National Textile University Faisalabad

Department of Computer Science

Name:	**M. Muqaddam Ali**
Registration Number:	23-NTU-CS-1071
Semester and Section:	5th B
Course Name:	IoT Systems
Course Code:	IDK
Assignment Number:	02
Submitted To:	Sir Nasir Mehmood
Submission Date:	17th Dec - 2025

Assignment 02 – After Mids

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?

This line creates a web server object and tells it to listen on port 80.

Port 80 is the default port for websites. When you type a website address like google.com, your browser automatically connects to port 80. That's why you don't need to type the port number when visiting websites.

In our ESP32 project, using port 80 means we can access the webpage by just typing the ESP32's IP address (like 192.168.1.100) without adding any port number at the end.

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

This line connects a URL path to a function. The "/" means the homepage or root page. The handleRoot is the function name.

So basically, when someone opens the ESP32's IP address in their browser, the server will automatically run the handleRoot() function. This function creates the webpage with temperature and humidity data and sends it back to the browser.

It's like setting up a rule: "When someone visits the homepage, run this specific function."

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

The server.handleClient(); needs to be in the loop because it checks for new web requests constantly.

Why it's in loop:

- The ESP32 needs to keep checking if anyone is trying to access the webpage
- Every time the loop runs, it checks for new visitors
- If someone made a request, it processes it and sends back the webpage
- This happens over and over, many times per second

What happens if you remove it:

- The web server stops working completely
- When you try to open the webpage in your browser, nothing happens
- The browser will show "Cannot connect" or timeout errors
- All the web server setup becomes useless because it can't actually serve any pages

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

This line sends the webpage back to the person's browser. It has three parts:

200: This is a status code that means "Success" or "OK." It tells the browser that everything worked fine. Other codes like 404 mean "Page Not Found."

"text/html": This tells the browser what type of content it's receiving. "text/html" means it's a webpage with HTML code, so the browser knows to display it as a webpage instead of just plain text.

html: This is the variable containing all the HTML code - the actual webpage with temperature and humidity data.

Together, this sends a proper response that the browser can understand and display.

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

Using Last Measured Values (what the code does now):

- Shows the temperature and humidity that were saved when the button was pressed
- The webpage loads instantly with no waiting
- Data might be old if the button wasn't pressed recently
- Better when multiple people are checking the webpage at the same time
- No delay or freezing

Taking Fresh Reading (alternative way):

- Gets new temperature and humidity data every time someone opens the webpage

- Always shows current data
- Makes the webpage load slower (takes about 2 seconds to read the sensor)
- Can cause errors if too many people refresh the page at once
- Might overwork the sensor

The current way is better because the webpage loads fast, and the button gives you control over when to take new readings. Plus, the page auto-refreshes every 5 seconds anyway.

Part B: Long Question

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

1. Starting Up (setup( ) function)

When you first power on the ESP32:

Hardware Setup:

- Serial monitor starts at 115200 speed for debugging messages
- Button pin (GPIO5) is set up with INPUT_PULLUP mode - this means it reads HIGH normally and goes LOW when you press it
- OLED display (the small screen) starts up on I2C address 0x3C
- DHT22 temperature sensor on GPIO23 gets initialized

What the OLED Shows: The screen displays "Booting..." so you know it's starting up.

2. Connecting to WiFi

How it Connects:

- ESP32 tries to connect to the WiFi network (in this code it's "Wokwi-GUEST")
- During connection, the OLED shows "WiFi Connecting..."
- A while loop keeps trying until it connects successfully
- Each attempt prints a dot in the serial monitor

Getting an IP Address:

- Once connected, the router gives the ESP32 an IP address (like 192.168.1.100)

- This IP shows up on both the serial monitor and OLED screen
- You need this IP address to open the webpage in your browser

OLED Instructions: After connecting, the screen shows the IP address and says "Press button to read DHT" to tell you how to get sensor readings.

3. Starting the Web Server

- The web server starts listening on port 80
- The route "/" is connected to the handleRoot() function
- server.begin() makes everything ready to accept visitors

Now the ESP32 is fully ready and waiting for two things: button presses or people visiting the webpage.

