UNIVERSIDAD CATÓLICA BOLIVIANA

INFORME
EXAMEN FINAL

ASIGNATURA:
INTERNET DE LAS COSAS

SISTEMA DE MONITOREO DE CONSUMO ELÉCTRICO DE UN CIRCUITO

Grupo: *Grupo 5*

INTEGRANTES

Alejandro Mollinedo Rodriguez
Ronald Marcelo Narvaez Estevez
Daira Fabiana Arce Quisbert
Valdir Adhemar Flores Moya
Kenneth Gabriel Lopez Castillo

DOCENTE:
ING. PAMELA VALENZUELA

CARRERA: INGENIERÍA DE SISTEMAS
FECHA DE ENTREGA: 09/12/2024
LA PAZ – BOLIVIA
2024

Índice

1. Introducción:

1.1. Contexto del proyecto

La medición del consumo eléctrico en un circuito o bajo la misma lógica aplicado a una vivienda, utilizando los medidores comunes instalados en los hogares, puede ser un proceso tedioso y poco práctico, especialmente cuando se busca un análisis detallado o en tiempo real. Este proyecto aborda esta problemática mediante el uso de tecnología IoT (Internet de las Cosas), que permite recopilar datos constantes y precisos sobre el consumo eléctrico a través de un microcontrolador ESP32. El sistema utiliza sensores de corriente ACS712 para medir el consumo eléctrico de los clientes simulados mediante cualquier dispositivo electrónico conectado. Estos datos son transmitidos desde el ESP32 a un servidor local, donde se almacenan en una base de datos. La información registrada es accesible mediante un sistema ABM para su gestión y se visualiza en un dashboard con gráficas interactivas, lo que facilita la comprensión y el análisis rápido de los datos. Al replicar un escenario práctico con los clientes simulados, este proyecto busca demostrar cómo la tecnología IoT puede aplicarse para mejorar la eficiencia energética en distintos entornos.

1.2. Descripción breve del problema que aborda el sistema IoT.

Los medidores tradicionales de consumo eléctrico no permiten un monitoreo en tiempo real ni ofrecen detalles sobre los patrones de consumo, lo que dificulta la optimización del uso de energía. Esto genera problemas para identificar hábitos ineficientes y controlar los gastos de manera óptima. El sistema IoT propuesto resuelve este problema al medir el consumo eléctrico en tiempo real con sensores conectados a un ESP32, almacenando los datos en una base de datos y presentándose en un dashboard interactivo. Esto facilita un análisis rápido y preciso, promoviendo el ahorro energético y un mejor manejo de los recursos.

1.3. Objetivo general del proyecto.

Implementar un prototipo de sistema IoT (Internet de las Cosas) para la medición precisa y observación del consumo de energía eléctrica en un circuito en tiempo real.

1.4. Objetivos específicos

- Estructurar una base de datos relacional para almacenar los datos de consumo de energía eléctrica de manera organizada y bien estructurada.
- Implementar un script en Micro Python para capturar y enviar los datos de consumo eléctrico desde el microcontrolador ESP32 al servidor local.
- Crear un dashboard para visualizar en tiempo real los datos de consumo de energía eléctrica.
- Desarrollar un script en PHP para el servidor que permita gestionar el alta, baja y modificación (ABM) de los registros en la base de datos.
- Desarrollar un script en PHP para realizar la inserción y consulta de los datos de consumo en la base de datos.

- Diseñar un diagrama de red para manejar la conexión entre cliente y servidor.
- Realizar una interfaz web para el manejo de todo el sistema accesible desde cualquier navegador y de fácil uso.

2. Marco Teórico

2.1. Explicación de conceptos esenciales

2.1.1. IoT: Principios básicos, ventajas y aplicaciones.

El Internet de las Cosas conecta dispositivos físicos a internet, permitiendo la recopilación y el intercambio de datos en tiempo real. Este enfoque facilita la automatización y la toma de decisiones inteligentes en aplicaciones como el hogar inteligente, la agricultura, la salud y la gestión de energía. Las ventajas principales del IoT incluyen:

- Monitoreo en tiempo real, permitiendo un control constante de sistemas y dispositivos.
- Optimización de recursos, ya que ayuda a reducir costos y mejorar la eficiencia energética.
- Los sistemas IoT pueden ampliarse para incluir más dispositivos o funcionalidades según sea necesario.

2.1.2. Microcontrolador seleccionado (ESP32): Capacidades técnicas y ventajas.

El ESP32 es un microcontrolador compacto y económico con conectividad Wi-Fi y Bluetooth integrada, ideal para aplicaciones IoT. Este dispositivo cuenta con un procesador dual-core de 240 MHz. Además, tiene una gran capacidad de memoria y múltiples pines GPIO, lo que le da mucha flexibilidad para interactuar con diferentes dispositivos. Es compatible con protocolos como HTTP y MQTT, lo que facilita su uso en aplicaciones de comunicación y control remoto. Su bajo consumo energético es otra ventaja, especialmente para proyectos que requieren operación autónoma o en dispositivos alimentados por baterías.

2.1.3. Protocolos de comunicación utilizados.

El protocolo HTTP es una tecnología estándar para transferir datos entre dispositivos IoT y servidores web. En este proyecto se utiliza para enviar datos desde el ESP32 a un servidor que gestiona la base de datos. Es simple, ampliamente soportado y fácil de implementar. Aunque no es tan eficiente como MQTT en términos de consumo energético, su versatilidad lo hace adecuado para esta aplicación.

2.1.4. Sensores utilizados.

- **Sensor ACS712**
  - **Tipo:** Sensor de corriente basado en efecto Hall.

- **Principio de funcionamiento:** Mide la corriente eléctrica que pasa por un circuito, generando una señal proporcional que puede ser interpretada por el ESP32.
- **Aplicaciones:** Monitoreo de consumo eléctrico en circuitos residenciales o industriales.
- **Ventajas:** Precisión, bajo costo y facilidad de integración con microcontroladores.

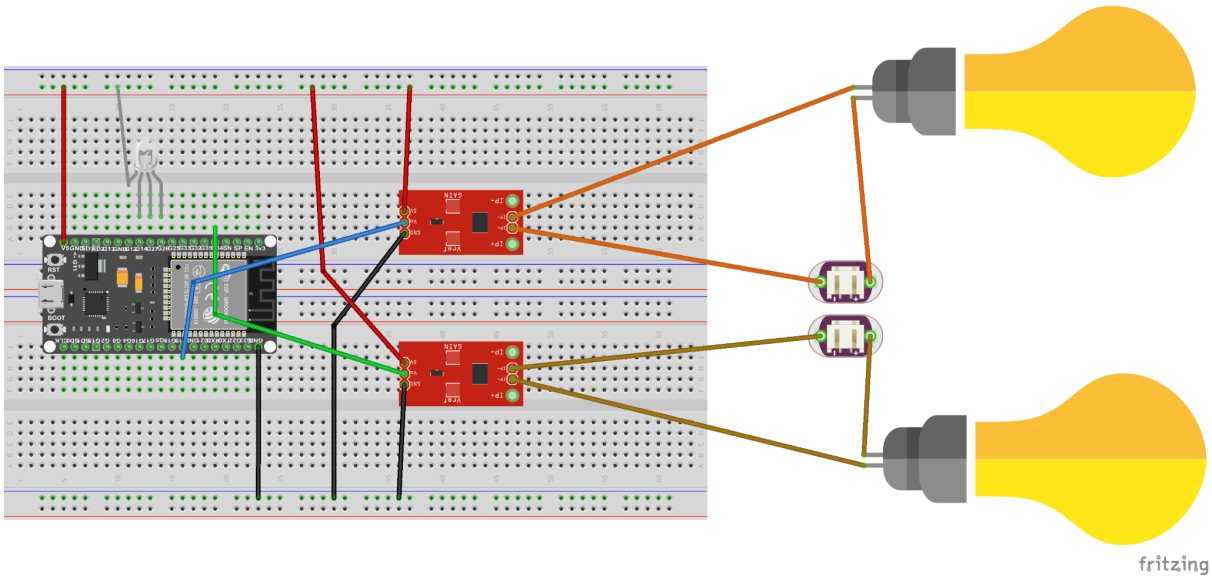
Este sensor se utiliza para medir tanto la corriente como el voltaje en el circuito monitoreado, generando los datos necesarios para calcular la potencia y el consumo energético (Kwh).

3. Desarrollo del Proyecto

3.1. Diseño del Circuito:

El diseño del circuito es una parte fundamental del proyecto, ya que asegura la correcta interacción entre los componentes de hardware: el microcontrolador ESP32, los sensores de corriente (ACS712), y los módulos de comunicación.

3.1.1. Diagrama esquemático con software (Fritzing).



Se realizó el diseño del circuito para la documentación utilizando el software Fritzing. En este se observa como el microcontrolador ESP32 se coloca sobre dos protoboards para que quepa bien. Se conectan a los pines de 5V y GND con sus líneas respectivas para alimentar a los dos sensores ACS712 que de igual manera se conectan al ESP32. A los sensores se conecta un circuito en serie, en este caso en el circuito se incluyen un foco que se conecta directamente al enchufe tomacorriente de 220V mostrando como seria la conexion y como se conecta el circuito al sensor,si bien en el diagrama se usa un foco, de ese modo se puede conectar cualquier dispositivo electrónico para realizar su medición. También conectado al ESP32 está un foco led RGB el cual parpadea cada vez que hace una medición de la corriente para hacerlo más dinámico.

