

UNIVERSIDAD CENTRAL DEL ECUADOR

INFRAESTRUCTURA TI 2

GRUPO #6

INTEGRANTES:

VICTOR CAÑAR

RICARDO CHAPI

LUIS LUNA

MATEO MALES

KEVIN TENORIO

CURSO: SI5-001

FECHA: 25/02/2025



MANUAL TECNICO DEL CIRCUITO

1. Introducción

Este manual describe el proceso de diseño, armado y funcionamiento de un sistema basado en un panel solar y un circuito electrónico para la recopilación de datos a una base de datos. Se detallan los materiales utilizados, el ensamblaje del circuito y las conexiones realizadas, así como el código empleado para la adquisición de datos. Además, se explican los principios de funcionamiento de cada componente y las posibles aplicaciones del sistema.

2. Objetivos

- Diseñar y ensamblar un circuito que permita la recopilación de datos de sensores utilizando un ESP32.
- Integrar un sistema de alimentación basado en energía solar.
- Enviar la información obtenida a una base de datos para su posterior análisis.
- Explicar el funcionamiento detallado de cada componente.
- Identificar posibles aplicaciones y mejoras del sistema.
- Evaluar el rendimiento y eficiencia del sistema en diferentes condiciones ambientales.
- Proponer soluciones para optimizar el uso de la energía captada.

3. Materiales

- Panel solar de 9W (9.1V, 0.36A)
- Módulo de carga para panel solar
- Regulador de voltaje *L7805*
- Capacitores de 10 μ F (x2)
- ESP32
- Sensores de corriente *ACS712* (x2)
- Sensor de temperatura *DS18B20*
- Resistencia de 4.7k Ω
- Placa de baquelita
- Cables y conectores
- Soldadura y cautín

- Batería recargable (opcional)
- Módulo de comunicación WiFi/Bluetooth (integrado en el ESP32)
- Software para monitoreo de datos (ejemplo: ThingSpeak, Firebase, InfluxDB)

4. Desarrollo y Ensamblaje del Proyecto

4.1 Conexión del Panel Solar

El panel solar de 9W se conectó al módulo de carga de panel solar a través de los terminales **IN+** y **IN-**, permitiendo la regulación de la energía generada. Este módulo gestiona la carga de baterías y proporciona un voltaje estable al sistema. Se verificó que la orientación del panel maximizara la captación de luz solar.

4.2 Regulación de Voltaje

Para estabilizar la tensión de salida, se empleó el regulador *L7805*, con la siguiente configuración:

- Pin 1 (IN): Entrada de voltaje desde el módulo de carga.
- Pin 2 (GND): Tierra común del sistema.
- Pin 3 (OUT): Salida estabilizada de 5V.

Se colocaron capacitores de $10\mu F$ en la entrada y salida para mejorar la estabilidad de la señal y reducir ruidos eléctricos.

4.3 Conexión de Sensores

Los sensores de corriente *ACS712* permiten medir el flujo de corriente eléctrica en un circuito. Su conexión se realizó de la siguiente manera:

- **VCC** a 5V.
- **GND** a tierra.
- **VOUT** a pines analógicos del ESP32.
- **IP+ e IP-** en serie con la línea de corriente a medir.

El sensor de temperatura *DS18B20* mide la temperatura ambiente y se conectó de la siguiente manera:

- **GND** a tierra.
- **DQ** a un GPIO digital del ESP32 con una resistencia de $4.7\text{k}\Omega$ a 3.3V.
- **VCC** a 3.3V.

4.4 Montaje y Soldadura

Los componentes fueron ensamblados sobre una placa de baquelita, asegurando la correcta distribución de la tierra común y la polaridad adecuada. Se utilizó soldadura de estaño para garantizar conexiones seguras y estables.

5. Funcionamiento del Circuito

El circuito desarrollado funciona de la siguiente manera:

1. El panel solar convierte la energía luminosa en energía eléctrica y la entrega al módulo de carga.
2. El módulo de carga regula el voltaje y carga una batería (si se usa) o alimenta el regulador de voltaje.
3. El regulador *L7805* estabiliza la salida a 5V para los sensores y algunos componentes del ESP32.
4. Los sensores recopilan información sobre la corriente y la temperatura, enviándola al ESP32.
5. El ESP32 procesa los datos y los transmite a una base de datos a través de una conexión inalámbrica (WiFi o Bluetooth).
6. Los datos pueden visualizarse en tiempo real mediante una aplicación o un dashboard en la nube.

6. Aplicaciones y Posibles Mejoras

6.1 Aplicaciones

- Monitoreo de consumo eléctrico en sistemas fotovoltaicos.
- Supervisión de temperatura en ambientes industriales o domésticos.
- Control de eficiencia en paneles solares.
- Uso en sistemas IoT para gestión energética.
- Aplicación en agricultura para monitoreo de condiciones ambientales.
- Implementación en sistemas autónomos de energía renovable.

6.2 Posibles Mejoras

- Incorporación de una pantalla LCD para visualizar los datos en tiempo real.
- Uso de una batería de respaldo para mejorar la autonomía del sistema.
- Implementación de algoritmos de análisis de datos para predecir tendencias de consumo y temperatura.
- Expansión del sistema con más sensores para monitoreo en múltiples puntos.
- Integración con inteligencia artificial para análisis predictivo de datos.
- Uso de un convertidor DC-DC más eficiente para reducir pérdidas de energía.

7. Código Fuente

Este código configura un ESP32 como un Access Point (AP) y un servidor web en el puerto 80. El ESP32 lee datos de sensores de corriente ACS712 y temperatura DS18B20, los convierte a JSON y los envía como respuesta a clientes que se conecten al servidor.

Explicación por partes

1. Inclusión de librerías

```
#include <WiFi.h>  
  
#include <OneWire.h>  
  
#include <DallasTemperature.h>  
  
#include <ArduinoJson.h>
```

WiFi.h: Permite al ESP32 manejar redes WiFi.

OneWire.h: Protocolo de comunicación para sensores como el DS18B20.

DallasTemperature.h: Librería específica para manejar sensores DS18B20.

ArduinoJson.h: Facilita la creación y manipulación de datos en formato JSON.

--

2. Configuración del Access Point

```
const char *ssid = "ESP32_AP";  
const char *password = "12345678";  
WiFiServer server(80);
```

Se define un SSID (ESP32_AP) y contraseña (12345678) para el Access Point.

Se crea un servidor en el puerto 80.

--

3. Definición de pines y parámetros

Pines de sensores ACS712 (medición de corriente)

```
const int sensorCorrientePanel = 34;  
const int sensorCorrienteBateria = 35;
```

Los sensores de corriente ACS712 están conectados a los pines 34 y 35.

Parámetros del sensor ACS712

```
const float sensibilidad = 0.100; // Sensibilidad del sensor en V/A  
const float Vref = 2.5; // Voltaje de referencia del sensor en V  
const float Vmax = 3.3; // Voltaje máximo del ESP32 en V  
const int resolucionADC = 4095; // Resolución del ADC (12 bits)
```

Estos valores permiten convertir la lectura analógica en corriente.

Configuración del sensor DS18B20

```
const int pinDatosDS18B20 = 21;  
OneWire oneWire(pinDatosDS18B20);  
DallasTemperature sensoresTemperatura(&oneWire);
```

El sensor de temperatura DS18B20 está conectado al pin 21 y usa el protocolo OneWire.