4. Main Loop - What Happens Constantly

The loop function runs continuously, doing two main jobs:

Handling Web Requests: server.handleClient() checks if anyone is trying to open the webpage. It runs hundreds of times per second, so it catches requests immediately.

Watching the Button: The code checks if the button is pressed by:

- Reading the current button state (HIGH or LOW)
- Comparing it to the previous state
- If it changed from HIGH to LOW (falling edge), that means someone pressed it
- A 50ms delay helps with "debouncing" - preventing false readings from the button bouncing
- If it's still LOW after the delay, the button press is confirmed

5. What Happens When You Press the Button

Reading the Sensor: The readDHTValues() function runs and:

- Asks the DHT22 for humidity reading
- Asks for temperature reading in Celsius
- Checks if the readings are valid using isnan() (is not a number)
- If valid, saves them in lastTemp and lastHum variables

- Prints the values to serial monitor for debugging
- If readings fail, shows an error message

Updating the OLED: The showOnOLED() function:

- Clears the screen
- If sensor had errors, displays "DHT Error!"
- If readings are good, shows:
    - "DHT22 Readings" as a title
    - Temperature with °C
    - Humidity with %
- Pushes everything to the physical screen

This way you only read the sensor when needed, which is better for the sensor's lifespan.

6. When Someone Opens the Webpage

Processing the Request:

- Browser sends a request to the ESP32's IP address
- server.handleClient() catches this request
- Since the URL is just "/" (the homepage), it calls handleRoot()

Creating the Webpage: The function builds an HTML string with:

Header Tags:

- UTF-8 encoding for proper characters
- Viewport tag for mobile phones
- Auto-refresh every 5 seconds

Page Content:

- Title: "ESP32 DHT Monitor"
- Heading: "ESP32 DHT22 Readings"
- If no valid data yet: shows "No valid data yet" message

- If data is available: shows temperature and humidity with one decimal place
- Instructions to press the button for updates

Sending the Response: Uses server.send(200, "text/html", html); to send everything to the browser.

7. Why Auto-Refresh is Useful

The <meta http-equiv='refresh' content='5'> tag refreshes the page every 5 seconds.

Benefits:

- You don't need to manually click refresh
- See new values automatically if you pressed the button
- Good balance - not too fast (annoying) or too slow (outdated)
- Acts like a live monitoring system

8. Common Problems and Fixes

Problem: Can't Open the Webpage

- Make sure your phone/computer is on the same WiFi network
- Double-check the IP address from the OLED screen

Problem: Sensor Always Shows Error

- Check if DHT22 is plugged into GPIO23 correctly
- Make sure you defined DHTTYPE as DHT22 (not DHT11)
- DHT needs at least 2 seconds between readings
- Try adding a 10kΩ resistor between VCC and data pin
- Sensor might be damaged

Problem: Webpage Doesn't Update

- Make sure server.handleClient() is in the loop
- Clear your browser cache (Ctrl+F5)

Problem: OLED Screen is Blank

- Verify I2C address is 0x3C (sometimes it's 0x3D)

- Check SDA and SCL wire connections
- Make sure display.begin() returned true

Problem: Button Doesn't Work

- Confirm INPUT_PULLUP mode is set
- Check if debounce delay is there (it is in this code)

Problem: WiFi Keeps Disconnecting

- ESP32 might be too far from router
- Power supply might be weak (ESP32 needs 500mA)
- Add code to reconnect automatically if connection drops

Question 2: Blynk Cloud Interfacing (blynk.cpp)

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 is like an identification number that connects your ESP32 code to your Blynk project in the cloud. It acts as a reference to your project setup and tells Blynk which dashboard and widgets to use. It also helps Blynk recognize what type of device is connecting and lets multiple ESP32 boards use the same project design.

The Template ID in your code has to be exactly the same as what's in Blynk Cloud because Blynk checks this ID to make sure your device belongs to your account. It needs to know which virtual pins go to which widgets, and your dashboard won't load correctly if IDs don't match. If they don't match, the ESP32 won't connect and your sensor data won't appear on the dashboard.

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

Template ID tells Blynk what type of project this is and it's the same for all devices using this project design. It's like a blueprint for a house that defines the structure and layout. Auth Token tells Blynk which specific device this is and it's different for every single device. It's like a key for a specific house used for security and authentication.

