Instituto Tecnológico Nacional de México
Instituto Tecnológico de Colima

Materia: APPS PARA EL INTERNET DE LAS COSAS

Docente: ANGEL VEJAR CORTES

Carrera: Ingeniería en informática

Equipo:

Daniel Ezequiel Alvarado Almánzar

Christian Rafael Legorreta Avalos

José Salvador Orozco Gutiérrez

Rogelio Valdez Macías

Diego Rodolfo Diaz Morfin

Martes 10 de octubre del 2023

INDICE

Introducción

En este extenso desarrollo de proyecto, se podrá observar la implementación de un sistema que involucra la comunicación entre un servidor y un ESP32, con el propósito de controlar un LED, monitorizar la temperatura y humedad, y llevar un contador. Para comprender mejor este proyecto, se han organizado las explicaciones en varias secciones clave que mostrarán lo siguiente:

En la sección de Configuración del Servidor, se detallarán las bibliotecas importadas, como BaseHTTPRequestHandler y HTTP Server, las cuales son esenciales para gestionar solicitudes HTTP y configurar el servidor. También se mencionarán las bibliotecas json y os, utilizadas para manejar datos JSON y trabajar con el sistema de archivos.

Las Variables Globales permitirán almacenar datos simulados, incluyendo un contador, el estado del LED (encendido o apagado), así como lecturas simuladas de temperatura y humedad.

Dentro del Manejo de Solicitudes HTTP, se explicará el método para establecer las cabeceras de respuesta HTTP, incluyendo el tipo de contenido y el código de estado. Además, se mostrará cómo se utiliza el método GET para manejar solicitudes, imprimiendo la ruta solicitada y haciendo referencia al uso de la variable global "led".

El Control del Estado del LED mostrará las líneas de código necesarias para devolver el estado del LED, encenderlo o apagarlo, y responder con el nuevo estado en formato JSON.

En la Gestión de Datos de Temperatura y Humedad, se podrán visualizar las explicaciones sobre cómo se responderá con lecturas simuladas de temperatura y humedad en formato JSON, así como cómo se gestionará la respuesta en caso de un error, como una ruta no válida.

La sección dedicada al Manejo de Datos POST - Contador detallará la verificación de la presencia de "action" y "quantity" en los datos JSON, respondiendo con un error si alguno de estos elementos falta, asegurando que "quantity" sea un número entero válido.

Para la Configuración y Ejecución del Servidor, se expondrá la función que se empleará para ejecutar el servidor, asignando automáticamente el puerto 7800. Las líneas finales demostrarán la ejecución exitosa del servidor.

El Código ESP32 exhibirá las bibliotecas necesarias para el funcionamiento del ESP32, incluyendo las bibliotecas para el sensor DHT, la conexión Wi-Fi, solicitudes HTTP y manipulación de JSON. Se explicará cómo se configurará la red Wi-Fi con el nombre y la contraseña.

Las Solicitudes HTTP desde el ESP32 mostrarán cómo se llevarán a cabo las solicitudes HTTP desde el ESP32, incluyendo la asignación de pines, tiempos de espera y la configuración inicial del contador.

Para el Control del LED desde el Servidor Remoto, se describirá cómo se controlará el LED de forma remota, cómo se leerá el JSON de respuesta y cómo se actualizará el estado del LED en la placa del Arduino.

El Envío de Datos al Servidor Remoto revelará cómo se enviarán datos al servidor remoto y se detallará la función para transmitir datos de aumento y disminución al servidor.

El Control de Botones y Contador explicará cómo se detectará el estado de los botones, cómo se incrementará el contador al presionarlos y cómo se enviarán los datos al servidor. También se mencionará cómo se actualizará el valor del contador en la interfaz.

La Lectura de Temperatura y Humedad mostrará cómo se obtendrán las lecturas de humedad y temperatura del sensor DHT y cómo se enviarán al servidor.

En lo que respecta a los Estilos de Página HTML, se describirán los estilos aplicados a la página web, incluyendo el centrado de elementos y estilos para títulos y botones.

Dentro del apartado Índex - Estilos de Página, se detallarán los estilos aplicados al cuerpo de la página, títulos y botones, con referencias a los botones "ON" y "OFF".

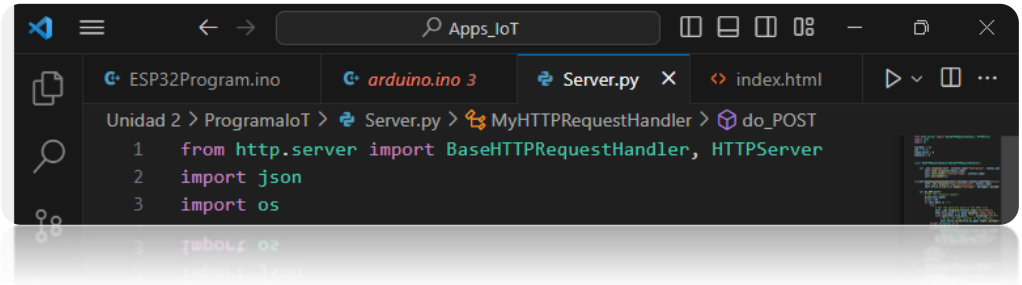
Finalmente, en la sección de Actualización de Elementos en la Interfaz, se explicarán las funciones utilizadas para actualizar elementos HTML en la interfaz, como el contador, la temperatura y la humedad, así como la frecuencia con la que se ejecutarán estas actualizaciones.

Este completo desarrollo de proyecto permitirá a los interesados observar de manera detallada la implementación de este sistema, desde la configuración del servidor hasta la interacción con el ESP32 y la interfaz web.

