



UNIVERSIDAD
CATÓLICA
BOLIVIANA

INFORME
TERCERA EVALUACIÓN

ASIGNATURA:
INTERNET DE LAS COSAS

Grupo: Grupo 5

INTEGRANTES
Alejandro Mollinedo Rodriguez
Ronald 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: 22/11/2024

LA PAZ – BOLIVIA

2024

Índice

1. Introducción:	4
1.1. Contexto del proyecto	4
1.2. Descripción breve del problema que aborda el sistema IoT	4
1.3. Objetivo general del proyecto	4
1.4. Objetivos específicos	4
2. Marco Teórico	5
2.1. Explicación de conceptos esenciales	5
2.1.1. IoT: Principios básicos, ventajas y aplicaciones	5
2.1.2. Microcontrolador seleccionado (ESP32): Capacidades técnicas y ventajas	5
2.1.3. Protocolos de comunicación utilizados	5
2.1.4. Sensores utilizados	5
3. Desarrollo del Proyecto	6
3.1. Diseño del Circuito	6
3.1.1. Diagrama esquemático con software (Fritzing, TinkerCAD, o similar)	6
3.1.2. Conexiones entre microcontroladores y sensores	6
3.1.3. Justificación del protocolo de comunicación utilizado	6
3.1.4. Fotografía del circuito físico (si aplica)	6
3.2. Diseño de la Red IoT	6
3.2.1. Diagrama de la red con conexiones entre nodos IoT, servidor y clientes	6
3.2.2. Explicación de los roles de cada dispositivo (nodos, servidor, clientes)	6
3.2.3. Selección de protocolo de comunicación	6
3.3. Diseño de la Base de Datos	6
3.3.1. Estructura de la tabla principal, con descripción detallada de los campos	6
3.3.2. Si hay más tablas, incluir un diagrama entidad-relación	6
3.3.3. Script SQL para crear las tablas	6
3.4. Desarrollo de Scripts	6
3.4.1. Script PHP para insertar y leer datos desde la base de datos	6
3.4.2. Script en MicroPython para el envío de datos desde los nodos IoT	6
3.4.3. Script en Python para ABM en la base de datos	6
3.4.4. Script en Python para el dashboard	6
3.4.5. Explicación y comentarios en cada script	6
4. Resultados y Análisis	6
4.1. Evidencias del sistema funcionando	6
4.1.1. Datos recolectados por los sensores	7
4.1.2. Almacenamiento correcto en la base de datos	7
4.1.3. Dashboard en tiempo real funcionando	7
4.2. Análisis de los resultados	7
4.2.1. ¿Cumplió el sistema con los objetivos planteados?	7
4.2.2. Problemas encontrados y cómo se resolvieron	7
4.2.3. Posibles mejoras	7
5. Conclusiones	7
5.1. Resumen del proyecto	7
5.2. Reflexión sobre las habilidades adquiridas	7
5.3. Impacto del proyecto IoT en su contexto	7

6. Bibliografía:	7
7. Anexos:	7
7.1. Código completo de los scripts	7
7.2. Capturas de comandos o logs de ejecución	7
7.3. Diagramas complementarios	7

1. Introducción:

1.1. Contexto del proyecto

La medición del consumo eléctrico en un circuito o 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 e inteligentes sobre el consumo eléctrico a través de un microcontrolador ESP32. El sistema utiliza sensores de corriente ACS712 para medir el consumo eléctrico de dos clientes simulados mediante focos conectados al circuito. 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 dos clientes simulados, este proyecto busca demostrar cómo la tecnología IoT puede aplicarse para mejorar la eficiencia energética en entornos residenciales.

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.

Implementación de un prototipo de sistema IoT para la medición del consumo de energía eléctrica en una vivienda.

1.4. Objetivos específicos

- Diseñar y estructurar la base de datos para almacenar los datos de consumo de energía eléctrica de manera organizada y accesible.
- Implementar un script en Micro Python para capturar y enviar los datos de consumo desde el dispositivo IoT a la base de datos en tiempo real
- Crear un dashboard en Python para visualizar en tiempo real los datos de consumo de energía eléctrica.
- Desarrollar un script en Python para el servidor que permita gestionar el alta, baja y modificación (ABM) de los registros de consumo 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.

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, TinkerCAD, o similar).



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:

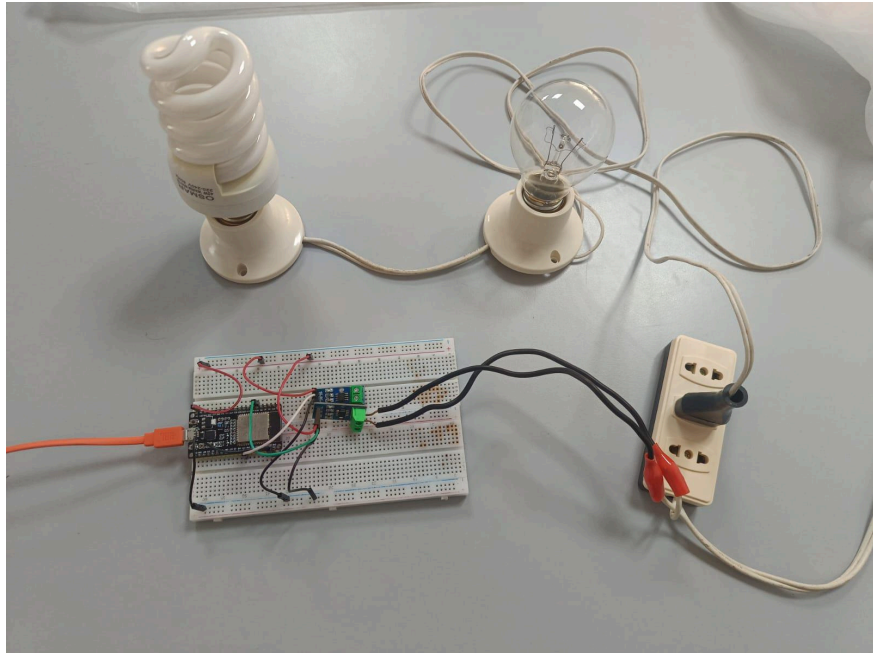
-
- ACS712 VCC → ESP32 3.3V o 5V: Alimenta el sensor.
- ACS712 GND → ESP32 GND: Establece el circuito de referencia.
- ACS712 OUT → ESP32 ADC: Lee el voltaje de salida para calcular la corriente

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.



3.2. Diseño de la Red IoT

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

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

Rol de los nodos

En el sistema que estás desarrollando, 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 y un ESP32, que actúa como puente entre el sensor y el resto del sistema. El sensor mide continuamente la energía consumida, mientras que el ESP32 procesa estos datos (si es necesario) y los envía al servidor mediante una conexión a Internet, como Wi-Fi. 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. En tu proyecto, este rol lo cumple un sistema desarrollado en PHP que incluye una base de datos para registrar el gasto de energía capturado por el ESP32. 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. 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 no generan datos ni los procesan directamente, pero permiten que los usuarios 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 las personas.