3.1.2. Conexiones entre microcontroladores y sensores.

La conexión entre el ESP32 y el ACS712 es sencilla, ya que el sensor se comunica mediante un canal analógico. Esto significa que la salida del sensor entrega un voltaje proporcional a la corriente detectada, que luego es leído por el ADC del ESP32.

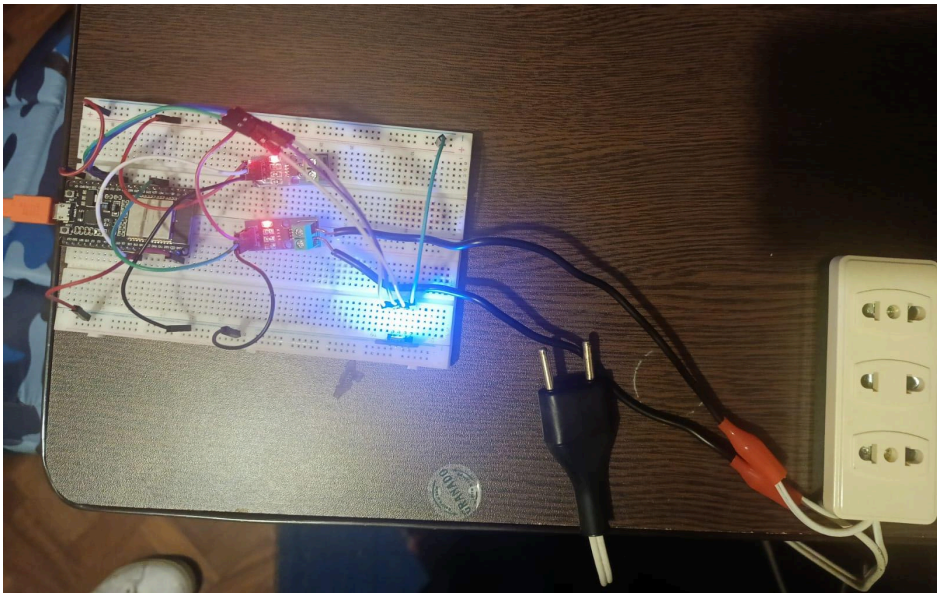
Conexiones específicas:

- ACS712 VCC → ESP32 5V: Alimenta el sensor.
- ACS712 GND → ESP32 GND: Establece el circuito de referencia.
- ACS712 OUT → ESP32 ADC GPIO 34: Lee la corriente para realizar los cálculos del consumo eléctrico.

3.1.3. Justificación del protocolo de comunicación utilizado.

En este proyecto se utiliza el protocolo HTTP (HyperText Transfer Protocol) para la comunicación entre el ESP32 y el servidor. Este protocolo se selecciona por su simplicidad y amplia compatibilidad, ya que puede implementarse fácilmente en el ESP32 mediante bibliotecas como `urequests` en MicroPython. Además, HTTP es compatible con servicios web y APIs RESTful, lo que lo convierte en una opción versátil y funcional.
Para el intercambio de datos, se emplea el formato JSON, que es liviano y ampliamente soportado, facilitando la lectura y el procesamiento tanto en el microcontrolador como en el servidor. Asimismo, la elección de HTTP garantiza una integración fluida con el backend desarrollado en PHP, dado que la mayoría de los servidores y bases de datos web lo admiten de forma nativa. Esto permite establecer una comunicación eficiente entre el hardware IoT y el sistema de gestión basado en la web.

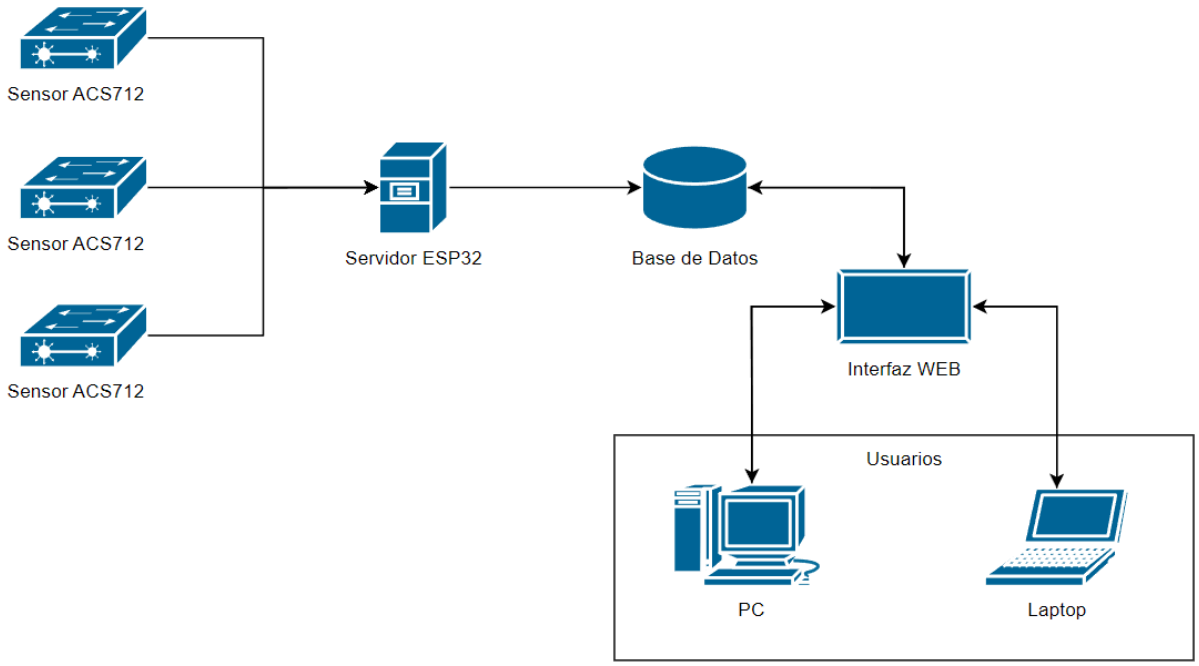
3.1.4. Fotografía del circuito físico.



En la imagen se observa la conexión física del circuito. Se ve que los sensores ACS712 y el led RGB de 4 pines conectan con el microcontrolador ESP32. Se observan también las conexiones de voltaje (5V) y tierra (GND). También se logra ver como se conecta el circuito en serie al sensor por medio de dos cables negros de cobre, estos se conectan con un alargador con 3 entradas de tomacorriente, por lo que la conexion sigue en serie y todo dispositivo electrico que se conecte a este sera parte del circuito y sera medido con el sensor, esto acaba conectando con un enchufe con el cual se provee de voltaje al circuito.

3.2. Diseño de la Red IoT

3.2.1. Diagrama de la red con conexiones entre nodos IoT, servidor y clientes.



El diagrama del consumo eléctrico se compone de varios elementos interconectados. Los **sensores ACS712** miden el consumo eléctrico y envían los datos al **servidor ESP32**, que actúa como un puente entre el sensor y la **base de datos**, donde se almacenan y organizan los registros de consumo. A través de una **interfaz web**, los **usuarios** pueden acceder de manera remota a los datos almacenados, visualizar gráficos y estadísticas, y en el caso de los administradores, gestionar los registros mediante funciones de ABM.

3.2.2. Explicación de los roles de cada dispositivo (nodos, servidor, clientes).

Rol de los nodos

En el sistema, los nodos son los encargados de captar y enviar los datos necesarios para monitorear el consumo de electricidad. En este caso, el nodo principal está compuesto por un sensor que mide el gasto energético conectado a un ESP32, que actúa como puente entre el sensor y el resto del sistema. El nodo es el sensor ACS712 que mide continuamente la energía consumida y al estar manejado por el microcontrolador los datos se envían al servidor creado por este mismo. Este rol es crucial porque los nodos son los responsables de captar la información en tiempo real desde el entorno físico y transformarla en datos digitales que puedan ser procesados.

Rol del servidor

El servidor es el núcleo del sistema, donde se almacenan, organizan y procesan los datos enviados por los nodos. Este rol lo cumple el ESP32, pues se conecta a una red Wifi e inicia un servidor local en la dirección IP correspondiente y los clientes pueden conectarse al servidor a través de esta dirección IP y acceder a los nodos (ACS712) para captar los datos del consumo eléctrico de manera remota. Por otro lado en adición a esto el rol del servidor es también el sistema desarrollado en PHP que incluye una base de datos para registrar el gasto de energía. Además de guardar los datos, el servidor ofrece funcionalidades como el ABM (Alta, Baja y Modificación), que permite gestionar los registros, y un dashboard visual que presenta estadísticas y análisis del consumo energético, todo mostrado en la interfaz Web. El servidor asegura que la información esté accesible para los usuarios en cualquier momento y que sea presentada de forma clara y estructurada. Es, en esencia, el cerebro del sistema, coordinando toda la información que fluye entre los nodos y los clientes.

Rol de los clientes

Los clientes son los dispositivos que permiten a los usuarios finales interactuar con el sistema. En este proyecto, los clientes acceden al servidor y su dashboard a través de navegadores web en computadoras, teléfonos móviles o tabletas. Estos dispositivos permiten que los usuarios comiencen el proceso de medición de consumo eléctrico con el nodo (sensor ACS712), consulten el historial de consumo eléctrico, analicen gráficos y estadísticas, y realicen cambios en los datos mediante las funcionalidades del ABM. El cliente actúa como la interfaz de usuario del sistema, traduciendo el trabajo realizado por los nodos y el servidor en una experiencia accesible y comprensible para los usuarios.