If you make 10 temperature monitors, all 10 use the same Template ID but each gets its own Auth Token. The Template ID is public info but the Auth Token should be kept secret because it gives access to control your device.

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

Using DHT22 code with a DHT11 sensor gives wrong readings because these sensors work differently. The DHT22 sends 16-bit data which is more detailed, while DHT11 sends only 8-bit data which is less detailed. The DHT22 library expects more data than DHT11 provides. Each sensor also has different timing for sending signals, so the library waits for specific pulse lengths. When the timing is wrong, the data gets corrupted.

DHT22 can send decimal values like 25.6°C but DHT11 only sends whole numbers like 25°C. When DHT22 code tries to read DHT11 data, it gets confused and you might see crazy values like 55°C when it's actually 25°C.

One key difference is that DHT22 has a temperature range of -40°C to 80°C with ±0.5°C accuracy and shows decimals, while DHT11 only works from 0°C to 50°C with ±2°C accuracy and only shows whole numbers. DHT22 is more accurate and has a bigger range but it costs more.

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

Virtual Pins are software pins that don't actually exist on the ESP32 board. They only exist in code and in the Blynk cloud, numbered V0, V1, V2, etc. up to V255. In the code we use them like `Blynk.virtualWrite(V0, t)` to send temperature data.

They're better than physical pins because ESP32 has only about 34 GPIO pins but you can use 256 virtual pins without running out. Physical pins only do HIGH/LOW or 0-1023 but virtual pins can send numbers, text, decimals, anything. If you move your DHT sensor from GPIO23 to GPIO15, you don't need to change anything in Blynk dashboard, just update the pin number in your code.

Virtual pins also work both ways for sending data to cloud and receiving commands from cloud, while physical pins would need complex setup for this. They protect your hardware from bad cloud commands because you control what happens when a virtual pin changes. One virtual pin can also update many widgets at once, like showing V0 on a gauge, chart, and label all at the same time.

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

BlynkTimer is used instead of delay() because delay() freezes everything but IoT apps need to do multiple things at once. When you use `delay(5000)` it freezes for 5 seconds and during that time the Blynk connection will timeout and disconnect, button presses are ignored, web requests fail, and nothing else can happen.

BlynkTimer is better because everything keeps running. The Blynk connection stays alive, buttons still work, and multiple timers can run together. You can have one timer running every 5 seconds for sensor reading, another every minute for reports, and another every second for checking buttons. The system stays responsive so user inputs work immediately and the cloud connection stays active.

In the loop function, you just call `Blynk.run()` and `timer.run()` continuously, and they both execute hundreds of times per second without blocking anything. BlynkTimer lets you schedule tasks without stopping everything else, which is essential for IoT projects.

Part B: Long Question

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

Setting Up Blynk Cloud:

1. First you need to create a template on blynk.cloud. After logging in, you go to Templates and click New Template. You give it a name like "DHT Monitor", select ESP32 as hardware, and WiFi as connection type. This template is like a blueprint for your project that defines how everything should work.

2. Next you create datastreams which are channels that carry your data. For temperature, you create a virtual pin V0 with settings like name "Temperature", type Double for decimals, units in °C, minimum -40, maximum 80, and show 1 decimal place. For humidity you create V1 with similar settings but units in %, minimum 0, maximum 100. These datastreams tell Blynk what kind of data to expect and how to display it.

3. After datastreams, you design the web dashboard by dragging widgets from the left side. You add a Gauge widget for temperature and connect it to the V0 datastream, then add another Gauge for humidity connected to V1. You can arrange them nicely and change colors and sizes. The mobile dashboard is similar but you make the widgets bigger for easier touch and arrange them for portrait view.

4. Finally you create your actual device by going to Devices, clicking New Device, selecting From Template, and picking your DHT Monitor template. You give it a name like "Living Room Sensor" and Blynk generates three important things: Template ID, Template Name, and Auth Token. You need to save these exact values because they go in your code.

Writing the ESP32 Code:

You put the three credentials at the very top of your code before including any libraries. The Template ID and Auth Token must be copied exactly from Blynk Console with no typos. Then you include the necessary libraries like WiFi, BlynkSimpleEsp32, DHT, and the display library. In the setup function you initialize the hardware like serial communication, button pin with INPUT_PULLUP, OLED display, and DHT sensor.

