Department of Computer Science

Name:	MUHAMMAD UMAIR
Class:	BSCS 5th-B
Registration No:	23-NTU-CS-1054
Assignment No:	# 02 After-Mids
Course Name:	IoT and Embedded Systems
Submitted To:	Sir Nasir Mehmood
Submission Date:	17 - 12 - 2025

Question No 1 ESP32 Webserver (webserver.cpp)

Part A: Short Questions

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

This line creates the actual web server object so the code can handle network traffic. **Port 80** is used because it is the standard default port for web pages, meaning you don't have to type a specific port number in the browser address bar to connect.

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

This function links the homepage (the root URL "/") to a specific function in your code called **handleRoot**. Basically, it tells the ESP32 that when someone visits the server's IP address, it should run that function to send the webpage back.

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

This command needs to run repeatedly to constantly check if a device is trying to connect or send a request. If you remove it from the loop, the **ESP32** will stop listening for incoming connections, and your web page will never load.

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

This line sends the final response back to the user's browser. The code **200** means "**Success**," text/html tells the browser to expect a webpage, and the html variable contains the actual code for the page design.

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

Displaying the last measured value is instant because the data is already stored in a variable. Taking a fresh reading inside the function forces the code to wait for the sensor to process new data, which can make the web page load noticeably slower.

Part B: Long Question

System Overview

This project creates a "smart" monitor where the **ESP32** acts like a tiny website host. It connects to your home **Wi-Fi**, reads temperature and humidity data when you press a button, displays it on a small **OLED** screen, and also shows it on a web page that you can access from your phone or laptop.

1. ESP32 Wi-Fi Connection & IP Address

When the ESP32 starts up, the first thing it does is try to join your Wi-Fi network. You have to hardcode your network name (**SSID**) and **password** into the code.

- ❖ **The Process:** It sends a request to your router asking to join. The code usually has a while loop that keeps checking the connection status until it succeeds.
- ❖ **IP Assignment:** Once connected, the router assigns a unique local IP address (like **192.168.1.5**) to the **ESP32**. This address is crucial because it's the "phone number" your browser needs to call to see the web page.

2. Web Server Initialization & Request Handling

After the Wi-Fi is ready, the code sets up the web server, typically on **port 80** (the standard channel for web traffic).

- ❖ **Routing:** The code uses "routes" to decide what to do. The most common one is the root route **(/)**. The command **server.on("/", handleRoot)** tells the ESP32: "If someone visits the main homepage, run the **handleRoot** function."
- ❖ **Listening:** Inside the main **loop()**, the command **server.handleClient()** runs thousands of times per second. It acts like a receptionist, constantly waiting for a phone or laptop to send a request.

3. Button-Based Sensor Reading & OLED Update

Instead of reading the sensor constantly (which can slow things down), this system often uses a button to take a reading on demand.

- ❖ **The Button:** The code checks if the button pin has gone **LOW** (pressed).

- **Reading Data:** When pressed, the **ESP32** wakes up the **DHT11** sensor and asks for the current temperature and humidity.
- **OLED Display:** Immediately after getting the numbers, the **ESP32** draws them onto the attached **SSD1306 OLED** screen so you can see the data physically on the device.

4. Dynamic HTML Webpage Generation

The web page isn't a static file saved somewhere; the **ESP32** builds it from scratch every time you visit.

- **String Building:** The code creates a long text string (String variable) containing standard **HTML** tags (like **<html>, <h1>,** etc.).
- **Injecting Data:** The critical part is that it takes the temperature and humidity variables, converts them to text, and inserts them right into that HTML string.
- **Sending it:** Finally, **server.send(200, "text/html", htmlString)** ships this newly created page back to your browser.

5. Purpose of Meta Refresh

The HTML code often includes a line like **<meta http-equiv="refresh" content="5">**.

