National Textile University, Faisalabad

Department of Computer Science

Name:	Aaiza Rashid
Class:	BSCS 5th _A
Registration No:	23-NTU-CS-1001
Course Name:	Embedded IOT
Submitted To:	Sir Nasir
Submission Date:	21/12/2025

Question-1

ESP32 Webserver (webserver.cpp)

Part A: Short Questions

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

The statement WebServer server(80); is used to set up a web server on the ESP32 so it can receive requests from a web browser. Port 80 is the standard port used for HTTP communication. When someone enters the ESP32's IP address in a browser, the request is automatically sent to port 80. Because of this, the ESP32 can act like a small web server and display a webpage.

---

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

This line tells the web server how to respond when the main page of the website is opened. When a user accesses the ESP32's IP address, the server calls the handleRoot() function. This function creates the webpage and sends it to the browser. Without this instruction, the server would not know what to display when the page is requested.

---

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

The function server.handleClient(); continuously checks if a browser is trying to connect to the ESP32. Placing it inside the loop() function allows the ESP32 to keep responding to requests while it is running. If this line is removed, the web server will stop responding, and the webpage may either load only once or not load at all.

---

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

This statement is used to send the webpage to the browser. The code 200 indicates that the request was successful. The "text/html" part tells the browser that the content being sent is an HTML page. The variable html contains the actual webpage content that will be shown to the user.

---

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

Displaying the last measured sensor values means the webpage shows the temperature and humidity readings that were previously captured when the button was pressed. Taking a fresh DHT reading inside handleRoot() would cause the sensor to be read every time the page is refreshed. In this program, the sensor is not read on each refresh, which helps avoid unnecessary readings and keeps sensor operation controlled by the physical button.

Part B: Long Question

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

This system uses an ESP32, a DHT sensor, an OLED display, a push button, and a web server to monitor temperature and humidity values both locally and through a web browser.

ESP32 Wi-Fi connection process and IP address assignment

When the ESP32 is powered ON, it connects to the Wi-Fi network using the SSID and password defined in the program. After a successful connection, the Wi-Fi router assigns a local IP address to the ESP32. This IP address is displayed on the OLED screen using WiFi.localIP(), which helps the user know which address to enter in the browser to access the web page.

Web server initialization and request handling

The ESP32 creates a web server using WebServer server(80);, where port 80 is the default port for HTTP communication. The line server.on("/", handleRoot); specifies what action should be taken when a user opens the ESP32 IP address in a browser. The web server is started using server.begin(). Inside the loop() function, server.handleClient() continuously checks for incoming browser requests and responds to them accordingly.

Button-based sensor reading and OLED update mechanism

A push button is connected to the ESP32 to control when the sensor data is updated. The program detects a button press using falling-edge detection by comparing the current and previous button states. When the button is pressed, the DHT sensor is read using the readDHTValues() function, and the latest temperature and humidity values are displayed on the OLED screen using showOnOLED(). This ensures that sensor readings are taken only when required and not continuously.

Dynamic HTML webpage generation

The handleRoot() function dynamically generates an HTML webpage as a string. This webpage displays the last measured temperature and humidity values instead of taking a new sensor reading on every refresh. The webpage is sent to the browser using server.send(200, "text/html", html);, allowing the user to monitor the sensor data through a web browser.

Purpose of meta refresh in the webpage

The HTML webpage includes a meta refresh tag that automatically reloads the page after a fixed time interval. This feature allows updated sensor values to appear on the webpage without the user needing to manually refresh the browser.

Common issues in ESP32 webserver projects and their solutions

- **Wi-Fi connection failure →** Usually caused by an incorrect SSID or password; this can be fixed by double-checking the Wi-Fi credentials in the code.
- **Webpage not loading →** Happens when the wrong IP address or port is used; ensure the correct IP address is entered in the browser and that port 80 is being used.
- **Sensor values not updating →** Occurs if the push button is not pressed; pressing the button triggers a fresh sensor reading.
- **ESP32 freezing or becoming unresponsive →** Can be avoided by minimizing long delay() statements and making sure server.handleClient() runs continuously inside the loop() function.