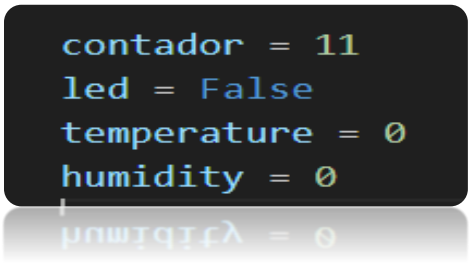
Desarrollo

Servidor:

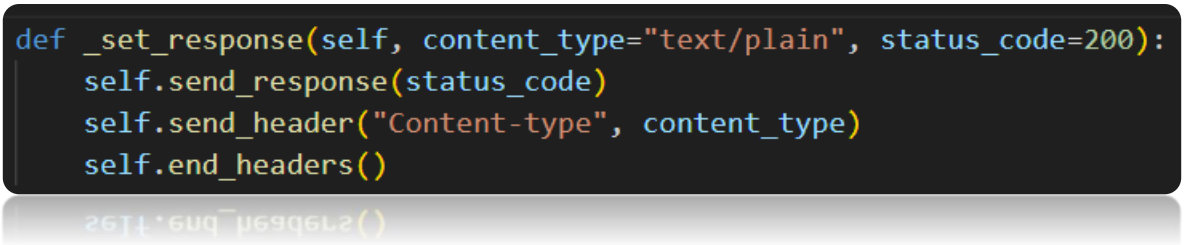
Esta parte del código es donde importamos las bibliotecas las cuales BaseHTTPRequestHandler y HTTP Server se importan desde la biblioteca http. Server para manejar solicitudes HTTP y configurar el servidor, respectivamente, json se utiliza para manipular datos JSON y os se utiliza para trabajar con el sistema de archivos y abrir el archivo HTML.



En esta parte definimos las variables globales las cuales son para almacenar datos simulados como un contador, el estado del led si este encendido o apagado, temperatura y humedad



Este método se utiliza para establecer las cabeceras de respuesta HTTP, incluyendo el tipo de contenido y el código de estado (Escribe el mensaje de error en formato json).



El método Get lo utilizamos para manejar las solicitudes, con el print imprime la ruta solicitada en la consola y global es la variable que utilizamos ‘led’

```
def do_GET(self):
    # Set the response headers
    print(self.path)
    global led
```

En esta parte del código tenemos las líneas de código donde en el primer elif devolvemos el estado del led

```
elif self.path == "/counter":
    self._set_response()
    self.wfile.write(json.dumps({"contador": contador}).encode())
```

En este responde con el estado del led en formato JSON

```
elif self.path == "/led":
    self._set_response()
    self.wfile.write(json.dumps({"status": led}).encode())
```

```
{"status": true}
```

El siguiente código activa el led y responde con el nuevo estado, al igual que apagarlo y responder con el estado

```
elif self.path == "/led/on":

    led = True
    self._set_response()
    self.wfile.write(json.dumps({"status": led}).encode())

elif self.path == "/led/off":

    led = False
    self._set_response()
    self.wfile.write(json.dumps({"status": led}).encode())
```

La siguiente imagen muestra la respuesta con la temperatura en formato JSON, al igual que el siguiente elif responde con la humedad en formato JSON y por último si mostrara un error mostrara Invalid path

```
elif self.path == "/temperature":

    self._set_response()
    self.wfile.write(json.dumps({"temperature": temperature}).encode())

elif self.path == "/humidity":

    self._set_response()
    self.wfile.write(json.dumps({"humidity": humidity}).encode())
else:
    # send bad request response
    self.throw_custom_error("Invalid path")
```

Con el siguiente fragmento de código lo que se realiza es Verifica si 'action' y 'quantity' están presentes en los datos JSON, si fuera el caso contrario, Envía una respuesta de error si falta 'action' o 'quantity', este apartado es parte de la variable global contador

```
# Check if action and quantity are present
if (body_json.get('action') is None or body_json.get('quantity') is None):
    self.throw_custom_error("Missing action or quantity")
    return

# Check if action is valid
if (body_json['action'] != 'asc' and body_json['action'] != 'desc'):
    self.throw_custom_error("Invalid action")
    return
```

Verificamos si 'quantity' es un número entero y si es así no habría problema alguno en caso contrario nos arrojaría una respuesta de error si 'quantity' no es válido

```
try:
    int(body_json['quantity'])
except:
    self.throw_custom_error("Invalid quantity")
    return
```

Con esta función es la que utilizamos para la ejecución de nuestro servidor el cual asigna el puerto 7800 automáticamente y por ultimo las líneas del código del final es la ejecución del servidor si el archivo se ejecuta correctamente

```
def run_server(server_class=HTTPServer, handler_class=MyHTTPRequestHandler
    server_address = ("", port)
    httpd = server_class(server_address, handler_class)
    print(f"Starting server on port {port}...")
    httpd.serve_forever()


if __name__ == "__main__":
    run_server()
```

Código ESP32

En esta parte del código son las bibliotecas necesarias para el correcto funcionamiento y evitar errores en el código sin estas bibliotecas el código no funcionaria, las cuales incluye la biblioteca para el sensor DHT, biblioteca para la conexión wifi, biblioteca para realizar solicitudes HTTP y por último la biblioteca para trabajar con JSON. La parte de los const char son la configuración del red wifi, donde colocamos el nombre y la contraseña de la red wifi

```
#include <DHT.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>

const char *ssid = "INFINITUME7D8";
const char *password = "kasumikasumi";
```

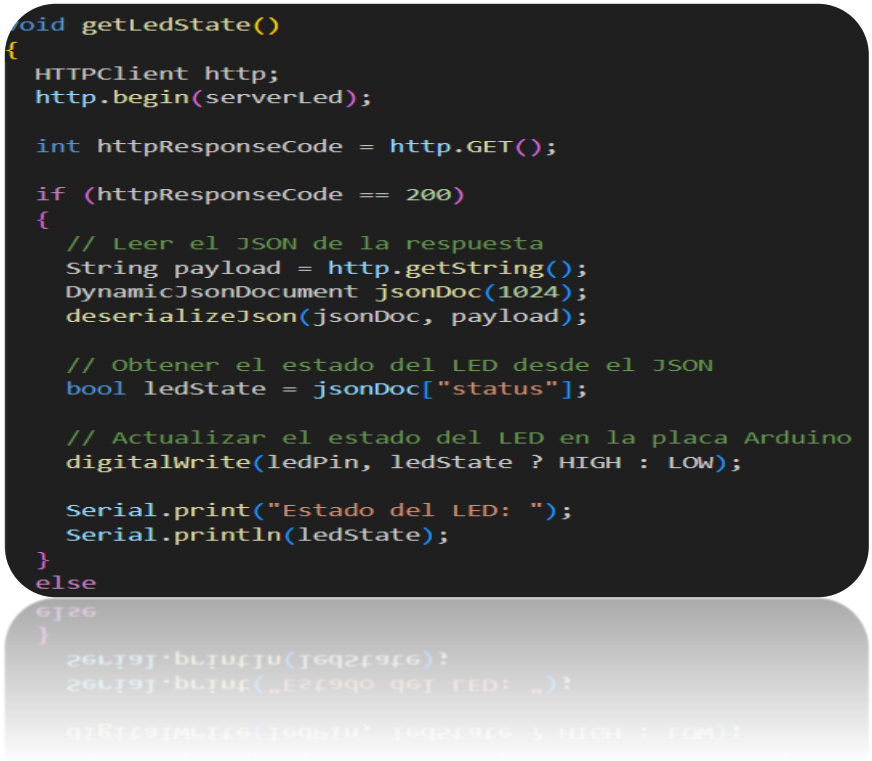
En esta parte tenemos las solicitudes de HTTP

```
String serverTemperatureURL = "http://192.168.1.73:7800/temperature";
String serverHumidityURL = "http://192.168.1.73:7800/humidity";
String serverName = "http://192.168.1.73:7800/";
String serverLed = "http://192.168.1.73:7800/led"; // Endpoint para obtener el estado del LED
```

