

National Textile University, Faisalabad



Name:	Aqdas Fatima
Class:	CS 5 th B
Registration No:	23-NTU-CS-1017
Course Name:	Embedded Systems and IOT
Submitted To:	Sir Nasir

Assignment 2

Question1:

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

Answer:

It is used for creating a web server on the device. Allowing microcontroller (ESP 32) to act like website. While port (80) is the standard port for HTTP (pages.).It allows the user the user to open the website without typing any port number.

Question2:

Explain the role of server. On("/", handle Root); in this program.

Answer:

It is used for setting route on web servers. If someone opens the home page(/) it tells the web server to execute the handle root function. It then sends a message/response to the user.

Question3:

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

Answer:

It is placed inside the loop so that the user or server can continuously check the browser request. If we remove the loop() function then web server will not respond to the request. The web server will be connected but it will not load the web page.

Question4:

In handle Root(), explain the statement: server. Send(200, "text/html", html);

Answer:

It is used for sending responses from webserver to browser.

200 → IT is HTTP standard code which indicates that the request is successfully received and proceeded.

text/html → It tells the browser that the incoming data is HTML webpage.

html → It is a variable where the full code of html is stored .For Example, headings, layout of page or text /buttons .

Question5:

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

Answer:

Displaying last measured sensor values:

It uses the old (already read) values. It is faster than others server sends the response quickly. It does not use real time data.

Taking a fresh DHT reading inside handle Root():

It takes new reading from the sensor on each request. It gives the browser real-time ,updated data. It is slower than the other because sensor takes new reading each time.

Part B: Long Question

ESP32 Wi-Fi Connection and Getting an IP

First, the ESP32 connects to your Wi-Fi using the SSID and password. It keeps trying until it successfully connects. Once connected, the router gives it an IP address—think of it as the ESP32's “home address” on your network. You can use this IP to open the ESP32 web page on your phone or computer.

2. Setting Up the Web Server

After connecting to Wi-Fi, we start a web server on port 80 using `Webserver server(80);`. We tell the server what to do when someone opens the home page using `server.on("/", handle Root);`.

The `handleRoot()` function sends the webpage to the browser. In the `loop()`, we keep calling `server.handleClient()`; so the ESP32 always listens and responds to requests.

3. Button to Read Sensor and Update OLED

You can add a button to take a fresh reading from the DHT sensor whenever you want. When the button is pressed, the ESP32 reads the temperature and humidity and updates the OLED display. This way, you can see real-time updates both on the display and on the web page.

4. Dynamic Webpage Creation

When someone opens the ESP32 web page, it creates an HTML page on the fly showing the current temperature and humidity. The readings are added to the HTML in `handleRoot()`. Then the server sends it using `server.send(200, "text/html", html)`; so the browser always shows the latest values.

5. Why We Use Meta Refresh

We can add `<meta http-equiv="refresh" content="5">` in the HTML. This makes the page refresh automatically every 5 seconds, so the values on the web page stay updated without needing to manually reload.

6. Common Problems and How to Fix Them

- Page not loading → Often because `server.handleClient()` is missing. Make sure it's in the `loop()`.
- ESP32 won't connect to Wi-Fi → Check your SSID and password. Use `WiFi.status()` to debug.
- Slow page updates → Reading the sensor every time in `handleRoot()` can slow things down. Using the last reading makes it faster.
- OLED not updating → Check your wiring, pins, and button logic.
- IP keeps changing → Use a static IP if you want to avoid tracking it every time.

Blynk Cloud Interfacing (blynk.cpp)

Part-A: Short Questions

Question1:

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

Answer:

It is used to connect **Blynk Template ID** ESP32 to the specific project template of Blynk Cloud. It tells the ESP32 server which pins or widgets(graph, or buttons) should be use. If ID doesn't match the cloud template then the data will not be displayed in the app so that the device is not connected.

Question2:

Differentiate between Blynk Template ID and Blynk Auth Token.

Answer:

The Blynk Template ID basically tells your ESP32 which Blynk project belongs to. It decides how the app looks, what widgets there are, and which virtual pins the device will use. **The Blynk Auth Token** is different—it's like a secret key for your ESP32 that proves to the Blynk Cloud that this device is allowed to connect. So, many devices can share the same Template ID because they follow the same project layout, but each device needs its own Auth Token to log in safely. Simply put, the Template ID says, "This is the project I'm part of", and the Auth Token says, "Yes, I'm allowed to connect!"

Question3:

Why does using DHT22 code with a DHT11 sensor produce incorrect readings? Mention one key difference between the two sensors.

Answer:

If you use DHT22 code with a DHT11 sensor, the readings will usually be wrong because the two sensors work differently. The DHT22 can measure a bigger range of temperature and humidity, and it sends data in a slightly different way than the DHT11.

Main difference:

- **DHT11:** Temp 0–50 °C, Humidity 20–80 %
- **DHT22:** Temp –40–80 °C, Humidity 0–100 %

So, the code expects DHT22 data, but DHT11 gives its own format, which makes the readings wrong or unreliable.

Question4:

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

Answer:

Virtual Pins in Blynk are like cloud mailboxes for your ESP32. They let your device send and get data without using any physical pins. They're really handy because you don't need wires, they work over the internet, and you can connect lots of widgets easily.

For example: physical pins are like passing a note by hand, while virtual pins are like putting a message in a cloud mailbox that any widget can check anytime.

-