Name	Muhammad Hamza
Registration#	23-NTU-CS-1186
Assignment#	2
Course Name	Embedded IoT Systems
Section	BSCS-5th-A
Department	Computer Science
Submit To	Dr. Nasir Mehmood
Date	20-12-2025

QUESTION 1

PART A

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

WebServer server(80); creates an HTTP web server on the ESP32 to display DHT22 sensor data in a browser.

Port 80 is the default HTTP port, so users can access the webpage by entering only the ESP32 IP address without specifying a port number.

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

This line connects the root URL / to the function handleRoot(). When a user opens the ESP32 IP address in a browser, handleRoot() is called to generate and send the HTML page containing temperature and humidity readings.

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

'server.handleClient();' continuously checks for incoming web requests. Placing it in loop() allows the ESP32 to respond at any time. If it is removed, the web server will not respond and the webpage will not load.

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

This command sends a response to the browser. 200 means the request was successful, "text/html" specifies the content type, and html contains the webpage data to be displayed.

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

This code displays previously measured values stored in lastTemp and lastHum, which are updated only when the button is pressed.

Taking a fresh reading inside handleRoot() would read the sensor on every page refresh, causing delays and unreliable readings. Using stored values improves performance and stability.

PART B

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

1. ESP32 Wi-Fi connection and IP address assignment

When the ESP32 starts, it initializes the Wi-Fi module using the provided SSID and password. The ESP32 continuously checks the connection status until it successfully connects to the network. Once connected, the router assigns an **IP address** to the ESP32 using DHCP. This IP address is printed on the Serial Monitor and also shown on the OLED display. The IP is used by any device on the same network to access the ESP32 web server.

2. Web server initialization and request handling

After establishing the Wi-Fi connection, a **WebServer** object is created on port 80. The root URL ("/") is linked to a handler function. When a browser sends an HTTP request to the ESP32's IP address, the server processes the request, executes the handler function, and sends back an HTML response. The server continuously listens for incoming client requests inside the loop function.

3. Button-based sensor reading and OLED update mechanism

A push button is connected to GPIO5 and configured using **INPUT_PULLUP**. The button is active-LOW.
When the button is pressed:

- The ESP32 detects the falling edge (HIGH → LOW).

- The DHT11 sensor is read for temperature and humidity.
- Valid readings are stored and displayed on the **SSD1306 OLED** screen.
  This design ensures sensor readings are taken only when required, reducing unnecessary updates.

4. Dynamic HTML webpage generation

The ESP32 dynamically creates an HTML webpage using a string. The webpage includes:

- A title and headings
- Current temperature and humidity values (if available)
- A message prompting the user to press the button for fresh readings
  The values are embedded directly into the HTML before sending it to the browser, making the webpage **dynamic** rather than static.

5. Purpose of meta refresh in the webpage

The webpage uses a **meta refresh tag** that automatically reloads the page every few seconds. Its purpose is to:

- Update displayed sensor values without manual refresh
- Show the latest readings after a button press
- Provide near real-time monitoring without using JavaScript

6. Common issues in ESP32 webserver projects and solutions

Wi-Fi not connecting

- Cause: Incorrect SSID/password or weak signal
- Solution: Verify credentials and ensure stable network

Webpage not opening

- Cause: Wrong IP address or server not started
- Solution: Check Serial Monitor for IP and ensure server.begin() is called

DHT sensor read failure

- Cause: Wrong sensor type or wiring issues

- Solution: Confirm DHT11 selection and secure connections

OLED not displaying data

- Cause: Incorrect I2C address or wiring
- Solution: Verify SDA/SCL pins and I2C address (0x3C)

Slow or unresponsive server

- Cause: Blocking delays
- Solution: Minimize delay() usage and keep loop non-blocking

7. Conclusion

The ESP32 connects to Wi-Fi, hosts a local web server, reads temperature and humidity using the DHT22 sensor upon button press, displays values on an OLED screen, and serves real-time data through a dynamically generated webpage accessible via the assigned IP address.

QUESTION 2

PART A

1. Role of Blynk Template ID and why it must match the cloud template

The **Blynk Template ID** links the ESP32 firmware to a specific device template on the Blynk Cloud. It defines the structure of the project, including datastreams and widgets. It must match the cloud template so that the ESP32 sends data to the correct project and the cloud knows how to handle incoming values.

2. Difference between Blynk Template ID and Blynk Auth Token

- **Template ID** identifies the device template and its configuration on the Blynk Cloud.
- **Auth Token** is a unique security key that authenticates a specific device and allows it to connect to the cloud.
  In short, the Template ID defines *what* the device is, while the Auth Token authorizes *which* device is connecting.

3. Why DHT22 code gives incorrect readings with a DHT11 sensor

DHT11 and DHT22 use different internal data formats and measurement ranges. Using DHT22 code with a DHT11 causes wrong data interpretation.
Key difference: DHT22 provides higher accuracy and supports decimal values, while DHT11 gives only integer values with lower accuracy.