La primera línea como podemos ver en la imagen nos muestra al pin al que está conectado en el scp32 el cual indicamos que es el 2, las siguientes líneas las cuales son unsigned son el tiempo de delay que tiene el led, el int del contador es la variable con la que se inició el contador el cual es 0, las siguientes líneas muestran el pin al cual está conectado lo cual es para aumentar o disminuir el contador dependiendo el botón que se pulse.

```
int ledPin = 2; // Pin al que está conectado el LED

unsigned long lastTime = 0;
unsigned long timerDelay = 5000;

int contador = 0; // Variable para el contador inicializada en 0

const int botonAumentarPin = 4; // Pin del botón para aumentar el contador
const int botonDisminuirPin = 5; // Pin del botón para disminuir el contador

#define DHTPIN 18
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);
```

Estas líneas de código son para la función del led desde el servidor de forma remota, en lo cual como podemos ver en la línea de código son tanto para leer el JSON de la respuesta como el obtener el estado del led desde el JSON y a su vez actualizar el estado del led en la placa del Arduino

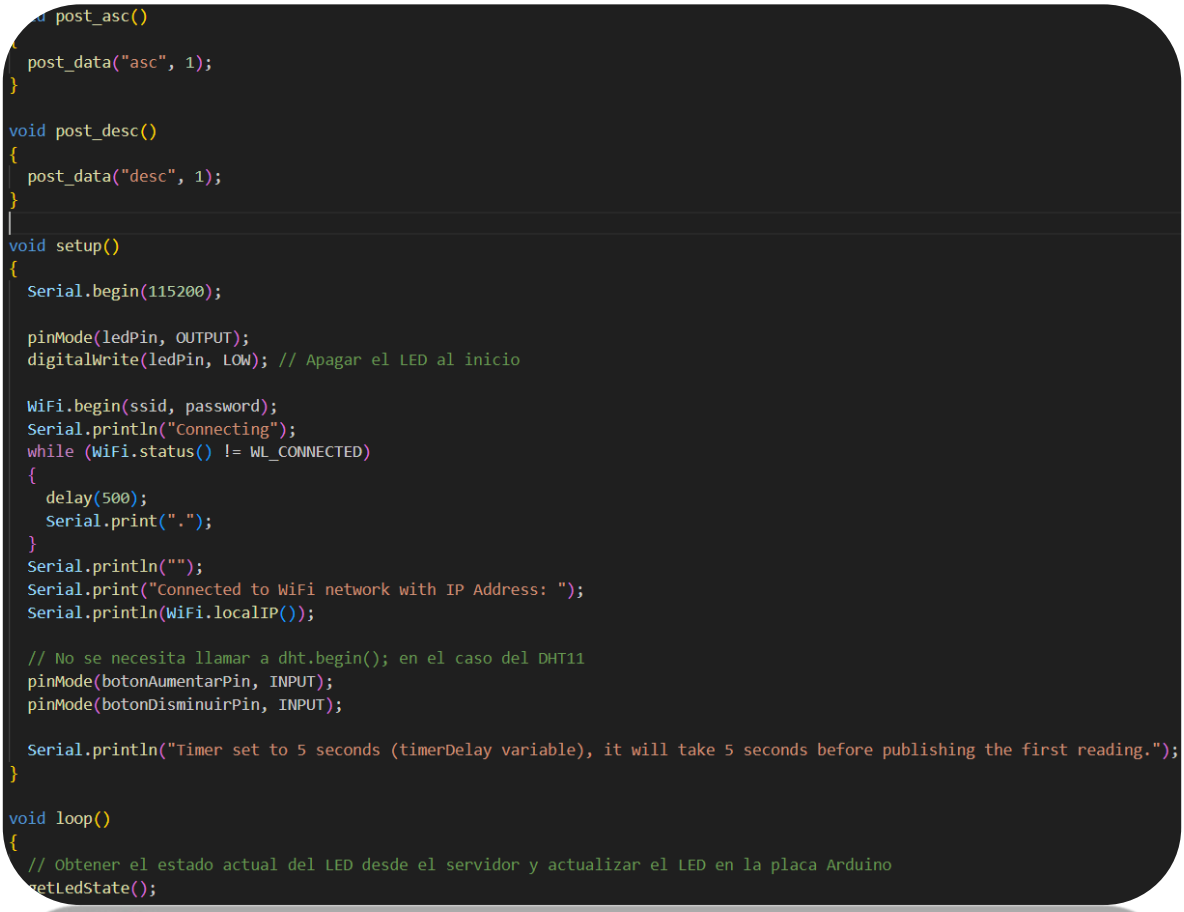


```
oid getLedState()
{
  HTTPClient http;
  http.begin(serverLed);

  int httpResponseCode = http.GET();

  if (httpResponseCode == 200)
  {
    // Leer el JSON de la respuesta
    String payload = http.getString();
    DynamicJsonDocument jsonDoc(1024);
    deserializeJson(jsonDoc, payload);

    // Obtener el estado del LED desde el JSON
    bool ledState = jsonDoc["status"];

    // Actualizar el estado del LED en la placa Arduino
    digitalWrite(ledPin, ledState ? HIGH : LOW);

    Serial.print("Estado del LED: ");
    Serial.println(ledState);
  }
  else
```

A continuación, tenemos el código el cual su función es enviar datos al servidor remoto