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

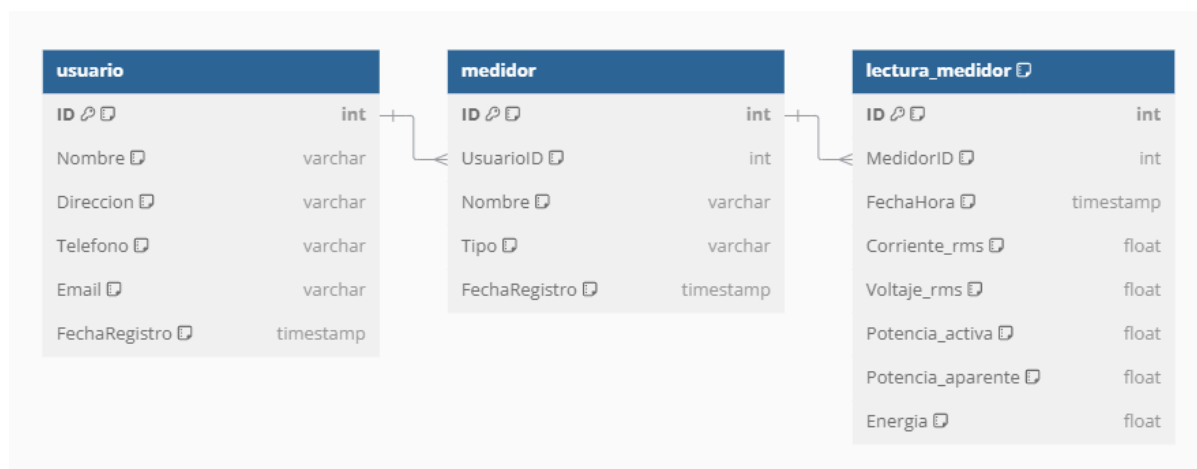
3.3.1. Estructura de la tabla principal, con descripción detallada de los campos.

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.

```

-- 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,
  `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 **EsplInsercionDatosMedidor.py** se ejecuta en el ESP32 y se encarga de 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 se envían al servidor mediante peticiones HTTP POST.

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

```
<?php
$host = "localhost";
$user = "root";
$password = "";
$dbname = "bd_consumo_electrico";

$conn = new mysqli($host, $user, $password, $dbname);

if ($conn->connect_error) {
    die("Conexión fallida: " . $conn->connect_error);
}

$sql = "SELECT * FROM usuario";
$result = $conn->query($sql);
?>

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Gestión de Usuarios</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css">
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/js/bootstrap.bundle.min.js"></s
cript>
</head>
<body>
    <div class="container mt-5">
        <h1 class="text-center">Gestión de Usuarios</h1>
        <div class="d-flex justify-content-end mb-3">
            <button class="btn btn-primary me-2" onclick="location.href='agregar.php'">Agregar
Usuario</button>
            <input type="number" id="idSeleccionado" class="form-control w-25 me-2"
placeholder="ID de Usuario" />
            <button class="btn btn-warning me-2" onclick="editarUsuario()">Editar
Usuario</button>
            <button class="btn btn-danger" onclick="eliminarUsuario()">Eliminar
Usuario</button>
        </div>
    </div>
</body>
</html>
```

```

</div>
<table class="table table-striped">
  <thead>
    <tr>
      <th>ID</th>
      <th>Nombre</th>
      <th>Dirección</th>
      <th>Teléfono</th>
      <th>Email</th>
      <th>Fecha de Registro</th>
    </tr>
  </thead>
  <tbody>
    <?php
    if ($result->num_rows > 0) {
      while ($row = $result->fetch_assoc()) {
        echo "<tr>
          <td>{$row['ID']}</td>
          <td>{$row['Nombre']}</td>
          <td>{$row['Direccion']}</td>
          <td>{$row['Telefono']}</td>
          <td>{$row['Email']}</td>
          <td>{$row['FechaRegistro']}</td>
        </tr>";
      }
    } else {
      echo "<tr><td colspan='6' class='text-center'>No hay usuarios
registrados</td></tr>";
    }
    ?>
  </tbody>
</table>
</div>

```

```

<script>
function editarUsuario() {
  const id = document.getElementById('idSeleccionado').value;
  if (id) {
    window.location.href = `editar.php?id=${id}`;
  } else {
    alert("Ingrese el ID del usuario que desea editar.");
  }
}

function eliminarUsuario() {
  const id = document.getElementById('idSeleccionado').value;
  if (id && confirm("¿Está seguro de eliminar este usuario?")) {
    window.location.href = `eliminar.php?id=${id}`;
  }
}

```

```

        } else if (!id) {
            alert("Ingrese el ID del usuario que desea eliminar.");
        }
    }
</script>
</body>
</html>

<?php
$conn->close();
?>

```

3.4.4. Script en PHP para el dashboard.

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dashboard</title>
    <style>
        body {
            margin: 0;
            padding: 0;
            font-family: 'Poppins', sans-serif;
            background-color: #f4f4f9;
            color: #333;
            display: flex;
            flex-direction: column;
            height: 100vh;
        }

        .header {
            background-color: #4CAF50;
            color: white;
            text-align: center;
            padding: 15px 0;
            font-size: 24px;
            font-weight: 600;
            box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
        }

        .tableau-container {
            flex: 1;
            display: flex;
            justify-content: center;

```

```

        align-items: center;
        padding: 10px;
        box-sizing: border-box;
    }

    .tableauPlaceholder {
        width: 100%;
        max-width: 1200px;
        height: 90%; /* Ocupa el 90% de la altura disponible */
        border: 1px solid #ddd;
        box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
        border-radius: 8px;
    }

    button {
        margin: 20px;
        padding: 10px 20px;
        background-color: #4CAF50;
        color: white;
        border: none;
        border-radius: 5px;
        cursor: pointer;
    }
</style>
<link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;600&display=swap"
rel="stylesheet">
</head>
<body>
    <div class="header">
        Dashboard medidor de energía
    </div>

    <button id="refreshButton">Actualizar Dashboard</button>

    <div class="tableau-container">
        <div class='tableauPlaceholder' id='viz1732298499950'></div>
    </div>

    <script type='text/javascript'
src="https://public.tableau.com/javascripts/api/tableau-2.min.js"></script>
    <script type='text/javascript'>
        var viz;

        function initViz() {
            var containerDiv = document.getElementById("viz1732298499950");
            var options = {
                hideTabs: true,

