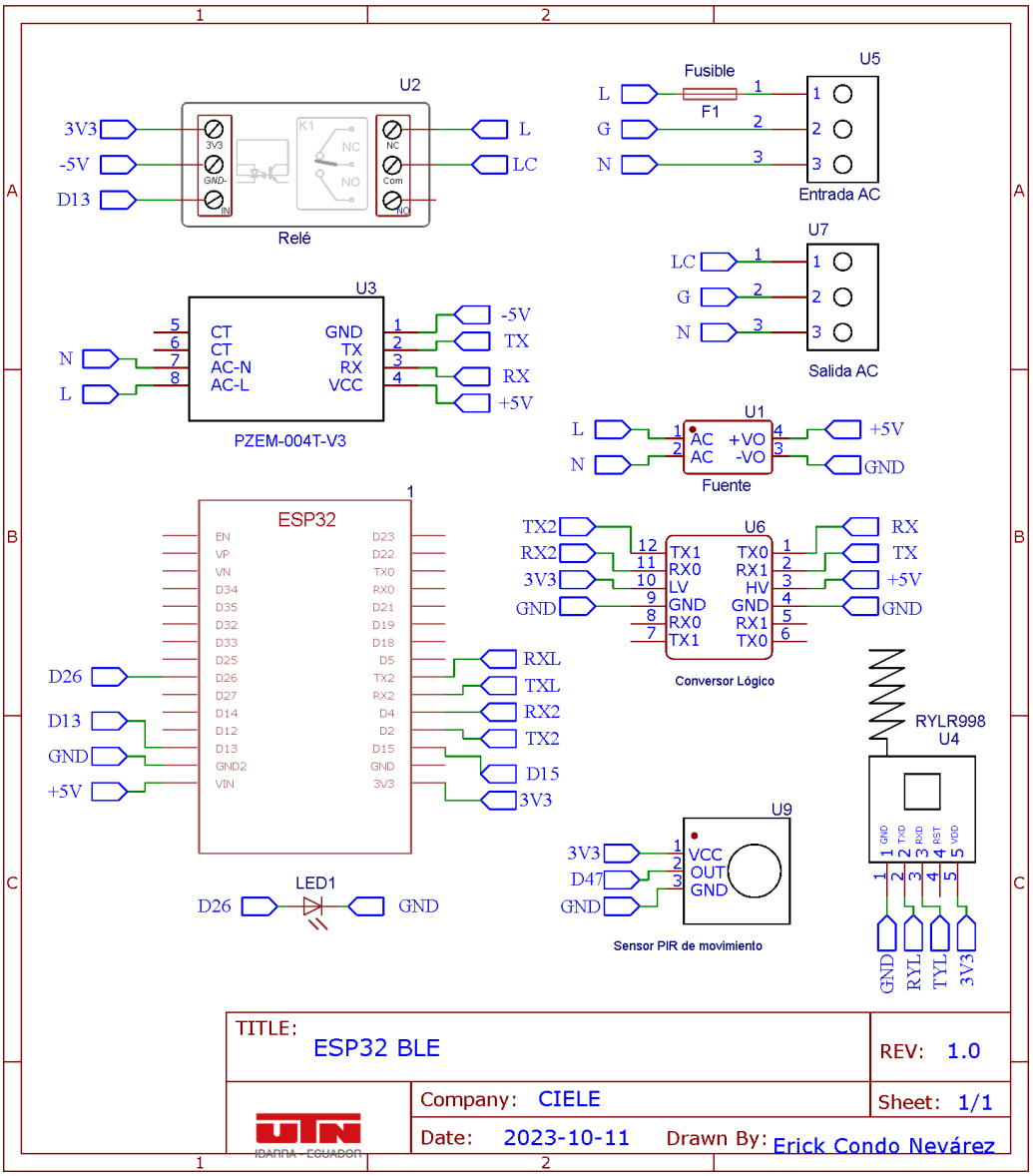
Principales componentes utilizados en el Nodo 5 con LoRa

Dispositivo	Descripción
ESP32-Devkit-C	Microcontrolador
REYAX RYLR998	Módulo LoRa
PZEM 004T V3	Módulo Medidor AC
Módulo Relé AEDIKO con JQC-T78-DC05V	Módulo Relé AEDIKO con JQC-T78-DC05V
HLK-PM01	Fuente de Alimentación 5V 0.5 A
Conversor de nivel lógico	Módulo de 4 canales
HC-SR501	Sensor de presencia
Fusible 15A	Protección

Como se observa en la Figura 11, al utilizar un ESP32, los voltajes de operación siguieron siendo los mismos, por lo cual se siguió utilizando el conversor de nivel lógico para la comunicación UART con el PZEM 004T V3. El conexionado del RYLR998 con el ESP32 se la realizó de forma directa usando los pines 16 y 17 del microcontrolador, los cuales son dedicados específicamente para comunicación UART.

Figura 11

Esquemático del Nodo5 implementado con LoRa



El sensor utilizado en este nodo fue el Sensor PIR (HC-SR501), dándonos la posibilidad de recibir una alerta al momento de detectar una persona dentro de una habitación a través de Home Assistant.

4.2.7. Nodo 6 comunicado mediante Zigbee

En este caso ocurre algo similar al Nodo anterior, obtenemos un conexionado directo del módulo de comunicación con el ESP32. El módulo Xbee S2C se conecta mediante comunicación UART a los mismos pines 16 y 17 del microcontrolador. No fue necesario utilizar el conversor de nivel lógico debido a que el módulo Xbee S2C opera con 3.3V al igual que el ESP32.

Tabla 19

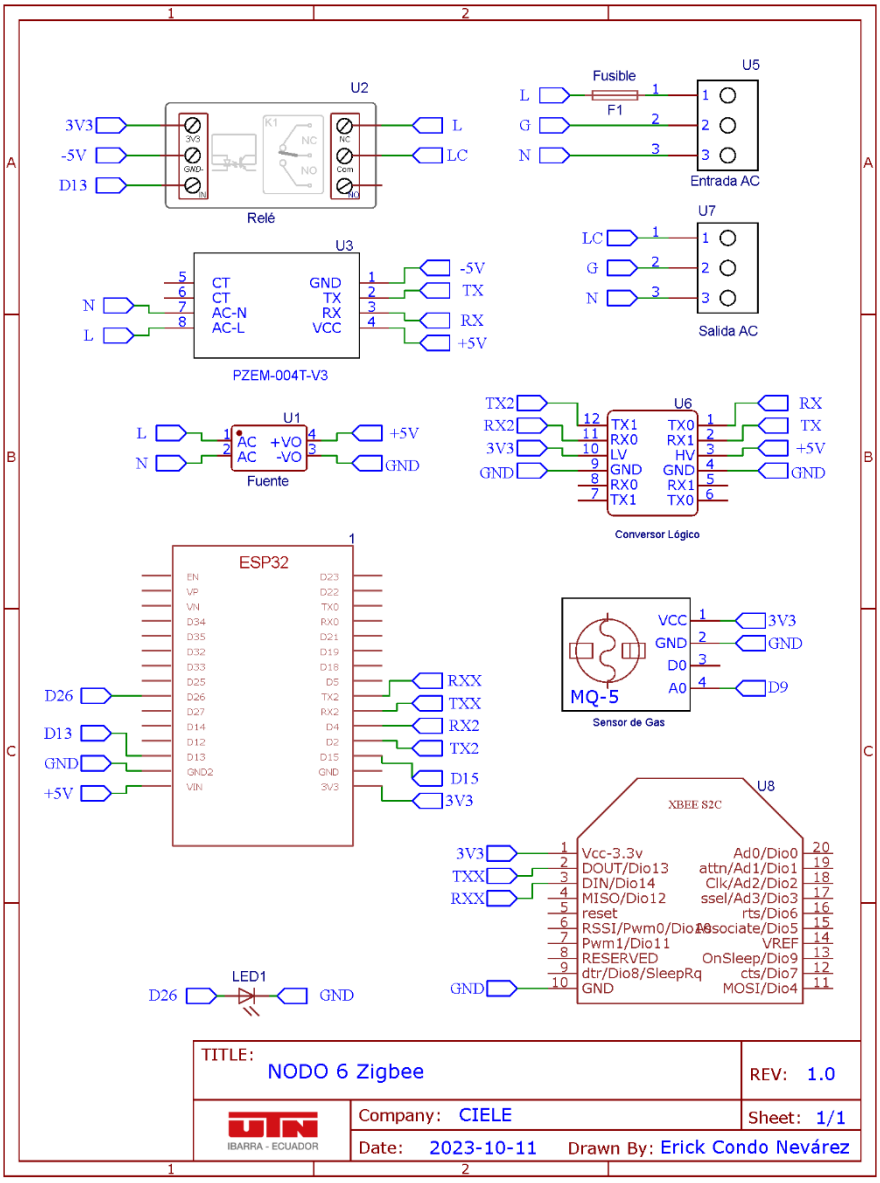
Principales componentes utilizados en el Nodo 6 con Zigbee

Dispositivo	Descripción
ESP32	Microcontrolador
Xbee S2C	Módulo Zigbee
PZEM 004T V3	Módulo Medidor AC
Módulo Relé AEDIKO con JQC-T78-DC05V	Módulo Relé AEDIKO con JQC-T78-DC05V
HLK-PM01	Fuente de Alimentación 5V 0.5 A
Conversor de nivel lógico	Módulo de 4 canales
MQ-5	Sensor de gas, etc
Fusible 15A	Protección

Como sensor, se integró un MQ-5 como sensor de gas natural y GLP, útil como medida de seguridad en un domicilio como prevención a fugas de gas.

Figura 12

Esquemático del Nodo 6 implementado con Zigbee



Como observamos en la Figura 12, no se agregaron componentes adicionales además del MQ-5, sin embargo, hubo cambios en el conexionado y orden de los pines debido a cambios en el microcontrolador.

5.1.1. Nodo 7 comunicado mediante Z-Wave

Luego de analizar las opciones de transceptores Z-Wave en el mercado, se utilizó un Z-Uno como módulo de comunicación aun siendo un microcontrolador. Éste se conecta con el ESP32 mediante serial, en modo de Gateway.

Tabla 20

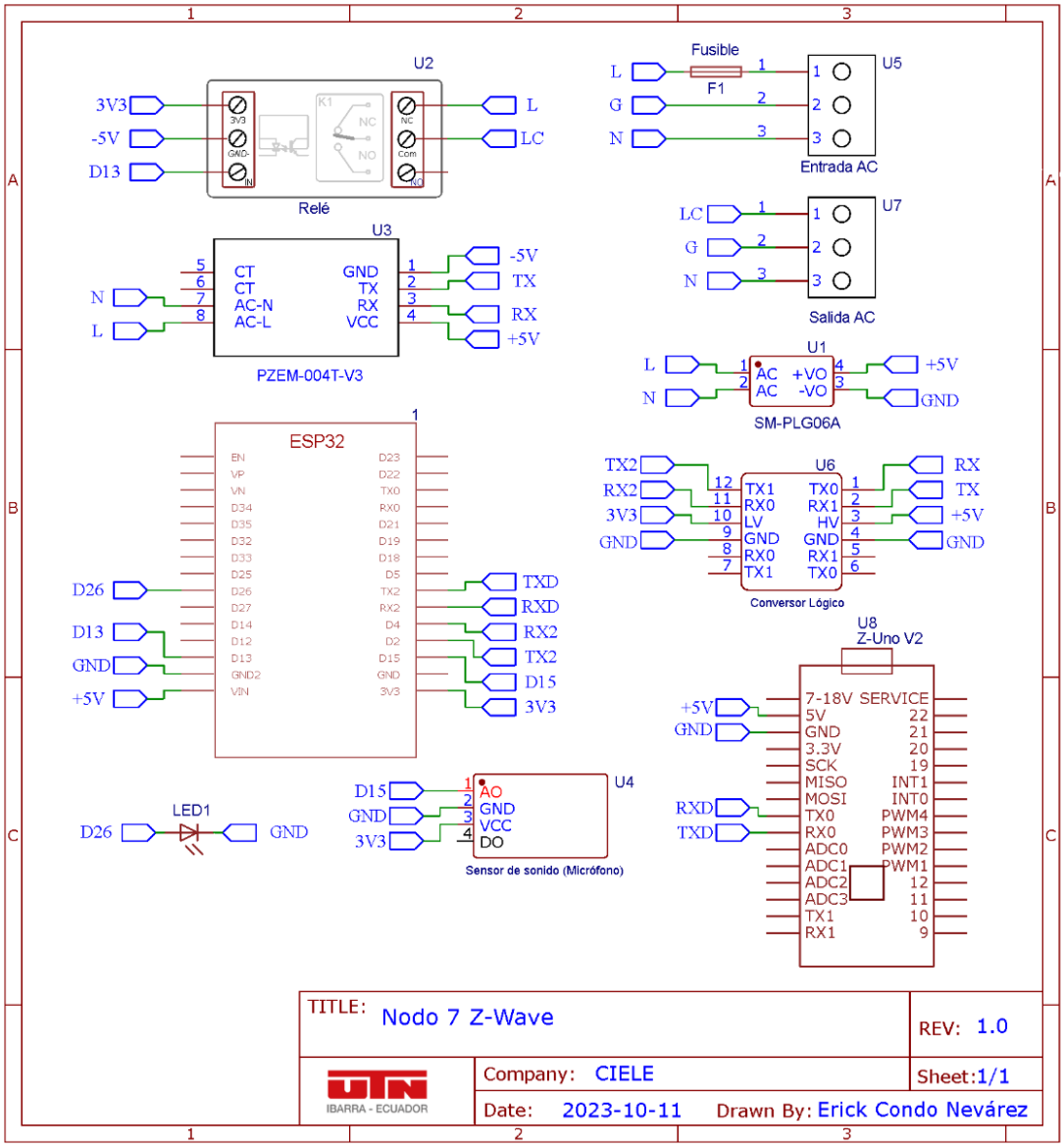
Principales componentes utilizados en el Nodo 7 con Z-Wave

Dispositivo	Descripción
ESP32-Devkit-C	Microcontrolador
Z-Uno 2	Microcontrolador con soporte Z-Wave
PZEM 004T V3	Módulo Medidor AC
Módulo Relé AEDIKO con JQC-T78-DC05V	Módulo Relé AEDIKO con JQC-T78-DC05V
SM-PLG06A	Fuente de Alimentación 5V 2A
Conversor de nivel lógico	Módulo de 4 canales
Micrófono	Sensor de sonido
Fusible 15A	Protección

El uso del Z-Uno 2 facilitó la implementación de la comunicación mediante Z-Wave, esto debido a que el microcontrolador utiliza Arduino IDE para su programación. Además, es uno de los únicos dispositivos del mercado que cuenta con certificación Z-Wave Plus, por lo cual será compatible como cualquier dispositivo Z-Wave comercial. Como se observa en la Figura 13, la comunicación entre el ESP32 y el Z-Uno se la realizó a través de UART, y se utilizaron pines digitales para emitir ordenes de activación o desactivación del relé y reinicio de energía.

Figura 13

Esquemático del Nodo 7 implementado con Z-Wave



Como sensor, se utilizó un micrófono con el fin de usarse como detector de sonido, además dándonos la posibilidad de utilizar efectos sonoros como aplausos para la activación de otro dispositivo a través de automatizaciones de Home Assistant.

5.2. Dimensionamiento de las fuentes de alimentación DC

El cálculo de la fuente de alimentación se la ha realizado en base a las potencias nominales de los dispositivos utilizados en cada nodo, por lo cual, se presentan los cálculos y resultados obtenidos.

5.2.1. Dimensionamiento de la fuente de alimentación DC del Nodo Maestro (USB Hub)

Para la Raspberry Pi 4B, Se seleccionó una fuente de alimentación de 5VDC de 3.5 A por recomendación de Home Assistant. Sin embargo, se calculó la alimentación utilizada en el USB Hub para soportar los USB Sticks y otros dispositivos conectados. Estos cálculos se presentan a continuación.

$$P_{nMaestro} = P_{ESP32Gateway\,(BLE)} + P_{SIM800C\,(GSM)} + P_{nRF24L01} + P_{SONOFFZigbee\,(Zigbee)} + P_{Zooz700(Z-Wave)}$$

$$P_{N1,2} = 0.5\,A\,(3.3\,V) + 0.9\,A\,(4.5\,V) + 0.01\,A\,(3.6\,V) + 0.9\,A\,(5\,V) + 0.9\,A\,(5\,V)$$

$$P_{nMaestro} = 14.758\,W$$

$$I_{nMaestro} = \frac{14.758\,W}{5\,VDC} = 2.9516 \approx 3\,A$$

El dispositivo RYLR998 fue conectado directamente a la Raspberry, con el fin de no sobrecargar el USB Hub. Se consideró una corriente de 0.9 A (la máxima soportada en puertos USB 3.0 y 3.1) en dispositivos donde no se ha otorgado información por el fabricante.

5.2.2. Dimensionamiento de la fuente de alimentación DC del Nodo 1 y 2 (Wi-Fi y BLE)

Los Nodos 1 y 2 son los dispositivos que requirieron una fuente de alimentación de menor amperaje, debido a que el ESP32-Devkit-C ya cuenta con soporte Wi-Fi y BLE, por lo cual no utiliza módulos de comunicaciones adicionales.

$$P_{N1,2} = P_{ESP32} + P_{MRELÉ} + P_{LED} + P_{DHT11} + P_{CLÓGICO} + P_{PZEM}$$

$$P_{N1,2} = 0.5\ A\ (3.3\ V) + 0.0714\ A\ (5\ V) + 0.02\ A\ (1.8\ V) + 0.0025\ A\ (3.3\ V) + 1 \cdot 10^{-7}\ A\ (1.5\ V) + 0.04\ A\ (5\ V)$$

$$P_{N1,2} = 2.32\ W$$

$$I_{N1,2} = \frac{2.32\ W}{5\ VDC} = 0.464\ A \approx 0.5\ A$$

En base a esto, se estimaron fuentes de 5VDC 0.5 A para el correcto funcionamiento de todos los componentes en ambos nodos.

5.2.3. Dimensionamiento de la fuente de alimentación DC del Nodo 3 (GSM)

En caso de los nodos donde se incluyen módulos de comunicación adicionales, se tomó la potencia total de 2.32 W de los nodos 1 y 2 con la potencia nominal de los módulos adicionales en cada dispositivo.

$$P_{N3} = P_{NODO1,2} + P_{SIM800L}$$

$$P_{N3} = 2.32\ W + 2\ W = 5.17\ W$$

$$I_{N3} = \frac{4.32W}{5\ VDC} = 0.864\ A \approx 1\ A$$

En base a esto, se estimó una fuente de 5VDC 1A para el correcto funcionamiento del nodo 3 implementado con SIM800L.

5.2.4. Dimensionamiento de la fuente de alimentación DC del Nodo 4 (nRF24L01)

En este caso, el componente adicional es el módulo de comunicación nRF24L01. De la misma forma, se tomó como base la potencia calculada de los nodos 1 y 2.

$$P_{N4} = P_{NODO1,2} + P_{NRF24L01}$$

$$P_{N4} = 2.32\ W + 6 \cdot 10^{-2}\ W = 2.38\ W$$

$$I_{N4} = \frac{2.38\ W}{5\ VDC} = 0.476\ A \approx 0.5\ A$$

Se estimó una fuente de 5VDC 0.5 A para el funcionamiento del nodo 4 integrándose el módulo nRF24L01.

5.2.5. Dimensionamiento de la fuente de alimentación DC del Nodo 5 (LoRa)

En este dispositivo, la potencia es la misma que los nodos 1 y 2, simplemente el microcontrolador ESP32 cambia de versión, donde se utilizó el ESP32-S3 con soporte de LoRa. La potencia consumida en todos los dispositivos de ESPRESSIF es la misma.

$$P_{N5} = P_{NODO1,2\ (con\ Heltec\ ESP32-S3\ LoRa)}$$

$$P_{N5} = 2.32\ W$$

$$I_{N5} = \frac{2.32\ W}{5\ VDC} = 0.464\ A \approx 0.5\ A$$

Se obtuvieron los mismos valores de potencia y corriente anteriores, puesto que según los datasheet de Espressif, los ESP32 no aumentan ni disminuyen su corriente nominal con el cambio de versiones. Se utilizó una fuente de 5 VDC a 0.5 A.

5.2.6. Dimensionamiento de la fuente de alimentación DC del Nodo 6 (Zigbee)

Igual que el caso del Nodo 5, se utiliza una variante de ESP32. Específicamente el ESP32 C3 con soporte nativo a Zigbee.

$$P_{N6} = P_{NODO1,2\ (con\ ESP32-C6)}$$

$$P_{N6} = 2.32\ W$$

$$I_{N6} = \frac{2.32\ W}{5\ VDC} = 0.432\ A \approx 0.5\ A$$

El resultado fue una fuente de 5 VDC a 0.5 A como anteriormente.

5.2.7. Dimensionamiento de la fuente de alimentación DC del Nodo 7 (Z-Wave)

En este nodo, se agregó el microcontrolador Z-Uno, que se encarga de la comunicación a través de la tecnología Z-Wave.

$$P_{N7} = P_{NODO1,2} + P_{Z-Uno}$$

$$P_{N7} = 2.32\ W\ + 0.4\ A\ (5\ V)$$

$$P_{N7} = 4.32\ W$$

$$I_{N7} = \frac{4.32\ W}{5\ VDC} = 0.864\ A \approx 1\ A$$

Puesto que se está integrando un microcontrolador más, se necesitó una fuente de alimentación más potente, por lo que se usó una fuente 5 VDC a 1 A.

5.3. Cálculo de Trazo

La corriente utilizada para el cálculo de trazo se basó en la fuente de alimentación utilizada para cada nodo. Las fórmulas utilizadas fueron las de la norma IPC-2221b expuesta en el capítulo anterior. A continuación, se presentan los cálculos realizados para la determinación del ancho de trazo.

Con 15 A:

$$\mathrm{A} = \left(\frac{I}{0.048\ \cdot\ 10^{0.44}}\right)^{\frac{1}{0.725}}$$

$$\mathrm{A} = 682.77\ mils^2$$

$$\mathrm{W} = \frac{A}{1\ \cdot 1.378} = 12.58\ mm$$

Con 1 A:

$$\mathrm{A} = \left(\frac{I}{0.048\ \cdot\ 10^{0.44}}\right)^{\frac{1}{0.725}}$$

$$\mathrm{A} = 16.30\ mils^2$$

$$W = \frac{A}{1 \cdot 1.378} = 0.3\ mm$$

Con 0.5 A:

$$A = \left(\frac{I}{0.048 \cdot 10^{0.44}}\right)^{\frac{1}{0.725}}$$

$$A = 6.26\ mils^2$$

$$W = \frac{A}{1 \cdot 1.378} = 0.11\ mm$$

A excepción del ancho de trazo que requiere el sistema de control AC (de 15 A), el resto de los dispositivos requieren un ancho de trazo bastante delgado, por lo que para estos, se determinó un ancho de trazo mínimo de 2 mm con el fin de evitar defectos en la elaboración de las pistas de cobre.

5.4. Diseño de PCBs y carcasas 3D

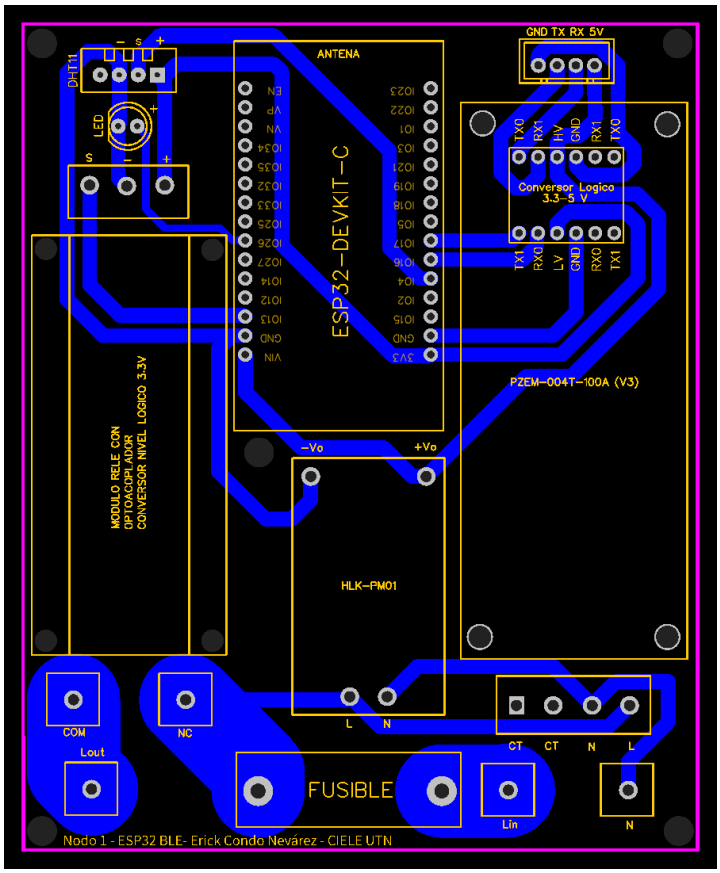
Puesto que cada nodo está compuesto de módulos con disposición de pines diferentes, las dimensiones de las PCBs varían, de la misma manera el diseño de las carcasas 3D. En las carcasas, se realizaron aperturas con el propósito de encajar los conectores de entrada y salida de la alimentación AC. Para las entradas se utilizaron conectores de tipo IEC C14, para las salidas, conectores NEMA 5-15p con el fin de evitar conectar los puertos a la inversa. Algunos cambios en las aperturas realizadas en el diseño de las carcasas 3D dependieron del sensor aplicado en el nodo.

5.4.1. Nodo 1 Wi-Fi

En la Figura 22 podemos observar el diseño de la PCB de una sola capa con dimensiones de 11.3 x 9 cm. Se utilizó un ancho de trazo mínimo de 2 mm debido a la disponibilidad de espacio en la placa, y se utilizó un ancho de trazo de 12.58 mm para el circuito de control de 15 A.

Figura 14

Esquemático de PCB del Nodo 1 implementado con Wi-Fi



Como se observa en la Figura 14, se diseñó una carcasa de dimensiones 12x11x5.45 cm con el fin de sostener la PCB. Se realizaron las aberturas correspondientes al conector IEC 320 C14 para la entrada de alimentación AC, al conector NEPA 5-15p para la salida, a dos leds

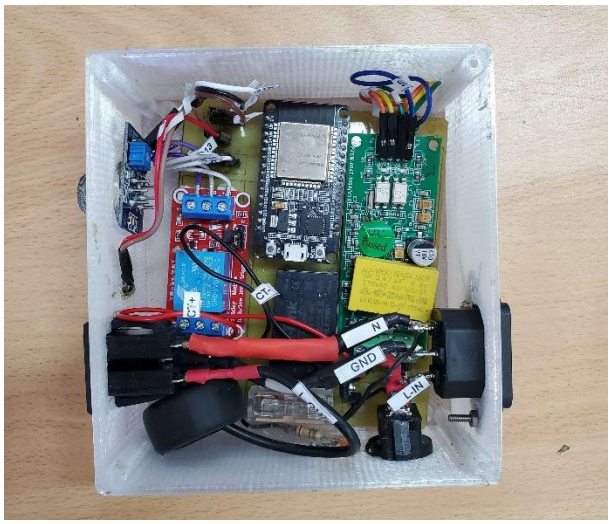
indicadores de alimentación y de constante de medir, a un switch pulsador de encendido y apagado, y una abertura para exponer el sensor de temperatura y ambiente DHT-11 al exterior.

Figura 15

Modelo 3D de la carcasa del Nodo 1 implementado con Wi-Fi

Figura 16

Nodo 1 completo con la carcasa impresa y su PCB



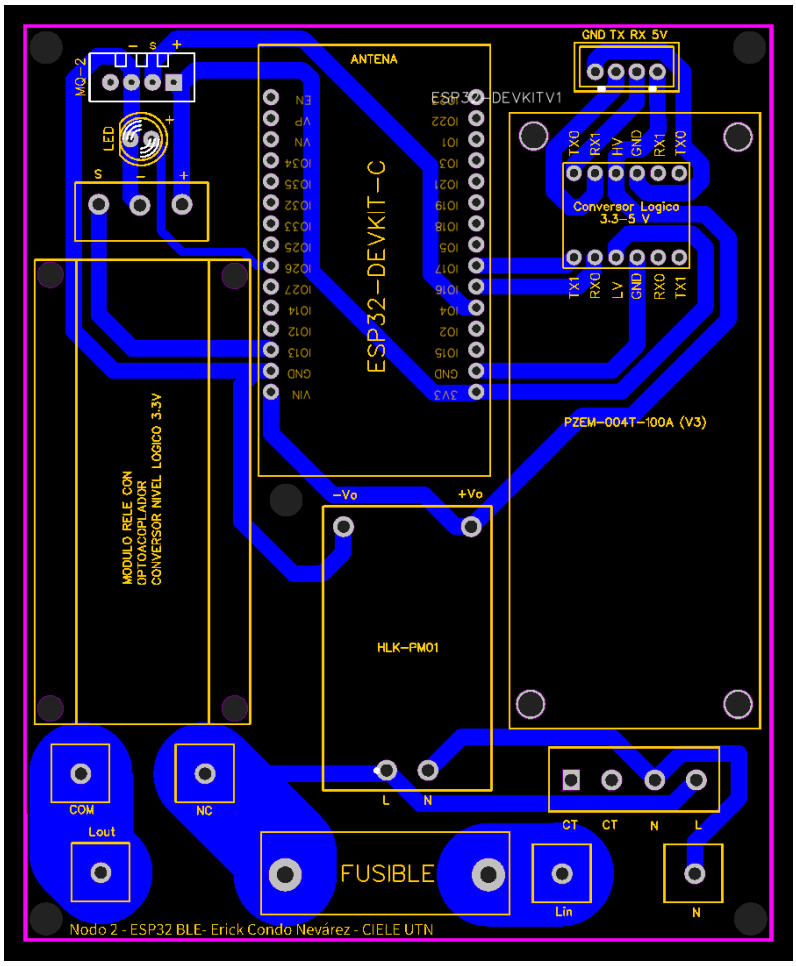
Como se observa en la Figura 16, se utilizó cable flexible de calibre 12 AWG como conductor del circuito de fuerza en el Nodo. También se conectaron los dispositivos externos a la PCB como el sensor DHT11 y los leds indicadores utilizando conectores.

5.4.2. Nodo 2 BLE

En la Figura 17 podemos observar el diseño de la PCB. En este caso, tanto las dimensiones de la PCB como de la caja son las mismas que el nodo anterior, solo que hay pequeños cambios en la serigrafía y en las dimensiones de la apertura para el sensor adicional, que en este caso es un sensor de humo MQ-2.

Figura 17

Esquemático de PCB del Nodo 2 ESP32 implementado con BLE



Como se observa en la Figura 18, el único cambio en la carcasa fue la apertura para el sensor adicional, que en este caso es el MQ2, ya que debía mantenerse expuesto a la intemperie.

Figura 18

Modelo 3D de la carcasa del Nodo 2 ESP32 implementado con BLE

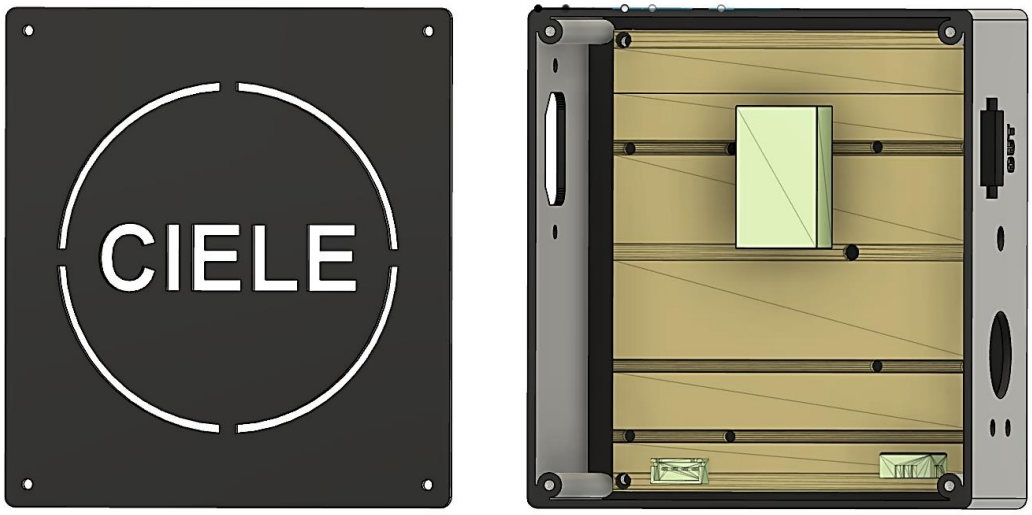
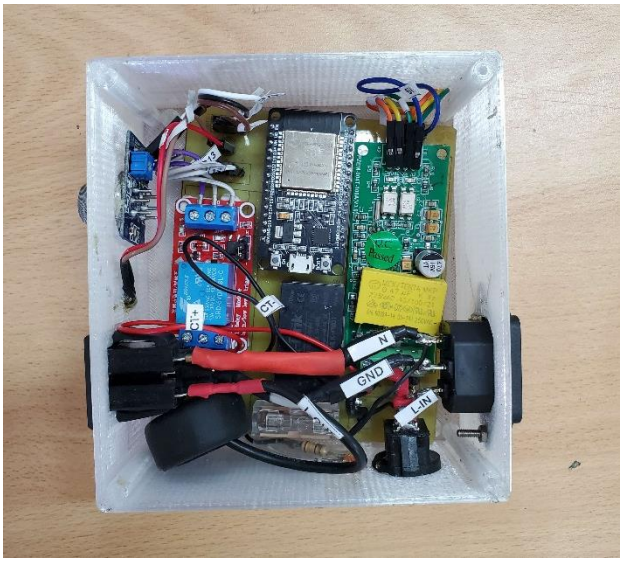


Figura 19

Nodo 2 completo con la carcasa impresa y su PCB.



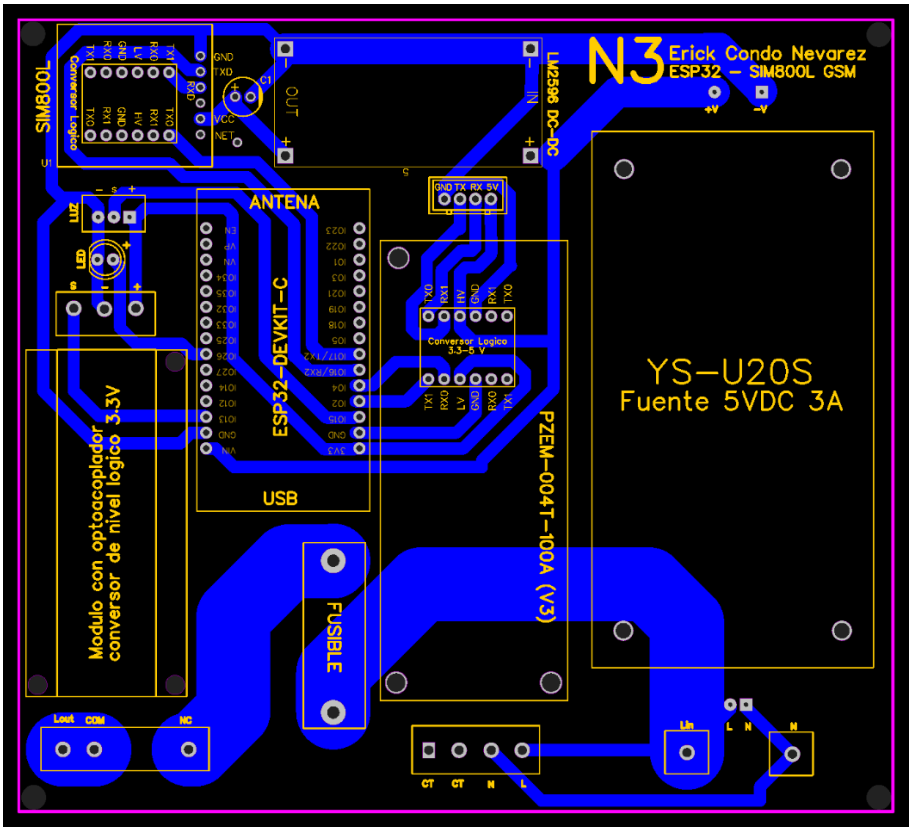
En la imagen de la Figura 19 observamos una caja similar al del Nodo 1, sin embargo, en el lateral izquierdo de la caja observamos el sensor MQ-2 sobresaliendo de la caja.

5.4.3. Nodo 3 GSM

En la Figura 20 podemos observar el diseño de la PCB para el Nodo 3. Se diseñó la PCB a una sola capa, con las dimensiones de 12.7 x 141.2 cm, siendo 3 cm más alargado que los nodos 1 y 2. Para la alimentación del módulo GSM, se realizaron trazos más anchos que el resto para minimizar la caída de voltaje por la distancia realizada.

Figura 20

Esquemático de PCB del Nodo 3 ESP32 implementado con GSM



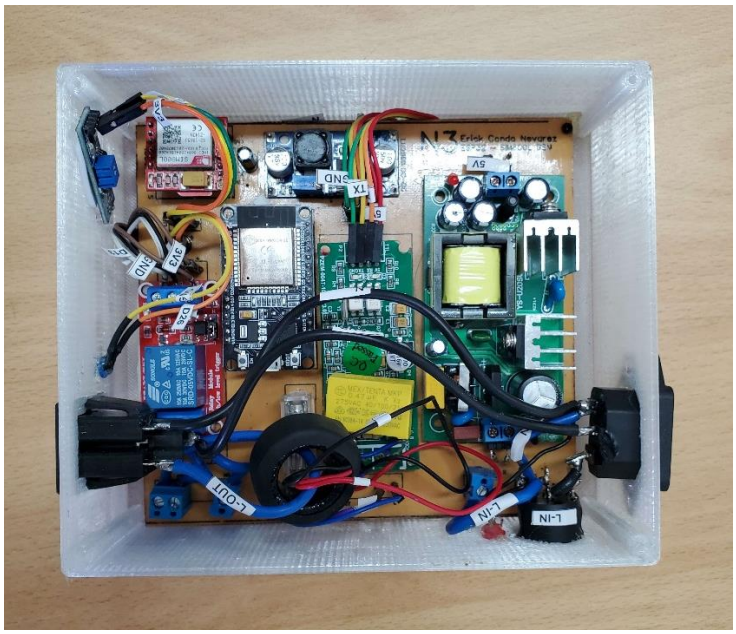
Debido al tamaño de la PCB, las dimensiones de la carcasa incrementaron a 15.5x 11x5.45. Se realizaron las mismas aberturas que los nodos anteriores, sin embargo se realizó el cambio en la apertura para el sensor, que en este caso correspondía a un agujero de 5.9 mm de diámetro para el fotodiodo de sensor de luz. El modelo de la carcasa se muestra a continuación.

Figura 21

Modelo 3D de la carcasa del Nodo 3 ESP32 implementado con GSM

Figura 22

Nodo 3 completo con la carcasa impresa y su PCB.



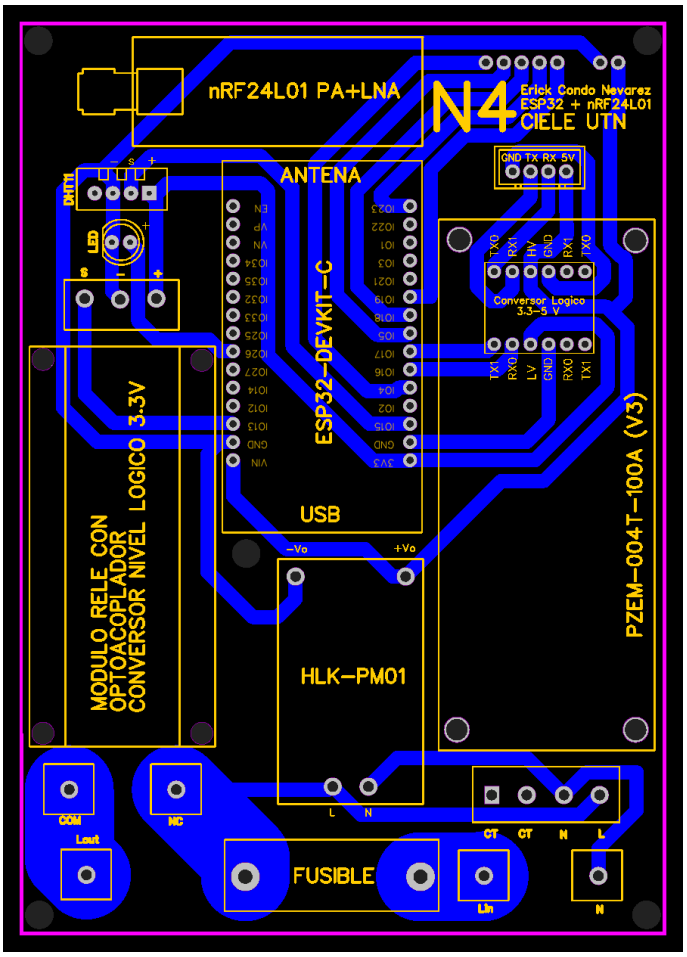
Finalmente, el Nodo 3 fue el de mayores dimensiones debido al tamaño de la fuente de alimentación del circuito de control, como se puede observar en la Figura 22.

5.4.4. Nodo 4 nRF24

La PCB para este nodo fue diseñada considerando que la mayoría de pines del lado izquierdo del ESP32 debían ser ocupados por el nRF24L01 PA+LNA para establecer la comunicación serial. Las dimensiones de la placa son de 12.7 x 9.0 cm, como se observa en la Figura 29.

Figura 23

Esquemático de PCB del Nodo 4 ESP32 implementado con nRF24.



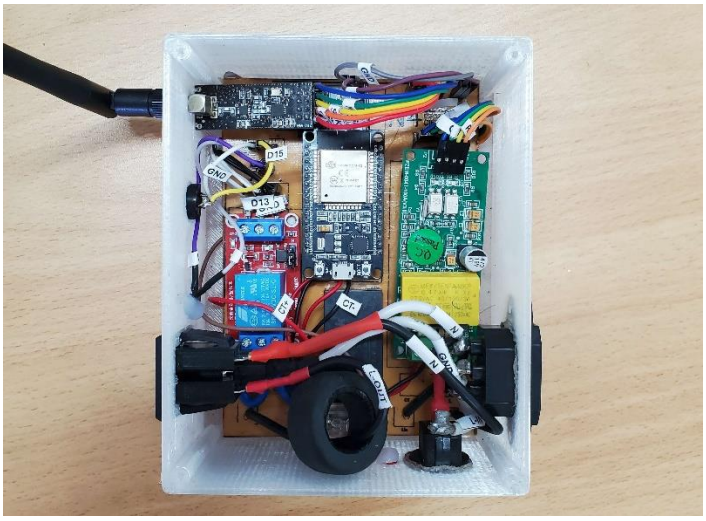
La principal diferencia entre las otras cajas es la apertura para la antena del transceptor nRF24 y una abertura adicional donde se utilizaron las dimensiones correspondientes al buzzer,

Figura 24

Modelo 3D del Nodo 4 ESP32 implementado con nRF24.

Figura 25

Nodo 4 completo con la carcasa impresa y su PCB.



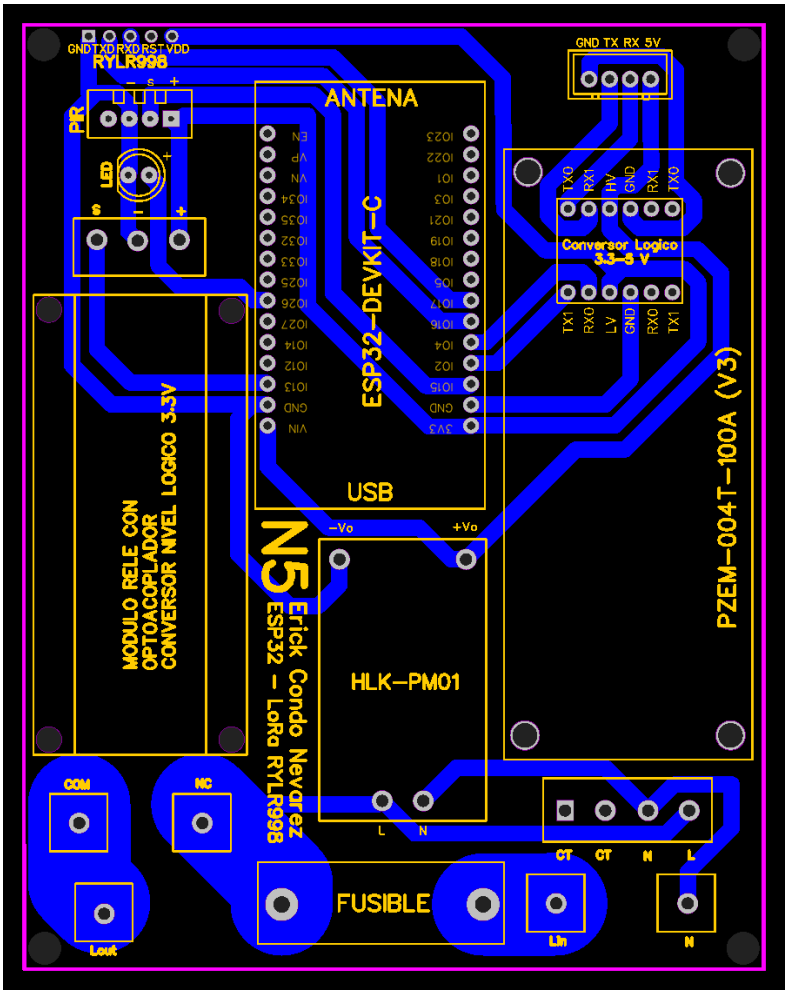
De esta manera, la antena queda sujeta a presión con la carcasa y es posible maniobrarla para cambiar su ángulo y dirección, como se observa en la Figura 25.

5.4.5. Nodo 5 LoRa

El Nodo 5 implementando el módulo LoRa RYLR998 fue sencillo de realizar, ya que este dispositivo utiliza una comunicación UART de 3.3V. Se cambiaron los pines de comunicación del PZEM a los de los pines 2 y 4 y el RYLR998 utilizó los pines 16 y 17. 11.5 x 9.0 cm

Figura 26

Esquemático de PCB del Nodo 5 ESP32 implementado con LoRa.



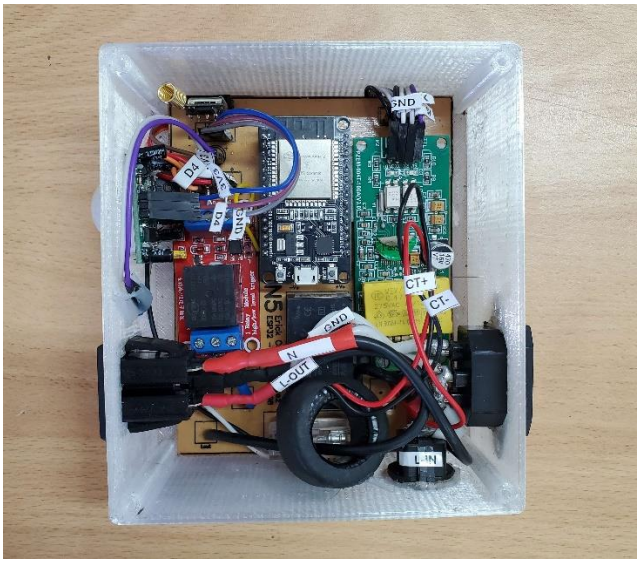
Debido a que el sensor de este nodo es el sensor de presencia PIR, se realizó una apertura de cuadrangular donde cabe perfectamente con el módulo, junto a los tornillos necesarios para la adhesión a las paredes de la placa.

Figura 27

Modelo 3D del Nodo 5 ESP32 implementado con LoRa.

Figura 28

Nodo 5 completo con la carcasa impresa y su PCB.



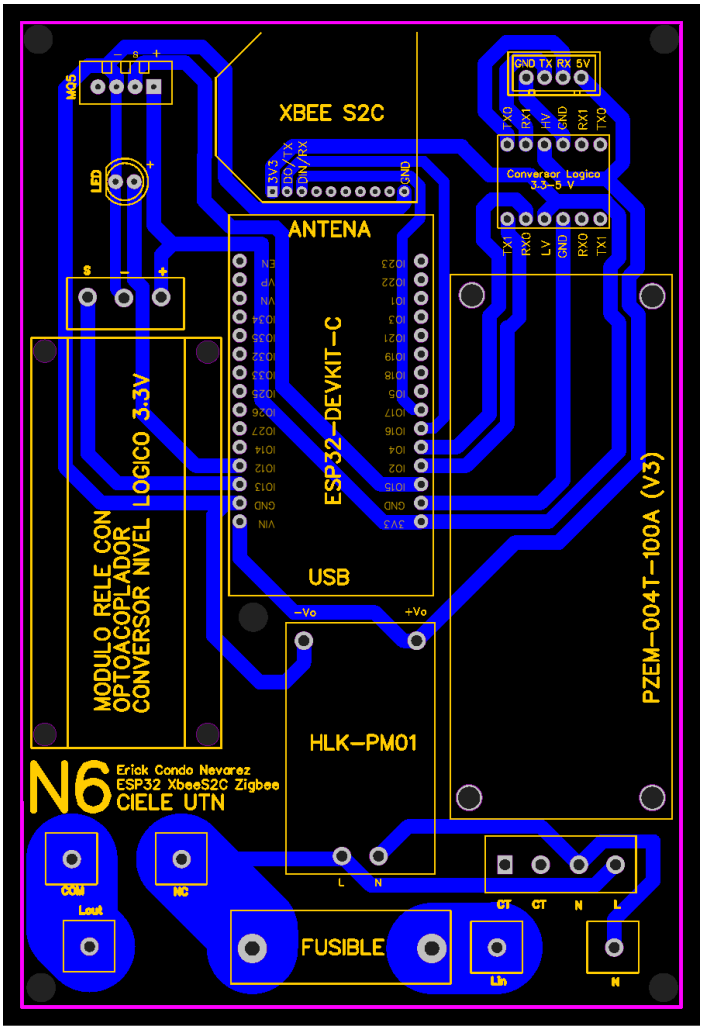
En la Figura 28 se observa que no existieron muchas diferencias en cuanto a las dimensiones del Nodo debido que el transceptor LoRa RYLR998 es bastante compacto.

5.4.6. Nodo 6 Zigbee

Como se mencionó anteriormente, se utilizará Zigbee implementando el módulo XBEE S2C, por lo cual la PCB incremento en la parte superior. En la Figura 32 se observa que se incrementaron las conexiones para el VCC, GND, TX Y RX del módulo de comunicación.

Figura 29

Esquemático de PCB del Nodo 6 ESP32 implementado con Zigbee



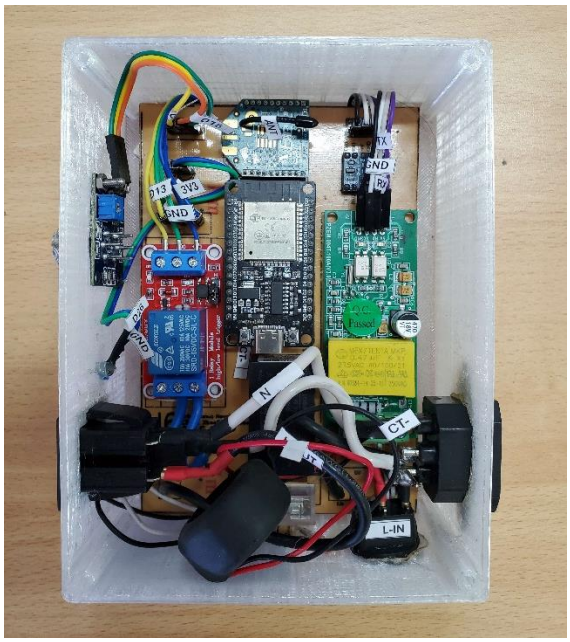
Los pines que cambiaron fueron los utilizados en comunicación UART y los GPIO dedicados a la entrada y salida de datos. En el modelo 3D de la Figura 29 observamos que la apertura del sensor es similar al del Nodo 2 BLE ya que utiliza un sensor del mismo tipo, el MQ2, con la misma estructura que el MQ5.

Figura 30

Esquemático de PCB del Nodo 6 ESP32 implementado con Zigbee

Figura 31

Nodo 6 completo con la carcasa impresa y su PCB.



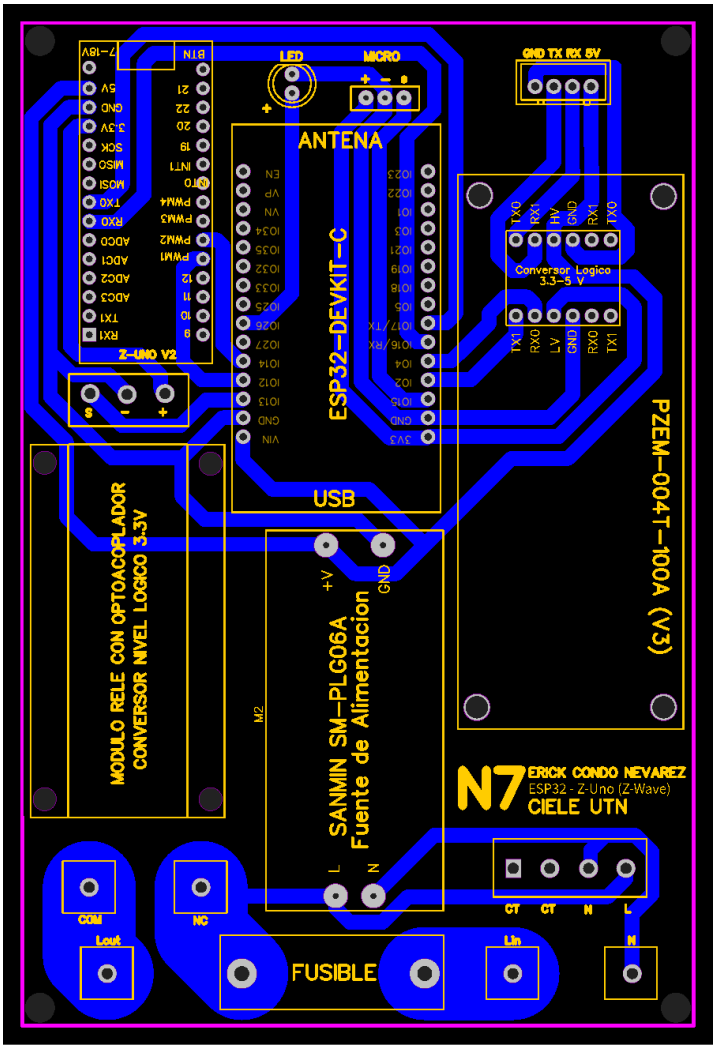
La Figura 30 nos muestra que también se expandió la caja de forma vertical para poder otorgarle el suficiente espacio a las conexiones del módulo Xbee S2C.

5.4.7. Nodo 7 Z-Wave

En el Nodo 7 se cambió la disposición de algunos elementos como el led indicador y el sensor adicional, con el fin de colocar el microcontrolador Z-uno sin incrementar demasiado las dimensiones de la placa, como se muestra en la Figura 32.

