**Mini-project #2:**

**Cloud Data Upload Using WiFi**

**TEAM MEMBERS**

**RAJ VARSHITH – 16325723**

**SHIVA REDDY – 16352875**

**KARTHIKEYA – 16354793**

**Introduction:**

This project utilizes an ESP32 microcontroller to read temperature and humidity data from a DHT11 sensor, format the data into JSON, and send it to a Flask server for storage and display. The server is set up to receive data via HTTP POST requests.

**Components Required**

* ESP32 Development Board
* DHT11 Temperature and Humidity Sensor
* Jumper wires
* Breadboard (optional)
* Computer with Python and Flask installed

**Libraries Used**

1. **WiFi.h**: For connecting the ESP32 to a Wi-Fi network.
2. **DHT.h**: To interface with the DHT11 sensor.
3. **ArduinoJson.h**: For formatting data into JSON.
4. **HTTPClient.h**: For making HTTP requests.

**Project Setup**

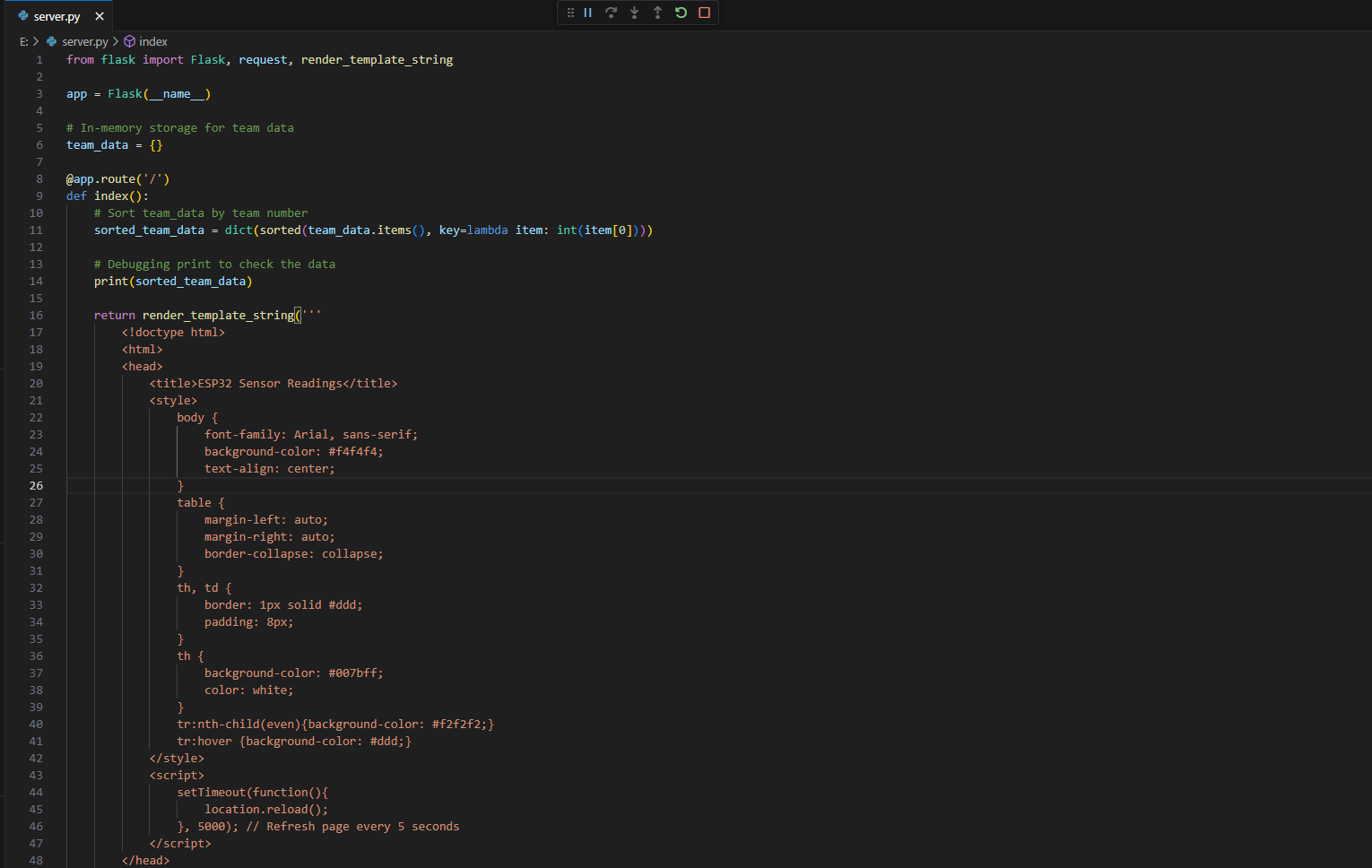
**Hardware Connections**

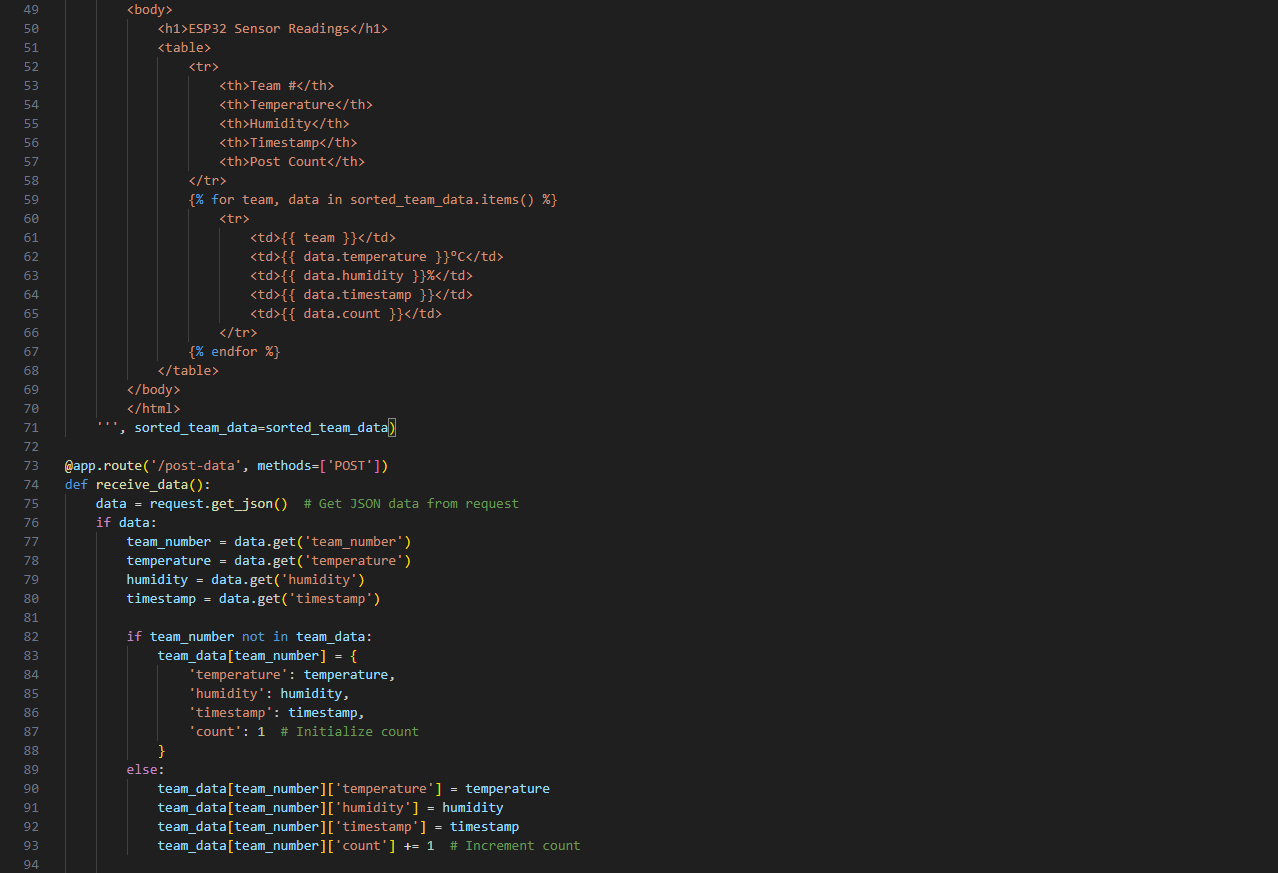
1. Connect the DHT11 sensor:
   * **VCC** to **3.3V** on the ESP32
   * **GND** to **GND** on the ESP32
   * **Data Pin** to **GPIO14** on the ESP32

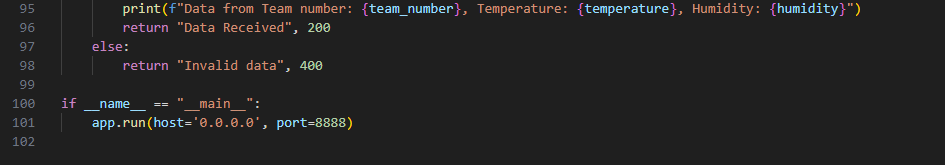
**Software Setup**

1. **Install Arduino IDE**: Ensure you have the Arduino IDE installed on your computer.
2. **Install Required Libraries**: Use the Library Manager in Arduino IDE to install the following:
   * DHT sensor library
   * ArduinoJson library
3. **Set Up Flask Server**:
   * Create a Python file named server.py (see the code below).
   * Ensure you have Flask installed: pip install Flask.
   * Run the server using the command: python server.py.