4. What are Virtual Pins in Blynk and why they are preferred

Virtual Pins are software-defined pins used to send and receive data between the ESP32 and Blynk Cloud.
They are preferred because:

- They are independent of hardware GPIO limitations
- They allow easy cloud communication

- They simplify mapping sensor data to widgets

5. Purpose of using BlynkTimer instead of delay()

BlynkTimer allows non-blocking task execution at fixed intervals.
Unlike delay(), it does not stop program execution, ensuring:

- Continuous Wi-Fi and Blynk connectivity
- Responsive button handling
- Stable real-time data updates

PART B

Explain the complete workflow of interfacing ESP32 with Blynk Cloud to display temperature and humidity values.

1. Creation of Blynk Template and Datastreams

In the Blynk Cloud, a **Template** is created for the ESP32 device. Inside this template:

- Datastreams are defined (for example, **V0 for Temperature** and **V1 for Humidity**).
- Each datastream is linked to a widget (like Gauge or Value Display) in the Blynk dashboard.
- This setup allows Blynk to receive and visualize sensor data sent from the ESP32.

2. Role of Template ID, Template Name, and Auth Token

- **Template ID** uniquely identifies the device template on Blynk Cloud and links the ESP32 firmware to that template.
- **Template Name** is a human-readable name used for identification in the Blynk platform.
- **Auth Token** is a unique authentication key that authorizes the ESP32 to connect securely to the Blynk Cloud. In the code, these credentials ensure that data from the ESP32 is sent to the correct Blynk project.

3. Sensor configuration issues (DHT11 vs DHT22)

The code uses **DHT22**, which is more accurate and has a wider range than DHT11. But as we have only DHT11 so we changed it in the code as well. Common issues include:

- Selecting the wrong sensor type in code (e.g., defining DHT22 instead of DHT11)
- Loose wiring or incorrect GPIO pin selection. These issues can result in **NaN readings** or failed sensor communication. Correct sensor selection and proper wiring solve this problem.

4. Sending data using Blynk.virtualWrite()

After reading temperature and humidity from the DHT11:

- The ESP32 sends temperature to **Virtual Pin V0**
- Humidity is sent to **Virtual Pin V1.** This is done using Blynk.virtualWrite(), which transmits sensor values to the Blynk Cloud, where they are displayed on the mobile or web dashboard in real time.

5. Common problems during configuration and solutions

ESP32 not connecting to Blynk

- Cause: Incorrect Wi-Fi credentials or Auth Token
- Solution: Verify SSID, password, and Auth Token

Data not appearing on Blynk dashboard

- Cause: Datastream virtual pins not matching code
- Solution: Ensure V0 and V1 are correctly configured in Blynk

DHT sensor returning NaN or trash values

- Cause: Wrong sensor type or unstable power
- Solution: Confirm DHT11 selection and secure connections

Frequent disconnections

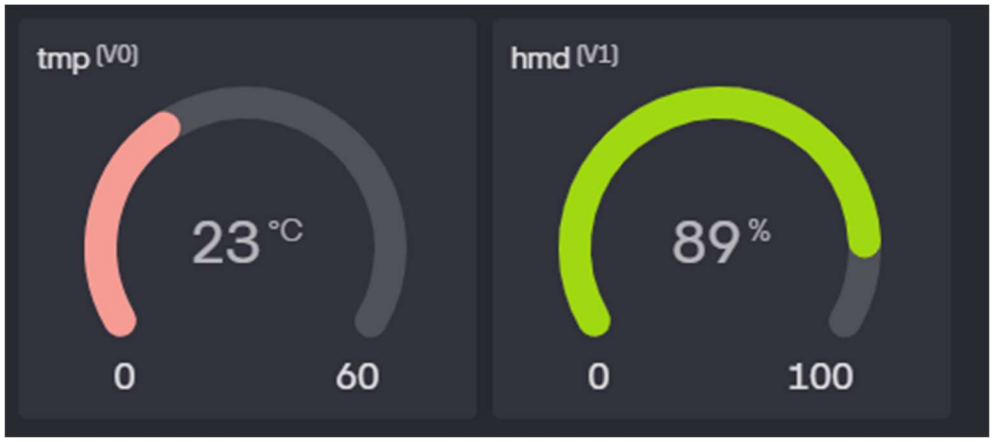
- Cause: Blocking delays or unstable network
- Solution: Use BlynkTimer instead of delay() and ensure stable Wi-Fi.

6. Conclusion

The ESP32 reads temperature and humidity from the DHT22 sensor, displays values locally on the OLED, and sends them to the Blynk Cloud using virtual pins. Blynk Cloud

processes this data and displays it in real time on the dashboard, enabling remote environmental monitoring.

Blynk Cloud



Mobile App