Figura 32

Esquemático de PCB del Nodo 7 ESP32 implementado con Z-Wave



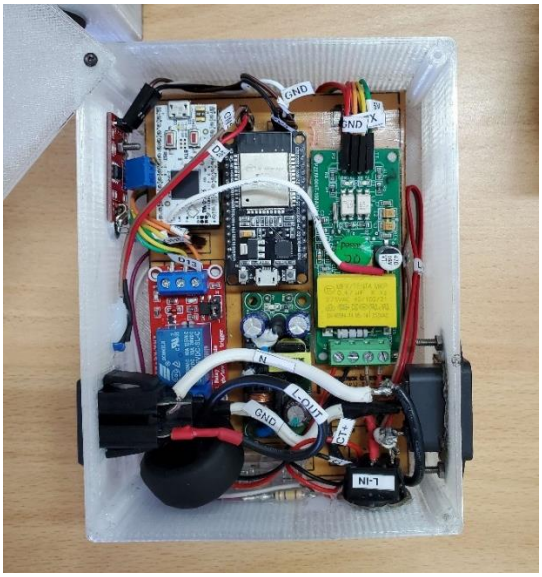
En la Figura 33, se observa que las dimensiones de la PCB resultaron de 13.4 x 11 x 5.45 cm, manteniendo un diseño compacto de los elementos y módulos. Como se mencionó anteriormente, el led indicador y el sensor adicional ahora se encuentran en la parte superior, intercambiando posición con el Z-Uno.

Figura 33

Modelo 3D de la carcasa del Nodo 7 ESP32 implementado con Z-Wave

Figura 34

Nodo 7 completo con la carcasa impresa y su PCB.



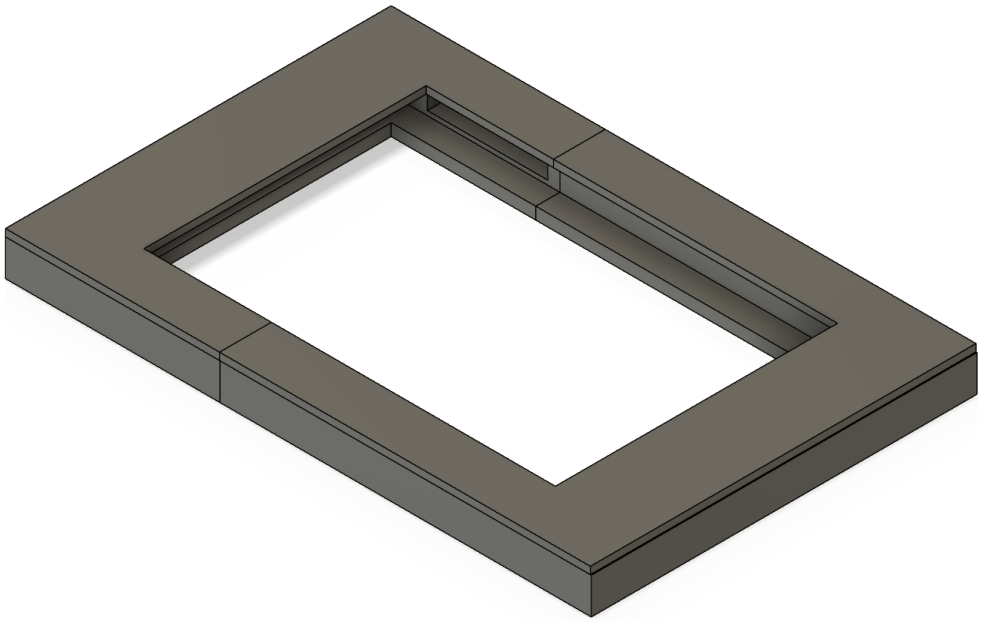
Se puede observar en la Figura 34 que a pesar de la colocación de un microcontrolador adicional, se logró mantener un tamaño adecuado como en el resto de nodos.

5.4.8. Monitor de Interfaz de Usuario

Como se mencionó anteriormente, la información es visualizable a través de cualquier dispositivo con acceso a la misma red que la Raspberry, sin embargo, se utilizó un monitor exclusivo para Home Assistant utilizando una tablet NEXTBOOK NXW8QC16G de 9 pulgadas con un soporte diseñado mediante modelado 3D mostrado en la Figura 35.

Figura 35

Modelo 3D del soporte de pared para el monitor de usuario.



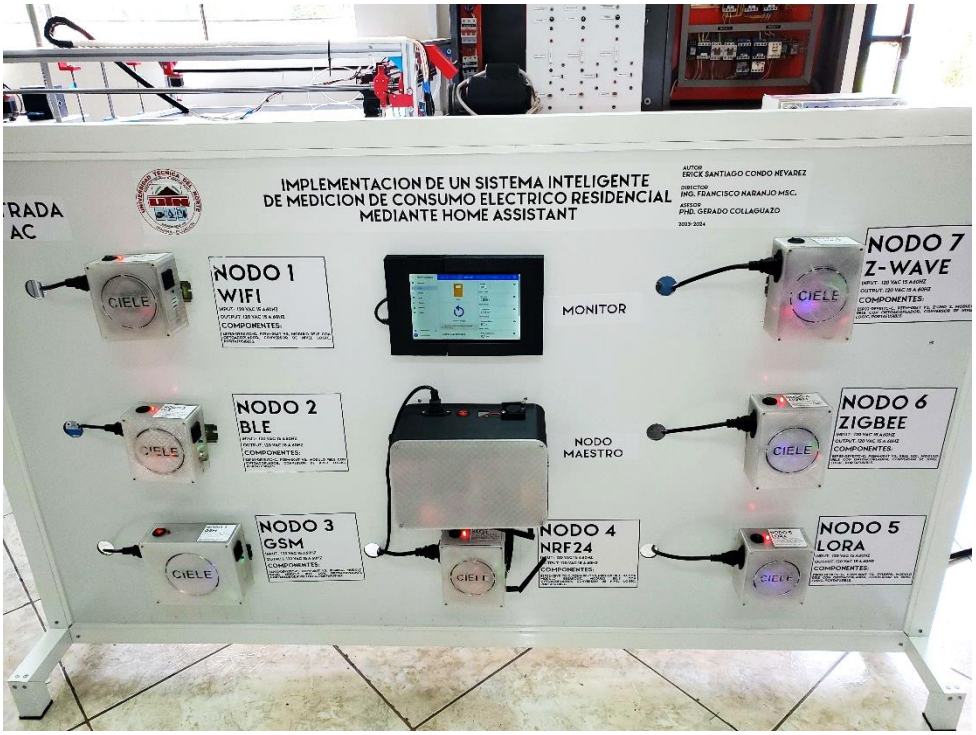
A través de este dispositivo, podemos visualizar las mediciones y controlar los relés de cada módulo, pero también es posible conectarse en modo administrador, visualizar extensiones, complementos y editar las configuraciones de Home Assistant ya que la tablet ofrece un navegador completo al utilizar Windows 8.1 como sistema operativo.

5.4.9. Tablero para exposición del Sistema

Se diseñó y construyó un tablero que sujeta todos los componentes del sistema, en donde se incluyeron los nodos, tanto individuales como el maestro, el monitor, router, tomacorrientes para la alimentación de los nodos, y una protección termomagnética.

Figura 36

Cara frontal del tablero de exposición del Sistema



El tablero que se observa en la Figura 36 está conformado de una base de alocubond de 1,6 m x 0,9 m y 3 mm de grosor, mientras que el marco que lo sostiene es de aluminio. Se utilizó los modelos 3D de los nodos y del monitor para la proyección del diseño el tablero usando Autodesk Fusion 360, para así mantener el espacio adecuado para el resto de elementos colocados posteriormente.

Figura 37

Cara trasera del tablero de exposición del Sistema

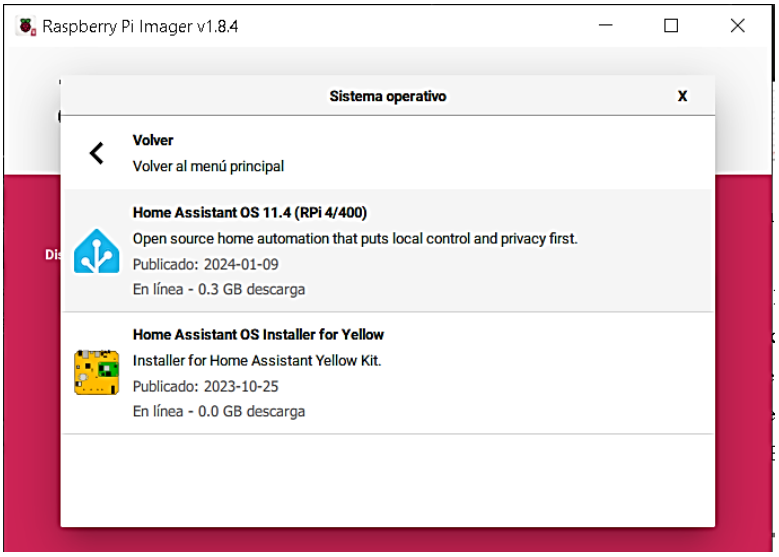
Como se observa en la Figura 37, se utilizaron tomacorrientes dobles para enchufar los dispositivos, conectados en paralelo y protegidos por tubos conduit de metal flexible con recubrimiento plástico. Se añadió una caja de paso metálica detrás del Nodo Maestro para mantener un mejor orden en el cableado de los componentes que lo contienen. Para proteger la instalación, se añadió un termomagnético de 1 polo de 32A dentro de una caja térmica. También colocó un router, encargado de crear la red en la que se hospeda Home Assistant y desde donde es posible conectarse a través de Wi-Fi. Finalmente, se agregaron los rótulos de etiquetado de cada elemento del tablero.

5.5. Instalación de Home Assistant

Para la instalación de Home Assistant en la Raspberrpi Pi 4B, se necesitó instalar la imagen del sistema operativo en una tarjeta microSD de Clase 10 por recomendación de Home Assistant, esto con el fin de obtener las mejores velocidades de escritura y lectura posibles. Se utilizó un software otorgado por Raspberry Pi para la instalación de imágenes en sus dispositivos llamado “Raspberry Pi Imager”.

Figura 38

Selección de Home Assistant OS en Raspberry Pi Imager.



Nota. Elaboración Propia.

La versión utilizada para Home Assistant OS fue la 11.4, sin embargo hay que tomar en cuenta que es posible mantener el sistema operativo actualizándose constantemente desde la interfaz de usuario de Home Assistant.

Antes de encenderse la Raspberry, se ingresó la microSD con el sistema operativo en él y se conectó el cable Ethernet en el puerto correspondiente. Luego, se esperó 5 minutos aproximadamente mientras se inicializa Home Assistant.

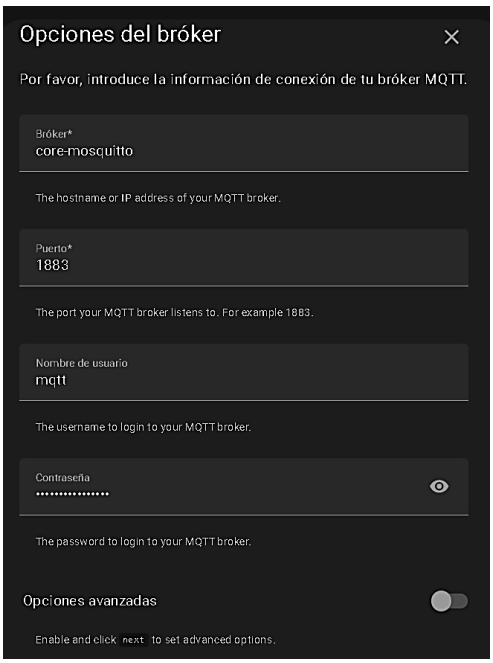
Después, desde un dispositivo conectado a la misma red, se ingresó a la interfaz mediante http://homeassistant:8123/, donde se empezó la configuración general de usuario. Si conectamos un monitor mediante el HDMI de la Raspberry, podremos observa la consola de salida, donde veremos el proceso de instalación, como las IPs desde las cuales es posible ingresar a Home Assistant. Luego de la configuración, ya fue posible utilizar todo el sistema operativo y las funcionalidades de Home Assistant.

5.6. Instalación del Nodo 1 Wi-Fi

Para la instalación del Nodo 1, se empezó incluyendo la integración "MQTT" en Home Assistant y se realizó la configuración inicial como se muestra en la Figura 39.

Figura 39

Configuración de MQTT en Home Assistant.

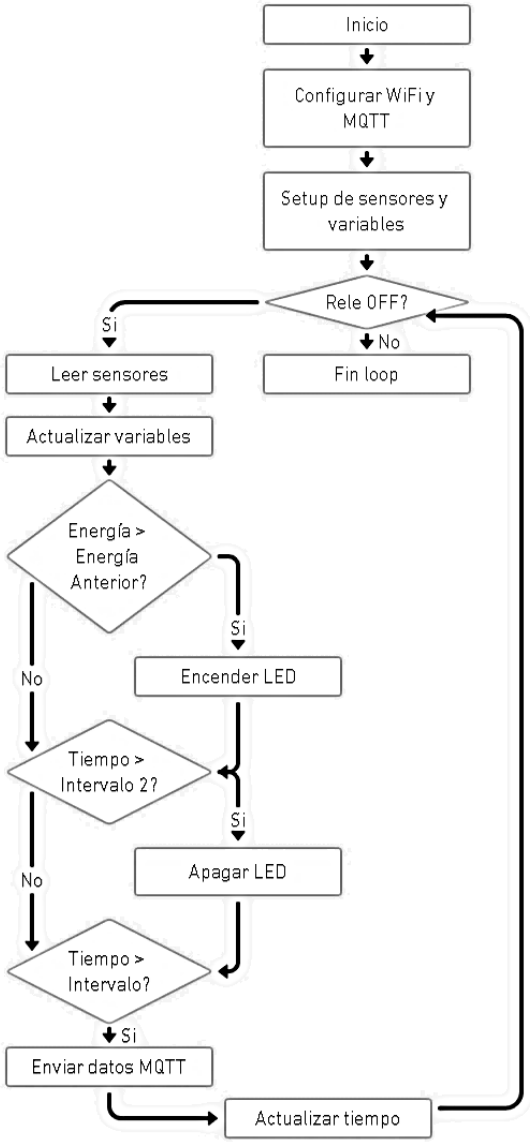


Al momento de programar el ESP32 en Arduino IDE, se ingresó el SSID y contraseña del Wi-Fi de la misma red en la que está conectado el Home Assistant, como también, el host y el port del MQTT. Luego, se establecieron los "Temas" de MQTT correspondientes a las

variables que se enviarían al Nodo Maestro. Las funciones cambiarán dependiendo de la librería utilizada para la comunicación MQTT, en este caso se usó AsynMqttClient.h.

Figura 40

Diagrama de flujo del código de Arduino IDE para el Nodo 1.



Nota. Código de programación en Anexos.

Como se observa en la Figura 32, el relé actúa dependiendo de la longitud del mensaje que se recibe a través de MQTT. Esto debido a que este método resultó ser más sencillo que esperar un caracter específico.

En Home Assistant se añadieron estas líneas de código en el archivo configuration.yaml, con el fin de enviar las órdenes al ESP32 a través de Mosquitto MQTT. En este archivo se definen los sensores en base a los datos MQTT recibidos, además del icono y la unidad del valor medido. También, define el interruptor para el control del relé y un botón para el reinicio de la energía medida por el PZEM-004t V3.

configuration.yaml

Configuración de variables de sensor y switch MQTT.

```
mqtt:
  - sensor:
      name: "ESP1 Voltaje"
      state_topic: "esp32-wifi/PZEM/Voltaje"
      icon: mdi:sine-wave
      unit_of_measurement: "V"

  - sensor:
      name: "ESP1 Corriente"
      state_topic: "esp32-wifi/PZEM/Corriente"
      icon: mdi:current-ac
      unit_of_measurement: "A"

  - sensor:
      name: "ESP1 Frecuencia"
      state_topic: "esp32-wifi/PZEM/Frecuencia"
      icon: mdi:current-ac
      unit_of_measurement: "Hz"

  - sensor:
      name: "ESP1 Factor de Potencia"
      state_topic: "esp32-wifi/PZEM/FactorDePotencia"
      icon: mdi:current-ac

  - sensor:
      name: "ESP1 Energia"
      state_topic: "esp32-wifi/PZEM/Energia"
      icon: mdi:lightning-bolt
      unit_of_measurement: "kWh"

  - sensor:
      name: "ESP1 Potencia"
      state_topic: "esp32-wifi/PZEM/Potencia"
      icon: mdi:flash
      unit_of_measurement: "W"
```

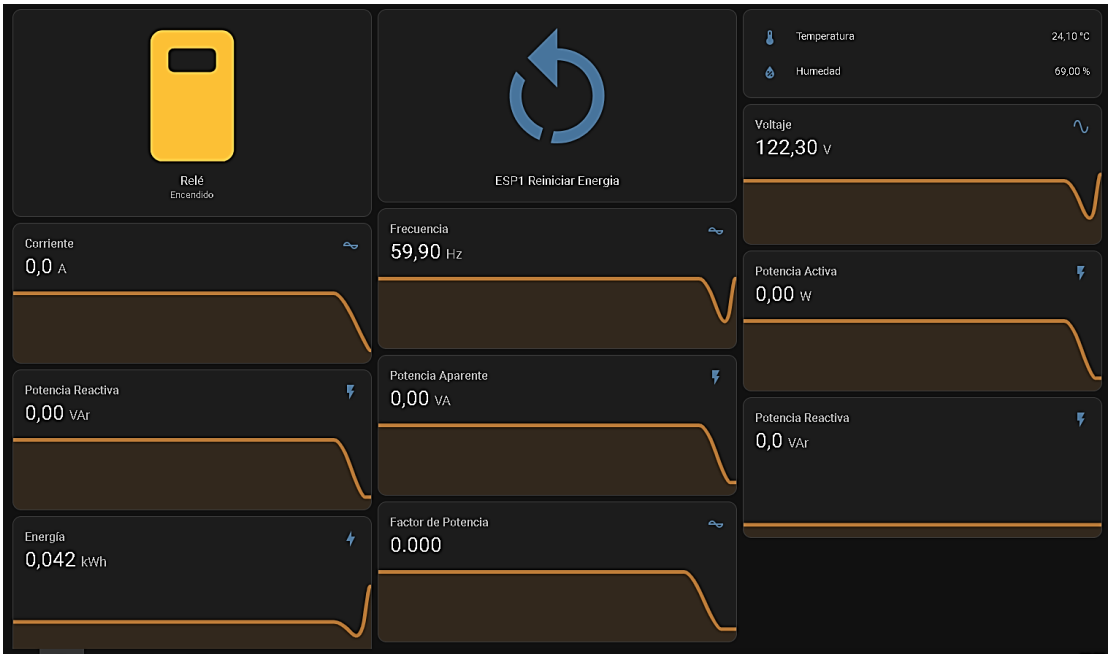
```
- sensor:
    name: "ESP1 Humedad"
    state_topic: "esp32-wifi/DHT11/Humedad"
    icon: mdi:water-percent
    unit_of_measurement: "%"

- sensor:
    name: "ESP1 Temperatura"
    state_topic: "esp32-wifi/DHT11/Temperatura"
    icon: mdi:thermometer
    unit_of_measurement: "°C"

- switch:
    unique_id: esp1_rele
    name: "ESP1 Rele"
    command_topic: "esp32-wifi/control"
    payload_on: "11"
    payload_off: "1"
    optimistic: true
    qos: 0
    retain: 1
    enabled_by_default: 1

- button:
    unique_id: esp1_reiniciarEnergia
    name: "ESP1 Reiniciar Energia"
    command_topic: "esp32-wifi/control"
    payload_press: "111"
    qos: 0
    retain: false
    entity_category: "config"
    device_class: "restart"
```

Nota. Código completo en Anexos.

Luego de obtener los sensores y botones establecidos, se colocaron en un panel llamado "Nodo 1" en la interfaz de usuario de Home Assistant. Además, se calculó la Potencia Reactiva y Aparente, utilizando las variables medidas por el nodo.

Figura 41

Interfaz de control y lectura de mediciones del Nodo 1.



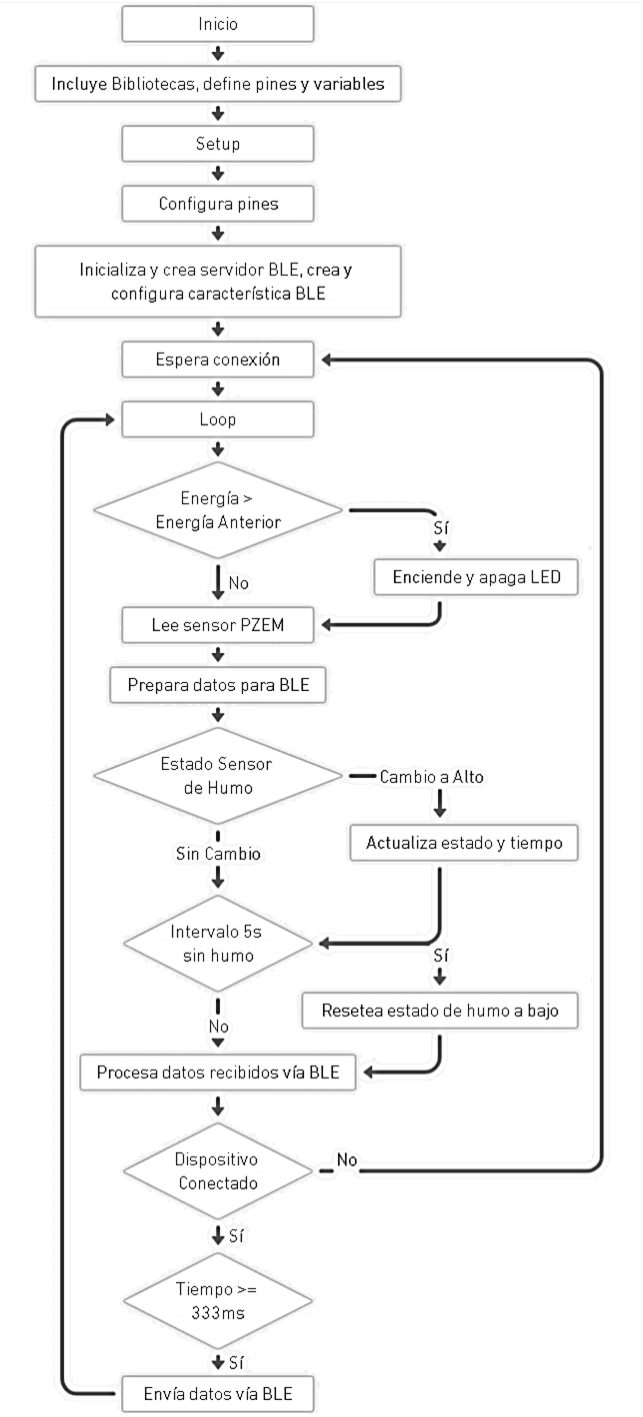
5.7. Instalación del Nodo 2 BLE

Para la instalación del Nodo 2, se utilizó un ESP32 como puerta de enlace BLE, dándonos la posibilidad de aumentar el rango de la señal con una antena y quitarle carga a la Raspberry.

Inicialmente, se escribió el código correspondiente al Nodo 2, utilizando las librerías BLE que se encuentran de forma nativa en la integración ESP32 de Arduino IDE. Se establecieron los pines de entrada y salida de las variables y los identificadores BLE, los cuales son los mismos que se ingresaron en el código del Gateway.

Figura 42

Diagrama de flujo del código de Arduino IDE para el Nodo 2.

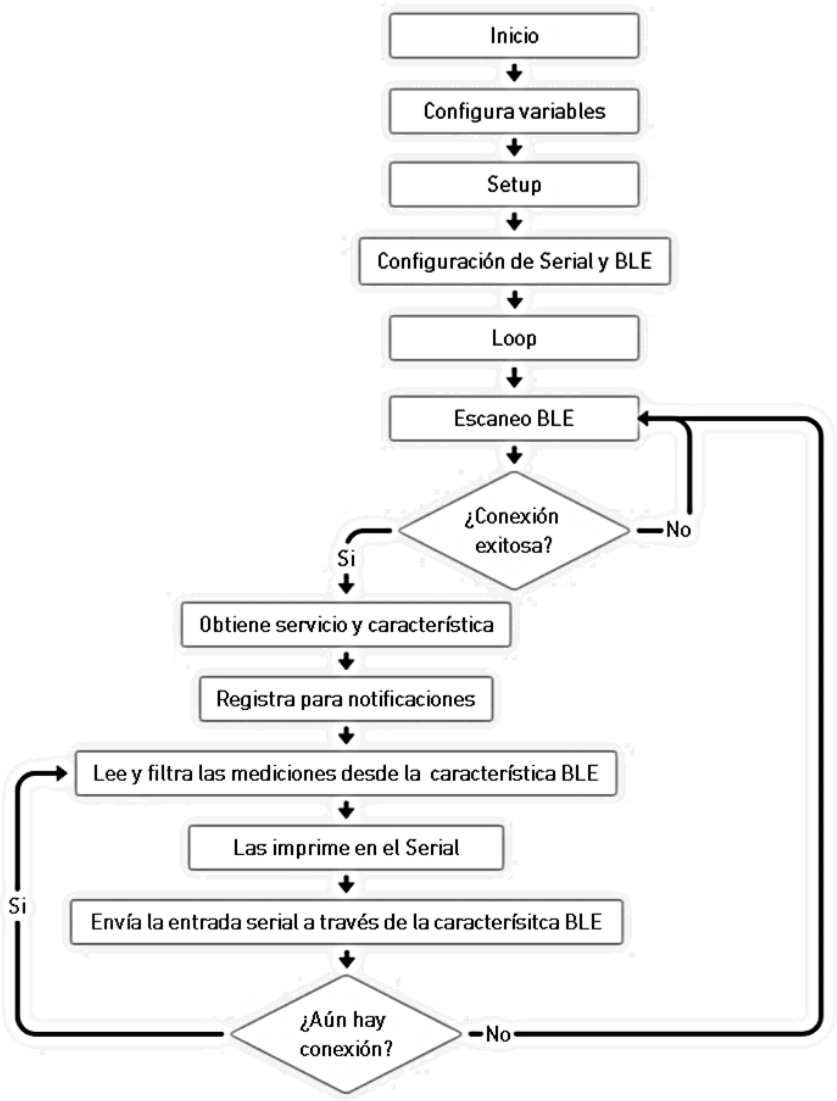


Nota. Código completo en Anexos.

Por otro lado, en el código correspondiente al Gateway, se usaron las mismas librerías, sin embargo, se añadió código encargado de transcribir la información otorgada por serial mediante el Nodo 2.

Figura 43

Diagrama de flujo del código de Arduino IDE para el Gateway del Nodo 2.



Nota. Código completo en Anexos.

Para la lectura del puerto serial del Gateway BLE, fue necesario establecer dicho dispositivo como sensor, espescificando el baudrate y asignándole un nombre. Luego, se obtuvieron las mediciones separadas utilizando la posición de las comas en el string. En el código a continuación, se muestra un ejemplo con la medición de Voltaje.

configuration.yaml

Configuración de entrada serial y declaración de sensor.

```
sensor:
  - platform: serial
    name: Serial1
    serial_port: /dev/ttyUSB0
    baudrate: 115200

  - platform: template
    sensors:
      voltage2:
        friendly_name: "ESP2 Voltaje"
        unique_id: "esp2_voltaje"
        unit_of_measurement: "V"
        value_template: "{{ (states('sensor.serial1').split(',')[1] |
float(default=0)|round(3)) }}"
```

Nota. Código completo en Anexos.

El control del relé fue realizado mediante comandos enviados al puerto USB a través de la consola de Home Assistant. El hexadecimal enviado por la Raspberry fue el que esperaba recibir el ESP32 para encender o apagar el relé.

configuration.yaml

Configuración de comandos de consola y declaración del switch.

```
shell_command:
  esp2_rele_off: /bin/bash -c "echo -ne '\x01\xAB\xFF' > /dev/ttyUSB0"
  esp2_rele_on: /bin/bash -c "echo -ne '\x02\xAB\xFF' > /dev/ttyUSB0"

  esp2_reiniciar_energia: /bin/bash -c "echo -ne '\x05\xAB\xFF' > /dev/ttyUSB0"

switch:
  - platform: template
    switches:
      esp2_rele:
        turn_on:
          service: shell_command.esp2_rele_on
```

```
13.         turn_off:
14.           service: shell_command.esp2_rele_off
```

Nota. Código completo en Anexos.

5.8. Instalación del Nodo 3 GSM

Para empezar la instalación del nodo 3, fue necesario adquirir 2 SIMs prepago y luego registrarlas. Después, se conectó el USB Stick del SIM800C al USB Hub del nodo maestro. Para una fácil implementación, se utilizó la integración "SMS notifications via GSM-modem" de Home Assistant, donde fue necesario identificar el módem GSM y el baudrate de comunicación. Después, la integración identificará el fabricante y modelo del módulo.

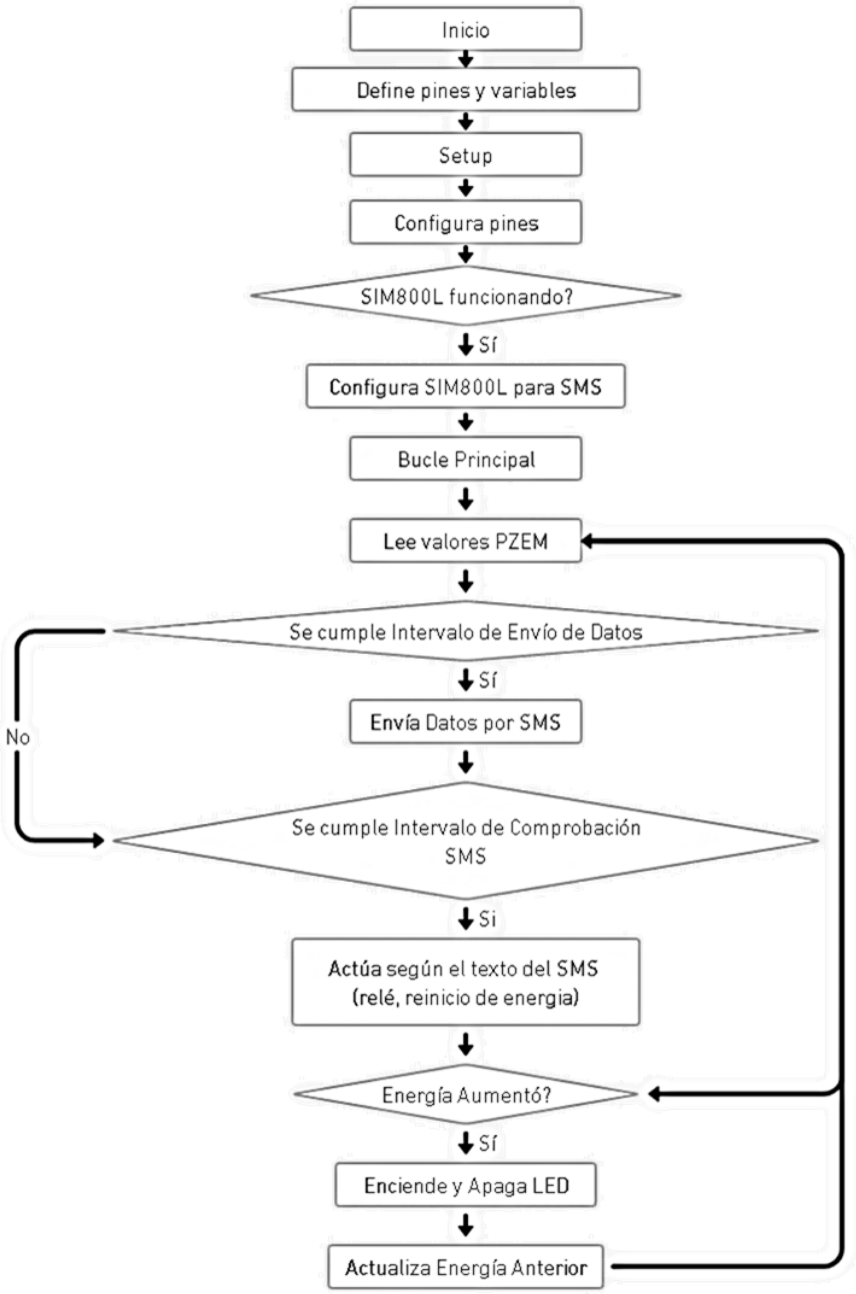
Figura 44

Configuración de la integración SMS notifications via GSM-modem.

Luego, fue necesario programar el ESP32 con el fin de que envíe y reciba SMS utilizando el SIM800L utilizando comandos AT. Para esto, se escribió el siguiente código en Ardunino IDE.

Figura 45

Diagrama de Flujo del código de Arduino IDE para el Nodo 3.



En este Nodo, fue necesario programar dos entradas UART. El módulo GSM se conectó en los pines TX1 y RX2 y el módulo de medición PZEM-004t V3 se conectó en los pines D2

y D4. Además de las variables eléctricas, fue necesario ingresar el número celular del otro dispositivo para el envío de SMS.

En Home Assistant, se realizaron los scripts correspondientes a la recepción y envío de mensajes. Para el envío de SMS, se utilizó el servicio notify.sms para enviar los caracteres que espera el ESP32 para el encendido y apagado del relé, y el reinicio de la energía en el módulo.

automations.yaml

Declaración de las automatizaciones para el envío de mensajes

```
- id: '1705419499042'
  alias: Encender Rele Nodo 3
  description: ''
  trigger:
  - platform: state
    entity_id:
    - input_boolean.nodo3switch
    from: 'off'
    to: 'on'
  condition: []
  action:
  - service: notify.sms
    data:
      message: '@1#'
      target: '+593994796445'
  mode: single

- id: '1705419538882'
  alias: Apagar Rele Nodo 3
  description: ''
  trigger:
  - platform: state
    entity_id:
    - input_boolean.nodo3switch
    from: 'on'
    to: 'off'
  condition: []
  action:
  - service: notify.sms
    data:
      message: '@0#'
      target: '+593994796445'
  mode: single

- id: '1705419981835'
  alias: resetear energia Nodo 3
  description: ''
  trigger:
```

```
39.    - platform: state
40.      entity_id:
41.      - input_button.reset_energia_nodo_3
42.    condition: []
43.    action:
44.    - service: notify.sms
45.      data:
46.        message: '@3#'
47.        target: '+593994796445'
48.    mode: single
49.
```

Nota. Código completo en Anexos.

Para la recepción, la integración utiliza el evento sms.incoming_sms, por lo cual, fue necesario declarar una variable que almacene el mensaje recibido cuando llega un mensaje, por lo cual, fue necesario realizarse a través de automatizaciones, de la siguiente manera.

automations.yaml

Declaración de las automatizaciones para la recepción de mensajes.

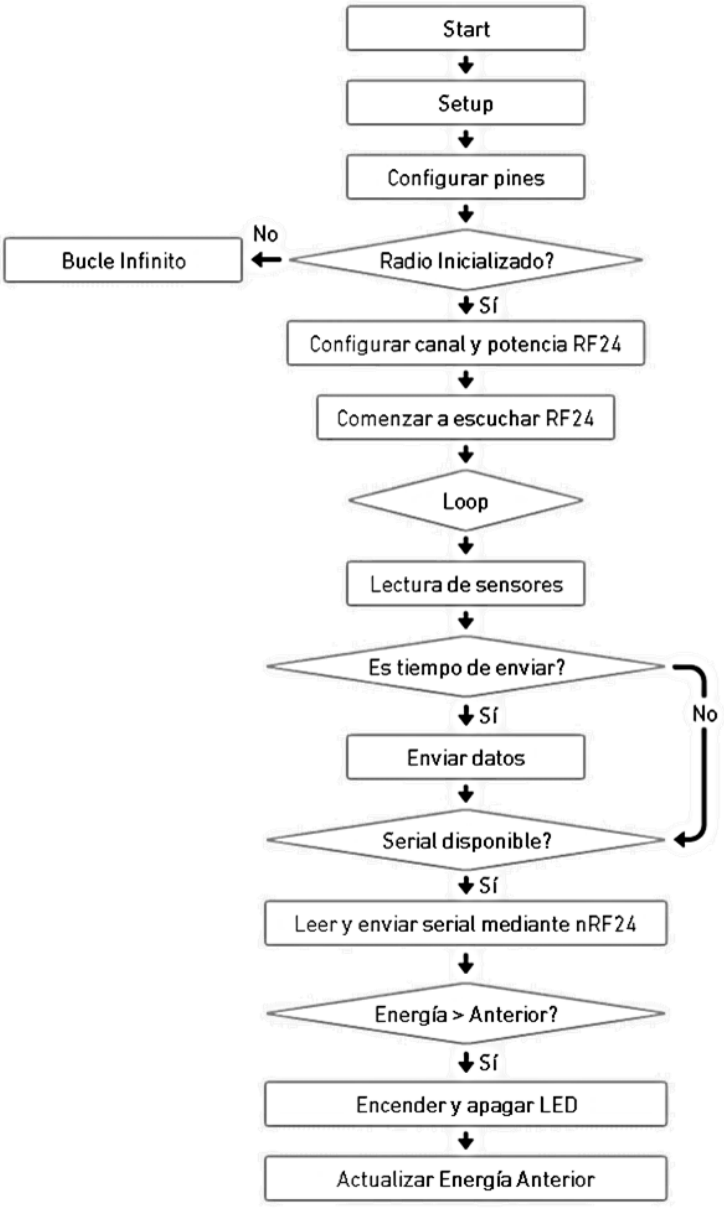
```
 1. - alias: Almacenar SMS
 2.   trigger:
 3.   - platform: event
 4.     event_type: sms.incoming_sms
 5.   action:
 6.   - service: input_text.set_value
 7.     data_template:
 8.       entity_id: input_text.serial_history
 9.       value: '{{ trigger.event.data.text }}'
10.   id: af497b3cb6e94c93ab4a197ae3bcf8dd
```

5.9. Instalación del Nodo 4 nRF24L01

Para la programación de los módulos nRF24L01 PA+LNA, se usaron las librerías RF24.h y SPI.h, ya que este módulo utiliza comunicación de este tipo. Como novedades del código, se diferenciaron ambos nodos con “nodo 1” y “nodo 2”, y fueron asignados a distinto canal para establecer la comunicación correctamente.

Figura 46

Diagrama de flujo del código de Arduino IDE para el Nodo 4.

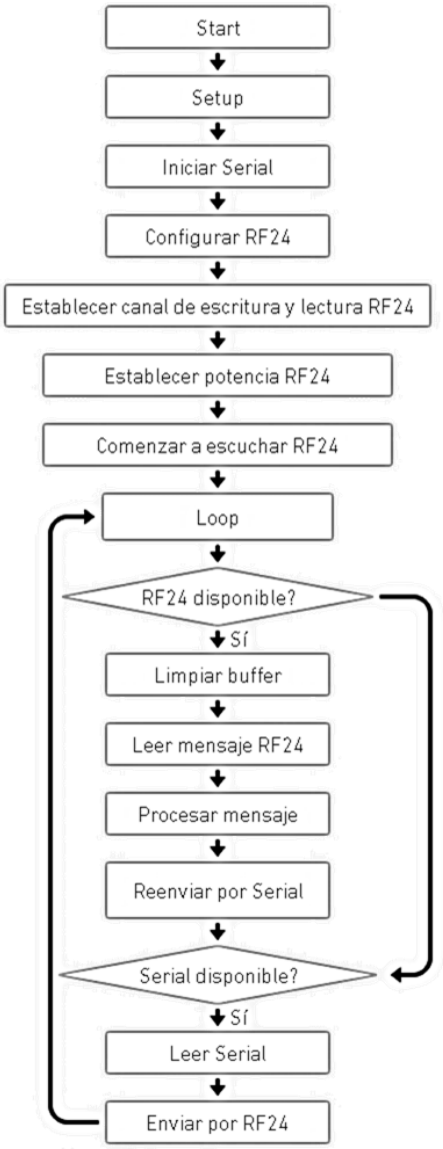


Nota. Código completo en Anexos.

Puesto que se usó el Gateway del Nodo 2 BLE para recibir los datos a través del otro nRF24L01 PA+LNA, se utilizó el mismo código para la recepción, sin embargo, se cambió la variable del radio a 0, para establecerlo como un canal distinto. La transcripción y el filtrado de los datos del serial se la realizó de la misma manera del nodo 2, se utilizó otra combinación de caracteres para obtener las variables requeridas.

Figura 47

Diagrama de flujo del código de Arduino IDE para el Gateway del Nodo 4.



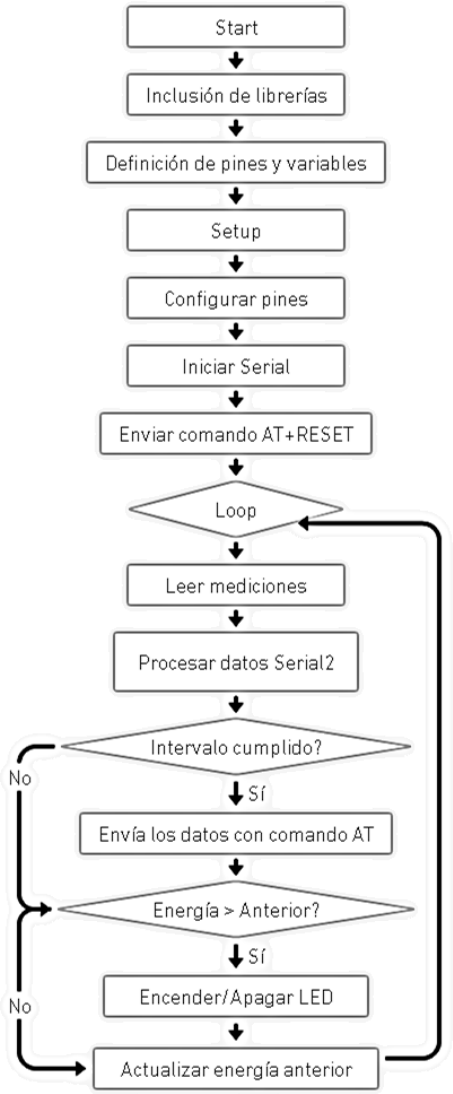
Nota. Código completo en Anexos.

5.10. Instalación del Nodo 5 LoRa

Los dispositivos transceptores LoRa en este caso son los RYRL998, los cuales trabajan a través de comunicación UART de forma sencilla. LoRa se integrará a Home Assistant utilizando el módulo conectado mediante un módulo conversor TTL a USB y leyendo el serial para acceder a la información enviada a través del ESP32.

Figura 48

Diagrama de flujo del código de Arduino IDE para el Nodo 5.

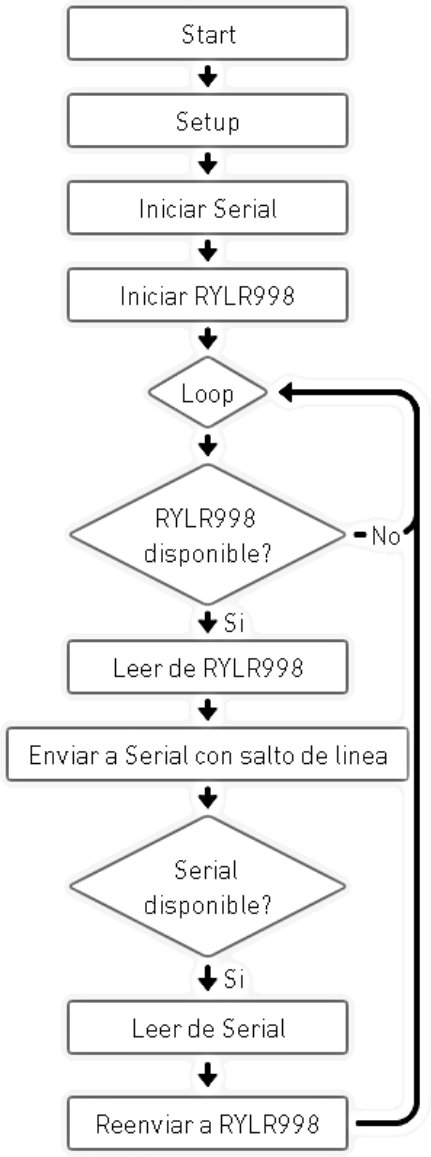


Nota. Código completo en Anexos.

Puesto que la comunicación con el RYLR998 se realiza a través de UART, no se requirieron librerías adicionales, por lo que se trabajó directamente con comandos AT para el envío y recepción de los datos.

Figura 49

Diagrama de flujo del código de Arduino IDE para el Gateway del Nodo 5.



Nota. Código completo en Anexos.

Para recibir la entrada de esta información a través de Serial, fue necesario captar el puerto USB donde se encuentra conectado el RYLR998 junto al conversor TTL a USB, por lo cual se incluyó este código a la configuración de Home Assistant

configuration.yaml

Definición del puerto USB del módulo RYLR998 como sensor en Home Assistant.

```
1. sensor:
2.   - platform: serial
3.     name: Serial2
4.     serial_port: /dev/ttyUSB0
5.     baudrate: 115200
6.
```

Luego, se realizó el filtrado de los datos basándonos en el formato con el que el módulo recibe los mensajes entrantes. RYLR998 los recibe con la estructura +RCV=<Address>, <Lengh>, <Data>, <RSSI>, <SNR>. Finalmente, se filtraron las variables medidas de la misma manera que el Nodo 2, 3 y 4.

5.11. Instalación del Nodo 6 (Zigbee)

En Home Assistant, se utilize como gateway el módulo Xbee S2C con el adaptador Xbee Explorer, luego se añadió la configuración el puerto con el que se identifica el dispositivo.

configuration.yaml

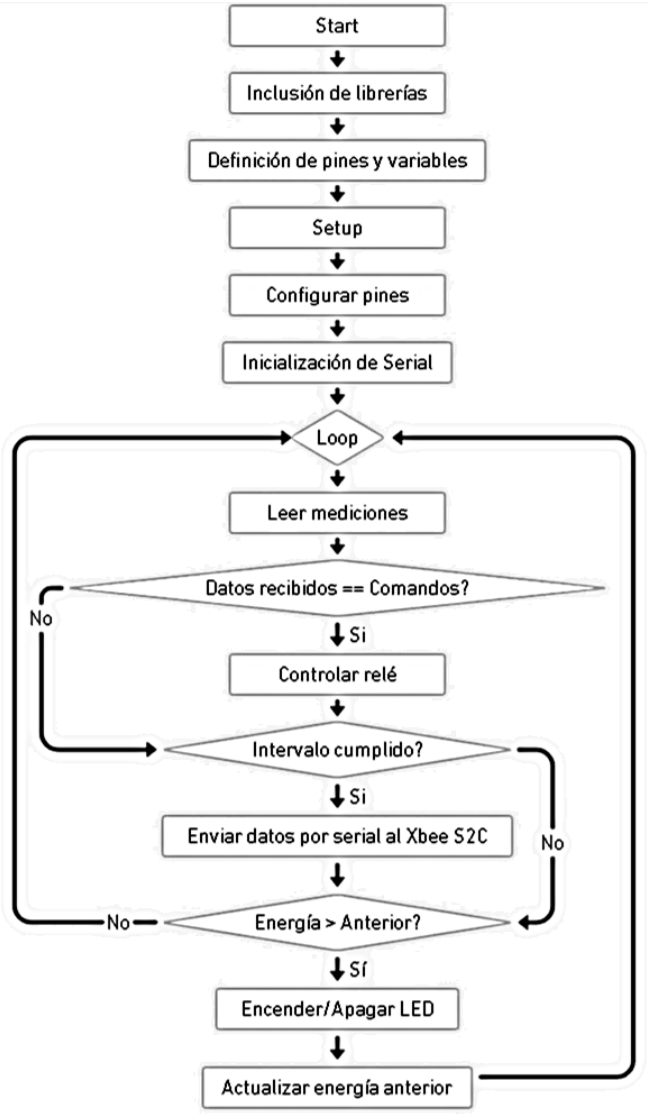
```
1.   - platform: serial
2.     name: Nodo6Zigbee
3.     serial_port: /dev/serial/by-path/platform-fd500000.pcie-pci-0000:01:00.0-usb-
0:1.1.1.4:1.0-port0
4.     baudrate: 9600
5.
```

Se utilizó un baudrate de 9600 ya que se comprobó que la entrada serial era más estable de esta forma. Además, se agregaron los sensores plantilla utilizando el sensor serial previamente asignado.

En Arduino IDE se escribió el código considerando que la comunicación con el módulo se la realiza a través de comandos AT.

Figura 50

Configuración de la red Zigbee en Home Assistant.



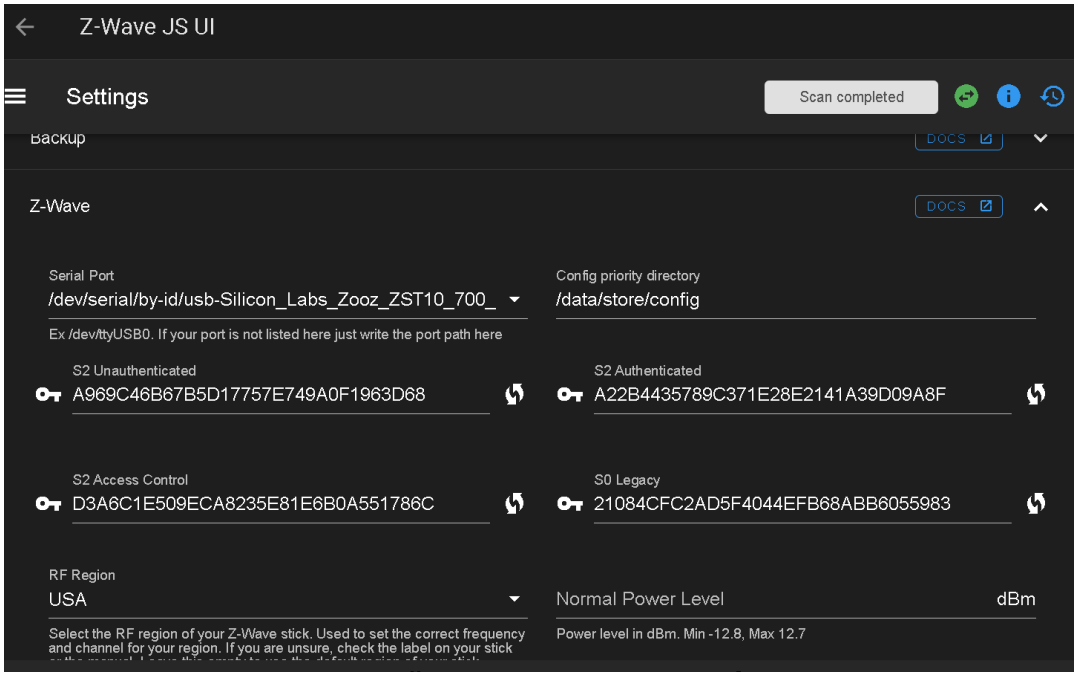
Nota. Código completo en Anexos.

5.12. Instalación del Nodo 7 (Z-Wave)

Se utilizó un complemento llamado Z-Wave JS UI como controlador de la red Z-Wave. En donde, luego de conectar el USB Stick, se seleccionó el puerto y automáticamente se sincronizaron los datos de la red.

Figura 51

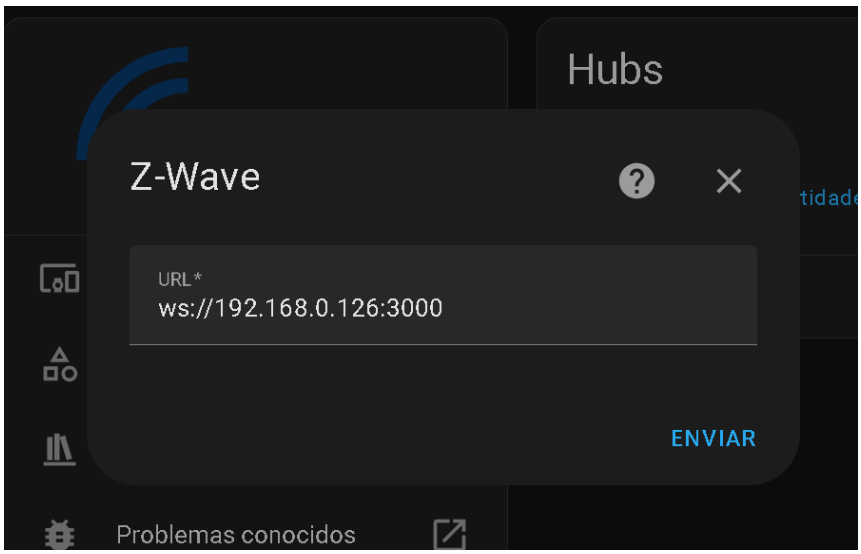
Configuración del complemento Z-Wave JS UI.



Se especificó el puerto 3000 que usaría la red, en la configuración del complemento desde Home Assistant. Luego, se añadió la red de Z-Wave JS UI a la integración Z-Wave de Home Assistant. De esta manera, las entidades que tendríamos en ZL-Wave JS UI se sincronizaría en Home Assistant, para controlar los actuadores y leer los sensores a través de la interfaz.

Figura 52

Sincronización de Z-Wave JS UI con Z-Wave de HA.



Con Z-Wave configurado desde Home Assistant, se procedió a la programación de los microcontroladores. Empezaríamos con ESP32, pero enviando las mediciones a través de serial.

Nodo_7_ESP32.ino

Declaración de librerías y variables del Nodo 7.

```
#include <PZEM004Tv30.h>

#if !defined(PZEM_RX_PIN) && !defined(PZEM_TX_PIN)
#endif

#if !defined(PZEM_SERIAL)
#define PZEM_SERIAL Serial1
#endif

#if defined(ESP32)

#define PZEM_RX_PIN 4
#define PZEM_TX_PIN 2
```

```
14.
15. PZEM004Tv30 pzem(PZEM_SERIAL, PZEM_RX_PIN, PZEM_TX_PIN);
16. #elif defined(ESP8266)
17. PZEM004Tv30 pzem(PZEM_SERIAL);
18. #endif
19.
20. #define RXp2 16
21. #define TXp2 17
22.
23. int micro  = 15;
24.
25. const int rele = 13;
26. const int estadorele = 12;
27. const int resetenergia = 14;
28.
29. int led = 26;
30.
31. float voltage = 0.0;
32. float current = 0.0;
33. float power = 0.0;
34. float energy = 0.0;
35. float frequency = 0.0;
36. float pf = 0.0;
37.
38. float energiaAnterior = 0;
39.
40. unsigned long previousMillis = 0;
41. const long interval = 1000;
42.
```

Luego se procedió a escribir el código correspondiente al Z-Uno, donde además de leer y filtrar las variables de medición eléctrica, se utilizaron funciones de las librerías del microcontrolador para el envío de datos a través de Z-Wave.

Nodo_7_ZUNO.ino

Declaración de librerías y variables del Nodo 7 en Z-Uno.

