National Textile University, Faisalabad

Department of Computer Science

Name:	Rizwana Bashir
Class:	BSCS-5th-B
Registration No:	23-NTU-CS-FL-1088
Homework No:	1
Submission Date:	17-Dec-2025
Course Name:	Embedded IoT Systems
Submitted To:	Sir Nasir Mahmood

Embedded IoT Systems Homework-01

Question-1 ESP32 Webserver

Part-A Short Questions

1. **<u>Webserver server(80); creates a web server object on the microcontroller (ESP32/ESP8266).</u>**

- **Purpose:**
  It allows the microcontroller to act like a **web server**, so it can send web pages (HTML) to a browser.
- **Port 80:**
  Port **80** is the **default port for HTTP (web) communication**.
  When you type an IP address in a browser (without specifying a port), the browser automatically uses port 80.

So, this means it Create a web server that listens for HTTP requests on port 80.

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

This line defines what happens when a client accesses the root URL (/).

- / Means the **home page** (for example: http://192.168.1.10/)
- handleRoot is a **function** that runs when this page is requested.

Simply when someone opens the ESP's IP address in a browser, the handleRoot() function is called.

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

server.handleClient(); **checks for incoming web requests** from browsers. It must run **continuously**, which is why it is placed inside loop().

The loop() function runs again and again, allowing the server to:

- Accept new connections.

- Respond to client requests

If we **remove** server.handleClient();

- The web server will **not respond**
- The browser will keep loading or show **connection failed**
- No webpage will be displayed

So, this line “server.handleClient();” is **essential** for the web server to work properly.

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

This line server.send(200, "text/html", html); sends a response to the browser.

200: HTTP status code meaning “OK / Successful request.”

"text/html" :Tells the browser that the content is HTML.

html: The actual HTML content being sent (webpage code)

Simply it sends an HTML webpage to the browser with a successful status.

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

Displaying last measured values

- Sensor is read earlier (e.g., in loop())
- Webpage shows stored values.
- Faster response
- Less load on the DHT sensor
- More reliable

Taking fresh DHT reading inside handleRoot()

- Sensor is read every time the page is refreshed.
- Slower response
- DHT sensors are slow and sensitive.
- May cause NaN (invalid readings) if accessed too frequently.

Part-B Long Question

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

This system uses an **ESP32**, a **DHT11 temperature and humidity sensor**, an **OLED display**, and a **push button** to monitor environmental conditions. The ESP32 hosts a **web server** that displays the latest sensor readings on a dynamically generated webpage, while the OLED shows the same data locally when the button is pressed.

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

When ESP32 starts, it initializes the Wi-Fi module using the WiFi.h library.

```
WiFi.begin(ssid, password);
```

- The ESP32 connects in **station (STA) mode** to the Wi-Fi network named **"NTU FSD"**
- Since no password is provided, it connects to an **open network.**
- The ESP32 continuously checks the connection status using:

```
while (WiFi.status() != WL_CONNECTED)
```

Once connected:

- The router assigns an **IP address** using **DHCP.**
- The assigned IP is printed on the **Serial Monitor.**
- The same IP address is displayed on the **OLED screen.**

The user uses this IP address to access the ESP32 web server from a browser.

2. Web Server Initialization and Request Handling

The ESP32 web server is created using:

```
WebServer server(80);
```

- Port **80** is the standard HTTP port.
- The ESP32 listens for incoming browser requests on this port.

The root URL (/) is linked to a handler function:

```
server.on("/", handleRoot);

server.begin();
```

- When a browser opens the ESP32 IP address, the handleRoot() function is executed.
- Inside the loop() function:

server.handleClient();

This line is essential because:

- It continuously checks for new client requests.
- It allows the ESP32 to respond to multiple browser refreshes.

Without this line, the web server would not respond.

3. <u>Button-Based Sensor Reading and OLED Update Mechanism</u>

A **push button** is connected to **GPIO 5** and configured using INPUT_PULLUP.

pinMode(BUTTON_PIN, INPUT_PULLUP);

Working:

- The button is normally HIGH.
- When pressed, it goes LOW.
- The code detects a **falling edge** (HIGH to LOW)

if (lastButtonState == HIGH && currentButtonState == LOW)

When the button is pressed:

1. The ESP32 reads temperature and humidity from the **DHT11 sensor.**
2. The values are stored in:
    - lastTemp
    - lastHum
3. The OLED is updated using showOnOLED()
4. The same values are later shown on the web page.