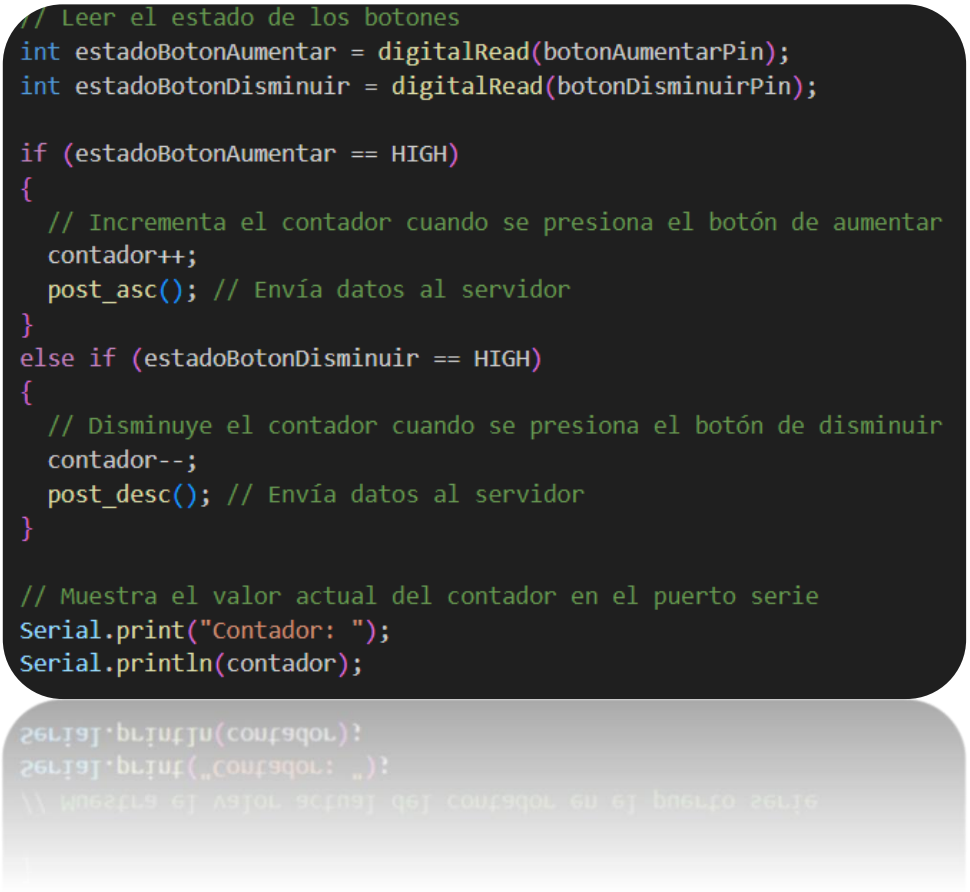
```
void post_data(String action, int quantity)
{
  DynamicJsonDocument json_chido(1024);
  json_chido["action"] = action;
  json_chido["quantity"] = quantity;

  String json_str;
  serializeJson(json_chido, json_str);

  HTTPClient http;
  http.begin(serverName);
  http.addHeader("Content-Type", "application/json");
  int httpResponseCode = http.POST(json_str);

  if (httpResponseCode > 0)
  {
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
    String payload = http.getString();
    Serial.println(payload);
  }
  else
  {
    Serial.print("Error code: ");
    Serial.println(httpResponseCode);
  }

  http.end();
}

void post_temperature(float temperature)
{
  DynamicJsonDocument json_doc(1024);
  json_doc["temperature"] = temperature;

  String json_str;
  serializeJson(json_doc, json_str);

  HTTPClient http;
```

En esta parte se muestra las funciones encargadas de enviar datos de aumento al servidor remoto y datos de disminución al mismo servidor. A su vez tenemos el serial para iniciar la comunicación y conectarse a una red wifi



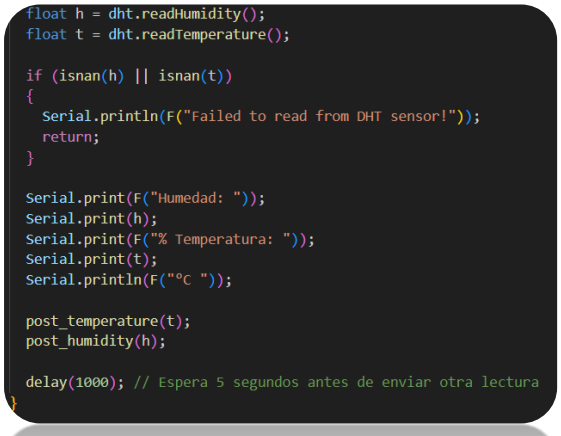
```
d post_asc()
{
  post_data("asc", 1);
}

void post_desc()
{
  post_data("desc", 1);
}

void setup()
{
  Serial.begin(115200);

  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW); // Apagar el LED al inicio

  WiFi.begin(ssid, password);
  Serial.println("Connecting");
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to WiFi network with IP Address: ");
  Serial.println(WiFi.localIP());

  // No se necesita llamar a dht.begin(); en el caso del DHT11
  pinMode(botonAumentarPin, INPUT);
  pinMode(botonDisminuirPin, INPUT);

  Serial.println("Timer set to 5 seconds (timerDelay variable), it will take 5 seconds before publishing the first reading.");
}

void loop()
{
  // Obtener el estado actual del LED desde el servidor y actualizar el LED en la placa Arduino
  etLedState();
```

En esta parte tenemos el código el cual muestra el estado de los botones, incrementar el contador cuando se presionen los botones en la protoboard los cuales se envían al servidor y muestra el valor actual del contador



```
// Leer el estado de los botones
int estadoBotonAumentar = digitalRead(botonAumentarPin);
int estadoBotonDisminuir = digitalRead(botonDisminuirPin);

if (estadoBotonAumentar == HIGH)
{
  // Incrementa el contador cuando se presiona el botón de aumentar
  contador++;
  post_asc(); // Envía datos al servidor
}
else if (estadoBotonDisminuir == HIGH)
{
  // Disminuye el contador cuando se presiona el botón de disminuir
  contador--;
  post_desc(); // Envía datos al servidor
}

// Muestra el valor actual del contador en el puerto serie
Serial.print("Contador: ");
Serial.println(contador);
```