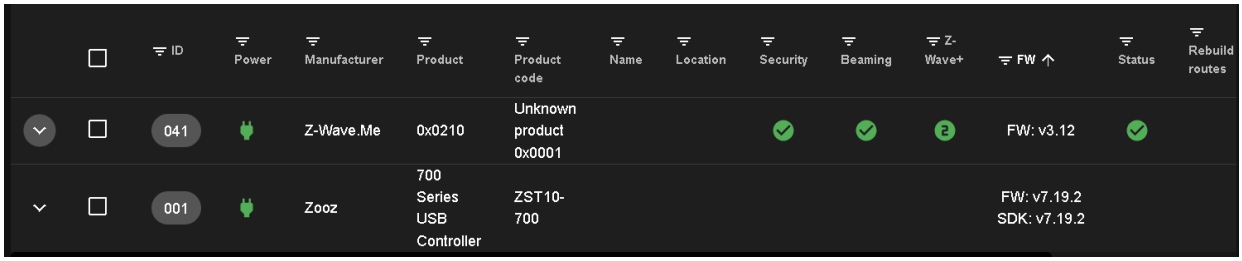
```
 1. #define RELE    13
 2. #define RESETENERGIA    14
 3.
 4. byte lastSetValue;
 5. byte lastSetValue2;
 6.
 7. unsigned long int voltage = 0.0;
 8. unsigned long int current = 0.0;
 9. unsigned long int power = 0.0;
10. unsigned long int energy = 0.0;
11. unsigned long int frequency = 0.0;
12. unsigned long int pf = 0.0;
13. unsigned long int micro = 0.0;
14.
```

```
15. unsigned long previousMillis = 0;
16. const long interval = 1000;
17.
18. // set up channels
19. ZUNO_SETUP_CHANNELS(
20.   ZUNO_SENSOR_MULTILEVEL(ZUNO_SENSOR_MULTILEVEL_TYPE_VOLTAGE,
21.                          SENSOR_MULTILEVEL_SCALE_VOLT,
22.                          SENSOR_MULTILEVEL_SIZE_FOUR_BYTES,
23.                          SENSOR_MULTILEVEL_PRECISION_ONE_DECIMAL,
24.                          getterVoltaje), // OK
25.
26.   ZUNO_SENSOR_MULTILEVEL(ZUNO_SENSOR_MULTILEVEL_TYPE_CURRENT,
27.                          SENSOR_MULTILEVEL_SCALE_AMPERE,
28.                          SENSOR_MULTILEVEL_SIZE_FOUR_BYTES,
29.                          SENSOR_MULTILEVEL_PRECISION_TWO_DECIMALS,
30.                          getterCorriente),  // OK
31.
32.   ZUNO_SENSOR_MULTILEVEL(ZUNO_SENSOR_MULTILEVEL_TYPE_POWER,
33.                          SENSOR_MULTILEVEL_SCALE_WATT,
34.                          SENSOR_MULTILEVEL_SIZE_FOUR_BYTES,
35.                          SENSOR_MULTILEVEL_PRECISION_TWO_DECIMALS,
36.                          getterPotencia), // OK
37.
38.   ZUNO_SENSOR_MULTILEVEL(ZUNO_SENSOR_MULTILEVEL_TYPE_GENERAL_PURPOSE_VALUE,
39.                          0,
40.                          SENSOR_MULTILEVEL_SIZE_FOUR_BYTES,
41.                          SENSOR_MULTILEVEL_PRECISION_TWO_DECIMALS,
42.                          getterEnergia), // OK
43.
44.   ZUNO_SENSOR_MULTILEVEL(ZUNO_SENSOR_MULTILEVEL_TYPE_FREQUENCY,
45.                          SENSOR_MULTILEVEL_SCALE_HERTZ,
46.                          SENSOR_MULTILEVEL_SIZE_FOUR_BYTES,
47.                          SENSOR_MULTILEVEL_PRECISION_ONE_DECIMAL,
48.                          getterFrecuencia),
49.
50.   ZUNO_SENSOR_MULTILEVEL(ZUNO_SENSOR_MULTILEVEL_TYPE_GENERAL_PURPOSE_VALUE,
51.                          0,
52.                          SENSOR_MULTILEVEL_SIZE_FOUR_BYTES,
53.                          SENSOR_MULTILEVEL_PRECISION_TWO_DECIMALS,
54.                          getterPF), // OK
55.
56.   ZUNO_SENSOR_MULTILEVEL(ZUNO_SENSOR_MULTILEVEL_TYPE_GENERAL_PURPOSE_VALUE,
57.                          0,
58.                          SENSOR_MULTILEVEL_SIZE_FOUR_BYTES,
59.                          SENSOR_MULTILEVEL_PRECISION_ZERO_DECIMALS,
60.                          getterMicro),
61.
62.   ZUNO_SWITCH_BINARY(getter, setter),
63.   ZUNO_SWITCH_BINARY(getter2, setter2)
64.
```

Después de subir el código a ambos microcontroladores, se necesitará emparejar el Z-Uno a la red Z-Wave presionando de forma rápida el botón SERVICE del microcontrolador. Finalmente, se podrá observar las mediciones correspondientes y los interruptores en la interfaz de Z-Wave JS UI y Home Assistant.

Figura 53

Conexión correcta del USB Stick de Z-Wave con el Z-Uno 2 del Nodo 7.



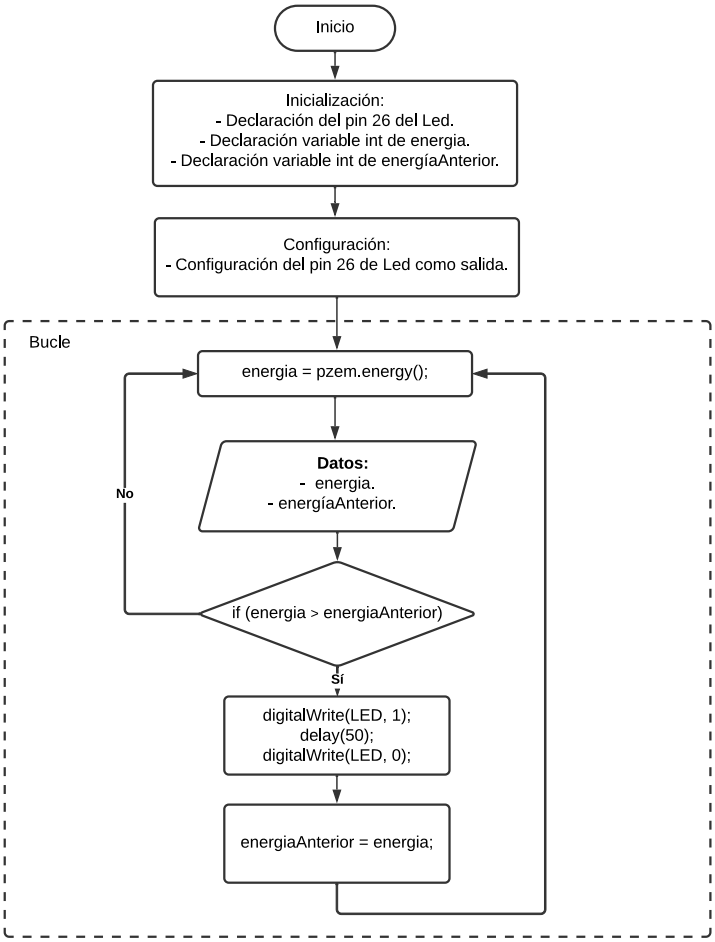
Ya con el complemento Z-Wave JS UI detectando el nodo, podremos agregar las mediciones y los controles a la interfaz de Home Assistant.

5.13. Funcionamiento del indicador Led de la constante del medidor

Por normativa, se representó la constante del medidor mediante un indicador led que parpadea cada 1 Wh (1000 imp/kWh). El diagrama de flujo de la Figura 14 representa el código de programación realizado en cada uno de los nodos. Este código fue utilizado en cada uno de los nodos, puesto que todos tienen un led de indicador de medidor.

Figura 54

Diagrama de flujo del indicador Led de la constante del medidor



Nota. Código completo en Anexos.

5.14. Costo de cada Nodo del sistema

Debido a los distintos componentes utilizados en cada uno de los nodos, existen diferencias en los costos principalmente relacionados con los requerimientos de los módulos transceptores. A continuación, se presenta una tabla donde se detallan los costos de elaboración de todos los dispositivos.

Tabla 21

Costo de cada Nodo del Sistema.

Nodo	Componente	Costo	Total
Nodo Maestro	Hardware	250	265
	PCB y carcasa	15	
Nodo 1	Hardware	20	30
	PCB y carcasa	10	
Nodo 2	Hardware	20	30
	PCB y carcasa	10	
Nodo 3	Hardware	30	40
	PCB y carcasa	10	
Nodo 4	Hardware	25	35
	PCB y carcasa	10	
Nodo 5	Hardware	30	40
	PCB y carcasa	10	
Nodo 6	Hardware	50	60
	PCB y carcasa	10	
Nodo 7	Hardware	70	80
	PCB y carcasa	10	

Como era predecible, el Nodo más costoso es el Maestro, puesto que contiene el Raspberry Pi 4B con el resto de los transceptores. Su costo es equivalente a lo que cuesta hoy en día un computador de sobremesa bastante sencillo. Por otro lado, los Nodos mas económicos son los Nodos 1 y 2, correspondientes a Wi-Fi y BLE debido que no fue necesario adicional un módulo transceptor adicional ya que el propio ESP32 integra esas tecnologías.

5.15. Pruebas de Precisión en las Mediciones Eléctricas

Se hicieron pruebas de precisión en las mediciones utilizando los equipos adecuados para cada magnitud. Para la energía, se utilizó el equipo Zera Meter Rack (mesa para la realización de pruebas a varios medidores a la vez) junto con el comparador COM3003 y el generador FG301 de la misma marca, equipos utilizados para la calibración de medidores eléctricos en empresas de distribución eléctrica como Emelnorte.

Para las pruebas de precisión del resto de las magnitudes (voltaje, corriente, potencia, frecuencia y factor de potencia) se utilizó el analizador de red METREL POWERQ4 PLUS.

5.15.1. Pruebas de Precisión de la Energía con Mesa de Pruebas

Podemos observar en la Figura 39 el conexionado del Nodo con la mesa de pruebas, donde se utilizaron 3 cables; 2 para la fase de entrada y de salida, y 1 para el neutro. El led que marca las pulsaciones del Nodo 7 fue captado con un fotodiodo, encargado de medir el tiempo entre pulsación para entregar el error dependiendo el voltaje, corriente y factor de potencia entrante.

Figura 55

Nodo 7 conectado a la mesa de pruebas de medidores de Zera

Se hicieron 2 pruebas de error, la primera con un factor de potencia de 1 y la segunda con 0,5. En la Figura 40 podemos observar los parámetros ingresados en la interfaz del equipo para la prueba con un factor de potencia de 1, en donde finalmente obtenemos un error de 0,12%, como se observa en la hoja de resultados de la Figura 41. La señal utilizada para ambas pruebas fue de 120VAC 60Hz a 5A.

Figura 56

Parámetros para la prueba 1 con un factor de potencia de 1.

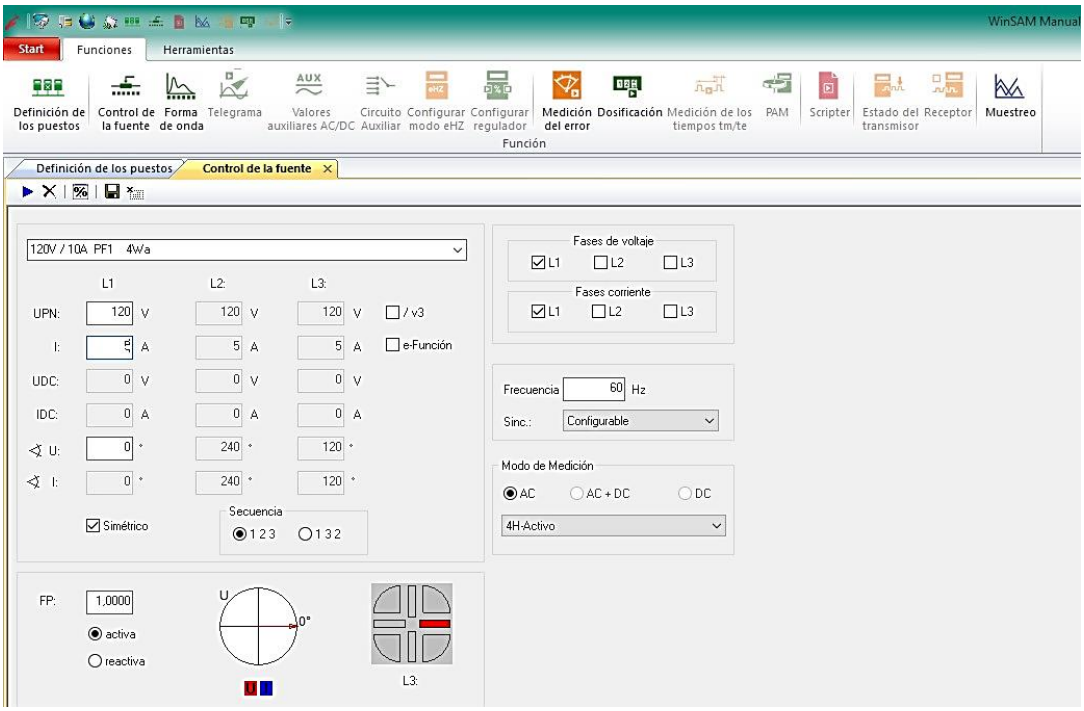


Figura 57

Hoja de resultados de la prueba 1 con un factor de potencia de 1.

Medición del error

MP	Cz	MT	error
1	1000 Imp/kWh	2	-0,12%
2		2	
3		2	
4		2	
5		2	
6		2	
7		2	
8		2	
9		2	
10		2	

A pesar de entregar un valor negativo, el error debe considerarse como un valor absoluto, por lo cual consideramos el error final de 0,12% para esa prueba.

Figura 58

Parámetros para la prueba 2 con un factor de potencia de 0,5.

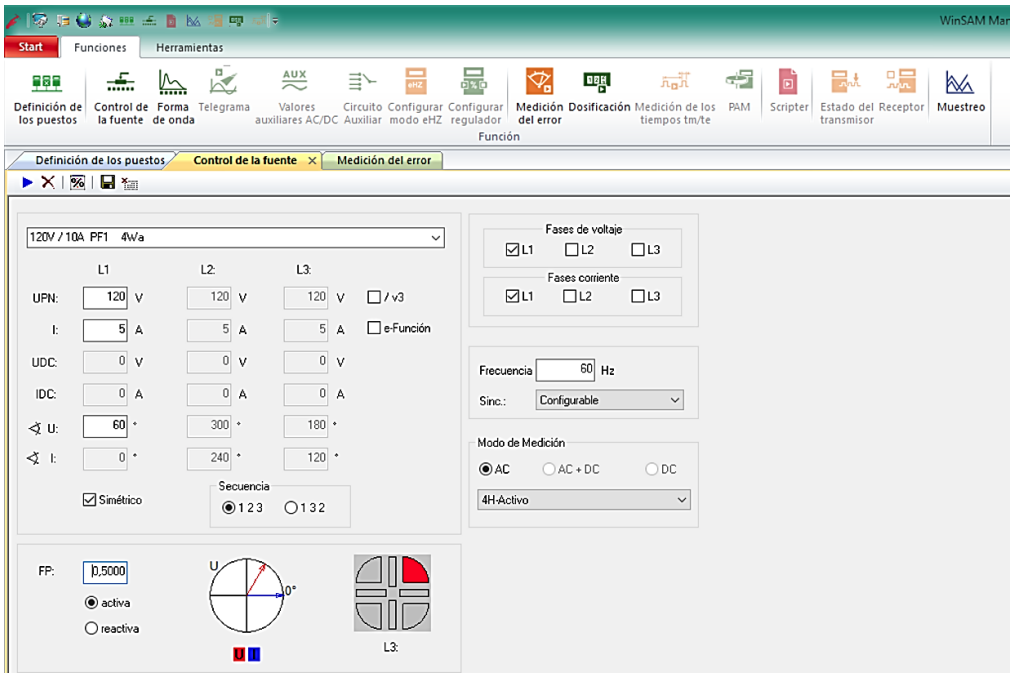


Figura 59

Hoja de resultados de la prueba 2 con un factor de potencia de 0,5.

Medición del error - Ejecutando medición del error

MP	Cz	MT	error
1	1000 Imp/kWh	1	4,51%
2		1	
3		1	
4		1	
5		1	
6		1	
7		1	
8		1	
9		1	
10		1	

En los resultados de la prueba observados en la Figura 43, usando los parámetros de la Figura 42, obtenemos un error de un 4.51%, valor considerablemente distante al 0.12% obtenido en la prueba 1 con un factor de potencia de 1.

3.8 Pruebas de Precisión de Energía con Medidor Residencial

Se utilizó un medidor analógico de la marca YUE QING BLUE SKY HI-TECH, calibrado y utilizado por EMELNORTE en distribución. Las mediciones empezaron el día 6 de febrero de 2024 cerca de las 5:30 pm. El medidor de Emelnorte empezó a contar desde 6114 kWh, mientras que el PZEM-004T V3 del Nodo 1 Wi-Fi empezó desde 0. El 7 de febrero del mismo año, a las 12:20 PM se detuvo el conteo, obteniendo una subida de aproximadamente 6 kWh (6,033 kWh en el PZEM-004t V3). En la siguiente Figura, observamos el historial de las mediciones realizadas por el PZEM-004T V3 almacenadas en Home Assistant.

Figura 60

Gráfico de Consumo Eléctrico en Home Assistant.

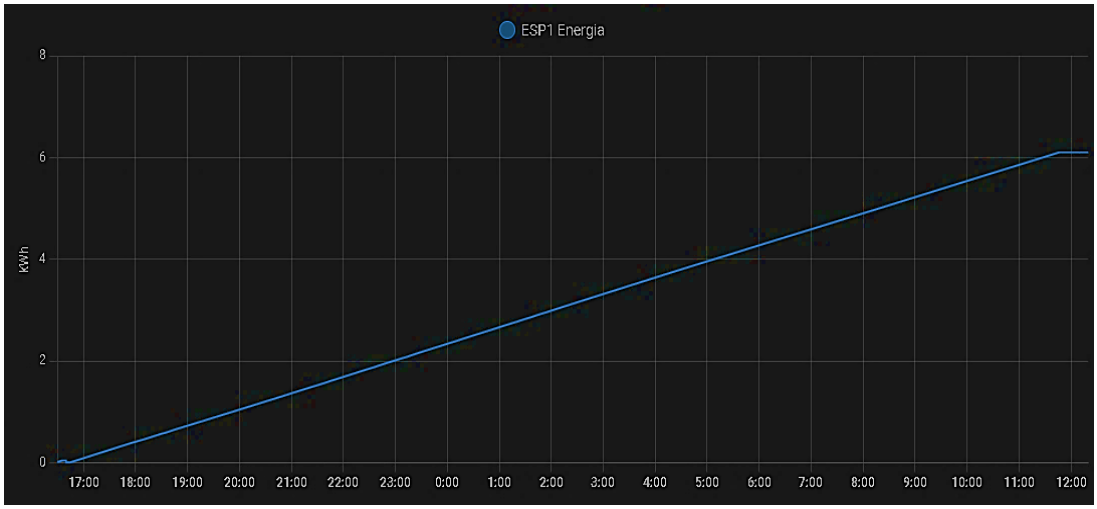
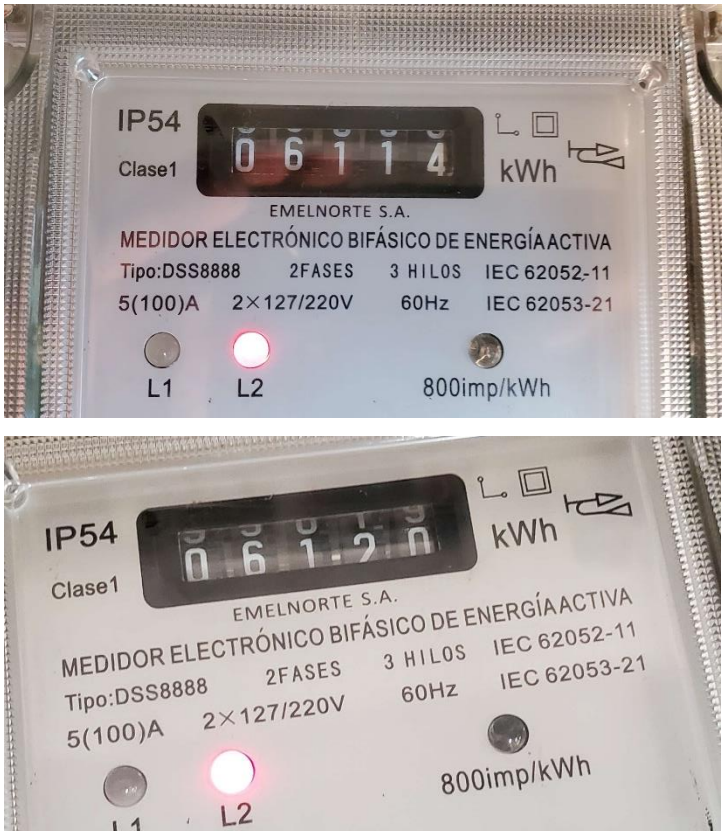


Figura 61

Medidor residencial antes y después de la prueba.



Podemos observar que la gráfica de consumo de energía dibuja una pendiente. Esto debido a que se utilizó una carga lineal durante la prueba, específicamente 3 focos incandescentes en paralelo, lo que representó una potencia activa de 325 W y una corriente de 2,45 A en 130 VAC en promedio.

5.15.2. Pruebas de Precisión de Voltaje, Corriente, Frecuencia, Potencia y Factor de Potencia

Como se observa en la Figura 62, se utilizaron varios tipos de carga para las pruebas de precisión usando el analizador de red METREL POWERQ4 PLUS, del tipo resistivo usando focos incandescentes, inductivo, usando un motor monofásico de 1 HP, y del tipo capacitivo usando capacitores permanentes como se observa en la Figura 63.

Figura 62

Mediciones en el Analizador de Red POWERQ4 PLUS

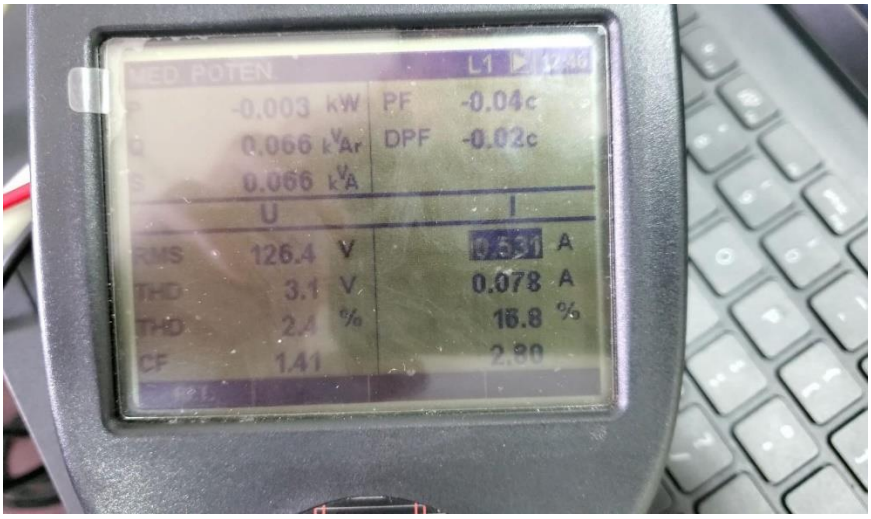


Tabla 22

Mediciones realizadas con carga resistiva.

	Voltaje [V]	**Corriente [A]**	**Potencia [W]**	**S [VA]**	**Q [VAr]**	**FP**	**Frecuencia [Hz]**
Pacefair PZEM-004T V3	125,4	0,347	44	42*	0*	1	60
METREL POWERQ4 PLUS	125,4	0,345	42	42	0	1	60
	0%	0,58%	4,55%	0%	0%	0%	0%

**Potencia Aparente y Reactiva del PZEM son calculadas utilizando las otras magnitudes medidas.*

Viendo los datos resultantes en la Tabla 19, podemos notar que no hay mucha diferencia entre las mediciones cuando las mediciones son de tipo resistivo.

Tabla 23

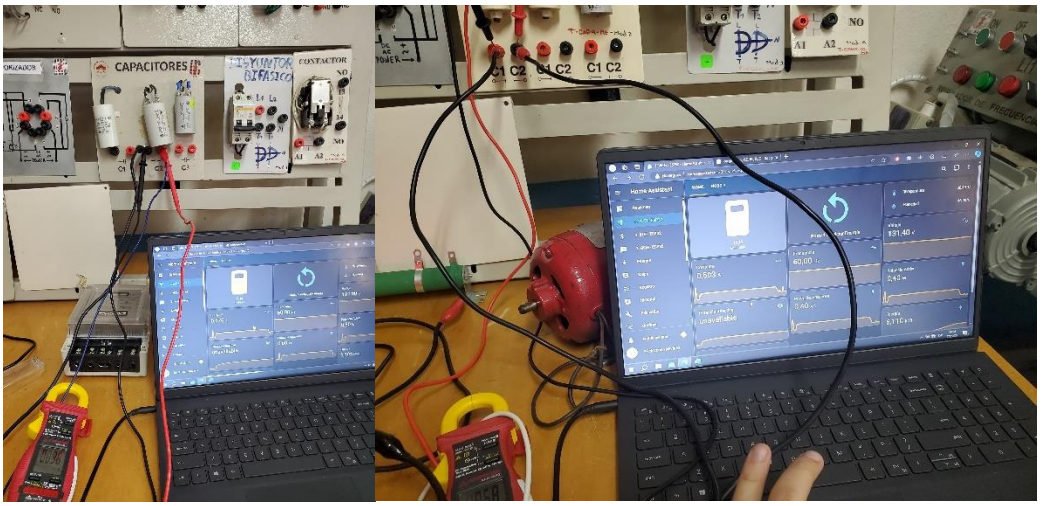
Mediciones realizadas con carga capacitiva

	Voltaje [V]	Corriente [A]	Potencia [W]	S [VA]	Q [VAr]	FP	Frecuencia [Hz[
Pacefair PZEM-004T V3	129,1	2,913	0,4	59*	61,38*	0,01	59,9
METREL POWERQ4 PLUS	129,1	2,913	0,35	63	63	0,02	59,9
	0%	0%	14%	6,51%	2,571	50 %	0%

**Potencia Aparente y Reactiva del PZEM son calculadas utilizando las otras magnitudes medidas.*

Figura 63

Pruebas realizadas con cargas resistivas, inductivas y capacitivas en laboratorio.



En la Tabla 20 de las mediciones de tipo capacitivo es donde podemos notar la imprecisión del PZEM en cargas de tipo capacitivo. La diferencia en cuanto a potencia activa no es abismal, pero en factor de potencia sí, lo que ocasiona que los cálculos de la potencia reactiva y aparente del módulo sea muy imprecisa.

Tabla 24

Mediciones realizadas con carga inductiva

	Voltaje [V]	Corriente [A]	Potencia [W]	S [VA]	Q [VAr]	FP	Frecuencia [Hz[
Peacefair PZEM-004T V3	129,1	2,913	82,5	373,3*	370*	0,2	59,9
METREL POWERQ4 PLUS	129,1	2,913	72	363	357	0,22	59,9
	0%	0%	14.58%	2.8%	3.6%	9.1%	0%

**Potencia Aparente y Reactiva del PZEM son calculadas utilizando las otras magnitudes medidas.*

En el caso de las mediciones con carga inductiva de la Tabla 21, se puede observar las diferencias en las mediciones, sin embargo, son menos radicales que con las cargas capacitivas, ya que mantienen solo una diferencia de 0,02 de factor de potencia entre ambas.

Tabla 25

Mediciones realizadas en circuito RL Paralelo.

	Voltaje [V]	Corriente [A]	Potencia [W]	S [VA]	Q [VAr]	FP	Frecuencia [Hz[
Peacefair PZEM-004T V3	129,1	3,021	129,5	385,59*	362,44*	0,31	60
METREL POWERQ4 PLUS	129,1	3,021	123	393	375	0,34	60
	0%	0%	5.28%	1.88%	3.3%	8.8%	0%

**Potencia Aparente y Reactiva del PZEM son calculadas utilizando las otras magnitudes medidas.*

Tabla 26

Mediciones realizadas en circuito RLC Paralelo.

	Voltaje [V]	Corriente [A]	Potencia [W]	S [VA]	Q [VAr]	FP	Frecuencia [Hz[
Peacefair PZEM-004T V3	129,1	2,52	124	334,1 *	307,17 *	0,39	59,9
METREL POWERQ4 PLUS	129,1	2,52	123	336.7	315	0,36	59,9
	0%	0%	0.81%	0.77%	2.48%	8.3%	0%

**Potencia Aparente y Reactiva del PZEM son calculadas utilizando las otras magnitudes medidas.*

Las Tablas 21 y 22 demuestran que la precisión del módulo PZEM-004T V3 con las cargas resistivas e inductivas se mantiene decentemente alta aun cuando se combina con cargas capacitivas, aun cuando el módulo es impreciso con cargas de ese tipo.

Tabla 27

Mediciones realizadas en circuito RC Paralelo.

	Voltaje [V]	Corriente [A]	Potencia [W]	S [VA]	Q [VAr]	FP	Frecuencia [Hz[
Peacefair PZEM-004T V3	129,6	2,429	84,7	313,7*	302,73 *	0,27	59,9
METREL POWERQ4 PLUS	129,9	2,498	78	324	315	0,24	59,9
	0%	0%	8.59%	3.17 %	3.89%	12.5%	0%

**Potencia Aparente y Reactiva del PZEM son calculadas utilizando las otras magnitudes medidas.*

Tabla 28

Mediciones realizadas en circuito LC Paralelo.

	Voltaje [V]	Corriente [A]	Potencia [W]	S [VA]	Q [VAr]	FP	Frecuencia [Hz[
Peacefair PZEM-004T V3	129,4	2,6	86,5	320*	298,43 *	0,27	59,9
METREL POWERQ4 PLUS	129,9	2,6	96	339	324	0,28	59,9
	0%	0%	9.89%	5.6%	7.89%	3.5%	0%

**Potencia Aparente y Reactiva del PZEM son calculadas utilizando las otras magnitudes medidas.*

Luego de comprobar en las Tablas 23 y 24 que existen diferencias en las mediciones, no son tan altas si de una carga puramente capacitiva se tratase, como el caso de las pruebas de la Tabla 20.

5.16. Pruebas de Transmisión

Se hicieron pruebas de transmisión en interiores con las 7 tecnologías de comunicación utilizadas en el Sistema. Para esto, se midió la potencia de transmisión o RSSI en dBm cada 5 metros hasta que el dispositivo dejase de responder. En el caso de nRF24, se calculó la potencia, ya que esta tecnología no proporciona esta magnitud de forma nativa.

Tabla 29

Potencia de transmisión en relación con la distancia

		RSSI				
		NODO 1 WI-FI	NODO 2 BLE	NODO 5 LORA	NODO 6 ZIGBEE	NODO 7 Z-WAVE
D	0	-17	-50	-39	-44	-50
	5	-48	-77	-45	-44	-83
I	10	-65	-84	-60	-44	-89
	15	-75		-65	-44	-91
	20	-82		-65	-72	-91
S	25			-70	-72	-91
	30			-75	-76	-91
T	35			-75	-76	-91
	40			-80		-91
A	45			-80		-91
	50			-85		-91
N	55			-85		-91
	60			-85		-95
C	65			-90		
	70			-90		
I	75			-95		
	80			-95		
A	85			-95		
	90			-100		
	95			-105		
	100			-115		

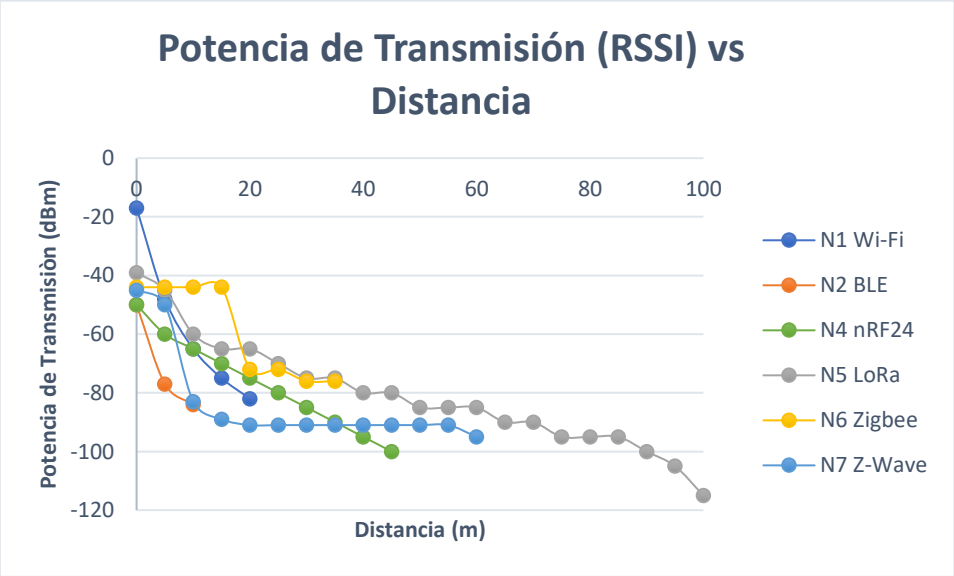
Como podemos observar en la Tabla 25, Wi-Fi muestra una rápida disminución en la potencia de transmisión conforme aumenta la distancia, con una caída notable ya a los 5 metros. BLE sigue una tendencia similar a Wi-Fi, aunque los datos solo están disponibles hasta los 10 metros. Zigbee muestra una transmisión bastante constante hasta los 15 metros, después de lo cual la potencia disminuye significativamente.

Z-Wave muestra una disminución más gradual en comparación con las otras tecnologías, manteniendo una potencia de transmisión más constante a mayores distancias. LoRa muestra una disminución menos pronunciada en la potencia de transmisión, sugiriendo que es efectiva a largas distancias, algo que se espera de esta tecnología, diseñada para la comunicación de largo alcance a baja potencia.

nRF24 logró establecer conexión hasta los 45 m, por lo que se supondrá que antes de esa distancia funcionaría con una potencia de transmisión de -100 dBm. Finalmente, el gráfico comparativo se vería como en la siguiente Figura.

Figura 64

Potencia de transmisión y Distancia



En el caso del Nodo 3 GSM, la recopilación de datos de potencia de transmisión es distinta, ya que no estaríamos comunicando ambos transceptores entre si (de punto a punto) como lo hemos hecho anteriormente, si no que cada dispositivo GSM se estaría comunicando con la torre más cercana.

5.17. Conclusiones

En cuanto a la contextualización teórica realiza en este trabajo de grado, cuyo contenido se basó, principalmente, en el estudio de las tecnologías de comunicación y microcontroladores, es viable señalar que la amplia variedad de estas herramientas permite la flexibilidad necesaria para abordar proyectos de ingeniería de manera diversa y adaptarse a sus requisitos específicos para el cumplimiento de los objetivos propuestos. Además, posibilita al usuario común la planeación y elaboración de dispositivos alternativos con prestaciones superiores a los comerciales, mejorando la accesibilidad y disponibilidad de datos con el manejo de redes inalámbricas de forma inmediata y local.

En la propuesta metodológica del proyecto, se utilizaron ciertos criterios de selección y construcción de los dispositivos que garantizaron un funcionamiento de acorde a lo esperado, priorizando aspectos como la Ley Orgánica de Telecomunicaciones con respecto al uso de bandas libres, que garantizaron no solo la conformidad con las regulaciones gubernamentales, sino que también permitió la implementación eficaz de las tecnologías de comunicación. El uso de la norma NEMA 5-15p no solo ha establecido una base estandarizada para la determinación de los valores nominales eléctricos de los nodos, sino que también ha contribuido a la compatibilidad y operabilidad con otros dispositivos. La inclusión de parámetros de control de calidad del INEN para medidores residenciales, basados en normativa IEC, determinó las características base de medición y, asimismo, la aplicación de la norma IPC-2221b para el cálculo del ancho de trazo en las placas de circuito impreso ha facilitado un proceso de fabricación preciso, garantizando la integridad y eficiencia de las conexiones eléctricas.

La implementación de los distintos nodos ha sido respaldada por una previa planificación y ejecución en diversas etapas. La realización de una diagramación detallada de los circuitos fue fundamental para guiar el diseño y la construcción de los nodos, proporcionando una representación técnica y visual precisa que facilitó la comprensión y la ejecución efectiva. Se realizaron varias pruebas antes de los resultados finales, principalmente

en la elaboración del código de programación y el funcionamiento de los módulos de comunicación, siempre procurando realizar una conexión directa de los nodos entre sí.

Con las pruebas realizadas se pudo concluir que el PZEM-004T V3 variará su precisión dependiendo el tipo de carga, el cual, a su vez, está relacionado con el factor de potencia. Con la mesa de pruebas de medidores de Zera se obtuvo un error del 0.12% con un factor de potencia de 1, y 4.51% en la medición de energía usando el led de la constante del medidor. Estos datos fueron validados posteriormente cuando verificamos una mayor diferencia de las mediciones al momento de comparar el PZEM-004T V3 con el Analizador de Red METREL POWERQ4 PLUS. Estas dos referencias nos demuestran que, en términos de precisión, un módulo de medición AC económico como el PZEM-004T V3 podría categorizarse como medidor de alta precisión de Clase 0.2S (±0.2%) con factores de potencia cercanos a 1, pero con factores de potencia menores, no entraría ni si quiera en la categoría Clase 3 (±3%) según la normativa IEC 62053 utilizada en Ecuador.

Sin embargo, el módulo PZEM-004T V3 sigue siendo un medidor adecuado para el uso residencial, puesto que, en un hogar común, gran parte de la energía consumida es del tipo resistiva, y mientras más se cumpla este estamento, menor sería el error perceptible en las mediciones de energía resultantes.

En cuanto a los resultados obtenidos de las pruebas de las tecnologías de transmisión, podemos concluir que, para aplicaciones de corto alcance y alta demanda de ancho de banda, Wi-Fi y BLE con el ESP32 podrían ser las opciones preferibles. Además de ser una tecnología con una oferta de mercado amplia por su popularidad, es bastante económica al momento de buscar otras alternativas. Zigbee con el Xbee S2C y Z-Wave con el Z-Uno 2 son aptas para aplicaciones domóticas y de automatización a distancias intermedias, con Z-Wave mostrando una mayor consistencia en la señal, sin embargo, podrían ser las dos tecnologías más costosas de las implementadas en este Trabajo de Grado.

Aunque la potencia de transmisión no fue analizada para nRF24L, su latencia se mantuvo constante hasta los 30 metros y aumentó significativamente más allá de esta distancia. Esto sugiere que, para aplicaciones donde la respuesta en tiempo real es crítica, nRF24 sería adecuada siempre y cuando la distancia de operación se mantenga dentro del rango de latencia aceptable.

La tecnología de comunicación más eficiente al momento de considerar precio, facilidad de configuración y alcance, fue LoRa. El módulo RYLR998 REYAX terminó siendo el que, a pesar de su compacto tamaño, llegó a alcanzar distancias de 100 metros en entornos cerrados, casi el doble que el segundo puesto, Z-Wave con el Z-Uno 2, que solo alcanzó los 60 metros y que es una tecnología significativamente más costosa.

5.18. Recomendaciones

Como se mencionó anteriormente, el auge de la comunicación inalámbrica ha incentivado a las empresas al desarrollo continuo de nuevas alternativas a las tradicionales. Tecnologías como LoRa, Zigbee o Z-Wave, que permiten comunicación a larga distancia, podrían ser útiles para la elaboración de prototipos analizadores de red que realicen la captación de mediciones eléctricas para estudios de calidad energética a nivel distribución, obteniendo una visualización de los datos obtenidos a tiempo real, contribuyendo así a proyectos de generación distribuida.

También, la especialización de una sola tecnología de comunicación permitiría profundizar en todo lo que la involucra, adquiriendo conocimiento más detallado que contribuya al aprovechamiento máximo de sus características, mejorando variables como latencia, rango de transmisión, seguridad.

Bibliografía

[1] U. A. Bakshi y L. A. V. Bakshi, *Electrical Measurements and Instrumentation*. UNICORN Publishing Group, 2020. [En línea]. Disponible en: https://books.google.com.ec/books?id=0sMZEAAAQBAJ

[2] International Electrotechnical Commission, "Electricity metering equipment - General requirements, tests and test conditions - Part 11: Metering equipment". el 29 de junio de 2020.

[3] R. Masnicki y J. Mindykowski, "What Should Be Measured Using Static Energy Meters", en *2018 International Conference and Exposition on Electrical and Power Engineering (EPE)*, Iasi, Romania: IEEE, oct. 2018, pp. 0183–0188. doi: 10.1109/ICEPE.2018.8559757.

[4] S. Solaimalai, P. Semwal, S. Palit, y S. Indulkar, "Smart Metering in Smart Grid", *Int. J. Eng. Adv. Technol.*, vol. 8, pp. 1021–1027, abr. 2019.

[5] Richard Albritton, "Set up Home Assistant with a Raspberry Pi". Adafruit Industries, el 29 de agosto de 2023. [En línea]. Disponible en: https://cdn-learn.adafruit.com/downloads/pdf/set-up-home-assistant-with-a-raspberry-pi.pdf

[6] M.-C. Șuvar, L. Munteanu, y C. Cioară, "Optimal Monitoring of Server Rooms with Home Assistant Platform", *MATEC Web Conf.*, vol. 373, p. 00044, 2022, doi: 10.1051/matecconf/202237300044.

[7] Tomás Tobajas Balsera, "Hogar inteligente DIY", Tesis de Grado, Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación, 2020.

[8] A. K. Meier, "New standby power targets", *Energy Effic.*, vol. 12, núm. 1, pp. 175–186, ene. 2019, doi: 10.1007/s12053-018-9677-x.

[9] Byron Michel Ayón Baque, "Análisis De Los Beneficios De La Tecnología Mesh En Las Redes Inalámbricas Del Complejo Universitario Unesum.", Tesis de Grado, Universidad Estatal de Sur de Manabí, Manabí, 2020. [En línea]. Disponible en: https://repositorio.unesum.edu.ec/bitstream/53000/2667/1/AYON%20BAQUE%20BYRON%20MICHEL.pdf

[10] Universidad Libre y F. Simanca H., *Las Redes MESH*. Universidad Libre, 2018. doi: 10.18041/UL-958-5466-20-3.

[11] Ttamnametab, "Mesh network", Wikimedia Commons. [En línea]. Disponible en: https://commons.wikimedia.org/wiki/File:Mesh_network.png

[12] Jenny Gabriela Chango Tonato, "Análisis Comparativo de la Privacidad de los Datos Exclusivamente en la Transmisión hacia las Plataformas de IoT más Utilizadas", Tésis de Grado, Universidad Politécnica Salseana Ecuador, Quito - Ecuador, 2022. [En línea]. Disponible en: https://dspace.ups.edu.ec/bitstream/123456789/22172/1/UPS%20-%20TTS658.pdf

[13] J. N. Dwivedi, "Internet of Things (IoT) and Machine to Machine (M2M) Communication Techniques for Cyber Crime Prediction", en *Intelligent Data Analytics for Terror Threat Prediction*, 1a ed., S. K. Pani, S. K. Singh, L. Garg, R. B. Pachori, y X. Zhang, Eds., Wiley, 2021, pp. 31–55. doi: 10.1002/9781119711629.ch2.

[14] J. Lloret, M. Garcia, D. Bri, y J. Diaz, "A Cluster-Based Architecture to Structure the Topology of Parallel Wireless Sensor Networks", *Sensors*, vol. 9, núm. 12, pp. 10513–10544, dic. 2009, doi: 10.3390/s91210513.

[15] Michelle Panico, "Wi-Fi Alliance® 2022 Wi-Fi® trends", 2022 Wi-Fi® trends. [En línea]. Disponible en: https://www.wi-fi.org/news-events/newsroom/wi-fi-alliance-2022-wi-fi-trends

[16] Bibek Devkota y Hari Bhandari, "Next Generation of Wireless Networks: Wi-Fi 6 and 5G", Tesis de Grado, Metropolia University of Applied Sciences, 2020. [En línea]. Disponible en: https://www.theseus.fi/bitstream/handle/10024/346134/Bhandari_Hari_and_Devkota_Bibek.pdf?sequence=2

[17] Richard van Ginkel, "Security in public Wi-Fi networks", Tesis de Grado, Radboud University, Nijmegen, Netherlands, 2019. [En línea]. Disponible en: https://www.cs.ru.nl/bachelors-theses/2019/Richard_van_Ginkel___4599047___Security_in_public_Wi-Fi_networks.pdf

[18] L. D. Tan, “A Survey on Internet of Things for Smart Health Technologies”, 2018, doi: 10.13140/RG.2.2.18701.56800.

[19] Arturo Mrozowski Handzel, “Implementación del núcleo de red LTE/5G virtualizado”, Tesis de Grado, Universitat Politècnica de València, Valencia - España, 2020. [En línea]. Disponible en: https://riunet.upv.es/handle/10251/152386

[20] H. Mukhtar, “Machine Learning Enabled-Localization in 5G and LTE Using Image Classification and Deep Learning”, jul. 2021, doi: 10.20381/RUOR-26669.

[21] Ricardo Albornoz y Eduardo Soto, “Redes de Computadoras I: Estudio del Estándar Zigbee”, Universidad T´ecnica Federico Santa Mar´ıa, Valparaiso - Chile, 2018. [En línea]. Disponible en: http://profesores.elo.utfsm.cl/~agv/elo322/1s18/projects/reports/Zigbee.pdf

[22] Francisco José Velasco Iglesias, “Sistema IoT basado en el control y monitoreo de una red ZigBee. Aplicación en servicio domótico.”, Tesis de Grado, Universidad de Sevilla, Sevilla, 2021. [En línea]. Disponible en: https://idus.us.es/bitstream/handle/11441/127378/TFG-3759-VELASCO%20IGLESIAS.pdf?sequence=1&isAllowed=y

[23] Martin Persson, “A Framework for Monitoring Data from a Smart Home Environment”, Tesis de Master, Luleå University of Technology, Luleå - Sweden, 2020. [En línea]. Disponible en: https://www.diva-portal.org/smash/get/diva2:1445707/FULLTEXT01.pdf

[24] Claudia Jiménez García, “Integración de tecnología domótica Z-Wave en la plataforma FIBARO”, Tesis de Grado, Universidad de Sevilla, 2019. [En línea]. Disponible en: https://idus.us.es/handle/11441/85472?show=full

[25] Tyler Wojciechowicz, “ZigBee vs. Z-Wave: What’s the Difference?”, Mouser Electronics. [En línea]. Disponible en: https://www.mouser.com/blog/zigbee-vs-z-wave-whats-the-difference

[26] Luis Roberto Sánchez Criollo, “Sistema de Automatización y Control para una silla bipedestadora en el Centro de Rehabilitación Física ‘Bendiciones’”, Tesis de Grado, Universidad Técnica de Ambato, Ambato - Ecuador, 2020. [En línea]. Disponible en: https://repositorio.uta.edu.ec/bitstream/123456789/31983/1/t1752ec.pdf

[27] B. Paul, "An Overview of LoRaWAN", *WSEAS Trans. Commun.*, vol. 19, pp. 231–239, ene. 2021, doi: 10.37394/23204.2020.19.27.

[28] Miko Nore y Caspar Westerberg, "Robotic Arm controlled by Arm Movements", Tesis de Grado, KTH Vetenskap Och Konst, Stockholm, Sweden, 2019. [En línea]. Disponible en: http://kth.diva-portal.org/smash/get/diva2:1373883/FULLTEXT01.pdf

[29] Jefferson Moreno, "Detección de eventos de contaminación acústica industrial basado en una red de sensores: desarrollo de una red de sensores inalámbrica; detección, transporte y recopilación de datos de eventos acústicos industriales de la empresa danilact.", Tesis de Grado, Escuela Politécnica Nacional, Quito - Ecuador, 2022. [En línea]. Disponible en: https://bibdigital.epn.edu.ec/handle/15000/23326

[30] O. Kurkutlu, MLONDI JEFFREY MCHUNU, y EZZADDIN ABDULSALAM HASAN SHBOOR, "Quadruped robot - Four Legged Robot", 2021, doi: 10.13140/RG.2.2.26784.92161.

[31] A. Pérez-Escoda y R. G. Ruiz, "Comunicación y Educación en un mundo digital y conectado", *Rev. ICONO14 Rev. Científica Comun. Tecnol. Emerg.*, vol. 18, núm. 2, pp. 1–15, jul. 2020, doi: 10.7195/ri14.v18i2.1580.

[32] Piyu Dhaker, "Introduction to SPI Interface", *Analog Dialogue*, vol. 52, núm. 3, p. 5, septiembre de 2018.

[33] José M González-García, José J Sepúlveda-Cisneros, Javier Villa-Chávez, y Roberto Paz-Vázquez, "Reemplazo del protocolo SENT por LIN", Tesis de Grado, Instituto Tecnológico y de Estudios Superiores de Occidente, Tlaquepaque, Jalisco, 2018. [En línea]. Disponible en: https://rei.iteso.mx/bitstream/handle/11117/5549/Tesina_Reemplazo_Protocolo_LIN_por_SENT.pdf?sequence=2&isAllowed=y

[34] Federico Pozzana, "RISC-V: an FPGA implementation for general purpose prototyping hardware", Tesis de Master, Politecnico Di Torino, Torino - Italia, 2020. [En línea]. Disponible en: https://webthesis.biblio.polito.it/16652/1/tesi.pdf

[35] Julio Enrique Cuji Rodriguez y Bladimir Sebastian Novoa Segovia, "Sistema de telecontrol y orientación remota de antenas direccionales punto a punto", Tesis de Grado,

Universidad Técnica de Ambato, Ambato - Ecuador, 2019. [En línea]. Disponible en: https://repositorio.uta.edu.ec/handle/123456789/29252

[36] Roberto Carlos Sierra Obregón, "Diseño y construcción de un sistema de control para un aparato rehabilitador físico", Tesis de Grado, Instituto Tecnológico de Matamoros, Matamoros - México, 2020. [En línea]. Disponible en: https://rinacional.tecnm.mx/bitstream/TecNM/1296/1/_Roberto%20Carlos%20Sierra%20Obreg%C3%B3n.pdf

[37] Raspberry Pi Trading Ltd, "Raspberry Pi 4 Computer Model B". Raspberry Pi, enero de 2021. [En línea]. Disponible en: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf

[38] Espressif, "ESP32-WROOM-32 (ESP-WROOM-32)", Espressif Systems, Datasheet 2.1, 2018. [En línea]. Disponible en: https://www.mouser.com/datasheet/2/891/esp-wroom-32_datasheet_en-1223836.pdf

[39] Luis Adolfo Villalonga Ávila y Ricardo Andrés Panchana Salinas, "Diseño y construcción de módulos para sensores de proximidad y transductores de temperatura para el laboratorio de instalaciones industriales.", Tesis de Grado, Universidad Politécnica Salseana Ecuador, Quito - Ecuador, 2020. [En línea]. Disponible en: https://dspace.ups.edu.ec/handle/123456789/19316

[40] Diego Andrés Núñez Lozano y José Andrés Páez Castros, "La Importancia de los Actuadores". Universidad ECCI, 2022. [En línea]. Disponible en: http://ingenierovizcaino.com/ecci/aut1/corte1/articulos/Actuadores.pdf

[41] Marek Ventur, "Spiralmodel", Wikimedia Commons. [En línea]. Disponible en: https://commons.wikimedia.org/wiki/File:Spiralmodel.svg

[42] B. W. Boehm, *Software engineering economics*. en Prentice Hall advances in computing science and technology series. Upper Saddle River, N.J: Prentice Hall PTR, 1981.

[43] Schneider Electric España, "NEMA 5-15 receptacle description", FA156527. [En línea]. Disponible en: https://www.se.com/es/es/faqs/FA156527/

[44] Comrade Mmirg, "NEMA 5-15P", Wikimedia Commons. [En línea]. Disponible en: https://en.m.wikipedia.org/wiki/File:NEMA_5-15P.svg

[45] Agencia de Regulación y Control de las Telecomunicaciones, *RESOLUCIÓN Nro. 04-02-ARCOTEL-2021*. 2021.

[46] Julio César Santiago Asto, "Implementación de una metodología para la medición de la interferencia inalámbrica en la banda ISM en zonas exteriores urbanas para garantizar la comunicación de una red inalámbrica de sensores", Tesis de Master, Universidad Católica de Perú, Lima - Perú, 2017. [En línea]. Disponible en: https://tesis.pucp.edu.pe/repositorio/handle/20.500.12404/9512

[47] Instituto Ecuatoriano de Normalización, "Equipos de Medición de la Energía Eléctrica (C.A.) requisitos particulares. Parte 21: Contadores Estáticos de Energía Activa (Clases 1 y 2)". 2010. [En línea]. Disponible en: https://es.scribd.com/document/512924479/nte-inen-iec-62053-21-extracto

[48] Hardkernel, "ODROID-C4", Odroid Board. [En línea]. Disponible en: https://www.hardkernel.com/shop/odroid-c4/

[49] tinker board, "ASUS Tinker Board series". ASUS IoT, 2023. [En línea]. Disponible en: https://iot.asus.com/files/2023_Tinker_Flyer_final.pdf

[50] Lenovo, "The Lenovo® ThinkCentre® M91 and M91p Desktop". 2010. [En línea]. Disponible en: https://www.levnapc.cz/ProductsFiles/Lenovo-ThinkCentre-M91-M91p-technicke-specifikace.pdf

[51] Home Assistant, "Installation", Home Assistant Instalation. [En línea]. Disponible en: https://www.home-assistant.io/installation/

[52] Joy IT, "ARDUINO NANO V3". SIMAC GmbH, 2020. [En línea]. Disponible en: https://joy-it.net/files/files/Produkte/ARD_NanoV3/ARD-NanoV3-Manual-20200326.pdf

[53] Raspberry Pi Ltd, "Raspberry Pi Pico Datasheet an RP2040-based microcontroller board". Raspberry Pi Ltd, 2023. [En línea]. Disponible en: https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf

[54] ST, "High-density performance line Arm®-based 32-bit MCU with 256 to 512KB Flash, USB, CAN, 11 timers, 3 ADCs, 13 communication interfaces". julio de 2018. [En línea]. Disponible en: https://www.st.com/resource/en/datasheet/stm32f103rc.pdf

[55] Ningbo songle relay co. ltd, "SRD-12VDC-SL-C Datasheet". [En línea]. Disponible en: https://pdf1.alldatasheet.es/datasheet-pdf/view/1132031/SONGLERELAY/SRD-12VDC-SL-C.html

[56] Ningbo songle relay co. ltd, "SRD-09VDC-SL-C Datasheet". [En línea]. Disponible en: https://datasheetspdf.com/datasheet/SRD-09VDC-SL-C.html

[57] Ningbo songle relay co. ltd, "SRD-05VDC-SL-C Datasheet". [En línea]. Disponible en: https://www.circuitbasics.com/wp-content/uploads/2015/11/SRD-05VDC-SL-C-Datasheet.pdf

[58] Ningbo songle relay co. ltd, "SLA-05VDC-SL-C Datasheet". [En línea]. Disponible en: https://www.alldatasheet.es/datasheet-pdf/pdf/1132202/SONGLERELAY/SLA-05VDC-SL-C.html

[59] OMRON, "G3MB-202P". [En línea]. Disponible en: https://www.openhacks.com/uploadsproductos/g3mb-ssr-datasheet.pdf

[60] TONGLING, "JQC-T73". [En línea]. Disponible en: https://denkovi.com/Documents/JQC-3FF-S-Z.pdf

[61] TAKAMISAWA, "A-5W-K DPDT 5V 1A DIP Relay Datasheet". [En línea]. Disponible en: https://www.futurlec.com/Relays/A-5W-K.shtml

[62] Meishuo, "MPA-S-112-C". [En línea]. Disponible en: https://www.msrelay.com/Electromagnetic_Relay/170.html

[63] Home Assistant, "SMS notifications via GSM-modem". [En línea]. Disponible en: https://www.home-assistant.io/integrations/sms/

[64] SIM Com, "SIM800C Hardware Design". el 27 de abril de 2015. [En línea]. Disponible en: https://www.elecrow.com/download/SIM800C_Hardware_Design_V1.02.pdf

[65] HUAWEI, "HUAWEI E3372 LTE USB Stick". el 19 de enero de 2015. [En línea]. Disponible en: https://www.manual.ar/huawei/e3372/manual

[66] Telia, "Huawei E3531". el 23 de marzo de 2014. [En línea]. Disponible en: https://usermanual.wiki/Huawei/HUAWEIE3531iE3531sHSPAUSBStickProductDescriptionV100R00102Enpdf.1009037510.pdf

[67] HUAWEI, "HUAWEI E3372h-320 LTE USB Stick". el 20 de diciembre de 2019. [En línea]. Disponible en: https://data.kommago.nl/files/pdf/huawei-e3372h-320-datasheet.pdf

[68] Vodafone, "Safety Manual K 3565-Z". 2023. [En línea]. Disponible en: https://usermanual.wiki/ZTE/K3565-Z.Safety-manual/pdf

[69] REYAX, "RYLR890". el 9 de marzo de 2020. [En línea]. Disponible en: https://reyax.com/products/RYLR890

[70] REYAX, "RYLR998". el 5 de julio de 2023. [En línea]. Disponible en: https://reyax.com/products/RYLR998

[71] REYAX, "RYLR896". el 1 de noviembre de 2021. [En línea]. Disponible en: https://reyax.com/products/RYLR896

[72] Home Assistant, "Zigbee Home Automation". 2023. [En línea]. Disponible en: https://www.home-assistant.io/integrations/zha/

[73] Home Assistant, "Home Assistant SkyConnect". 2022. [En línea]. Disponible en: https://skyconnect.home-assistant.io/resources/SkyConnect_HASKYCNCT_Datasheet_v1_0.pdf

[74] SONOFF, "SONOFF Zigbee 3.0 USB Dongle". el 12 de octubre de 2021. [En línea]. Disponible en: https://manuals.plus/m/2ce34ac527d7d6405b2788b253139bf41f145ef995910bb3c00281390c215867_optim.pdf

[75] POPP, "POPP ZB-Stick (Zigbee)". enero de 2021. [En línea]. Disponible en: http://manuals-backend.z-wave.info/make.php?lang=en&sku=POPE701554&cert=&type=mini

[76] Home Assistant, "Z-Wave", Integrations. [En línea]. Disponible en: https://www.home-assistant.io/integrations/zwave_js/

[77] Aeotec, "Z-Stick Gen5+ user guide". el 24 de noviembre de 2021. [En línea]. Disponible en: https://www.planete-domotique.com/notices/A/E/AEO_ZW090PLUS-C_M.pdf

[78] Nortek Security & Control LLC, "HUSBZB-1". 2015. [En línea]. Disponible en: http://www.gocontrol.com/manuals/HUSBZB-1-Operation.pdf

[79] Zooz, "Zooz ZST10". 2018. [En línea]. Disponible en: https://www.getzooz.com/downloads/zooz-z-wave-plus-radio-s2-usb-stick-zst10-manual.pdf

[80] Z-Wave PLUS, "ZME_UZB1". [En línea]. Disponible en: https://gzhls.at/blob/ldb/c/c/4/2/0f3b38132bfb8844663df72ed51e12693fa0.pdf

[81] IPC, "Generic Standard on Printed Board Design". 2012. [En línea]. Disponible en: https://www.ipc.org/TOC/IPC-2221B.pdf

[82] M. Fezari y A. Al Dahoud, "Integrated Development Environment 'IDE' For Arduino", oct. 2018.