3.2.3. Selección de protocolo de comunicación.

Para este sistema, el protocolo seleccionado es HTTP debido a su simplicidad y compatibilidad con la mayoría de las aplicaciones web.

Ventajas del uso de HTTP:

- Fácil implementación en el ESP32 utilizando librerías como WiFiClient y HTTPClient.
- Compatibilidad con servidores web y bases de datos.
- Adecuado para aplicaciones que no requieren comunicaciones extremadamente rápidas o de baja latencia.

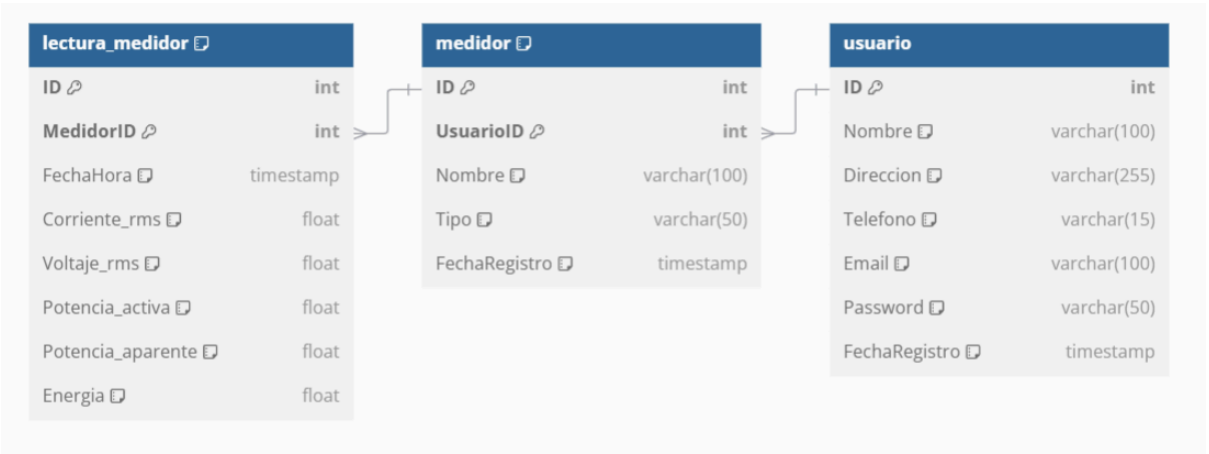
3.3. Diseño de la Base de Datos

3.3.1. Estructura de la tabla principal, con descripción detallada de los campos.

Dado que el proyecto trata sobre la medición de energía en una vivienda usando sensores, la tabla principal relevante sería **lectura_medidor**, ya que contiene los datos más críticos: las lecturas de corriente, voltaje, potencia y energía. Sus campos son los siguientes:

- **ID:** Identificador único de cada lectura. Esto asegura que cada registro sea único y puede ser referenciado.
- **MedidorID:** Relaciona cada lectura con el medidor correspondiente, permitiendo identificar el dispositivo que generó los datos.
- **FechaHora:** Registra el momento exacto en que se tomó la lectura, necesario para analizar patrones de consumo a lo largo del tiempo.
- **Corriente_rms**: Valor medido de la corriente (en amperios). Este campo representa la magnitud real de la corriente eléctrica registrada por el sensor ACS712.
- **Voltaje_rms**: Valor medido del voltaje (en voltios). Este dato proviene se asume del voltaje común de una vivienda ideal de 220 voltios.
- **Potencia_activa:** Calculada como el producto de corriente, voltaje y el factor de potencia, representa el consumo real de energía en vatios (W).
- **Potencia_aparente:** Calculada como el producto directo de corriente y voltaje, sin considerar el factor de potencia, expresada en voltios-amperios (VA).
- **Energia:** Energía acumulada consumida por el dispositivo o circuito, medida en kilovatios-hora (kWh), calculada a partir de la potencia activa en función del tiempo.

3.3.2. Diagrama entidad-relación.



3.3.3. Script SQL para crear las tablas.

```sql
-- Base de datos: `bd_consumo_electrico`
--
-- --------------------------------------------------------
--
-- Estructura de tabla para la tabla `lectura_medidor`

CREATE TABLE `lectura_medidor` (
  `ID` int(11) NOT NULL,
  `MedidorID` int(11) NOT NULL,
  `FechaHora` timestamp NOT NULL DEFAULT current_timestamp(),
  `Corriente_rms` float DEFAULT NULL,
  `Voltaje_rms` float DEFAULT NULL,
  `Potencia_activa` float DEFAULT NULL,
  `Potencia_aparente` float DEFAULT NULL,
  `Energia` float DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

-- --------------------------------------------------------
--
-- Estructura de tabla para la tabla `medidor`

CREATE TABLE `medidor` (
  `ID` int(11) NOT NULL,
  `UsuarioID` int(11) NOT NULL,
  `Nombre` varchar(100) NOT NULL,
  `Tipo` varchar(50) NOT NULL,
  `FechaRegistro` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

-- --------------------------------------------------------
--
-- Estructura de tabla para la tabla `usuario`

CREATE TABLE `usuario` (
  `ID` int(11) NOT NULL,
  `Nombre` varchar(100) NOT NULL,
  `Direccion` varchar(255) DEFAULT NULL,
  `Telefono` varchar(15) DEFAULT NULL,
  `Email` varchar(100) DEFAULT NULL,
  `Password` varchar(50) DEFAULT NULL,
  `FechaRegistro` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
--
```

3.4. Desarrollo de Scripts

En este apartado se describen los scripts utilizados en el sistema para la gestión de datos y visualización en tiempo real, organizados por funcionalidad específica.

3.4.1. Script PHP para insertar y leer datos desde la base de datos.

El archivo **ManejoBaseDeDatos.php** cumple esta función. Este script gestiona la interacción con una base de datos MySQL, permitiendo insertar nuevos registros, como usuarios o lecturas de los sensores, y leer información almacenada.

3.4.2. Script en MicroPython para el envío de datos desde los nodos IoT.

El script **servidor esp32.py** se ejecuta en el ESP32 y se encarga de inicializar el servidor local con una dirección IP donde los clientes pueden conectarse y capturar datos del sensor de corriente (ACS712). Los valores calculados incluyen corriente RMS, voltaje RMS, potencia aparente, potencia activa y energía acumulada. Una vez procesados estos datos por los nodos se envían al servidor local iniciado previamente que luego los clientes recuperan mediante peticiones HTTP POST.

3.4.3. Script en PHP para ABM en la base de datos.

Los archivos **abm1.php** y **abm2.php** se encargan de cumplir la función del ABM de la base de datos, estos se ejecutan en la aplicación WEB. Estos son los responsables de acceder a la base de datos y mostrar la información completa de los usuarios y los medidores, como también tener las opciones de agregar un nuevo elemento, editar y eliminar un elemento según se necesite. Solo se puede acceder mediante un usuario que sea administrador.

3.4.4. Script en PHP para el dashboard.

El sistema cuenta con una conexión a la base de datos configurada en **bd.php**, donde se establecen los parámetros necesarios para interactuar con bd_consumo_electrico. Posteriormente, en **data.php**, se procesan las consultas SQL correspondientes a diferentes métricas, como consumo mensual, distribución diaria, gasto acumulado y comparaciones entre usuarios, entre otras.
Los resultados obtenidos de data.php son utilizados en los dashboards:
dashboard_usuario.php, que muestra gráficos personalizados para cada usuario utilizando datos como consumo mensual, historial reciente y comparaciones con el presupuesto donde se destaca el gasto que se tiene en bolivianos y en el gasto energetico que se tiene.

dashboard_administrador.php, que presenta los datos de los diferentes usuarios viendo el consumo y el gasto que estos tienen, igualmente pudiendo comparar el presupuesto que se tiene con el gasto obtenido hasta el momento.
De esta manera, los datos procesados y visualizados permiten una interpretación clara y accesible tanto para usuarios finales como para administradores.