```



```

        onFirstInteractive: function () {
            console.log("Tableau está listo");
        }
    };
    var url = "https://public.tableau.com/views/Libro1_17322806692600/Historia1";
    viz = new tableau.Viz(containerDiv, url, options);
}

function refreshViz() {
    if (viz) {
        viz.refreshDataAsync().then(function () {
            console.log("Dashboard actualizado");
        }).catch(function (error) {
            console.error("Error al actualizar el dashboard: ", error);
        });
    }
}

document.addEventListener("DOMContentLoaded", initViz);
document.getElementById("refreshButton").addEventListener("click", refreshViz);
</script>
</body>
</html>

```

3.4.5. Explicación y comentarios en cada script.

- El script **ManejoBaseDeDatos.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;
    }
}
```

```

// 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 {
            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 **EsplInsercionDatosMedidor.py**:

```

from machine import Pin, ADC
import time
import math
import ujson
import urequests as requests
from Wifi_lib import wifi_init, get_html # Librerías externas

# Configuración de Wi-Fi
if wifi_init():
    print("Conexión Wi-Fi exitosa")
else:
    print("Error al conectar a Wi-Fi")

# 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

# URL del servidor
url = "http://192.168.0.9/ManejoBaseDeDatos.php"

# Cálculo de corriente RMS
def calcular_corriente_rms(samples=1250, sampling_time=0.0008):
    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)

# Enviar datos al servidor
def enviar_datos(url, data, headers):
    intentos = 3
    for intento in range(intentos):
        try:
            response = requests.post(url, data=ujson.dumps(data), headers=headers)
            if response.status_code == 200:
                print("Datos enviados:", data)
                print("Respuesta del servidor:", response.text)
                response.close()
                return True
            else:
                print("Error en la respuesta del servidor:", response.status_code)
        except Exception as e:
            print(f"Error en el envío (intento {intento+1}/{intentos}):", e)
            time.sleep(5)
    return False

# Bucle principal
while True:
    try:
        corriente_rms = calcular_corriente_rms()
        voltaje_rms = 220.0
        potencia_aparente = voltaje_rms * corriente_rms
        potencia_activa = potencia_aparente * 0.85
        energia = potencia_activa * (1 / 3600)

        data = {
            "action": "add_lectura",
            "MedidorID": 6,
            "Corriente_rms": corriente_rms,
            "Voltaje_rms": voltaje_rms,
            "Potencia_activa": potencia_activa,
            "Potencia_aparente": potencia_aparente,
            "Energia": energia,
        }

        headers = {'Content-Type': 'application/json'}
        enviar_datos(url, data, headers)

    except Exception as e:
        print("Error en la ejecución:", e)
        time.sleep(5)

    time.sleep(10)

```

- hols

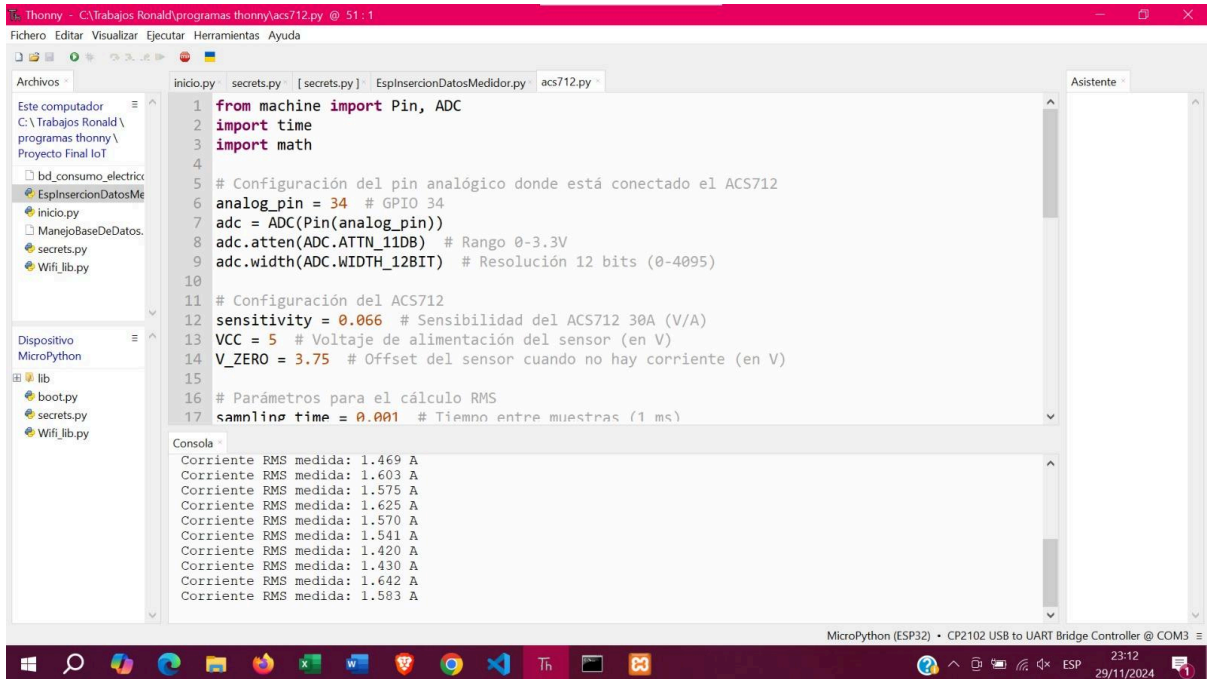
4. Resultados y Análisis

4.1. Evidencias del sistema funcionando:

4.1.1. Datos recolectados por los sensores.

Interfaz en thonny, se mostrara 2 imagenes donde se podrá apreciar el cambio en los datos.

- Imagen sensores funcionando desde Thonny:



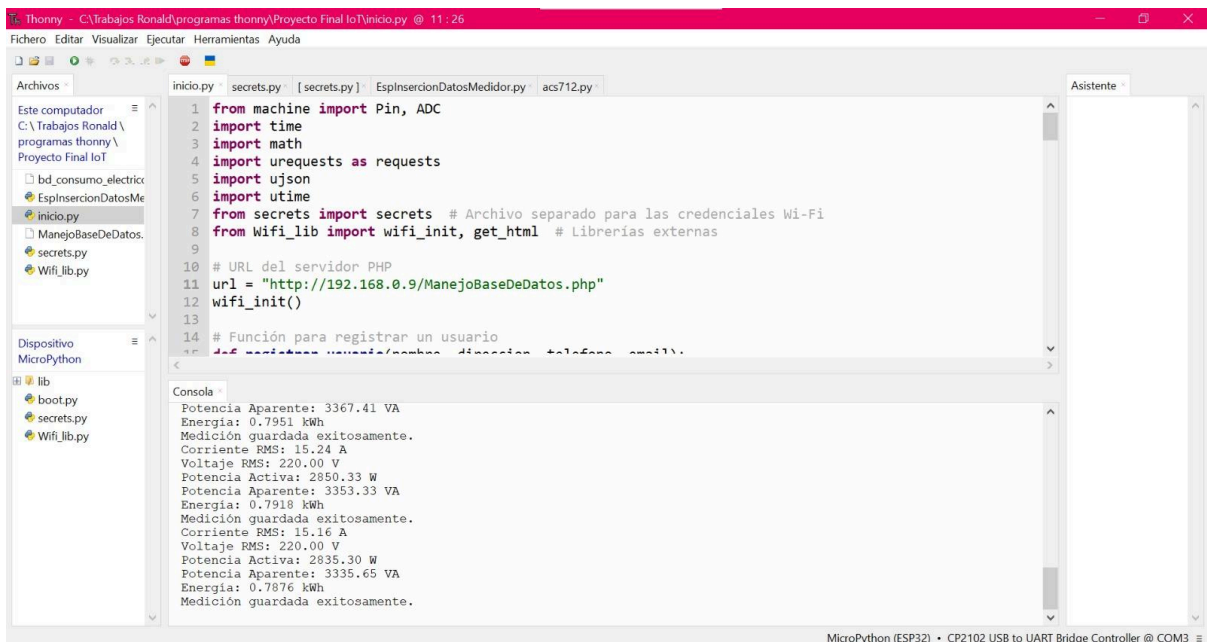
```
1 from machine import Pin, ADC
2 import time
3 import math
4
5 # Configuración del pin analógico donde está conectado el ACS712
6 analog_pin = 34 # GPIO 34
7 adc = ADC(Pin(analog_pin))
8 adc.atten(ADC.ATTN_11DB) # Rango 0-3.3V
9 adc.width(ADC.WIDTH_12BIT) # Resolución 12 bits (0-4095)
10
11 # Configuración del ACS712
12 sensitivity = 0.066 # Sensibilidad del ACS712 30A (V/A)
13 VCC = 5 # Voltaje de alimentación del sensor (en V)
14 V_ZERO = 3.75 # Offset del sensor cuando no hay corriente (en V)
15
16 # Parámetros para el cálculo RMS
17 sampling_time = 0.001 # Tiempo entre muestras (1 ms)
```

Consola

```
Corriente RMS medida: 1.469 A
Corriente RMS medida: 1.603 A
Corriente RMS medida: 1.575 A
Corriente RMS medida: 1.625 A
Corriente RMS medida: 1.570 A
Corriente RMS medida: 1.541 A
Corriente RMS medida: 1.420 A
Corriente RMS medida: 1.430 A
Corriente RMS medida: 1.642 A
Corriente RMS medida: 1.583 A
```

MicroPython (ESP32) • CP2102 USB to UART Bridge Controller @ COM3

- Imagen sensores funcionando desde Thonny despues de conectar la plancha:



```
1 from machine import Pin, ADC
2 import time
3 import math
4 import urequests as requests
5 import ujson
6 import utime
7 from secrets import secrets # Archivo separado para las credenciales Wi-Fi
8 from wifi_lib import wifi_init, get_html # Librerías externas
9
10 # URL del servidor PHP
11 url = "http://192.168.0.9/ManejoBaseDeDatos.php"
12 wifi_init()
13
14 # Función para registrar un usuario
15 def registrar_usuario(nombre, direccion, telefono, email):
```

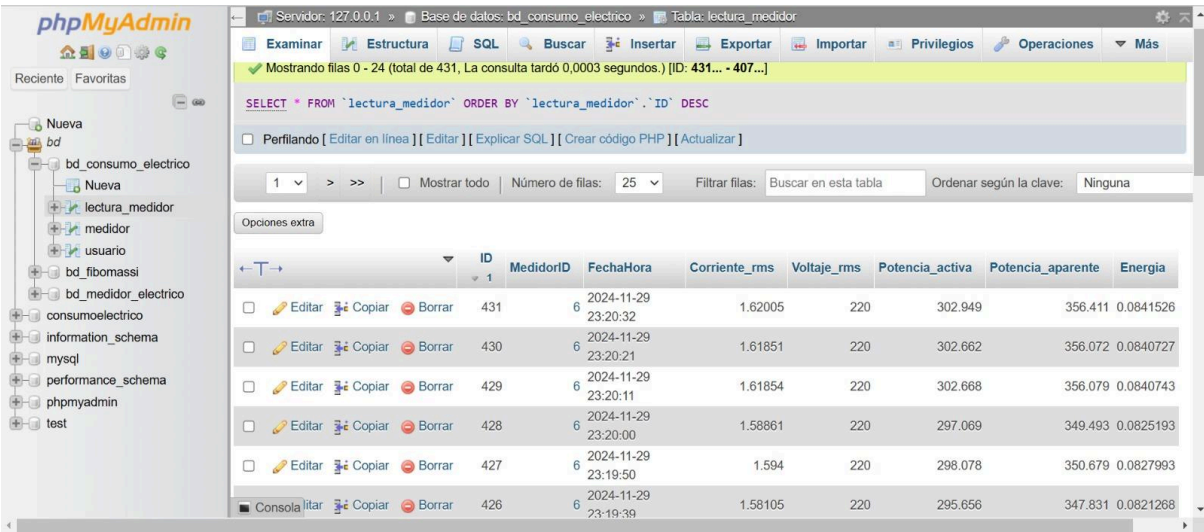
Consola

```
Potencia Aparente: 3367.41 VA
Energia: 0.7951 kWh
Medición guardada exitosamente.
Corriente RMS: 15.24 A
Voltaje RMS: 220.00 V
Potencia Activa: 2850.33 W
Potencia Aparente: 3353.33 VA
Energia: 0.7918 kWh
Medición guardada exitosamente.
Corriente RMS: 15.16 A
Voltaje RMS: 220.00 V
Potencia Activa: 2835.30 W
Potencia Aparente: 3335.65 VA
Energia: 0.7876 kWh
Medición guardada exitosamente.
```