[83] infinite, "MQTT: Connecting to the Mosquitto MQTT Broker". julio de 2021. [En línea]. Disponible en: https://www.infinite.com.gr/docs/Case_Studies/Case%20Study%20MQTT_Connecting%20to%20a%20Mosquitto%20Broker%20v1.pdf

[84] Erik Sandberg, "High Performance Querying of Time Series Market Data", Tesis de Master, Umeå University, 2021. [En línea]. Disponible en: http://www.diva-portal.org/smash/get/diva2:1638667/FULLTEXT01.pdf

[85] Jirson Gary Olvera Ronquillo, "Análisis de la Frecuencia Modal en Discos de Frenos a través de los Softwares Inventor y Fusión 360", Tesis de Grado, Universidad Internacional del Ecuador, Guayas - Ecuador, 2023.

[86] José Ariel Suárez Briones, "Prototipo de dispositivo electrónico IoT para la adquisiciónde diversos parámetros físicos en la transportación pública.", Universidad Estatal Península de Santa Elena, Libertad - Ecuador, 2019.

[87] ZERA, "ZERA Company", Company Philosophy. [En línea]. Disponible en: https://www.zera.de/en/company/#philosophy

[88] ZERA, "MTS – Communication", Standard Meter Test System for Testing Metrology and Communication. [En línea]. Disponible en: https://www.zera.de/en/product/meter-testing/stationary-meter-test-systems-ac/complete-systems/meter-testing-data-communication/

[89] METREL, “MI 2792A PowerQ4 Plus NEW”, metrel.si. [En línea]. Disponible en: https://www.metrel.si/assets/Metrel/PDF_dokumentacija/Single_leaflets/MI_2792_PowerQ4_Plus/Ang/Single_2012_MI_2792A_PowerQ4_Plus_Ang.pdf

[90] Yue Qing Blue Sky High Tech, “DDS8888 monofásico de dos hilos medidor de energía electrónico (LCD)”, Itgkj.com. [En línea]. Disponible en: http://www.ltgkj.com/es/Single-Phase-Electronic-Energy-Meter/DDS8888-monof%C3%A1sico-de-dos-hilos-medidor-de-energ%C3%ADa-electr%C3%B3nico-(LCD)-1990.html

ANEXOS

ANEXO A: Códigos de Programación

ANEXO A.1 Código de Programación del Nodo 1 para ESP32 en Arduino IDE

```
#include <WiFi.h> // Incluye la biblioteca para manejar la conexión WiFi del ESP32.

extern "C" { // Permite la inclusión de bibliotecas escritas en C dentro de un código C++.
#include "freertos/FreeRTOS.h" // Incluye las definiciones básicas de FreeRTOS para multitarea.
#include "freertos/timers.h" // Incluye las definiciones para manejo de temporizadores en FreeRTOS.
}

#include <AsyncMqttClient.h> // Incluye la biblioteca para manejo de cliente MQTT de forma asíncrona.

#include "DHT.h" // Incluye la biblioteca para el sensor de temperatura y humedad DHT.

#include <PZEM004Tv30.h> // Incluye la biblioteca para el sensor de potencia PZEM-004T v3.0.

// Definiciones de constantes para configuración
#define PIN_DHT 4 // Pin donde se conecta el sensor DHT.
#define TIPO_DHT DHT11 // Define el tipo de sensor DHT utilizado.

#define SSID_WIFI "CN's" // Nombre de la red WiFi.
#define CONTRASENA_WIFI "" // Contraseña de la red WiFi.

#define HOST_MQTT IPAddress(192, 168, 0, 104) // Dirección IP del servidor MQTT.
#define PUERTO_MQTT 1883 // Puerto utilizado por el servidor MQTT.

// Topics para la publicación de datos a través de MQTT.
#define MQTT_PUBLICAR_VOLTAJE "esp32-wifi/PZEM/Voltaje"
#define MQTT_PUBLICAR_CORRIENTE "esp32-wifi/PZEM/Corriente"
#define MQTT_PUBLICAR_FRECUENCIA "esp32-wifi/PZEM/Frecuencia"
#define MQTT_PUBLICAR_FACTOR_POTENCIA "esp32-wifi/PZEM/FactorDePotencia"
#define MQTT_PUBLICAR_POTENCIA "esp32-wifi/PZEM/Potencia"
#define MQTT_PUBLICAR_ENERGIA "esp32-wifi/PZEM/Energia"
#define MQTT_PUBLICAR_TEMPERATURA "esp32-wifi/DHT11/Temperatura"
#define MQTT_PUBLICAR_HUMEDAD "esp32-wifi/DHT11/Humedad"

// Configuración de pines y variables globales
int pinRele = 13; // Pin para el rele
int pinLed = 26; // Pin para el LED

PZEM004Tv30 pzem(Serial2, 16, 17); // Instancia del sensor PZEM usando Serial2.
DHT dht(PIN_DHT, TIPO_DHT); // Instancia del sensor DHT.

// Variables para almacenar las mediciones de los sensores.
float voltaje = 0;
float corriente = 0;
float potencia = 0;
float energia = 0;
float frecuencia = 0;
float factorPotencia = 0;

```

```
float humedad = 0;
float temperatura = 0;

float energiaAnterior = 0; // Almacena la última medición de energía para comparaciones.

// Cliente MQTT y temporizadores para reconexión
AsyncMqttClient clienteMqtt;
TimerHandle_t temporizadorReconexionMqtt;
TimerHandle_t temporizadorReconexionWifi;

unsigned long millisAnterior = 0; // Controla el intervalo de tiempo para acciones repetitivas.
const long intervalo = 5000; // Intervalo en milisegundos para la publicación de datos.

// Funciones para conexión a WiFi y MQTT, y manejo de eventos.
void conectarAWifi() {
  WiFi.begin(SSID_WIFI, CONTRASENA_WIFI); // Inicia la conexión a la red WiFi.
}

void conectarAMqtt() {
  clienteMqtt.connect(); // Inicia la conexión al servidor MQTT.
}

void eventoWiFi(WiFiEvent_t evento) {
    switch (evento) {
    case SYSTEM_EVENT_STA_GOT_IP: // Al obtener IP, intenta conectar a MQTT.
      conectarAMqtt();
      break;
    case SYSTEM_EVENT_STA_DISCONNECTED: // Al desconectar, reinicia los temporizadores de reconexión.
      xTimerStop(temporizadorReconexionMqtt, 0);
      xTimerStart(temporizadorReconexionWifi, 0);
      break;
  }
}

// Callbacks para eventos MQTT (conexión, desconexión, suscripción, publicación de mensajes).
void alConectarMqtt(bool sesionPresente) {
  // Suscribe a un topic específico al conectar.
  uint16_t paqueteIdSub = clienteMqtt.subscribe("esp32-wifi/control", 2);
}

void alDesconectarMqtt(AsyncMqttClientDisconnectReason motivo) {
  // Intenta reconectar si aún está conectado a WiFi.
  if (WiFi.isConnected()) {
    xTimerStart(temporizadorReconexionMqtt, 0);
  }
}

// Estas funciones manejan eventos de suscripción, publicación y mensajes MQTT.
void alSuscribirseMqtt(uint16_t paqueteId, uint8_t qos) {}
void alPublicarMqtt(uint16_t paqueteId) {}
void alDesuscribirseMqtt(uint16_t paqueteId) {}
void alRecibirMensajeMqtt(char* tema, char* carga, AsyncMqttClientMessageProperties propiedades, size_t longitud, size_t indice, size_t total) {
  // Aquí se manejarían los mensajes recibidos, por ejemplo, para controlar el relé.
}

void setup() {
  Serial.begin(115200); // Configura la velocidad de la comunicación serial.
```

```
106.
107.    pinMode(pinRele, OUTPUT); // Establece el pin del relé como salida.
108.    pinMode(pinLed, OUTPUT); // Establece el pin del LED como salida.
109.    digitalWrite(13,  1); // Estado inicial del relé.
110.
111.    // Configura temporizadores para reconexión automática a WiFi y MQTT.
112.    temporizadorReconexionMqtt = xTimerCreate("temporizadorMqtt", pdMS_TO_TICKS(2000),
pdFALSE, (void*)0, reinterpret_cast<TimerCallbackFunction_t>(conectarAMqtt));
113.    temporizadorReconexionWifi = xTimerCreate("temporizadorWifi", pdMS_TO_TICKS(2000),
pdFALSE, (void*)0, reinterpret_cast<TimerCallbackFunction_t>(conectarAWifi));
114.
115.    WiFi.onEvent(eventoWiFi); // Registra el manejador de eventos WiFi.
116.
117.    // Configura los callbacks para distintos eventos MQTT.
118.    clienteMqtt.onConnect(alConectarMqtt);
119.    clienteMqtt.onDisconnect(alDesconectarMqtt);
120.    clienteMqtt.onSubscribe(alSuscribirseMqtt);
121.    clienteMqtt.onUnsubscribe(alDesuscribirseMqtt);
122.    clienteMqtt.onMessage(alRecibirMensajeMqtt);
123.    clienteMqtt.onPublish(alPublicarMqtt);
124.
125.    clienteMqtt.setServer(HOST_MQTT, PUERTO_MQTT); // Establece el servidor MQTT.
126.    clienteMqtt.setCredentials("mqtt", "12125454qwxdxdA@"); // Configura credenciales para
MQTT.
127.
128.    conectarAWifi(); // Inicia la conexión WiFi.
129.
130.    dht.begin(); // Inicia el sensor DHT.
131. }
132.
133.  void loop() {
134.    // Verifica si el rele está desactivado.
135.    if(digitalRead(pinRele) == 0){
136.      unsigned long millisActual = millis(); // Almacena el tiempo actual en milisegundos.
137.
138.      // Lee los valores de los sensores.
139.      humedad = dht.readHumidity(); // Lee la humedad del sensor DHT.
140.      temperatura = dht.readTemperature(); // Lee la temperatura del sensor DHT.
141.
142.      // Lee los valores del sensor PZEM.
143.      voltaje = pzem.voltage();
144.      corriente = pzem.current();
145.      potencia = pzem.power();
146.      energia = pzem.energy();
147.      frecuencia = pzem.frequency();
148.      factorPotencia = pzem.pf();
149.
150.      // Enciende el LED si la energía ha aumentado desde la última medición.
151.      if (energia > energiaAnterior) {
152.        digitalWrite(pinLed, HIGH); // Enciende el LED.
153.        tiempoInicio = millisActual; // Actualiza el momento en que se encendió el LED.
154.      }
155.
156.      // Apaga el LED después del intervalo especificado.
157.      if (millisActual - tiempoInicio >= intervalo2) {
158.        digitalWrite(pinLed, LOW); // Apaga el LED.
159.      }
160.
161.      energiaAnterior = energia; // Actualiza la última medición de energía.
162.
163.      // Publica los valores a los topics de MQTT cada cierto intervalo.
```

```
164.     if (millisActual - millisAnterior >= intervalo) {
165.       millisAnterior = millisActual; // Actualiza el tiempo para la próxima publicación.
166.
167.       // Publica cada uno de los valores medidos en sus respectivos topics MQTT.
168.       uint16_t paqueteIdPub1 = clienteMqtt.publish(MQTT_PUBLICAR_VOLTAJE, 2, true,
String(voltaje).c_str());
169.       uint16_t paqueteIdPub2 = clienteMqtt.publish(MQTT_PUBLICAR_CORRIENTE, 2, true,
String(corriente, 3).c_str());
170.       uint16_t paqueteIdPub3 = clienteMqtt.publish(MQTT_PUBLICAR_POTENCIA, 2, true,
String(potencia, 2).c_str());
171.       uint16_t paqueteIdPub4 = clienteMqtt.publish(MQTT_PUBLICAR_ENERGIA, 2, true,
String(energia, 3).c_str());
172.       uint16_t paqueteIdPub5 = clienteMqtt.publish(MQTT_PUBLICAR_FACTOR_POTENCIA, 2, true,
String(factorPotencia, 3).c_str());
173.       uint16_t paqueteIdPub6 = clienteMqtt.publish(MQTT_PUBLICAR_FRECUENCIA, 2, true,
String(frecuencia).c_str());
174.       uint16_t paqueteIdPub7 = clienteMqtt.publish(MQTT_PUBLICAR_TEMPERATURA, 2, true,
String(temperatura).c_str());
175.       uint16_t paqueteIdPub8 = clienteMqtt.publish(MQTT_PUBLICAR_HUMEDAD, 2, true,
String(humedad).c_str());
176.     }
177.   }
178.   else{
179.     // esp_sleep_enable_timer_wakeup(10000000);
180.     // esp_light_sleep_start();
181.   }
182. }
183.
```

ANEXO A.2 Código de Programación del Nodo 2 para ESP32 en Arduino IDE

```
 1. // Incluye las bibliotecas necesarias para el funcionamiento de BLE y el sensor PZEM004Tv30
 2. #include <BLEDevice.h>
 3. #include <BLEServer.h>
 4. #include <BLEUtils.h>
 5. #include <BLE2902.h>
 6. #include <PZEM004Tv30.h>
 7.
 8. // Definición de pines
 9. PZEM004Tv30 pzem(Serial2, 16, 17); // Instancia del sensor PZEM usando Serial2.
10.
11. const int pinRele = 13; // Pin para el relé
12. int pinLed = 26; // Pin para el LED
13.
14. int pinHumo  = 4; // Pin para el sensor de humo
15. int estadoHumo = 0; // Estado actual del sensor de humo (0 = no hay humo, 1 = hay humo)
16.
17. // Variables para almacenar las mediciones del sensor PZEM
18. float voltaje = 0;
19. float corriente = 0;
20. float potencia = 0;
21. float energia = 0;
22. float frecuencia = 0;
23. float factorPotencia = 0;
24.
25. int numeroPaquete = 1; // Variable para controlar el paquete de datos a enviar vía BLE
```

```
26.
27. float energiaAnterior = 0; // Almacena la última medición de energía para comparaciones
28.
29. // Variables para control de tiempo
30. unsigned long tiempoActual = 0;
31. unsigned long tiempoAnterior = 0;
32.
33. // Variables para BLE
34. BLEServer *pServer = NULL;
35. BLECharacteristic *pCharacteristic = NULL;
36. bool dispositivoConectado = false;
37. bool dispositivoConectadoAnterior = false;
38.
39. // UUIDs para el servicio y la característica BLE
40. #define SERVICE_UUID        "4fafc201-1fb5-459e-8fcc-c5c9c331914c"
41. #define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"
42.
43. // Clase para manejar eventos de conexión y desconexión del servidor BLE
44. class MyServerCallbacks : public BLEServerCallbacks {
45.     void onConnect(BLEServer* pServer) {
46.       dispositivoConectado = true;
47.     };
48.
49.     void onDisconnect(BLEServer* pServer) {
50.       dispositivoConectado = false;
51.     }
52. };
53.
54. unsigned long millisAnterior2 = 0;
55. const unsigned long intervalo2 = 5000; // Intervalo para la verificación de humo
56.
57. void setup() {
58.   Serial.begin(115200); // Inicia la comunicación serial
59.
60.   // Configura los pines
61.   pinMode(pinRele, OUTPUT);
62.   pinMode(pinLed, OUTPUT);
63.   pinMode(pinHumo, INPUT);
64.   digitalWrite(pinRele,  1); // Inicializa el relé apagado
65.
66.   // Inicializa el servidor BLE con el nombre "ESP32_BLE_Server"
67.   BLEDevice::init("ESP32_BLE_Server");
68.   pServer = BLEDevice::createServer();
69.   pServer->setCallbacks(new MyServerCallbacks()); // Establece los callbacks para
conexión/desconexión
70.
71.   // Crea un servicio BLE
72.   BLEService *pService = pServer->createService(SERVICE_UUID);
73.
74.   // Crea una característica BLE en el servicio, con propiedades de lectura, notificación y
escritura
75.   pCharacteristic = pService->createCharacteristic(
76.                       CHARACTERISTIC_UUID,
77.                       BLECharacteristic::PROPERTY_READ |
78.                       BLECharacteristic::PROPERTY_NOTIFY |
79.                       BLECharacteristic::PROPERTY_WRITE
80.                     );
81.   pCharacteristic->addDescriptor(new BLE2902()); // Añade descriptor para notificaciones
82.
83.   pService->start(); // Inicia el servicio BLE
84.
```

```
85.   // Configura y comienza la publicidad del servidor BLE
86.   BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
87.   pAdvertising->addServiceUUID(SERVICE_UUID);
88.   pAdvertising->setScanResponse(true);
89.   pAdvertising->setMinPreferred(0x06);
90.   BLEDevice::startAdvertising();
91.   Serial.println("Esperando una conexión...");
92.
93.   tiempoAnterior = millis(); // Guarda el tiempo actual
94. }
95.
96. void loop() {
97.   // Si la energía medida es mayor a la anterior, enciende y apaga el LED rápidamente
98.   if (energia > energiaAnterior) {
99.     digitalWrite(pinLed, HIGH);  // Enciende el LED
100.    delay(50);                   // Espera 50 milisegundos
101.    digitalWrite(pinLed, LOW);    // Apaga el LED
102.  }
103.  energiaAnterior = energia; // Actualiza la última medición de energía
104.
105.  // Lee los valores del sensor PZEM
106.  voltaje = pzem.voltage();
107.  corriente = pzem.current();
108.  potencia = pzem.power();
109.  energia = pzem.energy();
110.  frecuencia = pzem.frequency();
111.  factorPotencia = pzem.pf();
112.
113.  // Prepara los datos a enviar mediante BLE, agrupándolos en tres paquetes diferentes
114.  String dato1 = "QWE" + String(voltaje) + "RTY" + String(corriente, 3)+ "UIO";
115.  String dato2 = "PAS" + String(potencia, 2) + "DFG" + String(energia, 3)+ "HJK";
116.  String dato3 = "LZX" + String(frecuencia) + "CVB" + String(factorPotencia)+ "NMQ" +
String(estadoHumo)+ "BTG";
117.
118.  String textoEnviar; // Variable para almacenar el texto a enviar
119.
120.  // Selecciona el paquete de datos a enviar
121.  if (numeroPaquete == 1) {
122.    textoEnviar = dato1;
123.  } else if (numeroPaquete == 2) {
124.    textoEnviar = dato2;
125.  } else if (numeroPaquete == 3) {
126.    textoEnviar = dato3;
127.  }
128.
129.  // Detecta cambio de estado en el sensor de humo y actualiza el estadoHumo
130.  if (digitalRead(pinHumo) == HIGH && estadoHumo == LOW) {
131.    estadoHumo = HIGH;
132.    millisAnterior2 = millis(); // Guarda el momento del cambio
133.  }
134.
135.  // Cambia estadoHumo a LOW después de 5 segundos sin detección de humo
136.  if (estadoHumo == HIGH && millis() - millisAnterior2 >= intervalo2) {
137.    estadoHumo = LOW;
138.  }
139.
140.  // Procesa los datos recibidos via BLE
141.  if (pCharacteristic->getValue().length() > 0) {
142.    std::string receivedData = pCharacteristic->getValue();
143.
144.    Serial.println(receivedData.length());
```

```
145.
146.     // Procesa los comandos recibidos para controlar el relé o reiniciar la energía del
PZEM
147.     if (receivedData.length() == 3 && receivedData[0] == 0x01 && receivedData[1] == 0xAB &&
receivedData[2] == 0xFF) {
148.       digitalWrite(pinRele, 0); // Enciende el relé
149.     }
150.
151.     if (receivedData.length() == 3 && receivedData[0] == 0x02 && receivedData[1] == 0xAB &&
receivedData[2] == 0xFF) {
152.       digitalWrite(pinRele, 1); // Apaga el relé
153.     }
154.
155.     if (receivedData.length() == 3 && receivedData[0] == 0x05 && receivedData[1] == 0xAB &&
receivedData[2] == 0xFF) {
156.       pzem.resetEnergy(); // Reinicia la energía medida por el PZEM
157.     }
158.
159.     pCharacteristic->setValue(""); // Limpia el valor de la característica después de
procesar el comando
160.   }
161.
162.   // Si hay un dispositivo conectado, gestiona la notificación de los datos
163.   if (dispositivoConectado) {
164.     tiempoActual = millis();
165.
166.     // Envía los datos cada 333ms
167.     if (tiempoActual >= tiempoAnterior + 333) {
168.       pCharacteristic->setValue(textoEnviar.c_str());
169.       pCharacteristic->notify();
170.
171.       numeroPaquete += 1; // Cambia al siguiente paquete de datos
172.       tiempoAnterior = tiempoActual; // Actualiza el tiempo para el próximo envío
173.     }
174.
175.     // Reinicia el número de paquete después de enviar los tres paquetes
176.     if (numeroPaquete == 4) {
177.       numeroPaquete = 1;
178.     }
179.   }
180.
181.   // Reinicia la publicidad BLE si un dispositivo se desconecta
182.   if (!dispositivoConectado && dispositivoConectadoAnterior) {
183.     delay(500); // Pequeña pausa antes de reiniciar la publicidad
184.     pServer->startAdvertising(); // Reinicia la publicidad
185.     Serial.println("Esperando una conexión...");
186.     dispositivoConectadoAnterior = dispositivoConectado;
187.   }
188.
189.   // Actualiza el estado del dispositivo conectado
190.   if (dispositivoConectado && !dispositivoConectadoAnterior) {
191.     dispositivoConectadoAnterior = dispositivoConectado;
192.   }
193. }
194.
```

ANEXO A.3 Código de Programación del Gateway Nodo 3 para ESP32 en Arduino IDE

```
1. #include "BLEDevice.h"
2.
```

```
// UUIDs del servicio y la característica a la que queremos conectarnos
static BLEUUID uuidServicio("4fafc201-1fb5-459e-8fcc-c5c9c331914c");
static BLEUUID uuidCaracteristica("beb5483e-36e1-4688-b7f5-ea07361b26a8");

// Variables de estado para la conexión y el escaneo
static boolean intentarConectar = false;
static boolean conectado = false;
static boolean intentarEscanear = true;
static BLERemoteCharacteristic* caracteristicaRemota;
static BLEAdvertisedDevice* miDispositivo;

// Variables para control de tiempo
unsigned long millisPrevios = 0;
const long intervalo = 5000;

// Mensajes divididos en líneas para ser reconstruidos
String lineaMensaje1, lineaMensaje2, lineaMensaje3;

// Callback cuando se recibe una notificación de la característica
static void callbackNotificaciones(
  BLERemoteCharacteristic* pCaracteristicaRemota,
  uint8_t* pData,
  size_t length,
  bool isNotify) {
    if (length > 0 && pData[0] != '\0') {
        String mensaje = String((char*)pData);
        if (mensaje.startsWith("QWE")) {
            lineaMensaje1 = mensaje;
        } else if (mensaje.startsWith("PAS")) {
            lineaMensaje2 = mensaje;
        } else if (mensaje.startsWith("LZX")) {
            lineaMensaje3 = mensaje;
        }

        // Procesamiento del mensaje combinado
        String mensajeCombinado = lineaMensaje1 + lineaMensaje2 + lineaMensaje3;
        // Función lambda para extraer subcadenas entre delimitadores
        auto extraerEntre = [](const String &str, const String &inicio, const String &fin)
-> String {
            int idxInicio = str.indexOf(inicio) + inicio.length();
            int idxFin = str.indexOf(fin, idxInicio);
            return idxFin != -1 ? str.substring(idxInicio, idxFin) : "";
        };

        // Ejemplo de cómo usar la función lambda `extraerEntre`
        String valorExtraido = extraerEntre(mensajeCombinado, "QWE", "RTY");
        Serial.println(valorExtraido);
    }
}

// Callbacks para el cliente BLE para manejar eventos de conexión y desconexión
class MiCallbackCliente : public BLEClientCallbacks {
  void onConnect(BLEClient* pcliente) {
    conectado = true;
    intentarEscanear = false;
    Serial.println("Conectado a BLE");
  }

  void onDisconnect(BLEClient* pcliente) {
    conectado = false;
    intentarEscanear = true;
```

```
63.     Serial.println("Desconectado de BLE, reiniciando escaneo...");
64.   }
65. };
66.
67. // Función para intentar conectar al servidor BLE
68. bool conectarAlServidor() {
69.     Serial.print("Formando una conexión a ");
70.     Serial.println(miDispositivo->getAddress().toString().c_str());
71.
72.     BLEClient* pCliente = BLEDevice::createClient();
73.     pCliente->setClientCallbacks(new MiCallbackCliente());
74.
75.     // Conecta al dispositivo BLE encontrado
76.     if (!pCliente->connect(miDispositivo)) {
77.         Serial.println("La conexión ha fallado");
78.         return false;
79.     }
80.
81.     // Obtiene el servicio BLE remoto
82.     BLERemoteService* pServicioRemoto = pCliente->getService(uuidServicio);
83.     if (pServicioRemoto == nullptr) {
84.         Serial.print("No se encontró el servicio UUID: ");
85.         Serial.println(uuidServicio.toString().c_str());
86.         pCliente->disconnect();
87.         return false;
88.     }
89.
90.     // Obtiene la característica BLE remota
91.     caracteristicaRemota = pServicioRemoto->getCharacteristic(uuidCaracteristica);
92.     if (caracteristicaRemota == nullptr) {
93.         Serial.print("No se encontró la característica UUID: ");
94.         Serial.println(uuidCaracteristica.toString().c_str());
95.         pCliente->disconnect();
96.         return false;
97.     }
98.
99.     // Registra el callback para notificaciones de la característica
100.     if (caracteristicaRemota->canNotify()) caracteristicaRemota-
>registerForNotify(callbackNotificaciones);
101.
102.     conectado = true;
103.     return true;
104. }
105.
106. // Callbacks para dispositivos anunciados durante el escaneo BLE
107. class MiCallbackDispositivoAnunciado: public BLEAdvertisedDeviceCallbacks {
108.     void onResult(BLEAdvertisedDevice advertisedDevice) {
109.       if (advertisedDevice.haveServiceUUID() &&
advertisedDevice.isAdvertisingService(uuidServicio)) {
110.         BLEDevice::getScan()->stop();
111.         miDispositivo = new BLEAdvertisedDevice(advertisedDevice);
112.         intentarConectar = true;
113.         intentarEscanear = false;
114.       }
115.     }
116. };
117.
118. void setup() {
119.   Serial.begin(115200);
120.   Serial.println("Iniciando cliente BLE...");
121.
```

```
122.   BLEDevice::init("");
123.   BLEScan* pBLEScan = BLEDevice::getScan();
124.   pBLEScan->setAdvertisedDeviceCallbacks(new MiCallbackDispositivoAnunciado());
125.   pBLEScan->setActiveScan(true);
126.   pBLEScan->start(5, false);
127. }
128.
129. void loop() {
130.   // Manejo de la conexión
131.   if (intentarConectar && !conectado) {
132.     if (conectarAlServidor()) {
133.       Serial.println("Conectado al servidor BLE");
134.       intentarConectar = false;
135.     } else {
136.       Serial.println("No se pudo conectar al servidor");
137.       intentarEscanear = true;
138.     }
139.   }
140.
141.   // Reinicia el escaneo si es necesario
142.   if (intentarEscanear && !conectado) {
143.     BLEDevice::getScan()->start(0); // 0 = escaneo continuo
144.   }
145. }
146.
```

ANEXO A.4 Código de Programación del Gateway Nodo 3 para ESP32 en Arduino IDE

```
  1. #include <PZEM004Tv30.h>
  2.
  3. // Si no se han definido los pines RX y TX para el PZEM, se establecen valores
predeterminados
  4. #if !defined(PZEM_RX_PIN) && !defined(PZEM_TX_PIN)
  5. #define PZEM_RX_PIN 4
  6. #define PZEM_TX_PIN 2
  7. #endif
  8.
  9. // Si no se ha definido el serial para el PZEM, se usa Serial1 como predeterminado
 10. #if !defined(PZEM_SERIAL)
 11. #define PZEM_SERIAL Serial1
 12. #endif
 13.
 14. // Configuración específica para el ESP32
 15. #if defined(ESP32)
 16. PZEM004Tv30 pzem(PZEM_SERIAL, PZEM_RX_PIN, PZEM_TX_PIN); // Instancia del PZEM usando
Serial1 y los pines definidos
 17. #elif defined(ESP8266)
 18. // Si se usa un ESP8266, la configuración del PZEM se simplifica ya que se puede conectar
directamente
 19. #else
 20. PZEM004Tv30 pzem(PZEM_SERIAL); // Instancia del PZEM para otros microcontroladores usando
Serial1
 21. #endif
 22.
 23. // Configuración de la comunicación con el SIM800L usando UART2
 24. HardwareSerial sim800(2);
 25. #define BAUD_RATE 9600 // Velocidad de baudios para la comunicación con el SIM800L
```

```
26.
27. // Declaración de pines
28. #define rxPin 16
29. #define txPin 17
30.
31. int pinLuz = 15; // Pin para el sensor de luz
32. int pinRele = 13; // Pin para el relé
33. int pinLed = 26; // Pin para el LED
34.
35. // Declaración de variables para almacenar mediciones del PZEM
36. float voltaje = 0.0;
37. float corriente = 0.0;
38. float potencia = 0.0;
39. float energia = 0.0;
40. float frecuencia = 0.0;
41. float factorPotencia = 0.0;
42. float energiaAnterior = 0; // Para almacenar la última medición de energía y detectar
cambios
43.
44. const String TELEFONO = "+593939236138"; // Número de teléfono para enviar SMS
45. unsigned long millisPrevios = 0; // Para controlar el envío de datos
46. const long intervalo = 30000; // Intervalo de 30 segundos para el envío de datos
47.
48. // Variables para almacenar la última medición válida de cada parámetro
49. float ultimoVoltajeValido = 0.0;
50. float ultimaCorrienteValida = 0.0;
51. float ultimaPotenciaValida = 0.0;
52. float ultimaEnergiaValida = 0.0;
53. float ultimaFrecuenciaValida = 0.0;
54. float ultimoFpValido = 0.0;
55.
56. unsigned long millisPrevios2 = 0;
57. const long intervalo2 = 5000; // Intervalo de 5 segundos para comprobar mensajes SMS
58.
59. // Configuración inicial
60. void setup() {
61.   pinMode(pinRele, OUTPUT);
62.   pinMode(pinLed, OUTPUT);
63.   pinMode(pinLuz, INPUT);
64.
65.   digitalWrite(pinRele, HIGH); // Inicialmente el relé está desactivado
66.
67.   Serial.begin(115200); // Inicia la comunicación Serial con la PC
68.
69.   // Inicia la comunicación con el SIM800L
70.   sim800.begin(BAUD_RATE, SERIAL_8N1, rxPin, txPin);
71.
72.   // Intenta conectarse con el SIM800L hasta que responda
73.   while (!esperarRespuestaSIM800L()) {
74.     Serial.println("No se pudo conectar con el SIM800L, reintentando...");
75.     delay(5000); // Espera 5 segundos antes de reintentar
76.   }
77.
78.   Serial.println("SIM800L listo y respondiendo.");
79.
80.   // Configura el SIM800L para el envío de SMS en modo texto y borra todos los mensajes
81.   sim800.print("AT+CMGF=1\r");
82.   delay(2000);
83.   sim800.print("AT+CMGDA=\"DEL ALL\"\r");
84.   delay(2000);
85. }
```

```
 86.
 87. // Bucle principal
 88. void loop() {
 89.   unsigned long millisActual = millis(); // Obtiene el tiempo actual
 90.
 91.   // Lee los valores del PZEM y almacena la última medición válida
 92.   voltaje = pzem.voltage();
 93.   corriente = pzem.current();
 94.   potencia = pzem.power();
 95.   energia = pzem.energy();
 96.   frecuencia = pzem.frequency();
 97.   pf = pzem.pf();
 98.
 99.   // Prepara los datos para enviar por SMS
100.   String datos = "V" + String(ultimoVoltajeValido) + "C" + String(ultimaCorrienteValida, 3)
+ "P" + String(ultimaPotenciaValida, 2) + "E" + String(ultimaEnergiaValida, 3) + "F" +
String(ultimaFrecuenciaValida) + "X" + String(ultimoFpValido) + "L" +
String((analogRead(pinLuz))) + "Z";
101.
102.   // Envia los datos por SMS cada intervalo establecido
103.   if (millisActual - millisPrevios >= intervalo) {
104.     Reply(datos); // Envia los datos como SMS
105.   }
106.
107.   // Comprueba si hay mensajes SMS no leídos cada intervalo2 y actúa en consecuencia
108.   if (millisActual - millisPrevios2 >= intervalo2) {
109.     envioDatos(); // Comprueba y actúa sobre mensajes SMS no leídos
110.   }
111.
112.   // Enciende el LED si la energía ha aumentado desde la última medición
113.   if (energia > energiaAnterior) {
114.     digitalWrite(pinLed, HIGH);
115.     delay(50);
116.     digitalWrite(pinLed, LOW);
117.   }
118.
119.   energiaAnterior = energia;
120. }
121.
122. // Espera una respuesta "OK" del SIM800L
123. bool esperarRespuestaSIM800L() {
124.   sim800.println("AT"); // Envia el comando AT al SIM800L
125.   long tiempoInicio = millis();
126.   while (millis() - tiempoInicio < 2000) { // Espera hasta 2 segundos por una respuesta
127.     if (sim800.available()) {
128.       String respuesta = sim800.readString();
129.       if (respuesta.indexOf("OK") >= 0) {
130.         return true; // El SIM800L respondió con "OK"
131.       }
132.     }
133.   }
134.   return false; // No se recibió respuesta o la respuesta no fue "OK"
135. }
136.
137. // Lee el texto entre marcadores específicos
138. String readTextBetweenMarkers(char endMarker) {
139.   String text = "";
140.   char c;
141.   while (sim800.available() && (c = sim800.read()) != endMarker) {
142.     text += c;
143.   }
```

```
144.   return text;
145. }
146.
147. // Controla el relé basado en el texto recibido
148. void controlRelay(String text) {
149.   int estadoRelay = text.toInt();
150.   if (estadoRelay == 1) {
151.     digitalWrite(rele, HIGH);
152.     Serial.println("Relé encendido");
153.   } else if (estadoRelay == 2) {
154.     digitalWrite(rele, LOW);
155.     Serial.println("Relé apagado");
156.   } else if (estadoRelay == 3) {
157.     pzem.resetEnergy();
158.     Serial.println("Energía reiniciada");
159.   } else {
160.     Serial.println("Texto no reconocido");
161.   }
162.   sim800.print("AT+CMGDA=\"DEL ALL\"\r"); // Borra todos los SMS después de actuar
163.   delay(1000);
164. }
165.
166. // Envía una respuesta por SMS
167. void Reply(String text) {
168.   sim800.print("AT+CMGF=1\r"); // Configura el modo de texto para SMS
169.   delay(1000);
170.   sim800.print("AT+CMGS=\"" + PHONE + "\"\r"); // Configura el número de destino
171.   delay(1000);
172.   sim800.print(text); // Escribe el texto del SMS
173.   delay(100);
174.   sim800.write(0x1A); // Envía el SMS
175.   delay(1000);
176.   Serial.println("SMS enviado correctamente.");
177.   millisPrevios = currentMillis; // Actualiza el tiempo para el próximo envío
178. }
179.
180. // Comprueba mensajes SMS no leídos y actúa según el contenido
181. void envioDatos() {
182.   sim800.print("AT+CMGL=\"REC UNREAD\"\r"); // Comando para listar SMS no leídos
183.   delay(2000);
184.   while (sim800.available()) {
185.     char c = sim800.read();
186.     Serial.write(c);
187.     if (c == '@') { // '@' es el marcador de inicio del contenido relevante
188.       String textoExtraido = readTextBetweenMarkers('#'); // '#' es el marcador de fin
189.       Serial.println("Texto extraído: " + textoExtraido);
190.       controlRelay(textoExtraido); // Actúa basado en el texto extraído
191.     }
192.   }
193.   millisPrevios2 = currentMillis2; // Actualiza el tiempo para la próxima comprobación
194. }
195.
```

ANEXO A.5 Código de Programación del Gateway Nodo 3 para ESP32 en Arduino IDE

```
1. #include "Adafruit_FONA.h"
2.
```

```
// Definición de pines para la comunicación y control del SIM800L
#define SIM800L_RX     27
#define SIM800L_TX     26
#define SIM800L_PWRKEY 4
#define SIM800L_RST    5
#define SIM800L_POWER  23

char bufferRespuesta[255]; // Almacena respuestas del SIM800L

HardwareSerial *serialSim800l = &Serial1; // Objeto para comunicación serial con el SIM800L
Adafruit_FONA sim800l = Adafruit_FONA(SIM800L_PWRKEY); // Instancia del SIM800L

uint8_t leerLinea(char *buff, uint8_t maxbuff, uint16_t timeout = 0); // Prototipo para función de lectura de línea

// Definiciones para el LED azul y el relé
#define LED_AZUL  13
#define RELE 14
const unsigned long TIEMPO_ESPERA = 60000; // Tiempo de espera en milisegundos

String cadenaSms = ""; // Almacena el SMS a enviar

// Función para leer texto entre marcadores específicos
String leerTextoEntreMarcadores(char marcadorInicio, char marcadorFin) {
  String texto = "";
  char c;

  // Leer caracteres hasta encontrar el marcador de inicio
  while (sim800l.available() && (c = sim800l.read()) != marcadorInicio);

  // Leer caracteres hasta encontrar el marcador de final
  while (sim800l.available() && (c = sim800l.read()) != marcadorFin) {
    texto += c;
  }

  return texto;
}

void setup()
{
  pinMode(LED_AZUL, OUTPUT);
  pinMode(RELE, OUTPUT);
  pinMode(SIM800L_POWER, OUTPUT);

  digitalWrite(LED_AZUL, LOW); // Apagar LED al inicio
  digitalWrite(SIM800L_POWER, HIGH); // Encender el módulo SIM800L

  Serial.begin(115200);
  Serial.println(F("ESP32 con GSM SIM800L"));
  Serial.println(F("Inicializando....(Puede tardar más de 10 segundos)"));

  delay(10000); // Esperar a que el módulo esté listo

  // Configuración de la comunicación serial a baja velocidad para facilitar la lectura
  serialSim800l->begin(4800, SERIAL_8N1, SIM800L_TX, SIM800L_RX);
  if (!sim800l.begin(*serialSim800l)) {
    Serial.println(F("No se encontró el GSM SIM800L"));
    while (1);
  }
  Serial.println(F("GSM SIM800L está OK"));

```

```
  char imei[16] = {0}; // Buffer para el IMEI del SIM
  uint8_t longitudImei = sim800l.getIMEI(imei);
  if (longitudImei > 0) {
    Serial.print("IMEI de la tarjeta SIM: "); Serial.println(imei);
  }

  // Configurar el FONA para enviar una notificación +CMTI cuando se recibe un SMS
  serialSim800l->print("AT+CNMI=2,2,0,0,0\r");
  serialSim800l->print("AT+CMGF=1\r");
  serialSim800l->print("AT+CMGDA=\"DEL ALL\"\r");

  Serial.println("GSM SIM800L Listo");
}

long prevMillis = 0;
int intervalo = 1000;
char bufferNotificacionesSim800l[64]; // Para notificaciones del FONA
char bufferSms[250];
boolean estadoLed = false;

void loop()
{
  unsigned long tiempoInicio = millis();

  // Esperar por datos entrantes o hasta que se cumpla el tiempo de espera
  while (!sim800l.available()) {
    if (millis() - tiempoInicio > TIEMPO_ESPERA) {
      Serial.println("Tiempo de espera superado. Reiniciando...");
      esp_restart(); // Reiniciar el ESP32
    }
  }

  String texto = "";

  // Leer datos enviados al ESP32 vía serial
  while (Serial.available()) {
    char c = Serial.read();
    texto += c;
  }

  if (texto.length() > 0) {


```

ANEXO A.6 Código de Programación del Nodo 4 para ESP32 en Arduino IDE

```
#include <SPI.h>
#include "RF24.h"
#include <PZEM004Tv30.h>

RF24 radio(4, 5); // Pines CE y CSN
const byte direccion[6] = "00001"; // Dirección del canal de comunicación
char mensajeRecibido[64]; // Buffer para almacenar el mensaje recibido

// Instancia del sensor PZEM-004T
PZEM004Tv30 pzem(Serial2, 16, 17);

// Definición de pines
int rele = 13;
```

```
14. int led = 26;
15. int buzzer = 15;
16. 
17. // Variables para almacenar las mediciones del PZEM-004T
18. float voltaje = 0;
19. float corriente = 0;
20. float potencia = 0;
21. float energia = 0;
22. float frecuencia = 0;
23. float factorPotencia = 0;
24. 
25. float energiaAnterior = 0; // Almacena la última medición de energía para detectar cambios
26. 
27. // Variables para control de tiempo
28. unsigned long tiempoActual = 0;
29. unsigned long tiempoAnterior = 0;
30. const long intervalo = 1000; // Intervalo de tiempo para acciones repetitivas
31. 
32. int paquete = 1; // Variable para alternar entre los paquetes de datos a enviar
33. 
34. void setup() {
35.     Serial.begin(115200);
36.     while (!Serial) {
37.         // Espera a que el puerto serial esté disponible (necesario para algunas placas)
38.     }
39. 
40.     // Configura los pines
41.     pinMode(rele, OUTPUT);
42.     pinMode(led, OUTPUT);
43.     pinMode(buzzer, OUTPUT);
44.     digitalWrite(rele, HIGH); // Inicia con el relé desactivado
45.     digitalWrite(buzzer, LOW); // Inicia con el buzzer desactivado
46. 
47.     // Inicializa el módulo RF24
48.     if (!radio.begin()) {
49.         Serial.println(F("El hardware del radio no responde!!"));
50.         while (1); // Bucle infinito si el radio no responde
51.     }
52. 
53.     // Configura el radio
54.     radio.openWritingPipe(direccion);
55.     radio.openReadingPipe(1, direccion);
56.     radio.setPALevel(RF24_PA_LOW);
57.     radio.startListening(); // Comienza a escuchar mensajes entrantes
58. }
59. 
60. void loop() {
61.     tiempoActual = millis(); // Obtiene el tiempo actual
62. 
63.     // Lee los valores del sensor PZEM
64.     voltaje = pzem.voltage();
65.     corriente = pzem.current();
66.     potencia = pzem.power();
67.     energia = pzem.energy();
68.     frecuencia = pzem.frequency();
69.     factorPotencia = pzem.pf();
70. 
71.     // Prepara los textos a enviar con los datos medidos
72.     String texto1 = "V" + String(voltaje) + "C" + String(corriente, 3) + "P";
73.     String texto2 = "P" + String(potencia, 2) + "E" + String(energia, 3) + "F";
74.     String texto3 = "F" + String(frecuencia) + "X" + String(factorPotencia) + "M";
```

```
75.
76.     String textoN; // Texto final a enviar
77.
78.     // Selecciona el texto a enviar basado en la variable paquete
79.     if (paquete == 1) {
80.         textoN = texto1;
81.     } else if (paquete == 2) {
82.         textoN = texto2;
83.     } else if (paquete == 3) {
84.         textoN = texto3;
85.     }
86.
87.     // Envia los datos cada intervalo especificado
88.     if (tiempoActual - tiempoAnterior >= intervalo) {
89.         radio.stopListening(); // Detiene la escucha para poder enviar
90.         radio.write(&textoN, sizeof(textoN)); // Envía el texto seleccionado
91.         radio.startListening(); // Reanuda la escucha
92.
93.         paquete += 1; // Cambia al siguiente paquete
94.         tiempoAnterior = tiempoActual; // Actualiza el tiempo para el próximo envío
95.     }
96.
97.     // Reinicia la secuencia de paquetes
98.     if (paquete == 4) {
99.         paquete = 1;
100.    }
101.
102.    // Lee mensajes del puerto serial y los envía a través del RF24
103.    if (Serial.available()) {
104.        String texto = Serial.readString();
105.        radio.stopListening();
106.        radio.write(&texto, sizeof(texto));
107.        radio.startListening();
108.    }
109.
110.    // Procesa los mensajes recibidos a través del RF24
111.    if (radio.available()) {
112.        memset(mensajeRecibido, 0, sizeof(mensajeRecibido)); // Limpia el buffer
113.        radio.read(&mensajeRecibido, sizeof(mensajeRecibido)); // Lee el mensaje
114.
115.        Serial.println(mensajeRecibido); // Imprime el mensaje recibido
116.
117.        // Ejecuta acciones basadas en el mensaje
118.        if (strcmp(mensajeRecibido, "1") == 0) {
119.            digitalWrite(rele, HIGH);
120.        } else if (strcmp(mensajeRecibido, "0") == 0) {
121.            digitalWrite(rele, LOW);
122.        } else if (strcmp(mensajeRecibido, "2") == 0) {
123.            pzem.resetEnergy();
124.        } else if (strcmp(mensajeRecibido, "3") == 0) {
125.            digitalWrite(buzzer, HIGH);
126.        } else if (strcmp(mensajeRecibido, "4") == 0) {
127.            digitalWrite(buzzer, LOW);
128.        }
129.    }
130.
131.    // Acciones basadas en la comparación de la energía medida
132.    if (energia > energiaAnterior) {
133.        digitalWrite(led, HIGH); // Enciende el LED brevemente
134.        delay(50); // Espera 50 milisegundos
135.        digitalWrite(led, LOW); // Apaga el LED
```

```
136.     }
137.
138.     energiaAnterior = energia; // Actualiza la última medición de energía
139. }
140.
```

ANEXO A.7 Código de Programación del Gateway Nodo 4 para Arduino Nano en Arduino IDE

```
  1. #include <SPI.h>
  2. #include <RF24.h>
  3. #include <ArduinoJson.h>
  4.
  5. RF24 radio(9, 10); // Pines CE, CSN para el módulo RF24
  6. const byte direccion[6] = "00001"; // Dirección del canal de comunicación
  7. char mensajeRecibido[64]; // Buffer para almacenar el mensaje recibido
  8.
  9. unsigned long ultimoTiempoMensajeValido = 0; // Variable para almacenar el tiempo del
último mensaje válido
 10.
 11. void setup() {
 12.     Serial.begin(115200); // Inicia la comunicación serial
 13.     radio.begin(); // Inicia el módulo RF24
 14.     radio.openWritingPipe(direccion); // Abre un canal de escritura con la dirección
especificada
 15.     radio.openReadingPipe(1, direccion); // Abre un canal de lectura con la dirección
especificada
 16.     radio.setPALevel(RF24_PA_MAX); // Establece el nivel de potencia del módulo RF24 al
máximo
 17.     radio.startListening(); // Comienza a escuchar mensajes entrantes
 18. }
 19.
 20. void loop() {
 21.     ultimoTiempoMensajeValido = millis(); // Actualiza el tiempo del último mensaje válido
 22.
 23.     if (Serial.available()) { // Verifica si hay datos disponibles para leer en el puerto
serial
 24.         String texto = Serial.readString(); // Lee el texto enviado a través del puerto
serial
 25.         radio.stopListening(); // Detiene la escucha para poder enviar datos
 26.         radio.write(&texto, sizeof(texto)); // Envia el texto leído a través del módulo
RF24
 27.         radio.startListening(); // Vuelve a comenzar la escucha de mensajes entrantes
 28.     }
 29.
 30.     if (radio.available()) { // Verifica si hay mensajes disponibles para leer en el módulo
RF24
 31.         memset(mensajeRecibido, 0, sizeof(mensajeRecibido)); // Limpia el buffer antes de
leer un nuevo mensaje
 32.         radio.read(&mensajeRecibido, sizeof(mensajeRecibido)); // Lee el mensaje entrante
 33.
 34.         // Procesa el mensaje recibido para eliminar caracteres de salto de línea
 35.         for (int i = 0; i < sizeof(mensajeRecibido); i++) {
 36.             if (mensajeRecibido[i] == '\n' || mensajeRecibido[i] == '\r') {
 37.                 mensajeRecibido[i] = '\0'; // Reemplaza saltos de línea con terminadores de
cadena
 38.                 break; // Sale del bucle al encontrar el primer salto de línea
 39.             }
```

```
 40.         }
 41.
 42.         DynamicJsonDocument jsonBuffer(256); // Crea un buffer para el objeto JSON de
tamaño 256
 43.         auto root = jsonBuffer.to<JsonObject>(); // Crea el objeto JSON raíz
 44.
 45.         // Procesamiento de los valores etiquetados en el mensaje recibido para asignarlos
al objeto JSON
 46.         // Utiliza la función strchr para buscar el inicio de cada etiqueta en el mensaje
 47.         // Luego extrae el valor como subcadena y lo convierte a flotante
 48.         // Finalmente, asigna cada valor al objeto JSON usando la etiqueta como clave
 49.
 50.         // Ejemplo para la etiqueta 'V' (voltaje)
 51.         char *inicioEtiqueta = strchr(mensajeRecibido, 'V');
 52.         if (inicioEtiqueta) {
 53.             char *finEtiqueta = strchr(inicioEtiqueta, 'C');
 54.             if (finEtiqueta) {
 55.                 String valorStr = String(inicioEtiqueta + 1, finEtiqueta - inicioEtiqueta -
1);
 56.                 root["V"] = valorStr.toFloat();
 57.             }
 58.         }
 59.
 60. / Procesar etiqueta 'C'
 61.         startPtr = strchr(receivedMessage, 'C');
 62.         if (startPtr != NULL) {
 63.             startIndex = startPtr - receivedMessage + 1; // +1 para empezar después de 'C'
 64.             endPtr = strchr(startPtr, 'P');
 65.             if (endPtr != NULL) {
 66.                 endIndex = endPtr - receivedMessage; // Busca el final del número antes de
'P'
 67.                 valueStr = String(receivedMessage).substring(startIndex, endIndex); //
Extrae la subcadena
 68.                 value = valueStr.toFloat(); // Convierte la subcadena a float
 69.                 root["C"] = value;
 70.             }
 71.         }
 72.
 73.         // Procesar etiqueta 'P'
 74.         startPtr = strchr(receivedMessage, 'P');
 75.         if (startPtr != NULL) {
 76.             startIndex = startPtr - receivedMessage + 1; // +1 para empezar después de 'P'
 77.             endPtr = strchr(startPtr, 'E');
 78.             if (endPtr != NULL) {
 79.                 endIndex = endPtr - receivedMessage; // Busca el final del número antes de
'E'
 80.                 valueStr = String(receivedMessage).substring(startIndex, endIndex); //
Extrae la subcadena
 81.                 value = valueStr.toFloat(); // Convierte la subcadena a float
 82.                 root["P"] = value;
 83.             }
 84.         }
 85.
 86.         // Procesar etiqueta 'E'
 87.         startPtr = strchr(receivedMessage, 'E');
 88.         if (startPtr != NULL) {
 89.             startIndex = startPtr - receivedMessage + 1; // +1 para empezar después de 'E'
 90.             endPtr = strchr(startPtr, 'F');
 91.             if (endPtr != NULL) {
 92.                 endIndex = endPtr - receivedMessage; // Busca el final del número antes de
'F'
```

```
                valueStr = String(receivedMessage).substring(startIndex, endIndex); // Extrae la subcadena
                value = valueStr.toFloat(); // Convierte la subcadena a float
                root["E"] = value;
            }
        }

        // Procesar etiqueta 'F'
        startPtr = strchr(receivedMessage, 'F');
        if (startPtr != NULL) {
            startIndex = startPtr - receivedMessage + 1; // +1 para empezar después de 'F'
            endPtr = strchr(startPtr, 'X');
            if (endPtr != NULL) {
                endIndex = endPtr - receivedMessage; // Busca el final del número antes de 'X'
                valueStr = String(receivedMessage).substring(startIndex, endIndex); // Extrae la subcadena
                value = valueStr.toFloat(); // Convierte la subcadena a float
                root["F"] = value;
            }
        }

        // Procesar etiqueta 'X'
        startPtr = strchr(receivedMessage, 'X');
        if (startPtr != NULL) {
            startIndex = startPtr - receivedMessage + 1; // +1 para empezar después de 'X'
            endPtr = strchr(startPtr, 'M');
            if (endPtr != NULL) {
                endIndex = endPtr - receivedMessage; // Busca el final del número antes de 'M'
                valueStr = String(receivedMessage).substring(startIndex, endIndex); // Extrae la subcadena
                value = valueStr.toFloat(); // Convierte la subcadena a float
                root["X"] = value;
            }

        // Serializa el objeto JSON y lo envía por el puerto serial
        serializeJson(root, Serial);
        Serial.println(); // Imprime una línea en blanco después del JSON para separar los mensajes
    }
```