4. Función setup()

```
void setup() {  
    Serial.begin(115200);  
    WiFi.softAP(ssid, password);  
    server.begin();  
    Serial.println("Access Point iniciado");
```

```
Serial.print("Dirección IP: ");
Serial.println(WiFi.softAPIP());
sensoresTemperatura.begin();
}
```

Inicia el puerto serie para depuración.

Crea un Access Point con el SSID y contraseña definidos.

Muestra la dirección IP del ESP32 como servidor.

Inicializa el sensor de temperatura.

5. Función loop()

```
WiFiClient client = server.available();
if (client) {
    Serial.println("Cliente conectado");
```

Si un cliente se conecta al servidor, se inicia el procesamiento.

Leer la solicitud HTTP

```
String request = client.readStringUntil('\r');
```

```
client.flush();
```

Se lee la solicitud HTTP que envía el cliente.

Manejo de solicitudes OPTIONS (CORS)

```
if (request.indexOf("OPTIONS") >= 0) {  
  
    client.println("HTTP/1.1 204 No Content");  
  
    client.println("Access-Control-Allow-Origin: *");  
  
    client.println("Access-Control-Allow-Methods: GET, POST, OPTIONS");  
  
    client.println("Access-Control-Allow-Headers: Content-Type");  
  
    client.println("Connection: close");  
  
    client.println();  
  
    client.stop();  
  
    return;  
}
```

Si el cliente envía una solicitud CORS (Cross-Origin Resource Sharing), se responde con 204 No Content, permitiendo peticiones desde otros dominios.

Leer los sensores ACS712 (corriente)

```
int lecturaPanel = analogRead(sensorCorrientePanel);  
  
int lecturaBateria = analogRead(sensorCorrienteBateria);
```

```
float voltajeCorrientePanel = (lecturaPanel * Vmax) / resolucionADC;
```

```
float corrientePanel = (voltajeCorrientePanel - Vref) / sensibilidad;
```

```
float voltajeCorrienteBateria = (lecturaBateria * Vmax) / resolucionADC;
```

```
float corrienteBateria = (voltajeCorrienteBateria - Vref) / sensibilidad;
```

1. Se lee el valor analógico de cada sensor.

2. Se convierte a voltaje.

3. Se convierte a corriente, restando Vref y dividiendo entre la sensibilidad (0.1 V/A).

Leer la temperatura con DS18B20

```
sensoresTemperatura.requestTemperatures();
```

```
float temperatura = sensoresTemperatura.getTempCByIndex(0);
```

Se solicita una medición de temperatura y se obtiene el valor en grados Celsius.

Crear el JSON con los datos

```
StaticJsonDocument<200> json;
```

```
json["corriente_panel"] = corrientePanel;
```

```
json["voltaje_corriente_panel"] = voltajeCorrientePanel;  
json["corriente_bateria"] = corrienteBateria;  
json["voltaje_corriente_bateria"] = voltajeCorrienteBateria;  
json["temperatura"] = temperatura;  
  
String jsonString;  
serializeJson(json, jsonString);
```

Se crea un objeto JSON con los datos de corriente y temperatura.

Se convierte en una cadena (jsonString).

Enviar la respuesta HTTP con JSON

```
client.println("HTTP/1.1 200 OK");  
client.println("Content-Type: application/json");  
client.println("Access-Control-Allow-Origin: *");  
client.println("Content-Length: " + String(jsonString.length()));  
client.println("Connection: close");  
client.println();  
client.println(jsonString);
```

Se envía una respuesta HTTP 200 OK con el JSON generado.

Se permite CORS (Access-Control-Allow-Origin: *).

Cerrar la conexión

```
Serial.println("Datos enviados: " + jsonString);
delay(1000);
client.stop();
```

Se imprime el JSON enviado en el puerto serie.

Se cierra la conexión con el cliente.

Resumen del funcionamiento

1. El ESP32 crea un Access Point con un servidor web en el puerto 80.

2. Cuando un cliente se conecta, lee los sensores:

Corriente de panel y batería con ACS712.

Temperatura con DS18B20.

3. Los datos se formatean en JSON y se envían al cliente.

4. Se maneja CORS, permitiendo peticiones desde cualquier dominio.

- - -

¿Cómo probar el servidor?

1. Conéctate al WiFi generado por el ESP32 (ESP32_AP).

2. Abre un navegador y ve a <http://192.168.4.1/>.

3. El ESP32 responderá con un JSON con los datos de corriente y temperatura.

4. También puedes hacer una petición HTTP con Postman o cURL.

Este código es útil para un sistema de monitoreo en tiempo real de corriente y temperatura.