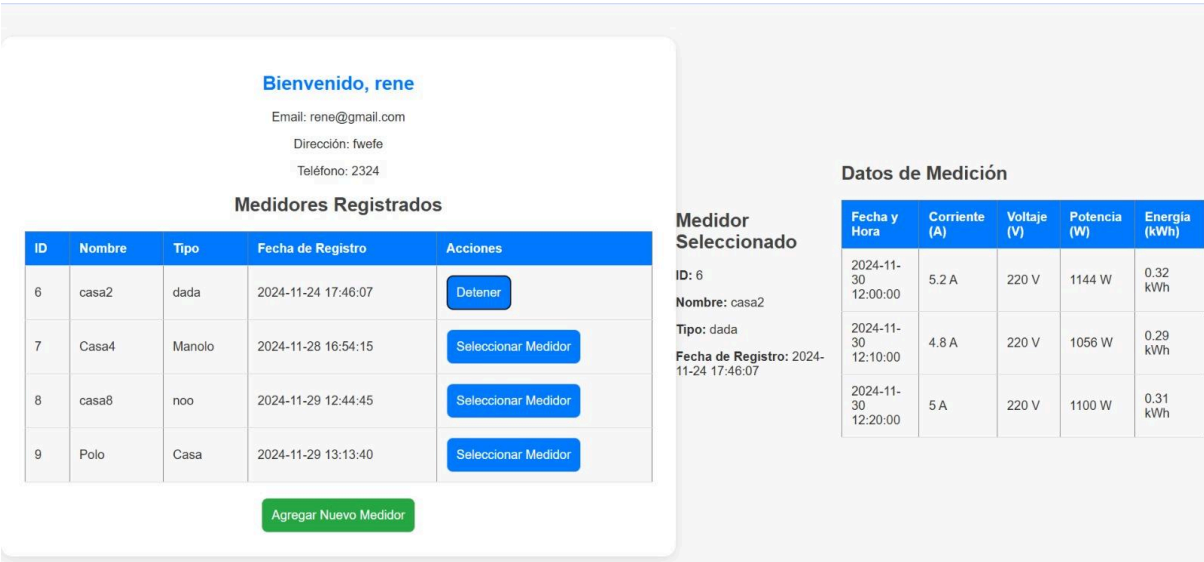
MicroPython (ESP32) • CP2102 USB to UART Bridge Controller @ COM3

4.1.2. Almacenamiento correcto en la base de datos.

- Imagen base de datos desde MySQL xampp:



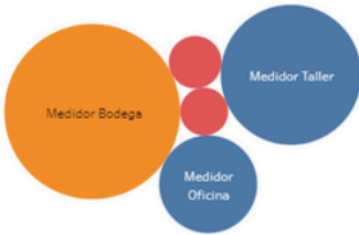
- Imagen base de datos desde php interfaz abm:



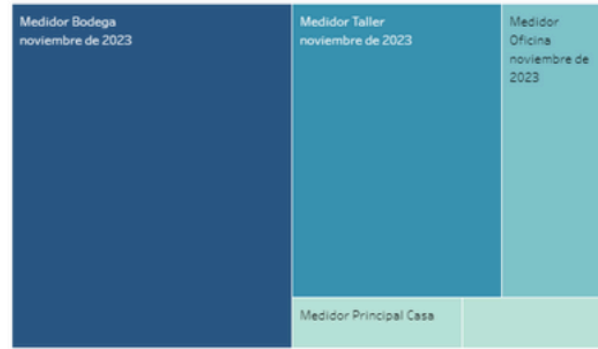
4.1.3. Dashboard en tiempo real funcionando.

- Imagen Dashboard funcionando:

Comparación por Tipo de Usuario



Costo total por medidor

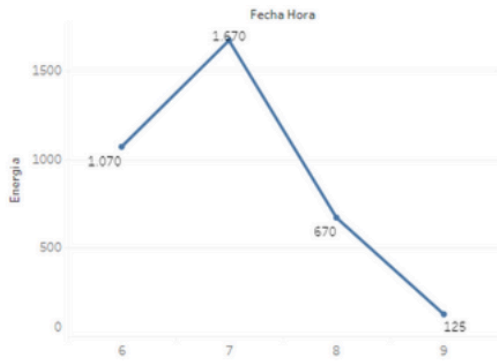


Tipo
☒ Comercial
☐ Industrial
☐ Residencial

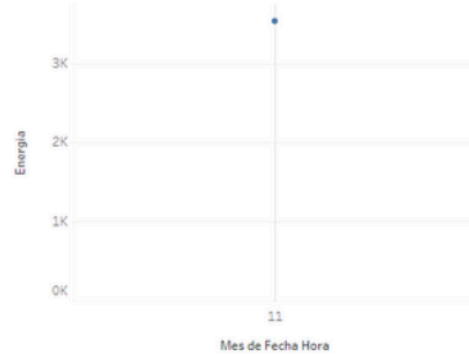
Email

Día de Fecha Hora

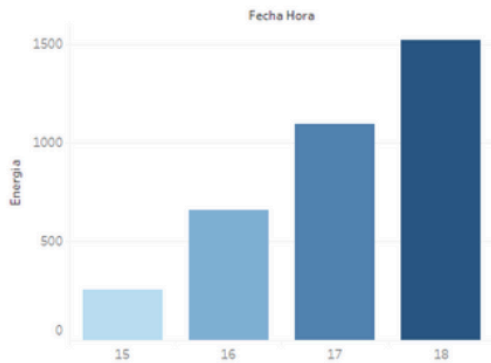
Horas Pico de Consumo



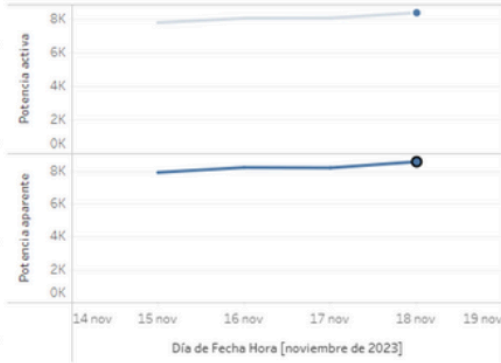
Evolución del Consumo Mensual



Consumo diario de energía



Comparación de Potencias

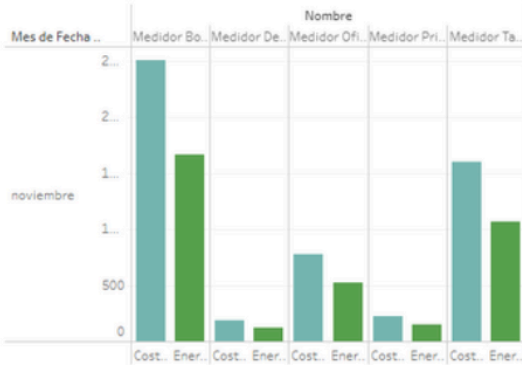


Nombres de medidas
☒ Costo en Bs
☒ Energía

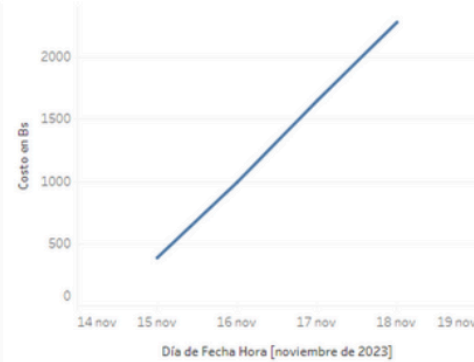
Email

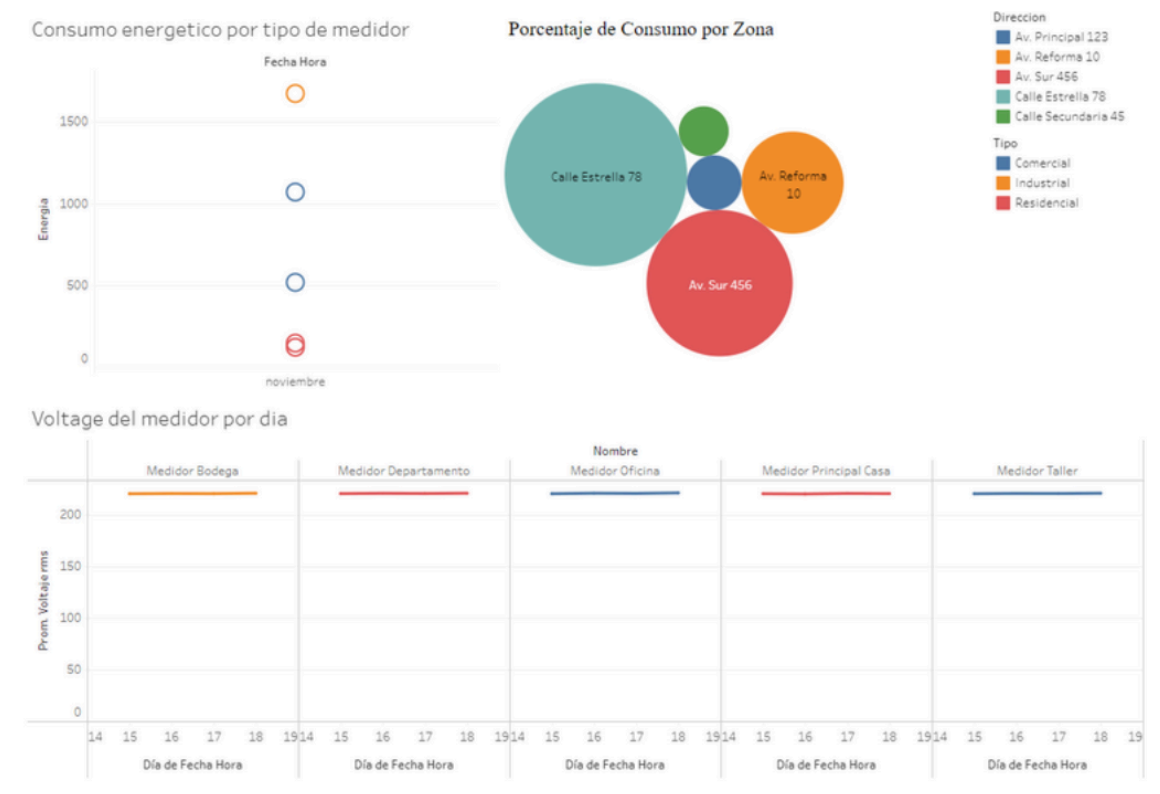
Día de Fecha Hora

Comparación de Costo vs Consumo (kWh)



Costo a lo largo del tiempo





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 proporcionó datos precisos de parámetros eléctricos clave como corriente, voltaje, potencia activa y energía consumida, gracias a la correcta integración de los sensores ACS712 y ZMPT101B 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.

Se encontraron dificultades y problemas a lo largo del proyecto, uno de ellos fue que la medición no era precisa a un inicio, marcaba lo que sea debido a que el sensor era de 30A y que el circuito era solo con focos. Por mas focos que se conectaran no habia cambios notorios, para esto se opto en usar dispositivos de mayor consumo como una plancha para notar la diferencia, pero antes de eso se modifiko el circuito original de modo que ahora se conectaba con un alargador de tres tomacorrientes, de ese modo se podia conectar directamente cualquier dispositivo sin tener que dividir sus cables. Ademas, investigando se configuro de mejor manera el sensor, se usan pines analogicos como el GPIO34 segun el modelo del sensor se configura el valor de sensitivity a 0.066 y usando un multmetro se halla el V_Zero que es el voltaje cuando no hay nada conectado al circuito. Luego para disminuir el ruido que puede haber entre mediciones se toma 1250 medidas que se promedian. De ese modo configurando el sensor adecuadamente y modificando el ciruito para poder medir el consumo de distintos aparatos mas facilmente con tan solo conectarlos e lograron abordar estos problemas de medicion haciendola mas precisa y eficiente

4.2.3. Posibles mejoras.

Precisión en las mediciones:

- Reemplazar los sensores ACS712 y ZMPT101B con versiones 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.

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 el ESP32 y el servidor.

Proteger la base de datos con autenticación más robusta.

5. Conclusiones:

5.1. Resumen del proyecto.

El proyecto consistió en el desarrollo e implementación de un prototipo IoT para la medición en tiempo real del consumo eléctrico en una vivienda, con el objetivo de optimizar el uso de recursos y promover la eficiencia energética. Utilizó un ESP32 como microcontrolador principal, sensores ACS712 para corriente y ZMPT101B para voltaje, y tecnologías como MicroPython, PHP, MySQL y Python para adquirir, procesar, almacenar y visualizar datos. La integración con la aplicación Blink permitió monitoreo remoto desde dispositivos móviles.

El sistema recopila datos en tiempo real, como corriente, voltaje, potencia aparente, potencia activa y energía consumida, que se transmiten al servidor mediante conexiones Wi-Fi y se almacenan en una base de datos MySQL. Estos datos se presentan en un dashboard interactivo desarrollado en Python, que facilita la visualización de tendencias y análisis en tiempo real. Además, los usuarios pueden acceder a información básica a través de Blink, mejorando la accesibilidad y practicidad del sistema.

El prototipo mostró resultados satisfactorios en términos de precisión y eficiencia operativa. Las mediciones energéticas fueron precisas, con actualizaciones en el dashboard cada 2 segundos. La conexión Wi-Fi demostró ser estable, asegurando una transmisión constante de datos. El diseño del dashboard y la integración móvil ofrecieron una experiencia de usuario intuitiva y efectiva.

5.2. Reflexión sobre las habilidades adquiridas.

El desarrollo de este proyecto permitió adquirir y reforzar diversas habilidades técnicas, conceptuales y prácticas.

Técnicas

Se logró un dominio más profundo en la programación de scripts optimizados para la interacción entre hardware y software, especialmente al configurar el ESP32 para realizar mediciones precisas y enviar datos a un servidor mediante peticiones HTTP. También se perfeccionaron las capacidades de integración de tecnologías IoT con bases de datos MySQL, asegurando que los datos recopilados por los sensores fueran procesados y almacenados de

manera eficiente. Adicionalmente, el diseño y desarrollo de un dashboard interactivo en Python representó un reto importante, ya que implicó trabajar con bibliotecas como Dash y Plotly para ofrecer visualizaciones claras y útiles.

Conceptuales

El proyecto implicó una comprensión sólida de conceptos eléctricos básicos, como la medición de corriente y voltaje, además de la interpretación de parámetros como potencia activa y energía consumida. Esta base teórica facilitó el análisis de patrones de consumo energético, lo que permitió obtener conclusiones relevantes para promover la eficiencia en el uso de recursos eléctricos.