Question-2

Blynk Cloud Interfacing (blynk.cpp)

Part-A:

Short Questions

1.Role of Blynk Template ID in an ESP32 IoT project

The Blynk Template ID is used to connect an ESP32 device with a specific project template created on the Blynk Cloud. This template contains the dashboard layout, widgets, and virtual pins that define how data will be displayed and controlled. The Template ID in the code must exactly match the one on the Blynk Cloud; otherwise, the ESP32 will not recognize the correct interface and will fail to connect properly.

Example:

```
#define BLYNK_TEMPLATE_ID "TMPL6UCKQ_P6D"

#define BLYNK_TEMPLATE_NAME "dht"
```

2. Difference Between Blynk Template ID and Blynk Auth Token

The Blynk Template ID represents the overall project structure on the Blynk Cloud, including widgets and datastreams. It remains the same for all devices linked to

that project.
On the other hand, the Blynk Auth Token is a unique security key assigned to each individual device. It is used to authenticate the ESP32 with the Blynk Cloud.
While both are essential for proper communication, the Template ID defines the project layout, whereas the Auth Token verifies the identity of the device.

---

3. Reason for Incorrect Readings When Using DHT22 Code with a DHT11 Sensor

The DHT11 and DHT22 sensors differ in data format, accuracy, and timing. The DHT22 supports decimal readings and has a wider temperature range, whereas the DHT11 provides only integer values with lower accuracy.
When the code is configured for a DHT22 sensor but a DHT11 is used in hardware, the ESP32 misinterprets the sensor data, resulting in incorrect or invalid readings.

Example:

```
#define DHTTYPE DHT22
```

This configuration is specifically meant for DHT22, and using a DHT11 will lead to wrong values.

---

4. Virtual Pins in Blynk and Their Importance

Virtual Pins are software-defined pins that allow data exchange between the ESP32 and the Blynk Cloud. Unlike physical GPIO pins, they do not correspond to actual hardware connections.
They are preferred because they offer flexibility, enabling changes in logic and data flow without modifying the circuit. In this project, temperature and humidity values are sent using virtual pins V0 and V1.

Example:

```
Blynk.virtualWrite(V0, t);
Blynk.virtualWrite(V1, h);
```

---

5. Purpose of Using BlynkTimer Instead of delay()

BlynkTimer is used to execute tasks at regular intervals without stopping the main program. Using delay() blocks the ESP32, which can interrupt cloud communication and make the system unresponsive.
By using BlynkTimer, multiple operations such as sensor readings, button handling, and cloud updates can run smoothly at the same time, ensuring better stability and responsiveness.

Example:

```
BlynkTimer timer;

timer.setInterval(5000L, periodicSend);
```

Using delay() instead could result in loss of Blynk connection and delayed button response.

Part B: Long Question

Complete Workflow of Interfacing ESP32 with Blynk Cloud to Display Temperature and Humidity

1. Creating the Blynk Template and Datastreams

The process begins by creating a template on the Blynk Cloud, which defines the layout and behavior of the IoT application. For this project, a template named **“dht”** is created with widgets to display temperature and humidity.
Two datastreams are configured:

- **V0** for temperature
- **V1** for humidity

These datastreams act as communication channels that receive sensor data from the ESP32 and display it on the Blynk dashboard.

---

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

The **Template ID** ensures that the ESP32 connects to the correct dashboard and datastreams.
The **Template Name** is mainly used for identification and debugging purposes.
The **Auth Token** is a unique key that authenticates the ESP32 device with the Blynk Cloud.
If any of these credentials are incorrect, the ESP32 will not be able to establish a connection.

---

3. Connecting ESP32 to the Blynk Cloud

Inside the setup() function, the ESP32 connects to Wi-Fi and then to the Blynk Cloud using:

```
Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
```

This function manages both network connection and authentication. The connection is continuously maintained in the loop() function using:

```
Blynk.run();
```

---

4. Sensor Configuration and Compatibility Issues

The code specifies the sensor type as:

```
#define DHTTYPE DHT22
```

The DHT22 provides more accurate readings and supports decimal values. If a DHT11 sensor is connected while the code is set for DHT22, incorrect data or NaN values may appear. Therefore, the sensor definition in the code must always match the actual hardware used.

---

5. Reading Sensor Data Using Button and Timer

Temperature and humidity values are read using:

dht.readTemperature();

dht.readHumidity();

Sensor readings are triggered in two ways:

- **Manually**, when a push button is pressed
- **Automatically**, every 5 seconds using BlynkTimer

This approach ensures efficient sensor usage and provides both on-demand and periodic updates.