ANEXO A.8 Código de Programación del Nodo 5 para ESP32 en Arduino IDE

```
#include <PZEM004Tv30.h>

// Verifica si los pines RX y TX del PZEM están definidos, si no, los define más adelante
#if !defined(PZEM_RX_PIN) && !defined(PZEM_TX_PIN)
#endif

// Verifica si el serial para comunicarse con el PZEM está definido, si no, lo define
#if !defined(PZEM_SERIAL)
#define PZEM_SERIAL Serial1 // Utiliza Serial1 como predeterminado para la comunicación
#endif

// Configuración específica para ESP32
#if defined(ESP32)
#define PZEM_RX_PIN 4 // Define el pin RX para la comunicación con el PZEM
#define PZEM_TX_PIN 2 // Define el pin TX para la comunicación con el PZEM
PZEM004Tv30 pzem(PZEM_SERIAL, PZEM_RX_PIN, PZEM_TX_PIN); // Inicializa el PZEM con los pines RX y TX
#elif defined(ESP8266)
```

```
18. PZEM004Tv30 pzem(PZEM_SERIAL); // Para ESP8266, no es necesario especificar los pines
19. #endif
20.
21. #define RXp2 16 // Define el pin RX para Serial2
22. #define TXp2 17 // Define el pin TX para Serial2
23.
24. String mensajeExtraido; // Almacena el mensaje extraído de la entrada serial
25.
26. int pinPIR = 15; // Define el pin para el sensor PIR
27. int estadoPIR = 0; // Almacena el estado del sensor PIR
28.
29. int pinRele = 13; // Define el pin para el relé
30. int pinLed = 26; // Define el pin para el LED
31.
32. // Variables para almacenar las mediciones del PZEM
33. float voltaje = 0.0;
34. float corriente = 0.0;
35. float potencia = 0.0;
36. float energia = 0.0;
37. float frecuencia = 0.0;
38. float factorPotencia = 0.0;
39.
40. float energiaAnterior = 0; // Almacena la última medición de energía
41.
42. unsigned long millisPrevios = 0; // Almacena el tiempo del último cambio de estado del LED
43. const long intervalo = 1000; // Intervalo de tiempo para acciones repetitivas
44.
45. unsigned long millisPrevios2 = 0;
46. const unsigned long intervalo2 = 5000; // Intervalo de tiempo para verificar el estado del
sensor PIR
47.
48. void setup() {
49.   pinMode(pinRele, OUTPUT);
50.   pinMode(pinLed, OUTPUT);
51.   pinMode(pinPIR, INPUT);
52.
53.   digitalWrite(pinRele, HIGH); // Inicia con el relé desactivado
54.
55.   Serial.begin(115200); // Inicia la comunicación serial a 115200 baudios
56.   Serial2.begin(115200, SERIAL_8N1, RXp2, TXp2); // Inicia Serial2 con configuración 8N1
57.
58.   Serial2.print("AT+RESET\r\n"); // Envía el comando AT para reiniciar el dispositivo
conectado a Serial2
59.
60.   delay(1000); // Espera 1000 ms para que el comando se ejecute
61. }
62.
63. void loop() {
64.   unsigned long millisActual = millis(); // Obtiene el tiempo actual
65.   // Lee los valores del sensor PZEM
66.   voltaje = pzem.voltage();
67.   corriente = pzem.current();
68.   potencia = pzem.power();
69.   energia = pzem.energy();
70.   frecuencia = pzem.frequency();
71.   factorPotencia = pzem.pf();
72.
73.   // Verifica el estado del sensor PIR
74.   if (digitalRead(pinPIR) == HIGH && estadoPIR == LOW) {
75.     estadoPIR = HIGH; // Cambia el estado a HIGH
76.     millisPrevios2 = millis(); // Guarda el momento en que se detectó movimiento
```

```
 77.   }
 78.
 79.   // Verifica si han pasado 5 segundos desde la última detección de movimiento
 80.   if (estadoPIR == HIGH && millis() - millisPrevios2 >= intervalo2) {
 81.     estadoPIR = LOW; // Cambia el estado a LOW después de 5 segundos sin detección
 82.   }
 83.
 84.   // Procesa los datos recibidos por Serial2
 85.   processSerialData();
 86.
 87.   // Envía los datos cada 1 segundo (1000 ms)
 88.   if (millisActual - millisPrevios >= intervalo) {
 89.     // Prepara los datos a enviar
 90.     String datos = "VV" + String(voltaje) + "CC" + String(corriente, 3) + "PP" +
String(potencia, 2) + "EE" + String(energia, 3) + "FF" + String(frecuencia) + "XX" +
String(factorPotencia) + "MM" + String(estadoPIR) + "ZZ";
 91.     int direccion = 0; // Dirección ficticia, reemplazar según sea necesario
 92.     int longitud = datos.length(); // Obtiene la longitud de los datos a enviar
 93.     String comandoAT = "AT+SEND=" + String(direccion) + "," + String(longitud) + "," +
datos + "\r\n";
 94.     Serial2.print(comandoAT); // Envía el comando AT con los datos a través de Serial2
 95.     millisPrevios = millisActual;
 96.   }
 97.
 98.   // Verifica si la energía medida ha aumentado
 99.   if (energia > energiaAnterior) {
100.     digitalWrite(pinLed, HIGH); // Enciende el LED
101.     delay(50); // Espera 50 milisegundos
102.     digitalWrite(pinLed, LOW); // Apaga el LED
103.   }
104.
105.   energiaAnterior = energia; // Actualiza la medición anterior de energía
106. }
107.
108. // Procesa los datos recibidos por Serial2 y ejecuta acciones basadas en el mensaje
109. void processSerialData() {
110.     if (Serial2.available()) {
111.         String datos = Serial2.readString(); // Lee todos los datos disponibles de Serial2
112.         Serial.print(datos); // Escribe los datos en el Serial
113.
114.         // Busca el inicio y el final del mensaje extraído
115.         int inicioIndex = datos.indexOf('J') + 1;
116.         int finalIndex = datos.indexOf('K', inicioIndex);
117.         if (inicioIndex > 0 && finalIndex > inicioIndex) {
118.             String mensajeExtraido = datos.substring(inicioIndex, finalIndex);
119.
120.             // Ejecuta acciones basadas en el mensaje extraído
121.             if (mensajeExtraido == "ERELE") {
122.                 digitalWrite(pinRele, HIGH); // Enciende el relé
123.             } else if (mensajeExtraido == "ARELE") {
124.                 digitalWrite(pinRele, LOW); // Apaga el relé
125.             } else if (mensajeExtraido == "ENERG") {
126.                 pzem.resetEnergy(); // Reinicia la medición de energía del PZEM
127.             }
128.         }
129.     }
130. }
```

ANEXO A.9 Código de Programación del Gateway Nodo 5 para ESP32 en Arduino IDE

```
1.  1. #include <HardwareSerial.h> // Incluimos la biblioteca para manejar los puertos seriales de hardware
 2.  2.
 3.  3. HardwareSerial RYLR998(2); // Declaramos RYLR998 utilizando el UART2
 4.  4.
 5.  5. void setup() {
 6.  6.   Serial.begin(115200); // Inicializamos la comunicación Serial (USB) a 115200 baudios
 7.  7.   RYLR998.begin(115200); // Inicializamos RYLR998 (UART2) a 115200 baudios
 8.  8. }
 9.  9.
10. 10. void loop() {
11. 11.   if (RYLR998.available()) { // Verificamos si hay datos disponibles en UART2 (RYLR998)
12. 12.     Serial.write(RYLR998.read()); // Leemos un byte de UART2 (RYLR998) y lo enviamos al Serial (USB)
13. 13.     Serial.println(); // Añadimos un salto de línea para mejorar la legibilidad
14. 14.   }
15. 15.
16. 16.   if (Serial.available()) { // Verificamos si hay datos disponibles en el Serial (USB)
17. 17.     RYLR998.write(Serial.read()); // Leemos un byte del Serial (USB) y lo enviamos a UART2 (RYLR998)
18. 18.   }
19. 19. }
```

ANEXO A.10 Código de Programación del Nodo 6 para ESP32 en Arduino IDE

```
 1. #include <PZEM004Tv30.h> // Incluye la biblioteca para trabajar con el módulo PZEM-004T
 2.
 3. // Define los pines RX y TX para la comunicación con el PZEM-004T si aún no están definidos
 4. #if !defined(PZEM_RX_PIN) && !defined(PZEM_TX_PIN)
 5. #endif
 6.
 7. // Usa Serial1 para la comunicación con el PZEM si PZEM_SERIAL aún no está definido
 8. #if !defined(PZEM_SERIAL)
 9. #define PZEM_SERIAL Serial1
10. #endif
11.
12. // Configuración específica para ESP32
13. #if defined(ESP32)
14. #define PZEM_RX_PIN 4 // Pin RX para el PZEM
15. #define PZEM_TX_PIN 2 // Pin TX para el PZEM
16.
17. String datos; // Variable para almacenar los datos recibidos por Serial2
18.
19. PZEM004Tv30 pzem(PZEM_SERIAL, PZEM_RX_PIN, PZEM_TX_PIN); // Inicializa el PZEM-004T con Serial1
20. #elif defined(ESP8266)
21. PZEM004Tv30 pzem(PZEM_SERIAL); // Inicializa el PZEM-004T con Serial para ESP8266
22. #endif
23.
24. #define RXp2 16 // Pin RX para Serial2
25. #define TXp2 17 // Pin TX para Serial2
26.
```

```
int pinGas = 15; // Pin para el sensor de gas

const int pinRele = 13; // Pin para el relé

int pinLed = 12; // Pin para el LED

// Variables para almacenar las mediciones del PZEM-004T
float voltaje = 0.0;
float corriente = 0.0;
float potencia = 0.0;
float energia = 0.0;
float frecuencia = 0.0;
float factorPotencia = 0.0;

float energiaAnterior = 0; // Almacena la última medición de energía

unsigned long millisPrevios = 0; // Almacena el tiempo del último cambio de estado del LED
const long intervalo = 1000; // Intervalo de tiempo para el envío de datos

void setup() {
  pinMode(pinRele, OUTPUT); // Configura el pin del relé como salida
  pinMode(pinLed, OUTPUT); // Configura el pin del LED como salida
  pinMode(pinGas, INPUT); // Configura el pin del sensor de gas como entrada

  digitalWrite(pinRele, HIGH); // Inicia con el relé apagado

  Serial.begin(115200); // Inicia la comunicación Serial (USB) a 115200 baudios
  Serial2.begin(9600, SERIAL_8N1, RXp2, TXp2); // Inicia Serial2 a 9600 baudios
}

void loop() {
  unsigned long millisActual = millis(); // Obtiene el tiempo actual
  // Lee los valores del sensor PZEM
  voltaje = pzem.voltage();
  corriente = pzem.current();
  potencia = pzem.power();
  energia = pzem.energy();
  frecuencia = pzem.frequency();
  factorPotencia = pzem.pf();

  // Procesa los datos recibidos por Serial2
  if (Serial2.available()) {
    datos = Serial2.readString(); // Lee los datos como una cadena
    Serial.print(datos); // Escribe los datos en Serial
  }
  datos.trim(); // Elimina espacios en blanco al inicio y al final de la cadena

  // Ejecuta acciones basadas en los comandos recibidos
  if (datos == "1") {
    digitalWrite(pinRele, HIGH); // Enciende el relé
  } else if (datos == "0") {
    digitalWrite(pinRele, LOW); // Apaga el relé
  }

  if (datos == "3") {
    pzem.resetEnergy(); // Reinicia la medición de energía del PZEM
  }

  // Envía los datos cada intervalo especificado
  if (millisActual - millisPrevios >= intervalo) {
```

```
87.     String datos2 = "V" + String(voltaje) + "C" + String(corriente, 3) + "P" +
String(potencia, 2) + "E" + String(energia, 3) + "F" + String(frecuencia) + "X" +
String(factorPotencia) + "M" + String((analogRead(pinGas))) + "Z";
88.
89.     Serial2.println(datos2); // Envía los datos a través de Serial2
90.     millisPrevios = millisActual;
91.   }
92.
93.   // Enciende el LED si la energía ha aumentado desde la última medición
94.   if (energia > energiaAnterior) {
95.     digitalWrite(pinLed, HIGH); // Enciende el LED
96.     delay(50); // Espera 50 milisegundos
97.     digitalWrite(pinLed, LOW); // Apaga el LED
98.   }
99.
100.  energiaAnterior = energia; // Actualiza la última medición de energía
101. }
```

ANEXO A.11 Código de Programación del Nodo 7 para ESP32 en Arduino IDE

```
1. #include <PZEM004Tv30.h>
2.
3. // Verifica y define los pines RX y TX para la comunicación con el PZEM si no se han
definido previamente
4. #if !defined(PZEM_RX_PIN) && !defined(PZEM_TX_PIN)
5. #define PZEM_RX_PIN 4
6. #define PZEM_TX_PIN 2
7. #endif
8.
9. // Define el Serial que se usará para comunicarse con el PZEM si no se ha definido
previamente
10. #if !defined(PZEM_SERIAL)
11. #define PZEM_SERIAL Serial1
12. #endif
13.
14. // Configuración específica para ESP32 y ESP8266
15. #if defined(ESP32)
16. PZEM004Tv30 pzem(PZEM_SERIAL, PZEM_RX_PIN, PZEM_TX_PIN); // Inicializa el PZEM usando los
pines definidos para ESP32
17. #elif defined(ESP8266)
18. PZEM004Tv30 pzem(PZEM_SERIAL); // Inicializa el PZEM para ESP8266 (los pines se definen de
manera diferente)
19. #endif
20.
21. // Define los pines para la comunicación Serial2 con otro dispositivo (por ejemplo, un
módulo SIM800L)
22. #define RXp2 17
23. #define TXp2 16
24.
25. // Declaración de pines y variables
26. int pinMicro  = 15; // Pin para el sensor (nombrado "micro")
27. int estadoMicro = 0; // Variable para almacenar el estado actual del sensor
28.
29. const int pinRele = 13; // Pin para el relé
30. const int pinEstadoRele = 12; // Pin para leer el estado del relé (no se usa en este
código)
31. const int pinResetEnergia = 14; // Pin para resetear la energía medida por el PZEM
32.
33. int pinLed = 26; // Pin para el LED
```

```
34.
35. // Variables para almacenar las mediciones del PZEM
36. float voltaje = 0.0;
37. float corriente = 0.0;
38. float potencia = 0.0;
39. float energia = 0.0;
40. float frecuencia = 0.0;
41. float factorPotencia = 0.0;
42.
43. float energiaAnterior = 0; // Almacena la última medición de energía para comparar cambios
44.
45. unsigned long millisPrevios = 0; // Almacena el tiempo del último cambio de estado del LED
46. const long intervalo = 1000; // Intervalo de tiempo para acciones repetitivas
(originalmente configurado incorrectamente)
47.
48. unsigned long millisPrevios2 = 0;
49. const unsigned long intervalo2 = 5000; // Intervalo de tiempo para verificar el estado del
sensor "micro"
50.
51. void setup() {
52.   // Configuración de pines
53.   pinMode(pinRele, OUTPUT);
54.   pinMode(pinLed, OUTPUT);
55.   pinMode(pinMicro, INPUT);
56.   pinMode(pinEstadoRele, INPUT);
57.   pinMode(pinResetEnergia, INPUT);
58.
59.   digitalWrite(pinRele, HIGH); // Inicialmente el relé está desactivado
60.
61.   // Inicia la comunicación Serial
62.   Serial.begin(115200);
63.   Serial2.begin(115200, SERIAL_8N1, RXp2, TXp2);
64. }
65.
66. void loop() {
67.   unsigned long millisActual = millis(); // Obtiene el tiempo actual
68.   // Lee los valores del sensor PZEM
69.   voltaje = pzem.voltage();
70.   corriente = pzem.current();
71.   potencia = pzem.power();
72.   energia = pzem.energy();
73.   frecuencia = pzem.frequency();
74.   factorPotencia = pzem.pf();
75.
76.   // Verifica el estado del sensor "micro"
77.   if (digitalRead(pinMicro) == HIGH && estadoMicro == LOW) {
78.     estadoMicro = HIGH; // Cambia el estado a HIGH
79.     millisPrevios2 = millis(); // Guarda el momento actual
80.   }
81.
82.   // Verifica si han pasado 5 segundos desde la última detección
83.   if (estadoMicro == HIGH && millis() - millisPrevios2 >= intervalo2) {
84.     estadoMicro = LOW; // Cambia el estado a LOW después de 5 segundos
85.   }
86.
87.   // Controla el relé basado en el estado de un pin
88.   if (digitalRead(pinEstadoRele) == HIGH) {
89.     digitalWrite(pinRele, LOW); // Enciende el relé
90.   } else if (digitalRead(pinEstadoRele) == LOW) {
91.     digitalWrite(pinRele, HIGH); // Apaga el relé
92.   }
```

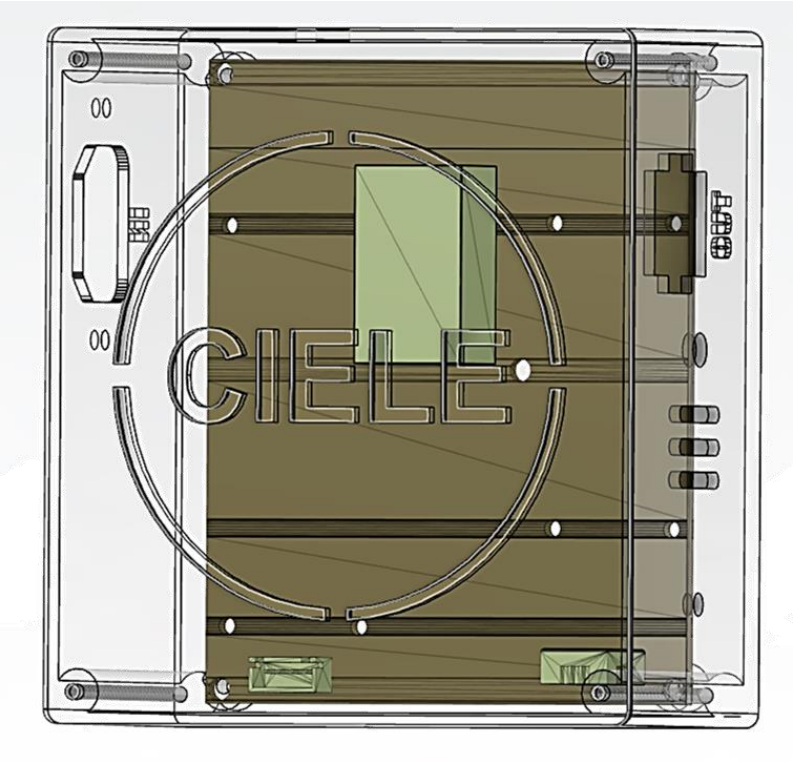
```
 93.
 94.   // Resetea las mediciones de energía del PZEM si se presiona un botón
 95.   if (digitalRead(pinResetEnergia) == HIGH) {
 96.     pzem.resetEnergy(); // Resetea la energía
 97.   }
 98.
 99.   // Envía los datos recogidos cada intervalo definido
100.   if (millisActual - millisPrevios >= intervalo) {
101.     String data = "V" + String(voltaje) + "C" + String(corriente, 3) + "P" +
String(potencia, 2) + "E" + String(energia, 3) + "F" + String(frecuencia) + "X" +
String(factorPotencia) + "M" + String(estadoMicro) + "Z";
102.
103.     Serial.println(data); // Muestra los datos en el Serial principal
104.     Serial2.println(data); // Envía los datos a través de Serial2
105.     millisPrevios = millisActual;
106.   }
107.
108.   // Enciende el LED si la energía ha aumentado desde la última medición
109.   if (energia > energiaAnterior) {
110.     digitalWrite(pinLed, HIGH); // Enciende el LED
111.     // No necesita guardar el tiempo en que se encendió el LED ya que se apaga
inmediatamente después de 50 ms
112.     delay(50); // Espera 50 milisegundos
113.     digitalWrite(pinLed, LOW); // Apaga el LED
114.   }
115.
116.   energiaAnterior = energia; // Actualiza la última medición de energía
117. }
```

ANEXO B.1

Manual de Usuario

MANUAL DE USUARIO

SISTEMA INTELIGENTE DE MEDICIÓN DE CONSUMO ELÉCTRICO RESIDENCIAL MEDIANTE HOME ASSISTANT.



ERICK CONDO NEVÁREZ

CIELE – UTN

2023 – 2024

TABLA DE CONTENIDO

INTRODUCCIÓN

Este texto ha sido elaborado con el objetivo de proporcionar una guía técnica del Sistema Inteligente de Medición de Consumo Eléctrico Residencial mediante Home Assistant, abarcando aspectos como la finalidad del sistema, sus funcionalidades, los componentes que lo integran y los procedimientos para su instalación.

Propósito del Sistema

El Sistema tiene como propósito la monitorización eléctrica de los dispositivos residenciales del hogar a través de Home Assistant, midiendo variables como el voltaje, corriente, frecuencia, factor de potencia, potencia activa, reactiva, aparente, y energía. Además, es posible controlar las cargas conectadas a los dispositivos que se están midiendo, y aprovecharse las funcionalidades del sistema operativo para disparar alarmas y evitar el estado prologando de consumo en standby.

Beneficios Clave

- Monitorización eléctrica constante: Mide variables eléctricas de dispositivos conectados para la evaluación de consumo desde la red.
- Control de las cargas: Permite conectar y desconectar dispositivos de la red a distancia.
- Programación de alarmas: Establece automatizaciones que accionen alarmas dependiendo del estado de los sensores.
- Evasión de consumo en standby: Desconecta de la red un dispositivo al mantenerse funcionando a una potencia muy baja.

Precauciones de Seguridad

Para utilizar los componentes del sistema de forma segura, hay que considerar los siguientes puntos:

- No mantener los nodos en exteriores y evitar excesiva humedad.
- En los nodos inalámbricos, utilizar fusibles de 15 A o menores, nunca mayores a dicho valor. En el nodo maestro, el valor recomendado de fusible es de 7A.
- En caso de manipular el interior de alguno de los nodos, hacerlo con el cable desconectado de la red, así esté el interruptor de alimentación en apagado.
- Asegurarse que la red eléctrica a la que se están conectando los dispositivos funciona a 120VAC 60 Hz monofásico.
- Poner los nodos en funcionamiento con la tapa colocada y atornillada.

CÓMO EMPEZAR

En esta sección, se explicará de forma breve y concisa el procedimiento a realizarse para poner en marcha el sistema en su totalidad.

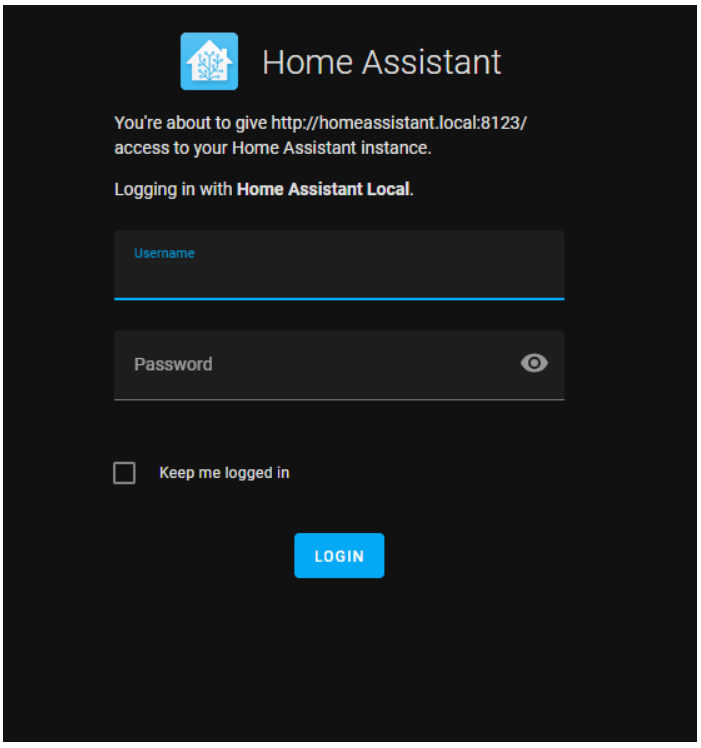
Nodo Maestro

1. Antes de conectar el cable de alimentación, es recomendable conectar el cable de red para que el sistema operativo de Home Assistant se inicialice al primer intento.

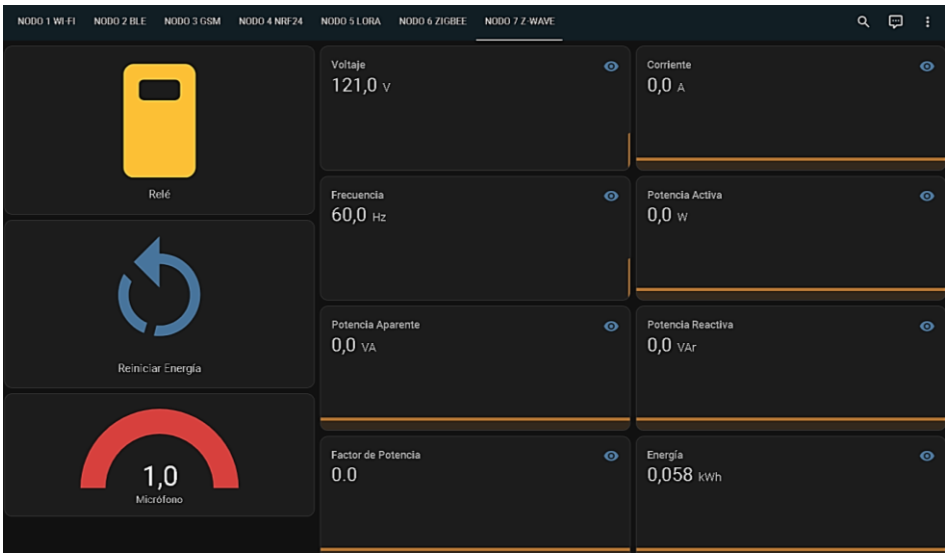
2. Verificar que la red eléctrica es de 125VAC 60 Hz monofásica y conectar el cable de alimentación al puerto C13, luego, encender el interruptor.

3. Esperar máximo 5 minutos, conectarse a la misma red desde otro dispositivo con acceso a un explorador web, a través de Wi-Fi o Ethernet. Luego, ingresar a http://homeassistant.local:8123/.

4. Ingresar el usuario y contraseña. El usuario predeterminado es el siguiente (usuario: interfaztablet ; contraseña: 123).



5. Se podrán observar las pestañas dedicadas al monitoreo y control de los 7 nodos, como también configuración más avanzada que es realizada a través de complementos de Home Assistant.



Nodos Independientes (cualquiera de los 7)

1. Colocar el nodo sobre una base plana, apoyándolo de una manera que no obstruya el acceso al puerto de entrada y/o salida AC.

2. Conectar el cable de alimentación a la entrada AC C13, es importante utilizar siempre un cable con pin de tierra y más cuando el dispositivo al que se conectará en la salida también lo tiene.

3. Encender el interruptor. Los leds del nodo se encenderán. Si observamos dos luces parpadeando de forma rápida dentro del nodo (correspondientes a la comunicación UART del PZEM) significa que el nodo entabló conexión con el gateway y los datos están ingresando a Home Assistant.

4. En Home Assistant, ingresar a la pestaña correspondiente del nodo que queremos conectar, verificar que las variables se estén actualizando constantemente y probar el accionamiento del relé.

5. Conectar una carga al nodo para observar cómo se actualizan los datos de forma rápida en Home Assistant.

FAQ – PREGUNTAS FRECUENTES

A continuación se responde a una serie de preguntas con el fin aclarar los inconvenientes que se podrían presentar en la utilización del Sistema.

1. **¿Cómo ingreso a Home Assistant si la dirección por defecto no sirve?**

Si http://homeassistant.local:8123/ no sirve, podemos probar entrando con una dirección IP distinta.

- Es posible conseguir la IP accediendo a la interfaz web de tu router ingresando su dirección IP en un navegador web. La dirección IP del router suele ser algo como 192.168.1.1 o 192.168.0.1, pero puede variar. Después, busca una sección llamada "Dispositivos Conectados", "DHCP Clients", "LAN Clients", o algo similar.
- Busca en la lista un dispositivo cuyo nombre sea reconocible como Home Assistant o algo similar, dependiendo de cómo esté configurado tu dispositivo Home Assistant (por ejemplo, podría aparecer como "hassio", "homeassistant", o el nombre del dispositivo o sistema operativo en el que se ejecuta).
- Si se tiene acceso a la interfaz de Home Assistant, es posible revisar la IP Local en la que el servidor se está alojando desde http://homeassistant.local:8123/config/network.
- Si se tiene acceso a la Raspberry Pi, podemos conectar un monitor al puerto HDMI del dispositivo. Ahí podremos observar el Terminal Linux del sistema operativo, donde también se muestra la IP desde donde está funcionando.

2. **¿Cómo sé que existe comunicación exitosa entre un nodo independiente y el nodo maestro?**

Existen varias formas de corroborar la conexión exitosa, las cuales se enlistan a continuación.

- Si hay conexión, los datos eléctricos se actualizarán constantemente, por lo que las mediciones se imprimirán con un valor numérico. Caso contrario, se mostrará el texto “Unavailable”.
- Si hay conexión, al presionar el botón del relé desde Home Assistant, se escuchará el clic del relé en el nodo.

3. ¿Qué debo verificar si no hay conexión?

Hay varias cosas que debemos verificar si no hay conexión, algunas se las realiza desde Home Assistant y otras desde el nodo. Revisar los siguientes puntos para solucionar esta problemática.

En Home Assistant:

- Verificar en el archivo configuration.yaml si es que el serial_port correspondiente al nodo que no se conecta, es el mismo que se asignó en http://homeassistant.local:8123/config/hardware. Si ese es el caso, modificar esa linea y la correspondiente a los switches.

- Quitar el Gateway USB de la tecnología de comunicación en la que tenemos problemas y conectarlo directamente en un PC. Utiliza algún software para leer el serial con el nodo encendido y verificar si ahí funciona. Si es que sí, el problema no es del nodo.

- Desconectar el cable USB y volverlo a conectar en la misma posición. Reiniciar la Raspberry Pi podría ser necesario.

En el Nodo:

- Verificar que el firmware del ESP32 es el correcto para el nodo que se está utilizando. Si no es el caso, volver a subir el código con Arduino IDE.

- Revisar el cableado en la PCB y si los módulos se encuentran bien posicionados en los puertos correspondientes.

4. ¿Por qué no se enciende un Nodo?

El que un Nodo no se encienda a pesar de haber red eléctrica tiene pocos motivos probables, los cuales son enlistados a continuación:

- Cable de alimentación defectuoso: Un cable puede estar roto por dentro y no ser visible a simple vista.

- Fusible por cambiar: Verificar si hay continuidad entre los dos extremos del fusible. Si no hay, cambiar por un fusible recomendado (7A para el Nodo Maestro y 15A para los Nodos Independientes)

- Componente Electrónico Dañado: Verificar el cableado eléctrico como las rutas de las en la PCB.

5. **En Home Assistant las variables de corriente, potencia, factor de potencia y energía se mantienen en 0 a pesar de agregar carga, ¿Por qué?**

Esto puede ocurrir por dos motivos concretos, especificado a continuación.

- En el caso haberse verificado que existe linea viva en la salida, este problema puede deberse a que el transformador de corriente del PZEM-004T V3 se desconectó de las borneras del módulo. Desconecta todo y vuelve a conectar los dos cables.

- El dispositivo que conectamos puede ser realmente un circuito abierto. Esto puede ocurrir con un dispositivo que, directamente, está dañado, o con uno que auto conmuta su alimentación para controlar su temperatura (ejemplo, una plancha o un tostador).

6. **¿Cómo sé que el relé del Nodo está funcionando?**

Cuando activamos o desactivamos el relé desde Home Assistant, escucharemos un clic en el nodo, y podremos alimentar un dispositivo en la salida, de lo contrario, puede deberse a los siguientes puntos:

- Cable flojo: Un mal contacto de los cables de control con las borneras del módulo relé puede ocasionar que el relé no obtenga suficiente corriente para conmutarse, se soluciona acomodando el cable y el tornillo de la bornera.

- Relé defectuoso: Si se verificó lo anterior y sigue sin funcionar, es probable que el relé ya no sirva. Quita el módulo de la PCB y pruébalo

externamente, si tampoco conmuta, es necesario adquirir otro. Si lo hace, vuelve a colocarlo en el PCB cambiando los cables de control.

7. ¿Cómo sé que el PZEM-004T V3 del Nodo está funcionando?

Cuando hay una conexión correcta del PZEM-004T V3 sucederá lo siguiente:

- Un led rojo que se encuentra debajo del capacitor grande cuadrado y amarillo se iluminará. Eso significa que si hay alimentación AC.

- Dos leds rojos que se encuentran cercanos a los pines de comunicación, uno al lado del otro, parpadearán muy rápido alternándose entre sí. Eso significa que hay comunicación serial exitosa y que el ESP32 está leyendo los datos de las mediciones.

Es necesario tomar en cuenta que el módulo necesita tanto de alimentación AC como DC para funcionar.

8. ¿Cómo añado otro Nodo Wi-Fi?

Para añadir otro nodo Wi-Fi, solo basta con subir el mismo código utilizado en el Nodo 1. Debemos procurar cambiar los nombres de los topics que estamos publicando para no confundirnos con los del otro nodo.

En Home Assistant, tendríamos que agregar los sensores plantilla en base a los temas MQTT que creamos, y configurar las otras opciones básicas.

8. ¿Cómo añado otro Nodo BLE?

Añadir otro Nodo BLE funcionaría también con el mismo código, simplemente hay que cambiar los datos enviados en el String. Es necesario agregar delimitadores que no concuerden con los utilizados por los otros nodos (Por ejemplo, en el Nodo 1 se utiliza V125.0C para filtrar el 125.0 como la medición de voltaje). Para el envío desde HA a el Nodo, procura que los datos que espera recibir el nuevo Nodo no coincidan con los que espera el Nodo anterior.

En el Gateway, debemos adaptar el código para que también procese estos datos y los imprima de la forma necesaria para que Home Assistant los

pueda leer. Después, conectamos el USB a la Raspberry, y en HA agregamos más sensores de plantilla en base a los delimitadores que estamos recibiendo del Nodo.

9. ¿Cómo añado otro Nodo GSM?

Este caso es igual a los anteriores, inclusive, se simplifica un poco el proceso. El número móvil al que se enviará seguirá siendo el mismo, solo es necesario el cambio de los datos que se envían y los que espera recibir para el disparo de distintas funciones. También sería necesario agregar los sensores plantilla en base a este nuevo Nodo.

10. ¿Cómo añado otro Nodo nRF24?

El código se mantiene y se realiza el mismo proceso necesario explicado en los Nodos anteriores.

11. ¿Cómo añado otro Nodo LoRa?

El código se mantiene y se realiza el mismo proceso necesario explicado en los Nodos anteriores.

12. ¿Cómo añado otro Nodo Zigbee?

Antes, será necesario configurar el Xbee S2C en el software XCTU. Los parámetros a agregar serían los mismos utilizados en el Nodo 6. Es importante que el dispositivo sea establecido como router. Luego podremos realizar la misma configuración ya explicada en los otros nodos.

13. ¿Cómo añado otro Nodo Z-Wave?

Debido a que en Z-Uno 2 la configuración de Z-Wave se encuentra bastante simplificada, es muy sencillo agregar un Nodo más. En el ESP32 configuramos las variables que queremos enviar por serial, en el Z-Uno 2 las filtramos y al encenderse y emparejar nuestro nuevo Z-Uno 2, se enviarían ya procesadas al controlador Z-Wave conectado a Home Assistant.

14. ¿Cómo accedo a Home Assistant fuera de la red lan?

Esto puede lograrse mediante la configuración de acceso remoto seguro, como Nabu Casa (de pago), o configurando una VPN como ZeroTier (gratuita).

15. ¿Cómo mantengo actualizado Home Assistant?

En la pestaña de Ajustes (http://homeassistant.local:8123/config/dashboard) nos aparecerá en la parte superior tanto las actualizaciones del sistema operativo, como las de los complementos.

16. ¿Cómo puedo crear Automatizaciones y escenas de Home Assitant?

Puedes crear automatizaciones y escenas dentro de la interfaz de usuario de Home Assistant. Ve a Configuración > Automatizaciones (o Escenas) y utiliza el editor visual para definir las condiciones y acciones. También puedes escribir tus automatizaciones manualmente en el archivo automations.yaml para una mayor personalización.

17. ¿Cómo puedo restaurar una copia de seguridad de Home Assistant?

Para restaurar una copia de seguridad, ve a Configuración > Sistema > Copias de seguridad en la interfaz de usuario de Home Assistant. Aquí puedes gestionar y restaurar copias de seguridad. Si no puedes acceder a la interfaz de usuario, puedes restaurar manualmente los archivos de la copia de seguridad en la carpeta de configuración de Home Assistant.

Puedes descargar la última copia de seguridad del Sistema Inteligente de Medición de Consumo Eléctrico Residencial en este enlace: MARZO_3.tar

MANTENIMIENTO

Revisaremos el mantenimiento preventivo, correctivo y predictivos que se le pueden hacer a los distintos elementos del Sistema.

Mantenimiento Preventivo

Para prevenir daños futuros, es recomendable hacer revisiones constantes del siguiente tipo:

Limpieza General:

Usar aire comprimido para remover el polvo de las cajas y de los componentes internos. Para áreas sensibles, utilizar un cepillo antiestático. La limpieza previene problemas de sobrecalentamiento y fallos en los componentes.

Revisión de Ventiladores:

En el caso del Nodo Maestro que tiene ventiladores, verificar que funcionen correctamente y que no estén obstruidos por polvo o suciedad. Reemplaza los ventiladores que presenten ruidos extraños o no giren adecuadamente.

Inspección Visual:

Buscar signos de desgaste o daño en los componentes, como condensadores hinchados en las fuentes de alimentación, quemaduras o corrosión en los PCBs y cables.

Verificación de Conexiones:

Asegúrate de que todas las conexiones, incluidos los cables USB del Nodo Maestro y otros conectores, estén firmes y no presenten signos de daño o de malos contactos.

Mantenimiento Correctivo

Cuando se identifiquen problemas durante el mantenimiento preventivo o en el uso diario, el mantenimiento correctivo será necesario.

Reemplazo de Componentes Dañados:

Si detectas componentes dañados o que funcionan mal, reemplázalos inmediatamente para evitar daños adicionales.

Resubida del Firmware:

Mantén los microcontroladores y dispositivos con el último firmware para asegurar el funcionamiento esperado. Verifica este manual para recordar en qué versiones de software fueron escritos los códigos.

Reparación de Circuitos:

Si hay problemas en los PCBs, como pistas rotas o soldaduras frías, es necesario realizar reparaciones con un soldador y material de soldadura adecuados.

Mantenimiento Predictivo

Con base en el funcionamiento de los equipos, intenta predecir cuándo podrían fallar los componentes y reemplazarlos o repararlos antes de que ocurra la falla.

Monitoreo de Temperatura:

Usa sensores de temperatura para monitorear el calor generado por los componentes. Altas temperaturas constantes pueden indicar problemas futuros y cortos.

Análisis de Rendimiento:

Evalúa el rendimiento de los sistemas regularmente para detectar posibles problemas antes de que se conviertan en fallas.

Registro de Mantenimiento:

Lleva un registro detallado de todas las acciones de mantenimiento, incluyendo cambios de componentes, actualizaciones de software y reparaciones. Esto te ayudará a identificar patrones y predecir futuras necesidades de mantenimiento.

CONTACTO

Para dudas y sugerencias, contáctate con los siguientes medios.


0994796445


erick-full@hotmail.com

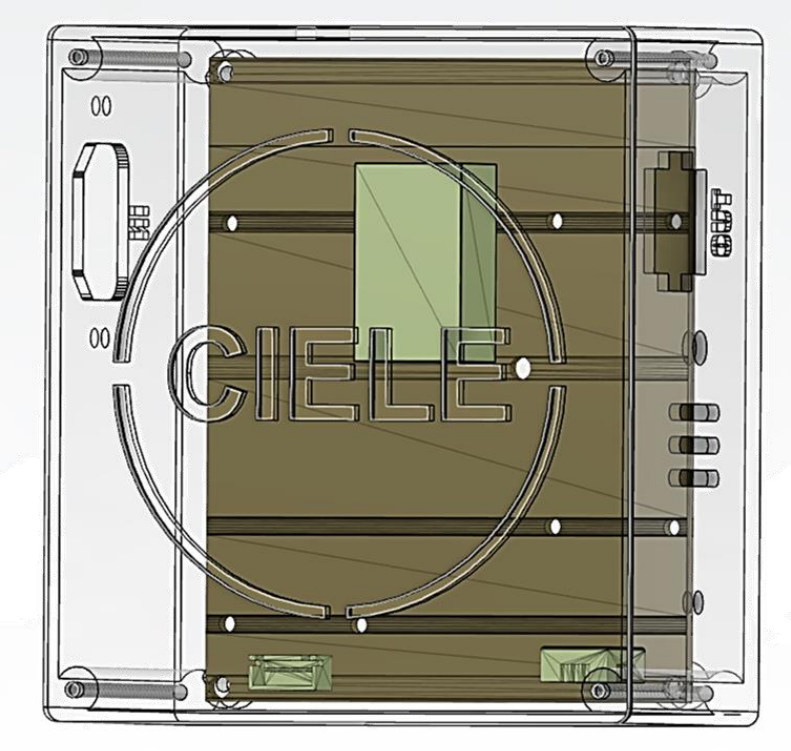

Nuevo Santo Domingo Sector 1, Juan Pablo Neruda y Joaquín Balaguer, Santo Domingo de los Colorados, Santo Domingo de los Tsáchilas

ANEXO B.2

Manual Técnico

MANUAL TÉCNICO

SISTEMA INTELIGENTE DE MEDICIÓN DE CONSUMO ELÉCTRICO RESIDENCIAL MEDIANTE HOME ASSISTANT.



ERICK CONDO NEVÁREZ

CIELE – UTN

2023 – 2024

TABLA DE CONTENIDO

INTRODUCCIÓN

Este texto ha sido elaborado con el objetivo de proporcionar una guía técnica del Sistema Inteligente de Medición de Consumo Eléctrico Residencial mediante Home Assistant, abarcando aspectos como la finalidad del sistema, sus funcionalidades, los componentes que lo integran y los procedimientos para su instalación.

Propósito del Sistema

El Sistema tiene como propósito la monitorización eléctrica de los dispositivos residenciales del hogar a través de Home Assistant, midiendo variables como el voltaje, corriente, frecuencia, factor de potencia, potencia activa, reactiva, aparente, y energía. Además, es posible controlar las cargas conectadas a los dispositivos que se están midiendo, y aprovecharse las funcionalidades del sistema operativo para disparar alarmas y evitar el estado prologando de consumo en standby.

Beneficios Clave

- Monitorización eléctrica constante: Mide variables eléctricas de dispositivos conectados para la evaluación de consumo desde la red.
- Control de las cargas: Permite conectar y desconectar dispositivos de la red a distancia.
- Programación de alarmas: Establece automatizaciones que accionen alarmas dependiendo del estado de los sensores.
- Evasión de consumo en standby: Desconecta de la red un dispositivo al mantenerse funcionando a una potencia muy baja.

Precauciones de Seguridad

Para utilizar los componentes del sistema de forma segura, hay que considerar los siguientes puntos:

- No mantener los nodos en exteriores y evitar excesiva humedad.
- En los nodos inalámbricos, utilizar fusibles de 15 A o menores, nunca mayores a dicho valor. En el nodo maestro, el valor recomendado de fusible es de 7A.
- En caso de manipular el interior de alguno de los nodos, hacerlo con el cable desconectado de la red, así esté el interruptor de alimentación en apagado.
- Asegurarse que la red eléctrica a la que se están conectando los dispositivos funciona a 120VAC 60 Hz monofásico.
- Poner los nodos en funcionamiento con la tapa colocada y atornillada.

ESPECIFICACIONES TÉCNICAS

Cada componente del Sistema cuenta con sus especificaciones técnicas y funcionalidades, las cuales son enlistadas a continuación.

Nodo Maestro

Características Físicas:

- Dimensiones:
  202.61x160x126 mm
- Grosor de las paredes:
  3 mm
- Material:
  PETG negro y transparente

Características Eléctricas:

- Entrada:
  120 VAC 60 Hz 6 A monofásico
- Puertos:
  C13 para alimentación
  RJ45 para conectividad
- Ventilación:
  2 ventiladores de 40 mm 5VDC
- Indicadores
  Led rojo para encendido (integrado en el interruptor)

Características inalámbricas:

- Comunicación: Wi-Fi, BLE, GSM, nRF24, LoRa, Zigbee, Z-Wave.

Nodo 1 Wi-Fi

Características Físicas:

- Dimensiones:
  120x110x54.5 mm
- Grosor de las paredes:
  2 mm
- Material:
  PETG transparente

Características Eléctricas:

- Entrada:
  120 VAC 60 Hz 15 A monofásico
- Puertos:
  C13 para entrada
  NEMA 5-15p para salida
- Ventilación:
  Por aire
- Indicadores:
  Led rojo para encendido
  Led azul para parpadeo de constante de medición (Imp/kwh)

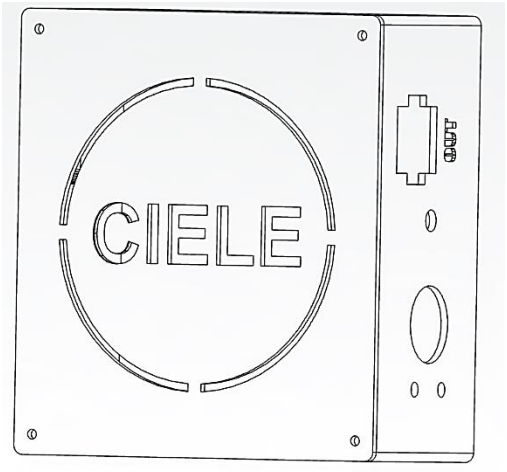
Características inalámbricas:

- Comunicación: Wi-Fi (MQTT).

Características de medición:

- Voltaje, Corriente, Potencia, Factor de Potencia, Potencia Activa, Energía (PZEM-004T V3)
- Temperatura y Humedad (DHT11)

Nodo 2 BLE



Características Físicas:

- Dimensiones:
  120x110x54.5 mm
- Grosor de las paredes:
  2 mm
- Material:
  PETG transparente

Características Eléctricas:

- Entrada:
  120 VAC 60 Hz 15 A monofásico
- Puertos:
  C13 para entrada
  NEMA 5-15p para salida
- Ventilación:
  Por aire
- Indicadores:
  Led rojo para encendido
  Led azul para parpadeo de constante de medición (Imp/kwh)

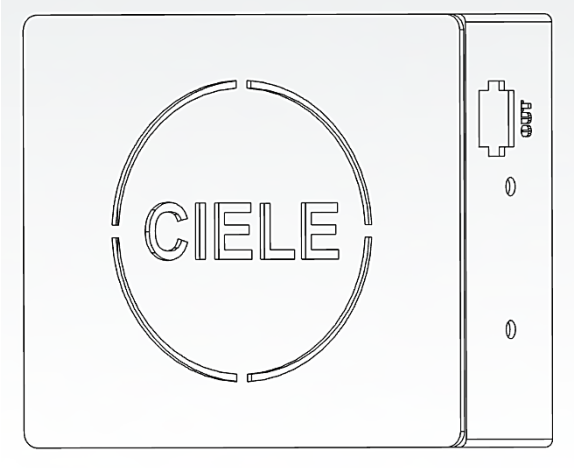
Características inalámbricas:

- Comunicación: BLE.

Características de medición:

- Voltaje, Corriente, Potencia, Factor de Potencia, Potencia Activa, Energía (PZEM-004T V3)
- Gas licuado, Metano, Alcohol, Humo y Propano (MQ-2).

Nodo 3 GSM



Características Físicas:

- Dimensiones:
  120x110x54.5 mm
- Grosor de las paredes:
  2 mm
- Material:
  PETG transparente

Características Eléctricas:

- Entrada:
  120 VAC 60 Hz 15 A monofásico
- Puertos:
  C13 para entrada
  NEMA 5-15p para salida
- Ventilación:
  Por aire
- Indicadores:
  Led rojo para encendido
  Led azul para parpadeo de constante de medición (Imp/kwh)

Características inalámbricas:

- Comunicación: GSM.

Características de medición:

- Voltaje, Corriente, Potencia, Factor de Potencia, Potencia Activa, Energía (PZEM-004T V3)
- Luz Ambiental (Fotodiodo).

Nodo 4 nRF24

Características Físicas:

- Dimensiones:
  120x110x54.5 mm
- Grosor de las paredes:
  2 mm
- Material:
  PETG transparente

Características Eléctricas:

- Entrada:
  120 VAC 60 Hz 15 A monofásico
- Puertos:
  C13 para entrada
  NEMA 5-15p para salida
- Ventilación:
  Por aire
- Indicadores:
  Led rojo para encendido
  Led azul para parpadeo de constante de medición (Imp/kwh)

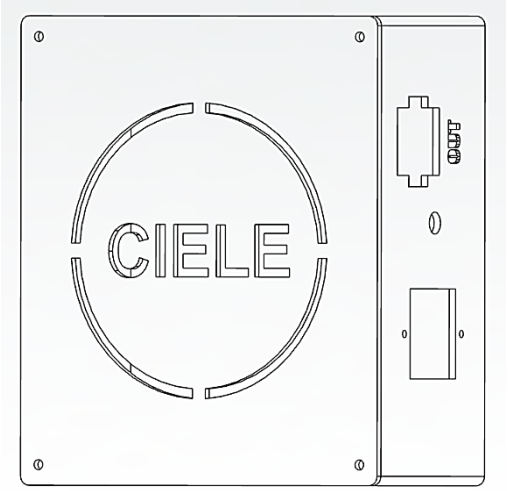
Características inalámbricas:

- Comunicación: nRF24.

Características de medición:

- Voltaje, Corriente, Potencia, Factor de Potencia, Potencia Activa, Energía (PZEM-004T V3)

Nodo 5 LoRa



Características Físicas:

- Dimensiones:
  120x110x54.5 mm
- Grosor de las paredes:
  2 mm
- Material:
  PETG transparente

Características Eléctricas:

- Entrada:
  120 VAC 60 Hz 15 A monofásico
- Puertos:
  C13 para entrada
  NEMA 5-15p para salida
- Ventilación:
  Por aire
- Indicadores:
  Led rojo para encendido
  Led azul para parpadeo de constante de medición (Imp/kwh)

Características inalámbricas:

- Comunicación: LoRa.

Características de medición:

- Voltaje, Corriente, Potencia, Factor de Potencia, Potencia Activa, Energía (PZEM-004T V3)
- Gas natural, Gas licuado, Coal gas (MQ-5)

Nodo 6 Zigbee

Características Físicas:

- Dimensiones:
  120x110x54.5 mm
- Grosor de las paredes:
  2 mm
- Material:
  PETG transparente

Características Eléctricas:

- Entrada:
  120 VAC 60 Hz 15 A monofásico
- Puertos:
  C13 para entrada
  NEMA 5-15p para salida
- Ventilación:
  Por aire
- Indicadores:
  Led rojo para encendido
  Led azul para parpadeo de constante de medición (Imp/kwh)

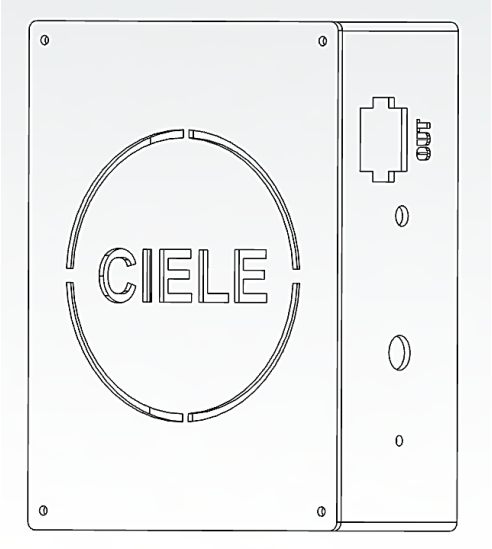
Características inalámbricas:

- Comunicación: Zigbee.

Características de medición:

- Voltaje, Corriente, Potencia, Factor de Potencia, Potencia Activa, Energía (PZEM-004T V3)
- Presencia (HC-SR501)

Nodo 7 Z-Wave



Características Físicas:

- Dimensiones:
  120x110x54.5 mm
- Grosor de las paredes:
  2 mm
- Material:
  PETG transparente

Características Eléctricas:

- Entrada:
  120 VAC 60 Hz 15 A monofásico
- Puertos:
  C13 para entrada
  NEMA 5-15p para salida
- Ventilación:
  Por aire
- Indicadores:
  Led rojo para encendido
  Led azul para parpadeo de constante de medición (Imp/kwh)

Características inalámbricas:

- Comunicación: Z-Wave.

Características de medición:

- Voltaje, Corriente, Potencia, Factor de Potencia, Potencia Activa, Energía (PZEM-004T V3)
- Sonido (Micrófono)

COMPONENTES

A continuación se exponen los componentes de cada nodo junto a una breve descripción de su función:

Nodo Maestro

Componentes:

- Fuente de Alimentación 3A y 3.5 5VDC:
  Alimenta la Raspberry y el USB Hub, respectivamente.

- Raspberry Pi 4B 4 GB:
  Funciona como cerebro central y almacena la información

  Atolla USB 3.0 HUB: Aumenta los puertos de la Raspberry y alimenta los dispositivos USB con una fuente externa.

- ESP32:
  Funciona como Gateway BLE a través de UART

- SIM800C USB Stick:
  Funciona como Gateway GSM a través de UART

- nRF24 + Arduino Nano:
  Funciona como Gateway nRF24, el Nano adapta los datos recibidos y los envía por serial.

- LoRa + ESP32 C6:
  Funciona como Gateway LoRa a través de UART, el ESP32 adapta los datos recibidos y los envía por serial.

- Xbee S2C + Xbee Explorer USB:
  Funciona como Gateway Zigbee, el Explorer adapta el módulo para usarse a través de USB.

- ZOOZ S2 Stick 700:
  Funciona como Gateway Z-Wave.

- Fusible 7A + Portafusibles:
  Protege el nodo a sobrecarga y cortocircuitos.

- Conector C13:
  Entrada a la alimentación AC.

- Conector hembra RJ45:
  Entrada para la conexión a la red LAN.

- Interruptor con iluminación:
  Conmuta la alimentación al dispositivo.

- 2 ventiladores 40mm 5VDC:
  Refrigera los componentes ingresando aire frio y expulsando el aire caliente.

Nodo 1 Wi-Fi

Componentes:

- Fuente de Alimentación 0.5 5VDC:
  Alimenta todos los componentes del nodo.

- Interruptor:
  Conmuta la alimentación AC del nodo.

- Conector C13:
  Entrada AC al nodo.

- Conector NEMA 5-15p:
  Salida AC del nodo.

- ESP32:
  Procesa los datos de mediciones del PZEM, controla el relé y el led azul de constante del medidor.

- PZEM-004T V3: Mide las variables eléctricas de la red y envía los datos a través de comunicación UART.

- DHT-11:

  Mide temperatura y humedad.

- Conversor de nivel 5V a 3.3V:

  Adapta el nivel lógico del PZEM (5V) al del ESP32 (3.3V)

- Módulo Relé con conversor de nivel 5V a 3.3V:
  Conmuta la fase de red eléctrica para el control de las cargas.

- Fusible 15A con portafusibles:
  Protege el nodo de sobrecargas y cortocircuitos.

- Led rojo:
  Indicador de encendido del nodo.

- Led azul:
  Indicador de la constante del medidor.

Nodo 2 BLE

Componentes:

- Fuente de Alimentación 0.5 5VDC:
  Alimenta todos los componentes del nodo.

- Interruptor:
  Conmuta la alimentación AC del nodo.

- Conector C13:
  Entrada AC al nodo.

- Conector NEMA 5-15p:
  Salida AC del nodo.

- ESP32:
  Procesa los datos de mediciones del PZEM, controla el relé y el led azul de constante del medidor.

- PZEM-004T V3: Mide las variables eléctricas de la red y envía los datos a través de comunicación UART.

- MQ-2:

  Mide LPG, Butano, Metano, Humo, Propano.