Prácticas

En el ámbito práctico, se enfrentaron y resolvieron problemas en tiempo real, desde ajustes en las conexiones físicas del hardware hasta la depuración de errores en sistemas distribuidos. La integración del hardware con el software demandó habilidades para diagnosticar y solucionar inconvenientes como la inestabilidad de la red Wi-Fi, errores en la transmisión de datos y validaciones incorrectas en la base de datos. Estos desafíos fortalecieron la capacidad para gestionar proyectos complejos y mejorar la resiliencia ante problemas técnicos.

5.3. Impacto del proyecto IoT en su contexto.

El prototipo desarrollado tiene un impacto en la promoción del uso eficiente de la energía en viviendas. Al proporcionar datos en tiempo real sobre el consumo eléctrico, los usuarios pueden identificar patrones de uso, optimizar el funcionamiento de dispositivos eléctricos y reducir costos. Además, este monitoreo fomenta prácticas responsables que contribuyen a disminuir el impacto ambiental asociado al consumo energético excesivo.

En un contexto más amplio, el sistema tiene el potencial de ser escalable para su aplicación en edificios, industrias o comunidades enteras. Al integrarse con fuentes de energía renovables y sistemas de automatización, podría convertirse en una herramienta clave para la sostenibilidad energética. Su diseño modular y adaptable lo posiciona como una solución viable para enfrentar desafíos relacionados con la gestión eficiente de recursos en el futuro.

6. Bibliografía:

Citar correctamente toda fuente utilizada: libros, manuales, documentación técnica, sitios web, etc.

7. Anexos:

7.1. Código completo de los scripts.

BD_medidor_electrico

```
-- phpMyAdmin SQL Dump
-- version 5.2.1
-- https://www.phpmyadmin.net/
--
```

```
-- Servidor: 127.0.0.1
-- Tiempo de generación: 24-11-2024 a las 21:47:09
-- Versión del servidor: 10.4.32-MariaDB
-- Versión de PHP: 8.0.30
```

```
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
START TRANSACTION;
SET time_zone = "+00:00";
```

```
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS
*/;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;
```

```
--
-- 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;
```

```
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,  
  `FechaRegistro` timestamp NOT NULL DEFAULT current_timestamp()  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

```
--  
-- Índices para tablas volcadas  
--
```

```
--  
-- Índices de la tabla `lectura_medidor`  
--
```

```
ALTER TABLE `lectura_medidor`  
  ADD PRIMARY KEY (`ID`),  
  ADD KEY `MedidorID` (`MedidorID`);
```

```
--  
-- Índices de la tabla `medidor`  
--
```

```
ALTER TABLE `medidor`  
  ADD PRIMARY KEY (`ID`),  
  ADD KEY `UsuarioID` (`UsuarioID`);
```

```
--  
-- Índices de la tabla `usuario`  
--
```

```
ALTER TABLE `usuario`  
  ADD PRIMARY KEY (`ID`),  
  ADD UNIQUE KEY `Email` (`Email`);
```

```

--
-- AUTO_INCREMENT de las tablas volcadas
--

--
-- AUTO_INCREMENT de la tabla `lectura_medidor`
--
ALTER TABLE `lectura_medidor`
  MODIFY `ID` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=227;

--
-- AUTO_INCREMENT de la tabla `medidor`
--
ALTER TABLE `medidor`
  MODIFY `ID` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=7;

--
-- AUTO_INCREMENT de la tabla `usuario`
--
ALTER TABLE `usuario`
  MODIFY `ID` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=19;

--
-- Restricciones para tablas volcadas
--

--
-- Filtros para la tabla `lectura_medidor`
--
ALTER TABLE `lectura_medidor`
  ADD CONSTRAINT `lectura_medidor_ibfk_1` FOREIGN KEY (`MedidorID`)
  REFERENCES `medidor` (`ID`) ON DELETE CASCADE;

--
-- Filtros para la tabla `medidor`
--
ALTER TABLE `medidor`
  ADD CONSTRAINT `medidor_ibfk_1` FOREIGN KEY (`UsuarioID`) REFERENCES
  `usuario` (`ID`) ON DELETE CASCADE;
COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;

```

Thonny:

SprinteoDatosMedidor.py:

```
from machine import Pin, ADC
import time
import math
import ujson
import urequests as requests
from Wifi_lib import wifi_init, get_html # Librerías externas

# Configuración de Wi-Fi
if wifi_init():
    print("Conexión Wi-Fi exitosa")
else:
    print("Error al conectar a Wi-Fi")

# 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

# URL del servidor
url = "http://192.168.0.9/ManejoBaseDeDatos.php"

# Cálculo de corriente RMS
def calcular_corriente_rms(samples=1250, sampling_time=0.0008):
    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)

# Enviar datos al servidor
def enviar_datos(url, data, headers):
    from machine import Pin, ADC
    import time
    import math
    import ujson
    import urequests as requests
    from Wifi_lib import wifi_init, get_html # Librerías externas

    # Configuración de Wi-Fi
    if wifi_init():
```



```

    print("Conexión Wi-Fi exitosa")
else:
    print("Error al conectar a Wi-Fi")

# 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

# URL del servidor
url = "http://192.168.0.9/ManejoBaseDeDatos.php"

# Cálculo de corriente RMS
def calcular_corriente_rms(samples=1250, sampling_time=0.0008):
    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)

# Enviar datos al servidor
def enviar_datos(url, data, headers):
    intentos = 3
    for intento in range(intentos):
        try:
            response = requests.post(url, data=json.dumps(data), headers=headers)
            if response.status_code == 200:
                print("Datos enviados:", data)
                print("Respuesta del servidor:", response.text)
                response.close()
                return True
            else:
                print("Error en la respuesta del servidor:", response.status_code)
        except Exception as e:
            print(f"Error en el envío (intento {intento+1}/{intentos}):", e)
            time.sleep(5)
    return False

# Bucle principal
while True:
    try:

```

```

corriente_rms = calcular_corriente_rms()
voltaje_rms = 220.0
potencia_aparente = voltaje_rms * corriente_rms
potencia_activa = potencia_aparente * 0.85
energia = potencia_activa * (1 / 3600)

data = {
    "action": "add_lectura",
    "MedidorID": 6,
    "Corriente_rms": corriente_rms,
    "Voltaje_rms": voltaje_rms,
    "Potencia_activa": potencia_activa,
    "Potencia_aparente": potencia_aparente,
    "Energia": energia,
}

headers = {'Content-Type': 'application/json'}
enviar_datos(url, data, headers)

except Exception as e:
    print("Error en la ejecución:", e)
    time.sleep(5)

time.sleep(10)

```

Conexion en php ManejoDeBasededatos.php:

```

<?php
header("Access-Control-Allow-Origin: *"); // Permitir acceso desde
cualquier origen
header("Access-Control-Allow-Methods: POST, GET, OPTIONS");
header("Access-Control-Allow-Headers: Content-Type, Authorization");
header("Content-Type: application/json");