8. Referencias

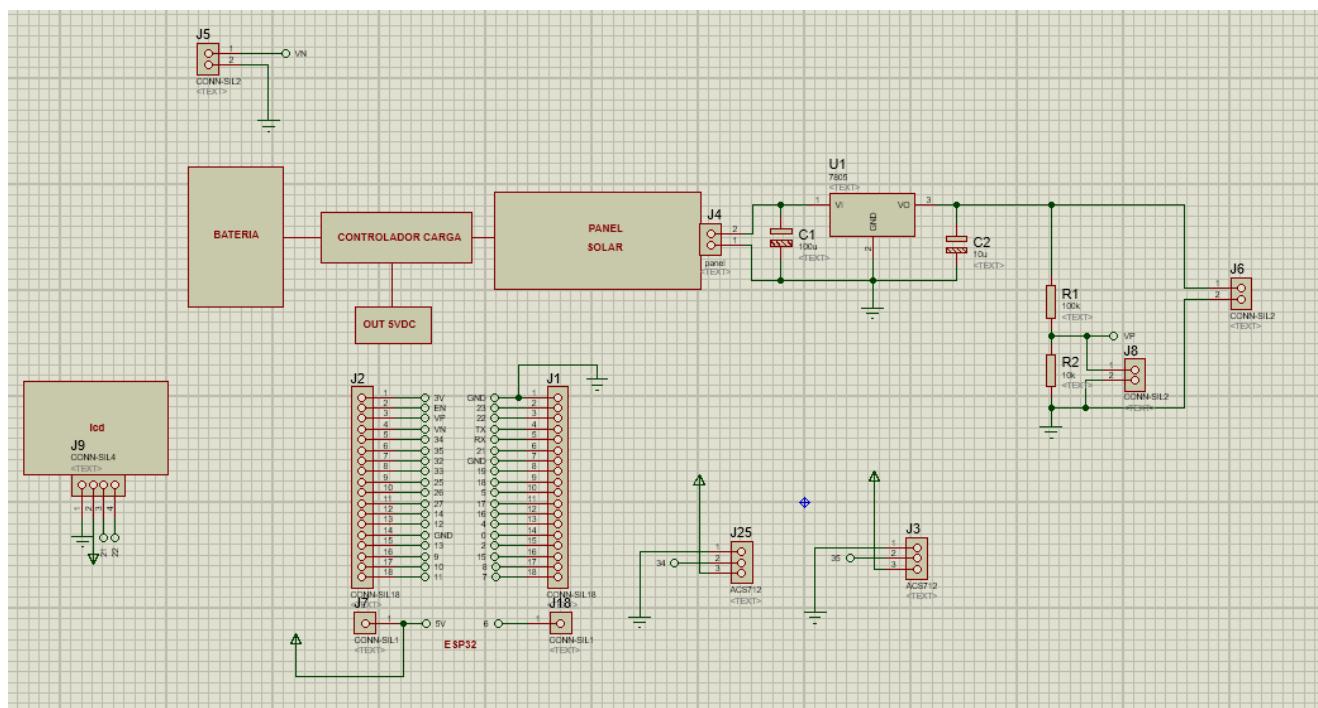


Ilustración 1 Diagrama de pines del circuito

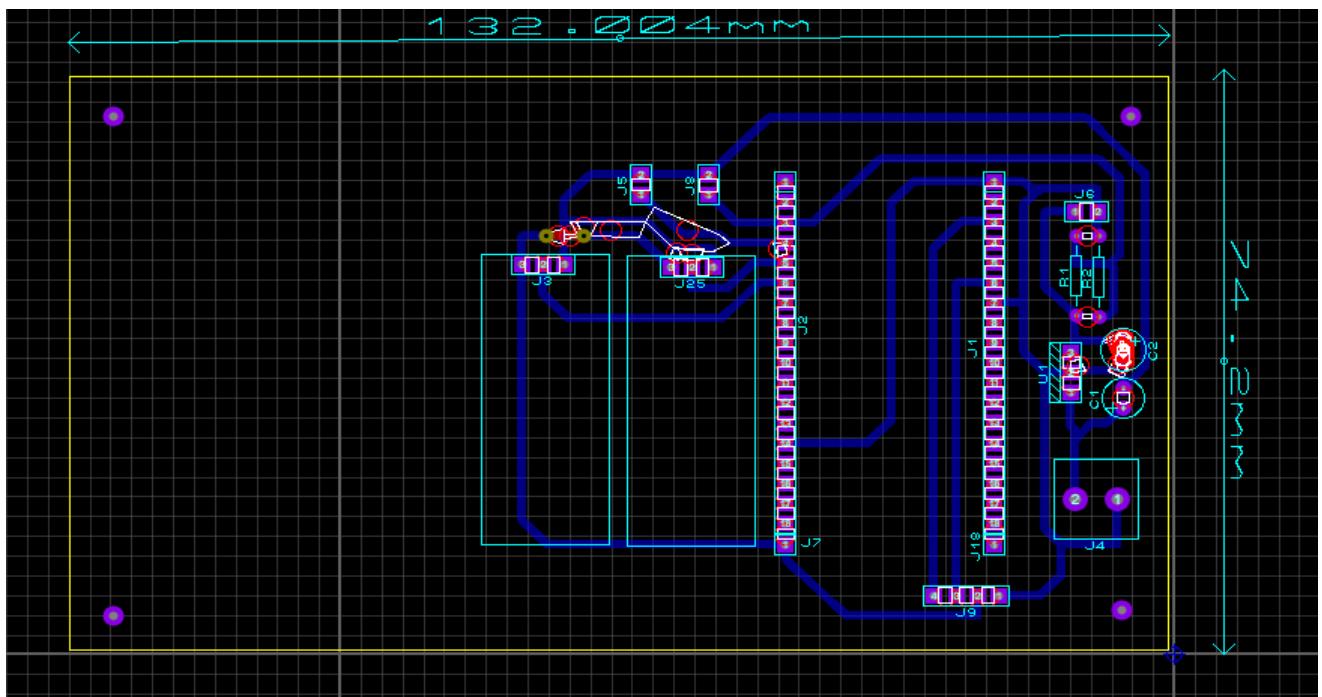


Ilustración 2 Diseño de las conexiones en la baquelita

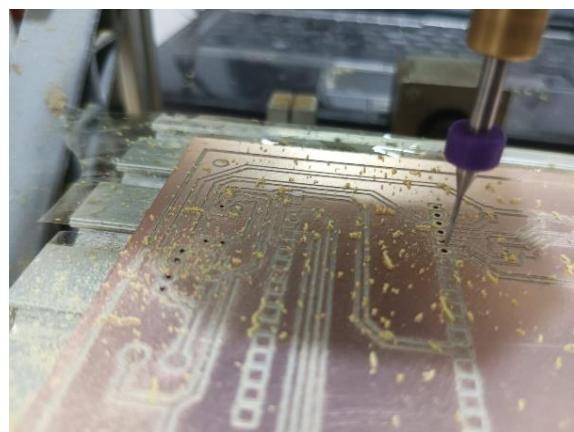


Ilustración 3 Diseño de pistas de la baquelita

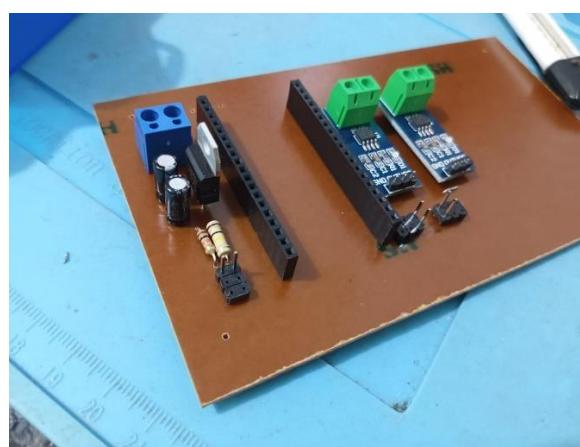


Ilustración 4 Armado y montaje de los distintos componentes

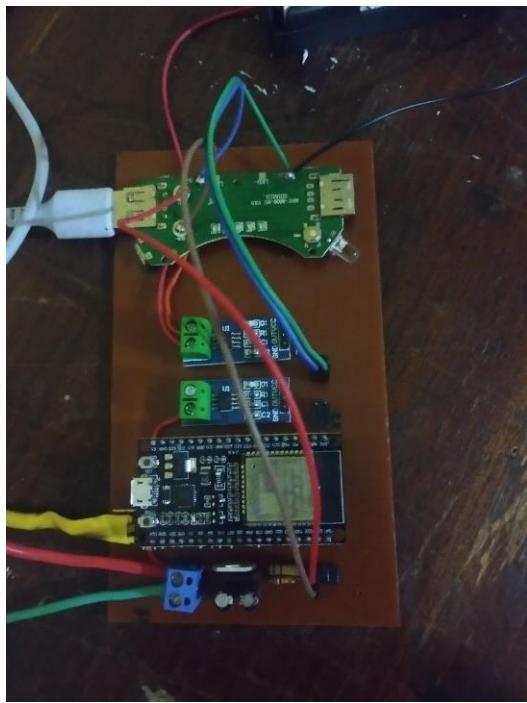


Ilustración 5 Resultado final del proyecto

9. Conclusiones

Este proyecto demuestra la viabilidad de utilizar energía solar para alimentar un sistema de adquisición de datos basado en ESP32. La correcta selección de componentes y una conexión precisa permiten una operación estable y eficiente. Además, se destaca su potencial en aplicaciones domésticas e industriales, con posibilidad de expansiones futuras. Se identificaron oportunidades de optimización en la eficiencia del sistema y en la implementación de análisis de datos avanzados.

Manual Dashboard de datos

Introducción

Este documento describe el proceso de desarrollo y funcionamiento del dashboard web para la recolección de datos de un sistema físico basado en ESP32. La solución permite visualizar, en tiempo real, datos críticos del panel solar, tales como corriente, voltaje, temperatura, potencia instantánea y energía generada (diaria, mensual y anual). Los datos se obtienen mediante sensores conectados al ESP32, el cual transmite información en formato JSON a través de la red WiFi.