3.4.5. Explicación y comentarios en cada script.

- El script **ManejoBaseDeDatos.php:**

Este archivo PHP actúa como una API para conectar una base de datos MySQL con una aplicación cliente, gestionando la comunicación a través de los **scripts script.js** y **script_pantalla.js**. Configura permisos CORS para aceptar solicitudes desde cualquier origen.
La conexión a la base de datos se realiza mediante PDO, lo que garantiza seguridad y flexibilidad. La API admite solicitudes **HTTP GET** y **POST** para realizar diferentes operaciones, como recuperar información de usuarios, medidores y lecturas eléctricas, además de registrar nuevos usuarios, medidores y lecturas. Para garantizar la validez de los datos, el código valida entradas y utiliza declaraciones preparadas, reduciendo riesgos como inyecciones SQL.
La API responde en formato JSON, facilitando la integración con la interfaz de usuario y manteniendo una comunicación clara entre el servidor y los scripts de cliente. En caso de errores, se devuelven mensajes descriptivos para que la aplicación pueda manejar las excepciones de forma adecuada.

```php
<?php
// Configuración de la conexión a la base de datos
header("Content-Type: application/json");

$servername = "localhost";
$username = "root";
$password = "";
$dbname = "bd_consumo_electrico";

try {
    // Crear conexión a la base de datos usando PDO
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // Leer el contenido de la solicitud POST
    $json = file_get_contents("php://input");
    $data = json_decode($json, true);

    // Guardar datos en log para verificación (opcional)
    file_put_contents("log.txt", $json . PHP_EOL, FILE_APPEND);

    // Verificar que se ha recibido la acción
    if (!isset($data['action'])) {
        echo json_encode(["status" => "error", "message" => "Acción no especificada"]);
        exit;
    }
```

```php
// Procesar la acción solicitada
switch ($data['action']) {
    case "register":
        if (isset($data['Nombre'], $data['Direccion'], $data['Telefono'], $data['Email'])) {
            $stmt = $conn->prepare("INSERT INTO usuario (Nombre, Direccion, Telefono, Email) VALUES (:Nombre, :Direccion, :Telefono, :Email)");
            $stmt->bindParam(':Nombre', $data['Nombre']);
            $stmt->bindParam(':Direccion', $data['Direccion']);
            $stmt->bindParam(':Telefono', $data['Telefono']);
            $stmt->bindParam(':Email', $data['Email']);

            if ($stmt->execute()) {
                echo json_encode(["status" => "success", "userID" => $conn->lastInsertId()]);
            } else {
                echo json_encode(["status" => "error", "message" => "Error al registrar usuario"]);
            }
        } else {
            echo json_encode(["status" => "error", "message" => "Datos incompletos para registro"]);
        }
        break;

    case "login":
        if (isset($data['Email'])) {
            $stmt = $conn->prepare("SELECT * FROM usuario WHERE Email = :Email");
            $stmt->bindParam(':Email', $data['Email']);
            $stmt->execute();

            $result = $stmt->fetch(PDO::FETCH_ASSOC);
            if ($result) {
                echo json_encode(["status" => "success", "user" => $result]);
            } else {
                echo json_encode(["status" => "error", "message" => "Usuario no encontrado"]);
            }
        } else {
```

```php
    } else {
        echo json_encode(["status" => "error", "message" => "Email no proporcionado"]);
    }
    break;

case "add_medidor":
    if (isset($data['UsuarioID'], $data['Nombre'], $data['Tipo'])) {
        $stmt = $conn->prepare("INSERT INTO medidor (UsuarioID, Nombre, Tipo) VALUES (:UsuarioID, :Nombre, :Tipo)");
        $stmt->bindParam(':UsuarioID', $data['UsuarioID']);
        $stmt->bindParam(':Nombre', $data['Nombre']);
        $stmt->bindParam(':Tipo', $data['Tipo']);

        if ($stmt->execute()) {
            echo json_encode(["status" => "success", "medidorID" => $conn->lastInsertId()]);
        } else {
            echo json_encode(["status" => "error", "message" => "Error al añadir medidor"]);
        }
    } else {
        echo json_encode(["status" => "error", "message" => "Datos incompletos para añadir medidor"]);
    }
    break;

case "add_lectura":
    if (isset($data['MedidorID'], $data['Corriente_rms'], $data['Voltaje_rms'], $data['Potencia_activa'], $data['Potencia_aparente'], $data['Energia'])) {
        $stmt = $conn->prepare("INSERT INTO lectura_medidor (MedidorID, FechaHora, Corriente_rms, Voltaje_rms, Potencia_activa, Potencia_aparente, Energia)
                                VALUES (:MedidorID, CURRENT_TIMESTAMP(), :Corriente_rms, :Voltaje_rms, :Potencia_activa, :Potencia_aparente, :Energia)");
        $stmt->bindParam(':MedidorID', $data['MedidorID']);
        $stmt->bindParam(':Corriente_rms', $data['Corriente_rms']);
        $stmt->bindParam(':Voltaje_rms', $data['Voltaje_rms']);
        $stmt->bindParam(':Potencia_activa', $data['Potencia_activa']);
        $stmt->bindParam(':Potencia_aparente', $data['Potencia_aparente']);
        $stmt->bindParam(':Energia', $data['Energia']);
```

```
            if ($stmt->execute()) {
                echo json_encode(["status" => "success", "message" => "Lectura añadida exitosamente"]);
            } else {
                echo json_encode(["status" => "error", "message" => "Error al añadir lectura"]);
            }
        } else {
            echo json_encode(["status" => "error", "message" => "Datos incompletos para añadir lectura"]);
        }
        break;

    case "get_medidores":
        $data = json_decode(file_get_contents("php://input"), true);
        $usuarioID = $data["UsuarioID"];

        // Consultar la base de datos para obtener los medidores
        $sql = "SELECT * FROM medidor WHERE UsuarioID = :usuarioID";
        $stmt = $conn->prepare($sql);
        $stmt->bindParam(':usuarioID', $usuarioID);
        $stmt->execute();

        // Obtener los resultados
        $medidores = $stmt->fetchAll(PDO::FETCH_ASSOC);

        if ($medidores) {
            echo json_encode(["status" => "success", "medidores" => $medidores]);
        } else {
            echo json_encode(["status" => "error", "message" => "No se encontraron medidores."]);
        }
```

```
        default:
            break;
    }
} catch (PDOException $e) {
    echo json_encode(["status" => "error", "message" => "Error en la conexión: " . $e->getMessage()]);
}

// Cerrar conexión
$conn = null;
?>
```

- El script **Servidor esp32.py:**

  Este script en Python implementa un servidor web en un dispositivo compatible con **MicroPython**, como un **ESP32**, para medir corriente y energía utilizando un sensor **ACS712**.
  Primero, el dispositivo se conecta a una red Wi-Fi mediante el módulo network, mostrando su dirección IP al establecer la conexión. Esto permite que el servidor web esté accesible en la red local.
  La función principal, **comenzar_medicion()**, realiza cálculos basados en los datos obtenidos del sensor ACS712. Utiliza un proceso de muestreo para calcular la corriente RMS, potencia aparente, potencia activa y energía consumida en kilovatios-hora (kWh). Los resultados se formatean como un diccionario y se preparan para ser enviados en formato JSON.
  El servidor web configurado escucha en el puerto 80. Responde a solicitudes específicas, como /comenzar_medicion, ejecutando la medición y devolviendo los datos. En caso de rutas no reconocidas, devuelve un error 404. El servidor incluye soporte para CORS, facilitando la integración con otras aplicaciones.

```python
from machine import ADC, Pin, PWM
import network
import socket
import time
import math
import json

# Configuración del LED RGB con PWM
led_red = PWM(Pin(14), freq=1000)
led_green = PWM(Pin(13), freq=1000)
led_blue = PWM(Pin(12), freq=1000)

def set_led_color(red, green, blue):
    """
    Establece el color del LED RGB usando valores PWM.
    Los valores deben estar en el rango de 0 a 1023.
    """
    led_red.duty(red)
    led_green.duty(green)
    led_blue.duty(blue)

# Función para realizar la medición
def comenzar_medicion():
    print("Procesando solicitud de medición...")

    # Encender LED verde para indicar que la medición ha comenzado
    set_led_color(0, 1023, 0)  # Verde

    try:
        # Configuración del sensor ACS712
        analog_pin = 34
        adc = ADC(Pin(analog_pin))
        adc.atten(ADC.ATTN_11DB)
        adc.width(ADC.WIDTH_12BIT)
        sensitivity = 0.066
        VCC = 5
        V_ZERO = 3.75  # Ajustar o calibrar dinámicamente

        def calcular_corriente_rms(samples=1250, sampling_time=0.001):
            sum_of_squares = 0
            for _ in range(samples):
                raw_value = adc.read()
                voltage = (raw_value / 4095) * VCC
                current = (voltage - V_ZERO) / sensitivity
                sum_of_squares += current ** 2
                time.sleep(sampling_time)
            return math.sqrt(sum_of_squares / samples)

        # Cálculo de corriente RMS
        corriente_rms = calcular_corriente_rms()

        # Datos de la medición
        voltaje_rms = 220.0  # Puedes ajustar esto según tu sistema
        potencia_aparente = voltaje_rms * corriente_rms
        potencia_activa = potencia_aparente * 0.85  # Factor de potencia
        energia = potencia_activa * (1 / 3600)  # Energía en kWh

        # Crear un diccionario con los resultados
        datos = {
            'corriente_rms': corriente_rms,
            'voltaje_rms': voltaje_rms,
            'potencia_aparente': potencia_aparente,
            'potencia_activa': potencia_activa,
            'energia': energia
        }

        # Cambiar el LED a rojo para indicar que la medición ha finalizado
        set_led_color(1023, 0, 0)  # Rojo

        return datos

    except Exception as e:
        print("Error al realizar la medición:", e)
        set_led_color(1023, 0, 0)  # Rojo en caso de error
        return {"error": "Hubo un problema con la medición"}
```

```python
# Función del servidor (sin cambios principales)
def start_server():
    sta_if = network.WLAN(network.STA_IF)
    sta_if.active(True)
    sta_if.connect('Flia Estevez', 'falquito2619')
    while not sta_if.isconnected():
        time.sleep(1)
    print('Conectado a la red Wi-Fi, IP:', sta_if.ifconfig()[0])
    addr = socket.getaddrinfo('0.0.0.0', 80)[0][-1]
    s = socket.socket()
    s.bind(addr)
    s.listen(1)
    print('Escuchando en', addr)
    while True:
        cl, addr = s.accept()
        print('Cliente conectado desde', addr)
        try:
            request = cl.recv(1024).decode('utf-8')
            print("Solicitud recibida:", request)
            if 'GET /comenzar_medicion' in request:
                print('Procesando solicitud de medición...')
                medicion = comenzar_medicion()
                cl.send('HTTP/1.1 200 OK\r\n')
                cl.send('Content-Type: application/json\r\n')
                cl.send('Access-Control-Allow-Origin: *\r\n')
                cl.send('Access-Control-Allow-Headers: Content-Type\r\n')
                cl.send('Access-Control-Allow-Methods: GET\r\n')
                cl.send('\r\n')
                cl.send(json.dumps(medicion))
                print('Datos enviados:', medicion)
            else:
                cl.send('HTTP/1.1 404 Not Found\r\n')
                cl.send('Content-Type: application/json\r\n')
                cl.send('Access-Control-Allow-Origin: *\r\n')
                cl.send('\r\n')
                cl.send(json.dumps({"error": "Ruta no encontrada"}))
        except Exception as e:
            print("Error en el procesamiento de la solicitud:", e)
            cl.send('HTTP/1.1 500 Internal Server Error\r\n')
            cl.send('Content-Type: application/json\r\n')
            cl.send('Access-Control-Allow-Origin: *\r\n')
            cl.send('\r\n')
            cl.send(json.dumps({"error": "Hubo un problema con la medición"}))
        finally:
            cl.close()
# Iniciar el servidor web
start_server()