The most important line is `Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass)` which does a lot of work. It connects to WiFi, gets an IP address, connects to Blynk cloud, sends the Auth Token for verification, and loads your template settings. You also setup a timer with `timer.setInterval(5000L, periodicSend)` which makes the ESP32 send data every 5 seconds automatically.

How the System Runs:

In the main loop you must call `Blynk.run()` constantly to keep the connection alive. This function receives commands from the app, sends buffered data, and must be called thousands of times per second or the connection will drop. You also call `timer.run()` which checks if any timers should execute their functions.

When it's time to read the sensor, the function reads humidity and temperature from the DHT22. It checks if the readings are valid using isnan() and if there's an error it returns early. For valid readings, it updates the OLED display by clearing it, printing the temperature and humidity with units, and calling display.display() to show everything. Then it sends the data to Blynk using `Blynk.virtualWrite(V0, t)` for temperature and `Blynk.virtualWrite(V1, h)` for humidity.

When you call virtualWrite, the data gets packaged up and sent over WiFi to the internet. It reaches the Blynk cloud server which updates the V0 and V1 datastreams. All widgets connected to these virtual pins update instantly and your phone app gets notified. The whole process takes about 100-500 milliseconds depending on internet speed.

Dashboard Updates and Common Problems:

On the web browser the gauge widgets move to new values, numbers change, and charts add new data points automatically. On the mobile app you get a notification if it's running and widgets update when you open it. You can also see history in the charts.

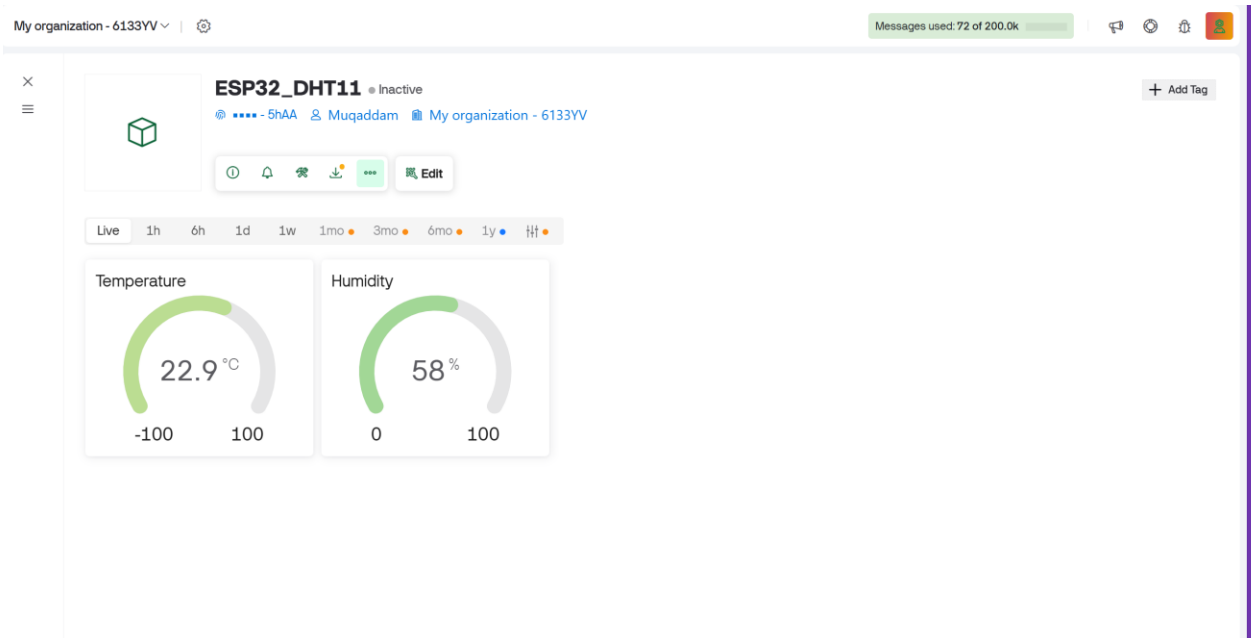
The most common problem is the device won't connect to Blynk. You should check if the Template ID and Auth Token are copied exactly with no spaces or typos. Make sure the WiFi name and password are correct and that ESP32 is connected to 2.4GHz WiFi not 5GHz. If it connects but shows no data, check that you created the V0 and V1 datastreams and that widgets are connected to the right virtual pins.

