Assignment: IOT **Teacher Name: Sir Nasir**
Name: Salman Malhi **Roll No: 1272**

This assignment covers two main topics:
Question 1: ESP32 Webserver
Question 2: Blynk Cloud Interfacing

Question 1: ESP32 Webserver (webserver.cpp)

This part is about making the ESP32 a simple local web server to check **temperature and humidity**.

Part A: Short Questions

1. What is the purpose of `WebServer server(80);` and what does port 80 represent?

- **Purpose of `WebServer server(80);`:** This line starts the web server object.
- The number **80** tells the server which "door" (TCP Port) to use to listen for incoming web requests (HTTP requests) from a web browser.
- **Port 80:** Port 80 is the standard, default port for the HTTP (Hypertext Transfer Protocol) service.
- When you type an IP address (like `192.168.1.10`) without a port number, the browser automatically tries to connect on Port 80.

---

2. Explain the role of `server.on("/", handleRoot);` in this program.

- **Role:** This function is a Router.
- It links a specific web address path to a specific function in your code.
- **"/" (The Root):** This is the main or home page address.
- When a user types only the ESP32's IP address, they are asking for this root path.
- **`handleRoot`:** This is the Callback Function.
- It is the name of the function in your Arduino code that must run when the server gets a request for the "/" path.

---

3. Why is `server.handleClient();` placed inside the `loop()` function? What will happen if it is removed?

- **Why it's in `loop()`:** The `server.handleClient()` function must be run constantly and very quickly.
- It checks the network connection for any new incoming requests from a client (like your web browser) and handles them.
- Placing it in the `loop()` guarantees that the server never misses a request.
- **If it is removed:** The web server will stop working.
- The ESP32 will not know when a browser is trying to connect or ask for the page.
- The browser will eventually show a **Connection Timeout** error.

---

4. In `handleRoot()`, explain the statement: `server.send(200, "text/html", html);`

- This statement is responsible for sending the web page back to the browser.
- **200:** This is the HTTP Status Code.
- **200 OK** means the request was successful, and the server is sending the content as requested.
- **"text/html":** This is the Content Type.
- It tells the browser that the data it is receiving is in HTML (web page language) format, so the browser knows how to display it correctly.
- **html:** This is the String Variable.
- It contains the complete HTML code that was dynamically created in the `handleRoot()` function, including the sensor values.

---

5. What is the difference between displaying last measured sensor values and taking a fresh DHT reading inside `handleRoot()`?

Feature	Displaying Last Measured Value	Taking a Fresh DHT Reading inside handleRoot()
Data Source	Values stored in global variables that were read moments ago in the main `loop()` or a timer	Reading the physical DHT sensor every time a browser requests the web page
Speed/Performance	Very Fast. The server sends the response immediately because it uses stored data	Slow. Reading the DHT sensor takes hundreds of milliseconds (ms), which slows down the web server's response time

Best Practice	Better way. Use a separate timer to read the slow sensor and store the value. This keeps the web server fast and responsive	Avoid this. The server could crash or become unresponsive if many users request the page at once

---

Part B: Long Question

Describe the complete working of the ESP32 webserver-based temperature and humidity monitoring system.

Here is the step-by-step process of how the system works:

---

1. ESP32 Wi-Fi Connection Process and IP Address Assignment

- The ESP32 starts and tries to connect to the local Wi-Fi network using the provided **SSID** (name) and **password**.
- Once connected, the router gives the ESP32 a **Local IP Address** (e.g., `192.168.1.100`).
- This address is needed for the browser to find the server.

---

2. Web Server Initialization and Request Handling

- The web server is started on Port 80 using commands like `server.begin()`.
- The `server.on("/", handleRoot)` command sets up the rule: "If someone asks for the home page ('/'), run the `handleRoot()` function."
- The `server.handleClient()` in the `loop()` constantly checks for and handles new web requests.

---

3. Button-based Sensor Reading and OLED Update Mechanism

- The system uses either a button press or a timer to decide when to read the sensor.
- When triggered, the DHT sensor is read, and the temperature/humidity values are updated.
- These new values are then sent to the OLED display to show the current reading.
- The values are also stored in variables for the web server to use.

---

4. Dynamic HTML Webpage Generation

- Inside the `handleRoot()` function, a String variable is built to hold the HTML code.
- The stored sensor values are inserted directly into this HTML string.
- This makes the web page content change (dynamic) every time it is requested.
- The final HTML page is sent back to the browser using `server.send()`.

---

5. Purpose of Meta Refresh in the Webpage

- The web page includes a special HTML tag called **meta refresh**.
- **Purpose:** This tells the user's browser to automatically reload the page after a set time (e.g., 5 seconds).
- Since the ESP32 doesn't push data automatically, this ensures the browser gets the latest sensor values without the user clicking the refresh button.

---

6. Common Issues in ESP32 Webserver Projects and their Solutions

- **Issue: Server is Slow/Unresponsive** → This usually happens when long `delay()` commands are used, which stop the server from handling requests.
    - **Solution:** Never use `delay()` in the `loop()`. Use non-blocking timers instead.
- **Issue: Wi-Fi Disconnection** → The ESP32 loses its network connection.
    - **Solution:** Constantly check the Wi-Fi status in the `loop()` and use code to automatically reconnect if the link is lost.

Question 2: Blynk Cloud Interfacing (blynk.cpp)

Part A: Short Questions

1. Role of Blynk Template ID

- Template ID is a **unique key** (like a serial number).
- It tells ESP32 which project structure (template) on Blynk Cloud it belongs to.
- Must match → otherwise Cloud won't know where to send data, and device won't connect.