- Conversor de nivel 5V a 3.3V:

  Adapta el nivel lógico del PZEM (5V) al del ESP32 (3.3V)

- Módulo Relé con conversor de nivel 5V a 3.3V:
  Conmuta la fase de red eléctrica para el control de las cargas.

- Fusible 15A con portafusibles:
  Protege el nodo de sobrecargas y cortocircuitos.

- Led rojo:
  Indicador de encendido del nodo.

- Led azul:
  Indicador de la constante del medidor.

Nodo 3 GSM

Componentes:

- Fuente de Alimentación 0.5 5VDC:
  Alimenta todos los componentes del nodo.

- Módulo Reductor de Voltaje DC Buck:
  Reduce el voltaje de 5VDC a 4.2VDC para el SIM800L

- Capacitor 100 uF:
  Reduce el ruido en la alimentación al SIM800L

- Interruptor:
  Conmuta la alimentación AC del nodo.

- Conector C13:
  Entrada AC al nodo.

- Conector NEMA 5-15p:
  Salida AC del nodo.

- ESP32:
  Procesa los datos de mediciones del PZEM, controla el SIM800L, el relé y el led azul de constante del medidor.

- PZEM-004T V3: Mide las variables eléctricas de la red y envía los datos a través de comunicación UART.

- Fotodiodo:

  Capta la luz ambiental.

- SIM800L:
  Transceptor encargado de la comunicación GSM.

- Conversor de nivel 5V a 3.3V:

  Adapta el nivel lógico del PZEM (5V) al del ESP32 (3.3V)

- Módulo Relé con conversor de nivel 5V a 3.3V:
  Conmuta la fase de red eléctrica para el control de las cargas.

- Fusible 15A con portafusibles:
  Protege el nodo de sobrecargas y cortocircuitos.

- Led rojo:
  Indicador de encendido del nodo.

- Led azul:
  Indicador de la constante del medidor.

Nodo 4 nRF24

Componentes:

- Fuente de Alimentación 0.5 5VDC:
  Alimenta todos los componentes del nodo.

- Interruptor:
  Conmuta la alimentación AC del nodo.

- Conector C13:
  Entrada AC al nodo.

- Conector NEMA 5-15p:
  Salida AC del nodo.

- ESP32:
  Procesa los datos de mediciones del PZEM, controla el SIM800L, el relé y el led azul de constante del medidor.

- nRF24L01 PA+LNA:
  Transceptor encargado de la comunicación nRF24.

- nRF24 Breakout Board:
  Adapta 5VDC a 3.6VDC, voltaje óptimo para nRF24.

- PZEM-004T V3:
  Mide las variables eléctricas de la red y envía los datos a través de comunicación UART.

- Buzzer
  Emite pitidos como alarmas.

- Conversor de nivel 5V a 3.3V:
  Adapta el nivel lógico del PZEM (5V) al del ESP32 (3.3V)

- Módulo Relé con conversor de nivel 5V a 3.3V:
  Conmuta la fase de red eléctrica para el control de las cargas.

- Fusible 15A con portafusibles:
  Protege el nodo de sobrecargas y cortocircuitos.

- Led rojo:
  Indicador de encendido del nodo.

- Led azul:
  Indicador de la constante del medidor.

Nodo 5 LoRa

Componentes:

- Fuente de Alimentación 0.5 5VDC:
  Alimenta todos los componentes del nodo.

- Interruptor:
  Conmuta la alimentación AC del nodo.

- Conector C13:
  Entrada AC al nodo.

- Conector NEMA 5-15p:
  Salida AC del nodo.

- ESP32:
  Procesa los datos de mediciones del PZEM, controla el SIM800L, el relé y el led azul de constante del medidor.

- RYLR998:
  Transceptor encargado de la comunicación LoRa.

- PZEM-004T V3:
  Mide las variables eléctricas de la red y envía los datos a través de comunicación UART.

- MQ-5:
  Mide presencia de gas natural, LPG, Coal gas.

- Conversor de nivel 5V a 3.3V:

Adapta el nivel lógico del PZEM (5V) al del ESP32 (3.3V)

- Módulo Relé con conversor de nivel 5V a 3.3V:
  Conmuta la fase de red eléctrica para el control de las cargas.

- Fusible 15A con portafusibles:
  Protege el nodo de sobrecargas y cortocircuitos.

- Led rojo:
  Indicador de encendido del nodo.

- Led azul:
  Indicador de la constante del medidor.

Nodo 6 Zigbee

Componentes:

- Fuente de Alimentación 0.5 5VDC:
  Alimenta todos los componentes del nodo.

- Interruptor:
  Conmuta la alimentación AC del nodo.

- Conector C13:
  Entrada AC al nodo.

- Conector NEMA 5-15p:
  Salida AC del nodo.

- ESP32:
  Procesa los datos de mediciones del PZEM, controla el SIM800L, el relé y el led azul de constante del medidor.

- Xbee S2C: Transceptor encargado de la comunicación Zigbee.

- PZEM-004T V3:
  Mide las variables eléctricas de la red y envía los datos a través de comunicación UART.

- HC-SR501:
  Funciona como sensor de presencia.

- Conversor de nivel 5V a 3.3V:
  Adapta el nivel lógico del PZEM (5V) al del ESP32 (3.3V)

- Módulo Relé con conversor de nivel 5V a 3.3V:
  Conmuta la fase de red eléctrica para el control de las cargas.

- Fusible 15A con portafusibles:
  Protege el nodo de sobrecargas y cortocircuitos.

- Led rojo:
  Indicador de encendido del nodo.

- Led azul:
  Indicador de la constante del medidor.

Nodo 7 Z-Wave

Componentes:

- Fuente de Alimentación 0.5A 5VDC:
  Alimenta todos los componentes del nodo.

- Interruptor:
  Conmuta la alimentación AC del nodo.

- Conector C13:
  Entrada AC al nodo.

- Conector NEMA 5-15p:
  Salida AC del nodo.

- ESP32:
  Procesa los datos de mediciones del PZEM, controla el SIM800L, el relé y el led azul de constante del medidor.

- Z-UNO 2
  Transceptor encargado de comunicación Z-Wave.

- PZEM-004T V3:
  Mide las variables eléctricas de la red y envía los datos a través de comunicación UART.

- Micrófono
  Capta el sonido y ruido ambiente.

- Conversor de nivel 5V a 3.3V:
  Adapta el nivel lógico del PZEM (5V) al del ESP32 (3.3V)

- Módulo Relé con conversor de nivel 5V a 3.3V:
  Conmuta la fase de red eléctrica para el control de las cargas.

- Fusible 15A con portafusibles:
  Protege el nodo de sobrecargas y cortocircuitos.

- Led rojo:
  Indicador de encendido del nodo.

- Led azul:
  Indicador de la constante del medidor.

CÓMO EMPEZAR

En esta sección, se explicará de forma breve y concisa el procedimiento a realizarse para poner en marcha el sistema en su totalidad.

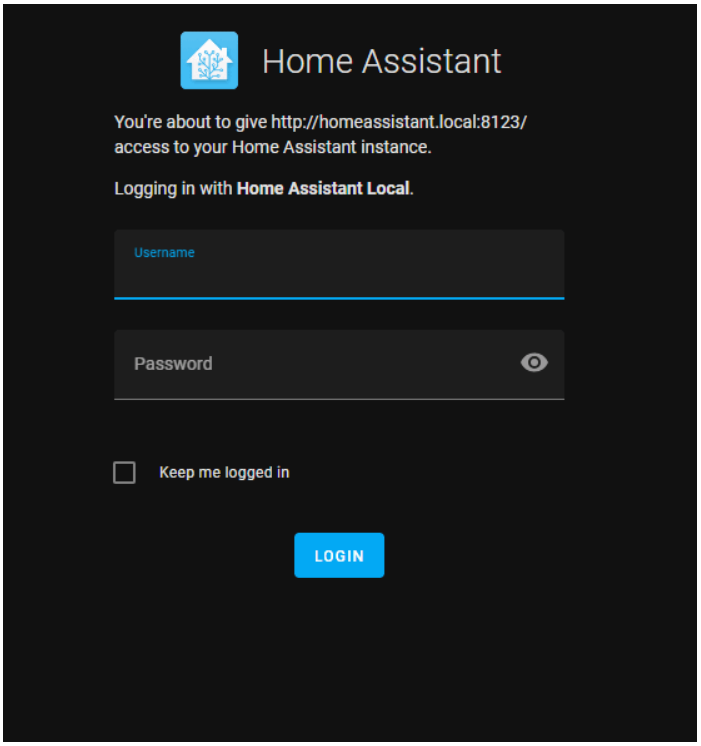
Nodo Maestro

1. Antes de conectar el cable de alimentación, es recomendable conectar el cable de red para que el sistema operativo de Home Assistant se inicialice al primer intento.

2. Verificar que la red eléctrica es de 125VAC 60 Hz monofásica y conectar el cable de alimentación al puerto C13, luego, encender el interruptor.

3. Esperar máximo 5 minutos, conectarse a la misma red desde otro dispositivo con acceso a un explorador web, a través de Wi-Fi o Ethernet. Ingresar a http://homeassistant.local:8123/.

4. Ingresar el usuario y contraseña. El usuario predeterminado es el siguiente (usuario: interfaztablet ; contraseña: 123123123).



5. Se podrán observar las pestañas dedicadas al monitoreo y control de los 7 nodos, como también configuración más avanzada que es realizada a través de complementos de Home Assistant.

Nodos Independientes (cualquiera de los 7)

1. Colocar el nodo sobre una base plana, apoyándolo de una manera que no obstruya el acceso al puerto de entrada y/o salida AC.

2. Conectar el cable de alimentación a la entrada AC C13, es importante utilizar siempre un cable con pin de tierra y más cuando el dispositivo al que se conectará en la salida también lo tiene.

3. Encender el interruptor. Los leds del nodo se encenderán. Si observamos dos luces parpadeando de forma rápida dentro del nodo (correspondientes a la comunicación UART del PZEM) significa que el nodo entabló conexión con el Gateway y los datos están ingresando a Home Assistant.

4. En Home Assistant, ingresar a la pestaña correspondiente al nodo que queremos conectar, verificar la constante actualización de las variables eléctricas y probar el accionamiento del relé.

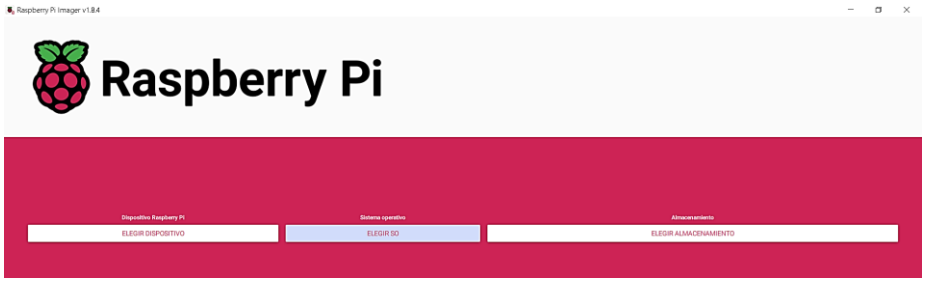
5. Conectar un dispositivo al nodo para observar cómo se actualizan los datos de forma rápida en Home Assistant.

INSTALACIÓN DEL SISTEMA

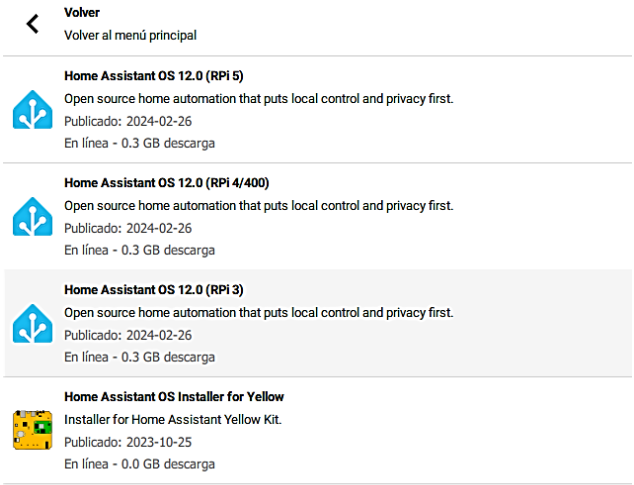
A continuación, se explicará detalladamente y paso por paso el procedimiento para la elaboración del Sistema Inteligente de Medición de Consumo Eléctrico Residencial mediante Home Assistant.

ETAPA 1: Home Assistant

Primero, insertamos la tarjeta microSD en el lector de tarjetas de nuestra computadora o usando un adaptador USB. Después, descargamos Raspberry Pi Imager desde este link directo.

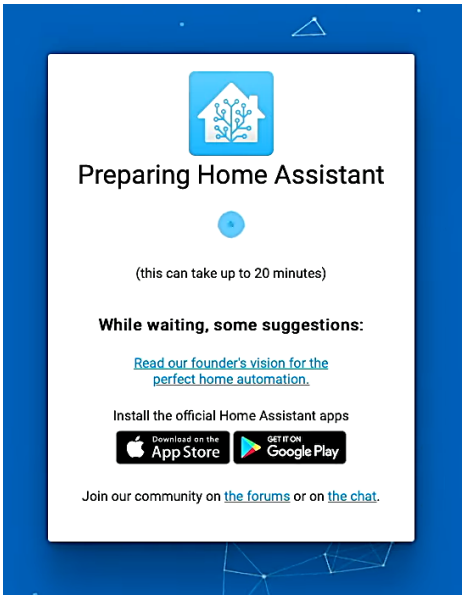


Seleccionamos el dispositivo Raspberry Pi y el Almacenamiento, el cual vendría a ser nuestra SD. En Sistema Operativo, hacemos clic en "ELEGIR SO". Nos desplazamos hacia abajo y seleccionamos "Other specific-purpose OS". Luego, elegimos "Home Assistant" y seleccionamos la versión de Home Assistant OS para Raspberry Pi 4. Nos aseguramos de elegir la opción que corresponde a la versión de nuestra Raspberry Pi (en este caso, Raspberry Pi 4).

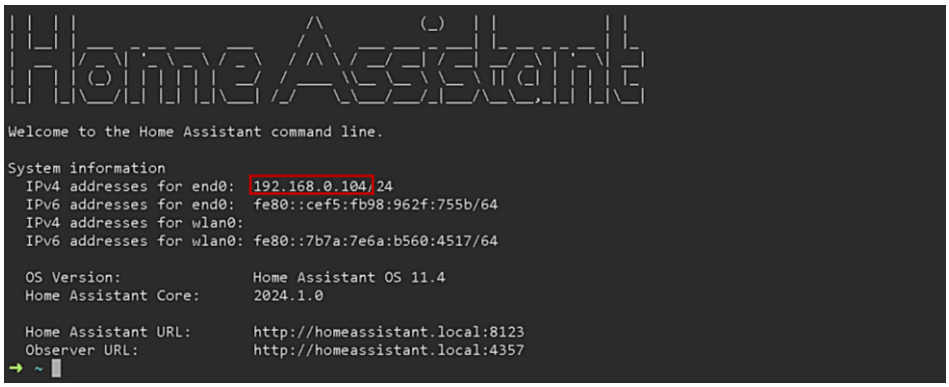


Hacemos clic en "ESCRIBIR" para comenzar el proceso de escritura. Este proceso borrará todos los datos existentes en la tarjeta microSD y escribirá la imagen de Home Assistant OS en ella. Esto puede tardar varios minutos.

Cuando aparezca una ventana con el texto “ESCRITURA EXITOSA”, podremos darle a Continuar, cerrar el software y expulsar el SD de forma segura. Luego, procederemos a ingresar la tarjeta al Raspberry Pi y conectar el minicomputador a la red eléctrica. Es recomendable conectar un cable de Ethernet antes de encender el dispositivo para que Home Assistant se inicialice a la primera.

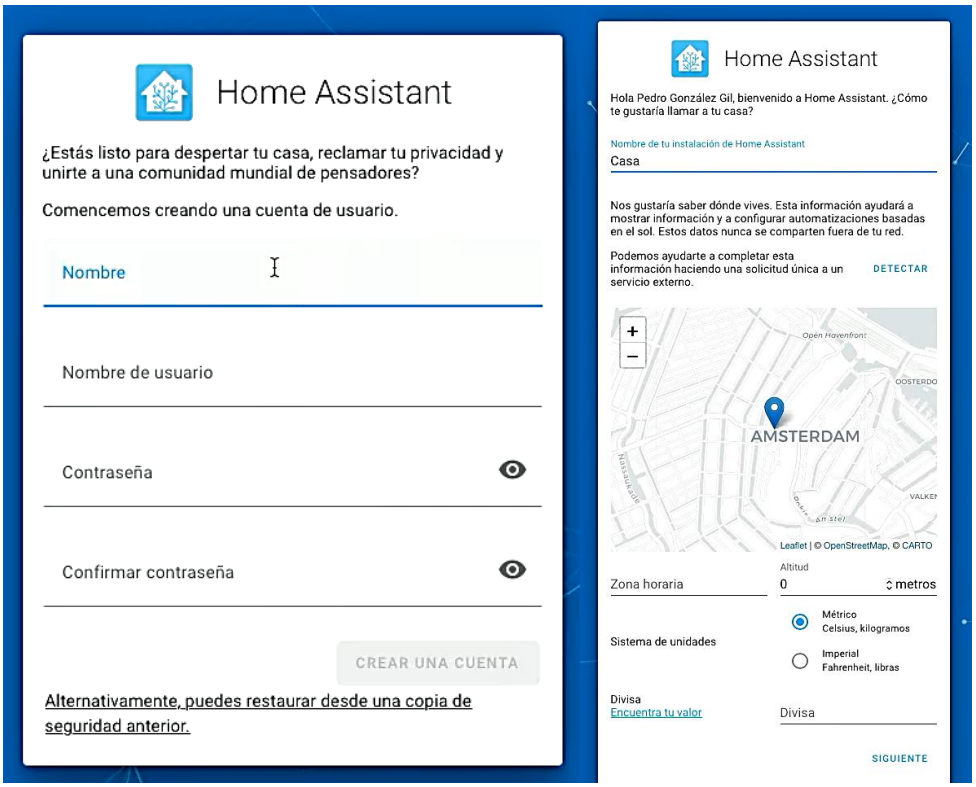


Cuando Home Assistant se inicia por primera vez, puede llegar a tardar hasta 20 minutos. Al completarse la instalación, podremos ingresar a la interfaz de usuario desde http://homeassistant.local:8123 desde otro dispositivo en la misma red desde un navegador web. Si este URL no funciona, podemos necesitar encontrar la dirección IP de nuestra Raspberry Pi en nuestro router y acceder usando esa IP, o podemos conectar un monitor a la salida HDMI para visualizar el terminal. Como se observa en mi consola, también puedo acceder a Home Assistant desde http://192.168.0.104:8123.



El primer paso es crear una cuenta de administrador para Home Assistant. Esto implica proporcionar un nombre de usuario, una contraseña y

un nombre para nuestra instancia de Home Assistant. Esta cuenta será la que usemos para administrar Home Assistant y todos los dispositivos conectados. Tenemos algunas configuraciones que vienen desde configuración de la ubicación, de la zona horaria, configuración de dispositivos, etc.



Cuando acabemos de configurar, ya podremos utilizar Home Assistant en su totalidad desde esa cuenta u otras que podemos crear desde la interfaz.

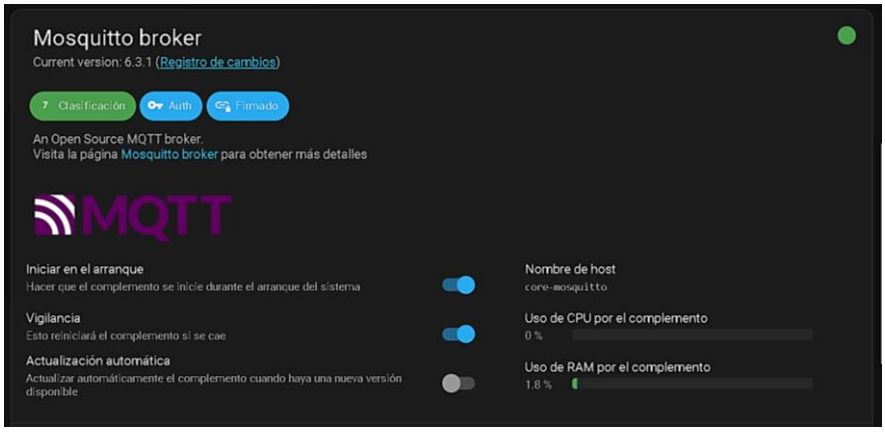
ETAPA 2: Instalación del Nodo 1 Wi-Fi

La instalación del Nodo 1 involucra dos procedimientos distintos. La configuración del bróker MQTT en Home Assistant, como de la definición de los sensores, y la programación del Nodo ESP32 en Arduino IDE.

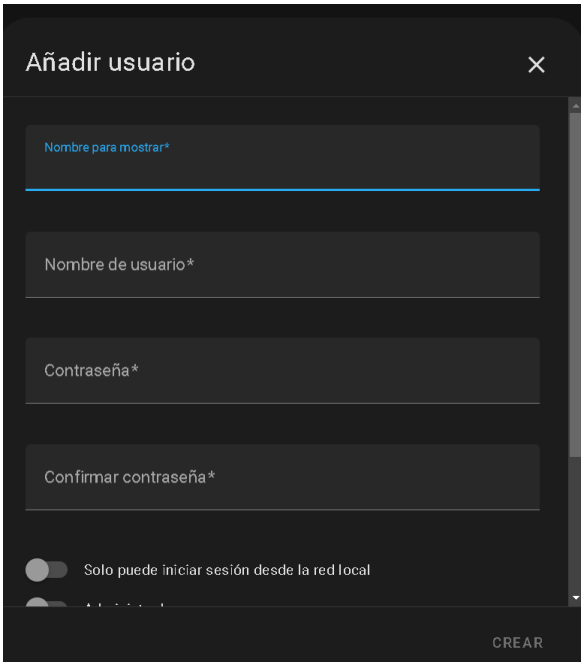
Procedimiento en Home Assistant

Para comenzar, necesitamos configurar el bróker MQTT en Home Assistant. Generalmente, utilizamos Mosquitto bróker por la cantidad de documentación que existe sobre este complemento.

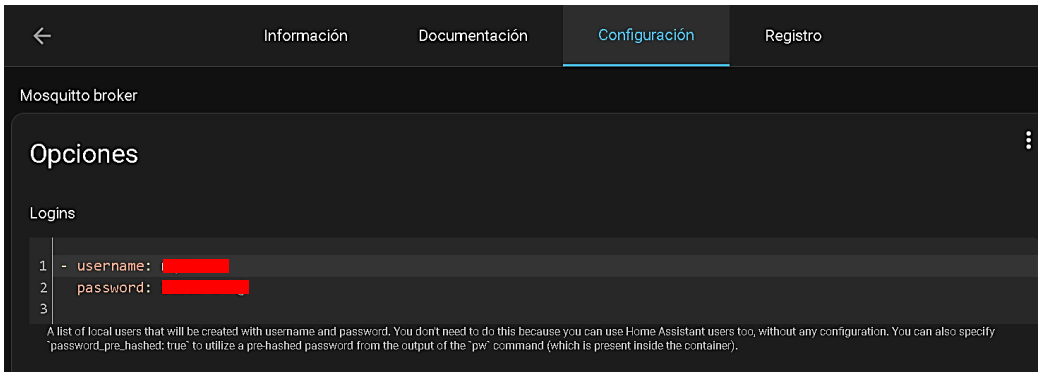
Nos dirigimos a Configuración > Complementos, buscamos Mosquitto broker y damos clic en “instalar”, abajo a la derecha.



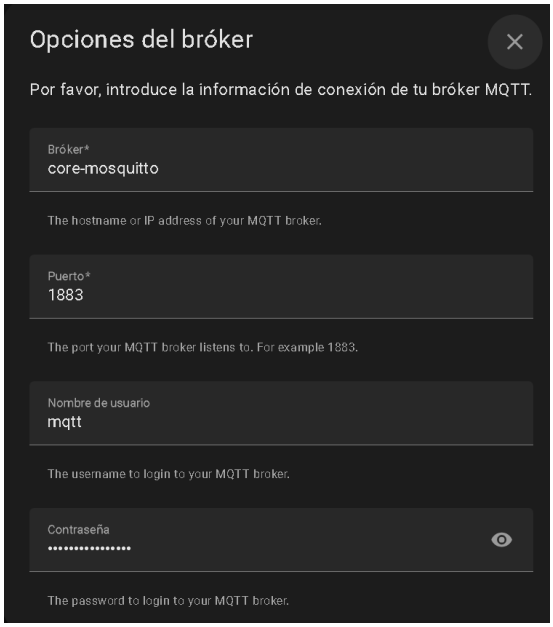
Una vez instalado, es importante crear un usuario dedicado para MQTT en Configuración > Personas > Usuarios. Esto se hace desde la configuración de Home Assistant.



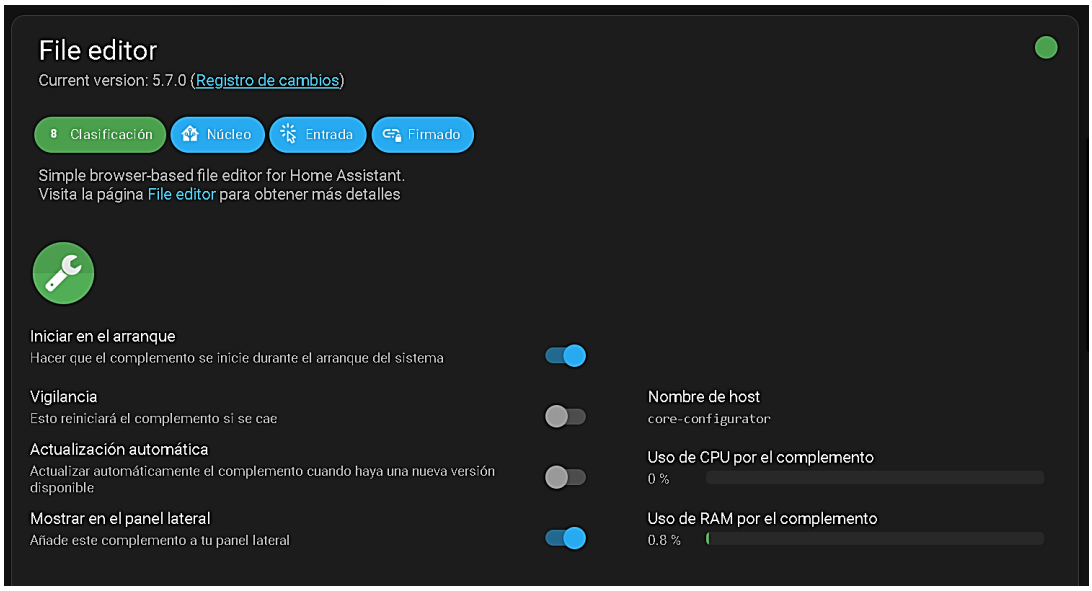
Luego, volvemos a el complemento de Mosquitto MQTT para ingresar el usuario y contraseña de nuestro usuario dedicado a MQTT.



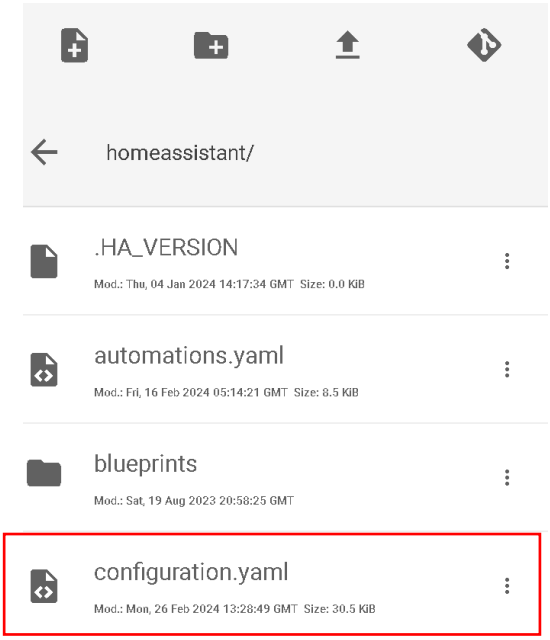
En Integraciones, configuramos MQTT utilizando los mismos datos del usuario que creamos, y le asignamos un Bróker y un puerto.



A partir de esto, ya tendremos MQTT funcionando en nuestra red. Ahora hay que configurar los sensores en base a los tópicos MQTT que el Nodo 1 creará. Para esto, instalamos un complemento que nos ayude a editar los archivos de la configuración de Home Assistant. El que usé se llama “File editor”



Ya con el complemento instalado, ingresamos a él. Después, en la barra de navegación que se encuentra en la izquierda, buscamos el archivo "configuration.yaml", este archivo contiene toda la configuración del Home Assistant.



En este archivo, definiremos los sensores y actuadores en base a los tópicos MQTT. Para esto, ingresamos el siguiente código (el cual se encuentra en formato Jinja2).

```yaml
mqtt:
  - sensor:
      name: "Nodo1 Voltaje"
      state_topic: "esp32-wifi/PZEM/Voltaje"
      icon: mdi:sine-wave
      unit_of_measurement: "V"

  - sensor:
      name: "Nodo1 Corriente"
      state_topic: "esp32-wifi/PZEM/Corriente"
      icon: mdi:current-ac
      unit_of_measurement: "A"

  - sensor:
      name: "Nodo1 Frecuencia"
      state_topic: "esp32-wifi/PZEM/Frecuencia"
      icon: mdi:current-ac
      unit_of_measurement: "Hz"

  - sensor:
      name: "Nodo1 Factor de Potencia"
      state_topic: "esp32-wifi/PZEM/FactorDePotencia"
      icon: mdi:current-ac

  - sensor:
      name: "Nodo1 Energia"
      state_topic: "esp32-wifi/PZEM/Energia"
      icon: mdi:lightning-bolt
      unit_of_measurement: "kWh"

  - sensor:
      name: "Nodo1 Potencia"
      state_topic: "esp32-wifi/PZEM/Potencia"
      icon: mdi:flash
      unit_of_measurement: "W"

  - sensor:
      name: "Nodo1 Humedad"
      state_topic: "esp32-wifi/DHT11/Humedad"
      icon: mdi:water-percent
      unit_of_measurement: "%"

  - sensor:
      name: "Nodo1 Temperatura"
      state_topic: "esp32-wifi/DHT11/Temperatura"
      icon: mdi:thermometer
      unit_of_measurement: "°C"

  - switch:
      unique_id: esp1_rele
      name: "Nodo1 Rele"
      command_topic: "esp32-wifi/control"
      payload_on: "1"
      payload_off: "11"
      optimistic: true
      qos: 0
      retain: 1
      enabled_by_default: 1

  - button:
      unique_id: esp1_reiniciarEnergia
      name: "Nodo1 Reiniciar Energia"
      command_topic: "esp32-wifi/control"
      payload_press: "111"
      qos: 0
      retain: false
      entity_category: "config"
      device_class: "restart"

```

En él, estamos configurando los sensores con su respectivo id, nombre e ícono, el interruptor para encender y apagar el relé, y un botón para reiniciar la energía almacenada en el PZEM.

Procedimiento en Arduino IDE

En primer lugar, necesitamos instalar Arduino IDE, el cual podemos encontrar en este link. Después, tendremos que instalar la librería dedicada para ESP32 en el IDE. Para esto, nos dirigimos a Archivo, Preferencias, Gestor de URLs Adicionales de Tarjetas, y pegamos el siguiente URL:

```
https://espressif.github.io/arduino-esp32/package_esp32_dev_index.json
```

Después, entramos a Herramientas, Placa, y entramos a Gestor de Tarjetas. Dentro, buscamos esp32 e instalamos la versión 2.0.10 (es la versión en la que el código está escrito, en versiones más recientes algunas funciones suelen cambiar y dará problemas al momento de querer compilar).

Ahora podemos empezar a escribir el código. Utilizaremos las librerías WiFi.h, FreeRTOS.h, AsyncMqttClient.h, DHT.h y PZEM004Tv30.h, los links necesarios se encuentran en los vínculos de cada librería mencionada.

El código utilizado en este nodo es el siguiente:

```
#include <WiFi.h> // Incluye la biblioteca para manejar la conexión WiFi del ESP32.

extern "C" { // Permite la inclusión de bibliotecas escritas en C dentro de un código C++.
#include "freertos/FreeRTOS.h" // Incluye las definiciones básicas de FreeRTOS para multitarea.
#include "freertos/timers.h" // Incluye las definiciones para manejo de temporizadores en FreeRTOS.
}

#include <AsyncMqttClient.h> // Incluye la biblioteca para manejo de cliente MQTT de forma asíncrona.

#include "DHT.h" // Incluye la biblioteca para el sensor de temperatura y humedad DHT.

#include <PZEM004Tv30.h> // Incluye la biblioteca para el sensor de potencia PZEM-004Tv3.0.

// Definiciones de constantes para configuración
```

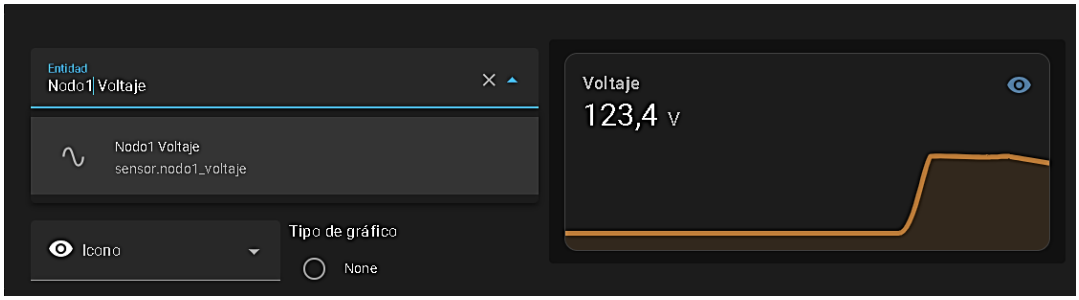
```
15. #define PIN_DHT 4 // Pin donde se conecta el sensor DHT.
16. #define TIPO_DHT DHT11 // Define el tipo de sensor DHT utilizado.
17.
18. #define SSID_WIFI "CN's" // Nombre de la red WiFi.
19. #define CONTRASENA_WIFI "" // Contraseña de la red WiFi.
20.
21. #define HOST_MQTT IPAddress(192, 168, 0, 104) // Dirección IP del servidor MQTT.
22. #define PUERTO_MQTT 1883 // Puerto utilizado por el servidor MQTT.
23.
24. // Topics para la publicación de datos a través de MQTT.
25. #define MQTT_PUBLICAR_VOLTAJE "esp32-wifi/PZEM/Voltaje"
26. #define MQTT_PUBLICAR_CORRIENTE "esp32-wifi/PZEM/Corriente"
27. #define MQTT_PUBLICAR_FRECUENCIA "esp32-wifi/PZEM/Frecuencia"
28. #define MQTT_PUBLICAR_FACTOR_POTENCIA "esp32-wifi/PZEM/FactorDePotencia"
29. #define MQTT_PUBLICAR_POTENCIA "esp32-wifi/PZEM/Potencia"
30. #define MQTT_PUBLICAR_ENERGIA "esp32-wifi/PZEM/Energia"
31. #define MQTT_PUBLICAR_TEMPERATURA "esp32-wifi/DHT11/Temperatura"
32. #define MQTT_PUBLICAR_HUMEDAD "esp32-wifi/DHT11/Humedad"
33.
34. // Configuración de pines y variables globales
35. int pinRele = 13; // Pin para el rele
36. int pinLed = 26; // Pin para el LED
37.
38. PZEM004Tv30 pzem(Serial2, 16, 17); // Instancia del sensor PZEM usando Serial2.
39. DHT dht(PIN_DHT, TIPO_DHT); // Instancia del sensor DHT.
40.
41. // Variables para almacenar las mediciones de los sensores.
42. float voltaje = 0;
43. float corriente = 0;
44. float potencia = 0;
45. float energia = 0;
46. float frecuencia = 0;
47. float factorPotencia = 0;
48.
49. float humedad = 0;
50. float temperatura = 0;
51.
52. float energiaAnterior = 0; // Almacena la última medición de energía para comparaciones.
53.
54. // Cliente MQTT y temporizadores para reconexión
55. AsyncMqttClient clienteMqtt;
56. TimerHandle_t temporizadorReconexionMqtt;
57. TimerHandle_t temporizadorReconexionWifi;
58.
59. unsigned long millisAnterior = 0; // Controla el intervalo de tiempo para acciones
repetitivas.
60. const long intervalo = 5000; // Intervalo en milisegundos para la publicación de datos.
61.
62. // Funciones para conexión a WiFi y MQTT, y manejo de eventos.
63. void conectarAWifi() {
64.   WiFi.begin(SSID_WIFI, CONTRASENA_WIFI); // Inicia la conexión a la red WiFi.
65. }
66.
67. void conectarAMqtt() {
68.   clienteMqtt.connect(); // Inicia la conexión al servidor MQTT.
69. }
70.
71. void eventoWiFi(WiFiEvent_t evento) {
72.     switch (evento) {
73.     case SYSTEM_EVENT_STA_GOT_IP: // Al obtener IP, intenta conectar a MQTT.
74.       conectarAMqtt();
75.       break;
76.     case SYSTEM_EVENT_STA_DISCONNECTED: // Al desconectar, reinicia los temporizadores
de reconexión.
77.       xTimerStop(temporizadorReconexionMqtt, 0);
78.       xTimerStart(temporizadorReconexionWifi, 0);
79.       break;
80.   }
81. }
82.
```

```
// Callbacks para eventos MQTT (conexión, desconexión, suscripción, publicación de mensajes).
void alConectarMqtt(bool sesionPresente) {
  // Suscribe a un topic específico al conectar.
  uint16_t paqueteIdSub = clienteMqtt.subscribe("esp32-wifi/control", 2);
}

void alDesconectarMqtt(AsyncMqttClientDisconnectReason motivo) {
  // Intenta reconectar si aún está conectado a WiFi.
  if (WiFi.isConnected()) {
    xTimerStart(temporizadorReconexionMqtt, 0);
  }
}

// Estas funciones manejan eventos de suscripción, publicación y mensajes MQTT.
void alSuscribirseMqtt(uint16_t paqueteId, uint8_t qos) {}
void alPublicarMqtt(uint16_t paqueteId) {}
void alDesuscribirseMqtt(uint16_t paqueteId) {}
 void alRecibirMensajeMqtt(char* tema, char* carga, AsyncMqttClientMessageProperties propiedades, size_t longitud, size_t indice, size_t total) {
  // Aquí se manejarían los mensajes recibidos, por ejemplo, para controlar el relé.
}

void setup() {
  Serial.begin(115200); // Configura la velocidad de la comunicación serial.

  pinMode(pinRele, OUTPUT); // Establece el pin del relé como salida.
  pinMode(pinLed, OUTPUT); // Establece el pin del LED como salida.
  digitalWrite(13,  1); // Estado inicial del relé.

  // Configura temporizadores para reconexión automática a WiFi y MQTT.
   temporizadorReconexionMqtt = xTimerCreate("temporizadorMqtt", pdMS_TO_TICKS(2000), pdFALSE, (void*)0, reinterpret_cast<TimerCallbackFunction_t>(conectarAMqtt));
   temporizadorReconexionWifi = xTimerCreate("temporizadorWifi", pdMS_TO_TICKS(2000), pdFALSE, (void*)0, reinterpret_cast<TimerCallbackFunction_t>(conectarAWifi));

  WiFi.onEvent(eventoWiFi); // Registra el manejador de eventos WiFi.

  // Configura los callbacks para distintos eventos MQTT.
  clienteMqtt.onConnect(alConectarMqtt);
  clienteMqtt.onDisconnect(alDesconectarMqtt);
  clienteMqtt.onSubscribe(alSuscribirseMqtt);
  clienteMqtt.onUnsubscribe(alDesuscribirseMqtt);
  clienteMqtt.onMessage(alRecibirMensajeMqtt);
  clienteMqtt.onPublish(alPublicarMqtt);

  clienteMqtt.setServer(HOST_MQTT, PUERTO_MQTT); // Establece el servidor MQTT.
   clienteMqtt.setCredentials("mqtt", "12125454qwxdxdA@"); // Configura credenciales para MQTT.

  conectarAWifi(); // Inicia la conexión WiFi.

  dht.begin(); // Inicia el sensor DHT.
}

 void loop() {
  // Verifica si el rele está desactivado.
  if(digitalRead(pinRele) == 0){
   unsigned long millisActual = millis(); // Almacena el tiempo actual en milisegundos.

    // Lee los valores de los sensores.
    humedad = dht.readHumidity(); // Lee la humedad del sensor DHT.
    temperatura = dht.readTemperature(); // Lee la temperatura del sensor DHT.

    // Lee los valores del sensor PZEM.
    voltaje = pzem.voltage();
    corriente = pzem.current();
    potencia = pzem.power();
    energia = pzem.energy();
    frecuencia = pzem.frequency();
```

```
148.     factorPotencia = pzem.pf();
149.
150.     // Enciende el LED si la energía ha aumentado desde la última medición.
151.     if (energia > energiaAnterior) {
152.       digitalWrite(pinLed, HIGH); // Enciende el LED.
153.       tiempoInicio = millisActual; // Actualiza el momento en que se encendió el LED.
154.     }
155.
156.     // Apaga el LED después del intervalo especificado.
157.     if (millisActual - tiempoInicio >= intervalo2) {
158.       digitalWrite(pinLed, LOW); // Apaga el LED.
159.     }
160.
161.     energiaAnterior = energia; // Actualiza la última medición de energía.
162.
163.     // Publica los valores a los topics de MQTT cada cierto intervalo.
164.     if (millisActual - millisAnterior >= intervalo) {
165.      millisAnterior = millisActual; // Actualiza el tiempo para la próxima publicación.
166.
167.       // Publica cada uno de los valores medidos en sus respectivos topics MQTT.
168.         uint16_t paqueteIdPub1 = clienteMqtt.publish(MQTT_PUBLICAR_VOLTAJE, 2, true,
String(voltaje).c_str());
169.        uint16_t paqueteIdPub2 = clienteMqtt.publish(MQTT_PUBLICAR_CORRIENTE, 2, true,
String(corriente, 3).c_str());
170.         uint16_t paqueteIdPub3 = clienteMqtt.publish(MQTT_PUBLICAR_POTENCIA, 2, true,
String(potencia, 2).c_str());
171.          uint16_t paqueteIdPub4 = clienteMqtt.publish(MQTT_PUBLICAR_ENERGIA, 2, true,
String(energia, 3).c_str());
172.         uint16_t paqueteIdPub5 = clienteMqtt.publish(MQTT_PUBLICAR_FACTOR_POTENCIA, 2,
true, String(factorPotencia, 3).c_str());
173.        uint16_t paqueteIdPub6 = clienteMqtt.publish(MQTT_PUBLICAR_FRECUENCIA, 2, true,
String(frecuencia).c_str());
174.       uint16_t paqueteIdPub7 = clienteMqtt.publish(MQTT_PUBLICAR_TEMPERATURA, 2, true,
String(temperatura).c_str());
175.          uint16_t paqueteIdPub8 = clienteMqtt.publish(MQTT_PUBLICAR_HUMEDAD, 2, true,
String(humedad).c_str());
176.     }
177.   }
178.   else{
179.     // esp_sleep_enable_timer_wakeup(10000000);
180.     // esp_light_sleep_start();
181.   }
182. }
183.
```

Debemos procurar ingresar correctamente la información de la red mqtt y wi-fi junto a sus contraseñas.

Finalmente, si encendemos el Nodo, podremos observar toda la información funcionando correctamente en la interfaz de Home Assistant. Simplemente tendremos que ordenar las entidades en alguna pestaña de la interfaz.

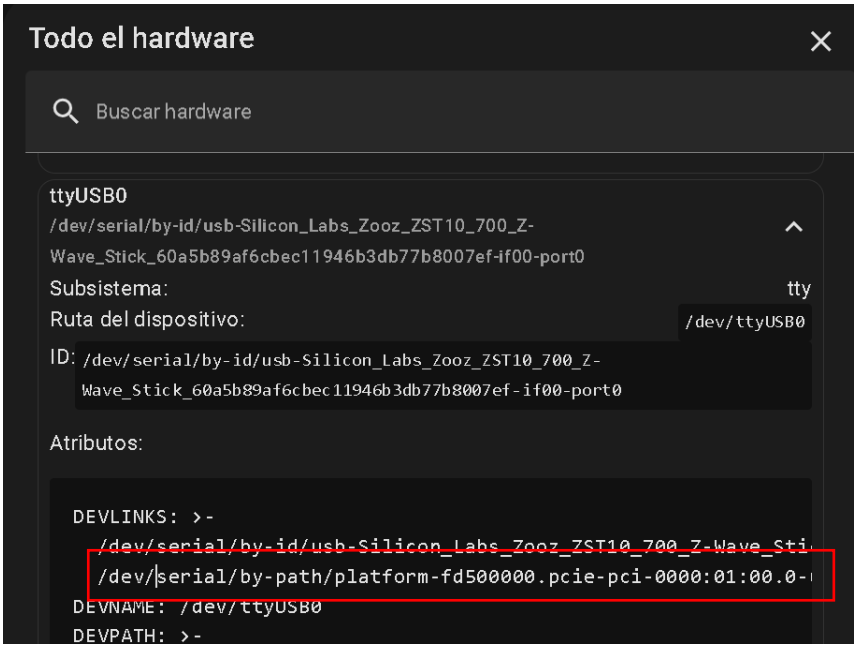


ETAPA 3: Instalación del Nodo 2 BLE

En la instalación del Nodo 2 el procedimiento cambiará, puesto que utilizaremos comunicación UART para recibir la información de la red BLE usando otro ESP32 como Gateway. Por lo cual, antes de nada, empezaremos con la configuración serial en Home Assistant.

Procedimiento en Home Assistant

Conectamos en ESP32 a uno de los puertos USB de la Raspberry y accedemos a Ajustes > Sistema > Hardware > Todo el Hardware. Ahí, buscaremos ttyUSB0 o ttyACM0, ese será nuestro dispositivo USB. Ahí copiaremos del segundo devlinks que aparece cuando desplegamos más información sobre el dispositivo.



Ingresamos al archivo Configuration.yaml desde Home Assistant utilizando la ya anterior instalada, herramienta de "File editor". En él, debemos configurar el puerto serial del ESP32. Para eso, utilizaremos el siguiente fragmento de código:

```
1.   - platform: serial
2.     name: Nodo2BLE
```

```
3.          serial_port:  /dev/serial/by-path/platform-fd500000.pcie-pci-0000:01:00.0-usb-
0:1.1.3:1.0-port0
4.     baudrate: 115200
5.
```

Es importante colocar el serial_port y el baudrate correcto para que el serial pueda ser leído correctamente.

Ahora, definiremos las variables medidas por el Nodo2, para esto, asignaremos los valores a los sensores de la siguiente manera:

```
 1.   - platform: template
 2.     sensors:
 3.       voltage2:
 4.         friendly_name: "Nodo2 Voltaje"
 5.         unique_id: "nodo2_voltaje"
 6.         unit_of_measurement: "V"
 7.        value_template: "{{ states('sensor.nodo2ble').split('V')[1].split('C')[0] | float
}}"
 8.       current2:
 9.         friendly_name: "Nodo2 Corriente"
10.         unique_id: "nodo2_corriente"
11.         unit_of_measurement: "A"
12.        value_template: "{{ states('sensor.nodo2ble').split('C')[1].split('P')[0] | float
}}"
13.       power2:
14.         friendly_name: "Nodo2 Potencia"
15.         unique_id: "nodo2_potencia"
16.         unit_of_measurement: "W"
17.        value_template: "{{ states('sensor.nodo2ble').split('P')[1].split('E')[0] | float
}}"
18.       energy2:
19.         friendly_name: "Nodo2 Energia"
20.         unique_id: "nodo2_energia"
21.         unit_of_measurement: "kWh"
22.        value_template: "{{ states('sensor.nodo2ble').split('E')[1].split('F')[0] | float
}}"
23.       frequency2:
24.         friendly_name: "Nodo2 Frecuencia"
25.         unique_id: "nodo2_frecuencia"
26.         unit_of_measurement: "Hz"
27.        value_template: "{{ states('sensor.nodo2ble').split('F')[1].split('M')[0] | float
}}"
28.       powerfactor2:
29.         friendly_name: "Nodo2 Factor de Potencia"
30.         unique_id: "nodo2_factordepotencia"
31.        value_template: "{{ states('sensor.nodo2ble').split('M')[1].split('Z')[0] | float
}}"
32.       humo2:
33.         friendly_name: "Nodo2 Humo"
34.         unique_id: "nodo2_humo"
35.        value_template: "{{ states('sensor.nodo2ble').split('Z')[1].split('L')[0] | float
}}"
36.       current2_calculated:
37.         friendly_name: "Nodo2 Corriente  Calculada"
38.         unique_id: "nodo2_corriente_calculada"
39.         unit_of_measurement: "A"
40.                                                 value_template:     "{{     0.0     if
(((states('sensor.power2')|float)*(states('sensor.powerfactor2')|float))|float) == 0.0 else
(((states('sensor.power2')|float)         /         ((states('sensor.powerfactor2')|float)
*(states('sensor.voltage2')|float))) |round(3)) }}"
41.       powerapp2_calculated:
42.         friendly_name: "Nodo2 Potencia Aparente"
```

```
43.         unique_id: "nodo2_potencia_aparente"
44.         unit_of_measurement: "VA"
45.                     value_template:  "{{  0.0  if  (((states('sensor.power2')|float)  *
(states('sensor.powerfactor2')|float))|float) == 0.0 else (( (states('sensor.power2')|float)
/ ( (states('sensor.powerfactor2')|float) )) | round(2)) }}"
46.       powerreact2_calculated:
47.         friendly_name: "Nodo2 Potencia Reactiva"
48.         unique_id: "nodo2_potencia_reactiva"
49.         unit_of_measurement: "VAr"
50.                                                 value_template:     "{{     0.0     if
(((states('sensor.power2')|float)*(states('sensor.powerfactor2')|float))|float)  ==0.0  or
(states('sensor.powerfactor2')|float)               ==1               else               (
((states('sensor.voltage2')|float)*((states('sensor.current2_calculated')|float))*((sin((a
cos((states('sensor.powerfactor2'))|round(3)))))))) | round(2))}}"
51.
```

El ESP32 enviará un String de la siguiente estructura "V120.5C0.004P0.04F60.00E0.002X1.00M", entonces la plantilla utilizada captura el valor flotante utilizando las letras como delimitadores. Por ejemplo, el sensor de voltage2, el valor que se define como del sensor, es el que se encuentra entre "V" y "C".

También, se utilizan las variables medidas por el sensor para calcular la potencia reactiva y aparente utilizando plantillas de la misma manera.

Para el relé y el reinicio de energía, habría que escribir en el serial para accionar las distintas ordenes escritas en el código. Para eso, estableceremos comandos que se enviarían por la n del sistema operativo.

```
1. shell_command:
2.     nodo2_rele_off: /bin/bash -c "echo -ne '\x01\xAB\xFF' > /dev/serial/by-id/usb-
Silicon_Labs_CP2102_USB_to_UART_Bridge_Controller_0001-if00-port0"
3.     nodo2_rele_on: /bin/bash -c "echo -ne '\x02\xAB\xFF' > /dev/serial/by-id/usb-
Silicon_Labs_CP2102_USB_to_UART_Bridge_Controller_0001-if00-port0"
4.
5.    nodo2_resetenergia: /bin/bash -c "echo -ne '\x05\xAB\xFF' > /dev/serial/by-id/usb-
Silicon_Labs_CP2102_USB_to_UART_Bridge_Controller_0001-if00-port0"
6.
```

'/bin/bash -c' Indica que el comando debe ejecutarse utilizando el intérprete de comandos Bash. El parámetro -c permite pasar un string de comandos para ejecutar en Bash. 'echo' es un comando que imprime su argumento (en este caso, una secuencia de bytes) a la salida estándar. '\x01\xAB\xFF' es la secuencia de bits que espera el ESP32 para apagar el relé, por ejemplo. Y lo que continúa luego del ">" corresponde al id del puerto serial.

