National Textile University, Faisalabad

Department of Computer Science

Name:	Muhammad Umer
Class:	BSCS (B)
Registration No:	23-NTU-CS-1078
Lab Report:	Assignment 2
Course Code:	CSE-3080
Course Name:	Embedded and IOT Systems
Submitted To:	Sir Nasir
Submission Date:	17/12/2025

Question no 1:

Part A – Short Questions

1) One line of code used in ESP32 webserver code is webserver server(80). This line's function is to create an HTTP request-listening web server object. and enables gadgets like the ESP32 to function as websites in web browsers like Firefox and Chrome.
The HTTPS port number is 80. HTTPS requests are listened to on this port.

2) What a web server should do when a particular URL is requested is specified by server.on(\"/handleroot). handleroot is a function that instructs the server on what response to send, and "-" is the website's root page. In the absence of this, the server is unsure of what to display on the browser.

3) To continuously check and process the data coming from HTTP requests, server.handleClient() is put inside the loop. It sustains the server.

the server.handleClient() determines whether a client is attempting to establish a connection. Additionally, if a client is connected, it looks for HTTPS requests from the client, matches the requested URL, calls the appropriate function, like handleroot, and returns the response to the client.

4) The handleroot() function uses it to transmit the ESP32's HTTP response to the web server. There are three parameters for it. The first is the status code, 200, which indicates that the request was fulfilled. The second is content type, which indicates the type of data to be sent; in this case, it is an HTML web page. The content we are sending is the response body, which comes in third. This example uses an HTML string variable to represent HTML code.

5) The DHT sensor is read outside of handleroot() and stored in the temp and hum variables, which are displayed on the webpage when the last measured sensor value is displayed.
Every time a button is pressed or the webpage is refreshed, the sensor takes a new reading.

Part B – Long Question

- There are two methods for connecting an ESP32: dynamic IP and static IP. 1) In order to connect dynamically, you must first provide your wifi's SSID and password. Wifi.begin(ssid, password) in the setup() function initiates the connection, and Wifi.status() in the while loop determines whether or not WiFi is connected. You can then use Wifi.localIP to see your assigned IP address if the wifi is connected. 2) In order to establish a static connection, you must configure your local IP, gateway, subnet, and DNS. Here, you set a static IP using the Wifi.config() function. The process is then the same as dynamic.

- To get a web server going, you start by setting up this WifiServer server on port 80, that initializes the whole protocol thing. It seems like thats the first step anyway.
  Once you're connected to the WiFi, in the main loop you check with server.available, you know, to see if some clients trying to link up. If there is one, the connection just happens, I guess.
  Handling the actual requests gets a bit tricky. You call server.handleClient to look for those HTTP things coming from the client and deal with them. For the root page stuff, its the handleRoot function that takes care of it, and then server.send lets you specify what kind of data youre pushing out, plus whether it worked or not. That part feels kind of important, but Im not totally sure on the details yet.

- To get the sensor readings from a button press, you need to add a button first. Then define its pin and set the pinmode to INPUT_PULLUP, that seems like the usual way. In the loop part, it checks the buttons current state, handles some debounce stuff, adds a little delay. If the button press is valid, it runs the readDHTValues function and the showOnOLED one to put the readings up.
  The showOnOLED function has this update thing for the OLED. It clears the display first with display.cleardisplay. Then, if theres an error, it shows that, otherwise it puts up the temperature and humidity readings. I might be missing a bit on how the debounce works exactly, but it keeps things from triggering too much. That part is kind of important for not getting false reads.

- The handleroot() function has the the HTML code to generate a dynamic webpage. First of all, in the HTML header we define the page and sets encoding and ensures mobile-friendly display and sets to refresh the page every 5 seconds. And to geneate dynamic content we set isnan() that inserts the latest values into the HTML and the HTML footer provides instructions for the user to press the physical button snd the in the end server.send() is used to tell the type of the content we are sending.

- The Meta refresh tells the browser to automatically reload the page according to the time we have set.