```

3.5. Aporte Ingenieril

3.5.1. Descripción Breve

El sistema desarrollado es una aplicación web integrada con IoT que permite monitorear en tiempo real el consumo eléctrico en viviendas. Utiliza un ESP32 como nodo principal, sensores ACS712 para medir corriente, y tecnologías como PHP y MySQL para almacenar y extraer datos mediante una página web alojada en XAMPP. Python se utiliza exclusivamente para capturar y calcular datos de medición en el ESP32.

3.5.2. Problema que resuelve

El sistema aborda la falta de monitoreo en tiempo real en los medidores tradicionales, que no proporcionan detalles suficientes sobre el consumo eléctrico ni permiten identificar patrones de uso ineficientes. Esto dificulta la optimización energética y el control de gastos. La solución permite a los usuarios acceder a información detallada y actualizada sobre el consumo, promoviendo un uso más consciente de la energía.

3.5.3. Características clave

- Medición en tiempo real: Captura de parámetros eléctricos como corriente, potencia activa y energía acumulada mediante sensores conectados a un ESP32 y procesados en Python.
- Extracción y gestión de datos: Uso de PHP para almacenar y recuperar datos en una base de datos MySQL, accesibles desde una página web alojada en XAMPP.
- Página web interactiva: Permite a los usuarios visualizar datos históricos, gestionar registros y monitorear el consumo eléctrico.
- Escalabilidad: Posibilidad de integrar nuevos nodos, sensores o funcionalidades para aplicaciones en edificios o industrias.
- Gestión de datos (ABM): Funcionalidad de alta, baja y modificación para registros de consumo, accesible desde la interfaz web.
- Dashboard interactivo: Visualización de datos en tiempo real a través de gráficas intuitivas, desarrolladas con bibliotecas como Dash y Plotly.

3.5.4. Beneficio para el usuario

El sistema ofrece optimización energética al permitir que los usuarios identifiquen hábitos de consumo ineficientes y tomen decisiones informadas para reducir costos. Además, proporciona control en tiempo real, brindando acceso inmediato a datos actualizados desde cualquier dispositivo con conexión a internet. Su facilidad de uso se garantiza mediante una interfaz gráfica intuitiva, que hace que la información sea comprensible incluso para quienes no tienen experiencia técnica. Finalmente, promueve la sostenibilidad al incentivar el uso responsable de la energía, contribuyendo a reducir el impacto ambiental asociado al consumo excesivo.

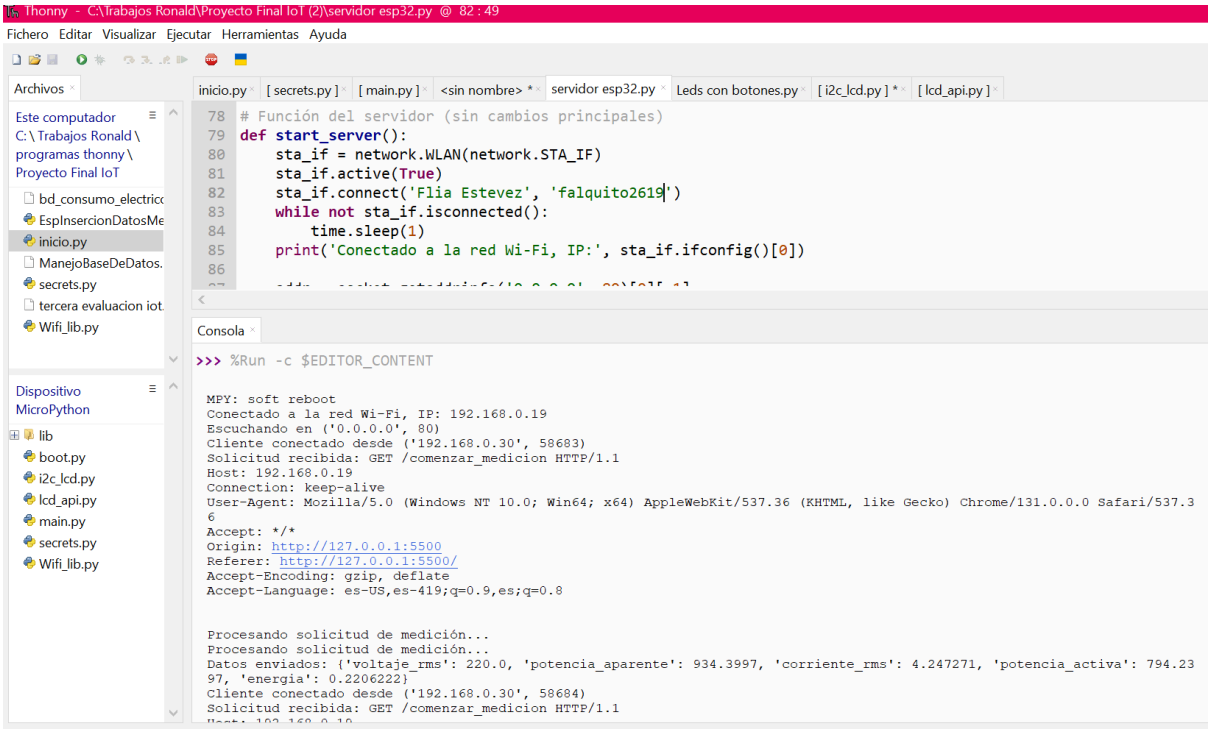
4. Resultados y Análisis

4.1. Evidencias del sistema funcionando:

4.1.1. Datos recolectados por los sensores.

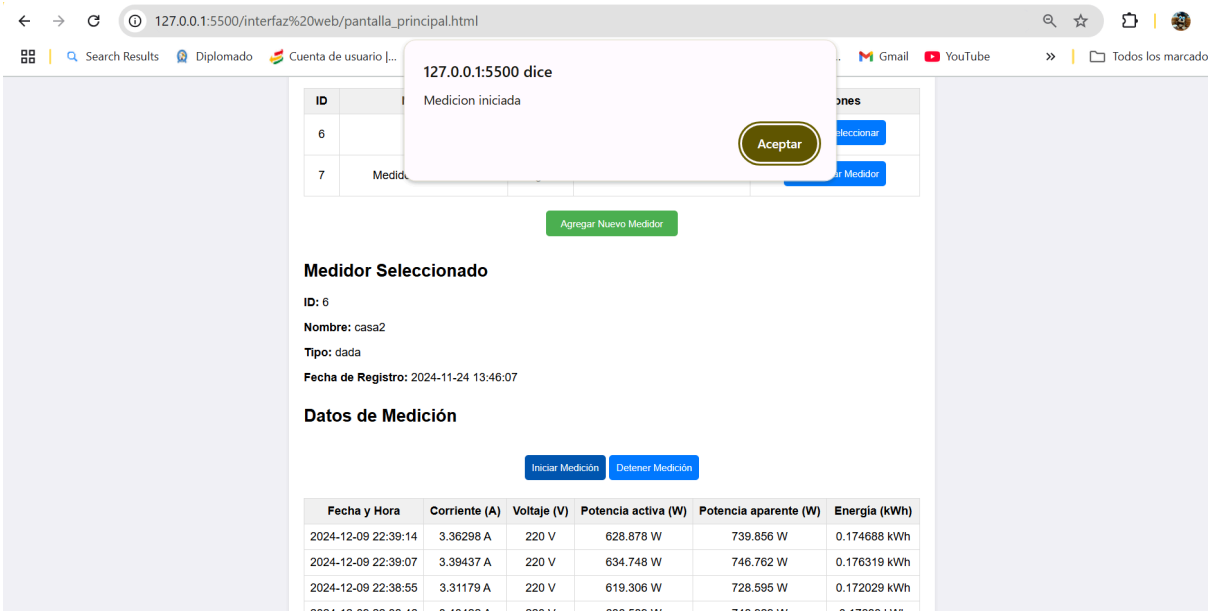
Interfaz en thonny, se muestra una imagen donde se ve el funcionamiento del servidor local creado por el ESP32 y cómo se envían los datos de las mediciones captadas por los sensores.