Objetivos

- **Monitoreo en Tiempo Real:** Visualizar en una interfaz web los datos capturados por el ESP32.
- **Procesamiento y Validación:** Calcular la potencia y la energía generada, validando rangos de operación para detectar posibles fallas o condiciones anómalas.
- **Generación de Reportes:** Permitir la exportación de la información recolectada a un archivo Excel para análisis posterior.
- **Interfaz Intuitiva:** Proveer una experiencia de usuario clara y responsive que facilite el análisis del comportamiento del sistema físico.

Herramientas Utilizadas

- **HTML5 & CSS3:** Desarrollo de la estructura y el diseño visual del dashboard.
- **JavaScript:** Implementación de la lógica para la actualización, validación y procesamiento de datos en tiempo real.
- **Chart.js:** Creación de gráficos interactivos para la representación dinámica de los datos.
- **SheetJS (xlsx.full.min.js):** Herramienta para la generación y descarga de reportes en formato Excel.
- **ESP32:** Microcontrolador encargado de la adquisición de datos desde sensores de corriente, voltaje y temperatura.

Desarrollo del Modelo

Diseño del Dashboard

- **Estructura Web:**

Se desarrolló una página HTML que integra:

- **Pantalla de Bienvenida:** Presenta una introducción al sistema y un botón para iniciar la visualización.
- **Dashboard Principal:** Incluye tarjetas informativas para cada parámetro (corriente, voltaje, temperatura, potencia y energía) y un gráfico interactivo.
- **Botones de Navegación:** Permiten alternar entre la vista de tablas y la de gráfico, además de generar reportes en Excel.

- **Estilos y Animaciones:**

Se aplicaron estilos modernos mediante CSS, utilizando degradados, sombras y transiciones para realzar la experiencia visual del usuario.

Integración con ESP32

- **Conexión y Comunicación:**

El ESP32 funciona como servidor, enviando datos en formato JSON a una dirección IP predefinida (por ejemplo, `http://192.168.4.1`).

- **Actualización Periódica:**

La función `actualizarDatos()` se invoca cada 5 segundos utilizando `setInterval()`, asegurando la actualización constante de los valores mostrados.

- **Validación y Procesamiento:**

Se establecen rangos de operación para determinar si el panel se encuentra desconectado, defectuoso o en baja carga. En caso de anomalías, se muestran mensajes de alerta y se evita el cálculo de valores erróneos.

Visualización de Datos

- **Tarjetas Informativas:**

Cada tarjeta muestra un parámetro específico, tales como la corriente y voltaje del panel y la batería, la temperatura, así como los cálculos de potencia y energía generada.

- **Gráfico Dinámico:**

Se implementa un gráfico de líneas con `Chart.js`, que actualiza en tiempo real los valores relevantes, manteniendo un historial de las últimas mediciones.

- **Exportación a Excel:**

La función `generarReporteExcel()` utiliza `SheetJS` para transformar el historial de datos en un archivo Excel, facilitando su descarga para un análisis más detallado.

Funcionamiento del Código

Lógica Principal

1. Inicio y Visualización:

Al cargar la página, se muestra una pantalla de bienvenida. Tras pulsar el botón “Comenzar”, se oculta dicha pantalla y se revela el dashboard con las tablas de datos y el gráfico.

2. Obtención de Datos:

La función actualizarDatos() realiza una solicitud HTTP al ESP32 para obtener datos en formato JSON. Los datos se procesan, validan y, en función de los rangos establecidos, se actualizan tanto las tarjetas informativas como el gráfico.

3. Cálculos y Validación:

- **Cálculo de Potencia:** Se multiplica la corriente y el voltaje del panel para obtener la potencia instantánea.
- **Cálculo de Energía:** Utilizando una estimación de 12 horas de luz solar diarias, se calcula la energía generada a nivel diario, mensual y anual.
- **Validación:** Si los datos se encuentran fuera de los rangos esperados, se activa un contador de registros no válidos y se muestra una alerta.

4. Actualización de la Interfaz:

Los valores validados se muestran en tiempo real en las tarjetas del dashboard y en el gráfico interactivo, permitiendo al usuario monitorear el comportamiento del sistema físico de manera intuitiva.

Fragmento Representativo del Código

```
<script>

const ESP32_IP = "http://192.168.4.1";
const ctx = document.getElementById("sensorChart").getContext("2d");
let datosRecolectados = [];
const horasLuzSolar = 12;
const corrienteDesconectadaMin = 0.011;
const voltajeDesconectadaMin = 2;
const corrienteDesconectadaMax = 10;
const voltajeDesconectadaMax = 60;
let invalidCount = 0;

let sensorChart = new Chart(ctx, {
    type: 'line',
    data: {
        labels: [],
        datasets: [
            { label: "Corriente Panel (A)", borderColor: "#FFA726", backgroundColor: "rgba(255, 167, 38, 0.2)", data: [], fill: true },
            { label: "Voltaje Panel (V)", borderColor: "#4CAF50", backgroundColor: "rgba(76, 175, 80, 0.2)", data: [], fill: true },
            // Otros datasets para batería y temperatura
        ]
    },
    options: { responsive: true, scales: { y: { beginAtZero: true } } }
});
```

```

function actualizarDatos() {
    fetch(ESP32_IP)
        .then(response => response.json())
        .then(datos => {
            let corrientePanel = Math.abs(datos.corriente_panel) / 100;
            let voltajePanel = Math.abs(datos.voltaje_corriente_panel);
            // Validación de datos y actualización de la UI
            // Cálculo de potencia y energía, actualización del gráfico
        })
        .catch(error => console.error("Error obteniendo datos:", error));
}

setInterval(actualizarDatos, 5000);
</script>

```

El fragmento anterior ilustra la inicialización del gráfico, la obtención de datos desde el ESP32 y la actualización de la interfaz.

Visualización en el Navegador

- **Interfaz de Usuario:**

El dashboard cuenta con un encabezado con un degradado y título, una pantalla de bienvenida, tarjetas informativas y un gráfico interactivo que muestra la evolución de los parámetros del sistema.

- **Experiencia del Usuario:**

La interfaz es responsive y moderna, facilitando la interacción y permitiendo que la información del sistema físico se presente de manera clara y precisa.

Conclusión

El dashboard web desarrollado permite monitorear en tiempo real el funcionamiento de un sistema de energía solar, mostrando datos críticos como corriente, voltaje, temperatura, potencia y energía generada. La integración con el ESP32 y el uso de tecnologías web (HTML, CSS, JavaScript, Chart.js y SheetJS) hacen de esta solución una herramienta eficaz para el análisis y seguimiento del rendimiento del panel físico.

