National Textile University, Faisalabad

Department of Computer Science

Name:	**Ayesha Iftikhar**
Class:	CS 5th (A)
Registration No:	23-NTU-CS-1021
Homework:	1
Course Name:	IOT
Submitted To:	*Sir Nasir*
Submission Date:	20-12-2025

Question-1

ESP32 Webserver (webserver.cpp)

Part A:
Short Questions

Question 1

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

WebServer server(80); create webserver on esp32 . The work of webserver is to receive requests from browser. Port 80 is the default http port for web communication.

Question 2

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

server.on("/", handleRoot); handles requests to the root URL ("/") by calling the handleRoot function.

Question 3

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

The esp32 continously check if any browser request has arrived or not.Thats why it is placed inside the loop() function. If it is removed the webserver will not respond to the browser requests.

Question 4

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

200 -> HTTP requests success
"text/html" -> tells the browser that the data is in HTML
Html -> actual webpage content
Simply means it send the html content to the browser along with http 200 response

Question 5

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

Sensor value reads when button is pressed . These values store in variable . The webpage shows these values .

Fresh reading inside handleRoot()
Sensor values read everytime when webpage refresh. This gives real time data.

Part B
Long Question

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

1. ESP32 Wi-Fi connection and IP address assignment

The Wi-Fi name (SSID) and password are written in the ESP32 code.

The ESP32 connects to the router using `WiFi.begin()`.

When the connection is successful, the router automatically gives an **IP address** to the ESP32 using DHCP.

This IP address is displayed on the **Serial Monitor**.

The user enters this IP address in a web browser to open the ESP32 web page.

2. Web server initialization and request handling

A **WebServer / WiFiServer** object is created in the ESP32 code.

The server is started on a port, usually **port 80**.

The server keeps listening for requests coming from the web browser.

When the browser sends a request, the ESP32 receives and processes it.

The ESP32 then sends an **HTML webpage** as a response back to the browser.

3.Button-based sensor reading and OLED update mechanism

A push button is connected to a GPIO pin of the ESP32.

When the button is pressed:

The ESP32 reads temperature and humidity from the DHT sensor.

The new values are saved in variables.

These values are shown on the OLED display.

This method avoids reading the sensor again and again and keeps the system controlled.

4 .Dynamic HTML webpage generation

The ESP32 creates the HTML webpage while the program is running.

The current temperature and humidity values are added into the HTML code.

For example:

1. Temperature: 28°C
2. Humidity: 65%

When the webpage loads, the browser shows the latest sensor readings.

This is called a dynamic webpage because the values keep changing.

5. Purpose of meta refresh in the webpage

The tag is used in the HTML page.

It automatically refreshes the webpage after a fixed time.

The purpose is:

The user does not need to refresh the page manually.

Updated temperature and humidity values are shown automatically.