- Imagen **servidor y sensor funcionando** desde Thonny**:**



Se observa como el ESP32 se conecta al Wifi para luego iniciar un servidor local en la dirección IP 192.168.0.19 que es por la cual acceden los clientes. Se ve en la consola como se conecta un cliente y desde qué dirección además de las solicitudes que éste hace al momento de darle click a iniciar medición. Luego está el procesamiento de la solicitud y los datos captados por el sensor que son enviados al servidor para que la interfaz los recupere y muestre.

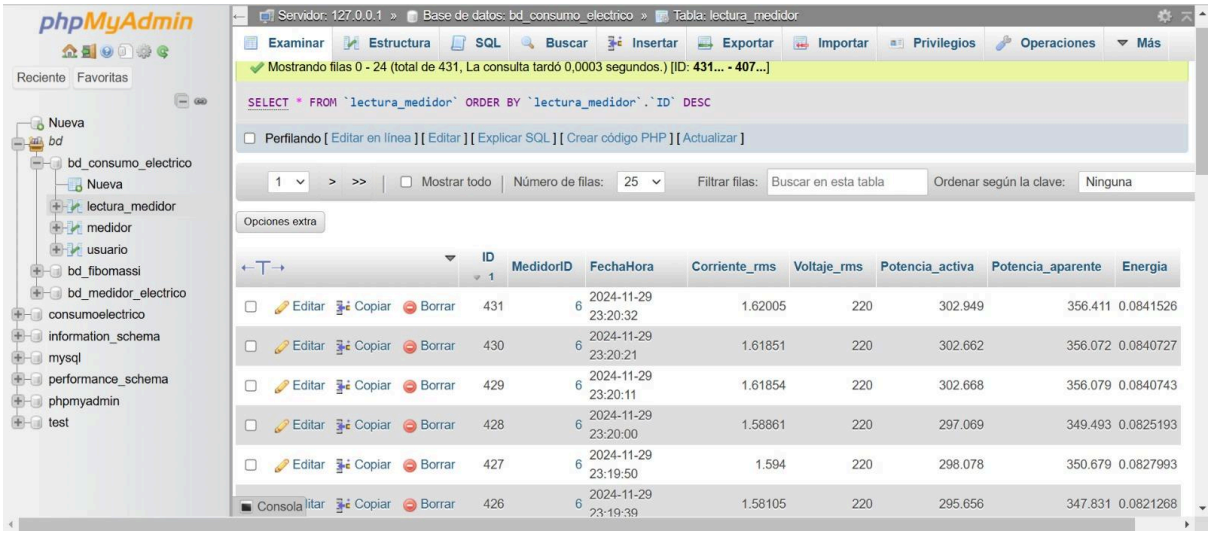
- Imagen **datos recolectados por el sensor** recibidos en la interfaz**:**



Aquí se ve una captura de como desde la interfaz de la aplicación Web se da a iniciar medición y los datos se van mostrando en la tabla de abajo y es la solicitud que recibe el servidor mostrada en la anterior imagen. Aquí se recuperan los datos y se los guarda en la base de datos.

4.1.2. Almacenamiento correcto en la base de datos.

- Imagen **base de datos** desde MySQL xampp:



- Imagen **base de datos** desde php interfaz abm:

Gestión de Usuarios

Agregar Usuario | ID de Usuario | Editar Usuario | Eliminar Usuario | Volver

ID	Nombre	Dirección	Teléfono	Email	Contraseña	Fecha de Registro
1	Juan Pérez	Av. Principal 123	5551234567	juan.perez@gmail.com	hola	2023-01-01 12:00:00
2	Ana López	Calle Secundaria 45	5557654321	ana.lopez@gmail.com		2023-02-01 13:30:00
3	Carlos Ruiz	Av. Reforma 10	5559876543	carlos.ruiz@gmail.com		2023-03-01 14:15:00
4	Lucía Gómez	Calle Estrella 78	5556543210	lucia.gomez@gmail.com		2023-04-01 15:00:00
5	María Torres	Av. Sur 456	5553456789	maria.torres@gmail.com		2023-05-01 16:45:00
6	rene	fwefe	2324	rene@gmail.com	rene123	2024-11-24 17:15:45
7	rene	fef	2323	dwd		2024-11-24 17:19:09
8	Ronald	Calle 1	123457	rere		2024-11-24 17:20:26
10	rene	fefe	4343	sdw		2024-11-24 17:22:58
11	Valdir	Alto Seguencoma	7857483	valdir.flores@gmail.com	val123	2024-11-24 17:23:36
12	Rene	ec	2323	fsf		2024-11-24 17:26:06

ID	Usuario ID	Nombre	Tipo	Fecha de Registro
1	1	Medidor Principal Casa	Residencial	2023-01-10 16:00:00
2	2	Medidor Sotano	Residencial	2023-02-10 17:30:00
3	3	Medidor Oficina	Comercial	2023-03-10 18:15:00
4	4	Medidor Bodega	Industrial	2023-04-10 19:00:00
5	5	Medidor Taller	Comercial	2023-05-10 20:45:00
6	6	casa2	dada	2024-11-24 17:46:07
7	6	Casa4	Manolo	2024-11-28 16:54:15
8	6	casa8	noo	2024-11-29 12:44:45
9	6	Polo	Casa	2024-11-29 13:13:40
13	11	Agora	Analogico	2024-12-09 14:52:17
14	44	Cocian	Dijital	2024-12-09 15:27:11

4.1.3. Dashboard en tiempo real funcionando.

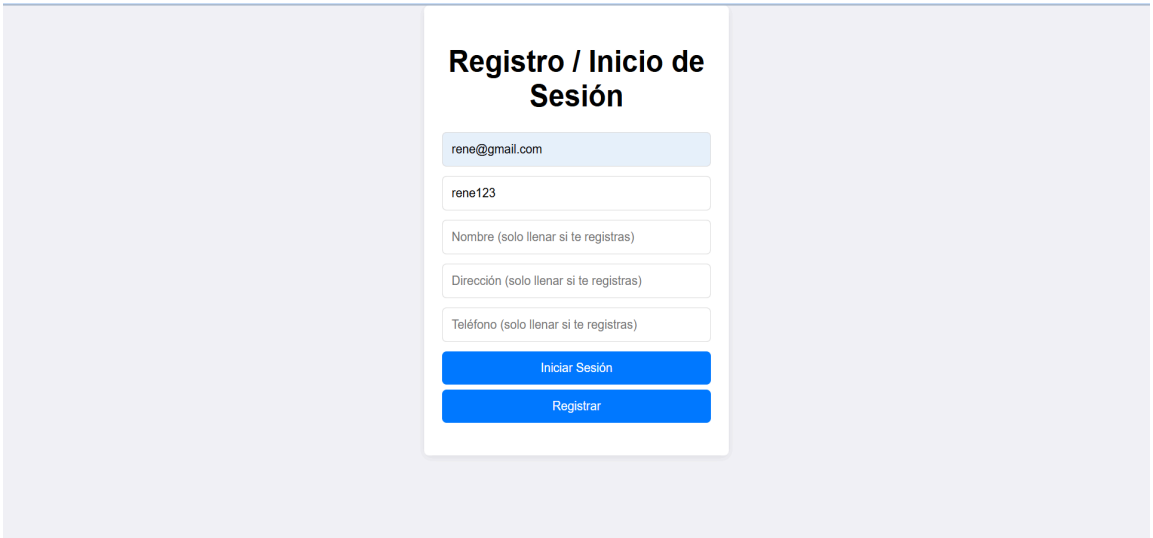
- Imagen **Dashboard funcionando:**

4.1.4. Aporte Ingenieril

Se adjuntan evidencias del aporte ingenieril las cuales son capturas de la interfaz web creada y las funcionalidades implementadas como ser el inicio de sesión y registro de usuarios y demás componentes.

4.1.4.1. Demostración de la Interfaz web creada

4.1.4.1.1. Registro y Inicio de Sesion



Para nuestra interfaz agregamos lo que sería un registro y inicio de sesión, donde los usuarios podrán registrarse y iniciar sesión, siendo que los datos requeridos para su registro serian (Email, Contraseña, Nombre, Dirección, Teléfono) de tal forma que estos mismos se guardarán en nuestra tabla de usuarios de nuestra base de datos, para el inicio de sesión sólo se necesitará el Email y Contraseña. En el caso de los Administradores estos ya cuentan con un registro previo dentro el código, entonces solo se necesitará verificar el Email y Contraseña otorgados.

4.1.4.1.2. Pantalla de Usuario

Ya habiendo iniciado sesión como usuario entonces podremos visualizar lo que sería los datos del usuario que inicio sesion y los medidores que tiene registrados, con la posibilidad de agregar un nuevo medidor si es

necesario, también presenciamos lo que seria el botón del dashboard por usuario (cliente), en el cual tendríamos un dashboard persona mostrando el gasto en Bs que tendría.

4.1.4.1.3. Selector de Medidor

6	casa2	dada	2024-11-24 17:46:07	Dejar de seleccionar

En este punto nuestra interfaz web está a la espera de que seleccionemos un medidor del cual quisiéramos agregar mediciones en tiempo real o visualizar las mediciones anteriores, de tal forma que tendremos el botón de seleccionar el cual al ser presionado cambia de estado a “Dejar de seleccionar”.