---

6. Sending Data to Blynk Dashboard

After obtaining sensor values, the ESP32 sends the data to the Blynk Cloud using:

Blynk.virtualWrite(V0, t);

Blynk.virtualWrite(V1, h);

The values are instantly reflected on the dashboard widgets connected to these virtual pins.

---

7. Role of BlynkTimer in System Stability

BlynkTimer allows periodic task execution without blocking the main program. This keeps the ESP32 responsive and ensures stable communication with the Blynk Cloud, even while handling multiple operations simultaneously.

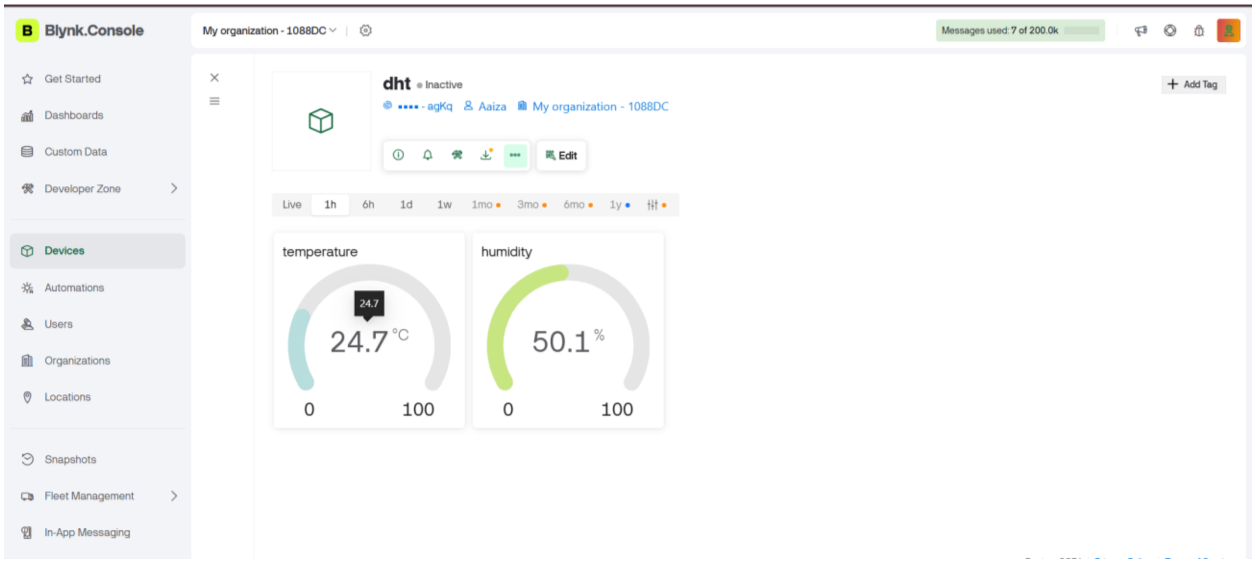
---

8. Common Issues and Their Solutions

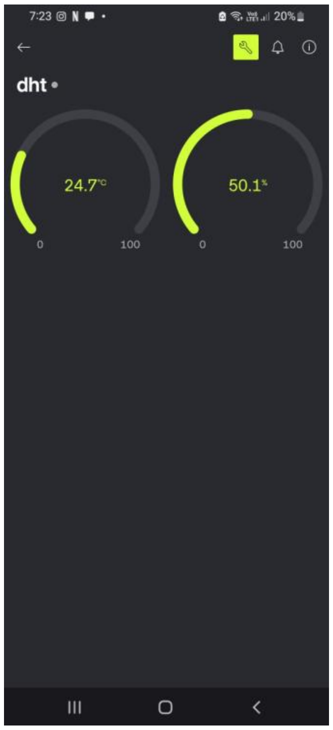
A common issue is incorrect Template ID or Auth Token, which prevents cloud connection. This can be resolved by carefully copying the correct credentials from the Blynk Cloud.
Another frequent problem is sensor mismatch, such as using a DHT11 with DHT22 code. Correctly defining the sensor type in the code resolves this issue.

SCREENSHOTS:

Web Dashboard:



Mobile App dashboard


7:23
20%
dht
24.7°C
0
100
50.1%
0
100