Mini-project #2: Cloud Data Upload Using WiFi

Objectives (1/2)

- Read Sensor Data:
  - Acquire temperature and humidity data from the DHT11 sensor using the ESP32 microcontroller.
- Format Data into JSON:
  - Convert the sensor data into a JSON format suitable for transmission to a server.
- Establish WiFi Connectivity:
  - Configure the ESP32 to connect to a specified WiFi network to enable cloud communication.

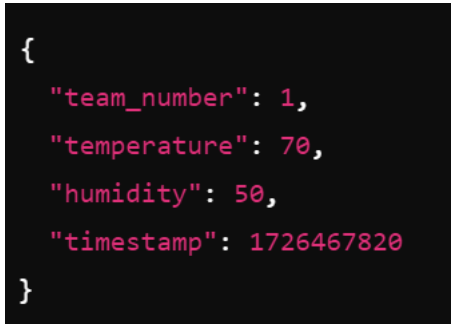
Objectives (2/2)

- Upload Data to a Cloud Server:
  - Implement an HTTP POST request to send the JSON-formatted data to a remote server.

- (Optional) Implement Data Encryption:
  - Secure the data transmission by encrypting the sensor data before sending it to the server.
  - Decrypt the data on the server side and demonstrate successful data decryption and storage.

Project Requirements (1/2)

1. Basic Implementation:
   - Sensor Data Acquisition: Correctly read temperature and humidity data from the DHT11 sensor.
   - JSON Formatting: Format the sensor data in the specified JSON structure:



   - WiFi Connectivity: Configure the ESP32 to connect to a local WiFi network.
   - Data Upload: Use the HTTP POST method to upload the formatted JSON data to a server.

Project Requirements (2/2)

2. **(Optional for extra credit: 3 points)** Advanced Implementation:
   - Data Encryption on ESP32:
     - Encrypt the sensor data (temperature, humidity, and timestamp) before transmission.
     - Ensure that the encrypted data is correctly formatted into JSON and sent to the server.
   - Data Decryption on Server:
     - Modify the provided *server.py* Flask server code to include logic for decrypting the received data.
     - Verify that the decrypted data is correctly displayed and stored on the server.

Debug Guide (1/3)

- To assist with testing your code locally, we have provided a *server.py* file. This Python script creates a Flask server that will receive data from the ESP32 via the HTTP POST method. The port number is set to 8888 by default, and the actual local IP address will be displayed in the third row of the server output. This is the IP address you will need to use when uploading your ESP32's data to your server.

```
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:8888
* Running on http://10.0.0.140:8888
```

Debug Guide (2/3)

Steps for Local Testing:

1. Set Up the Flask Server:
   - Run the server.py file on your computer to start the Flask server.
   - This will simulate the cloud server that your ESP32 will send data to.

2. Connect to Your Home Wi-Fi:
   - During the debugging process, configure your ESP32 to connect to your home Wi-Fi network.

Debug Guide (3/3)

Steps for Local Testing:

3. Modify Server Details:
   - Configure the ESP32 to send data to the IP address of your computer running the Flask server. Use the appropriate local network IP (e.g., http://10.0.0.140:8888/post-data) in place of the demo server details.

4. Test Your Code:
   - Run your ESP32 code, sending the JSON-formatted sensor data to your local Flask server.Verify that the server receives the data correctly by checking the Flask server's console output.

Submissions

- Source Code: Provide well-documented code (Arduino code *.ino and (Optional) modified *server.py*) with comments explaining each part of the code. **(Deadline: 11: 59 PM on March 2, 2025)**

- Project Report: A concise report describing the project's implementation and any challenges faced. **(Deadline: 11: 59 PM on March 2, 2025)**

- In-class Demo: Showcase the working project in real-time. **(Deadline: 5:30 PM – 6:45 PM on March 3, 2025)**

Evaluation Criteria

- Basic implementation of uploading the sensor data from the ESP32 to a cloud server using WiFi.
- **(Optional for extra credit)** Advanced implementation of data encryption and decryption.
- Code quality (readability, comments, and structure).
- Clarity and thoroughness of the project report and demonstration.