**Flask Server Code (server.py)**

****

****

****

from flask import Flask, request, render\_template\_string

app = Flask(\_\_name\_\_)

# In-memory storage for team data

team\_data = {}

@app.route('/')

def index():

# Sort team\_data by team number

sorted\_team\_data = dict(sorted(team\_data.items(), key=lambda item: int(item[0])))

return render\_template\_string('''

<!doctype html>

<html>

<head>

<title>ESP32 Sensor Readings</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

text-align: center;

}

table {

margin-left: auto;

margin-right: auto;

border-collapse: collapse;

}

th, td {

border: 1px solid #ddd;

padding: 8px;

}

th {

background-color: #007bff;

color: white;

}

tr:nth-child(even) { background-color: #f2f2f2; }

tr:hover { background-color: #ddd; }

</style>

<script>

setTimeout(function() { location.reload(); }, 5000);

</script>

</head>

<body>

<h1>ESP32 Sensor Readings</h1>

<table>

<tr>

<th>Team #</th>

<th>Temperature</th>

<th>Humidity</th>

<th>Timestamp</th>

<th>Post Count</th>

</tr>

{% for team, data in sorted\_team\_data.items() %}

<tr>

<td>{{ team }}</td>

<td>{{ data.temperature }}°C</td>

<td>{{ data.humidity }}%</td>

<td>{{ data.timestamp }}</td>

<td>{{ data.count }}</td>

</tr>

{% endfor %}

</table>

</body>

</html>

''', sorted\_team\_data=sorted\_team\_data)

@app.route('/post-data', methods=['POST'])

def receive\_data():

team\_number = request.json['team\_number']

if team\_number not in team\_data:

team\_data[team\_number] = {

'temperature': request.json['temperature'],

'humidity': request.json['humidity'],

'timestamp': request.json['timestamp'],

'count': 1 # Initialize count

}

else:

team\_data[team\_number]['temperature'] = request.json['temperature']

team\_data[team\_number]['humidity'] = request.json['humidity']

team\_data[team\_number]['timestamp'] = request.json['timestamp']

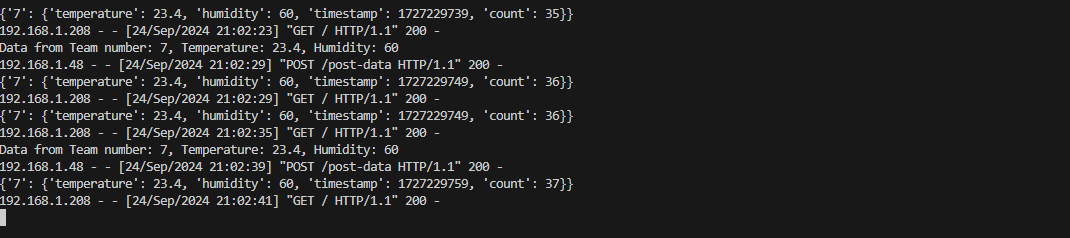
team\_data[team\_number]['count'] += 1 # Increment count

return "Data Received"