- There are several issues. 1) First one is that ESP32 fails to connect to Wifi it shows

“Connecting to Wifi….” Forever and no IP is assigned. Its solution is that check whether your ssid or password is correct and check your internet connection especially if you are using “NTU FSD”. 2) The second one is webpage not loading it is because the server cannot each ESP32 and it browser says “Server not found” to solve this ensure that server.handleClient() is inside the loop and type the correct assigned IP on the browser. 3) Another one is the DHT sensor fails to read values and shoes error on the OLED. To solve this ensure that the wires are connected correctly and ensure the DHT type matches the sensor.

Question no 2:

Part A – Short Questions

1) A device made on the Blynk cloud is uniquely identified by its Blynk Template ID. It serves as a conduit between the cloud dashboard and the ESP32. The ESP32 won't connect, widgets won't function, and the device will appear offline if the template ID doesn't match.

2) While the Blynk Auth Token authenticates a particular device, the Blynk template ID connects ESP32 to the appropriate dashboard on the Blynk Cloud. While the Auth Token is specific to each device, the Template ID is shared by all devices.

3) Because the two sensors use different data formats, timing protocols, and measurements, the results are inaccurate. A mismatch results in an incorrect interpretation of the signals because the DHT library decodes sensor data according to the specified sensor type.

4) Instead of using real hardware GPIO pins, Blynk uses software-defined virtual pins to transfer data between the ESP32 and Blynk Cloud. Because they enable internet-based communication, are hardware independent, and can transport any kind of data, they are favoured over physical pins.

5) Blynk Timer is used in place of delay() because it adds no overhead by carrying out operations without interfering with program execution. This keeps Wi-Fi connected and guarantees constant communication.

Part B – Long Question

- In order to create a Blynk template, we click the "New Template" button, give the template a name, specify the hardware (in our case, ESP32), and specify the connectivity type (in our case, Wi-Fi). The auth token is then displayed in a popup, along with the template ID and name, which we save. Next, we create a datastream of the virtual pin type using the Datastream button, give it a name, virtual pin number, and data type, and set the minimum and maximum values.

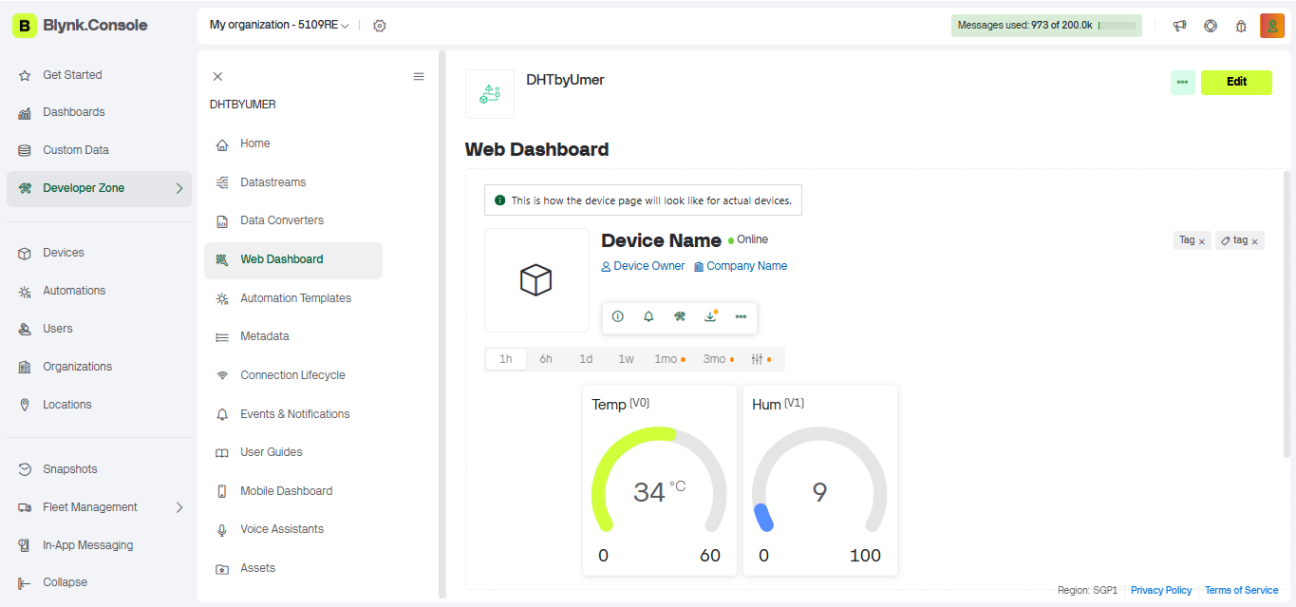
- The Blynk template ID, it basically connects the ESP32 to the right dashboard up on the Blynk Cloud.While the Auth Token handles authenticating just one specific device.I think the Template ID stays the same for every device in the setup, but the Auth Token, that is unique to each one.Kind of makes sense for security or something.Then there is the Template Name, which is just a label for the whole project to help identify it.Not sure if that is all, but yeah.