This mechanism:

- Prevents frequent DHT readings.
- Improves sensor reliability.
- Gives user-controlled updates.

4. <u>Dynamic HTML Webpage Generation</u>

The webpage is created dynamically inside the handleRoot() function.

String html = "<!DOCTYPE html><html>...";

Key features:

- HTML is built using a **String object.**
- Latest stored sensor values (lastTemp, lastHum) are embedded into the webpage.
- If no valid data is available, a message is shown asking the user to press the button.

The webpage is sent to the browser using:

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

This sends:

- **200:** It gives Successful HTTP response.
- **text/html:** Content type.
- **html:** The generated webpage

5. Purpose of Meta Refresh in the Webpage

The webpage includes the following line:

<meta http-equiv="refresh" content="5">

Purpose:

- Automatically refreshes the webpage every **5 seconds.**
- Updates displayed sensor values without manual reload.
- Simulates real-time monitoring.
- Keeps the design simple without JavaScript.

The refreshed page shows the **latest stored readings** updated by the button.

6. Common Issues in ESP32 Webserver Projects and Their Solutions

Issue	Cause	Solution
Webpage not loading:	server.handleClient() missing	Called inside loop()
ESP32 not connecting to Wi-Fi:	Wrong SSID or weak signal	Verify SSID and network
NaN sensor values:	Frequent DHT reads	Button-based controlled reading

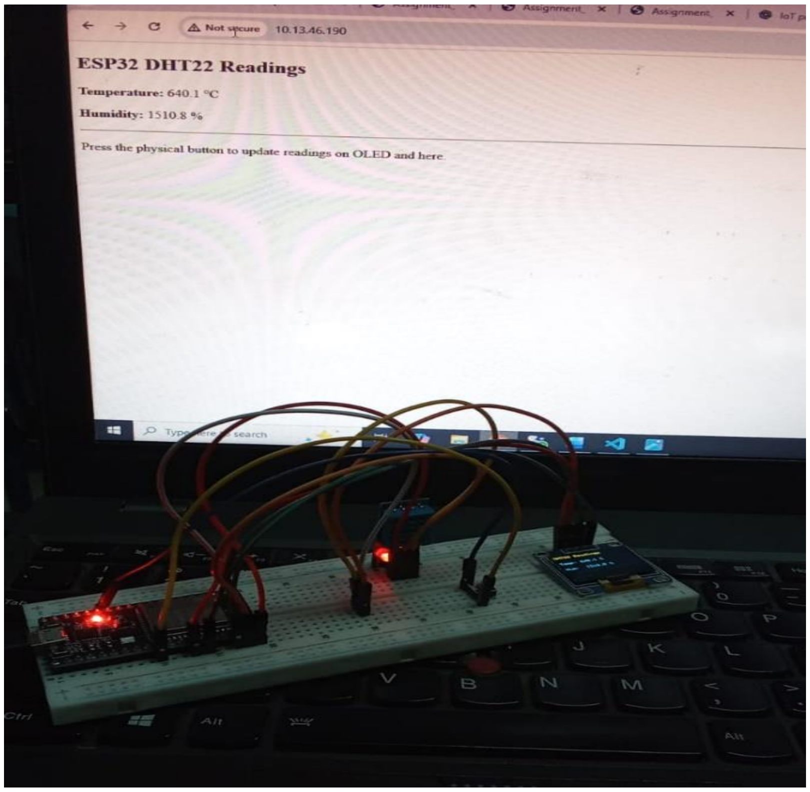
Slow webpage:	Reading DHT in handler	Use stored values
OLED not working:	Wrong I2C address	Correct address 0x3C
Button false triggers:	No debounce	Added 50 ms debounce
ESP32 resets:	Power instability	Use stable USB supply

Conclusion

This ESP32-based system efficiently combines **Wi-Fi networking, web server functionality, sensor interfacing, and local display control**. The button-controlled DHT reading ensures reliable sensor data, while the dynamically generated webpage allows remote monitoring through any browser. The use of meta refresh provides automatic updates, making the system simple, stable, and effective for IoT-based environmental monitoring.