---

2. Difference between Template ID and Auth Token

Feature	Blynk Template ID	Blynk Auth Token
What it identifies	The type of project (e.g., "Monitor Project"). Same for all devices of that project type.	A specific device (e.g., "My Living Room ESP32"). Each device gets a unique token.
Main Purpose	Links code to correct project structure on Cloud.	Authenticates (logs in) the specific device to Blynk Cloud.

---

3. Why DHT22 code with DHT11 gives wrong readings

- DHT sensors use specific timing rules.
- If ESP32 expects DHT22 format but sensor is DHT11 → data bits are misread.
- **Key Difference:**
    - DHT11 → whole numbers only, less accurate.
    - DHT22 → decimal values, more accurate, wider range.

---

4. Virtual Pins in Blynk

- Virtual Pins (V0, V1, V2…) are **logical data channels**, not real GPIO pins.
- Work like numbered mailboxes for sending/receiving data.
- **Preferred because:**
    - Flexible → send any type of data (temperature, text, status).
    - Two-way → ESP32 → Cloud (sensor data) and Cloud → ESP32 (control commands).

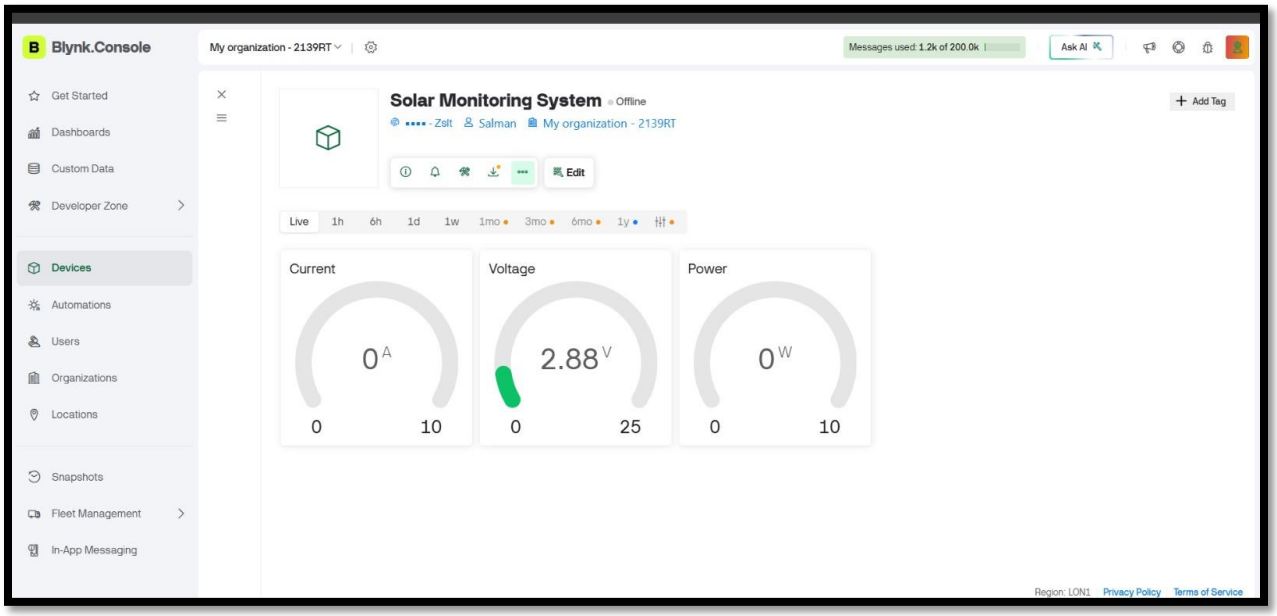
---

5. Purpose of BlynkTimer vs delay()

- **Problem with delay():** Blocks program, Wi-Fi check stops, Blynk connection fails.
- **BlynkTimer:** Non-blocking scheduler.
    - Runs functions at intervals (e.g., every 5s).
    - Meanwhile `loop()` keeps running `Blynk.run()`, keeping device responsive.

Part B: Long Question – Complete Workflow

1. **Create Template + Datastreams:**



---

2. Role of Template ID, Name, Auth Token

- After creating device, you get Template ID, Template Name, Auth Token.
- Copy these into ESP32 code → ensures correct login and connection to Cloud.

---

3. Sensor Configuration Issues (DHT11 vs DHT22)

- Must define correct sensor type in code.
- Wrong type → incorrect values sent to Cloud.

---

4. Sending Data with `Blynk.virtualWrite()`

- `BlynkTimer` calls function every few seconds.
- Function reads Temp (T) and Voltage (V).
- `Blynk.virtualWrite(V0, T);` → sends Temp to Cloud.
- `Blynk.virtualWrite(V1, H);` → sends Humidity to Cloud.

---

5. Dashboard Display & Visualization

- On Blynk dashboard (web/mobile), add widgets (Gauge, Chart).
- Link widgets to Datastreams (V0 = Temo, V1 = humidity).
- As soon as ESP32 sends data, widgets update instantly.

---

6. Common Problems & Solutions

- **Device Offline:** Wrong Template ID/Auth Token/SSID.
  - Fix → check for typos, ensure `Blynk.run()` is active in `loop()`.
- **Widget shows old data:** Data not updating.
  - Fix → set correct `BlynkTimer` interval (e.g., 5s) and ensure `Blynk.virtualWrite()` runs properly.