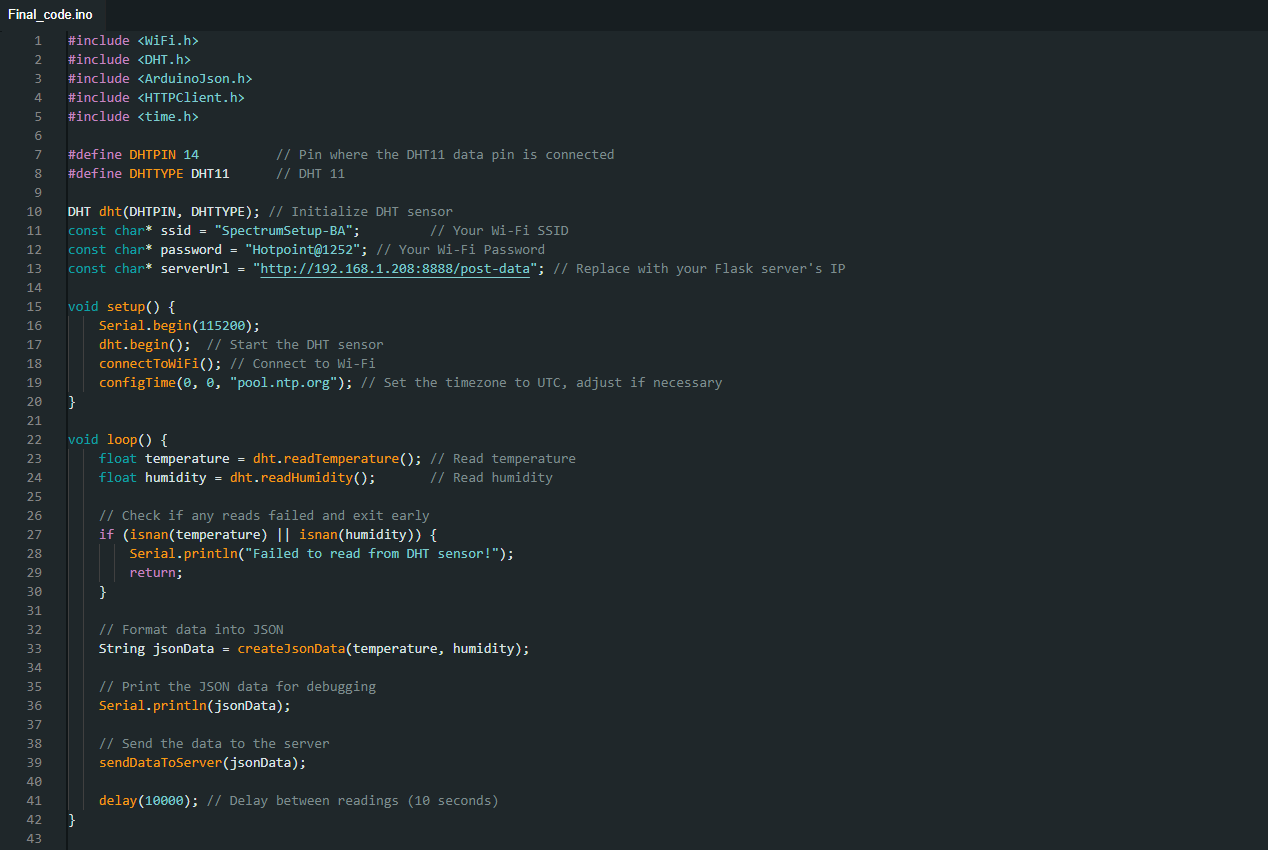
if \_\_name\_\_ == "\_\_main\_\_":