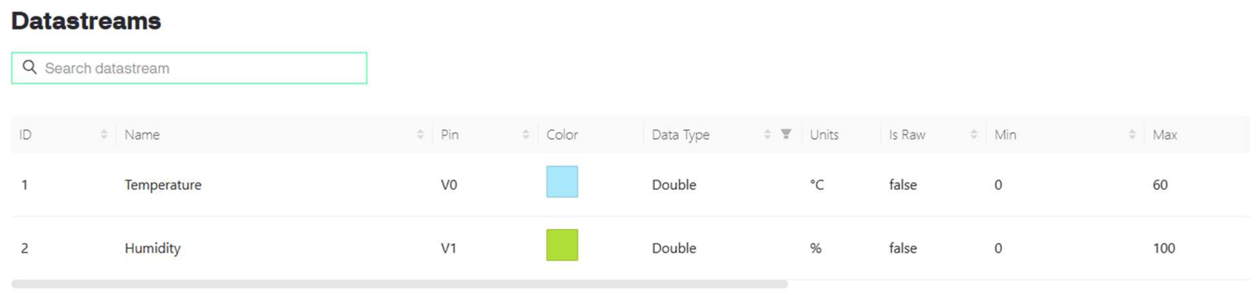
- **Why we need it:** Since the web page is just a snapshot of the data at that specific moment, it doesn't update automatically if the temperature changes.
- **What it does:** This tag tells your web browser to automatically reload the page every 5 seconds (or whatever number is set). This forces the browser to ask the **ESP32** for the page again, ensuring you see the latest sensor readings without having to hit the refresh button manually.

6. Common Issues & Solutions

- **"IP address not found":** This usually happens if the computer and **ESP32** are on different Wi-Fi networks (like one on **5GHz** and one on **2.4GHz** guest). *Solution:* Ensure both are on the exact same network.
- **Sensor shows "NaN" or 0:** This means "Not a Number." It happens if the sensor isn't wired correctly or the interval between readings is too fast for the **DHT11**. *Solution:* Check wiring and ensure you wait at least 2 seconds between readings.
- **Web page loads forever/times out:** This often happens if you have "blocking" code (like a **long delay(5000))** inside your main loop. This prevents the server from answering requests. Avoid **delay()** and use **millis()** timers instead.

Question-2 Blynk Cloud Interfacing (blynk.cpp)

Data Streams:

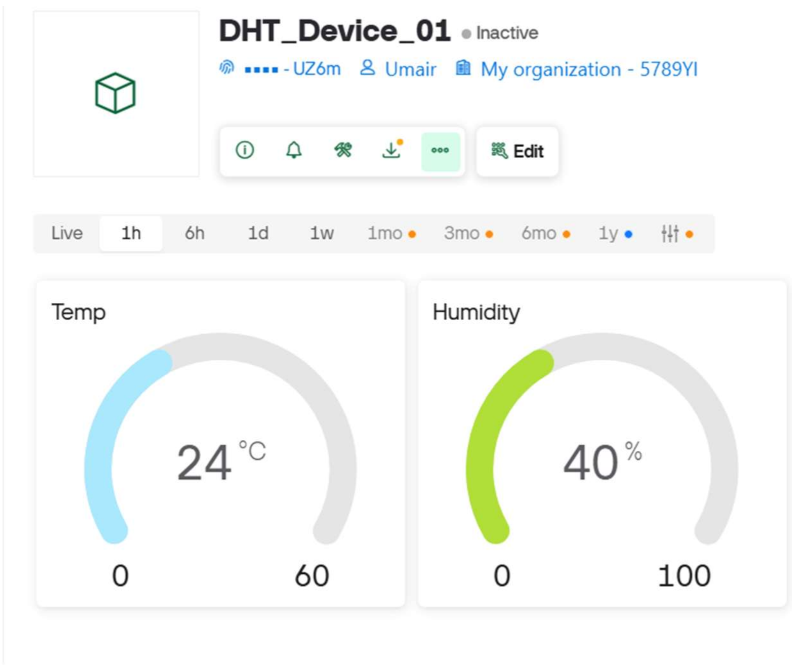


Datastreams

Search datastream

ID	Name	Pin	Color	Data Type	Units	Is Raw	Min	Max
1	Temperature	V0	🟦	Double	°C	false	0	60
2	Humidity	V1	🟩	Double	%	false	0	100

Dashboard:



Part A: Short Questions

1. What is the role of Blynk Template ID in an ESP32 IoT project? Why must it match the cloud template?

The Template ID acts like an ID card that tells the Blynk server exactly which dashboard layout and settings your device should use. It must match the cloud version perfectly because if the ID is wrong, the server won't know which widgets to link to your device, and the connection will fail.

2. Differentiate between Blynk Template ID and Blynk Auth Token.

The **Template ID** identifies the design of your project (like which buttons and charts exist), while the **Auth Token** is a unique password for your specific **ESP32** board. Basically, the ID defines what the project is, and the Token proves *who* the device is so it can log in.