4.1.4.1.4. Datos de Medición

Medidor Seleccionado

ID: 6

Nombre: casa2

Tipo: dada

Fecha de Registro: 2024-11-24 17:46:07

Datos de Medición

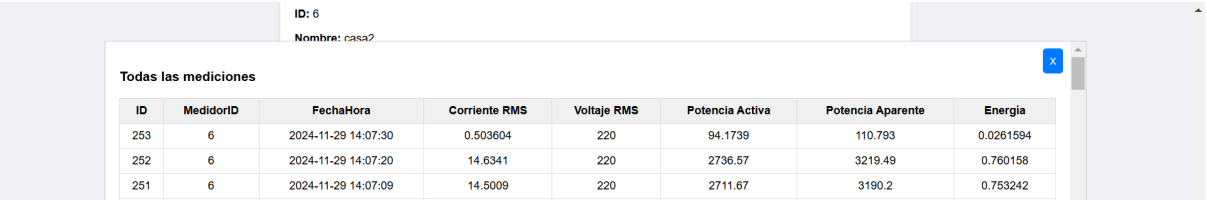
Iniciar Medición | Detener Medición | Ver todas las mediciones

Fecha y Hora	Corriente (A)	Voltaje (V)	Potencia activa (W)	Potencia aparente (W)	Energía (kWh)
2024-11-29 14:07:30	0.503604 A	220 V	94.1739 W	110.793 W	0.0261594 kWh
2024-11-29 14:07:20	14.6341 A	220 V	2736.57 W	3219.49 W	0.760158 kWh
2024-11-29 14:07:09	14.5009 A	220 V	2711.67 W	3190.2 W	0.753242 kWh
2024-11-29 14:06:59	14.5443 A	220 V	2719.79 W	3199.75 W	0.755498 kWh
2024-11-29 14:06:48	1.29446 A	220 V	242.064 W	284.782 W	0.0672401 kWh
2024-11-29 14:06:37	1.46095 A	220 V	273.198 W	321.41 W	0.0758884 kWh
2024-11-29 14:06:27	1.17922 A	220 V	220.514 W	259.428 W	0.0612538 kWh
2024-11-29 14:06:16	1.22367 A	220 V	228.826 W	269.207 W	0.0635628 kWh
2024-11-29 14:06:06	1.21725 A	220 V	227.626 W	267.795 W	0.0632294 kWh
2024-11-29 14:05:55	1.35273 A	220 V	252.96 W	297.599 W	0.0702665 kWh

Visualización Grafica del Consumo

Ya habiendo seleccionado el medidor esté mostrará una sección que trata del medidor seleccionado, donde primero se mostrarán los datos del medidor seleccionado como ID, Nombre, Tipo y Fecha de Registro del medidor. Después tendremos los Datos de Medición, en esta apartado mostrará los últimos 10 mediciones que se hicieron en este medidor, no obstante también tendremos el botón de “Ver todas las mediciones”.

4.1.4.1.5. Ventana de todas las Mediciones

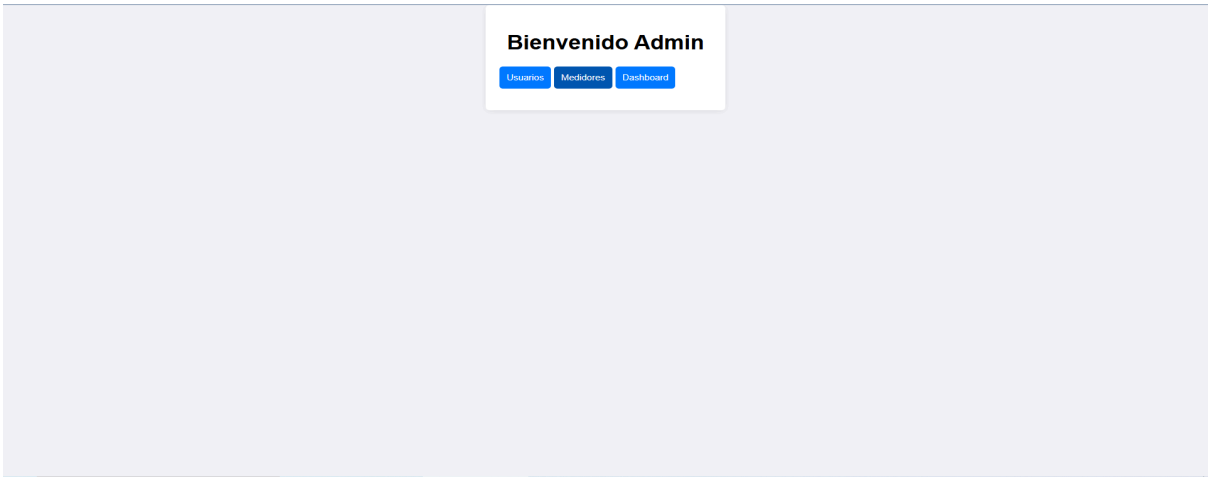


Ya habiendo presionado el botón de “Ver todas las Mediciones” se abrirá una ventana la cual de igual forma mostrará las mediciones del medidor pero de forma completa, desde la primera medición que se realizó hasta la última medición.

4.1.4.1.6. Iniciar y Detener Medición en tiempo real

También tendremos en “Datos de Medición” lo que serían los botones para Iniciar Medición y Detener Medición, estos ayudaran a tomar las mediciones en tiempo real, siendo que se mostraran en pantalla cada 10 segundos.

4.1.4.1.7. Pantalla de Administrador



Se puede visualizar la pantalla de Administrador en la que después de verificar nuestras credenciales tendremos lo que sería nuestros 2 ABM de “Usuarios” y “Medidores” de tal forma que podremos realizar nuestras altas, bajas y modificaciones, de todos los datos de la base de datos. Por último se tendrá el dashboard el cual mostrará un resumen gráfico por usuario al administrador. Todos los botones al ser presionados serán llevados a sus respectivas pantallas en las que se cumpliran con la función ya dicha.

4.2. Análisis de los resultados:

4.2.1. ¿Cumplió el sistema con los objetivos planteados?

Sí, el sistema cumplió con los objetivos planteados, tanto a nivel técnico como funcional, alcanzando los siguientes logros:

1. Monitoreo del consumo eléctrico en tiempo real
El sistema proporciona datos precisos de parámetros eléctricos clave como corriente, voltaje, potencia activa y energía consumida, gracias a la correcta integración de los sensores ACS712 con el ESP32. Estos datos fueron enviados al servidor y almacenados en una base de datos MySQL de forma eficiente, lo que permitió su análisis posterior.

2. Visualización accesible y comprensible
El dashboard interactivo desarrollado con Dash y Plotly cumplió con el objetivo de proporcionar una visualización clara y amigable. Los usuarios pudieron observar gráficas en tiempo real de los datos de consumo, identificar patrones y tomar decisiones informadas para optimizar su uso de energía.

3. Gestión de datos efectiva
La estructura de la base de datos permitió manejar adecuadamente los registros de usuarios, medidores y lecturas eléctricas. Las funcionalidades de ABM (Alta, Baja, Modificación) se implementaron con éxito, garantizando la correcta administración de los datos.

4. Conexión y estabilidad del sistema IoT
El ESP32, configurado con scripts en MicroPython, mantuvo una conexión estable con la red Wi-Fi y el servidor. Además, se implementaron medidas para gestionar interrupciones, como reconexión automática y manejo de excepciones, asegurando la continuidad del sistema.

5. Escalabilidad del sistema
El diseño modular del sistema lo hace escalable para incorporar más sensores, usuarios o funcionalidades en el futuro, cumpliendo con el objetivo de ser adaptable a contextos más complejos, como edificios o comunidades.

4.2.2. Problemas encontrados y cómo se resolvieron.

A lo largo del proyecto, se enfrentaron diversas dificultades y problemas. Uno de los principales fue la imprecisión inicial en las mediciones, ya que el sensor utilizado era de 30 A y el circuito de prueba solo incluía focos. Por más focos que se conectaran, no se observaban cambios significativos. Para solucionar esto, se optó por utilizar dispositivos de mayor consumo, como una plancha, para evidenciar diferencias más notorias. Sin embargo, antes de realizar estas pruebas, se modificó el circuito original. Se incorporó un alargador con tres tomacorrientes, lo que permitió conectar dispositivos directamente sin necesidad de dividir sus cables.

Además, se realizaron investigaciones para configurar adecuadamente el sensor. Se utilizaron pines analógicos, como el GPIO34, y se ajustó el valor de sensibilidad a 0.066 según el modelo del sensor. Con ayuda de un multímetro, se determinó el valor de **V_Zero**, que representa el voltaje

cuando no hay carga conectada al circuito. Para reducir el ruido entre las mediciones, se tomaron 1250 lecturas, cuyos valores fueron promediados.

De esta forma, al configurar correctamente el sensor y modificar el circuito para facilitar la conexión de diversos dispositivos, se logró abordar los problemas de medición, haciéndola más precisa y eficiente.

4.2.3. Posibles mejoras.

Precisión en las mediciones:

- Reemplazar el sensor ACS712 con una versión de mayor precisión o compensar el error mediante algoritmos de calibración.
- Implementar un sistema de auto-calibración para mantener la exactitud del sistema a lo largo del tiempo.
- Se podría implementar el sensor ZMPT101B el cual sirve para calcular el voltaje, esto dándonos una mayor precisión.