Screenshots

DHT11 reading on webpage:



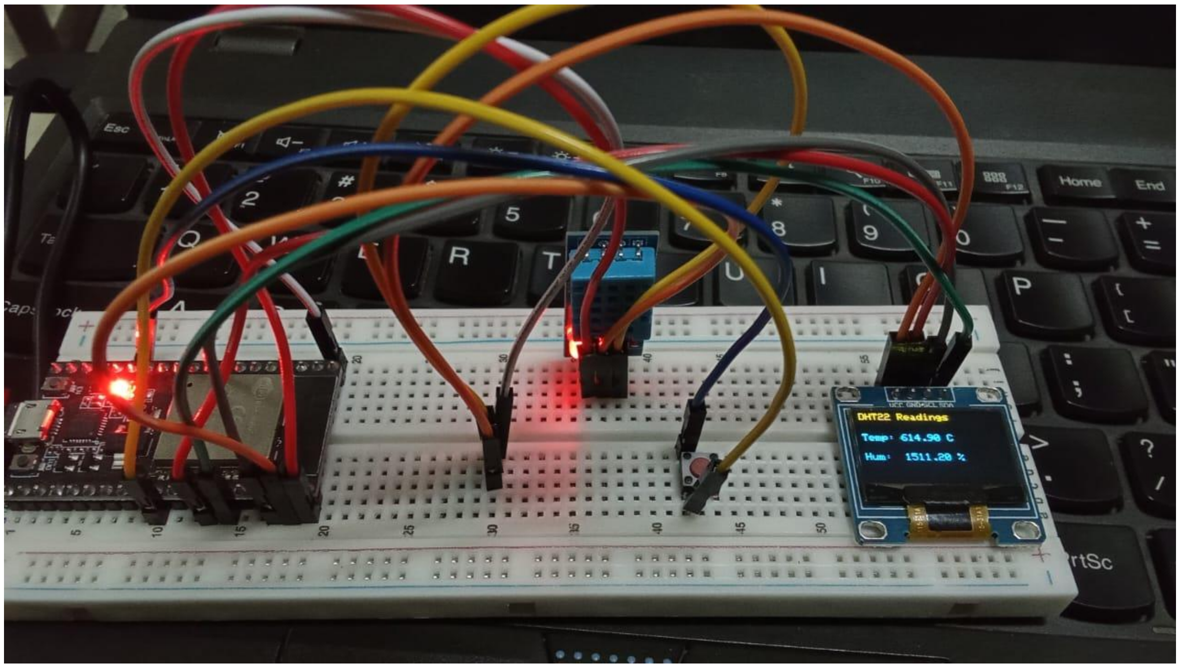
ESP32 DHT22 Readings

Temperature: 640.1 °C

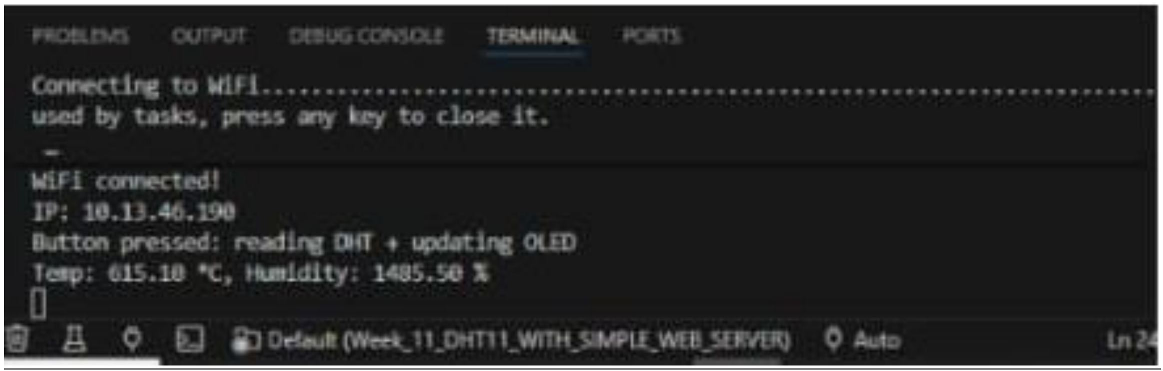
Humidity: 1510.8 %

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

Hardware:



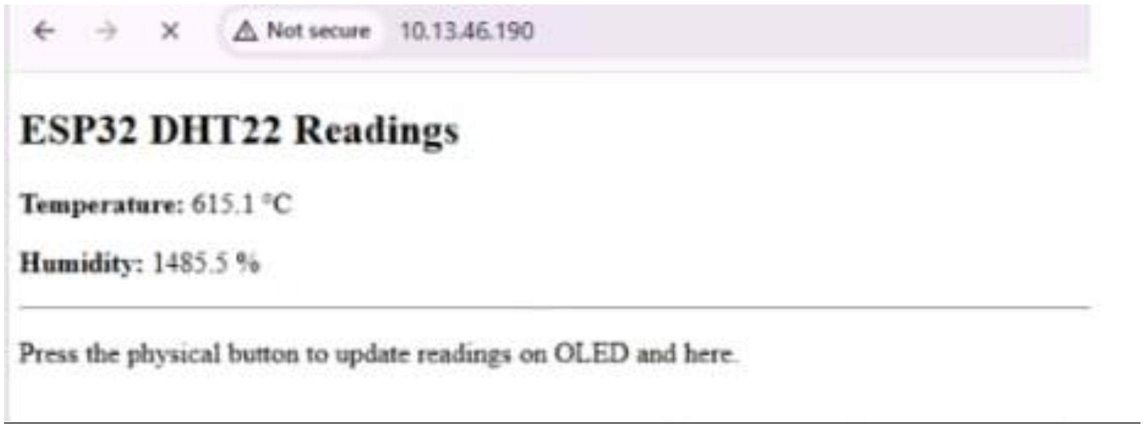
IP address connected to webpage:



```
Connecting to WiFi..........................................................
used by tasks, press any key to close it.

WiFi connected!
IP: 10.13.46.190
Button pressed: reading DHT + updating OLED
Temp: 615.10 °C, Humidity: 1485.50 %
```

Output webpage:



ESP32 DHT22 Readings

Temperature: 615.1 °C

Humidity: 1485.5 %

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

Question-2 Blynk Cloud Interfacing

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 **Blynk Template ID** uniquely identifies the **device template** created on the Blynk Cloud.

Role:

- Links the ESP32 firmware to a specific Blynk Cloud template**.**
- Defines available widgets, virtual pins, and device structure.
- Ensures correct dashboard layout and data mapping.

Why it must match:

- If the Template ID in the code does not match the cloud template:
  - Device fails to connect properly.
  - Widgets do not respond.
  - Data is not displayed on the Blynk app.

The Template ID ensures the ESP32 connects to the **correct cloud dashboard**.

2. Difference between Blynk Template ID and Blynk Auth Token

Aspect	**Blynk Template ID**	**Blynk Auth Token**
Purpose:	Identifies the device template	Authenticates a specific device
Scope:	Same for all devices using the template	Unique for each device
Use:	Dashboard structure & widget mapping	Secure cloud communication
Generated:	When template is created	When device is added

Security role:	Defines dashboard structure, provide no security	Defines device authentication and **ensures secure cloud access** ,provides security

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

Using DHT22 code with a DHT11 sensor causes incorrect readings because the two sensors have different communication protocols and data formats.

Key difference:

DHT11:

- Temperature range: 0–50 °C
- Accuracy: ±2 °C

DHT22:

- Temperature range: –40 to 80 °C
- Accuracy: ±0.5 °C

ESP32 interprets DHT11 data incorrectly when using DHT22 settings, leading to wrong values or NaN errors.

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

Virtual Pins in Blynk are **software-defined pins** used to exchange data between a hardware device (ESP32) and the **Blynk Cloud**, without being linked to any physical GPIO pin. They act as **data channels** in the cloud. Used to send sensor values, button states, and control commands. Do not control hardware directly.

Why preferred:

We preferred virtual pin due to following points.

- Not tied to physical hardware pins
- Can transmit sensor values, button states, and commands.
- One virtual pin can control multiple actions.
- Enables cloud-based communication and automation.

Physical GPIO pins control hardware while Virtual pins control **cloud data flow.**

5. Purpose of using BlynkTimer instead of delay() in ESP32 IoT applications?

BlynkTimer is a **non-blocking software timer** used in Blynk-based IoT applications to execute functions at fixed time intervals without stopping the main program execution. It allows the ESP32 to perform **periodic tasks** (like reading sensors or sending data). It keeps the Wi-Fi and Blynk cloud connection active.