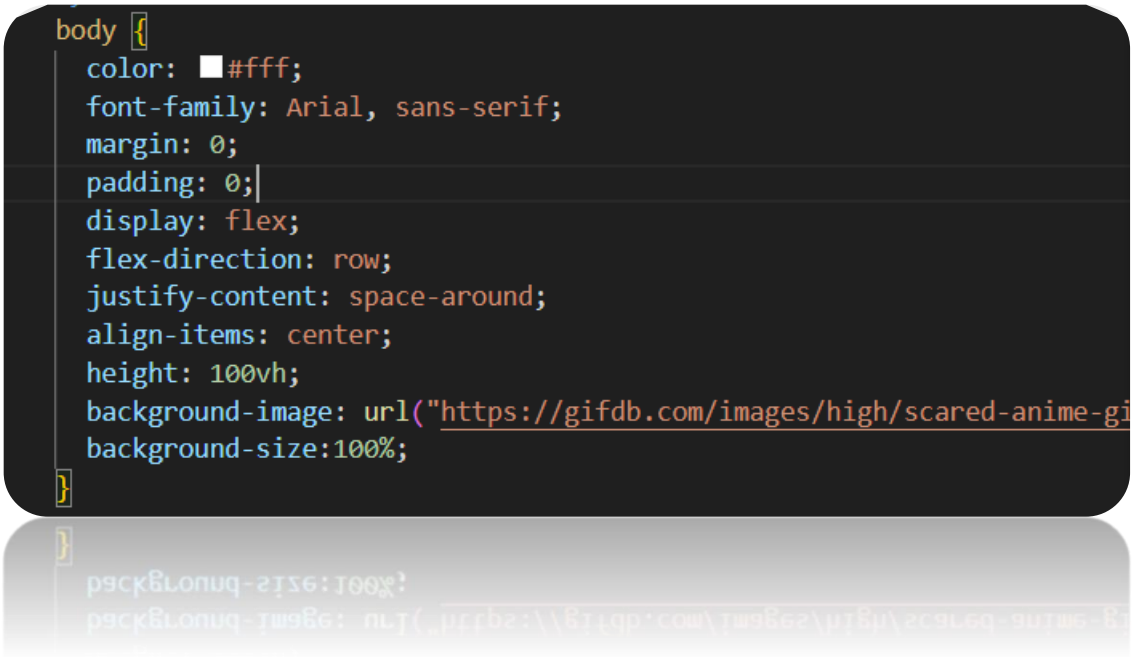
Por último, pero no menos importante tenemos la parte del código donde nos muestra la lectura de humedad y la temperatura del sensor DHT, la cual se envía al servidor



```
float h = dht.readHumidity();
float t = dht.readTemperature();

if (isnan(h) || isnan(t))
{
  Serial.println(F("Failed to read from DHT sensor!"));
  return;
}

Serial.print(F("Humedad: "));
Serial.print(h);
Serial.print(F("% Temperatura: "));
Serial.print(t);
Serial.println(F("°C "));

post_temperature(t);
post_humidity(h);

delay(1000); // Espera 5 segundos antes de enviar otra lectura
}
```

Índex

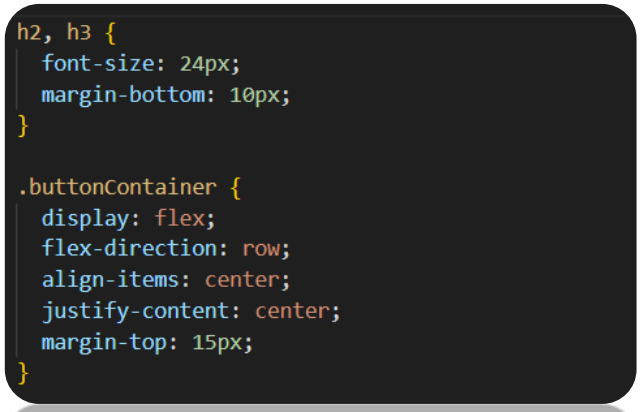
Son los estilos del cuerpo de la página desde el margen, si está centrado etc.



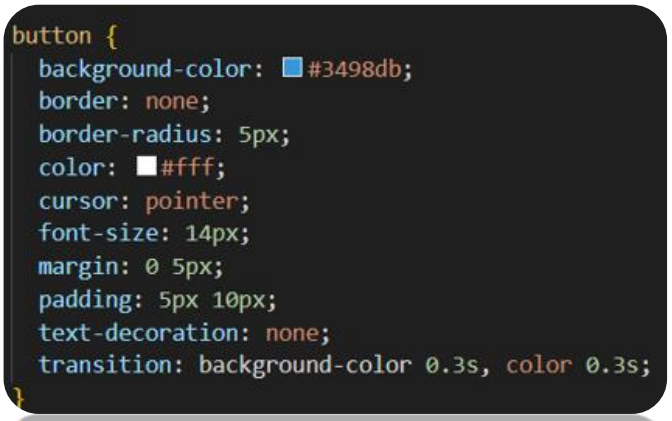
```css
body {
  color: #fff;
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
  display: flex;
  flex-direction: row;
  justify-content: space-around;
  align-items: center;
  height: 100vh;
  background-image: url("https://gifdb.com/images/high/scared-anime-gi
  background-size:100%;
}
```

En este código podemos ver los estilos de h2 y h3, al igual que los estilos para los botones

```
</body>
<script>
  const onButton = document.getElementById("onButton");
  const offButton = document.getElementById("offButton");

  onButton.addEventListener("click", () => {
    fetch("/led/on");
  });

  offButton.addEventListener("click", () => {
    fetch("/led/off");
  });

  const updateCounter = () => {
    fetch("/counter")
      .then((response) => response.json())
      .then((data) => {
        document.getElementById("counter").innerHTML = data.contador;
      });

    fetch("/temperature")
      .then((response) => response.json())
      .then((data) => {
        document.getElementById("temperature").innerHTML =
          data.temperature + " °C";
      });

    fetch("/humidity")
      .then((response) => response.json())
      .then((data) => {
        document.getElementById("humidity").innerHTML = data.humidity + " %";
      });
  };

  setInterval(updateCounter, 1000);
</script>
</html>
```



```css
h2, h3 {
  font-size: 24px;
  margin-bottom: 10px;
}

.buttonContainer {
  display: flex;
  flex-direction: row;
  align-items: center;
  justify-content: center;
  margin-top: 15px;
}
```