Ahora, solo tendríamos que definir los interruptores utilizando la entidad de comando que creamos.

```
1. switch:
```

```
  - platform: template
    switches:
      ################################## NODO 2

      nodo2_rele:
        turn_on:
          service: shell_command.nodo2_rele_on
        turn_off:
          service: shell_command.nodo2_rele_off
      nodo2_resetenergia:
        turn_on:
          service: shell_command.nodo2_resetenergia
        turn_off:
          service: shell_command.nodo2_resetenergia

```

Procedimiento en Arduino IDE

Ahora programaremos el ESP32 para que utilice su funcionalidad BLE, para eso, utilizaremos bibliotecas ya integradas con la librería de esp32 del IDE. En el ESP32 del Nodo, se usó el siguiente código:

```
// Incluye las bibliotecas necesarias para el funcionamiento de BLE y el sensor PZEM004Tv30
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <PZEM004Tv30.h>

// Definición de pines
PZEM004Tv30 pzem(Serial2, 16, 17); // Instancia del sensor PZEM usando Serial2.

const int pinRele = 13; // Pin para el relé
int pinLed = 26; // Pin para el LED

int pinHumo  = 4; // Pin para el sensor de humo
int estadoHumo = 0; // Estado actual del sensor de humo (0 = no hay humo, 1 = hay humo)

// Variables para almacenar las mediciones del sensor PZEM
float voltaje = 0;
float corriente = 0;
float potencia = 0;
float energia = 0;
float frecuencia = 0;
float factorPotencia = 0;

int numeroPaquete = 1; // Variable para controlar el paquete de datos a enviar vía BLE

float energiaAnterior = 0; // Almacena la última medición de energía para comparaciones

// Variables para control de tiempo
unsigned long tiempoActual = 0;
unsigned long tiempoAnterior = 0;

// Variables para BLE
BLEServer *pServer = NULL;
BLECharacteristic *pCharacteristic = NULL;
bool dispositivoConectado = false;
bool dispositivoConectadoAnterior = false;

// UUIDs para el servicio y la característica BLE
#define SERVICE_UUID        "4fafc201-1fb5-459e-8fcc-c5c9c331914c"
```

```
41. #define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"
42.
43. // Clase para manejar eventos de conexión y desconexión del servidor BLE
44. class MyServerCallbacks : public BLEServerCallbacks {
45.     void onConnect(BLEServer* pServer) {
46.       dispositivoConectado = true;
47.     };
48.
49.     void onDisconnect(BLEServer* pServer) {
50.       dispositivoConectado = false;
51.     }
52. };
53.
54. unsigned long millisAnterior2 = 0;
55. const unsigned long intervalo2 = 5000; // Intervalo para la verificación de humo
56.
57. void setup() {
58.   Serial.begin(115200); // Inicia la comunicación serial
59.
60.   // Configura los pines
61.   pinMode(pinRele, OUTPUT);
62.   pinMode(pinLed, OUTPUT);
63.   pinMode(pinHumo, INPUT);
64.   digitalWrite(pinRele,  1); // Inicializa el relé apagado
65.
66.   // Inicializa el servidor BLE con el nombre "ESP32_BLE_Server"
67.   BLEDevice::init("ESP32_BLE_Server");
68.   pServer = BLEDevice::createServer();
69.     pServer->setCallbacks(new MyServerCallbacks()); // Establece los callbacks para
conexión/desconexión
70.
71.   // Crea un servicio BLE
72.   BLEService *pService = pServer->createService(SERVICE_UUID);
73.
74.  // Crea una característica BLE en el servicio, con propiedades de lectura, notificación
y escritura
75.   pCharacteristic = pService->createCharacteristic(
76.                       CHARACTERISTIC_UUID,
77.                       BLECharacteristic::PROPERTY_READ |
78.                       BLECharacteristic::PROPERTY_NOTIFY |
79.                       BLECharacteristic::PROPERTY_WRITE
80.                     );
81.  pCharacteristic->addDescriptor(new BLE2902()); // Añade descriptor para notificaciones
82.
83.   pService->start(); // Inicia el servicio BLE
84.
85.   // Configura y comienza la publicidad del servidor BLE
86.   BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
87.   pAdvertising->addServiceUUID(SERVICE_UUID);
88.   pAdvertising->setScanResponse(true);
89.   pAdvertising->setMinPreferred(0x06);
90.   BLEDevice::startAdvertising();
91.   Serial.println("Esperando una conexión...");
92.
93.   tiempoAnterior = millis(); // Guarda el tiempo actual
94. }
95.
96. void loop() {
97.   // Si la energía medida es mayor a la anterior, enciende y apaga el LED rápidamente
98.   if (energia > energiaAnterior) {
99.     digitalWrite(pinLed, HIGH);  // Enciende el LED
100.    delay(50);                   // Espera 50 milisegundos
101.    digitalWrite(pinLed, LOW);    // Apaga el LED
102.  }
103.  energiaAnterior = energia; // Actualiza la última medición de energía
104.
105.  // Lee los valores del sensor PZEM
106.  voltaje = pzem.voltage();
107.  corriente = pzem.current();
108.  potencia = pzem.power();
```

```
109.    energia = pzem.energy();
110.    frecuencia = pzem.frequency();
111.    factorPotencia = pzem.pf();
112.
113.    // Prepara los datos a enviar mediante BLE, agrupándolos en tres paquetes diferentes
114.    String dato1 = "QWE" + String(voltaje) + "RTY" + String(corriente, 3)+ "UIO";
115.    String dato2 = "PAS" + String(potencia, 2) + "DFG" + String(energia, 3)+ "HJK";
116.     String dato3 = "LZX" + String(frecuencia) + "CVB" + String(factorPotencia)+ "NMQ" +
String(estadoHumo)+ "BTG";
117.
118.    String textoEnviar; // Variable para almacenar el texto a enviar
119.
120.    // Selecciona el paquete de datos a enviar
121.    if (numeroPaquete == 1) {
122.      textoEnviar = dato1;
123.    } else if (numeroPaquete == 2) {
124.      textoEnviar = dato2;
125.    } else if (numeroPaquete == 3) {
126.      textoEnviar = dato3;
127.    }
128.
129.    // Detecta cambio de estado en el sensor de humo y actualiza el estadoHumo
130.    if (digitalRead(pinHumo) == HIGH && estadoHumo == LOW) {
131.      estadoHumo = HIGH;
132.      millisAnterior2 = millis(); // Guarda el momento del cambio
133.    }
134.
135.    // Cambia estadoHumo a LOW después de 5 segundos sin detección de humo
136.    if (estadoHumo == HIGH && millis() - millisAnterior2 >= intervalo2) {
137.      estadoHumo = LOW;
138.    }
139.
140.    // Procesa los datos recibidos via BLE
141.    if (pCharacteristic->getValue().length() > 0) {
142.      std::string receivedData = pCharacteristic->getValue();
143.
144.      Serial.println(receivedData.length());
145.
146.      // Procesa los comandos recibidos para controlar el relé o resetear la energía del
PZEM
147.       if (receivedData.length() == 3 && receivedData[0] == 0x01 && receivedData[1] ==
0xAB && receivedData[2] == 0xFF) {
148.        digitalWrite(pinRele, 0); // Enciende el relé
149.      }
150.
151.       if (receivedData.length() == 3 && receivedData[0] == 0x02 && receivedData[1] ==
0xAB && receivedData[2] == 0xFF) {
152.        digitalWrite(pinRele, 1); // Apaga el relé
153.      }
154.
155.       if (receivedData.length() == 3 && receivedData[0] == 0x05 && receivedData[1] ==
0xAB && receivedData[2] == 0xFF) {
156.        pzem.resetEnergy(); // Resetea la energía medida por el PZEM
157.      }
158.
159.      pCharacteristic->setValue(""); // Limpia el valor de la característica después de
procesar el comando
160.    }
161.
162.    // Si hay un dispositivo conectado, gestiona la notificación de los datos
163.    if (dispositivoConectado) {
164.      tiempoActual = millis();
165.
166.      // Envía los datos cada 333ms
167.      if (tiempoActual >= tiempoAnterior + 333) {
168.        pCharacteristic->setValue(textoEnviar.c_str());
169.        pCharacteristic->notify();
170.
171.        numeroPaquete += 1; // Cambia al siguiente paquete de datos
172.        tiempoAnterior = tiempoActual; // Actualiza el tiempo para el próximo envío
```

```
    }

    // Reinicia el número de paquete después de enviar los tres paquetes
    if (numeroPaquete == 4) {
      numeroPaquete = 1;
    }
  }

  // Reinicia la publicidad BLE si un dispositivo se desconecta
  if (!dispositivoConectado && dispositivoConectadoAnterior) {
    delay(500); // Pequeña pausa antes de reiniciar la publicidad
    pServer->startAdvertising(); // Reinicia la publicidad
    Serial.println("Esperando una conexión...");
    dispositivoConectadoAnterior = dispositivoConectado;
  }

  // Actualiza el estado del dispositivo conectado
  if (dispositivoConectado && !dispositivoConectadoAnterior) {
    dispositivoConectadoAnterior = dispositivoConectado;
  }
}
```

Es importante tomar en cuenta los UUIDs utilizados para la comunicación BLE, puesto que el Gateway usará el mismo para que haya comunicación entre ambos dispositivos. En el segundo ESP32, el código que subiremos será el siguiente:

```
#include "BLEDevice.h"

// UUIDs del servicio y la característica a la que queremos conectarnos
static BLEUUID uuidServicio("4fafc201-1fb5-459e-8fcc-c5c9c331914c");
static BLEUUID uuidCaracteristica("beb5483e-36e1-4688-b7f5-ea07361b26a8");

// Variables de estado para la conexión y el escaneo
static boolean intentarConectar = false;
static boolean conectado = false;
static boolean intentarEscanear = true;
static BLERemoteCharacteristic* caracteristicaRemota;
static BLEAdvertisedDevice* miDispositivo;

// Variables para control de tiempo
unsigned long millisPrevios = 0;
const long intervalo = 5000;

// Mensajes divididos en líneas para ser reconstruidos
String lineaMensaje1, lineaMensaje2, lineaMensaje3;

// Callback cuando se recibe una notificación de la característica
static void callbackNotificaciones(
  BLERemoteCharacteristic* pCaracteristicaRemota,
  uint8_t* pData,
  size_t length,
  bool isNotify) {
    if (length > 0 && pData[0] != '\0') {
        String mensaje = String((char*)pData);
        if (mensaje.startsWith("QWE")) {
            lineaMensaje1 = mensaje;
        } else if (mensaje.startsWith("PAS")) {
            lineaMensaje2 = mensaje;
```

```
        } else if (mensaje.startsWith("LZX")) {
            lineaMensaje3 = mensaje;
        }

        // Procesamiento del mensaje combinado
        String mensajeCombinado = lineaMensaje1 + lineaMensaje2 + lineaMensaje3;
        // Función lambda para extraer subcadenas entre delimitadores
         auto extraerEntre = [](const String &str, const String &inicio, const String &fin) -> String {
            int idxInicio = str.indexOf(inicio) + inicio.length();
            int idxFin = str.indexOf(fin, idxInicio);
            return idxFin != -1 ? str.substring(idxInicio, idxFin) : "";
        };

        // Ejemplo de cómo usar la función lambda `extraerEntre`
        String valorExtraido = extraerEntre(mensajeCombinado, "QWE", "RTY");
        Serial.println(valorExtraido);
    }
}

// Callbacks para el cliente BLE para manejar eventos de conexión y desconexión
class MiCallbackCliente : public BLEClientCallbacks {
  void onConnect(BLEClient* pcliente) {
    conectado = true;
    intentarEscanear = false;
    Serial.println("Conectado a BLE");
  }

  void onDisconnect(BLEClient* pcliente) {
    conectado = false;
    intentarEscanear = true;
    Serial.println("Desconectado de BLE, reiniciando escaneo...");
  }
};

// Función para intentar conectar al servidor BLE
bool conectarAlServidor() {
    Serial.print("Formando una conexión a ");
    Serial.println(miDispositivo->getAddress().toString().c_str());

    BLEClient* pCliente = BLEDevice::createClient();
    pCliente->setClientCallbacks(new MiCallbackCliente());

    // Conecta al dispositivo BLE encontrado
    if (!pCliente->connect(miDispositivo)) {
        Serial.println("La conexión ha fallado");
        return false;
    }

    // Obtiene el servicio BLE remoto
    BLERemoteService* pServicioRemoto = pCliente->getService(uuidServicio);
    if (pServicioRemoto == nullptr) {
        Serial.print("No se encontró el servicio UUID: ");
        Serial.println(uuidServicio.toString().c_str());
        pCliente->disconnect();
        return false;
    }

    // Obtiene la característica BLE remota
    caracteristicaRemota = pServicioRemoto->getCharacteristic(uuidCaracteristica);
    if (caracteristicaRemota == nullptr) {
        Serial.print("No se encontró la característica UUID: ");
        Serial.println(uuidCaracteristica.toString().c_str());
        pCliente->disconnect();
        return false;
    }

    // Registra el callback para notificaciones de la característica
                    if    (caracteristicaRemota->canNotify())    caracteristicaRemota->registerForNotify(callbackNotificaciones);
```

```
101.
102.     conectado = true;
103.     return true;
104. }
105.
106. // Callbacks para dispositivos anunciados durante el escaneo BLE
107. class MiCallbackDispositivoAnunciado: public BLEAdvertisedDeviceCallbacks {
108.     void onResult(BLEAdvertisedDevice advertisedDevice) {
109.                                     if     (advertisedDevice.haveServiceUUID()      &&
advertisedDevice.isAdvertisingService(uuidServicio)) {
110.         BLEDevice::getScan()->stop();
111.         miDispositivo = new BLEAdvertisedDevice(advertisedDevice);
112.         intentarConectar = true;
113.         intentarEscanear = false;
114.       }
115.     }
116. };
117.
118. void setup() {
119.   Serial.begin(115200);
120.   Serial.println("Iniciando cliente BLE...");
121.
122.   BLEDevice::init("");
123.   BLEScan* pBLEScan = BLEDevice::getScan();
124.   pBLEScan->setAdvertisedDeviceCallbacks(new MiCallbackDispositivoAnunciado());
125.   pBLEScan->setActiveScan(true);
126.   pBLEScan->start(5, false);
127. }
128.
129. void loop() {
130.   // Manejo de la conexión
131.   if (intentarConectar && !conectado) {
132.     if (conectarAlServidor()) {
133.       Serial.println("Conectado al servidor BLE");
134.       intentarConectar = false;
135.     } else {
136.       Serial.println("No se pudo conectar al servidor");
137.       intentarEscanear = true;
138.     }
139.   }
140.
141.   // Reinicia el escaneo si es necesario
142.   if (intentarEscanear && !conectado) {
143.     BLEDevice::getScan()->start(0); // 0 = escaneo continuo
144.   }
145. }
146.
```

Con los dos ESP32 funcionando, habrá una comunicación inmediata cuando se encuentren en un rango aceptable. Ahora, el ESP32 que sirve como Gateway se conecta al mismo puerto USB que antes, y los sensores deberían publicarse exitosamente en Home Assistant, si es que no lo hacen, tal vez el ID del puerto USB cambió, sería cuestión de cambiar esta información en el configuration.yaml tanto de los sensores como de los actuadores.

ETAPA 4: Instalación del Nodo 3 GSM

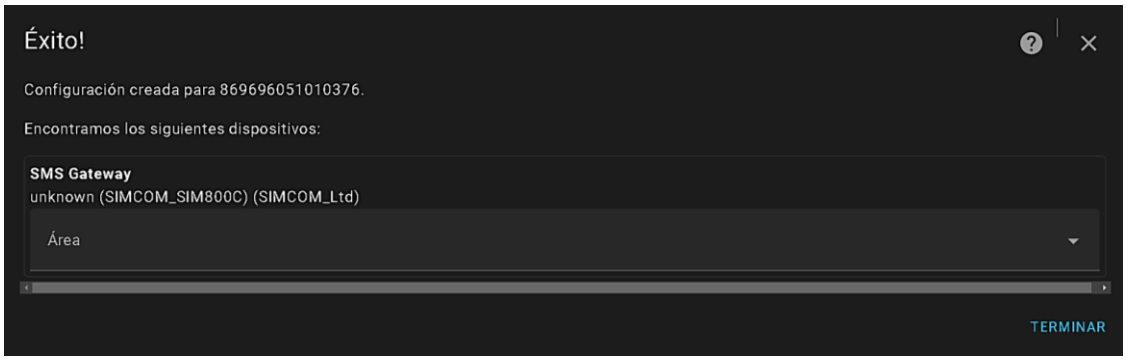
En este Nodo utilizamos el módulo SIM800L para la comunicación GSM, por lo cual, usamos uno que envíe los datos desde el módulo, y otro que funcione como Gateway desde Home Assistant.

Procedimiento en Home Assistant

En primer lugar, conectamos el stick USB de SIM800 a un puerto USB, como se mencionó anteriormente, Home Assistant deberá reconocerlo en la sección de Hardware.

Copiamos el segundo DEVLINK y nos vamos a Ajustes > Dispositivos y Servicios > Añadir una Integración. Ahí, buscamos "SMS notifications via GSM-modem", hacemos clic y tendremos que agregar el ID del dispositivo que copiamos. Lo pegamos y le damos a Enviar.

Cargará por unos segundos y finalmente detectará el modelo del dispositivo, el cual es "SMS Gateway", finalmente le damos a "terminar".



Ahora, crearemos una automatización que nos ayudará a almacenar los datos recibidos como un sensor y además, definir los actuadores. Para eso, nos vamos a "automatizatinos.yaml" con el File Editor y copiamos el siguiente código:

```yaml
- alias: Forward SMS
  trigger:
  - platform: event
    event_type: sms.incoming_sms
  action:
  - service: script.notify_sms_user1
    data:
      message: 'From: {{trigger.event.data.phone}}

        {{trigger.event.data.text}}  mode: single

        '
  id: fd457874dba245d3a2166eb3d2be48cf
- alias: Almacenar SMS
  trigger:
  - platform: event
    event_type: sms.incoming_sms
  action:
  - service: input_text.set_value
    data_template:
      entity_id: input_text.serial_history
      value: '{{ trigger.event.data.text }}'
  id: af497b3cb6e94c93ab4a197ae3bcf8dd



- id: '1705419499042'
  alias: Encender Rele Nodo 3
  description: ''
  trigger:
  - platform: state
    entity_id:
    - input_boolean.nodo3switch
    from: 'off'
    to: 'on'
  condition: []
  action:
  - service: notify.sms
    data:
      message: '@1#'
      target: '+593939738662'
      data:
        unicode: False
  mode: single


- id: '1705419538882'
  alias: Apagar Rele Nodo 3
  description: ''
  trigger:
  - platform: state
    entity_id:
    - input_boolean.nodo3switch
    from: 'on'
    to: 'off'
  condition: []
  action:
  - service: notify.sms
    data:
      message: '@2#'
      target: '+593939738662'
      data:
```

```
        unicode: False
  mode: single

- id: '1705419981835'
  alias: resetear energia Nodo 3
  description: ''
  trigger:
  - platform: state
    entity_id:
    - input_button.reset_energia_nodo_3
  condition: []
  action:
  - service: notify.sms
    data:
      message: '@3#'
      target: '+593939738662'
      data:
        unicode: False
  mode: single

```

El mensaje recibido finalmente se almacena en la entidad "input_text.serial_history", con esta, necesitamos crear los sensores de la plantilla de la misma forma que anteriormente

```
  - platform: template
    sensors:
      voltage3:
        friendly_name: "Nodo3 Voltaje"
        unique_id: "nodo3_voltaje"
        unit_of_measurement: "V"
                                                          value_template:       "{{
states('input_text.serial_history').split('V')[1].split('C')[0] | float }}"

      current3:
        friendly_name: "Nodo3 Corriente"
        unique_id: "nodo3_corriente"
        unit_of_measurement: "A"
                                                          value_template:       "{{
states('input_text.serial_history').split('C')[1].split('P')[0] | float }}"

      power3:
        friendly_name: "Nodo3 Potencia"
        unique_id: "nodo3_potencia"
        unit_of_measurement: "W"
                                                          value_template:       "{{
states('input_text.serial_history').split('P')[1].split('E')[0] | float }}"

      energy3:
        friendly_name: "Nodo3 Energia"
        unique_id: "nodo3_energia"
        unit_of_measurement: "kWh"
                                                          value_template:       "{{
states('input_text.serial_history').split('E')[1].split('F')[0] | float }}"

      frequency3:
        friendly_name: "Nodo3 Frecuencia"
        unique_id: "nodo3_frecuencia"
        unit_of_measurement: "Hz"
                                                          value_template:       "{{
states('input_text.serial_history').split('F')[1].split('X')[0] | float }}"

      powerfactor3:
```

```
34.         friendly_name: "Nodo3 Factor de Potencia"
35.         unique_id: "nodo3_factordepotencia"
36.                                                                 value_template:       "{{
states('input_text.serial_history').split('X')[1].split('L')[0] | float }}"
37.
38.       luz_3:
39.         friendly_name: "Nodo3 Luz"
40.         unique_id: "nodo3_luz"
41.                                                                 value_template:       "{{
states('input_text.serial_history').split('L')[1].split('Z')[0] | float }}"
42.
43.       current3_calculated:
44.         friendly_name: "Nodo3 Corriente  Calculada"
45.         unique_id: "nodo3_corriente_calculada"
46.         unit_of_measurement: "A"
47.                                                 value_template:     "{{     0.0     if
(((states('sensor.power3')|float)*(states('sensor.powerfactor3')|float))|float) == 0.0 else
(((states('sensor.power3')|float)         /         ((states('sensor.powerfactor3')|float)
*(states('sensor.voltage3')|float))) |round(3)) }}"
48.       powerapp3_calculated:
49.         friendly_name: "Nodo3 Potencia Aparente"
50.         unique_id: "nodo3_potencia_aparente"
51.         unit_of_measurement: "VA"
52.                     value_template:  "{{  0.0  if  (((states('sensor.power3')|float)  *
(states('sensor.powerfactor3')|float))|float) == 0.0 else (( (states('sensor.power3')|float)
/ ( (states('sensor.powerfactor3')|float) )) | round(2)) }}"
53.       powerreact3_calculated:
54.         friendly_name: "Nodo3 Potencia Reactiva"
55.         unique_id: "nodo3_potencia_reactiva"
56.         unit_of_measurement: "VAr"
57.                                                 value_template:     "{{     0.0     if
(((states('sensor.power3')|float)*(states('sensor.powerfactor3')|float))|float)  ==0.0  or
(states('sensor.powerfactor3')|float)               ==1               else              (
((states('sensor.voltage3')|float)*((states('sensor.current3_calculated')|float))*((sin((a
cos((states('sensor.powerfactor3'))|round(3)))))))) | round(2))}}"
58.
```

Hay que mantener los SIMs registrados y con saldo para poder enviar las mediciones y las órdenes para el disparo de los actuadores.

Procedimiento en Arduino IDE

El código en Arduino IDE se basa el aprovechar los comandos AT del módulo SIM800L para el recibo y envío de los SMS, por lo cual, es el siguiente:

```
 1. #include <PZEM004Tv30.h>
 2.
 3. // Si no se han definido los pines RX y TX para el PZEM, se establecen valores
predeterminados
 4. #if !defined(PZEM_RX_PIN) && !defined(PZEM_TX_PIN)
 5. #define PZEM_RX_PIN 4
 6. #define PZEM_TX_PIN 2
 7. #endif
 8.
 9. // Si no se ha definido el serial para el PZEM, se usa Serial1 como predeterminado
10. #if !defined(PZEM_SERIAL)
11. #define PZEM_SERIAL Serial1
12. #endif
13.
14. // Configuración específica para el ESP32
15. #if defined(ESP32)
16. PZEM004Tv30 pzem(PZEM_SERIAL, PZEM_RX_PIN, PZEM_TX_PIN); // Instancia del PZEM usando
Serial1 y los pines definidos
```

```
#elif defined(ESP8266)
// Si se usa un ESP8266, la configuración del PZEM se simplifica ya que se puede conectar directamente
#else
 PZEM004Tv30 pzem(PZEM_SERIAL); // Instancia del PZEM para otros microcontroladores usando Serial1
#endif

// Configuración de la comunicación con el SIM800L usando UART2
HardwareSerial sim800(2);
#define BAUD_RATE 9600 // Velocidad de baudios para la comunicación con el SIM800L

// Declaración de pines
#define rxPin 16
#define txPin 17

int pinLuz = 15; // Pin para el sensor de luz
int pinRele = 13; // Pin para el relé
int pinLed = 26; // Pin para el LED

// Declaración de variables para almacenar mediciones del PZEM
float voltaje = 0.0;
float corriente = 0.0;
float potencia = 0.0;
float energia = 0.0;
float frecuencia = 0.0;
float factorPotencia = 0.0;
float energiaAnterior = 0; // Para almacenar la última medición de energía y detectar cambios

const String TELEFONO = "+593939236138"; // Número de teléfono para enviar SMS
unsigned long millisPrevios = 0; // Para controlar el envío de datos
const long intervalo = 30000; // Intervalo de 30 segundos para el envío de datos

// Variables para almacenar la última medición válida de cada parámetro
float ultimoVoltajeValido = 0.0;
float ultimaCorrienteValida = 0.0;
float ultimaPotenciaValida = 0.0;
float ultimaEnergiaValida = 0.0;
float ultimaFrecuenciaValida = 0.0;
float ultimoFpValido = 0.0;

unsigned long millisPrevios2 = 0;
const long intervalo2 = 5000; // Intervalo de 5 segundos para comprobar mensajes SMS

// Configuración inicial
void setup() {
  pinMode(pinRele, OUTPUT);
  pinMode(pinLed, OUTPUT);
  pinMode(pinLuz, INPUT);

  digitalWrite(pinRele, HIGH); // Inicialmente el relé está desactivado

  Serial.begin(115200); // Inicia la comunicación Serial con la PC

  // Inicia la comunicación con el SIM800L
  sim800.begin(BAUD_RATE, SERIAL_8N1, rxPin, txPin);

  // Intenta conectarse con el SIM800L hasta que responda
  while (!esperarRespuestaSIM800L()) {
    Serial.println("No se pudo conectar con el SIM800L, reintentando...");
    delay(5000); // Espera 5 segundos antes de reintentar
  }

  Serial.println("SIM800L listo y respondiendo.");

  // Configura el SIM800L para el envío de SMS en modo texto y borra todos los mensajes
  sim800.print("AT+CMGF=1\r");
  delay(2000);
  sim800.print("AT+CMGDA=\"DEL ALL\"\r");
```

```
  delay(2000);
}

// Bucle principal
void loop() {
  unsigned long millisActual = millis(); // Obtiene el tiempo actual

  // Lee los valores del PZEM y almacena la última medición válida
  voltaje = pzem.voltage();
  corriente = pzem.current();
  potencia = pzem.power();
  energia = pzem.energy();
  frecuencia = pzem.frequency();
  pf = pzem.pf();

  // Prepara los datos para enviar por SMS
 String datos = "V" + String(ultimoVoltajeValido) + "C" + String(ultimaCorrienteValida, 3) + "P" + String(ultimaPotenciaValida, 2) + "E" + String(ultimaEnergiaValida, 3) + "F" + String(ultimaFrecuenciaValida)    +    "X"    +    String(ultimoFpValido)    +    "L"    + String((analogRead(pinLuz))) + "Z";

  // Envia los datos por SMS cada intervalo establecido
  if (millisActual - millisPrevios >= intervalo) {
    Reply(datos); // Envia los datos como SMS
  }

  // Comprueba si hay mensajes SMS no leídos cada intervalo2 y actúa en consecuencia
  if (millisActual - millisPrevios2 >= intervalo2) {
    envioDatos(); // Comprueba y actúa sobre mensajes SMS no leídos
  }

  // Enciende el LED si la energía ha aumentado desde la última medición
  if (energia > energiaAnterior) {
    digitalWrite(pinLed, HIGH);
    delay(50);
    digitalWrite(pinLed, LOW);
  }

  energiaAnterior = energia;
}

// Espera una respuesta "OK" del SIM800L
bool esperarRespuestaSIM800L() {
  sim800.println("AT"); // Envia el comando AT al SIM800L
  long tiempoInicio = millis();
  while (millis() - tiempoInicio < 2000) { // Espera hasta 2 segundos por una respuesta
    if (sim800.available()) {
      String respuesta = sim800.readString();
      if (respuesta.indexOf("OK") >= 0) {
        return true; // El SIM800L respondió con "OK"
      }
    }
  }
  return false; // No se recibió respuesta o la respuesta no fue "OK"
}

// Lee el texto entre marcadores específicos
String readTextBetweenMarkers(char endMarker) {
  String text = "";
  char c;
  while (sim800.available() && (c = sim800.read()) != endMarker) {
    text += c;
  }
  return text;
}

// Controla el relé basado en el texto recibido
void controlRelay(String text) {
  int estadoRelay = text.toInt();
  if (estadoRelay == 1) {
```

```
151.     digitalWrite(rele, HIGH);
152.     Serial.println("Relé encendido");
153.   } else if (estadoRelay == 2) {
154.     digitalWrite(rele, LOW);
155.     Serial.println("Relé apagado");
156.   } else if (estadoRelay == 3) {
157.     pzem.resetEnergy();
158.     Serial.println("Energía reiniciada");
159.   } else {
160.     Serial.println("Texto no reconocido");
161.   }
162.   sim800.print("AT+CMGDA=\"DEL ALL\"\r"); // Borra todos los SMS después de actuar
163.   delay(1000);
164. }
165.
166. // Envía una respuesta por SMS
167. void Reply(String text) {
168.   sim800.print("AT+CMGF=1\r"); // Configura el modo de texto para SMS
169.   delay(1000);
170.   sim800.print("AT+CMGS=\"" + PHONE + "\"\r"); // Configura el número de destino
171.   delay(1000);
172.   sim800.print(text); // Escribe el texto del SMS
173.   delay(100);
174.   sim800.write(0x1A); // Envía el SMS
175.   delay(1000);
176.   Serial.println("SMS enviado correctamente.");
177.   millisPrevios = currentMillis; // Actualiza el tiempo para el próximo envío
178. }
179.
180. // Comprueba mensajes SMS no leídos y actúa según el contenido
181. void envioDatos() {
182.   sim800.print("AT+CMGL=\"REC UNREAD\"\r"); // Comando para listar SMS no leídos
183.   delay(2000);
184.   while (sim800.available()) {
185.     char c = sim800.read();
186.     Serial.write(c);
187.     if (c == '@') { // '@' es el marcador de inicio del contenido relevante
188.       String textoExtraido = readTextBetweenMarkers('#'); // '#' es el marcador de fin
189.       Serial.println("Texto extraído: " + textoExtraido);
190.       controlRelay(textoExtraido); // Actúa basado en el texto extraído
191.     }
192.   }
193.   millisPrevios2 = currentMillis2; // Actualiza el tiempo para la próxima comprobación
194. }
195.
```

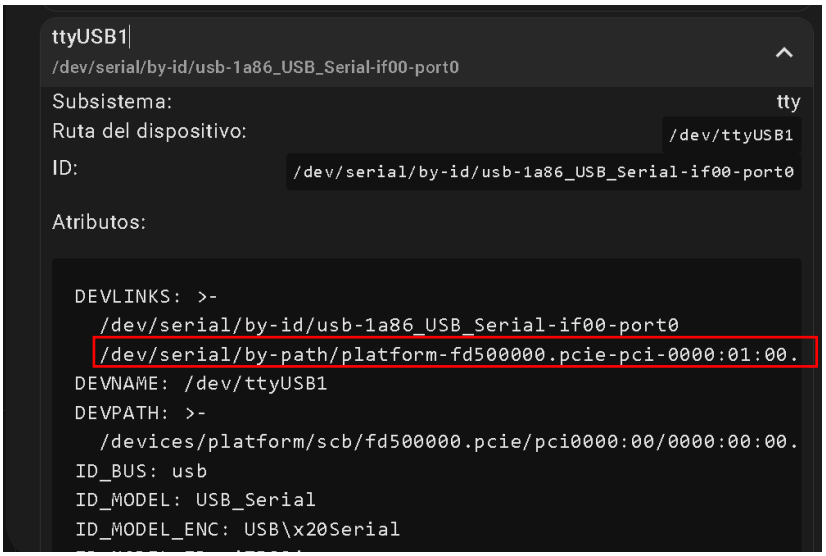
Al usarse este código, hay que procurar cambiar la variable TELEFONO con el número del dispositivo al cual queremos enviar las mediciones.

ETAPA5: Instalación del Nodo 4 nRF24

En este nodo no utilizamos ningún complemento como dispositivo de Home Assistant, ya que usamos comunicación serial, por lo cual debemos realizar algunas configuraciones manuales.

Procedimiento en Home Assistant

Puesto que aquí usamos un Arduino Nano como Gateway, necesitamos conectar le dispositivo a uno de os puertos USB e identificar el ID. Como lo hicimos anteriormente, el que tomaremos será el segundo DEVLINK que se encuentra en Ajustes > Sistema > Hardware > Todo el Hardware.



Luego de copiar el ID, nos dirigimos a configuration.yaml y asignamos el puerto USB como sensor, para esto, pegamos el siguiente fragmento de código.

```
1. sensor:
2.   - platform: serial
3.     name: Nodo4nRF24
4.          serial_port:  /dev/serial/by-path/platform-fd500000.pcie-pci-0000:01:00.0-usb-
0:1.1.1.3:1.0-port0
5.     baudrate: 115200
6.
```

Con el sensor serial creado, podemos crear los sensores plantilla en base a esta entidad, para eso, escribimos el siguiente código en el mismo documento.

```
 1.   - platform: template
 2.     sensors:
 3.       voltage4:
 4.         friendly_name: "Nodo4 Voltaje"
 5.         unique_id: "nodo4_voltaje"
 6.         unit_of_measurement: "V"
 7.           value_template: "{{ states('sensor.nodo4nrf24').split('V')[1].split('C')[0] |
float }}"
 8.       current4:
 9.         friendly_name: "Nodo4 Corriente"
10.         unique_id: "nodo4_corriente"
11.         unit_of_measurement: "A"
12.           value_template: "{{ states('sensor.nodo4nrf24').split('C')[1].split('P')[0] |
float }}"
13.       power4:
14.         friendly_name: "Nodo4 Potencia"
```

```
15.         unique_id: "nodo4_potencia"
16.         unit_of_measurement: "W"
17.          value_template: "{{ states('sensor.nodo4nrf24').split('PP')[1].split('E')[0] |
float }}"
18.       energy4:
19.         friendly_name: "Nodo4 Energia"
20.         unique_id: "nodo4_energia"
21.         unit_of_measurement: "kWh"
22.          value_template: "{{ states('sensor.nodo4nrf24').split('E')[1].split('FF')[0] |
float }}"
23.       frequency4:
24.         friendly_name: "Nodo4 Frecuencia"
25.         unique_id: "nodo4_frecuencia"
26.         unit_of_measurement: "Hz"
27.          value_template: "{{ states('sensor.nodo4nrf24').split('FF')[1].split('X')[0] |
float }}"
28.       powerfactor4:
29.         friendly_name: "Nodo4 Factor de Potencia"
30.         unique_id: "nodo4_factordepotencia"
31.          value_template: "{{ states('sensor.nodo4nrf24').split('X')[1].split('M')[0] |
float }}"
32.
33.       current4_calculated:
34.         friendly_name: "Nodo4 Corriente  Calculada"
35.         unique_id: "nodo4_corriente_calculada"
36.         unit_of_measurement: "A"
37.                                                 value_template:     "{{     0.0     if
(((states('sensor.power4')|float)*(states('sensor.powerfactor4')|float))|float) == 0.0 else
(((states('sensor.power4')|float)         /         ((states('sensor.powerfactor4')|float)
*(states('sensor.voltage4')|float))) |round(3)) }}"
38.       powerapp4_calculated:
39.         friendly_name: "Nodo4 Potencia Aparente"
40.         unique_id: "nodo4_potencia_aparente"
41.         unit_of_measurement: "VA"
42.                     value_template:  "{{  0.0  if  (((states('sensor.power4')|float)  *
(states('sensor.powerfactor4')|float))|float) == 0.0 else (( (states('sensor.power4')|float)
/ ( (states('sensor.powerfactor4')|float) )) | round(2)) }}"
43.       powerreact4_calculated:
44.         friendly_name: "Nodo4 Potencia Reactiva"
45.         unique_id: "nodo4_potencia_reactiva"
46.         unit_of_measurement: "VAr"
47.                                                 value_template:     "{{     0.0     if
(((states('sensor.power4')|float)*(states('sensor.powerfactor4')|float))|float)  ==0.0  or
(states('sensor.powerfactor4')|float)               ==1               else              (
((states('sensor.voltage4')|float)*((states('sensor.current4_calculated')|float))*((sin((a
cos((states('sensor.powerfactor4'))|round(3)))))) | round(2))}}"
48.
```

Luego, necesitamos crear los comandos del terminal junto a los actuadores.

```
 1.    nodo4_rele_off: /bin/bash -c "echo -ne '0' > /dev/serial/by-id/usb-1a86_USB_Serial-
if00-port0"
 2.    nodo4_rele_on: /bin/bash -c "echo -ne '1' > /dev/serial/by-id/usb-1a86_USB_Serial-
if00-port0"
 3.
 4.   nodo4_resetenergia: /bin/bash -c "echo -ne '2' > /dev/serial/by-id/usb-1a86_USB_Serial-
if00-port0"
 5.
 6.   nodo4_buzzer_off: /bin/bash -c "echo -ne '4' > /dev/serial/by-id/usb-1a86_USB_Serial-
if00-port0"
 7.    nodo4_buzzer_on: /bin/bash -c "echo -ne '3' > /dev/serial/by-id/usb-1a86_USB_Serial-
if00-port0"
 8.
```

```
      nodo4_rele:
        turn_on:
          service: shell_command.nodo4_rele_on
        turn_off:
          service: shell_command.nodo4_rele_off
      nodo4_resetenergia:
        turn_on:
          service: shell_command.nodo4_resetenergia
        turn_off:
          service: shell_command.nodo4_resetenergia
      nodo4_buzzer:
        turn_on:
          service: shell_command.nodo4_buzzer_on
        turn_off:
          service: shell_command.nodo4_buzzer_off

```

En este caso, tenemos un actuador adicional además del relé, que corresponde al buzzer. Simplemente añadimos más entidades según las necesidades del nodo.

Procedimiento en Arduino IDE

De la misma manera, aquí usamos dos microcontroladores para la comunicación, donde el segundo funciona como Gateway, que en este caso es un Arduino Nano.

En el ESP32, subiremos este código:

```
#include <SPI.h>
#include "RF24.h"
#include <PZEM004Tv30.h>

RF24 radio(4, 5); // Pines CE y CSN
const byte direccion[6] = "00001"; // Dirección del canal de comunicación
char mensajeRecibido[64]; // Buffer para almacenar el mensaje recibido

// Instancia del sensor PZEM-004T
PZEM004Tv30 pzem(Serial2, 16, 17);

// Definición de pines
int rele = 13;
int led = 26;
int buzzer = 15;

// Variables para almacenar las mediciones del PZEM-004T
float voltaje = 0;
float corriente = 0;
float potencia = 0;
float energia = 0;
float frecuencia = 0;
float factorPotencia = 0;

 float energiaAnterior = 0; // Almacena la última medición de energía para detectar cambios

// Variables para control de tiempo
unsigned long tiempoActual = 0;
```

```
unsigned long tiempoAnterior = 0;
const long intervalo = 1000; // Intervalo de tiempo para acciones repetitivas

int paquete = 1; // Variable para alternar entre los paquetes de datos a enviar

void setup() {
    Serial.begin(115200);
    while (!Serial) {
       // Espera a que el puerto serial esté disponible (necesario para algunas placas)
    }

    // Configura los pines
    pinMode(rele, OUTPUT);
    pinMode(led, OUTPUT);
    pinMode(buzzer, OUTPUT);
    digitalWrite(rele, HIGH); // Inicia con el relé desactivado
    digitalWrite(buzzer, LOW); // Inicia con el buzzer desactivado

    // Inicializa el módulo RF24
    if (!radio.begin()) {
        Serial.println(F("El hardware del radio no responde!!"));
        while (1); // Bucle infinito si el radio no responde
    }

    // Configura el radio
    radio.openWritingPipe(direccion);
    radio.openReadingPipe(1, direccion);
    radio.setPALevel(RF24_PA_LOW);
    radio.startListening(); // Comienza a escuchar mensajes entrantes
}

void loop() {
    tiempoActual = millis(); // Obtiene el tiempo actual

    // Lee los valores del sensor PZEM
    voltaje = pzem.voltage();
    corriente = pzem.current();
    potencia = pzem.power();
    energia = pzem.energy();
    frecuencia = pzem.frequency();
    factorPotencia = pzem.pf();

    // Prepara los textos a enviar con los datos medidos
    String texto1 = "V" + String(voltaje) + "C" + String(corriente, 3) + "P";
    String texto2 = "P" + String(potencia, 2) + "E" + String(energia, 3) + "F";
    String texto3 = "F" + String(frecuencia) + "X" + String(factorPotencia) + "M";

    String textoN; // Texto final a enviar

    // Selecciona el texto a enviar basado en la variable paquete
    if (paquete == 1) {
        textoN = texto1;
    } else if (paquete == 2) {
        textoN = texto2;
    } else if (paquete == 3) {
        textoN = texto3;
    }

    // Envia los datos cada intervalo especificado
    if (tiempoActual - tiempoAnterior >= intervalo) {
        radio.stopListening(); // Detiene la escucha para poder enviar
        radio.write(&textoN, sizeof(textoN)); // Envía el texto seleccionado
        radio.startListening(); // Reanuda la escucha

        paquete += 1; // Cambia al siguiente paquete
        tiempoAnterior = tiempoActual; // Actualiza el tiempo para el próximo envío
    }

    // Reinicia la secuencia de paquetes
    if (paquete == 4) {
```

```
        paquete = 1;
    }

    // Lee mensajes del puerto serial y los envía a través del RF24
    if (Serial.available()) {
        String texto = Serial.readString();
        radio.stopListening();
        radio.write(&texto, sizeof(texto));
        radio.startListening();
    }

    // Procesa los mensajes recibidos a través del RF24
    if (radio.available()) {
        memset(mensajeRecibido, 0, sizeof(mensajeRecibido)); // Limpia el buffer
        radio.read(&mensajeRecibido, sizeof(mensajeRecibido)); // Lee el mensaje

        Serial.println(mensajeRecibido); // Imprime el mensaje recibido

        // Ejecuta acciones basadas en el mensaje
        if (strcmp(mensajeRecibido, "1") == 0) {
            digitalWrite(rele, HIGH);
        } else if (strcmp(mensajeRecibido, "0") == 0) {
            digitalWrite(rele, LOW);
        } else if (strcmp(mensajeRecibido, "2") == 0) {
            pzem.resetEnergy();
        } else if (strcmp(mensajeRecibido, "3") == 0) {
            digitalWrite(buzzer, HIGH);
        } else if (strcmp(mensajeRecibido, "4") == 0) {
            digitalWrite(buzzer, LOW);
        }
    }

    // Acciones basadas en la comparación de la energía medida
    if (energia > energiaAnterior) {
        digitalWrite(led, HIGH); // Enciende el LED brevemente
        delay(50); // Espera 50 milisegundos
        digitalWrite(led, LOW); // Apaga el LED
    }

    energiaAnterior = energia; // Actualiza la última medición de energía
}

```

Y en el Arduino Nano, subimos los siguientes (hay que procurar cambiar la placa, el procesador y el puerto antes de darle a subir):

```
#include <SPI.h>
#include <RF24.h>
#include <ArduinoJson.h>

RF24 radio(9, 10); // Pines CE, CSN para el módulo RF24
const byte direccion[6] = "00001"; // Dirección del canal de comunicación
char mensajeRecibido[64]; // Buffer para almacenar el mensaje recibido

unsigned long ultimoTiempoMensajeValido = 0; // Variable para almacenar el tiempo del último mensaje válido

void setup() {
    Serial.begin(115200); // Inicia la comunicación serial
    radio.begin(); // Inicia el módulo RF24
     radio.openWritingPipe(direccion); // Abre un canal de escritura con la dirección especificada
    radio.openReadingPipe(1, direccion); // Abre un canal de lectura con la dirección especificada
```

```
    radio.setPALevel(RF24_PA_MAX); // Establece el nivel de potencia del módulo RF24 al máximo
    radio.startListening(); // Comienza a escuchar mensajes entrantes
}

void loop() {
     ultimoTiempoMensajeValido = millis(); // Actualiza el tiempo del último mensaje válido

     if (Serial.available()) { // Verifica si hay datos disponibles para leer en el puerto serial
       String texto = Serial.readString(); // Lee el texto enviado a través del puerto serial
        radio.stopListening(); // Detiene la escucha para poder enviar datos
       radio.write(&texto, sizeof(texto)); // Envia el texto leído a través del módulo RF24
        radio.startListening(); // Vuelve a comenzar la escucha de mensajes entrantes
    }

     if (radio.available()) { // Verifica si hay mensajes disponibles para leer en el módulo RF24
        memset(mensajeRecibido, 0, sizeof(mensajeRecibido)); // Limpia el buffer antes de leer un nuevo mensaje
      radio.read(&mensajeRecibido, sizeof(mensajeRecibido)); // Lee el mensaje entrante

        // Procesa el mensaje recibido para eliminar caracteres de salto de línea
        for (int i = 0; i < sizeof(mensajeRecibido); i++) {
            if (mensajeRecibido[i] == '\n' || mensajeRecibido[i] == '\r') {
               mensajeRecibido[i] = '\0'; // Reemplaza saltos de línea con terminadores de cadena
                break; // Sale del bucle al encontrar el primer salto de línea
            }
        }

         DynamicJsonDocument jsonBuffer(256); // Crea un buffer para el objeto JSON de tamaño 256
        auto root = jsonBuffer.to<JsonObject>(); // Crea el objeto JSON raíz

           // Procesamiento de los valores etiquetados en el mensaje recibido para asignarlos al objeto JSON
          // Utiliza la función strchr para buscar el inicio de cada etiqueta en el mensaje
        // Luego extrae el valor como subcadena y lo convierte a flotante
        // Finalmente, asigna cada valor al objeto JSON usando la etiqueta como clave

        // Ejemplo para la etiqueta 'V' (voltaje)
        char *inicioEtiqueta = strchr(mensajeRecibido, 'V');
        if (inicioEtiqueta) {
            char *finEtiqueta = strchr(inicioEtiqueta, 'C');
            if (finEtiqueta) {
              String valorStr = String(inicioEtiqueta + 1, finEtiqueta - inicioEtiqueta - 1);
                root["V"] = valorStr.toFloat();
            }
        }

/ Procesar etiqueta 'C'
        startPtr = strchr(receivedMessage, 'C');
        if (startPtr != NULL) {
            startIndex = startPtr - receivedMessage + 1; // +1 para empezar después de 'C'
            endPtr = strchr(startPtr, 'P');
            if (endPtr != NULL) {
               endIndex = endPtr - receivedMessage; // Busca el final del número antes de 'P'
                valueStr = String(receivedMessage).substring(startIndex, endIndex); // Extrae la subcadena
                value = valueStr.toFloat(); // Convierte la subcadena a float
                root["C"] = value;
            }
```

```
        }

        // Procesar etiqueta 'P'
        startPtr = strchr(receivedMessage, 'P');
        if (startPtr != NULL) {
            startIndex = startPtr - receivedMessage + 1; // +1 para empezar después de 'P'
            endPtr = strchr(startPtr, 'E');
            if (endPtr != NULL) {
                endIndex = endPtr - receivedMessage; // Busca el final del número antes de 'E'
                valueStr = String(receivedMessage).substring(startIndex, endIndex); // Extrae la subcadena
                value = valueStr.toFloat(); // Convierte la subcadena a float
                root["P"] = value;
            }
        }

        // Procesar etiqueta 'E'
        startPtr = strchr(receivedMessage, 'E');
        if (startPtr != NULL) {
            startIndex = startPtr - receivedMessage + 1; // +1 para empezar después de 'E'
            endPtr = strchr(startPtr, 'F');
            if (endPtr != NULL) {
                endIndex = endPtr - receivedMessage; // Busca el final del número antes de 'F'
                valueStr = String(receivedMessage).substring(startIndex, endIndex); // Extrae la subcadena
                value = valueStr.toFloat(); // Convierte la subcadena a float
                root["E"] = value;
            }
        }

        // Procesar etiqueta 'F'
        startPtr = strchr(receivedMessage, 'F');
        if (startPtr != NULL) {
            startIndex = startPtr - receivedMessage + 1; // +1 para empezar después de 'F'
            endPtr = strchr(startPtr, 'X');
            if (endPtr != NULL) {
                endIndex = endPtr - receivedMessage; // Busca el final del número antes de 'X'
                valueStr = String(receivedMessage).substring(startIndex, endIndex); // Extrae la subcadena
                value = valueStr.toFloat(); // Convierte la subcadena a float
                root["F"] = value;
            }
        }

        // Procesar etiqueta 'X'
        startPtr = strchr(receivedMessage, 'X');
        if (startPtr != NULL) {
            startIndex = startPtr - receivedMessage + 1; // +1 para empezar después de 'X'
            endPtr = strchr(startPtr, 'M');
            if (endPtr != NULL) {
                endIndex = endPtr - receivedMessage; // Busca el final del número antes de 'M'
                valueStr = String(receivedMessage).substring(startIndex, endIndex); // Extrae la subcadena
                value = valueStr.toFloat(); // Convierte la subcadena a float
                root["X"] = value;
            }

        // Serializa el objeto JSON y lo envía por el puerto serial
        serializeJson(root, Serial);
        Serial.println(); // Imprime una línea en blanco después del JSON para separar los mensajes
    }
```

```
128. }
129.
```

El código del Gateway recibe los datos y los procesa con el fin de que la ingración de serial de Home Assistant pueda leer la información.

ETAPA 6: Instalación del Nodo 5 LoRa

El procedimiento para la instalación de este nodo es idéntico al del Nodo 4, simplemente cambia el módulo transceptor. Para LoRa, usamos un RYLR998 en este caso.

Procedimiento en Home Assistant

De la misma manera, conectaremos el dispositivo que usaremos como Gateway y copiamos su segundo DEVLINK en la sección de Hardware de Home Assistant.

En configuration.yaml, establecemos la lectura del puerto ttyACM0 como sensor.

```
1.    - platform: serial
2.      name: Nodo5LoRa
3.            serial_port:  /dev/serial/by-path/platform-fd500000.pcie-pci-0000:01:00.0-usb-
0:1.1.2:1.0
```

Con este sensor definido, establecemos los sensores plantilla utilizando la misma estructura que las veces anteriores.

```
1.          - platform: template
2.      sensors:
```

```
      voltage5:
        friendly_name: "Nodo5 Voltaje"
        unique_id: "nodo5_voltaje"
        unit_of_measurement: "V"
         value_template: "{{ states('sensor.nodo5lora').split('VV')[1].split('CC')[0] | float }}"
      current5:
        friendly_name: "Nodo5 Corriente"
        unique_id: "nodo5_corriente"
        unit_of_measurement: "A"
         value_template: "{{ states('sensor.nodo5lora').split('CC')[1].split('PP')[0] | float }}"
      power5:
        friendly_name: "Nodo5 Potencia"
        unique_id: "nodo5_potencia"
        unit_of_measurement: "W"
         value_template: "{{ states('sensor.nodo5lora').split('PP')[1].split('EE')[0] | float }}"
      energy5:
        friendly_name: "Nodo5 Energia"
        unique_id: "nodo5_energia"
        unit_of_measurement: "kWh"
         value_template: "{{ states('sensor.nodo5lora').split('EE')[1].split('FF')[0] | float }}"
      frequency5:
        friendly_name: "Nodo5 Frecuencia"
        unique_id: "nodo5_frecuencia"
        unit_of_measurement: "Hz"
         value_template: "{{ states('sensor.nodo5lora').split('FF')[1].split('XX')[0] | float }}"
      powerfactor5:
        friendly_name: "Nodo5 Factor de Potencia"
        unique_id: "nodo5_factordepotencia"
         value_template: "{{ states('sensor.nodo5lora').split('XX')[1].split('MM')[0] | float }}"
      pir_5:
        friendly_name: "Nodo5 PIR"
        unique_id: "nodo5_PIR"
         value_template: "{{ states('sensor.nodo5lora').split('MM')[1].split('ZZ')[0] | float }}"

      current5_calculated:
        friendly_name: "Nodo5 Corriente  Calculada"
        unique_id: "nodo5_corriente_calculada"
        unit_of_measurement: "A"
                                              value_template:     "{{     0.0     if (((states('sensor.power5')|float)*(states('sensor.powerfactor5')|float))|float) == 0.0 else (((states('sensor.power5')|float)         /         ((states('sensor.powerfactor5')|float)*(states('sensor.voltage5')|float))) |round(3)) }}"
      powerapp5_calculated:
        friendly_name: "Nodo5 Potencia Aparente"
        unique_id: "nodo5_potencia_aparente"
        unit_of_measurement: "VA"
                    value_template:  "{{  0.0  if  (((states('sensor.power5')|float)  *(states('sensor.powerfactor5')|float))|float) == 0.0 else (( (states('sensor.power5')|float) / ( (states('sensor.powerfactor5')|float) )) | round(2)) }}"
      powerreact5_calculated:
        friendly_name: "Nodo5 Potencia Reactiva"
        unique_id: "nodo5_potencia_reactiva"
        unit_of_measurement: "VAr"
        value_template:
                                                                "{{      0.0      if (((states('sensor.power5')|float)*(states('sensor.powerfactor5')|float))|float)  ==0.0  or (states('sensor.powerfactor5')|float)               ==1               else               ( ((states('sensor.voltage5')|float)*((states('sensor.current5_calculated')|float))*((sin((acos((states('sensor.powerfactor5'))|round(3)))))))) | round(2))}}"

```

Y también lo hacemos con los comandos Shell y los interruptores.

```
shell_command:
 nodo5_rele_off: /bin/bash -c "echo -ne 'AT+SEND=0,7,JARELEK\r\n' > /dev/serial/by-path/platform-fd500000.pcie-pci-0000:01:00.0-usb-0:1.1.2:1.0"
   nodo5_rele_on: /bin/bash -c "echo -ne 'AT+SEND=0,7,JERELEK\r\n' > /dev/serial/by-path/platform-fd500000.pcie-pci-0000:01:00.0-usb-0:1.1.2:1.0"

  nodo5_resetenergia: /bin/bash -c "echo -ne 'AT+SEND=0,7,JENERG\r\n' > /dev/serial/by-path/platform-fd500000.pcie-pci-0000:01:00.0-usb-0:1.1.2:1.0"

switch:
  - platform: template
    switches:
      nodo5_rele:
        turn_on:
          service: shell_command.nodo5_rele_on
        turn_off:
          service: shell_command.nodo5_rele_off
      nodo5_resetenergia:
        turn_on:
          service: shell_command.nodo5_resetenergia
        turn_off:
          service: shell_command.nodo5_resetenergia

```

Con esto, ya podemos proceder a la programación del Nodo en esp32.