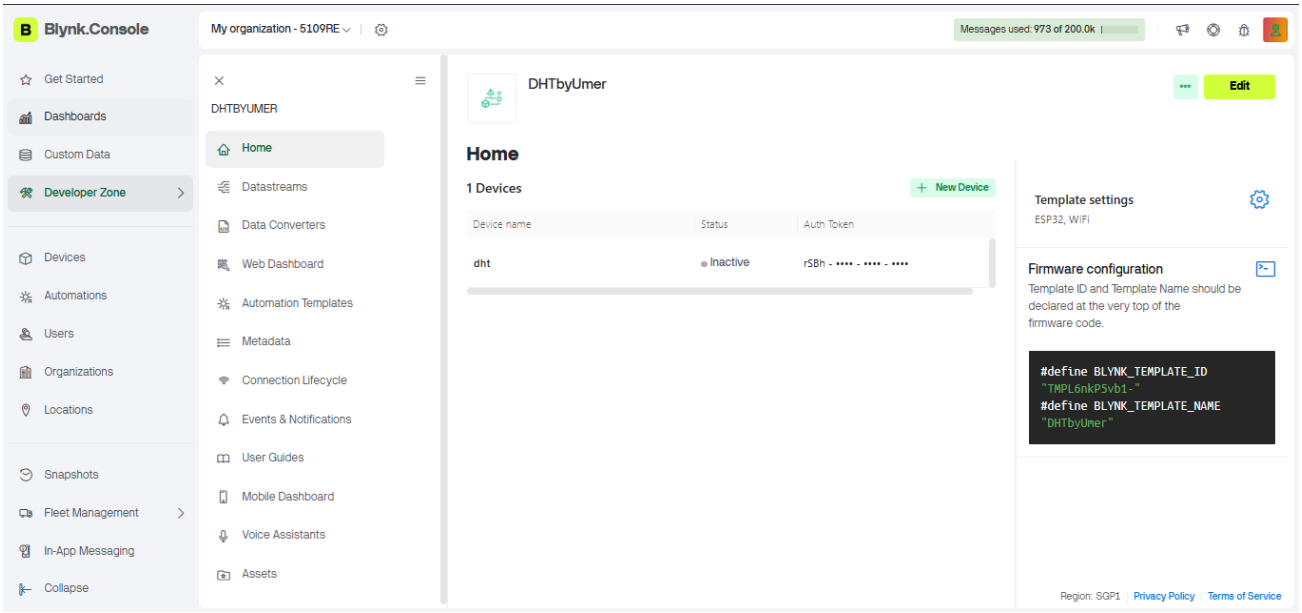
- If you choose the wrong DHT type, it messes up the readings pretty bad. The two sensors have different ways of handling data, like their formats and timing, and even the measurements they take. I think that is why it goes wrong so easily. The DHT library is supposed to decode everything based on what type you tell it. So when there is a mismatch, it just interprets the signals all incorrect. Using the wrong sensor type in your code, that causes the readings to be off. It is kind of obvious once you see it happen, but yeah, it does lead to those incorrect results.

- The function `Blynk.virtualWrite()` used to send the data from the ESP32 to the Blynk Cloud. It updates widgets linked to virtual pins and allows real-time monitoring over the Internet.