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

try {
    // Conectar 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);

    // Manejar solicitudes GET

```

```

if ($_SERVER['REQUEST_METHOD'] === 'GET') {
    if (isset($_GET['action'])) {
        if ($_GET['action'] === 'get_usuario') {
            // Obtener datos de un usuario por ID
            $id = intval($_GET['id']);
            $stmt = $conn->prepare("SELECT * FROM usuario WHERE ID
= :ID");

            $stmt->bindParam(':ID', $id);
            $stmt->execute();
            $user = $stmt->fetch(PDO::FETCH_ASSOC);
            echo json_encode($user);
        } elseif ($_GET['action'] === 'get_medidores') {
            // Obtener medidores asociados a un usuario
            $usuarioID = intval($_GET['usuarioID']);
            $stmt = $conn->prepare("SELECT * FROM medidor WHERE
UsuarioID = :UsuarioID");
            $stmt->bindParam(':UsuarioID', $usuarioID);
            $stmt->execute();
            $medidores = $stmt->fetchAll(PDO::FETCH_ASSOC);
            echo json_encode($medidores);
        } else {
            echo json_encode(["status" => "error", "message" =>
"Acción no válida"]);
        }
    } else {
        echo json_encode(["status" => "error", "message" => "Acción
no especificada"]);
    }
}

// Manejar solicitudes POST
elseif ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $data = json_decode(file_get_contents("php://input"), true);

    if (!isset($data['action'])) {
        echo json_encode(["status" => "error", "message" => "Acción
no especificada"]);
        exit;
    }

    switch ($data['action']) {
        case "register":
            // Registrar un nuevo usuario

```

```

        $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", "message"
=> "Usuario registrado"]);
        } else {
            echo json_encode(["status" => "error", "message" =>
"Error al registrar"]);
        }
        break;

    case "login":
        // Inicio de sesión
        $stmt = $conn->prepare("SELECT * FROM usuario WHERE
Email = :Email");
        $stmt->bindParam(':Email', $data['Email']);
        $stmt->execute();
        $user = $stmt->fetch(PDO::FETCH_ASSOC);

        if ($user) {
            echo json_encode(["status" => "success", "user" =>
$user]);
        } else {
            echo json_encode(["status" => "error", "message" =>
"Usuario no encontrado"]);
        }
        break;

    case "add_medidor":
        // Agregar un nuevo medidor
        $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", "message"
=> "Medidor agregado"]);

```

```

        } else {
            echo json_encode(["status" => "error", "message" =>
"Error al agregar medidor"]);
        }
        break;

        default:
            echo json_encode(["status" => "error", "message" =>
"Acción no válida"]);
            break;
    }
} else {
    echo json_encode(["status" => "error", "message" => "Método no
permitido"]);
}
} catch (PDOException $e) {
    echo json_encode(["status" => "error", "message" => "Error en la
conexión: " . $e->getMessage()]);
}

// Cerrar conexión
$conn = null;
?>

```

7.2. Capturas de comandos o logs de ejecución.

The screenshot shows the Thonny IDE interface. The main editor displays a Python script for an ACS712 sensor. The script imports necessary modules, configures the pin, initializes the ADC, and sets up the sensor parameters. The console window at the bottom shows the output of the script, displaying a series of RMS current measurements.

```

1 from machine import Pin, ADC
2 import time
3 import math
4
5 # Configuración del pin analógico donde está conectado el ACS712
6 analog_pin = 34 # GPIO 34
7 adc = ADC(Pin(analog_pin))
8 adc.atten(ADC.ATTN_11DB) # Rango 0-3.3V
9 adc.width(ADC.WIDTH_12BIT) # Resolución 12 bits (0-4095)
10
11 # Configuración del ACS712
12 sensitivity = 0.066 # Sensibilidad del ACS712 30A (V/A)
13 VCC = 5 # Voltaje de alimentación del sensor (en V)
14 V_ZERO = 3.75 # Offset del sensor cuando no hay corriente (en V)
15
16 # Parámetros para el cálculo RMS
17 sampling_time = 0.001 # Tiempo entre muestras (1 ms)

```

Console output:

```

Corriente RMS medida: 1.469 A
Corriente RMS medida: 1.603 A
Corriente RMS medida: 1.575 A
Corriente RMS medida: 1.625 A
Corriente RMS medida: 1.570 A
Corriente RMS medida: 1.541 A
Corriente RMS medida: 1.420 A
Corriente RMS medida: 1.430 A
Corriente RMS medida: 1.642 A
Corriente RMS medida: 1.583 A

```

```
1 from machine import Pin, ADC

Consola
Coneccion exitosa
IP: 192.168.0.19
RSSI: -54
MAC: a8:42:e3:91:72:dc
Canal: 11
SSID: Flia Estevez
power: 19.5
Hostname: mpy-esp32
Bienvenido al sistema de gestión de medidores.
Seleccione una opción:
1. Registrar nuevo usuario
2. Iniciar sesión
Opción: 2
Ingrese su correo electrónico: rene@gmail.com
Inicio de sesión exitoso. Datos del usuario: {'Nombre': 'rene', 'Telefono': '2324', 'ID': '6', 'Direccion': 'fwe fe', 'Email': 'rene@gmail.com', 'FechaRegistro': '2024-11-24 13:15:45'}
Bienvenido, usuario ID: 6

Medidores registrados:
1. casa2 (ID: 6)
Seleccione un medidor (número): 1
Medidor seleccionado: casa2

1. Comenzar medición
2. Volver al menú principal
Opción: 1
Corriente RMS: 1.53 A
Voltaje RMS: 220.00 V
Potencia Activa: 285.83 W
Potencia Aparente: 336.27 VA
Energía: 0.0794 kWh
Medición guardada exitosamente.
```

```
1 from machine import Pin, ADC
2 import time
3 import math
4 import urequests as requests
5 import ujson
6 import utime
7 from secrets import secrets # Archivo separado para las credenciales Wi-Fi
8 from wifi_lib import wifi_init, get_html # Librerías externas
9
10 # URL del servidor PHP
11 url = "http://192.168.0.9/ManejoBaseDeDatos.php"
12 wifi_init()
13
14 # Función para registrar un usuario
15 def registrar_usuario(nombre, direccion, telefono, email):

Consola
Potencia Aparente: 262.67 VA
Energía: 0.0620 kWh
Medición guardada exitosamente.
Corriente RMS: 1.26 A
Voltaje RMS: 220.00 V
Potencia Activa: 236.39 W
Potencia Aparente: 278.10 VA
Energía: 0.0657 kWh
Medición guardada exitosamente.
Corriente RMS: 1.31 A
Voltaje RMS: 220.00 V
Potencia Activa: 245.04 W
Potencia Aparente: 288.28 VA
Energía: 0.0681 kWh
Medición guardada exitosamente.
```

7.3. Diagramas complementarios.