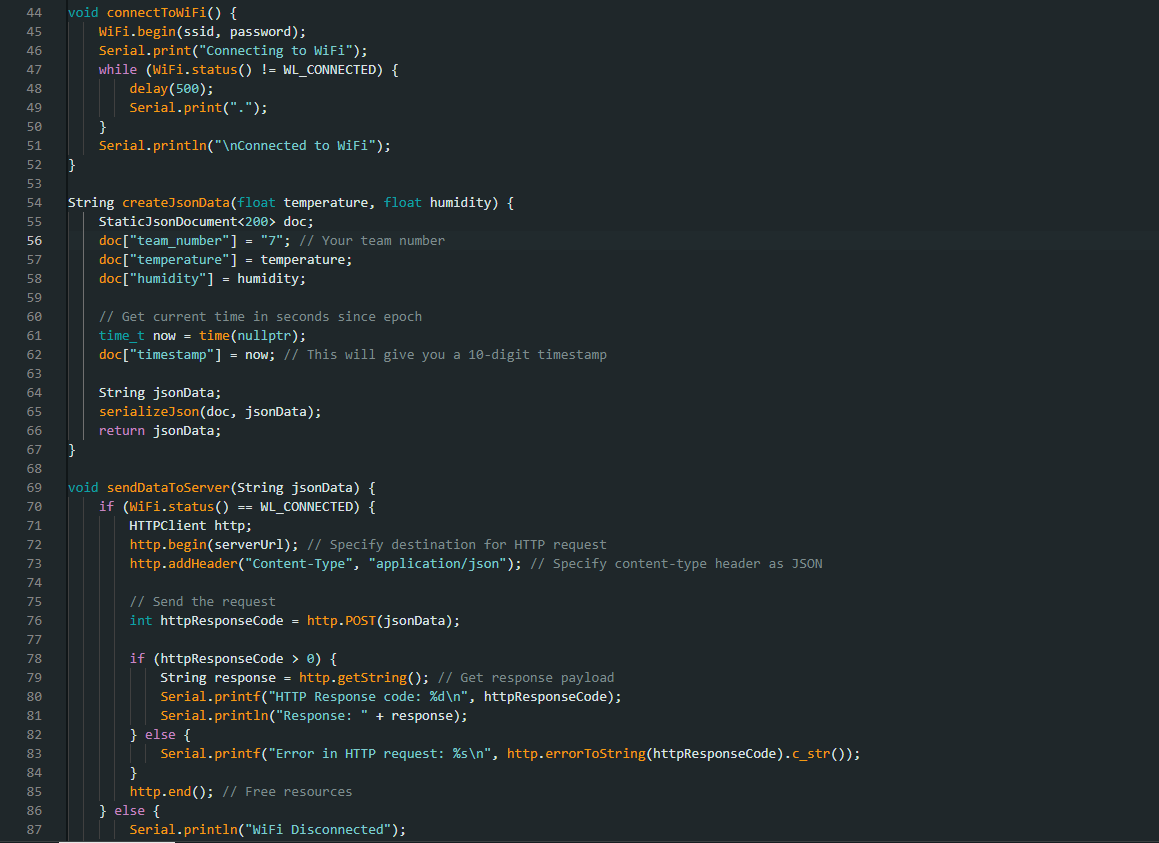
app.run(host='0.0.0.0', port=8888)

**Output:**

****

**ESP32 Code (main.ino)**

****

****

#include <WiFi.h>

#include <DHT.h>

#include <ArduinoJson.h>

#include <HTTPClient.h>

#include <time.h>

#define DHTPIN 14 // Pin where the DHT11 data pin is connected

#define DHTTYPE DHT11 // DHT 11

DHT dht(DHTPIN, DHTTYPE); // Initialize DHT sensor

const char\* ssid = "Your\_SSID"; // Your Wi-Fi SSID

const char\* password = "Your\_Password"; // Your Wi-Fi Password

const char\* serverUrl = "http://192.168.1.208:8888/post-data"; // Replace with your Flask server's IP

void setup() {

Serial.begin(115200);

dht.begin(); // Start the DHT sensor

connectToWiFi(); // Connect to Wi-Fi

configTime(0, 0, "pool.ntp.org"); // Set the timezone to UTC

}

void loop() {

float temperature = dht.readTemperature(); // Read temperature

float humidity = dht.readHumidity(); // Read humidity

// Check if any reads failed and exit early

if (isnan(temperature) || isnan(humidity)) {

Serial.println("Failed to read from DHT sensor!");

return;

}

// Format data into JSON

String jsonData = createJsonData(temperature, humidity);

// Print the JSON data for debugging

Serial.println(jsonData);

// Send the data to the server

sendDataToServer(jsonData);

delay(10000); // Delay between readings (10 seconds)

}

void connectToWiFi() {

WiFi.begin(ssid, password);

Serial.print("Connecting to WiFi");

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("\nConnected to WiFi");

}

String createJsonData(float temperature, float humidity) {

StaticJsonDocument<200> doc;

doc["team\_number"] = "9"; // Your team number

doc["temperature"] = temperature;

doc["humidity"] = humidity;

// Get current time in seconds since epoch

time\_t now = time(nullptr);

doc["timestamp"] = now; // This will give you a 10-digit timestamp

String jsonData;

serializeJson(doc, jsonData);

return jsonData;

}

void sendDataToServer(String jsonData) {

if (WiFi.status() == WL\_CONNECTED) {

HTTPClient http;

http.begin(serverUrl); // Specify destination for HTTP request

http.addHeader("Content-Type", "application/json"); // Specify content-type header as JSON

// Send the request

int httpResponseCode = http.POST(jsonData);

if (httpResponseCode > 0) {

String response = http.getString(); // Get response payload

Serial.printf("HTTP Response code: %d\n", httpResponseCode);

Serial.println("Response: " + response);

} else {

Serial.printf("Error in HTTP request: %s\n", http.errorToString(httpResponseCode).c\_str());