Anexos

- **Código Fuente Completo:**

```

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dashboard ESP32 - Energía Solar</title>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script src="libs/xlsx.full.min.js"></script> <!-- SheetJS local -->

```

```
<link
  href="https://fonts.googleapis.com/css2?family=Inter:wght@300;400;500;600;700&display=swap"
  rel="stylesheet">
<style>
  /* Estilos generales */
  body {
    font-family: 'Inter', sans-serif;
    margin: 0;
    padding: 0;
    background-color: #fafafa;
    color: #333;
  }

  /* Encabezado */
  header {
    background: linear-gradient(135deg, #FFA726, #FB8C00); /* Degradado naranja */
    color: white;
    padding: 20px 0;
    text-align: center;
    box-shadow: 0px 4px 15px rgba(0, 0, 0, 0.2);
  }

  header h1 {
    margin: 0;
    font-size: 2rem;
    font-weight: 600;
  }

  /* Pantalla de inicio */
  .welcome-screen {
    text-align: center;
    padding: 50px 20px;
  }

  .welcome-screen h2 {
    font-size: 2rem;
    color: #333;
    margin-bottom: 20px;
  }

  .welcome-screen p {
    font-size: 1.1rem;
    color: #555;
    max-width: 600px;
    margin: 0 auto 30px;
  }

  .welcome-screen button {
    background: #FFA726;
    color: white;
    border: none;
    padding: 10px 20px;
    font-size: 1rem;
    border-radius: 5px;
    cursor: pointer;
    transition: background 0.3s;
  }

  .welcome-screen button:hover {
    background: #FB8C00;
```

```
}

/* Contenedor principal del dashboard */
.dashboard {
    width: 90%;
    max-width: 1200px;
    margin: 40px auto;
    display: none; /* Oculto inicialmente */
}

/* Alerta para el panel */
#panelAlert {
    color: red;
    font-weight: bold;
    text-align: center;
    margin-bottom: 10px;
}

/* Tarjetas de datos */
.card {
    background: white;
    padding: 20px;
    border-radius: 15px;
    box-shadow: 0px 6px 20px rgba(0, 0, 0, 0.1);
    text-align: left;
    transition: transform 0.3s, box-shadow 0.3s;
    position: relative;
    overflow: hidden;
}

.card::before {
    content: "";
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 4px;
    background: linear-gradient(90deg, #FFA726, #FB8C00); /* Degradado naranja */
}

.card:hover {
    transform: translateY(-10px);
    box-shadow: 0px 12px 30px rgba(0, 0, 0, 0.2);
}

.card h3 {
    margin: 0 0 10px;
    font-size: 1.1rem;
    font-weight: 500;
    color: #333;
    display: flex;
    align-items: center;
}

.card h3::before {
    content: "✓";
    color: #4CAF50; /* Verde natural */
    font-size: 1.2rem;
    margin-right: 10px;
}
```

```
.card p {
    margin: 0;
    font-size: 1.4rem;
    font-weight: 600;
    color: #333;
}

/* Gráfico de área */
.chart-container {
    grid-column: 1 / -1;
    background: white;
    padding: 20px;
    border-radius: 15px;
    box-shadow: 0px 6px 20px rgba(0, 0, 0, 0.1);
    margin-top: 20px;
    transition: box-shadow 0.3s;
}

.chart-container:hover {
    box-shadow: 0px 12px 30px rgba(0, 0, 0, 0.2);
}

.chart-container canvas {
    max-width: 100%;
}

/* Botones de navegación */
.nav-buttons {
    text-align: center;
    margin: 20px 0;
}

.nav-buttons button {
    background: #FFA726;
    color: white;
    border: none;
    padding: 10px 20px;
    font-size: 1rem;
    border-radius: 5px;
    cursor: pointer;
    transition: background 0.3s;
    margin: 0 10px;
}

.nav-buttons button:hover {
    background: #FB8C00;
}

/* Pie de página */
footer {
    background: linear-gradient(135deg, #FFA726, #FB8C00); /* Degradado naranja */
    color: white;
    text-align: center;
    padding: 15px 0;
    margin-top: 40px;
    box-shadow: 0px -4px 15px rgba(0, 0, 0, 0.2);
}

footer p {
```

```
        margin: 0;
        font-size: 14px;
    }

```

```
</style>

```

```
</head>

```

```
<body>
    <!-- Encabezado -->
    <header>
        <h1>Dashboard ESP32 - Energía Solar</h1>
    </header>

    <!-- Pantalla de inicio -->
    <div class="welcome-screen">
        <h2>¡Bienvenido al Sistema de Monitoreo de Energía Solar!</h2>
        <p>
            Este proyecto está diseñado para monitorear en tiempo real los datos de un sistema de energía solar,
            incluyendo corriente, voltaje y temperatura. Utiliza un ESP32 para recopilar y enviar los datos.
        </p>
        <p><strong>Créditos:</strong> Desarrollado por _Grupo 6_ Infraestructura De Las TI II</p>
        <button onclick="mostrarDashboard()">Comenzar</button>
    </div>

    <!-- Contenedor principal del dashboard -->
    <div class="dashboard">
        <!-- Alerta para el estado del panel -->
        <div id="panelAlert"></div>

        <!-- Botones de navegación -->
        <div class="nav-buttons">
            <button onclick="mostrarTablas()">Ver Tablas</button>
            <button onclick="mostrarGrafico()">Ver Gráfico</button>
            <button onclick="generarReporteExcel()">Generar Reporte (Excel)</button>
        </div>

        <!-- Tarjetas de datos -->
        <div id="tablas">
            <div class="dashboard-grid">
                <div class="card" id="corrientePanelCard">
                    <h3>Corriente Panel (A)</h3>
                    <p id="corrientePanel">0.000</p>
                </div>

                <div class="card" id="voltajePanelCard">
                    <h3>Voltaje Panel (V)</h3>
                    <p id="voltajePanel">0.000</p>
                </div>

                <div class="card" id="corrienteBateriaCard">
                    <h3>Corriente Batería (A)</h3>
                    <p id="corrienteBateria">0.000</p>
                </div>

                <div class="card" id="voltajeBateriaCard">
                    <h3>Voltaje Batería (V)</h3>
                    <p id="voltajeBateria">0.000</p>
                </div>
            </div>
        </div>
    </div>

```

```

<div class="card" id="temperaturaCard">
    <h3>Temperatura (°C)</h3>
    <p id="temperatura">0.00</p>
</div>

<!-- Nuevas tarjetas para potencia y energía -->
<div class="card" id="potenciaCard">
    <h3>Potencia Actual (W)</h3>
    <p id="potencia">0.00</p>
</div>

<div class="card" id="energiaDiaCard">
    <h3>Energía Generada Hoy (Wh)</h3>
    <p id="energiaDia">0.00</p>
</div>

<div class="card" id="energiaMesCard">
    <h3>Energía Generada Este Mes (kWh)</h3>
    <p id="energiaMes">0.00</p>
</div>

<div class="card" id="energiaAnioCard">
    <h3>Energía Generada Este Año (kWh)</h3>
    <p id="energiaAnio">0.00</p>
</div>
</div>

<!-- Gráfico de área -->
<div id="grafico" class="chart-container" style="display: none;">
    <canvas id="sensorChart"></canvas>
</div>
</div>

<!-- Pie de página -->
<footer>
    <p>Desarrollado por Grupo_6 - Todos los derechos reservados</p>
</footer>

<script>
    const ESP32_IP = "http://192.168.4.1";
    const ctx = document.getElementById("sensorChart").getContext("2d");

    // Datos recolectados
    let datosRecolectados = [];

    // Se asume que el panel solar en Quito recibe aproximadamente 12 horas de luz solar
    // diarias
    const horasLuzSolar = 12;

    // Rangos de validación para el panel
    const corrienteDesconectadaMin = 0.011; // Por debajo, indica desconexión
    const voltajeDesconectadaMin = 2;
    const corrienteDesconectadaMax = 10; // Por encima, datos erróneos
    const voltajeDesconectadaMax = 60;
    // Rango para baja carga (pero aún en operación)
    const corrienteBajaCarga = 0.0118;
    const voltajeBajaCarga = 2;

    // Contador de registros no válidos (para la advertencia cada 5 registros)