Procedimiento en Arduino IDE

En el código del Nodo, utilizaremos el siguiente código, el cual utiliza únicamente comunicación serial y comandos AT para la comunicación con el RYLR998.

```
#include <PZEM004Tv30.h>

// Verifica si los pines RX y TX del PZEM están definidos, si no, los define más adelante
#if !defined(PZEM_RX_PIN) && !defined(PZEM_TX_PIN)
#endif

// Verifica si el serial para comunicarse con el PZEM está definido, si no, lo define
#if !defined(PZEM_SERIAL)
#define PZEM_SERIAL Serial1 // Utiliza Serial1 como predeterminado para la comunicación
#endif

// Configuración específica para ESP32
#if defined(ESP32)
#define PZEM_RX_PIN 4 // Define el pin RX para la comunicación con el PZEM
#define PZEM_TX_PIN 2 // Define el pin TX para la comunicación con el PZEM
PZEM004Tv30 pzem(PZEM_SERIAL, PZEM_RX_PIN, PZEM_TX_PIN); // Inicializa el PZEM con los pines RX y TX
#elif defined(ESP8266)
PZEM004Tv30 pzem(PZEM_SERIAL); // Para ESP8266, no es necesario especificar los pines
#endif
```

```
20.
21. #define RXp2 16 // Define el pin RX para Serial2
22. #define TXp2 17 // Define el pin TX para Serial2
23.
24. String mensajeExtraido; // Almacena el mensaje extraído de la entrada serial
25.
26. int pinPIR = 15; // Define el pin para el sensor PIR
27. int estadoPIR = 0; // Almacena el estado del sensor PIR
28.
29. int pinRele = 13; // Define el pin para el relé
30. int pinLed = 26; // Define el pin para el LED
31.
32. // Variables para almacenar las mediciones del PZEM
33. float voltaje = 0.0;
34. float corriente = 0.0;
35. float potencia = 0.0;
36. float energia = 0.0;
37. float frecuencia = 0.0;
38. float factorPotencia = 0.0;
39.
40. float energiaAnterior = 0; // Almacena la última medición de energía
41.
42. unsigned long millisPrevios = 0; // Almacena el tiempo del último cambio de estado del
LED
43. const long intervalo = 1000; // Intervalo de tiempo para acciones repetitivas
44.
45. unsigned long millisPrevios2 = 0;
46. const unsigned long intervalo2 = 5000; // Intervalo de tiempo para verificar el estado
del sensor PIR
47.
48. void setup() {
49.   pinMode(pinRele, OUTPUT);
50.   pinMode(pinLed, OUTPUT);
51.   pinMode(pinPIR, INPUT);
52.
53.   digitalWrite(pinRele, HIGH); // Inicia con el relé desactivado
54.
55.   Serial.begin(115200); // Inicia la comunicación serial a 115200 baudios
56.    Serial2.begin(115200, SERIAL_8N1, RXp2, TXp2); // Inicia Serial2 con configuración
8N1
57.
58.    Serial2.print("AT+RESET\r\n"); // Envía el comando AT para resetear el dispositivo
conectado a Serial2
59.
60.   delay(1000); // Espera 1000 ms para que el comando se ejecute
61. }
62.
63. void loop() {
64.   unsigned long millisActual = millis(); // Obtiene el tiempo actual
65.   // Lee los valores del sensor PZEM
66.   voltaje = pzem.voltage();
67.   corriente = pzem.current();
68.   potencia = pzem.power();
69.   energia = pzem.energy();
70.   frecuencia = pzem.frequency();
71.   factorPotencia = pzem.pf();
72.
73.   // Verifica el estado del sensor PIR
74.   if (digitalRead(pinPIR) == HIGH && estadoPIR == LOW) {
75.     estadoPIR = HIGH; // Cambia el estado a HIGH
76.     millisPrevios2 = millis(); // Guarda el momento en que se detectó movimiento
77.   }
78.
79.   // Verifica si han pasado 5 segundos desde la última detección de movimiento
80.   if (estadoPIR == HIGH && millis() - millisPrevios2 >= intervalo2) {
81.     estadoPIR = LOW; // Cambia el estado a LOW después de 5 segundos sin detección
82.   }
83.
84.   // Procesa los datos recibidos por Serial2
85.   processSerialData();
```

```
 86.
 87.   // Envía los datos cada 1 segundo (1000 ms)
 88.   if (millisActual - millisPrevios >= intervalo) {
 89.     // Prepara los datos a enviar
 90.       String datos = "VV" + String(voltaje) + "CC" + String(corriente, 3) + "PP" +
String(potencia, 2) + "EE" + String(energia, 3) + "FF" + String(frecuencia) + "XX" +
String(factorPotencia) + "MM" + String(estadoPIR) + "ZZ";
 91.     int direccion = 0; // Dirección ficticia, reemplazar según sea necesario
 92.     int longitud = datos.length(); // Obtiene la longitud de los datos a enviar
 93.     String comandoAT = "AT+SEND=" + String(direccion) + "," + String(longitud) + "," +
datos + "\r\n";
 94.     Serial2.print(comandoAT); // Envía el comando AT con los datos a través de Serial2
 95.     millisPrevios = millisActual;
 96.   }
 97.
 98.   // Verifica si la energía medida ha aumentado
 99.   if (energia > energiaAnterior) {
100.     digitalWrite(pinLed, HIGH); // Enciende el LED
101.     delay(50); // Espera 50 milisegundos
102.     digitalWrite(pinLed, LOW); // Apaga el LED
103.   }
104.
105.   energiaAnterior = energia; // Actualiza la medición anterior de energía
106. }
107.
108. // Procesa los datos recibidos por Serial2 y ejecuta acciones basadas en el mensaje
109. void processSerialData() {
110.     if (Serial2.available()) {
111.           String datos = Serial2.readString(); // Lee todos los datos disponibles de
Serial2
112.         Serial.print(datos); // Escribe los datos en el Serial
113.
114.         // Busca el inicio y el final del mensaje extraído
115.         int inicioIndex = datos.indexOf('J') + 1;
116.         int finalIndex = datos.indexOf('K', inicioIndex);
117.         if (inicioIndex > 0 && finalIndex > inicioIndex) {
118.             String mensajeExtraido = datos.substring(inicioIndex, finalIndex);
119.
120.             // Ejecuta acciones basadas en el mensaje extraído
121.             if (mensajeExtraido == "ERELE") {
122.                 digitalWrite(pinRele, HIGH); // Enciende el relé
123.             } else if (mensajeExtraido == "ARELE") {
124.                 digitalWrite(pinRele, LOW); // Apaga el relé
125.             } else if (mensajeExtraido == "ENERG") {
126.                 pzem.resetEnergy(); // Resetea la medición de energía del PZEM
127.             }
128.         }
129.     }
130. }
131.
```

Para el código del Gateway, se necesitará transcribir estos datos y enviar los valores entrantes por serial hace la red LoRa. Para eso, se utiliza este sencillo código.

```
 1.   1. #include <HardwareSerial.h> // Incluimos la biblioteca para manejar los puertos
seriales de hardware
 2.   2.
 3.   3. HardwareSerial RYLR998(2); // Declaramos RYLR998 utilizando el UART2
 4.   4.
 5.   5. void setup() {
 6.    6.    Serial.begin(115200); // Inicializamos la comunicación Serial (USB) a 115200
baudios
 7.   7.   RYLR998.begin(115200); // Inicializamos RYLR998 (UART2) a 115200 baudios
 8.   8. }
 9.   9.
```

```
10. 10. void loop() {
11.  11.    if (RYLR998.available()) { // Verificamos si hay datos disponibles en UART2
(RYLR998)
12.  12.     Serial.write(RYLR998.read()); // Leemos un byte de UART2 (RYLR998) y lo enviamos
al Serial (USB)
13.  13.     Serial.println(); // Añadimos un salto de línea para mejorar la legibilidad
14.  14.   }
15.  15.
16.  16.    if (Serial.available()) { // Verificamos si hay datos disponibles en el Serial
(USB)
17.  17.     RYLR998.write(Serial.read()); // Leemos un byte del Serial (USB) y lo enviamos
a UART2 (RYLR998)
18.  18.   }
19.  19. }
20.  20.
21.
```

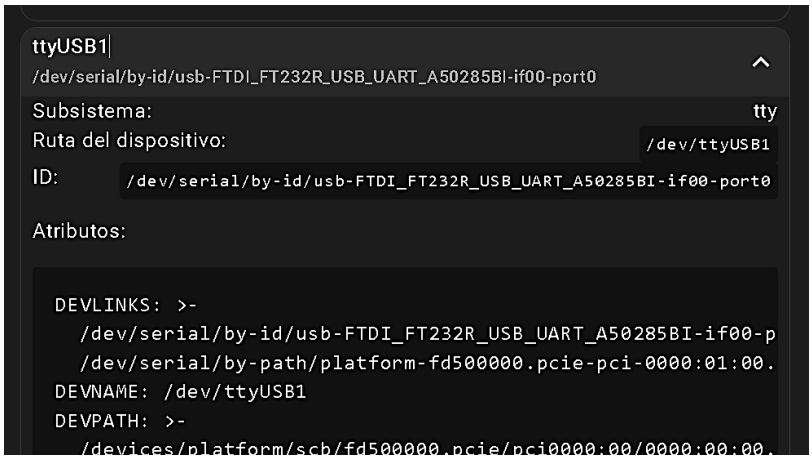
El código en este caso se acorta, ya que los datos que se recibe directamente con el serial del ESP32C6 junto al RYLR998 son lo suficientemente aptos para que Home Assistant pueda ller sin ningún inconveniente.

ETAPA 7: Instalación de Nodo 6 Zigbee

La instalación de Zigbee es aún más parecida al anterior, sin embargo es necesario del reacondicionamiento de datos del serial para Home Assistant.

Procedimiento en Home Assistant

Como en los casos anteriores, conectaremos el dispositivo USB y verificaremos y copiaremos el segundo DEVLINK.



Entraremos a configuration.yaml y definiremos el puerto serial como sensor para el Nodo6.

```
1.   - platform: serial
```

```
2.     name: Nodo6Zigbee
3.           serial_port:  /dev/serial/by-path/platform-fd500000.pcie-pci-0000:01:00.0-usb-
0:1.1.1.4:1.0-port0
4.     baudrate: 9600
5.
```

Luego, definiremos los sensores utilizando este sensor serial.

```
 1.   - platform: template
 2.     sensors:
 3.       voltage6:
 4.         friendly_name: "Nodo6 Voltaje"
 5.         unique_id: "nodo6_voltaje"
 6.         unit_of_measurement: "V"
 7.          value_template: "{{ states('sensor.nodo6zigbee').split('V')[1].split('C')[0] |
float }}"
 8.       current6:
 9.         friendly_name: "Nodo6 Corriente"
10.         unique_id: "nodo6_corriente"
11.         unit_of_measurement: "A"
12.          value_template: "{{ states('sensor.nodo6zigbee').split('C')[1].split('P')[0] |
float }}"
13.       power6:
14.         friendly_name: "Nodo6 Potencia"
15.         unique_id: "nodo6_potencia"
16.         unit_of_measurement: "W"
17.          value_template: "{{ states('sensor.nodo6zigbee').split('P')[1].split('E')[0] |
float }}"
18.       energy6:
19.         friendly_name: "Nodo6 Energia"
20.         unique_id: "nodo6_energia"
21.         unit_of_measurement: "kWh"
22.          value_template: "{{ states('sensor.nodo6zigbee').split('E')[1].split('F')[0] |
float }}"
23.       frequency6:
24.         friendly_name: "Nodo6 Frecuencia"
25.         unique_id: "nodo6_frecuencia"
26.         unit_of_measurement: "Hz"
27.          value_template: "{{ states('sensor.nodo6zigbee').split('F')[1].split('X')[0] |
float }}"
28.       powerfactor6:
29.         friendly_name: "Nodo6 Factor de Potencia"
30.         unique_id: "nodo6_factordepotencia"
31.          value_template: "{{ states('sensor.nodo6zigbee').split('X')[1].split('M')[0] |
float }}"
32.       mq5_6:
33.         friendly_name: "Nodo6 MQ5"
34.         unique_id: "nodo6_MQ5"
35.          value_template: "{{ states('sensor.nodo6zigbee').split('M')[1].split('Z')[0] |
float }}"
36.
37.       current6_calculated:
38.         friendly_name: "Nodo6 Corriente  Calculada"
39.         unique_id: "nodo6_corriente_calculada"
40.         unit_of_measurement: "A"
41.                                                 value_template:     "{{     0.0     if
(((states('sensor.power6')|float)*(states('sensor.powerfactor6')|float))|float) == 0.0 else
(((states('sensor.power6')|float)         /         ((states('sensor.powerfactor6')|float)
*(states('sensor.voltage6')|float))) |round(3)) }}"
42.       powerapp6_calculated:
43.         friendly_name: "Nodo6 Potencia Aparente"
44.         unique_id: "nodo6_potencia_aparente"
45.         unit_of_measurement: "VA"
46.                     value_template:  "{{  0.0  if  (((states('sensor.power6')|float)  *
(states('sensor.powerfactor6')|float))|float) == 0.0 else (( (states('sensor.power6')|float)
/ ( (states('sensor.powerfactor6')|float) )) | round(2)) }}"
47.       powerreact6_calculated:
```

```
        friendly_name: "Nodo6 Potencia Reactiva"
        unique_id: "nodo6_potencia_reactiva"
        unit_of_measurement: "VAr"
        value_template:
                                                                "{{      0.0      if
(((states('sensor.power6')|float)*(states('sensor.powerfactor6')|float))|float)  ==0.0  or
(states('sensor.powerfactor6')|float)               ==1             else              (
((states('sensor.voltage6')|float)*((states('sensor.current6_calculated')|float))*((sin((a
cos((states('sensor.powerfactor6'))|round(3)))))))) | round(2))}}"

```

También haremos lo mismo para los comandos del terminal, como los interruptores del relé y de reinicio de energía.

```
shell_command:
     nodo6_rele_off:  /bin/bash  -c  "echo  -ne  '0'  >  /dev/serial/by-path/platform-
fd500000.pcie-pci-0000:01:00.0-usb-0:1.1.1.4:1.0-port0"
  nodo6_rele_on: /bin/bash -c "echo -ne '1' > /dev/serial/by-path/platform-fd500000.pcie-
pci-0000:01:00.0-usb-0:1.1.1.4:1.0-port0"
    nodo6_resetenergia: /bin/bash -c "echo -ne '3' > /dev/serial/by-path/platform-
fd500000.pcie-pci-0000:01:00.0-usb-0:1.1.1.4:1.0-port0"
switch:
  - platform: template
    switches:
      nodo6_rele:
        turn_on:
          service: shell_command.nodo6_rele_on
        turn_off:
          service: shell_command.nodo6_rele_off
      nodo6_resetenergia:
        turn_on:
          service: shell_command.nodo6_resetenergia
        turn_off:
          service: shell_command.nodo6_resetenergia
```

Con esto, ya tendríamos configurado Home Assistant para que funcionen los dispositivos del Nodo 6.

Procedimiento en Arduino IDE

En este caso no hay Gateway, ya que el Xbee S2C imprime los datos a través del serial de una forma en la que Home Assistant los puede entender, por lo cual, solo usa un adaptador USB a TTL. Por lo mismo, el código que usa el Nodo es bastante corto.

```
#include <PZEM004Tv30.h> // Incluye la biblioteca para trabajar con el módulo PZEM-004T

// Define los pines RX y TX para la comunicación con el PZEM-004T si aún no están
definidos
#if !defined(PZEM_RX_PIN) && !defined(PZEM_TX_PIN)
#endif

// Usa Serial1 para la comunicación con el PZEM si PZEM_SERIAL aún no está definido
#if !defined(PZEM_SERIAL)
#define PZEM_SERIAL Serial1
#endif

// Configuración específica para ESP32
```

```
#if defined(ESP32)
#define PZEM_RX_PIN 4 // Pin RX para el PZEM
#define PZEM_TX_PIN 2 // Pin TX para el PZEM

String datos; // Variable para almacenar los datos recibidos por Serial2

PZEM004Tv30 pzem(PZEM_SERIAL, PZEM_RX_PIN, PZEM_TX_PIN); // Inicializa el PZEM-004T con Serial1
#elif defined(ESP8266)
PZEM004Tv30 pzem(PZEM_SERIAL); // Inicializa el PZEM-004T con Serial para ESP8266
#endif

#define RXp2 16 // Pin RX para Serial2
#define TXp2 17 // Pin TX para Serial2

int pinGas = 15; // Pin para el sensor de gas

const int pinRele = 13; // Pin para el relé

int pinLed = 12; // Pin para el LED

// Variables para almacenar las mediciones del PZEM-004T
float voltaje = 0.0;
float corriente = 0.0;
float potencia = 0.0;
float energia = 0.0;
float frecuencia = 0.0;
float factorPotencia = 0.0;

float energiaAnterior = 0; // Almacena la última medición de energía

unsigned long millisPrevios = 0; // Almacena el tiempo del último cambio de estado del LED
const long intervalo = 1000; // Intervalo de tiempo para el envío de datos

void setup() {
  pinMode(pinRele, OUTPUT); // Configura el pin del relé como salida
  pinMode(pinLed, OUTPUT); // Configura el pin del LED como salida
  pinMode(pinGas, INPUT); // Configura el pin del sensor de gas como entrada

  digitalWrite(pinRele, HIGH); // Inicia con el relé apagado

  Serial.begin(115200); // Inicia la comunicación Serial (USB) a 115200 baudios
  Serial2.begin(9600, SERIAL_8N1, RXp2, TXp2); // Inicia Serial2 a 9600 baudios
}

void loop() {
  unsigned long millisActual = millis(); // Obtiene el tiempo actual
  // Lee los valores del sensor PZEM
  voltaje = pzem.voltage();
  corriente = pzem.current();
  potencia = pzem.power();
  energia = pzem.energy();
  frecuencia = pzem.frequency();
  factorPotencia = pzem.pf();

  // Procesa los datos recibidos por Serial2
  if (Serial2.available()) {
    datos = Serial2.readString(); // Lee los datos como una cadena
    Serial.print(datos); // Escribe los datos en Serial
  }
  datos.trim(); // Elimina espacios en blanco al inicio y al final de la cadena

  // Ejecuta acciones basadas en los comandos recibidos
  if (datos == "1") {
    digitalWrite(pinRele, HIGH); // Enciende el relé
  } else if (datos == "0") {
    digitalWrite(pinRele, LOW); // Apaga el relé
  }

```

```
  if (datos == "3") {
    pzem.resetEnergy(); // Resetea la medición de energía del PZEM
  }

  // Envía los datos cada intervalo especificado
  if (millisActual - millisPrevios >= intervalo) {
      String datos2 = "V" + String(voltaje) + "C" + String(corriente, 3) + "P" +
String(potencia, 2) + "E" + String(energia, 3) + "F" + String(frecuencia) + "X" +
String(factorPotencia) + "M" + String((analogRead(pinGas))) + "Z";

    Serial2.println(datos2); // Envía los datos a través de Serial2
    millisPrevios = millisActual;
  }

  // Enciende el LED si la energía ha aumentado desde la última medición
  if (energia > energiaAnterior) {
    digitalWrite(pinLed, HIGH); // Enciende el LED
    delay(50); // Espera 50 milisegundos
    digitalWrite(pinLed, LOW); // Apaga el LED
  }

  energiaAnterior = energia; // Actualiza la última medición de energía
}
```

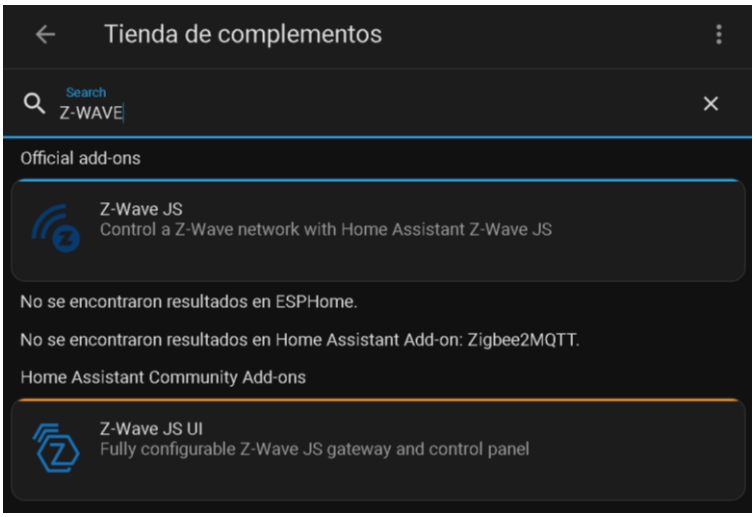
Con esto, ya tendríamos el nodo listo para que funcione en Home Assistant.

ETAPA 8: Instalación de Nodo 7 Z-Wave

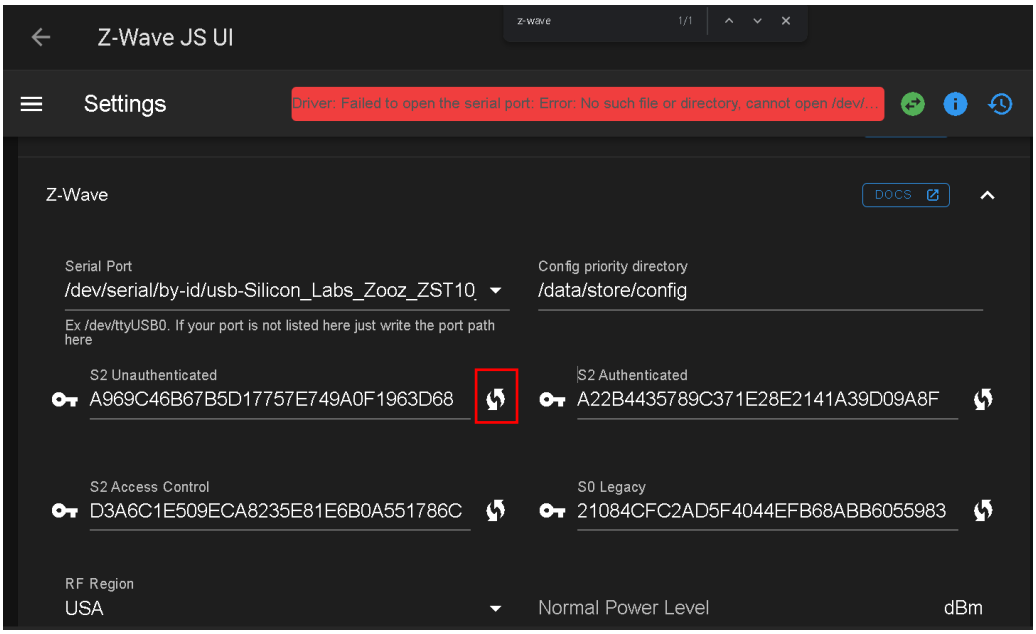
Para esta implementación usaremos una extensión para manejar correctamente el controlador Z-Wave.

Procedimiento en Home Assistant

Empezamos instalando la integración que necesitamos, para esto, nos dirigimos a Ajustes > Complementos > Tiendas de Complementos, y buscamos "Z-Wave JS UI" y lo instalamos.

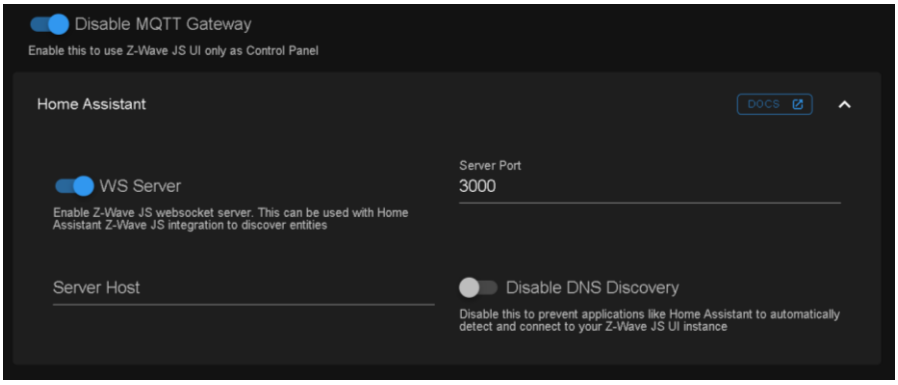


Con el complemento instalado, lo iniciamos. En el panel izquierdo, entramos a Settings > Z-Wave y configuramos todos los parámetros que necesitamos. Generalmente, solo basta con especificar el puerto serial del controlador Z-Wave (que ya debe estar conectado) y el resto de los datos deberían autocompletarse al hacer clic en el icono de recargar.



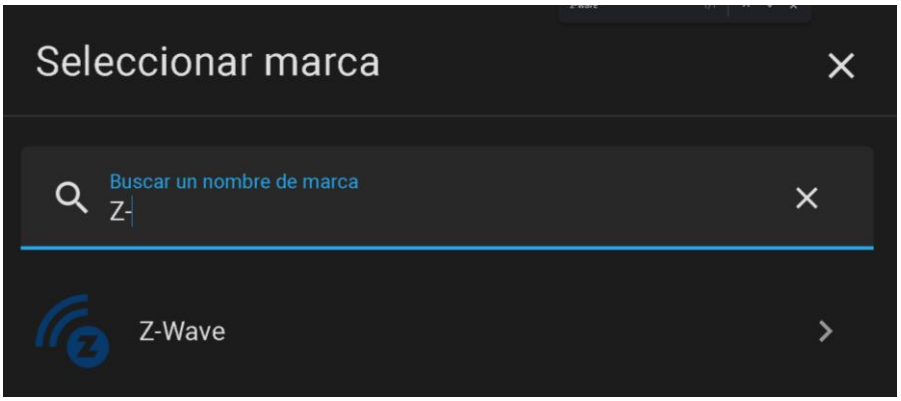
Asegúrate escoger la región de acuerdo con la frecuencia establecida tanto del controlador como del nodo o dispositivo Z-Wave final.

Más abajo, necesitamos configurar el servidor WS que servirá para que Home Assistant capte los sensores y actuadores y los establezca como entidades en Home Assistant.

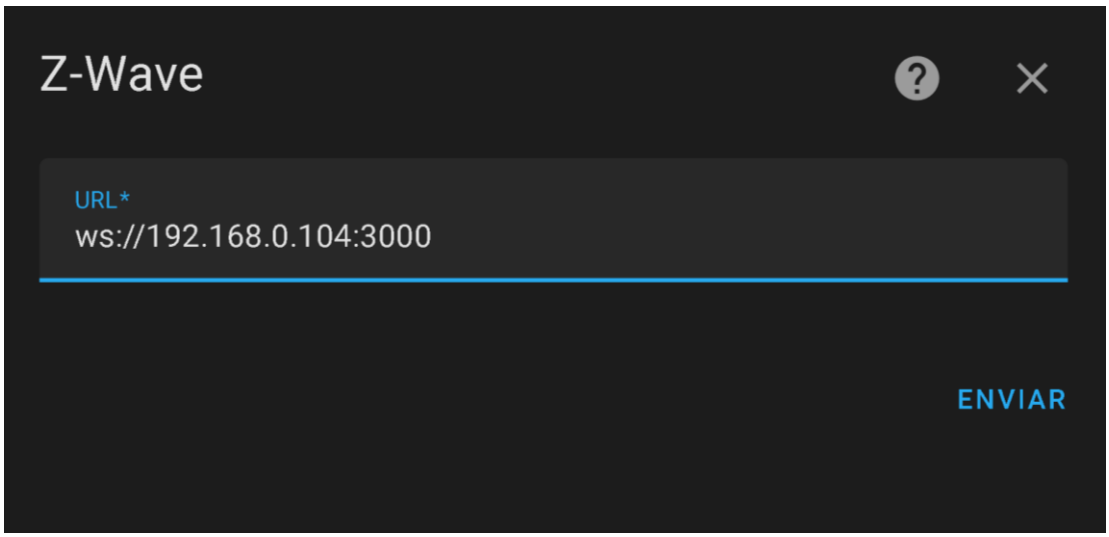


También es posible enviar estos datos al mismo servidor MQTT que hicimos en el Nodo 1, sin embargo, aquí se utilizó el complemento propio de Home Assitant de Z-Wave. Para esto, ingresamos a Ajustes > Dispositivos y

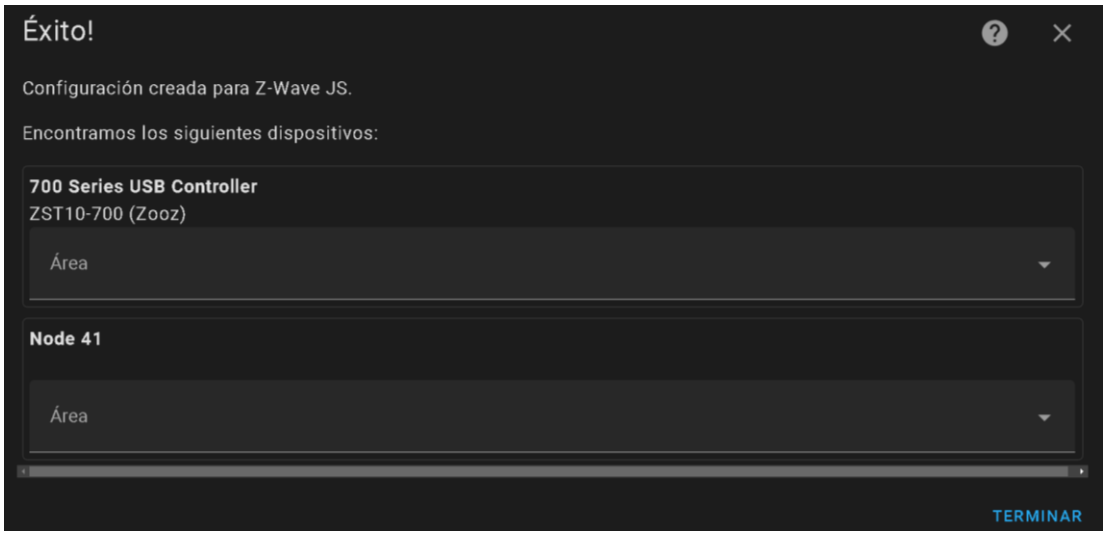
Servicios > Integraciones > Añadir Integración, y buscamos “Z-Wave”, luego hacemos clic en “Añadir un dispositivo Z-Wave”.



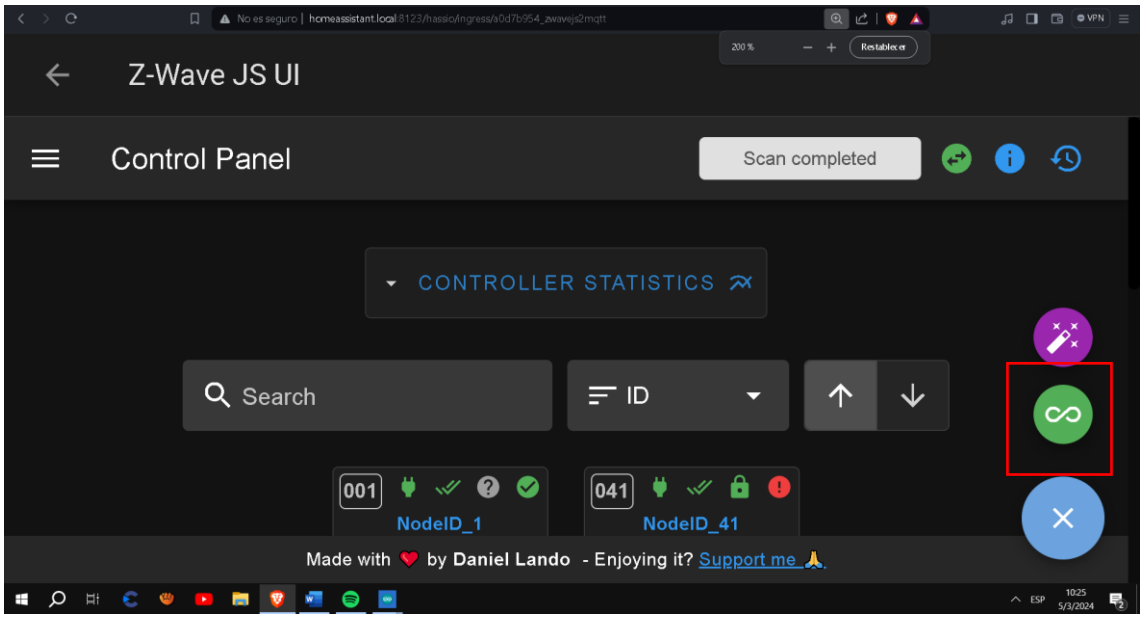
En la siguiente ventana, desactivamos el visto de “Utilizar el complemento del Supervisor Z-Wave JS”, y en la siguiente ventana, ingresamos la IP de nuestro Home Assistant, junto a el puerto de nuestro servidor WS de Z-Wave.



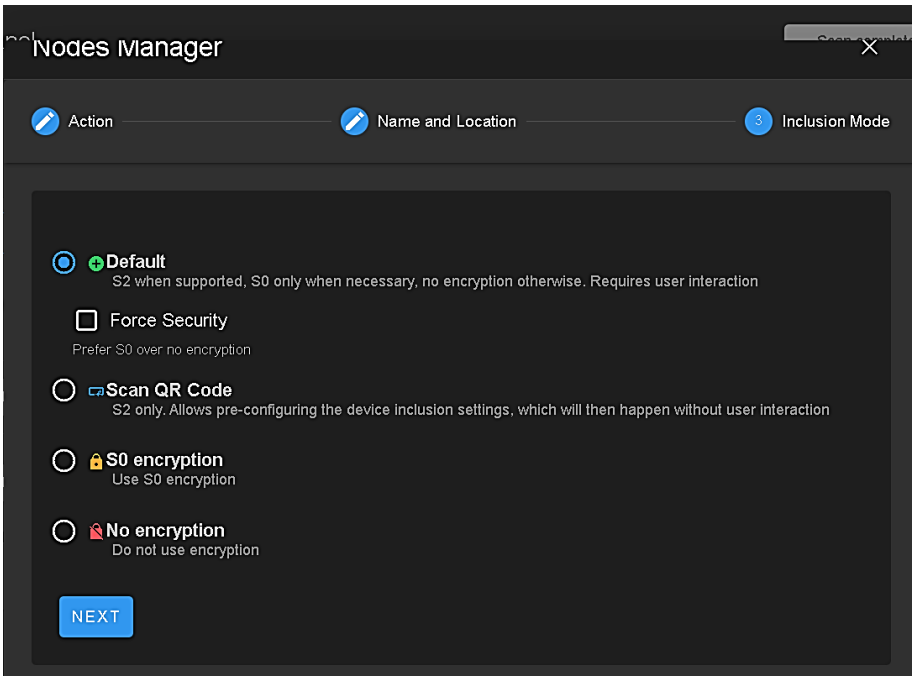
Finalmente, Home Assistant reconocerá tanto el controlador como los nodos.



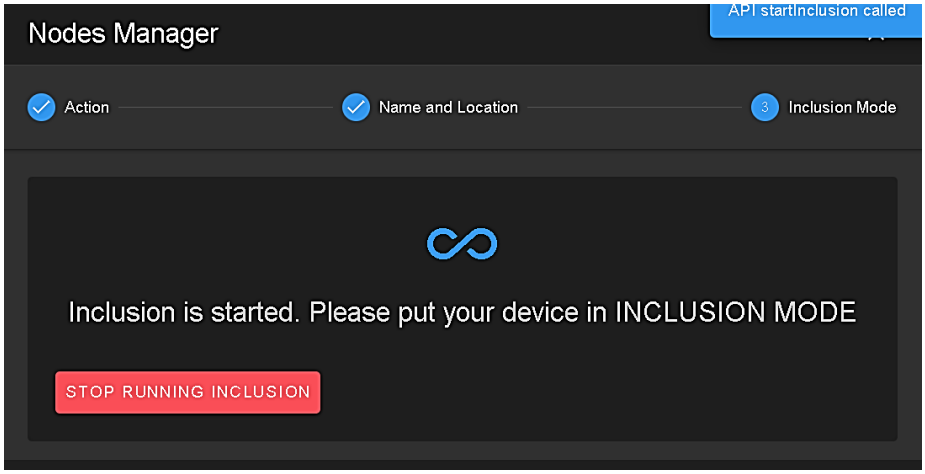
Para agregar un nodo a Z-Wave, ingresamos al complemento de Z-Wave JS UI y nos dirigimos abajo a la izquierda en el panel de control, hacemos clic ahí y después en el icono verde que dice "Manage nodes" para agregar nuestro nodo.



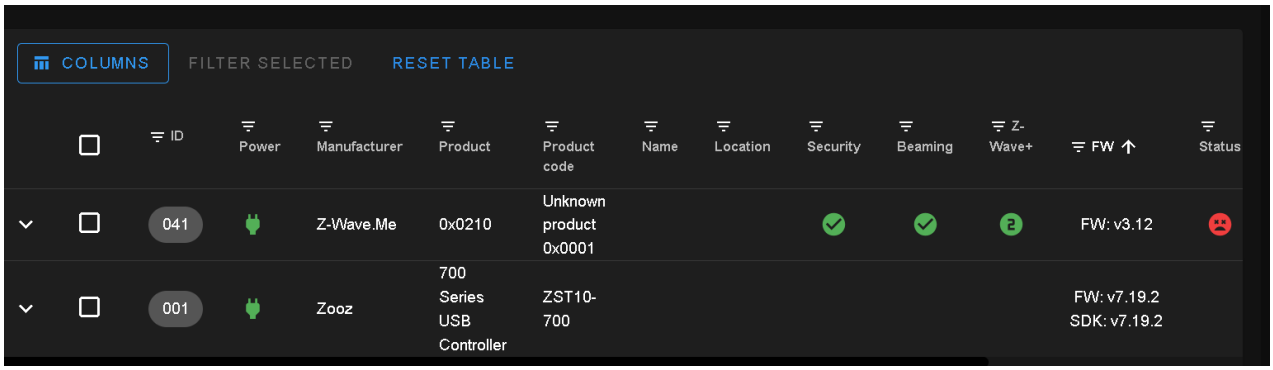
Hacemos clic en Inclusión, le asignamos un nombre a nuestro dispositivo junto a la localización (opcional), le damos a Next y en el modo de inclusión, seleccionamos "Default"



Cuando el controlador entre en modo emparejamiento, lo que tendremos que hacer para enlazar nuestro Z-UNO 2, es aplastar 3 veces el botón "BTN" rápidamente en el microcontrolador.



Luego de esto, el Nodo ya estará conectado en la red Z-Wave.



Con esta configuración realizada, cada dispositivo que conectemos a la red Z-Wave desde Z-Wave JS UI aparecerá como entidad den Home Assistant.

Procedimiento en Arduino IDE

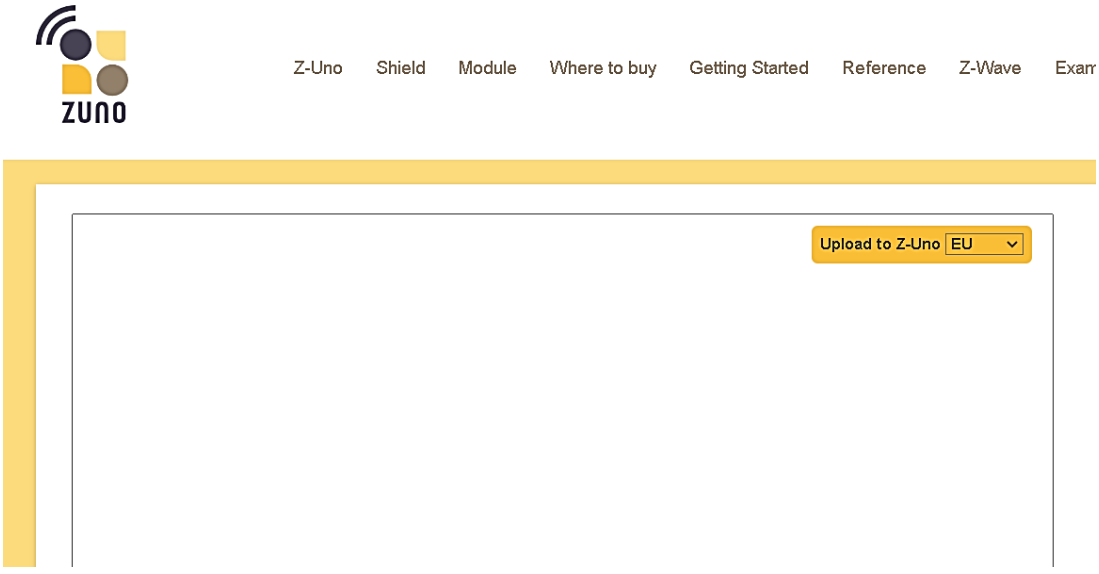
En Arduino IDE, programaremos dos dispositivos, ESP32 y Z-Uno 2. El ESP32 leerá las mediciones del PZEM-004T V3 y esperará señales digitales en sus pines para realizar los accionamientos, por lo cual, el código es el siguiente.

```
#include <PZEM004Tv30.h>

// Verifica y define los pines RX y TX para la comunicación con el PZEM si no se han definido previamente
#if !defined(PZEM_RX_PIN) && !defined(PZEM_TX_PIN)
#define PZEM_RX_PIN 4
#define PZEM_TX_PIN 2
#endif

// Define el Serial que se usará para comunicarse con el PZEM si no se ha definido previamente
#if !defined(PZEM_SERIAL)
#define PZEM_SERIAL Serial1
#endif

// Configuración específica para ESP32 y ESP8266
#if defined(ESP32)
PZEM004Tv30 pzem(PZEM_SERIAL, PZEM_RX_PIN, PZEM_TX_PIN); // Inicializa el PZEM usando los pines definidos para ESP32
#elif defined(ESP8266)
PZEM004Tv30 pzem(PZEM_SERIAL); // Inicializa el PZEM para ESP8266 (los pines se definen de manera diferente)
#endif

// Define los pines para la comunicación Serial2 con otro dispositivo (por ejemplo, un módulo SIM800L)
#define RXp2 17
#define TXp2 16

// Declaración de pines y variables
int pinMicro  = 15; // Pin para el sensor (nombrado "micro")
int estadoMicro = 0; // Variable para almacenar el estado actual del sensor

const int pinRele = 13; // Pin para el relé
```

```
30. const int pinEstadoRele = 12; // Pin para leer el estado del relé (no se usa en este código)
31. const int pinResetEnergia = 14; // Pin para resetear la energía medida por el PZEM
32.
33. int pinLed = 26; // Pin para el LED
34.
35. // Variables para almacenar las mediciones del PZEM
36. float voltaje = 0.0;
37. float corriente = 0.0;
38. float potencia = 0.0;
39. float energia = 0.0;
40. float frecuencia = 0.0;
41. float factorPotencia = 0.0;
42.
43. float energiaAnterior = 0; // Almacena la última medición de energía para comparar cambios
44.
45. unsigned long millisPrevios = 0; // Almacena el tiempo del último cambio de estado del LED
46. const long intervalo = 1000; // Intervalo de tiempo para acciones repetitivas (originalmente configurado incorrectamente)
47.
48. unsigned long millisPrevios2 = 0;
49. const unsigned long intervalo2 = 5000; // Intervalo de tiempo para verificar el estado del sensor "micro"
50.
51. void setup() {
52.   // Configuración de pines
53.   pinMode(pinRele, OUTPUT);
54.   pinMode(pinLed, OUTPUT);
55.   pinMode(pinMicro, INPUT);
56.   pinMode(pinEstadoRele, INPUT);
57.   pinMode(pinResetEnergia, INPUT);
58.
59.   digitalWrite(pinRele, HIGH); // Inicialmente el relé está desactivado
60.
61.   // Inicia la comunicación Serial
62.   Serial.begin(115200);
63.   Serial2.begin(115200, SERIAL_8N1, RXp2, TXp2);
64. }
65.
66. void loop() {
67.   unsigned long millisActual = millis(); // Obtiene el tiempo actual
68.   // Lee los valores del sensor PZEM
69.   voltaje = pzem.voltage();
70.   corriente = pzem.current();
71.   potencia = pzem.power();
72.   energia = pzem.energy();
73.   frecuencia = pzem.frequency();
74.   factorPotencia = pzem.pf();
75.
76.   // Verifica el estado del sensor "micro"
77.   if (digitalRead(pinMicro) == HIGH && estadoMicro == LOW) {
78.     estadoMicro = HIGH; // Cambia el estado a HIGH
79.     millisPrevios2 = millis(); // Guarda el momento actual
80.   }
81.
82.   // Verifica si han pasado 5 segundos desde la última detección
83.   if (estadoMicro == HIGH && millis() - millisPrevios2 >= intervalo2) {
84.     estadoMicro = LOW; // Cambia el estado a LOW después de 5 segundos
85.   }
86.
87.   // Controla el relé basado en el estado de un pin
88.   if (digitalRead(pinEstadoRele) == HIGH) {
89.     digitalWrite(pinRele, LOW); // Enciende el relé
90.   } else if (digitalRead(pinEstadoRele) == LOW) {
91.     digitalWrite(pinRele, HIGH); // Apaga el relé
92.   }
93.
94.   // Resetea las mediciones de energía del PZEM si se presiona un botón
```

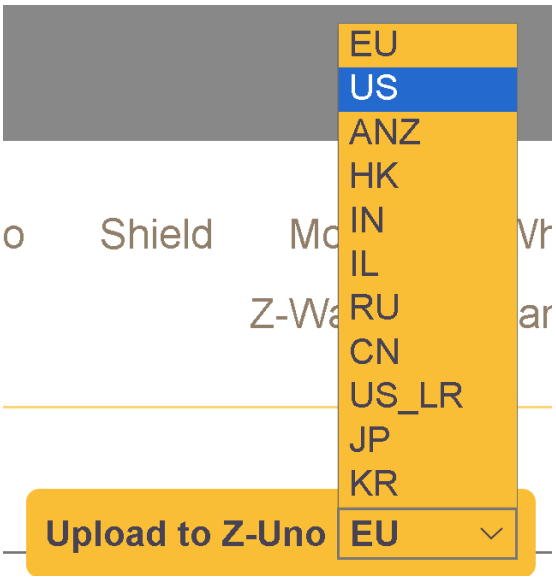
```
 95.   if (digitalRead(pinResetEnergia) == HIGH) {
 96.     pzem.resetEnergy(); // Resetea la energía
 97.   }
 98.
 99.   // Envía los datos recogidos cada intervalo definido
100.   if (millisActual - millisPrevios >= intervalo) {
101.       String data = "V" + String(voltaje) + "C" + String(corriente, 3) + "P" +
String(potencia, 2) + "E" + String(energia, 3) + "F" + String(frecuencia) + "X" +
String(factorPotencia) + "M" + String(estadoMicro) + "Z";
102.
103.     Serial.println(data); // Muestra los datos en el Serial principal
104.     Serial2.println(data); // Envía los datos a través de Serial2
105.     millisPrevios = millisActual;
106.   }
107.
108.   // Enciende el LED si la energía ha aumentado desde la última medición
109.   if (energia > energiaAnterior) {
110.     digitalWrite(pinLed, HIGH); // Enciende el LED
111.       // No necesita guardar el tiempo en que se encendió el LED ya que se apaga
inmediatamente después de 50 ms
112.     delay(50); // Espera 50 milisegundos
113.     digitalWrite(pinLed, LOW); // Apaga el LED
114.   }
115.
116.   energiaAnterior = energia; // Actualiza la última medición de energía
117. }
118.
```

Por otro lado, utilizaremos la web de Z-Uno para subir el código a nuestro Z-Uno 2. Existen librerías para el IDE de Arduino, sin embargo contiene algunos errores. El editor se encuentra en este link.

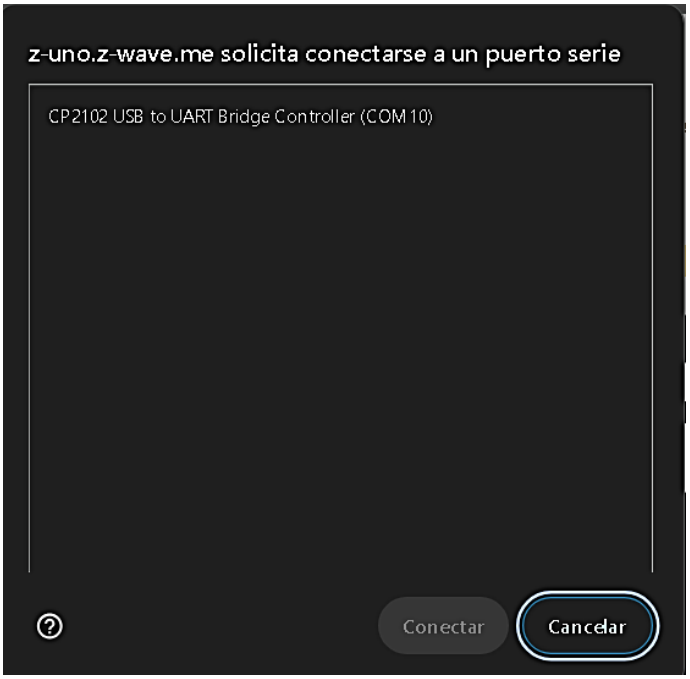


En esta página, copiaremos el siguiente código.

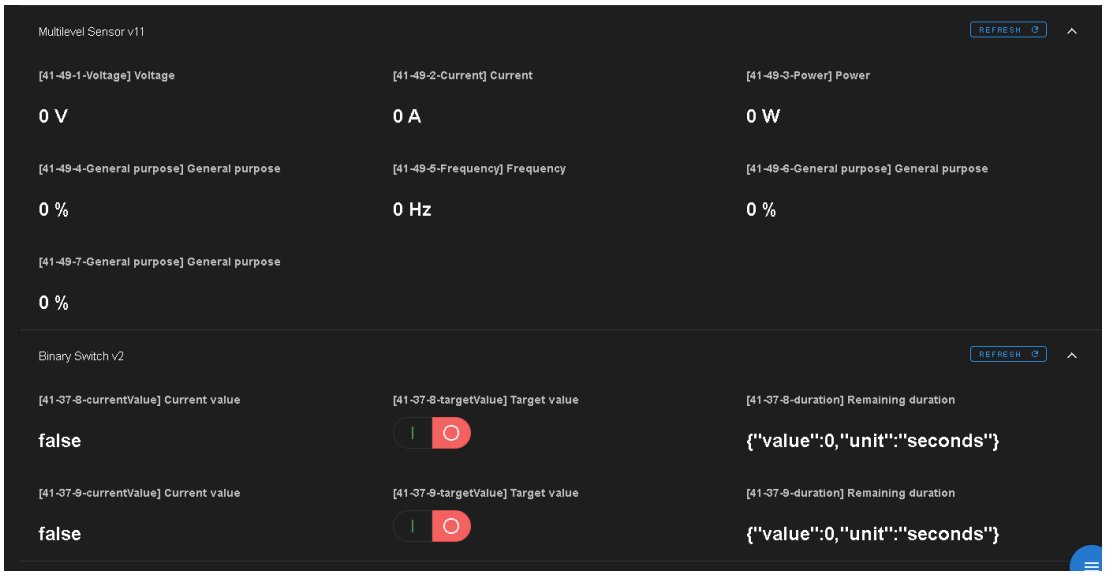
Es importante seleccionar la localización correcta antes de subir el código. En este caso seleccionamos US.



Cuando hacemos clic en subir, nos aparecerá una ventana que nos pedirá seleccionar el dispositivo USB. Ahí seleccionamos el puerto correspondiente a nuestro Z-Uno 2.



A partir de esto, el código se subirá en nuestro dispositivo y los sensores y actuadores se configurarán correctamente en Z-Wave JS UI y en Home Assistant.



Hay algunas mediciones que debemos reacondicionar colocándoles la coma en el lugar que corresponde, por eso, en configuration.yaml de Home Assistant se utilizaron sensores plantilla para asignarles un nombre de Nodo identificable y colocar correctamente estas mediciones.

```yaml
  - platform: template
    sensors:
      voltage7:
        friendly_name: "Nodo7 Voltaje"
        unique_id: "nodo7_voltaje"
        unit_of_measurement: "V"
        value_template: "{{ states('sensor.node_41_voltage') | float }}"
      current7:
        friendly_name: "Nodo7 Corriente"
        unique_id: "nodo7_corriente"
        unit_of_measurement: "A"
        value_template: "{{ (states('sensor.node_41_current') | float) * 0.1 }}"
      power7:
        friendly_name: "Nodo7 Potencia"
        unique_id: "nodo7_potencia"
        unit_of_measurement: "W"
        value_template: "{{ states('sensor.node_41_power') | float }}"
      energy7:
        friendly_name: "Nodo7 Energia"
        unique_id: "nodo7_energia"
        unit_of_measurement: "kWh"
        value_template: "{{ (states('sensor.node_41_general_purpose') | float) * 0.1 }}"
      frequency7:
        friendly_name: "Nodo7 Frecuencia"
        unique_id: "nodo7_frecuencia"
        unit_of_measurement: "Hz"
        value_template: "{{ states('sensor.node_41_frequency') | float }}"
      powerfactor7:
        friendly_name: "Nodo7 Factor de Potencia"
        unique_id: "nodo7_factordepotencia"
        value_template: "{{ states('sensor.node_41_general_purpose_6') | float }}"
      microfono7:
        friendly_name: "Nodo7 Microfono"
        unique_id: "nodo7_microfono"
        value_template: "{{ states('sensor.node_41_general_purpose_7') | float }}"
```

```
36.
37.       current7_calculated:
38.         friendly_name: "Nodo7 Corriente  Calculada"
39.         unique_id: "nodo7_corriente_calculada"
40.         unit_of_measurement: "A"
41.                                                 value_template:     "{{     0.0     if
(((states('sensor.power7')|float)*(states('sensor.powerfactor7')|float))|float) == 0.0 else
(((states('sensor.power7')|float)         /         ((states('sensor.powerfactor7')|float)
*(states('sensor.voltage7')|float))) |round(3)) }}"
42.       powerapp7_calculated:
43.         friendly_name: "Nodo7 Potencia Aparente"
44.         unique_id: "nodo7_potencia_aparente"
45.         unit_of_measurement: "VA"
46.                     value_template:  "{{  0.0  if  (((states('sensor.power7')|float)  *
(states('sensor.powerfactor7')|float))|float) == 0.0 else (( (states('sensor.power7')|float)
/ ( (states('sensor.powerfactor7')|float) )) | round(2)) }}"
47.       powerreact7_calculated:
48.         friendly_name: "Nodo7 Potencia Reactiva"
49.         unique_id: "nodo7_potencia_reactiva"
50.         unit_of_measurement: "VAr"
51.                                                 value_template:     "{{     0.0     if
(((states('sensor.power7')|float)*(states('sensor.powerfactor7')|float))|float)  ==0.0  or
(states('sensor.powerfactor7')|float)               ==1               else               (
((states('sensor.voltage7')|float)*((states('sensor.current7_calculated')|float))*((sin((a
cos((states('sensor.powerfactor7'))|round(3)))))))) | round(2))}}"
52.
```

Con esto, ya tendremos nuestras mediciones y los actuadores correctamente configurados con la red Z-Wave y Home Assistant.

MANTENIMIENTO

Revisaremos el mantenimiento preventivo, correctivo y predictivos que se le pueden hacer a los distintos elementos del Sistema.

Mantenimiento Preventivo

Para prevenir daños futuros, es recomendable hacer revisiones constantes del siguiente tipo:

Limpieza General:

Usar aire comprimido para remover el polvo de las cajas y de los componentes internos. Para áreas sensibles, utilizar un cepillo antiestático. La limpieza previene problemas de sobrecalentamiento y fallos en los componentes.

Revisión de Ventiladores:

En el caso del Nodo Maestro que tiene ventiladores, verificar que funcionen correctamente y que no estén obstruidos por polvo o suciedad. Reemplaza los ventiladores que presenten ruidos extraños o no giren adecuadamente.

Inspección Visual:

Buscar signos de desgaste o daño en los componentes, como condensadores hinchados en las fuentes de alimentación, quemaduras o corrosión en los PCBs y cables.

Verificación de Conexiones:

Asegúrate de que todas las conexiones, incluidos los cables USB del Nodo Maestro y otros conectores, estén firmes y no presenten signos de daño o de malos contactos.

Mantenimiento Correctivo

Cuando se identifiquen problemas durante el mantenimiento preventivo o en el uso diario, el mantenimiento correctivo será necesario.

Reemplazo de Componentes Dañados:

Si detectas componentes dañados o que funcionan mal, reemplázalos inmediatamente para evitar daños adicionales.

Resubida del Firmware:

Mantén los microcontroladores y dispositivos con el último firmware para asegurar el funcionamiento esperado. Verifica este manual para recordar en qué versiones de software fueron escritos los códigos.

Reparación de Circuitos:

Si hay problemas en los PCBs, como pistas rotas o soldaduras frías, es necesario realizar reparaciones con un soldador y material de soldadura adecuados.

Mantenimiento Predictivo

Con base en el funcionamiento de los equipos, intenta predecir cuándo podrían fallar los componentes y reemplazarlos o repararlos antes de que ocurra la falla.

Monitoreo de Temperatura:

Usa sensores de temperatura para monitorear el calor generado por los componentes. Altas temperaturas constantes pueden indicar problemas futuros y cortos.

Análisis de Rendimiento:

Evalúa el rendimiento de los sistemas regularmente para detectar posibles problemas antes de que se conviertan en fallas.

Registro de Mantenimiento:

Lleva un registro detallado de todas las acciones de mantenimiento, incluyendo cambios de componentes, actualizaciones de software y reparaciones. Esto te ayudará a identificar patrones y predecir futuras necesidades de mantenimiento.

CONTACTO

Para dudas y sugerencias, contáctate con los siguientes medios.


0994796445


erick-full@hotmail.com


Nuevo Santo Domingo Sector 1, Juan Pablo Neruda y Joaquín Balaguer, Santo Domingo de los Colorados, Santo Domingo de los Tsáchilas

Cronograma de Actividades

CRONOGRAMA DE ACTIVIDADES

Objetivos específicos / Actividades		22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	39	40	41	42	43	44	45	46	47	48	49	50	51	52	1	2	3	4	% ponderado
1. Describir los tipos de medidores y protocolos de comunicación inalámbrica.	1.1. Descripción de conceptos básicos de electricidad.	X	X																																	30
	1.2. Descripción de los medidores y sus tipos.			X	X	X	X																													
	1.3. Descripción los protocolos de comunicación.						X	X	X	X																										
2. Diseñar un Sistema Inteligente de Medición de Consumo Eléctrico.	2.1 Desarrollo de la metodología.										X	X																								35
	2.2 Diseño de las PCBs.												X	X	X	X	X	X	X	X																
	2.3 Programación de los dispositivos.																X	X	X	X	X	X	X	X	X	X										
3. Implementar un Sistema Inteligente de Medición de Consumo Eléctrico en una red residencial.	3.1 Implementación de dispositivos.																							X	X	X	X	X	X	X						35
	3.2 Realizar pruebas de funcionamiento.																														X	X	X	X	X	
	3.3 Descripción de resultados.																																			

100

Recursos y Presupuesto

Este proyecto fue financiado con fondos propios en su totalidad.

Tabla 30

Recursos y Presupuesto del Proyecto

Descripción	Precio Unitario	Cantidad	Precio Total ($)	Porcentaje
Ordenadores Monoplaca	80	1	80	7.12%
Microcontroladores	5	7	35	3.11%
Módulos de comunicación inalámbrica	24	10	240	21.6%
Módulos de medición eléctrica	15	7	105	9.34%
Módulos actuadores	2	7	14	1.25%
Sensores	4	7	28	2.5%
Componentes electrónicos comunes	20	1	20	1.77%
Impresión 3D	150	1	150	13.34%
PCBs	150	1	150	13.34%
Tablet (Monitor)	50	1	50	4.44%
Componentes del Tablero	150	1	150	13.34%
Gastos Imprevistos (10%)	102	1	102	9.07%
		TOTAL	1124	100%