6. Common issues in ESP32 webserver projects and their solutions

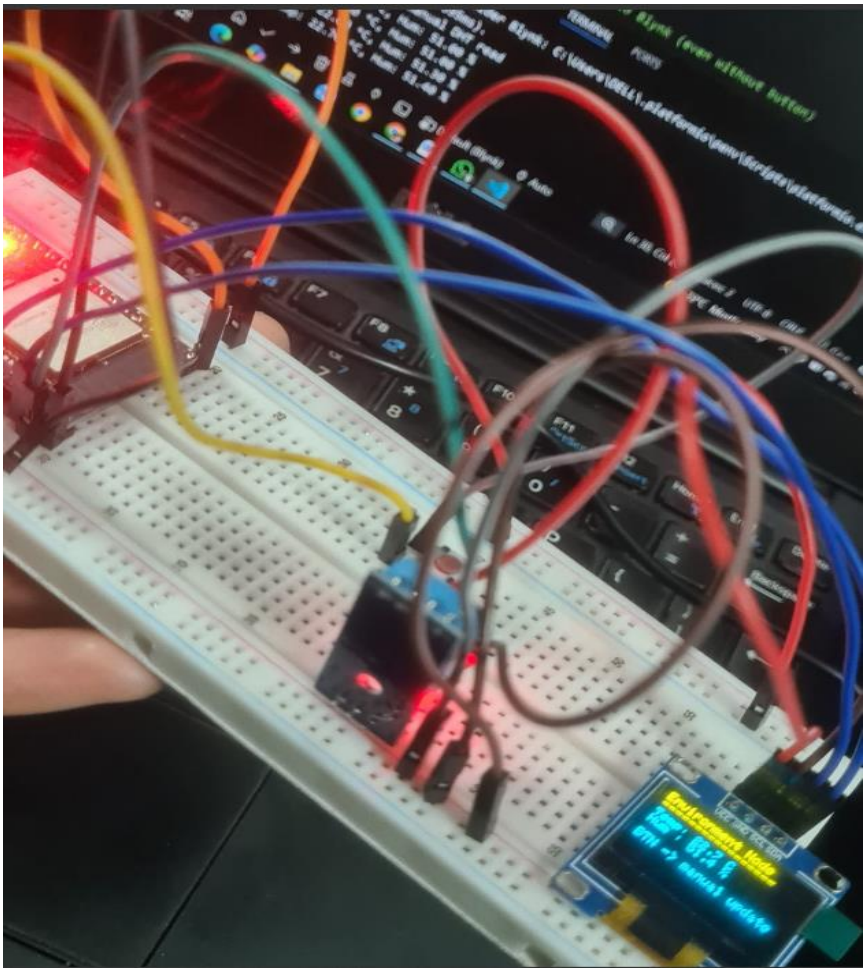
Wi-Fi does not connect

- Reason: Wrong Wi-Fi name (SSID) or password
- Solution: Double-check the credentials and make sure the router is within range

Sensor gives incorrect values

- Reason: Wrong wiring or missing delay between readings
- Solution: Check the wiring and add a proper delay in the code

Screenshots:



Question-2

Blynk Cloud Interfacing (blynk.cpp)

Part-A:

Short Questions

Question 1

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

The BlynkTemplate links the ESP32 project with a specific template on the Blynk Cloud. It must match the cloud template so the device can correctly connect and use the widgets, data streams, and settings defined in that template.

Question 2

Difference between Blynk Template ID and Blynk Auth Token

Blynk Template ID

Identifies the project template on Blynk Cloud

Same for all devices using that template

Blynk Auth Token

Unique security key for each device

Used to authenticate and allow the device to connect to the Blynk Cloud

Question 3

Why DHT22 code gives wrong readings with a DHT11 sensor

DHT11 and DHT22 use different data formate and ranges
Using DHT22 code with a DHT11 causes incorrect values because the ESP32 interprets the sensor data wrongly.

Key difference:

DHT11 has lower accuracy and a smaller temperature range than DHT22.

Question 4

What are Virtual Pins in Blynk and why they are preferred

Virtual pins are software-based pins used to send and receive data between the ESP32 and the Blynk Cloud.

They are preferred because:

- They are not limited by physical GPIO pins
- They make cloud communication easy and flexible
- Multiple widgets can work without hardware pin conflicts

Question 5

Purpose of using BlynkTimer instead of delay()

Blynktimer allows tasks to run at specific intervals without blocking the program

Using delay() stops the ESP32 from handling:

- Cloud communication
- Sensor updates
- Button presses

So, BlynkTimer keeps the ESP32 responsive and the Blynk connection stable.

Part-B:

Long Question

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

- Creation of Blynk Template and Datastreams

First, a Blynktemplate is created on the Blynk Cloud.

Inside the template, Datastreams are created:

- One datastream for temperature

- One datastream for humidity

Each datastream is linked to a **Virtual Pin** (for example, V0 for temperature and V1 for humidity).
Widgets such as labels or gauges are then connected to these datastreams.

- Role of Template ID, Template Name, and Auth Token

- **Template ID** connects the ESP32 code to the correct Blynk Cloud template.
- **Template Name** is used for identification and display purposes in the Blynk system.
- **Auth Token** is a unique security key that allows the ESP32 to authenticate and communicate with the Blynk Cloud.

All three values must be correctly written in the ESP32 code, otherwise the device will not connect to Blynk.

Sensor configuration issues (DHT11 vs DHT22)

The ESP32 uses a DHT sensor to measure temperature and humidity.
DHT11 and DHT22 are different sensors with different accuracy and data formats.

If DHT22 code is used with a DHT11 sensor:

- The readings become incorrect
- Temperature and humidity values may appear unrealistic

- Sending data using Blynk.virtualWrite()

After reading temperature and humidity values from the sensor, the ESP32 sends the data to Blynk Cloud using:

```
Blynk.virtualWrite(Vx , value)
```

Here

vx is the virtual pin linked to a datastream

`value` is the sensor reading

The Blynk Cloud receives this data and updates the widgets on the dashboard in real time.

- Common problems faced during configuration and their solutions

ESP32 does not connect to Blynk Cloud

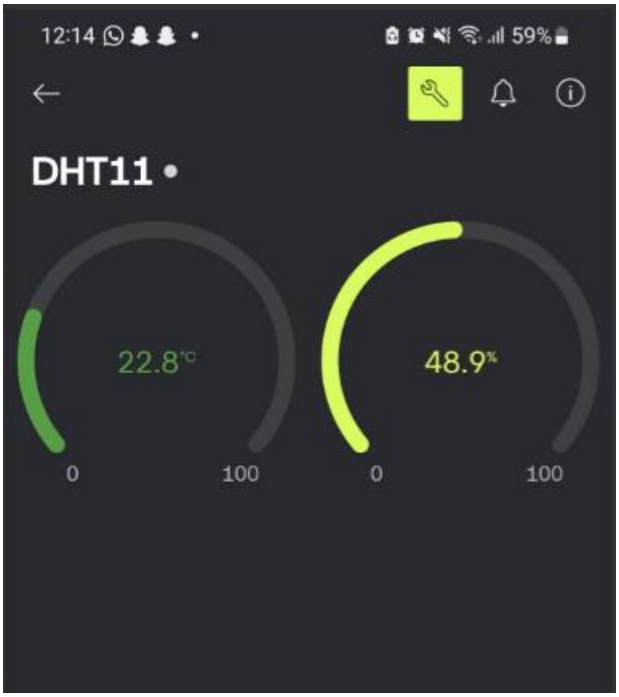
- Cause: Wrong Template ID, Auth Token, or Wi-Fi credentials

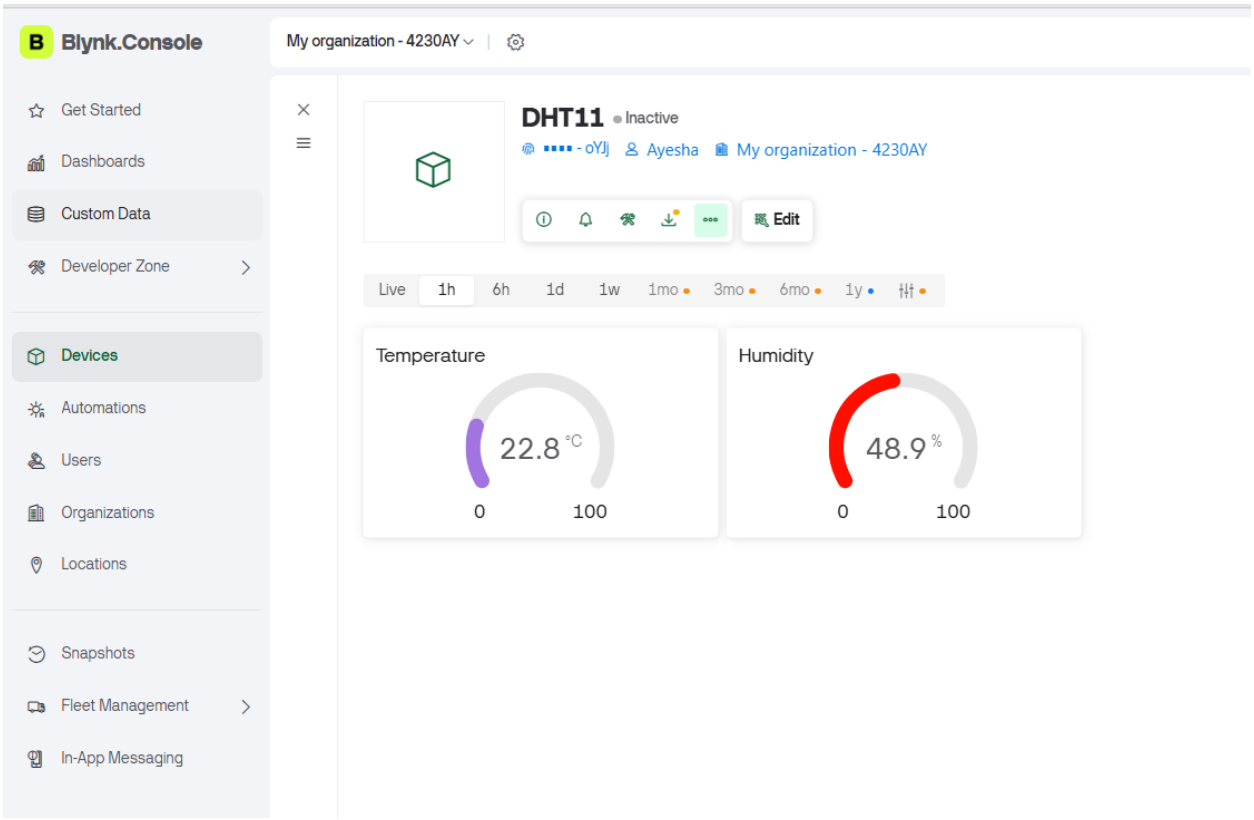
- Solution: Recheck all credentials and ensure internet connection

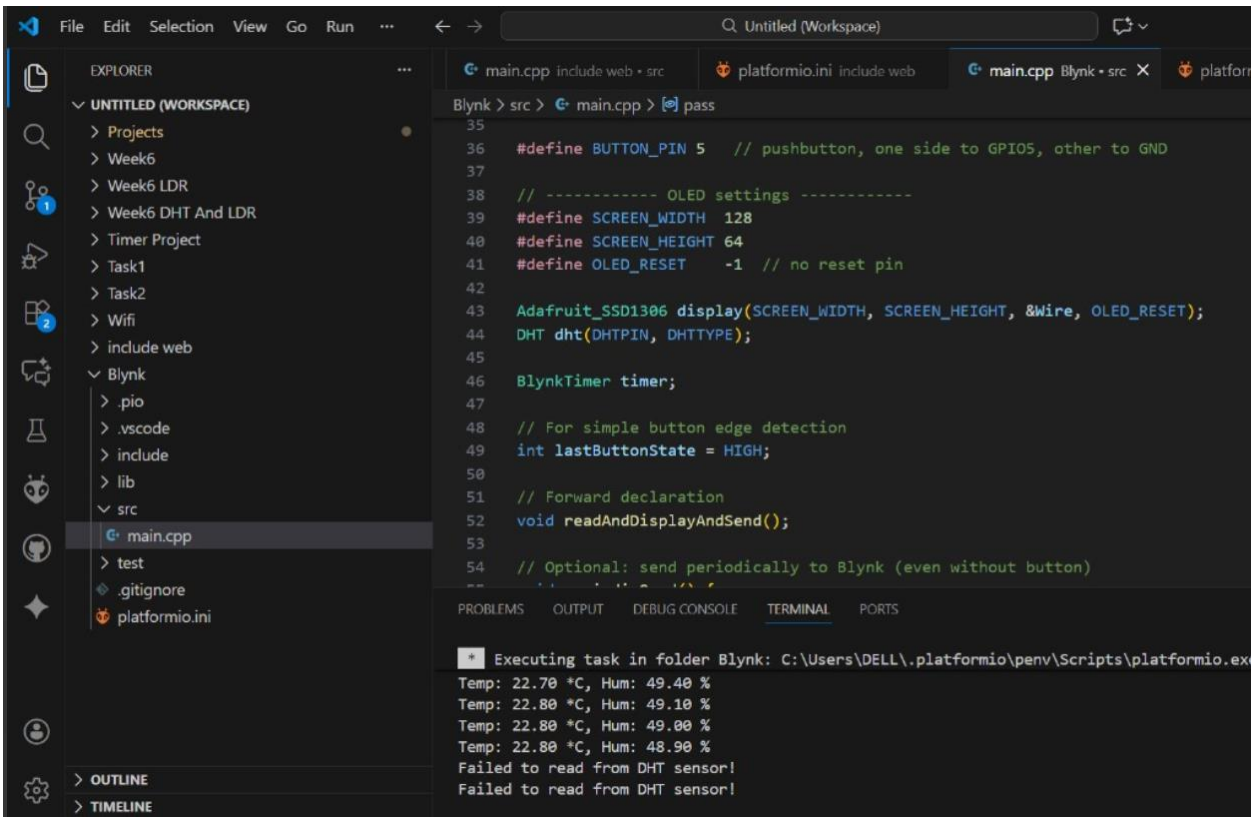
Data not updating on Blynk dashboard

- Cause: Wrong virtual pin or datastream not created
- Solution: Match virtual pins in code with Blynk datastreams

Screenshots:




Blynk.Console
My organization - 4230AY
Get Started
Dashboards
Custom Data
Developer Zone
Devices
Automations
Users
Organizations
Locations
Snapshots
Fleet Management
In-App Messaging
DHT11
Inactive
Ayesha
My organization - 4230AY
Edit
Live
1h
6h
1d
1w
1mo
3mo
6mo
1y
Temperature
22.8 °C
0
100
Humidity
48.9 %
0
100


File Edit Selection View Go Run
Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
Projects
Week6
Week6 LDR
Week6 DHT And LDR
Timer Project
Task1
Task2
Wifi
include web
Blynk
.pio
.vscode
include
lib
src
main.cpp
test
.gitignore
platformio.ini
OUTLINE
TIMELINE
main.cpp include web • src
platformio.ini include web
main.cpp Blynk • src
Blynk > src > main.cpp > pass
#define BUTTON_PIN 5 // pushbutton, one side to GPIO5, other to GND
// ------------ OLED settings ------------
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1 // no reset pin
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
DHT dht(DHTPIN, DHTTYPE);
BlynkTimer timer;
// For simple button edge detection
int lastButtonState = HIGH;
// Forward declaration
void readAndDisplayAndSend();
// Optional: send periodically to Blynk (even without button)
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Executing task in folder Blynk: C:\Users\DELL\.platformio\penv\Scripts\platformio.ex
Temp: 22.70 *C, Hum: 49.40 %
Temp: 22.80 *C, Hum: 49.10 %
Temp: 22.80 *C, Hum: 49.00 %
Temp: 22.80 *C, Hum: 48.90 %
Failed to read from DHT sensor!
Failed to read from DHT sensor!