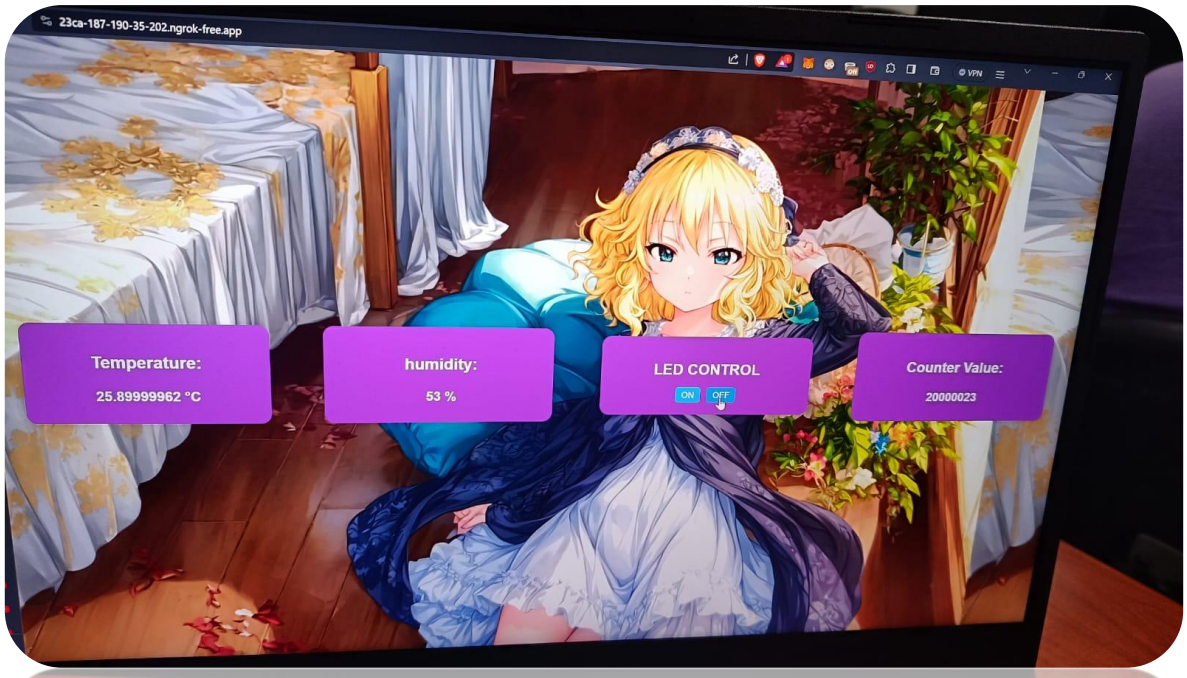


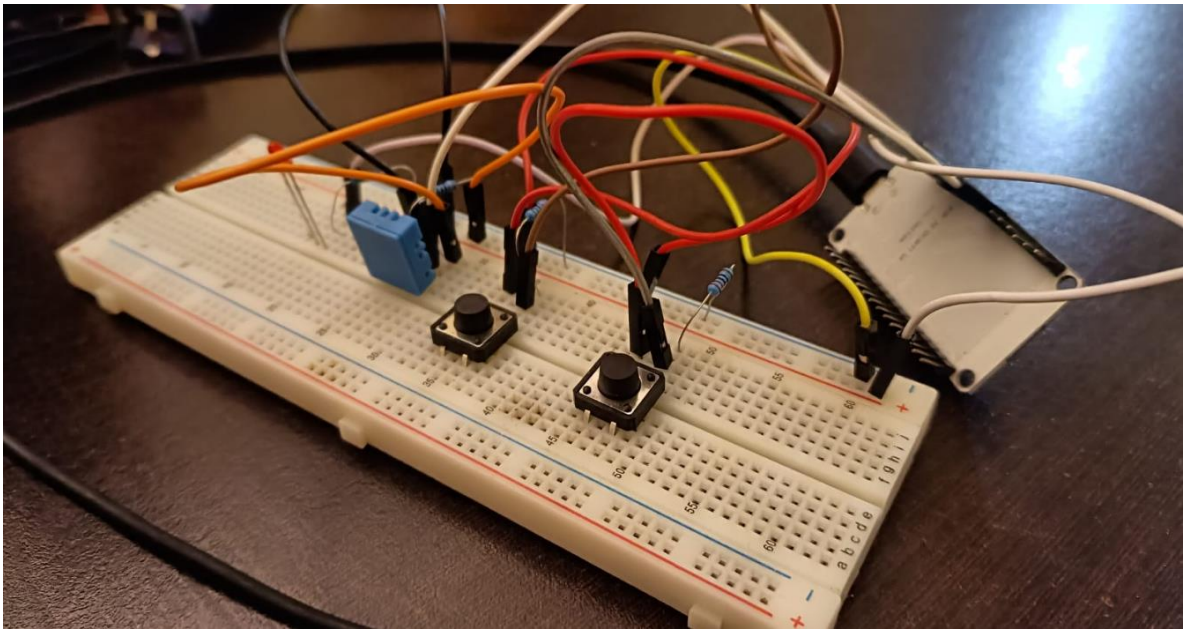
```css
button {
  background-color: #3498db;
  border: none;
  border-radius: 5px;
  color: #fff;
  cursor: pointer;
  font-size: 14px;
  margin: 0 5px;
  padding: 5px 10px;
  text-decoration: none;
  transition: background-color 0.3s, color 0.3s;
}
```

A continuación, y por último tenemos las siguientes líneas las cuales son las referencias de los botones on y off, seguidamente tenemos la solicitud fetch a “/led/on” al hacer clic en el botón “ON”, tenemos también las funciones para actualizar el contador, temperatura y humedad, actualizar los elementos del HTML con id “counter” con el valor del contador y Realizar solicitudes para obtener la humedad y llamar a la función “updatecounter” periódicamente cada 1000 milisegundos