3. Why does using DHT22 code with a DHT11 sensor produce incorrect readings? Mention one key difference between the two sensors.

The **DHT22** reads temperature with decimal precision (e.g., 25.4°C), while the **DHT11** typically only gives whole numbers, so they transmit data signals differently. If the code expects the detailed **DHT22** signal but gets the simpler **DHT11** one, the math fails, and you see random numbers.

4. What are Virtual Pins in Blynk? Why are they preferred over physical GPIO pins for cloud communication?

Virtual Pins (like V0, V1) are "imaginary" channels used to send data between the code and the phone app. They are preferred because they allow you to send any kind of data (like a calculated average or a text message) rather than just the raw High/Low voltage states that physical pins use.

5. What is the purpose of using BlynkTimer instead of delay() in ESP32 IoT applications?

The **delay()** function freezes the entire **ESP32**, which stops it from talking to the Wi-Fi and causes it to disconnect from the server. **BlynkTimer** is better because it handles timing in the background without stopping the processor, keeping the internet connection alive.

Part B: Long Question

Workflow: Connecting ESP32 to Blynk Cloud

This project is basically about creating a bridge between your hardware (the ESP32) and your phone (Blynk App). Here is step-by-step how the whole system works.

1. Creating the Blynk Template and Datastreams

Everything starts on the Blynk website (the Console). Before writing any code, you have to build the "digital home" for your device.

- **The Template:** You create a new Template, which is like a blueprint. You give it a name (e.g., "Weather Monitor") and tell it you are using an **ESP32**.
- **Datastreams:** This is the most important part. You can't just send "**Temperature**" to the cloud; you have to create specific channels for it. You set up **Virtual Pins** (V-Pins) here. For example, you might assign **V0** for Temperature and **V1** for Humidity. It's like tuning a radio: the **ESP32** transmits on channel **V0**, and the app listens on channel **V0**.

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

Once you finish the template, Blynk gives you three lines of code that are super important.

- **Template ID & Name:** These tell the cloud *what* kind of device is trying to connect. It matches your device to the blueprint you just made.

```
#define BLYNK_TEMPLATE_ID "TMPL6UCKQ_P6D"
#define BLYNK_TEMPLATE_NAME "dht"
```

- **Auth Token:** This is the secret password. Every single device gets a unique token. When your ESP32 connects to the internet, it shows this token to the Blynk server. If the token is wrong, the server rejects the connection immediately. You must paste these three lines at the very top of your Arduino code.

```
#define BLYNK_AUTH_TOKEN    "S2P_gLWBb5E-DINBtsXmlOcndOq6xJnU"
```

3. Sensor Configuration (DHT11 vs DHT22)

On the hardware side, you have to be careful which sensor you choose because the code is slightly different for each.

- **DHT11:** This is the blue one. It's cheaper but **less accurate**. It only gives whole numbers (like 25°C) and is slow—you can only read it once every 2 seconds.
- **DHT22:** This is the white one. It's **more precise** (gives 25.4°C) and reads a wider range of temperatures.
- **The Issue:** A common mistake is selecting the wrong type in the code (**e.g., #define DHTTYPE DHT11 when using a DHT22**). If you do this, the math will be wrong, and you'll usually see weird numbers or "NaN" (Not a Number) on your dashboard.

4. Sending Data using Blynk.virtualWrite()

This is the command that actually pushes the data from your board to the cloud.

- Instead of just printing to the Serial Monitor, you use commands like **Blynk.virtualWrite(V0, temperature);**
- **How it works:** This command takes the value stored in the temperature variable and pushes it out through the internet to Virtual **Pin V0.**
- **Important Note:** You should never put this inside the **loop()** without a timer. If you try to push data **100** times a second, the Blynk server will think you are spamming it and will disconnect you. You should use a timer to send it every 1 or 2 seconds.

5. Common Problems & Solutions

- **"Device Offline" in App:** This is the most annoying error. It usually means your Wi-Fi credentials **(SSID/Password)** in the code are wrong, or the **ESP32** is too far from the router.
- **Code Won't Compile:** This often happens if you forget to install the specific "**Blynk**" library in the Library Manager.
- **Random Disconnects:** If you use **delay()** in your main loop, the ESP32 stops talking to the server for that duration. The server thinks the device died and cuts the connection. *Solution:* Always use **BlynkTimer** instead of **delay().**