National Textile University, Faisalabad

Department of Computer Science

Name:	Noor Ul Huda
Class:	BSCS_B 5th Semester
Registration No:	23-NTU-CS-1084
Assignment :	02
Course Name:	Embedded IOT System
Submitted To:	**Sir Nasir Mehmood**
Submission Date:	17/12/2025

Assignment 02

Question #01:

ESP32 Webserver

Part A: Short Questions :

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

This line creates a web server object inside the ESP32 so it can act like a small website. Port 80 is the standard HTTP port, which means the web page can be opened directly in any browser using the ESP32 IP address without adding any extra port number.

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

This statement defines what should happen when a user opens the main page of the ESP32 web server.
When the root URL (/) is requested, the function handleRoot() is called, which prepares and sends the webpage content to the browser.

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

It is placed inside loop() so the ESP32 can continuously check for incoming browser requests.
If this line is removed, the ESP32 will connect to Wi-Fi but will not respond to any HTTP requests, so the webpage will never load or refresh.

Q#4. In handleRoot(), explain the statement:

server.send(200, "text/html", html);

This statement sends the webpage response to the client.
The **200** status code indicates the request was successful, **text/html** tells the browser that the content is a web page, and **html** contains the actual data that will be displayed on the screen.

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

Displaying the last measured values is safer and faster because DHT sensors need time between readings.
Taking a fresh reading inside handleRoot() may cause slow page loading and unreliable values due to frequent sensor access.

Part B: Long Question:

Describe the complete Working of ESP32 Webserver-Based Temperature and Humidity Monitoring System.

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

The ESP32 connects to the Wi-Fi network using predefined SSID and password.
After successful connection, the router assigns an IP address, which is used to access the ESP32 web server through a browser.

2. Web Server Initialization and Request Handling:

The web server is initialized using server.begin().
Specific URLs are linked with handler functions using server.on().
The function server.handleClient() continuously listens for incoming requests and serves the appropriate response.

3. Button-Based Sensor Reading and OLED Update Mechanism:

A push button is used to manually trigger DHT sensor readings.
When pressed, the ESP32 reads temperature and humidity and updates the OLED display, reducing unnecessary sensor polling and improving system stability.

4. Dynamic HTML Webpage Generation:

The ESP32 dynamically generates the HTML page by embedding real-time sensor values into the webpage code.
This allows the browser to show updated temperature and humidity data without storing static values.

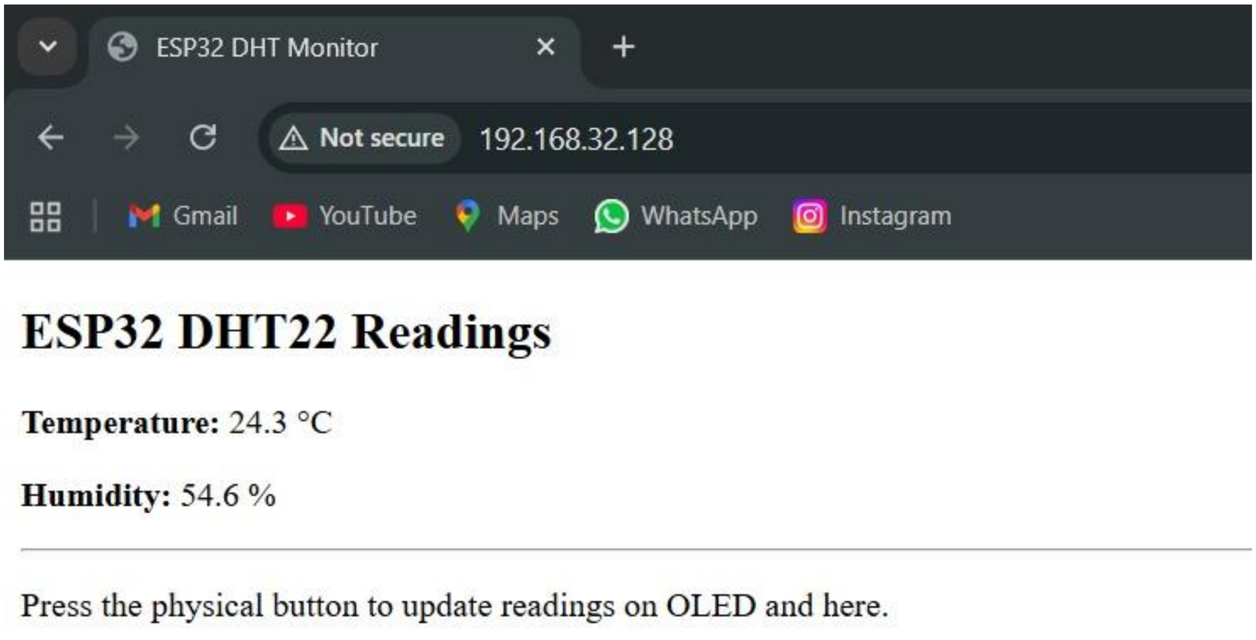
5. Purpose of Meta Refresh in the Webpage:

Meta refresh automatically reloads the webpage after a fixed time interval.
This ensures updated sensor values are displayed without requiring the user to manually refresh the page.

6. Common Issues in ESP32 Webserver Projects and Their Solutions:

- **Webpage not opening:** Check Wi-Fi connection and correct IP address.
- **Incorrect sensor readings:** Avoid reading DHT sensor too frequently.

- **ESP32 freezing:** Do not use long delay() statements.
- **Slow response:** Keep sensor reading separate from HTTP request handling.



ESP32 DHT22 Readings

Temperature: 24.3 °C

Humidity: 54.6 %

Press the physical button to update readings on OLED and here.

Question #02:

Blynk Cloud Interfacing

Part A: Short Questions :

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

The Template ID links the ESP32 code to a specific Blynk cloud project.
If it does not match the cloud template, the ESP32 will fail to connect and data will not appear on the dashboard.

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

Template ID defines the project structure and widgets on the cloud.
Auth Token is a unique key that authorizes a specific ESP32 device to communicate securely with that template.

Q#3. Why does using DHT22 code with a DHT11 sensor produce incorrect readings? Mention one key difference.

DHT11 and DHT22 use different data formats and measurement ranges.
DHT22 provides higher accuracy and wider range, while DHT11 has limited precision and slower response.

Q#4. What are Virtual Pins in Blynk? Why are they preferred over physical GPIO pins?

Virtual Pins are software-based communication channels between the ESP32 and Blynk Cloud. They are preferred because they are flexible, scalable, and independent of physical hardware pins.

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

BlynkTimer allows scheduled tasks without blocking program execution.
Using delay() can freeze the ESP32, disrupt Wi-Fi communication, and cause Blynk server disconnection.

Part B: Long Question:

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

1. Creation of Blynk Template and Datastreams:

First, a template is created in the Blynk Cloud dashboard.
Datastreams for temperature and humidity are added and linked with virtual pins.
These datastreams define how data will be shown on the Blynk app.

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

The Template ID connects the ESP32 firmware with the correct cloud project.
Template Name helps in identifying the project easily on the dashboard.
The Auth Token is used to authenticate the ESP32 and allow secure data transfer.

3. Sensor configuration issues (DHT11 vs DHT22):

DHT11 and DHT22 sensors work differently and have different data formats.
Using the wrong sensor definition in the code leads to incorrect readings.
Selecting the correct sensor type ensures stable and accurate values.