Why BlynkTimer is better:

- delay() stops all code execution.
- Causes Wi-Fi disconnection.
- Freezes Blynk communication.

BlynkTimer advantages:

- Allows multitasking.
- Keeps Blynk cloud connection alive.
- Enables periodic sensor reading.
- Improves system stability and responsiveness.

delay() cause blocking bad for IoT. **BlynkTimer** is non-blocking best practice for IoT applications.

Part-B Long Question

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

This project demonstrates an IoT-based environmental monitoring system using **ESP32**, **DHT sensor**, **OLED display**, **push button**, and **Blynk Cloud**. The system measures temperature and humidity, displays the values locally on an OLED screen, and remotely on the Blynk mobile dashboard using virtual pins.

1. Creation of Blynk Template and Datastreams

Before programming the ESP32, a **Blynk Template** is created on the Blynk Cloud.

Steps involved:

1. Firstly, I create a new template in Blynk Console
   - Template Name: **“DHT Monitor”**
2. Then, Add **Datastreams** for temperature and humidity.
   - **V0** is Temperature (Float)

    - **V1** is Humidity (Float)

3. Add widgets in the Blynk mobile app:
    - Two Gauge widgets
    - Map them to **V0** and **V1**

These datastreams act as communication channels between ESP32 and Blynk Cloud.

2. <u>Role of Template ID, Template Name, and Auth Token</u>

In the code, the following identifiers are used:

```
#define BLYNK_TEMPLATE_ID "TMPL6Tc4OgmBL"

#define BLYNK_TEMPLATE_NAME "DHT Monitor"

#define BLYNK_AUTH_TOKEN "U-vtYQV4x6JvBtA7xGxvyg71qu51S1oC"
```

Explanation of their roles:

- **Template ID**
    - Links the ESP32 firmware to the correct Blynk Cloud template.
    - Ensures correct dashboard and datastream mapping.
- **Template Name**
    - Human-readable name for identification
    - Used for documentation and cloud reference.
- **Auth Token**
    - Acts as a **security key.**
    - Authenticates the ESP32 device with the Blynk Cloud.
    - Without a valid Auth Token, the device cannot connect.

3. <u>Sensor Configuration Issues (DHT11 vs DHT22)</u>

In the code:

```
#define DHTPIN  23

#define DHTTYPE DHT11
```

Important consideration:

- The sensor type in code must match the physical sensor.

- Using DHT22 code with a DHT11 sensor (or vice versa) causes:
    - Incorrect readings
    - NaN (Not a Number) values
    - Communication errors

Key difference:

Feature	**DHT11**	**DHT22**
Temperature range	0–50°C	–40 to 80°C
Accuracy	±2°C	±0.5°C
Resolution	Lower	Higher

Correct configuration ensures reliable sensor readings.

4. Sending Data Using Blynk.virtualWrite()

After reading sensor values, the ESP32 sends data to the Blynk Cloud using **Virtual Pins**.

Blynk.virtualWrite(V0, t); // Temperature

Blynk.virtualWrite(V1, h); // Humidity

How it works:

- virtualWrite() sends data to the cloud.
- Virtual pins (**V0, V1**) are linked to widgets in the Blynk app.
- Widgets update in real time with new sensor values.

This method is preferred because:

- It is hardware independent.
- Ideal for cloud-based communication.
- Supports multiple data types.

5. Button-Based Reading, OLED Display, and Timer Operation

Button mechanism:

- Button connected to **GPIO5**
- Configured as INPUT_PULLUP
- Active LOW (pressed LOW)

When pressed:

- The ESP32 reads temperature and humidity.
- Updates the OLED display.
- Sends updated values to Blynk.

OLED display:

- Uses I2C (GPIO21 SDA, GPIO22 SCL)
- Displays:
    - Temperature
    - Humidity
    - System status

BlynkTimer usage:

timer.setInterval(5000L, periodicSend);

- Reads and sends data every 5 seconds.
- Non-blocking.
- Keeps Blynk connection alive.
- Replaces delay().