If the connection keeps dropping, make sure `Blynk.run()` is in the loop function and you're not using delay() instead of timer. The power supply needs to be strong enough to provide

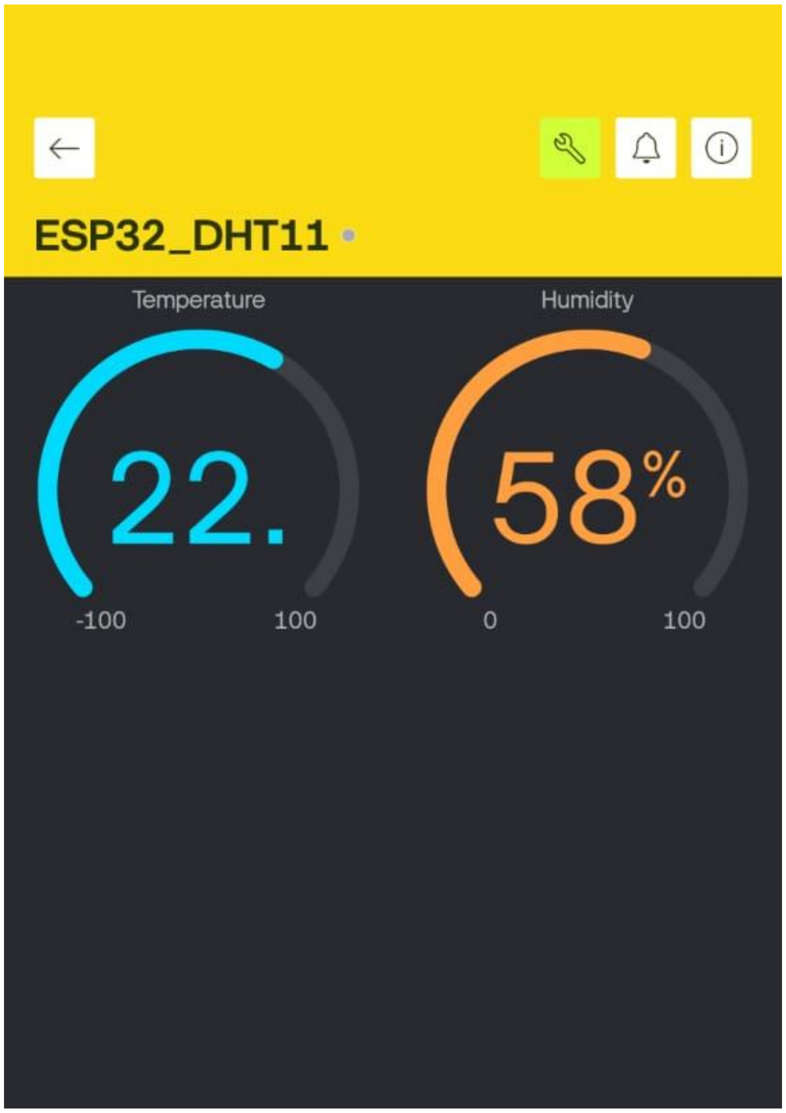
500mA and the WiFi signal shouldn't be too weak. For sensor errors, check that DHTTYPE is set to the right sensor type, the wiring is correct with data pin to GPIO23, and you're waiting at least 2 seconds between readings.

The complete data flow works like this: you press the button, ESP32 reads the sensor, temperature and humidity are captured, the OLED shows the values locally, data is sent to Blynk cloud via WiFi, the cloud updates the datastreams, and dashboard widgets update so you see it on your phone or computer. This system is useful because you can check temperature from anywhere in the world, multiple people can view at the same time, data gets logged automatically, you can set up alerts for high or low temps, and the local OLED display still works even without internet.

Blynk Cloud (Web):



Blynk Mobile:


ESP32_DHT11
Temperature
22.
-100
100
Humidity
58%
0
100