}

http.end(); // Free resources

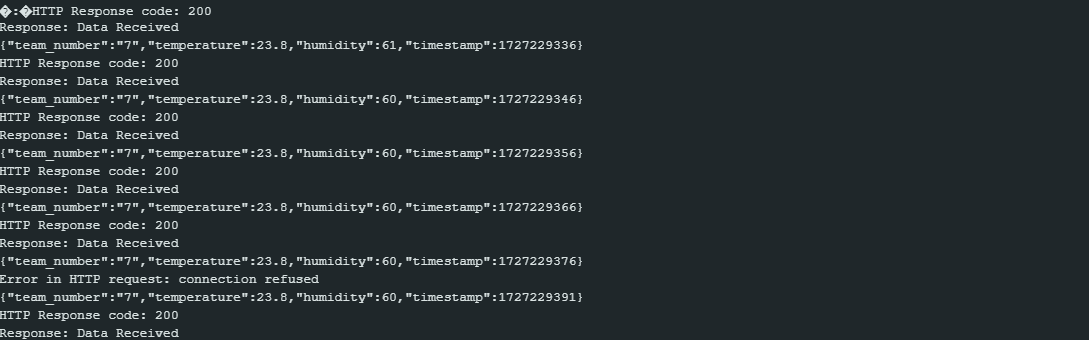
} else {

Serial.println("WiFi Disconnected");

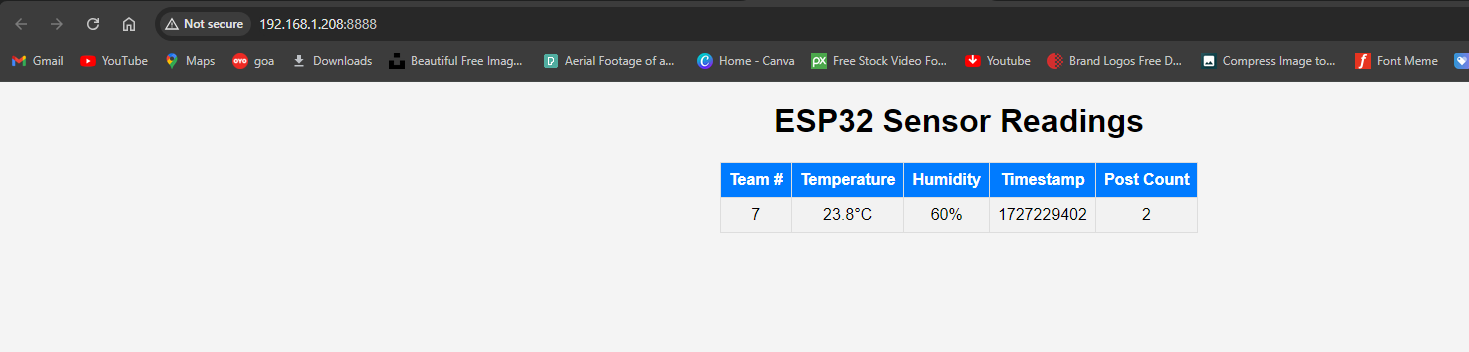
}

}

**Output:**

****

**Final Output from the Web Page:**



**Explanation of the Code**

**Flask Server (server.py)**

* **Imports**: The necessary Flask modules are imported.
* **Data Storage**: A dictionary team\_data is used to store sensor readings from different teams.
* **Routes**:
  + **GET /**: Displays a webpage with the sensor readings in a table format.
  + **POST /post-data**: Receives data sent from the ESP32 and updates or initializes the data for each team.
* **HTML Template**: A simple HTML table is generated to display the sensor data, which refreshes every 5 seconds.

**ESP32 Code (main.ino)**

* **Libraries**: Necessary libraries for Wi-Fi, DHT sensor, JSON handling, and HTTP communication are included.
* **Setup Function**:
  + Initializes serial communication.
  + Starts the DHT sensor.
  + Connects to Wi-Fi and synchronizes time using an NTP server.
* **Loop Function**:
  + Reads temperature and humidity values.
  + Checks if the readings are valid.
  + Formats the data into JSON.
  + Sends the JSON data to the Flask server.
* **Helper Functions**:
  + connectToWiFi(): Connects to the specified Wi-Fi network.
  + createJsonData(): Creates a JSON string containing team number, temperature, humidity, and timestamp.
  + sendDataToServer(): Sends the JSON data to the Flask server using an HTTP POST request.

**Testing**

1. **Start the Flask Server**: Run the server.py file to set up the local server.
2. **Upload ESP32 Code**: Load the ESP32 code into your board using Arduino IDE.
3. **Check Serial Monitor**: Monitor the ESP32 serial output to verify successful readings and HTTP responses.
4. **View Data on Server**: Open a web browser and go to http://<your\_computer\_ip>:8888 to view the sensor data.