```

```

let invalidCount = 0;

// Inicializar el gráfico de área
let sensorChart = new Chart(ctx, {
    type: 'line',
    data: {
        labels: [],
        datasets: [
            { label: "Corriente Panel (A)", borderColor: "#FFA726", backgroundColor:
"rgba(255, 167, 38, 0.2)", data: [], fill: true },
            { label: "Voltaje Panel (V)", borderColor: "#4CAF50", backgroundColor: "rgba(76,
175, 80, 0.2)", data: [], fill: true },
            { label: "Corriente Batería (A)", borderColor: "#2196F3", backgroundColor:
"rgba(33, 150, 243, 0.2)", data: [], fill: true },
            { label: "Voltaje Batería (V)", borderColor: "#9C27B0", backgroundColor:
"rgba(156, 39, 176, 0.2)", data: [], fill: true },
            { label: "Temperatura (°C)", borderColor: "#F44336", backgroundColor:
"rgba(244, 67, 54, 0.2)", data: [], fill: true }
        ]
    },
    options: {
        responsive: true,
        maintainAspectRatio: false,
        scales: {
            y: { beginAtZero: true }
        },
        plugins: {
            legend: {
                labels: {
                    font: { size: 14 }
                }
            }
        },
        interaction: {
            mode: 'nearest',
            intersect: false
        }
    }
});

// Función para mostrar el dashboard
function mostrarDashboard() {
    document.querySelector(".welcome-screen").style.display = "none";
    document.querySelector(".dashboard").style.display = "block";
    mostrarTablas();
}

// Función para mostrar las tablas
function mostrarTablas() {
    document.getElementById("tablas").style.display = "grid";
    document.getElementById("grafico").style.display = "none";
}

// Función para mostrar el gráfico
function mostrarGrafico() {
    document.getElementById("tablas").style.display = "none";
    document.getElementById("grafico").style.display = "block";
}

// Función para calcular la energía generada

```

```

function calcularEnergia(corrientePanel, voltajePanel) {
    const potenciaInstantanea = corrientePanel * voltajePanel; // Potencia en Watts (W)
    const energiaDiaria = potenciaInstantanea * horasLuzSolar; // Energía en Wh
    const energiaMensual = energiaDiaria * 30 / 1000; // Energía en kWh para un mes
    const energiaAnual = energiaMensual * 12; // Energía en kWh para un año
    return {
        potencia: potenciaInstantanea.toFixed(2),
        energiaDia: energiaDiaria.toFixed(2),
        energiaMes: energiaMensual.toFixed(2),
        energiaAnio: energiaAnual.toFixed(2)
    };
}

// Función para actualizar los datos
function actualizarDatos() {
    fetch(ESP32_IP)
        .then(response => response.json())
        .then(datos => {
            // Procesar datos del panel
            let corrientePanel = Math.abs(datos.corriente_panel) / 100;
            let voltajePanel = Math.abs(datos.voltaje_corriente_panel);

            // Actualizar datos de batería y temperatura (siempre se actualizan)
            document.getElementById("corrienteBateria").textContent =
                (Math.abs(datos.corriente_bateria) / 100).toFixed(3);
            document.getElementById("voltajeBateria").textContent =
                Math.abs(datos.voltaje_corriente_bateria).toFixed(3);
            document.getElementById("temperatura").textContent =
                Math.abs(datos.temperatura).toFixed(2);

            // Variable para almacenar el estado del panel
            let alertaMsg = "";
            let actualizarGraficoPanel = true; // Bandera para actualizar datos del panel en el
gráfico

            // Validación de panel: condición de desconexión o datos defectuosos
            if (
                corrientePanel < corrienteDesconectadaMin ||
                voltajePanel < voltajeDesconectadaMin ||
                corrientePanel > corrienteDesconectadaMax ||
                voltajePanel > voltajeDesconectadaMax
            ) {
                invalidCount++; // Incrementar contador de registros no válidos
                // Solo mostrar la advertencia cada 5 registros no válidos
                if (invalidCount % 5 === 0) {
                    alertaMsg = "⚠️ Panel desconectado o defectuoso";
                } else {
                    alertaMsg = "";
                }
            }

            // Se asignan valores 0 para fines de cálculo y se muestran '---' en la tabla
            corrientePanel = 0;
            voltajePanel = 0;
            document.getElementById("corrientePanel").textContent = "---";
            document.getElementById("voltajePanel").textContent = "---";
            document.getElementById("potencia").textContent = "---";
            document.getElementById("energiaDia").textContent = "---";
            document.getElementById("energiaMes").textContent = "---";
            document.getElementById("energiaAnio").textContent = "---";
            actualizarGraficoPanel = false; // No se actualiza el gráfico con estos datos
        });
}

```

```

    // Validación de baja carga: datos por debajo del rango esperado pero sin ser
defectuosos
    else if (corrientePanel < corrienteBajaCarga || voltajePanel < voltajeBajaCarga) {
        // Al recibir datos válidos, reiniciamos el contador de registros no válidos
        invalidCount = 0;
        alertaMsg = "⚠️ Panel con baja carga (datos fuera del rango válido)";
        const { potencia, energiaDia, energiaMes, energiaAnio } =
calcularEnergia(corrientePanel, voltajePanel);
        document.getElementById("corrientePanel").textContent =
corrientePanel.toFixed(3);
        document.getElementById("voltajePanel").textContent =
voltajePanel.toFixed(3);
        document.getElementById("potencia").textContent = potencia;
        document.getElementById("energiaDia").textContent = energiaDia;
        document.getElementById("energiaMes").textContent = energiaMes;
        document.getElementById("energiaAnio").textContent = energiaAnio;
    }
    // Operación normal
else {
    // Datos válidos: reiniciamos el contador de registros no válidos
    invalidCount = 0;
    alertaMsg = "";
    const { potencia, energiaDia, energiaMes, energiaAnio } =
calcularEnergia(corrientePanel, voltajePanel);
    document.getElementById("corrientePanel").textContent =
corrientePanel.toFixed(3);
    document.getElementById("voltajePanel").textContent =
voltajePanel.toFixed(3);
    document.getElementById("potencia").textContent = potencia;
    document.getElementById("energiaDia").textContent = energiaDia;
    document.getElementById("energiaMes").textContent = energiaMes;
    document.getElementById("energiaAnio").textContent = energiaAnio;
}
// Mostrar alerta en el dashboard
document.getElementById("panelAlert").textContent = alertaMsg;

// Guardar datos recolectados
datosRecolectados.push({
    fecha: new Date().toLocaleString(),
    corrientePanel: corrientePanel,
    voltajePanel: voltajePanel,
    corrienteBateria: (Math.abs(datos.corriente_bateria) / 100),
    voltajeBateria: Math.abs(datos.voltaje_corriente_bateria),
    temperatura: Math.abs(datos.temperatura),
    potencia: (actualizarGraficoPanel ? calcularEnergia(corrientePanel,
voltajePanel).potencia : "0.00"),
    energiaDia: (actualizarGraficoPanel ? calcularEnergia(corrientePanel,
voltajePanel).energiaDia : "0.00"),
    energiaMes: (actualizarGraficoPanel ? calcularEnergia(corrientePanel,
voltajePanel).energiaMes : "0.00"),
    energiaAnio: (actualizarGraficoPanel ? calcularEnergia(corrientePanel,
voltajePanel).energiaAnio : "0.00")
});

// Actualizar gráfico
let now = new Date().toLocaleTimeString();
sensorChart.data.labels.push(now);
if (sensorChart.data.labels.length > 10) {
    sensorChart.data.labels.shift();
}