6. <u>Common Problems Faced During Configuration and Their Solutions</u>

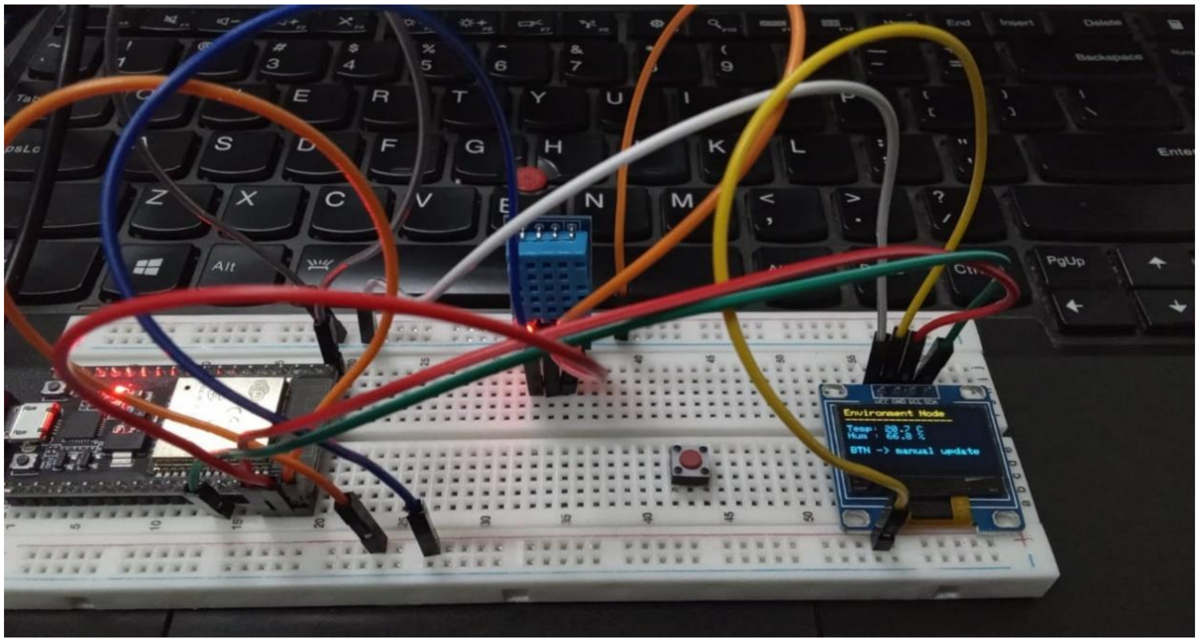
<u>Problem</u>	**<u>Cause</u>**	**<u>Solution</u>**
ESP32 not connecting to Blynk:	Wrong Auth Token	Use correct token
Widgets not updating:	Wrong virtual pin	Match V0, V1 correctly
NaN sensor values:	Wrong DHT type	Match DHT11/DHT22
Device disconnects:	Using delay()	Use BlynkTimer
OLED not displaying:	Wrong I2C address	Use 0x3C
Button misbehavior:	No pull-up	Use INPUT_PULLUP

<u>Conclusion</u>

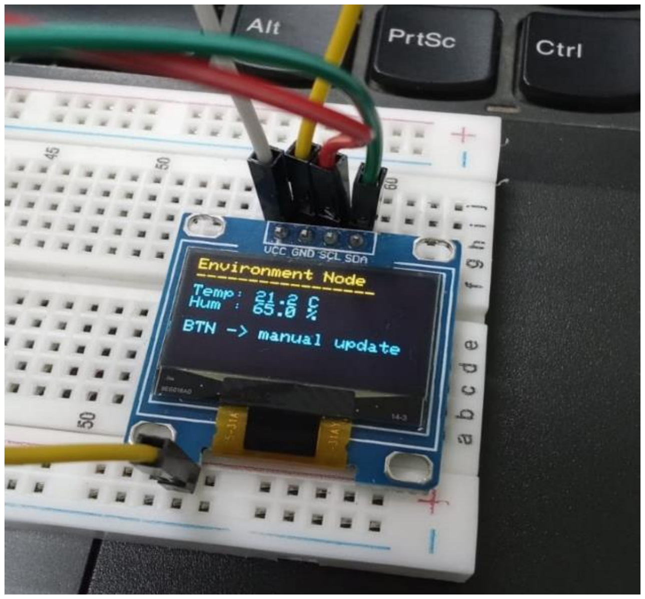
The ESP32–Blynk Cloud temperature and humidity monitoring system integrates sensor data acquisition, cloud communication, and local display efficiently. By using **Blynk Templates**, **Virtual Pins**, and **BlynkTimer**, the system ensures secure, real-time, and reliable data visualization. Proper sensor configuration, correct authentication, and non-blocking programming practices make the system robust and suitable for real-world IoT applications.