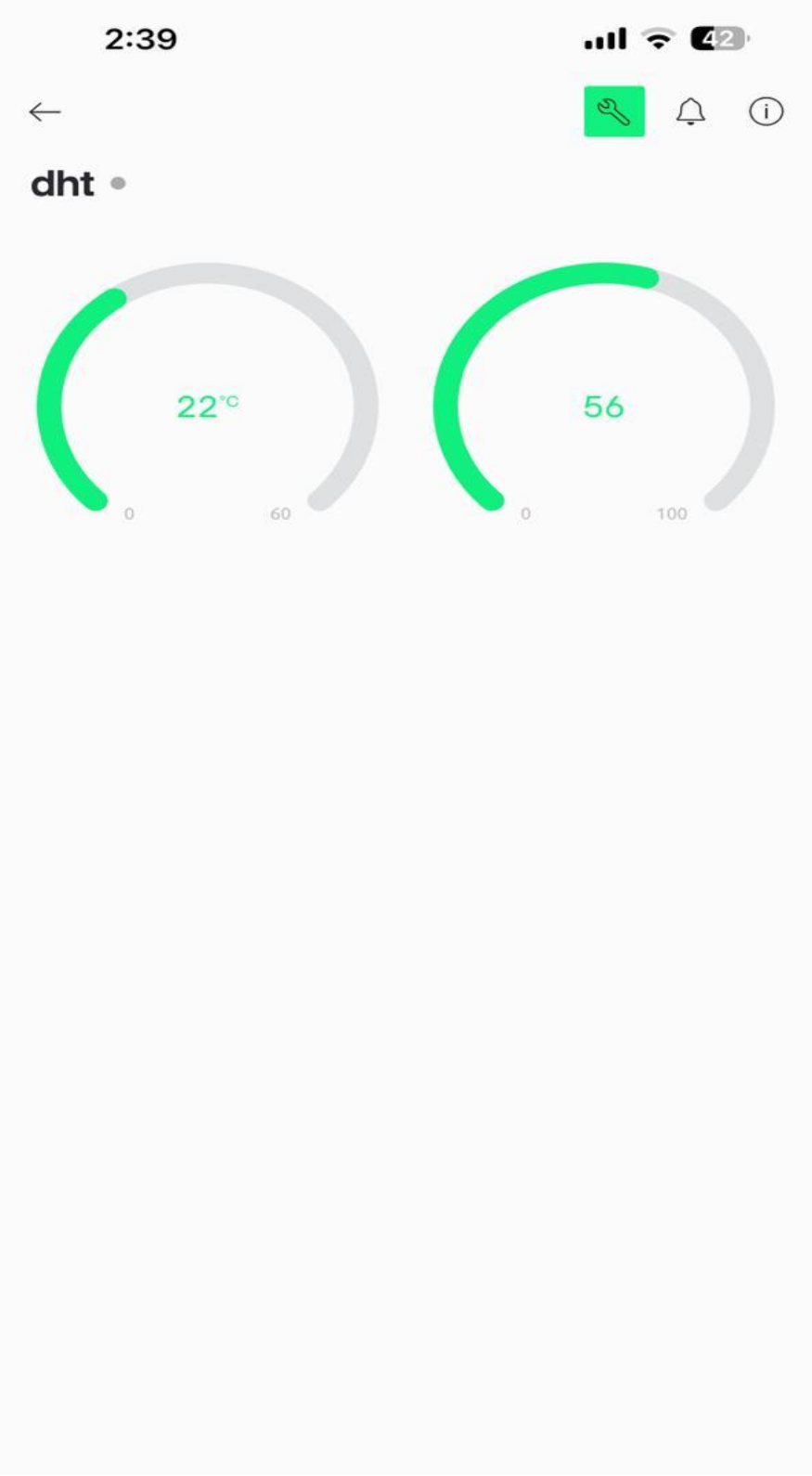
- The most commonplace situations that arise include: 1) Device showing offline. The solution for that would be to verify the correct Auth token from the Blynk Cloud and to double-check the Wi-Fi ssid and password. 2) Widgets not updating. This could easily be solved by matching the exact Virtual Pins with that of the data type. 3) The other one is wrong temperature and humidity readings, which could be solved by using the correct DHT type and sensor and also adding an interval to read or update the next value.

Blynk Cloud (Web):





Blynk Cloud (App):



Github Link:

https://github.com/theholybeing/Embedded-IoT-Systems/