```

```

        }
        if (actualizarGraficoPanel) {
            sensorChart.data.datasets[0].data.push(corrientePanel.toFixed(3));
            sensorChart.data.datasets[1].data.push(voltagjePanel.toFixed(3));
        } else {
            // Si el panel está desconectado/defectuoso, se actualiza el gráfico con 0 para
            representar la ausencia de datos
            sensorChart.data.datasets[0].data.push("0.000");
            sensorChart.data.datasets[1].data.push("0.000");
        }
        // Actualizar los datos de batería y temperatura en el gráfico
        sensorChart.data.datasets[2].data.push((Math.abs(datos.corriente_bateria) /
100).toFixed(3));

sensorChart.data.datasets[3].data.push(Math.abs(datos.voltagje_corriente_bateria).toFixed(3));
sensorChart.data.datasets[4].data.push(Math.abs(datos.temperatura).toFixed(2));

        // Limitar el número de puntos en el gráfico
        sensorChart.data.datasets.forEach(dataset => {
            if (dataset.data.length > 10) {
                dataset.data.shift();
            }
        });
        sensorChart.update();
    })
    .catch(error => console.error("Error obteniendo datos:", error));
}

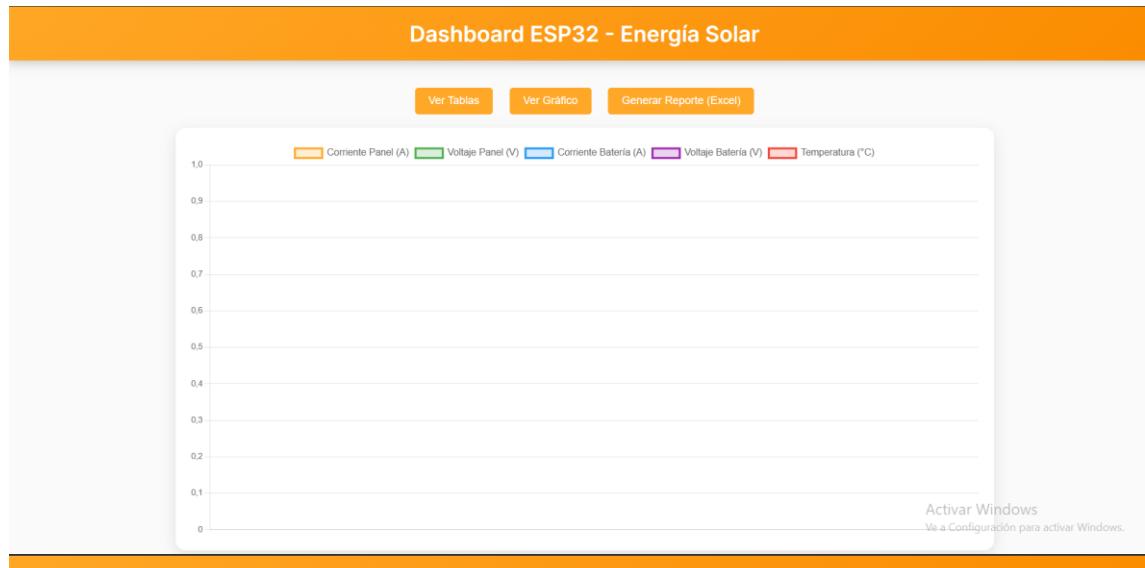
// Función para generar reporte en Excel
function generarReporteExcel() {
    const ws = XLSX.utils.json_to_sheet(datosRecolectados);
    const wb = XLSX.utils.book_new();
    XLSX.utils.book_append_sheet(wb, ws, "Reporte");
    XLSX.writeFile(wb, "reporte_datos.xlsx");
}

// Actualizar datos cada 5 segundos
setInterval(actualizarDatos, 5000);
</script>
</body>
</html>

```

- **Capturas de Pantalla:**





- **Documentación de Librerías:**
- **Chart.js:** <https://www.chartjs.org/>
- **SheetJS:** <https://sheetjs.com/>

Manual Modelo Digital

Introducción

Este documento describe el proceso de desarrollo y funcionamiento del gemelo digital de un sistema basado en ESP32, diseñado en SketchUp y exportado a Unity. El gemelo digital permite visualizar datos en tiempo real provenientes de sensores de corriente y temperatura conectados al ESP32.

Objetivos

Crear una representación digital exacta del sistema físico.

Visualizar en Unity los datos en tiempo real obtenidos del ESP32. Simular la interacción de luz solar con el sistema.

Herramientas Utilizadas

SketchUp: Diseño del modelo 3D.

Unity: Creación del gemelo digital e integración con ESP32. C#: Programación de la comunicación con ESP32.

ESP32: Microcontrolador para adquisición de datos.

Sensores: Dos sensores de corriente y un sensor de temperatura.

Desarrollo del Modelo

- Diseño en SketchUp

Se modeló el sistema físico, incluyendo panel solar, baterías, sensores y conexiones eléctricas.

Se exportó el modelo a un formato compatible con Unity.

- Importación a Unity

Se creó un entorno 3D que incluye el modelo importado, una superficie base y una fuente de luz.

Se implementó una animación para simular los rayos solares.

- Integración con ESP32

El ESP32 actúa como un servidor que transmite datos en formato JSON a través de una red WiFi.

Unity accede a estos datos mediante solicitudes HTTP.

Funcionamiento del Código en Unity ESP32DataFetcher.cs

1. Obtiene datos del ESP32 cada 2 segundos.
2. Verifica la validez de los datos antes de mostrarlos en pantalla.
3. Muestra valores en la UI de Unity.