Screenshots

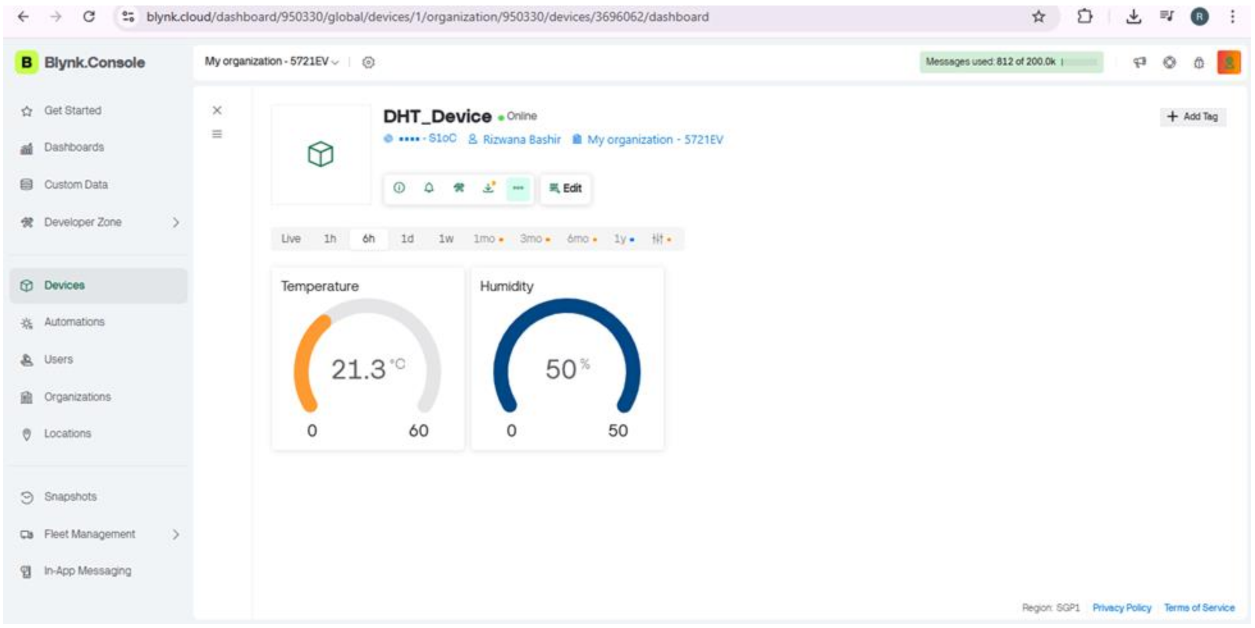
Hardware:



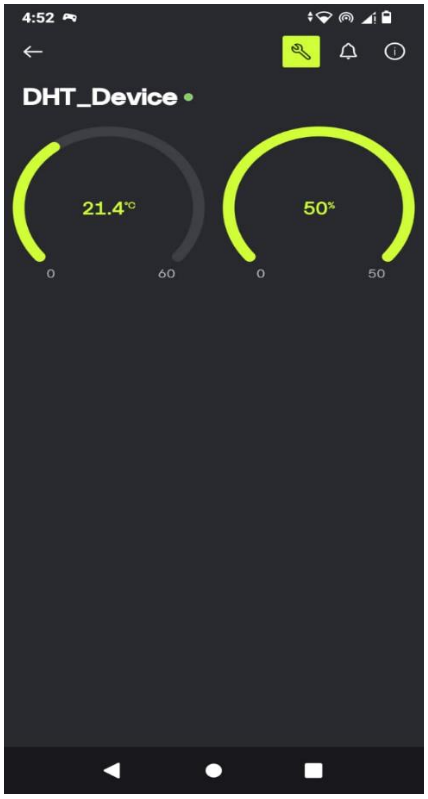
Sensor reading on OLED screen:



Blynk Cloud Web:



Mobile Blynk App Data:


4:52
DHT_Device
21.4°C
0
60
50%
0
50