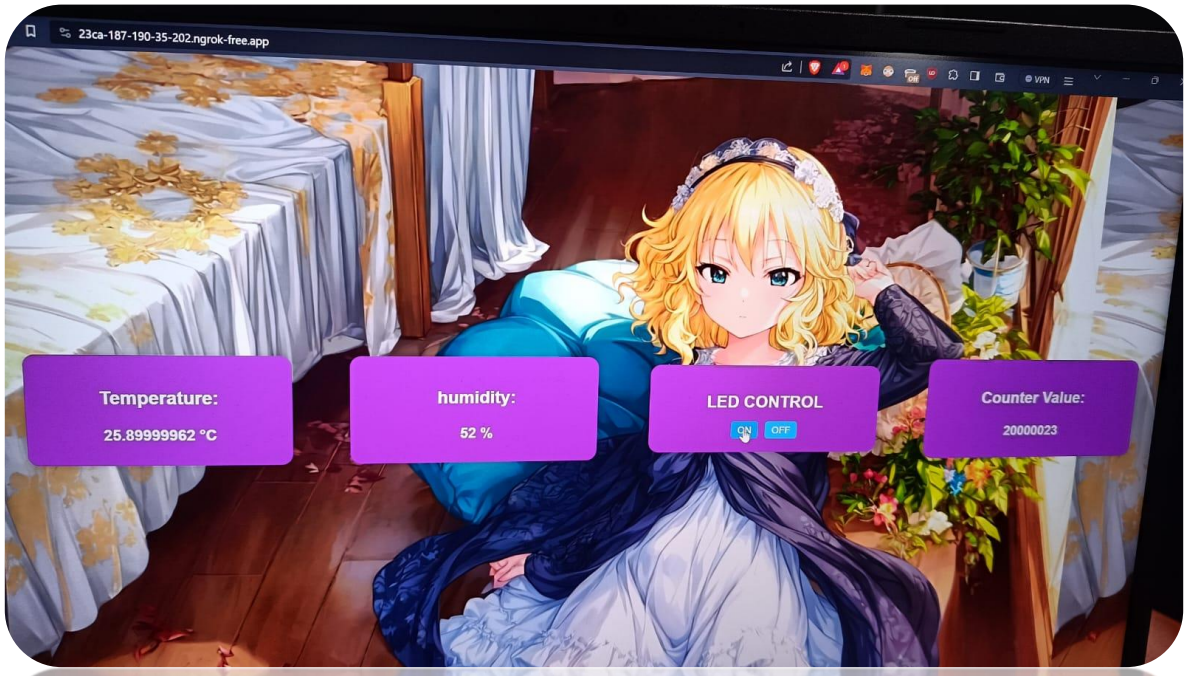
Funcionando:

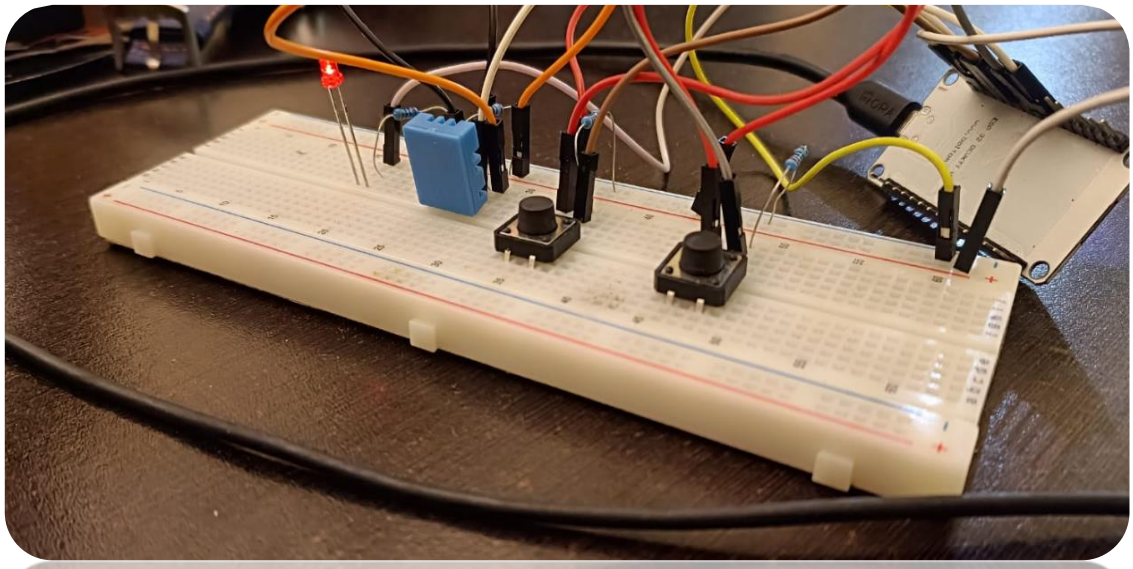
LED apagado





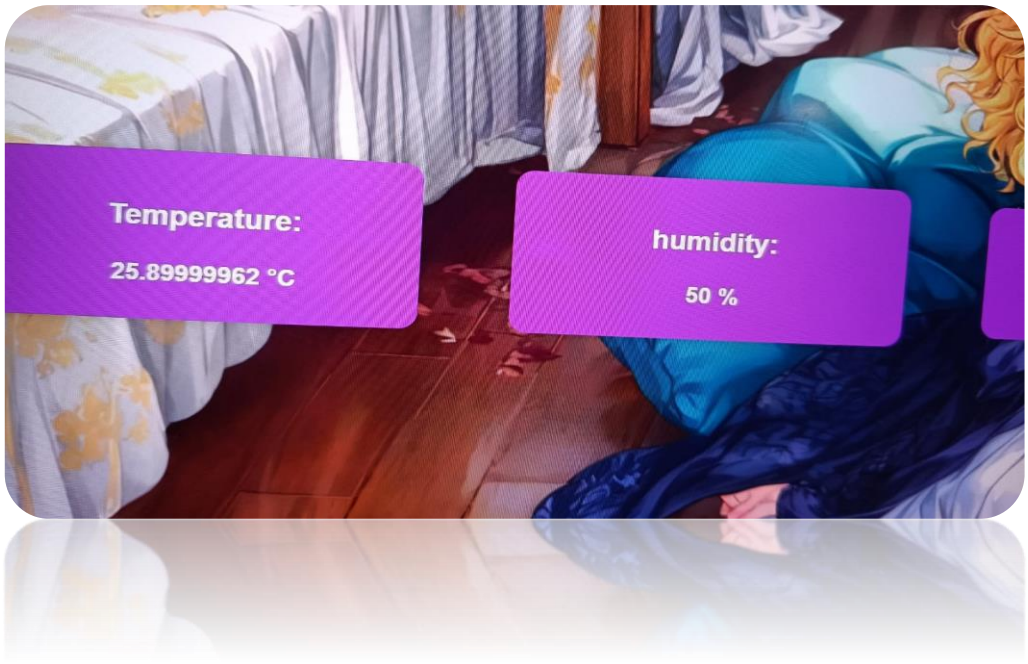
LED ENCENDIDO



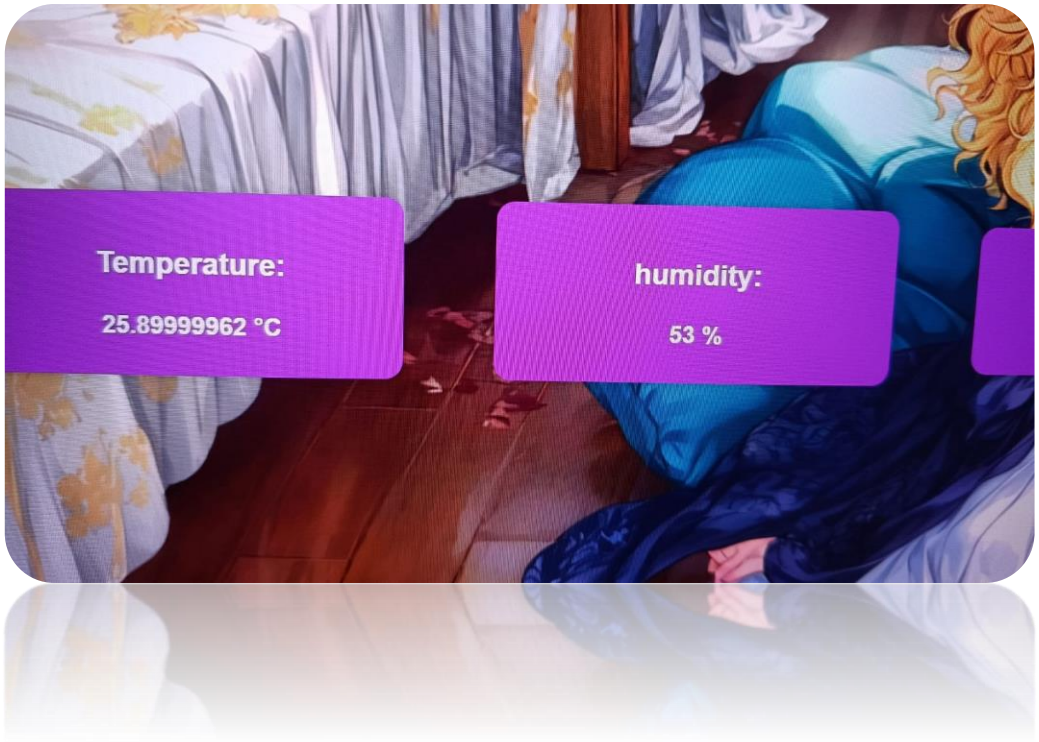


Temperatura y Humedad

Antes



Después



Conclusión

Rogelio Valdez Macias

El protocolo HTTP (Hypertext Transfer Protocol) tiene un impacto y una importancia enormes en mi experiencia en el desarrollo de aplicaciones para el Internet de las Cosas (IoT).

En mi opinión personal, desarrollar una aplicación de IoT ha sido una experiencia emocionante y desafiante. El campo de IoT implica conectar dispositivos físicos a través de Internet, lo que abre un mundo de posibilidades para mejorar la eficiencia y la comodidad en diversas áreas, desde el hogar inteligente hasta la industria y la salud.

Sin embargo, también he encontrado desafíos significativos. El desarrollo de aplicaciones de IoT puede ser complejo debido a la diversidad de dispositivos, protocolos y tecnologías involucrados. Además, la seguridad y la privacidad son aspectos críticos que siempre están en mi mente, ya que la interconexión de dispositivos conlleva riesgos potenciales de ciberseguridad.

En mi experiencia, elegir el protocolo adecuado, como HTTP u otros protocolos de IoT, es esencial para el éxito de una aplicación de IoT. También es fundamental abordar con cuidado los aspectos éticos y legales relacionados con la recopilación y el uso de datos personales en proyectos de IoT.

José Salvador Orozco Gutiérrez

Considero que el desarrollo de la actividad se ha hecho con éxito ya que se cumplió con los objetivos de la práctica. Al principio tuvimos varios problemas ya que no teníamos ESP32, teníamos un ESP8266, el cual necesitaba diferentes librerías y formas de escribir el código al momento de utilizar ciertos métodos. Aunque si nos funcionaban las prácticas con el ESP8266, optamos por conseguir un ESP32 ya que el ESP8266 nos presentaba varios problemas como desconectarse repentinamente o simplemente no cargar el código. Al final pudimos realizar la actividad con éxito con el ESP32, y pudimos comprender más el tema de IoT, así como también la importancia del protocolo HTTP para la conexión y transferencia de datos entre dispositivos de IoT.

Daniel Ezequiel Alvarado Almánzar

El impacto del protocolo HTTP es muy importante ya que gracias a el hay comunicación por sitios web, facilita la Transferencia de Datos y en cierto modo da seguridad, aparte de que utiliza el REST (Representational State Transfer), que se han convertido en un enfoque estándar para diseñar APIs web eficientes y escalables. Y mi opinión personal sobre el desarrollo del programa es que no entendía mucho de su desarrollo ya que estaba un poco verde en cuanto al uso del Lenguaje Ino y la el uso del ESP32 y el sensor, sin embargo, con forme más fueron pasando las sesiones fue haciéndose más entendible.

Christian Rafael Legorreta Avalos

En esta actividad el conjunto de los códigos conforma un sistema integrado que incluye un esquema de Arduino encargado de la recopilación de datos de sensores, el control de un LED y la comunicación con un servidor; una página web en HTML y JavaScript que ofrece una visualización amigable de los datos y permite el control remoto del LED; y un servidor HTTP en Python que sirve la página web, gestiona las solicitudes HTTP y administra los estados del sistema. En conjunto, estos componentes posibilitan la supervisión remota de la temperatura, la humedad y un valor de contador a través de una interfaz web intuitiva, al tiempo que permiten el control del LED. El sistema ofrece flexibilidad para su personalización y expansión según las necesidades y aplicaciones específicas.

Diego Rodolfo Diaz Morfin

El protocolo HTTP (Protocolo de Transferencia de Hipertexto) es fundamental en el mundo de la tecnología y la comunicación en línea. Permite la transferencia de datos a través de la web, lo que lo convierte en la base de muchas aplicaciones y servicios en Internet. HTTP facilita la comunicación entre clientes y servidores, permitiendo la solicitud y entrega de recursos, como páginas web, imágenes, videos y más. Su importancia radica en su papel central en la interconexión de dispositivos y servicios en la era digital.

En cuanto a la experiencia de desarrollar una aplicación de Internet de las Cosas (IoT), fue un proyecto interesante y educativo. en donde pudimos observar una implementación básica de un servidor HTTP que maneja solicitudes GET y POST para controlar un contador, un LED, y recibir datos de temperatura y humedad. El desarrollar aplicaciones IOT puede ser algo desafiante pero cuando se obtienen resultados es algo gratificante de hacer