Optimización del dashboard:

- Incluir más filtros avanzados, como selección de periodos personalizados o comparación entre múltiples días.
- Mejorar la visualización con gráficos más intuitivos, como diagramas de barras o mapas de calor para identificar patrones de consumo.

Seguridad de datos:

- Implementar cifrado en la transmisión de datos entre la aplicación web y el servidor local iniciado en el ESP32.
- Proteger la base de datos con autenticación más robusta.

5. Conclusiones

5.1. Resumen del proyecto.

El proyecto desarrollado consistió en la implementación de un sistema IoT para el monitoreo en tiempo real del consumo eléctrico de un circuito, utilizando un ESP32 como microcontrolador y un sensor ACS712 para la medición de corriente eléctrica. Los datos recolectados incluyen parámetros como corriente RMS, voltaje, potencia activa, potencia aparente y energía consumida, los cuales se procesan y almacenan en una base de datos MySQL en Xampp. La comunicación entre los componentes del sistema se estableció utilizando el protocolo HTTP, lo que permitió una integración fluida entre el hardware y la interfaz web desarrollada en PHP.
La interfaz web incluye un sistema de login que permite a los usuarios acceder a un dashboard personalizado, donde pueden visualizar sus patrones de consumo eléctrico a través de gráficos dinámicos generados con Chart.js. Los datos se actualizan cada 10 segundos, lo que garantiza información en tiempo real para la toma de decisiones. Adicionalmente, los administradores cuentan con un dashboard y una ABM específico que les

permite gestionar usuarios y medidores. Esta separación de roles asegura un manejo eficiente y seguro de los datos dentro del sistema.
El sistema también brinda la posibilidad de registrar múltiples medidores asociados a un usuario, lo que lo hace ideal para hogares con diferentes dispositivos o pequeños negocios con varios puntos de consumo. La capacidad de consultar tanto las lecturas más recientes como el historial completo de mediciones agrega un nivel de profundidad al análisis del consumo eléctrico.
Este proyecto, realizado en equipo, integró hardware, software y protocolos de comunicación de manera eficiente, logrando una solución funcional y escalable que puede ser aplicada en diversos contextos residenciales y comerciales.

5.2. Reflexión sobre las habilidades adquiridas.

Durante el desarrollo del proyecto, el trabajo en equipo jugó un papel fundamental, permitiéndonos complementar nuestras habilidades y aprender unos de otros. En términos técnicos, adquirimos conocimientos prácticos en la programación del ESP32 mediante MicroPython, aprendiendo a calcular parámetros eléctricos clave y enviar estos datos al servidor mediante el protocolo HTTP. También desarrollamos habilidades en el manejo y calibración del sensor ACS712, ajustando valores para asegurar mediciones precisas y confiables.
En el ámbito de desarrollo web, nos fortalecimos en la creación de interfaces dinámicas utilizando PHP y Chart.js. El diseño de un sistema de login con roles diferenciados para usuarios y administradores nos permitió entender la importancia de la seguridad y personalización en aplicaciones web. Además, la implementación del módulo ABM para los administradores, que les permite gestionar usuarios y medidores, añadió un nivel de complejidad que mejoró nuestras capacidades en gestión de bases de datos y desarrollo backend.
Otro aspecto clave fue la actualización de los datos en tiempo real cada 10 segundos, lo que nos exigió optimizar la comunicación entre el ESP32 y el servidor, así como implementar métodos eficientes para el manejo de grandes volúmenes de datos. Este desafío nos permitió mejorar nuestras habilidades en procesamiento y visualización de datos, asegurando que el dashboard ofreciera una experiencia fluida y comprensible.
Finalmente, el trabajo en grupo nos enseñó a coordinar tareas, gestionar el tiempo y resolver problemas de manera colaborativa. Cada integrante tuvo un rol específico, y al combinar nuestros esfuerzos logramos superar obstáculos como la calibración del sensor, la integración de módulos ABM y la optimización de la interfaz gráfica. Este proyecto no solo reforzó nuestras competencias técnicas, sino también nuestras habilidades blandas, esenciales para el éxito en proyectos multidisciplinarios.

5.3. Impacto del proyecto IoT en su contexto.

El sistema desarrollado tiene un impacto significativo en la gestión del consumo energético, permitiendo a los usuarios tomar decisiones informadas basadas en datos en tiempo real. Los usuarios pueden registrar múltiples

medidores, consultar lecturas históricas y observar gráficos interactivos, lo que facilita la identificación de hábitos ineficientes y promueve el ahorro energético.

El sistema de roles diferenciados entre usuarios y administradores añade una capa de funcionalidad que lo hace más completo y adaptable. Los administradores pueden gestionar los datos de usuarios y medidores desde su propio AMB, además de observar comportamiento desde un dashboard específico, asegurando un control centralizado del sistema. Esta capacidad es especialmente útil en contextos donde se requiere un monitoreo y administración más detallados, como en edificios residenciales o pequeños negocios.

La actualización de los datos cada 10 segundos asegura que la información presentada sea precisa y oportuna, mejorando la experiencia del usuario y aumentando la utilidad del sistema en la toma de decisiones. El uso de un protocolo HTTP, combinado con herramientas como Chart.js, permitió la creación de un sistema eficiente y visualmente atractivo, que puede ser comprendido incluso por usuarios sin conocimientos técnicos avanzados.

En un contexto más amplio, este proyecto demuestra cómo la integración de tecnologías IoT con plataformas web puede abordar problemas cotidianos, como el monitoreo y optimización del consumo energético, de manera práctica y accesible. Su diseño modular y escalable lo hace adaptable a otras aplicaciones, como sistemas industriales o edificios inteligentes, posicionando una solución flexible para necesidades futuras. Este trabajo no solo destaca por su implementación técnica, sino también por su impacto en la sostenibilidad y el manejo eficiente de recursos.

6. Bibliografía:

- Programar Fácil. (s. f.). *Guía completa sobre ESP32: Qué es, para qué sirve y cómo utilizarlo*. Programar Fácil. Recuperado el 8 de diciembre de 2024, de https://programarfacil.com/esp8266/esp32/
- Naylamp Mechatronics. (s. f.). *Tutorial: Sensor de corriente ACS712*. Naylamp Mechatronics. Recuperado el 8 de diciembre de 2024, de https://naylampmechatronics.com/blog/48_tutorial-sensor-de-corriente-acs712.html

7. Anexos:

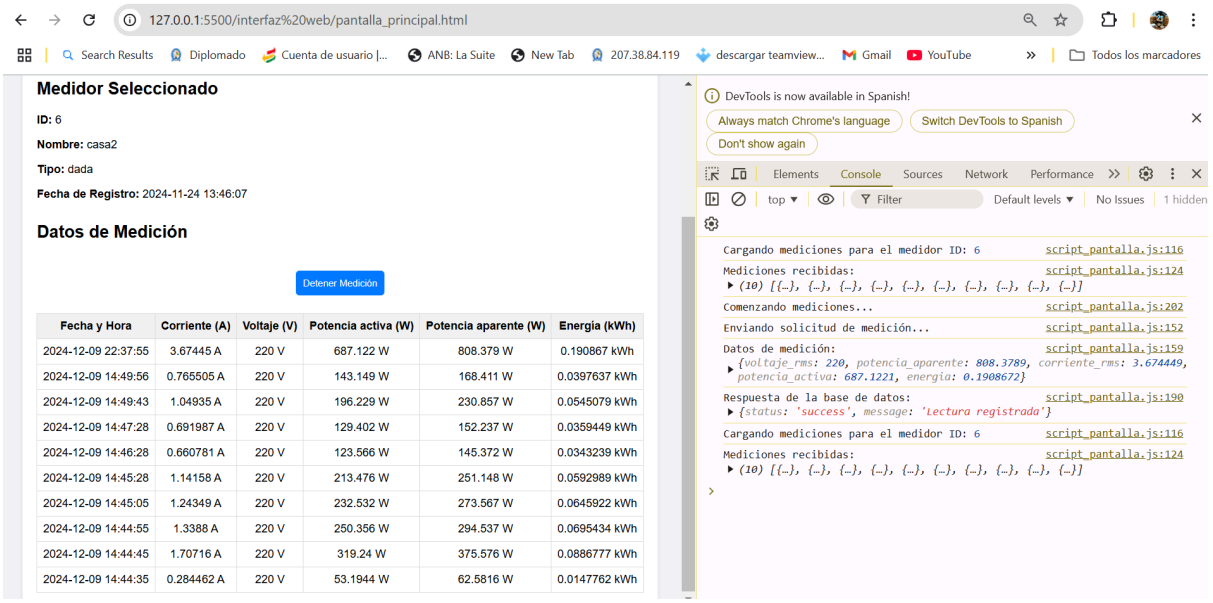
7.1. Código completo del proyecto.

Como se realizaron varios archivos con códigos de distintos lenguajes de programación como html, javascript, css, php y python, se subió el proyecto completo a un repositorio en github con visibilidad pública para observar todos los códigos de manera más ordenada y entendible. El link del repositorio es el siguiente y se encuentra en la rama “main”:

https://github.com/Roundalz/Consumo-Electrico

7.2. Capturas de comandos o logs de ejecución.

Imagen **de los logs mostrados en la consola** desde Interfaz Web**:**



En la imagen se observan los mensajes de los procesos que se realizan al momento de seleccionar un medidor y comenzar una medición. Se muestran las solicitudes enviadas, los datos recibidos y las respuestas de la base de datos para tener más información de estos procesos y conocer el error por si lo hay.