**Code Explanation:**

**Flask Server Code (server.py)**

**1. Imports and Setup**

from flask import Flask, request, render\_template\_string

app = Flask(\_\_name\_\_)

* **Flask Imports**: Here, we import the Flask class and the request object from the Flask module to handle HTTP requests and responses.
* **App Initialization**: We create an instance of the Flask class, which will be our web application.

**2. In-Memory Data Storage**

team\_data = {}

* **Dictionary Initialization**: This dictionary will store the sensor data from each team. Each entry will have the team number as the key and a dictionary of sensor data as the value.

**3. Routes**

**a. Index Route**

@app.route('/')

def index():

sorted\_team\_data = dict(sorted(team\_data.items(), key=lambda item: int(item[0])))

return render\_template\_string('''

<!doctype html>

...

''', sorted\_team\_data=sorted\_team\_data)

* **Route Definition**: The @app.route('/') decorator defines the root URL of the web application. When this URL is accessed, the index() function is called.
* **Sorting Data**: The sorted\_team\_data variable sorts the team\_data dictionary by team number, ensuring the displayed data is organized.
* **Rendering HTML**: The function uses render\_template\_string to generate an HTML page dynamically. The HTML template contains a table that displays the sensor readings.

**b. Post Data Route**

@app.route('/post-data', methods=['POST'])

def receive\_data():

...

* **Route Definition**: The @app.route('/post-data', methods=['POST']) decorator defines an endpoint for receiving data from the ESP32 via HTTP POST requests.
* **Data Handling**:
  + The function checks if the incoming team\_number is already in team\_data.
  + If it is not, it initializes the entry with temperature, humidity, timestamp, and a count of how many times data has been sent.
  + If it is, it updates the existing data and increments the count.

**4. HTML Template**

The HTML template is written in the render\_template\_string function, which uses Jinja2 syntax to create a dynamic web page:

<!doctype html>

<html>

<head>

<title>ESP32 Sensor Readings</title>

<style>

...

</style>

<script>

setTimeout(function() { location.reload(); }, 5000);

</script>

</head>

<body>

<h1>ESP32 Sensor Readings</h1>

<table>

<tr>

<th>Team #</th>

<th>Temperature</th>

<th>Humidity</th>

<th>Timestamp</th>

<th>Post Count</th>

</tr>

{% for team, data in sorted\_team\_data.items() %}

<tr>

<td>{{ team }}</td>

<td>{{ data.temperature }}°C</td>

<td>{{ data.humidity }}%</td>

<td>{{ data.timestamp }}</td>

<td>{{ data.count }}</td>

</tr>

{% endfor %}

</table>

</body>

</html>

* **HTML Structure**: The page includes basic HTML elements, a title, and a header.
* **Styling**: Basic CSS is applied for better visual presentation.
* **JavaScript**: A script is included to automatically refresh the page every 5 seconds to get the latest data.
* **Dynamic Content**: The table is populated with data from sorted\_team\_data, iterating over each entry using Jinja2 syntax.

**5. Main Block**

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host='0.0.0.0', port=8888)

* **Main Execution Block**: This checks if the script is being run directly and starts the Flask application on all available IP addresses at port 8888.

**ESP32 Code:**

**1. Includes and Definitions**

#include <WiFi.h>

#include <DHT.h>

#include <ArduinoJson.h>

#include <HTTPClient.h>

#define DHTPIN 14 // Pin where the DHT11 data pin is connected

#define DHTTYPE DHT11 // DHT 11

* **Library Inclusions**: This section includes libraries for Wi-Fi functionality, DHT sensor interface, JSON handling, and HTTP client communication.
* **Pin Definitions**: DHTPIN defines the GPIO pin used to read data from the DHT11 sensor, and DHTTYPE specifies the type of DHT sensor.

1. **Global Variables**

DHT dht(DHTPIN, DHTTYPE); // Initialize DHT sensor

const char\* ssid = "Your\_SSID"; // Your Wi-Fi SSID

const char\* password = "Your\_Password"; // Your Wi-Fi Password

const char\* serverUrl = "http://192.168.1.208:8888/post-data"; // Flask server URL

* **Sensor Initialization**: An instance of the DHT class is created to manage the sensor.
* **Wi-Fi Credentials**: The SSID and password for connecting to the Wi-Fi network are defined.
* **Server URL**: The URL of the Flask server's endpoint to send data is specified.