Código:

```
using UnityEngine;
using UnityEngine.Networking; using UnityEngine.UI;
using System.Collections;
using TMPro; // Necesario para TextMeshPro
public class ESP32DataFetcher : MonoBehaviour
{
    public TextMeshProUGUI panelCorrienteText, panelVoltajeText, bateriaCorrienteText,
    bateriaVoltajeText, temperaturaText;
    private string url = "http://192.168.4.1"; // IP del ESP32 en modo Access Point
    private const float MIN_CORRIENTE = 0.011f, MAX_CORRIENTE = 20.0f;
    private const float MIN_VOLTAJE = 2.3f, MAX_VOLTAJE = 50.0f;
    void Start()
    {
        InvokeRepeating("GetESP32Data", 2.0f, 2.0f); // Llama la función cada 2 segundos
    }
    void GetESP32Data()
    {
        StartCoroutine(GetRequest(url));
    }

    IEnumerator GetRequest(string uri)
```

```
{  
    using (UnityWebRequest webRequest = UnityWebRequest.Get(uri))  
    {  
        yield return webRequest.SendWebRequest();  
        if (webRequest.result == UnityWebRequest.Result.Success)  
        {  
            string jsonData = webRequest.downloadHandler.text; Debug.Log("JSON recibido: " +  
            jsonData); // Depuración ProcessData(jsonData);  
        }  
        else  
        {  
            Debug.LogError("Error en la conexión: " + webRequest.error);  
        }  
    }  
}  
  
void ProcessData(string json)  
{  
    ESP32Data data = JsonUtility.FromJson<ESP32Data>(json); Debug.Log($"Corriente Panel sin  
procesar: {data.corriente_panel}"); Debug.Log($"Corriente Batería sin procesar:  
{data.corriente_bateria}"); float corrientePanel = Mathf.Abs(data.corriente_panel) / 100.0f;  
float corrienteBateria = Mathf.Abs(data.corriente_bateria) / 100.0f; float voltajePanel =  
Mathf.Abs(data.voltaje_corriente_panel);  
float voltajeBateria = Mathf.Abs(data.voltaje_corriente_bateria); float temperatura =  
Mathf.Abs(data.temperatura);
```

```
bool datosValidos = (corrientePanel >= MIN_CORRIENTE CC corrientePanel <=
MAX_CORRIENTE CC
voltajePanel >= MIN_VOLTAJE CC voltajePanel <= MAX_VOLTAJE); if (datosValidos)
{
panelCorrienteText.text = $"Corriente Panel: {corrientePanel:F2} A"; panelVoltajeText.text =
$"Voltaje Panel: {voltajePanel:F2} V"; bateriaCorrienteText.text = $"Corriente Batería:
{corrienteBateria:F2} A"; bateriaVoltajeText.text = $"Voltaje Batería: {voltajeBateria:F2} V";
}
else
{
panelCorrienteText.text = "Corriente Panel: ---"; panelVoltajeText.text = "Voltaje Panel: ---";
bateriaCorrienteText.text = "Corriente Batería: ---"; bateriaVoltajeText.text = "Voltaje Batería: -
---";
}
temperaturaText.text = $"Temperatura: {temperatura:F2} °C";
}
}

[System.Serializable]
public class ESP32Data
{
public float corriente_panel;
public float voltaje_corriente_panel; public float corriente_bateria;
public float voltaje_corriente_bateria; public float temperatura;
```

```
}
```

NewBehaviourScript.cs

1. Realiza peticiones HTTP al ESP32 cada 5 segundos.
2. Convierte la respuesta JSON en datos utilizables dentro de Unity.
3. Actualiza los valores de corriente, voltaje y temperatura en la UI.

Código:

```
using UnityEngine;
using UnityEngine.Networking; using TMPro;
using System.Collections;
public class ESP32DataFetcher1 : MonoBehaviour
{
    public TextMeshProUGUI panelCorrienteText; public TextMeshProUGUI panelVoltajeText;
    public TextMeshProUGUI bateriaCorrienteText; public TextMeshProUGUI bateriaVoltajeText;
    public TextMeshProUGUI temperaturaText;
    private string ESP32_IP = "http://192.168.4.1"; // IP del ESP32 void Start()
    {
        panelCorrienteText.text = "Corriente Panel: Esperando datos..."; panelVoltajeText.text =
        "Voltaje Panel: Esperando datos..."; bateriaCorrienteText.text = "Corriente Batería: Esperando
        datos..."; bateriaVoltajeText.text = "Voltaje Batería: Esperando datos..."; temperaturaText.text =
        "Temperatura: Esperando datos...";

        StartCoroutine(ActualizarDatos());
    }
```

```
IEnumerator ActualizarDatos()
{
    while (true)
    {
        using (UnityWebRequest request = UnityWebRequest.Get(ESP32_IP))
        {
            yield return request.SendWebRequest();
            if (request.result == UnityWebRequest.Result.Success)
            {
                string jsonResponse = request.downloadHandler.text;
                ESP32Data datos = JsonUtility.FromJson<ESP32Data>(jsonResponse);
                // Actualizar los textos en la UI
                panelCorrienteText.text = $"Corriente Panel: {datos.corriente_panel / 100:F3} A";
                panelVoltajeText.text = $"Voltaje Panel:
                {datos.voltaje_corriente_panel:F2} V";
                bateriaCorrienteText.text = $"Corriente Batería: {datos.corriente_bateria / 100:F3} A";
                bateriaVoltajeText.text = $"Voltaje Batería:
                {datos.voltaje_corriente_bateria:F2} V";
                temperaturaText.text = $"Temperatura: {datos.temperatura:F2} °C";
            }
            else
            {
                Debug.LogError("Error al conectar con ESP32: " + request.error);
            }
        }
        yield return new WaitForSeconds(5); // Actualizar cada 5 segundos
    }
}
```

```
}

[System.Serializable]
public class ESP32Data
{
    public float corriente_panel;
    public float voltaje_corriente_panel; public float corriente_bateria;
    public float voltaje_corriente_bateria; public float temperatura;
}
```

Visualización en Unity

Se utilizan elementos UI (Canvas y TextMeshPro) para mostrar los datos en tiempo real. Los valores de corriente, voltaje y temperatura se actualizan automáticamente. Se representa la interacción de la luz solar con el sistema.

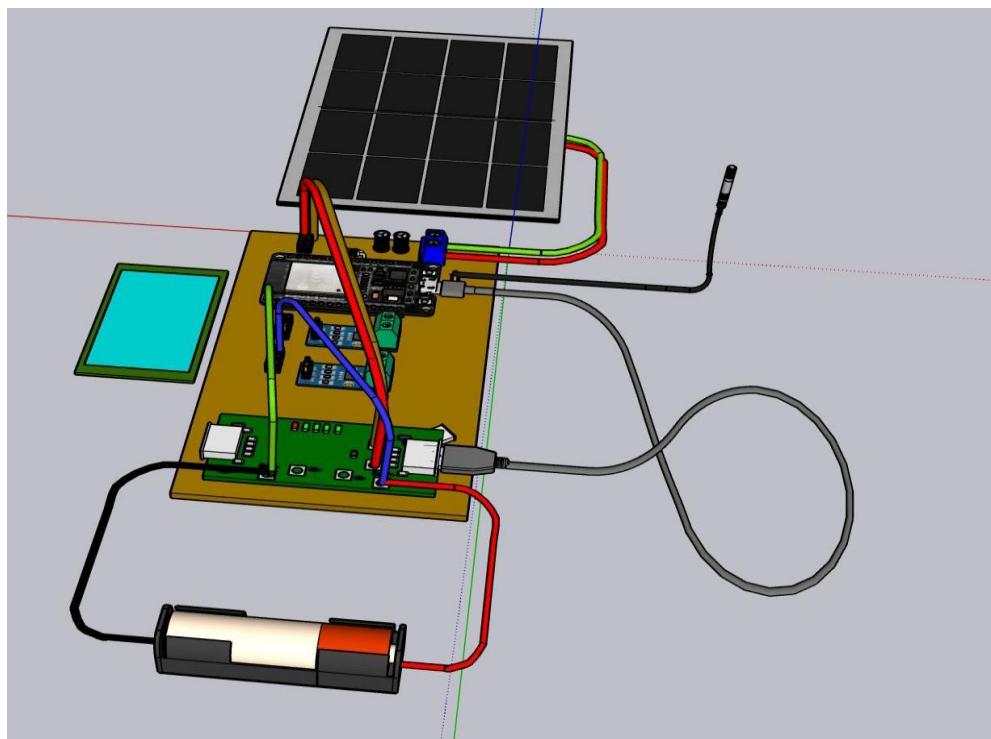
Anexos

Conclusión

Este gemelo digital permite monitorear el sistema en tiempo real y proporciona una interfaz intuitiva para la visualización de datos. La integración con ESP32 y la representación gráfica en Unity hacen de este modelo una herramienta efectiva para análisis y simulación

ANEXOS:

Modelo en SketchUP:



Modelo en Unity:

