National Textile University, Faisalabad

Department of Computer Science

Name:	Meerab Zahra
Class:	BSCS 5th A
Registration No:	23-NTU-CS-1049
Assignment:	2
Course Name:	Embedded IoT Systems
Submitted To:	Sir Nasir Mahmood
Submission Date:	December 17, 2025

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?

WebServer server(80) is a command used in programming libraries (like in C++ for platforms like Arduino or ESP microcontrollers) to initialize a web server instance. The purpose of this command is to create an object named server of the WebServer class. The (80) argument passed to the constructor tells this server object to listen for incoming network requests on Port 80.

Port 80 is the default port used for HTTP websites. When we open the ESP32 IP address in a browser, the browser communicates with the ESP32 using this port.

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

This line tells the ESP32 what to do when someone opens the main page (/) of the web server. When the user enters the ESP32 IP address in the browser, the function handleRoot() is called and the web page is shown. This function contains all the code responsible for layout of webpage components.

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

server.handleClient() checks if there is any user requesting access to the web page.
It is placed inside loop() so the ESP32 can continuously listen for requests as the loop runs repeatedly but if it is removed the server will stop processing data sent by clients that are already connected, there will be no new connections and the server becomes unresponsive.

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

This line sends the web page to the browser.

- **200** means the request was successful. This is the HTTP status code for **"OK"**.
- **text/html** is the Content-Type header which tells the browser that the content is Hyper Text Markup Language (HTML) and should be rendered as a web page.
- **html** is a variable containing a string of HTML code that the browser will display to the user and it contains the actual webpage code.

5. What is the difference between displaying last measured sensor values and taking a

fresh DHT reading inside handleRoot()?

Displaying last measured value shows the data already stored in memory. Taking a fresh reading means the DHT sensor is read again every time the page is opened. The actual difference lies in performance, reliability, and responsiveness of web server. Reading the sensor inside handleRoot() is slow and can cause the web server to crash or become unresponsive, whereas using a last value is efficient and fast.

Part B: Long Question

Working of ESP32 Webserver-Based Temperature and Humidity Monitoring System

This system requires an ESP32, DHT22 sensor, OLED display, push button, and a web server. ESP32 is the microcontroller that is responsible for processing of circuit, DHT22 sensor for reading temperature and humidity, OLED display for showing current temperature and humidity values, push button whenever pressed gives latest values for temperature and humidity.

- First, the ESP32 in station mode connects to Wi-Fi using the SSID and password through function **WiFi.begin(ssid, password)**. Once connected, it gets an IP address from the router. This IP address is shown on the OLED screen so the user can open it in a web browser.
- An object named server of WebServer class is created. After Wi-Fi connection, this web server starts on port 80. The ESP32 listens to incoming web requests through this port. The web server route is defined using **server.on("/", handleRoot)**. This line connects the root URL of the web server with the handleRoot() function.
- When a user opens the ESP32 IP address in browser, the **handleRoot()** function runs to generate a simple HTML webpage. It shows temperature and humidity values using the last stored readings. If no valid data is available, it displays a message asking the user to press the button.
- The webpage also includes a **meta refresh tag** which automatically refreshes the page after every 5 seconds so the user can see updated values (html += "<meta http-equiv='refresh' content='5'>";).
- A **push button** is connected to the ESP32 using INPUT_PULLUP mode which doesn't require any external resistor. In the loop() function, the button state is monitored repeatedly. When the button is pressed, the ESP32 reads fresh temperature and humidity values from the DHT22 sensor using **the readDHTValues()** function. These values are stored in variables and they are also printed on the Serial Monitor and OLED.
- The function **server.handleClient()** is continuously executed inside the loop() function. This allows ESP32 to handle new client requests without stopping the program. If this

function is not used, the web server will not respond to browser requests and will become unresponsive.

- Some **common issues** in ESP32 webserver projects include Wi-Fi connection delay, invalid DHT sensor readings, and webpage not updating. These issues can be solved by checking the Wi-Fi credentials, double checking the sensor pins, adding small delays, and by placing the server handling code inside the loop.

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?

Blynk Template ID is a unique identifier to recognize which project template the ESP32 is connected to on Blynk Cloud. It must match the cloud template credentials so that the ESP32 can correctly send and receive data from the correct dashboard. If it does not match, the device will fail to connect or data will not be displayed properly.

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

Blynk Template ID is a unique identifier to recognize which project template the ESP32 is connected to on Blynk Cloud while Blynk Auth Token is a unique key assigned to a device to authenticate it and connect it with Blynk Cloud. Template ID is same for a project but each device has a different Blynk Authentication Token.

3. Why does using DHT22 code with a DHT11 sensor produce incorrect readings?

Mention one key difference between the two sensors.

DHT22 and DHT11 have different ranges for measuring temperature and humidity. **DHT22** can measure temperature from -40 °C to 80 °C (with ±0.5°C accuracy) and humidity from 0% to 100% RH (with ±2% RH accuracy) while **DHT11** can measure temperature from 0 °C to 50 °C (with 2°C accuracy) and humidity from 20% to 90% RH (with ±5% RH accuracy)

Reason for incorrect readings
Using DHT22 code with DHT11 causes wrong values because the sensor type does not match and the code is trying to read data that isn't there. DHT11 only provides whole numbers (like 25 degrees), while DHT22 provides floating numbers (like 25.5 degrees). The code for DHT22 expects those extra decimal bits and when it doesn't find them in the DHT11's data stream, it provides incorrect readings.

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

Virtual Pins are software-based pins used to send and receive data between ESP32 and Blynk Cloud. These logical channels are used to exchange any type of data between hardware, the Blynk cloud server, and the mobile/web app. They have no physical representation on the microcontroller They are preferred because they are flexible and not tied to actual hardware pins. This makes cloud communication easier, and more organized.

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

BlynkTimer allows tasks to run at fixed intervals without stopping the program as it has non-blocking execution. Using delay() halts the program execution for a specified time and stops Wi-Fi and Blynk communication. We don't even use ESP32 timers because they run ISRs and it is not safe to call Wi-Fi, Blynk, Serial or OLED functions inside ISRs. So, the BlynkTimer keeps the system responsive and stable.

Part-B: Long Question

Workflow of Interfacing ESP32 with Blynk Cloud

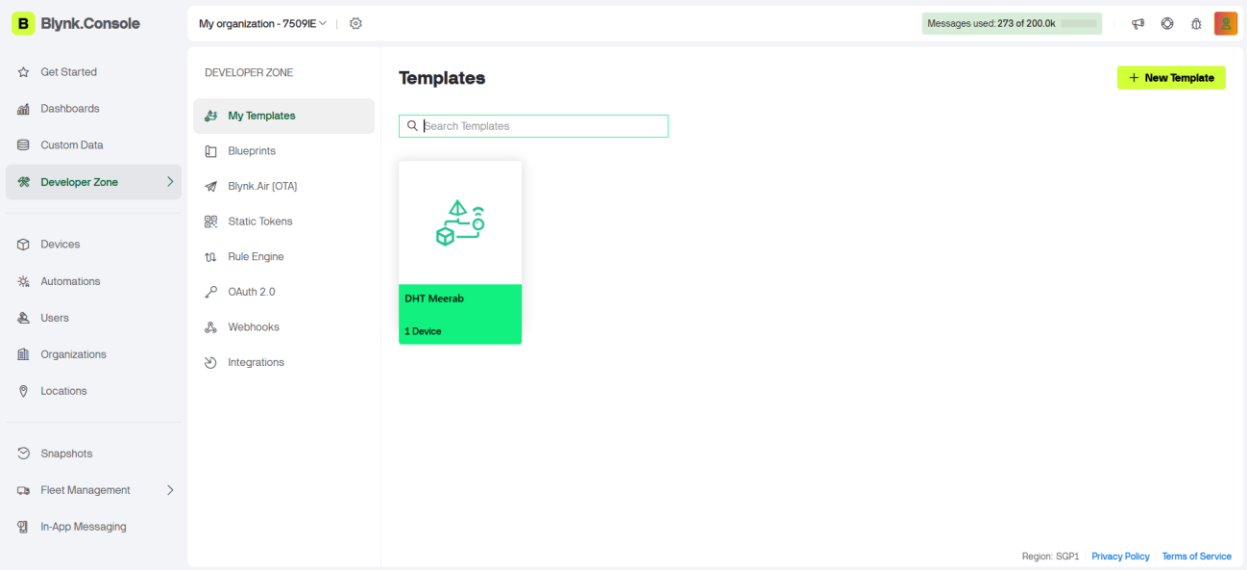
This project uses ESP32, DHT22 sensor, OLED display, push button, and Blynk Cloud to display temperature and humidity.

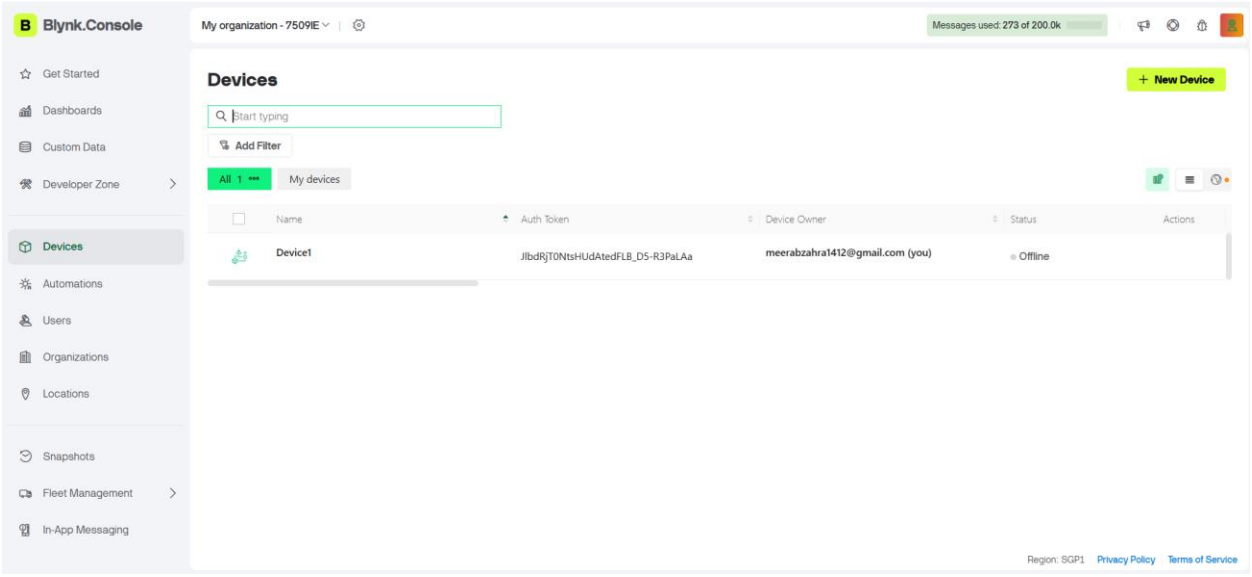
- First, a **template** is created on Blynk Cloud. This template has **datastreams** for temperature and humidity. In the code, V0 is used for temperature and V1 is used for humidity. Datastreams allow sending data from ESP32 to the cloud in an organized and representable way.
- After creating a template we get different credentials like BLYNK_TEMPLATE_ID, BLYNK_TEMPLATE_NAME, BLYNK_AUTH_TOKEN. Blynk Template ID is a unique identifier to recognize which project template the ESP32 is connected to on Blynk Cloud while Blynk Auth Token is a unique key assigned to a device to authenticate it and connect it with Blynk Cloud. Blynk Template name is just the name of the template (like "dht" in given code).

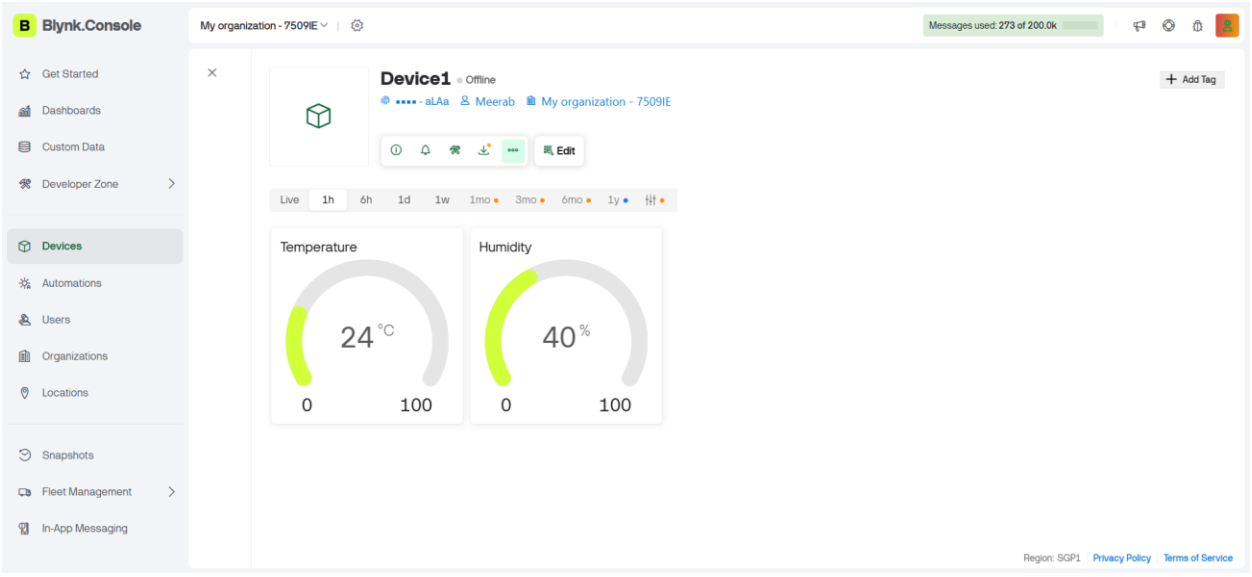
- DHT22 is used in this project. If you use DHT11 in circuit as a component with DHT22 code, readings will be wrong. DHT11 only provides whole numbers (like 25 degrees), while DHT22 provides floating numbers (like 25.5 degrees). The code for DHT22 expects those extra decimal bits and when it doesn't find them in the DHT11's data stream, it provides incorrect readings. Always make sure the code matches the sensor type.
- The function readAndDisplayAndSend() reads the DHT22 sensor. It updates the OLED display with the latest temperature and humidity values. Then it sends these values to Blynk Cloud using Blynk.virtualWrite(V0, t) for temperature and Blynk.virtualWrite(V1, h) for humidity. A push button is also used for manually getting new readings. Pressing the button calls the same function. Additionally, BlynkTimer is used to automate data sending every 5 seconds without blocking the program.
- Some common problems in ESP32 Blynk projects include the device not connecting to Blynk Cloud, invalid sensor readings or the Blynk dashboard not updating according to readings. These issues can be solved by rechecking Wi-Fi credentials, making sure the Template ID and Auth Token match the cloud settings, ensuring the DHT sensor type in the code matches the actual sensor and keeping Blynk.run() and timer.run() inside the loop.

Web Dashboard

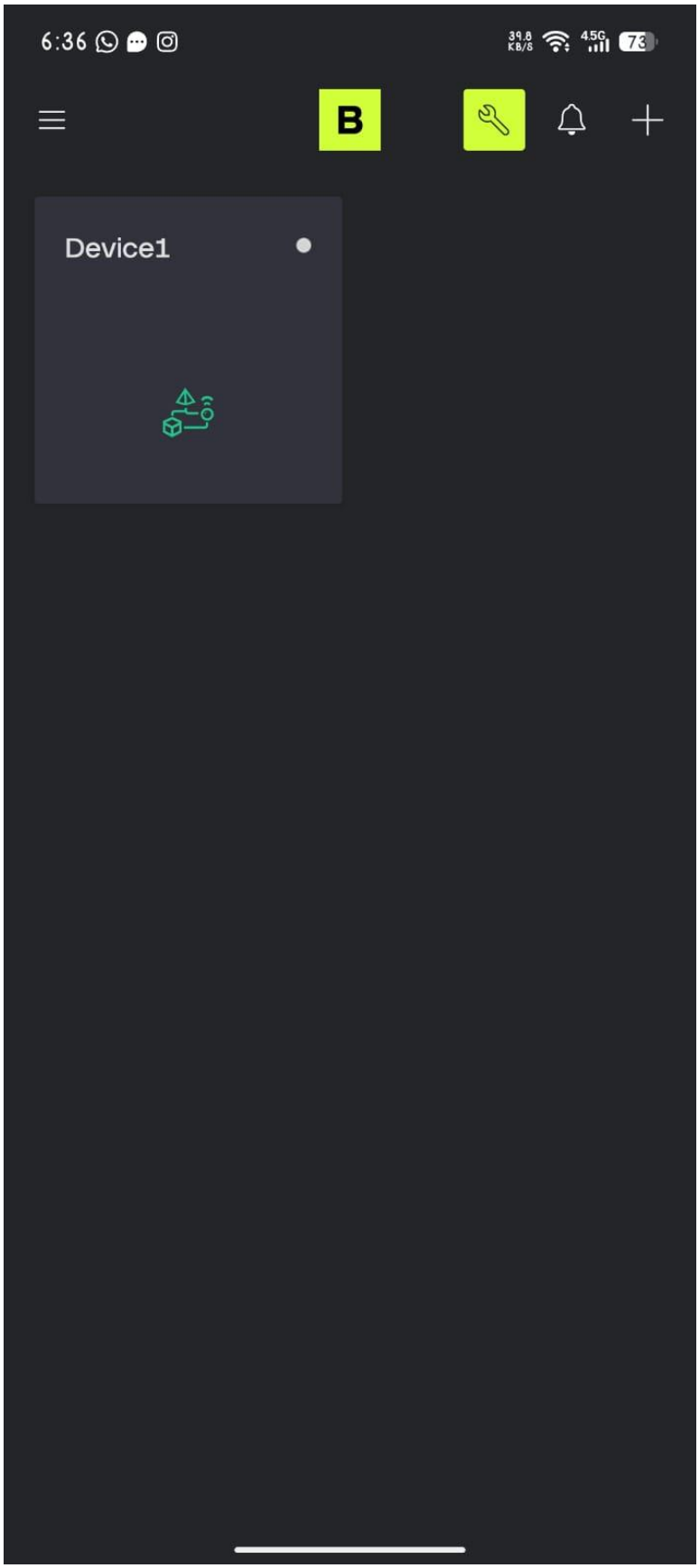
After logging in with email, go to developer zone to create a new template with required datastreams, then add a device to that template. Adding a device will give credentials like Template ID, Template Name and Auth Token to be used in code that will be pushed to ESP32.

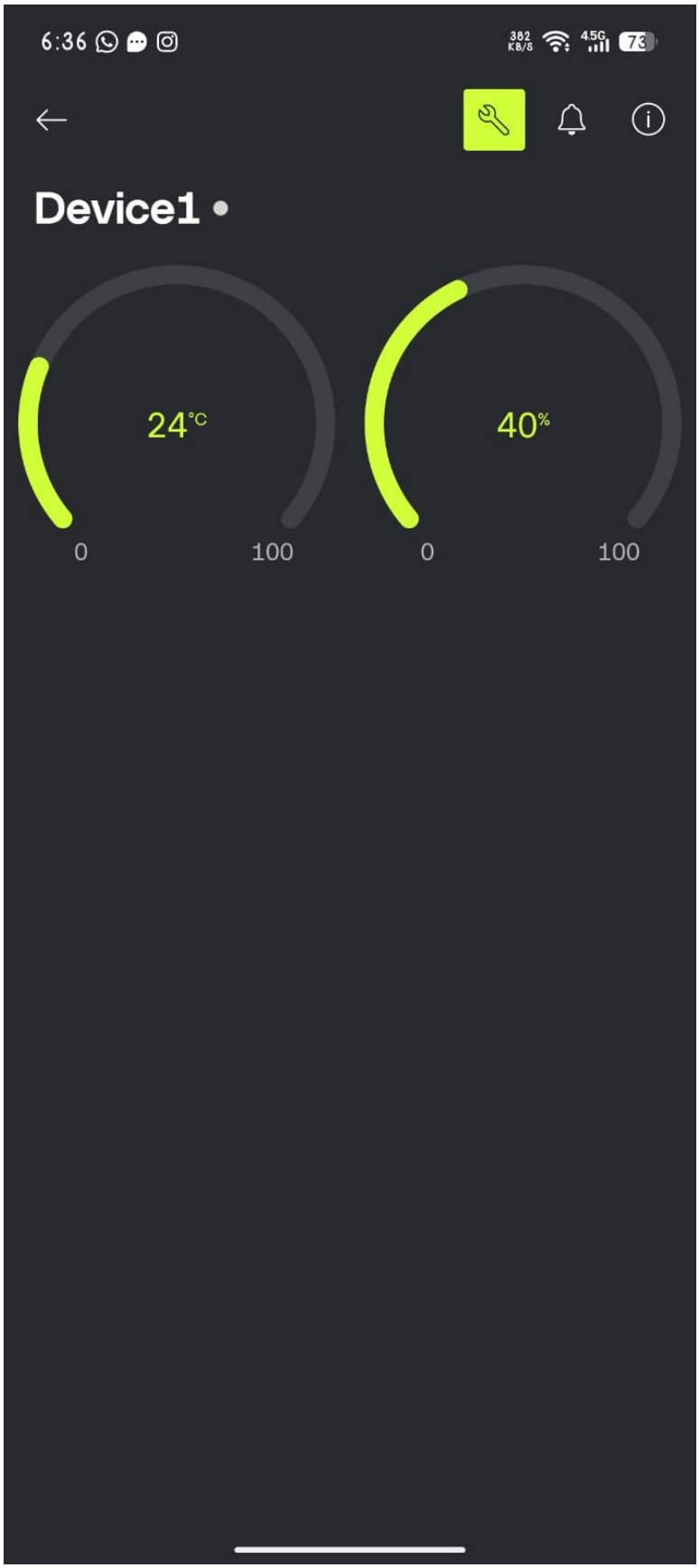






Mobile Dashboard


6:36
B
Device1


6:36
382 KB/s
4.5G
73
Device1
24°C
0
100
40%
0
100