**3. Setup Function**

void setup() {

Serial.begin(115200);

dht.begin(); // Start the DHT sensor

connectToWiFi(); // Connect to Wi-Fi

configTime(0, 0, "pool.ntp.org"); // Set the timezone to UTC

}

* **Serial Communication**: Initializes serial communication at a baud rate of 115200 for debugging purposes.
* **Sensor Initialization**: The DHT sensor is started.
* **Wi-Fi Connection**: Calls the connectToWiFi() function to connect to the specified network.
* **NTP Configuration**: Sets the time configuration using the Network Time Protocol (NTP) server, which will be used to get the current timestamp.

**4. Loop Function**

void loop() {

float temperature = dht.readTemperature(); // Read temperature

float humidity = dht.readHumidity(); // Read humidity

// Check if any reads failed and exit early

if (isnan(temperature) || isnan(humidity)) {

Serial.println("Failed to read from DHT sensor!");

return;

}

// Format data into JSON

String jsonData = createJsonData(temperature, humidity);

// Print the JSON data for debugging

Serial.println(jsonData);

// Send the data to the server

sendDataToServer(jsonData);

delay(10000); // Delay between readings (10 seconds)

}

* **Sensor Readings**: The ESP32 reads the temperature and humidity values from the DHT11 sensor.
* **Error Checking**: The code checks if the readings are valid using isnan(). If they are not valid, it prints an error message and exits the loop.
* **JSON Formatting**: Calls the createJsonData() function to format the data into a JSON string.
* **Debug Output**: Prints the JSON data to the serial monitor for debugging.
* **Data Transmission**: Calls the sendDataToServer() function to send the JSON data to the Flask server.
* **Delay**: Introduces a 10-second delay between readings.

**5. Wi-Fi Connection Function**

void connectToWiFi() {

WiFi.begin(ssid, password);

Serial.print("Connecting to WiFi");

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("\nConnected to WiFi");

}

* **Connection Attempt**: The ESP32 attempts to connect to the Wi-Fi network using the provided SSID and password.
* **Status Checking**: It checks the connection status in a loop, printing dots to indicate progress.
* **Confirmation**: Once connected, it prints a confirmation message.

**6. JSON Data Creation Function**

String createJsonData(float temperature, float humidity) {

StaticJsonDocument<200> doc;

doc["team\_number"] = "9"; // Your team number

doc["temperature"] = temperature;

doc["humidity"] = humidity;

// Get current time in seconds since epoch

time\_t now = time(nullptr);

doc["timestamp"] = now; // This will give you a 10-digit timestamp

String jsonData;

serializeJson(doc, jsonData);

return jsonData;

}

* **Static JSON Document**: A StaticJsonDocument is created to hold the JSON data. The size of 200 bytes is specified, which should be sufficient for the data.
* **Data Assignment**: The team number, temperature, humidity, and current timestamp are assigned to the document.
* **Timestamp Generation**: The current time is fetched using the time(nullptr) function, which returns the time in seconds since the epoch (January 1, 1970).
* **Serialization**: The document is serialized into a JSON string using serializeJson(), and the resulting string is returned.

**7. Data Transmission Function**

void sendDataToServer(String jsonData) {

if (WiFi.status() == WL\_CONNECTED) {

HTTPClient http;

http.begin(serverUrl); // Specify destination for HTTP request

http.addHeader("Content-Type", "application/json"); // Specify content-type header as JSON

// Send the request

int httpResponseCode = http.POST(jsonData);

if (httpResponseCode > 0) {

String response = http.getString(); // Get response payload

Serial.printf("HTTP Response code: %d\n", httpResponseCode);

Serial.println("Response: " + response);

} else {

Serial.printf("Error in HTTP request: %s\n", http.errorToString(httpResponseCode).c\_str());

}

http.end(); // Free resources

} else {

Serial.println("WiFi Disconnected");

}

}

* **Wi-Fi Check**: Before attempting to send data, it checks if the ESP32 is connected to Wi-Fi.
* **HTTP Client Setup**: An HTTPClient object is created to manage the HTTP request.
* **Request Initialization**: The target server URL is specified, and the content type is set to application/json.
* **Sending the Request**: The JSON data is sent via an HTTP POST request. The response code is checked to determine if the request was successful.
* **Response Handling**: If successful, it prints the HTTP response code and the server’s response message. If there’s an error, it prints the error message.
* **Resource Cleanup**: The http.end() function frees up resources used by the HTTP client.