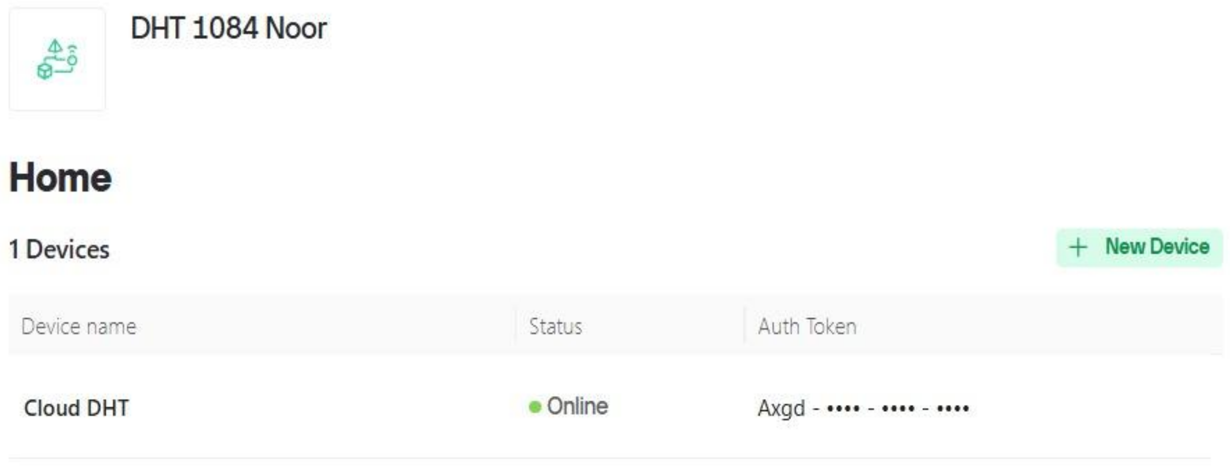
4. Sending data using Blynk.virtualWrite():

After reading the sensor, the ESP32 sends data to Blynk Cloud using virtual pins.
The Blynk.virtualWrite() function updates the values on widgets like labels and gauges.
This allows real-time monitoring from anywhere.

5. Common problems faced and their solutions:

Common issues include device going offline, data not updating, or frequent disconnections.
These problems are usually caused by wrong tokens, incorrect virtual pins, or using delay().
Using BlynkTimer and proper configuration helps keep the system stable.

Screenshots:

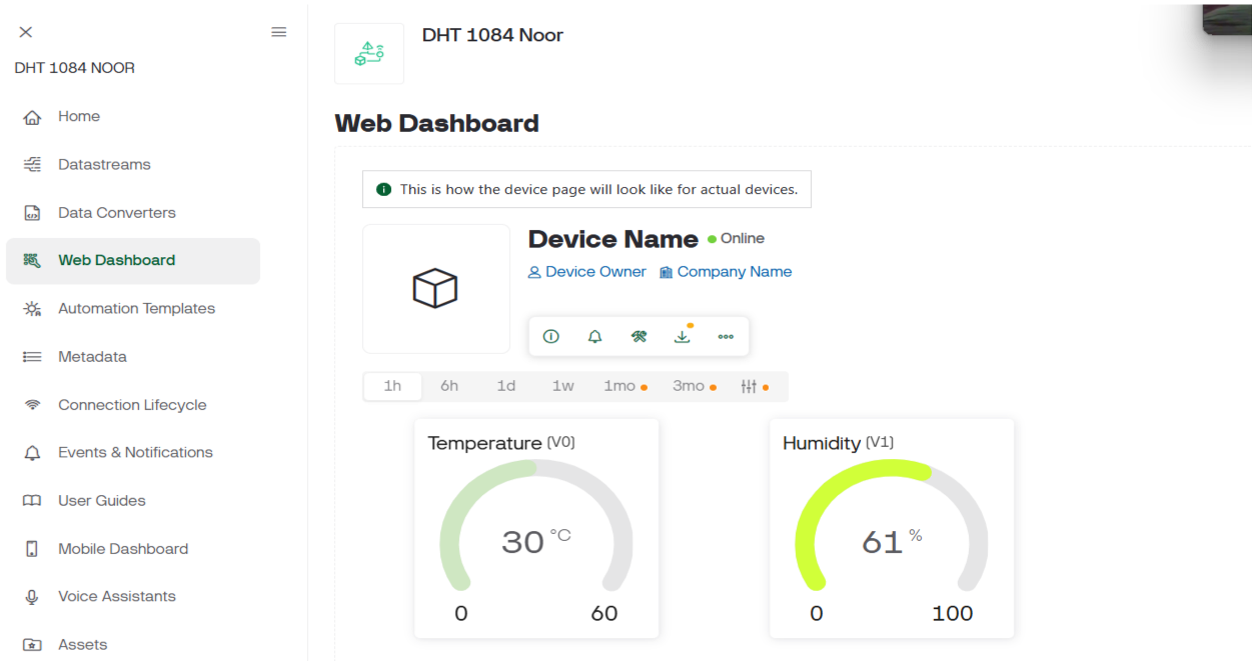


DHT 1084 Noor

Home

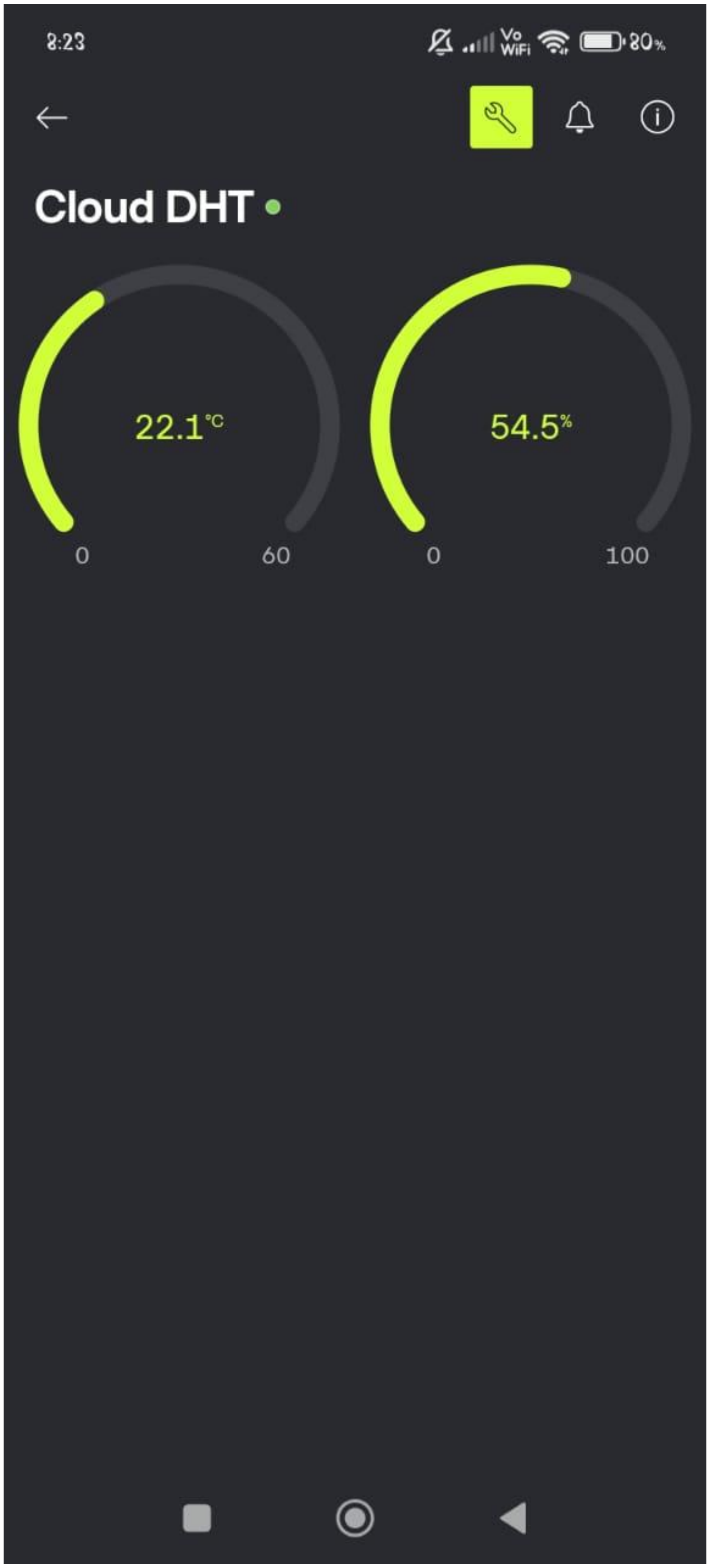
1 Devices

Device name	Status	Auth Token
Cloud DHT	● Online	Axgd - •••• - •••• - ••••



This is how the device page will look like for actual devices.

Device Name ● Online
Device Owner Company Name


8:23
80%
Cloud DHT
22.1°C
0
60
54.5%
0
100