Homework-01: Embedded IoT Systems (Fall 2025)

Name: SYED ZAIN RAZA ZAIDI

Roll No: 23-NTU-CS-FL-1096

Class: CS-B (5th)

Submitted To: Sir Nasir Mehmood

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?**

- This line creates an object named server from the WebServer class, which is used to manage incoming HTTP requests.
- Port 80 represents the standard network port for HTTP traffic, allowing the ESP32 to serve web pages to a browser.

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

This command defines a route handler for the web server. It tells the ESP32 that when a client (web browser) requests the root URL ("/"), the server should execute the specific function named handleRoot() to generate and send the response.

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

- It is placed in the loop() so the ESP32 can continuously listen for and process incoming HTTP requests from clients.
- If removed, the web server will stop responding to browser requests because the processor will never check the network buffer for incoming connections

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

This statement sends the final HTTP response back to the client.

- **200:** The HTTP status code indicating "OK" (successful request).

- **"text/html":** It specifies the content type so the browser knows to render the data as a webpage that the content being sent is an HTML document.
- **html:** The actual string variable containing the HTML code to be rendered by the browser. It contains webpage content,

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

- Displaying the last measured values (stored in global variables lastTemp and lastHum) is non-blocking and instant.
- Taking a fresh reading inside handleRoot() would force the code to wait for the DHT sensor to process a new reading (which can take up to 2 seconds for a DHT22). This delay would make the web page loading feel sluggish and block the processor during the request.

Part B: Long Question

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

The ESP32 webserver system functions as an integrated IoT node that monitors environmental data and serves it via a local web page. The complete workflow is as follows:

1. **ESP32 Wi-Fi Connection and IP Assignment**

Upon booting, the setup() function initiates the Wi-Fi connection using WiFi.begin(ssid, password). The system enters a while loop, checking WiFi.status() until it confirms a connection (WL_CONNECTED). Once connected, the router assigns a local IP address to the ESP32, which is printed to the Serial Monitor and displayed on the OLED screen so the user knows where to access the web server.

2. **Web Server Initialization and Request Handling**

The web server is initialized on port 80 using WebServer server(80). Routes are defined using server.on("/", handleRoot), linking the root URL path to the handleRoot function. In the main loop(), server.handleClient() runs continuously to listen for incoming HTTP requests and route them to the appropriate handler.

3. **Dynamic HTML Webpage Generation**

When a user accesses the ESP32's IP address, the handleRoot() function generates a dynamic HTML page. It constructs a String variable html that includes standard HTML tags combined with the sensor data variables (lastTemp and lastHum). This allows the page to display the current state of the sensors rather than static text.

4. **Button-Based Reading and OLED Update**

To ensure efficient operation, sensor readings are not taken continuously. A physical button is connected to BUTTON_PIN. In the loop(), the code checks for a falling edge (button press). When detected, it calls readDHTValues() to fetch fresh data from the sensor and showOnOLED() to update the local display. This updates the global variables that the web server will display on the next page load.

5. **Purpose of Meta Refresh**

The generated HTML includes the tag <meta http-equiv='refresh' content='5'>. This instructs the web browser to automatically reload the page every 5 seconds. This ensures the user sees the most recent data without manually refreshing the browser.

6. **Common Issues and Solutions**

- **Blocking Code:** Using delay() in the loop prevents server.handleClient() from running, causing timeouts.
- *Solution:* Use non-blocking timers (like millis()).
- **Power Instability:** The ESP32 requires significant current during Wi-Fi transmission.
- *Solution:* Use a high-quality 5V power supply or a capacitor across power rails.
- **Sensor Errors:** NAN readings occur if the sensor wiring is loose or the wrong DHT type or wrong i2c address is defined.
- *Solution:* Verify wiring , i2c address and the #define DHTTYPE line.

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?

The Blynk Template ID acts as a unique identifier that links the hardware code to a specific project configuration in the Blynk Cloud. It must match the cloud template so that the Blynk server knows which datastreams, mobile dashboard layout, and widgets are associated with the device connecting.

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

- **BLYNK_TEMPLATE_ID** identifies the *design*, layout or blueprint of the project (which datastreams and widgets exist).

- **BLYNK_AUTH_TOKEN** is a unique secret key for a specific *device instance,* serving as the "password" to authenticate that specific board to the cloud.
- **For example**, you might have one Template ID for a "Smart Home" project, but five different Auth Tokens for five different ESP32 units running that project.

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

- Using DHT22 code for a DHT11 sensor fails because they use different data protocols and timing for signal transmission.
- A key difference is the resolution and range: DHT22 reads decimal values (e.g., 25.4°C) with higher accuracy, whereas DHT11 typically provides integer values (e.g., 25°C) and has a smaller sensing range.

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

- Virtual Pins (V0, V1, V2, etc.) are logical channels used to exchange data between the hardware and the Blynk app.
- They are preferred over physical GPIO pins because they are platform-independent and can carry various data types (integers, floats, strings) to any widget, whereas physical pins are limited to direct hardware states like HIGH/LOW.

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

- BlynkTimer allows functions to execute at specific set intervals (e.g., sending data every 5 seconds) without blocking the main processor.
- delay() halts the entire program, preventing the Blynk.run() command from executing, which often leads to the device disconnecting from the Blynk Cloud due to a timeout.

Part B: Long Question

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

The workflow for interfacing an ESP32 with the Blynk Cloud involves configuring both the web dashboard and the hardware firmware to establish a bidirectional data link.

1. Creation of Blynk Template and Datastreams

The process begins in the Blynk Console by creating a new **Template**. Within this template, **Datastreams** are defined to handle specific data types. For this project, two datastreams are created on Virtual Pins:

- **V0:** Configured as a Double data type for Temperature.
- **V1:** Configured as a Double data type for Humidity.

2. Role of Identifiers (Template ID, Name, Auth Token)

Once the template and a device are created in the cloud, three critical credentials are generated: the **Template ID, Template Name, and Auth Token**. These must be copied into the top of the blynk.cpp.

- BLYNK_TEMPLATE_ID tells the server which project configuration to load.
- BLYNK_AUTH_TOKEN acts as the password, allowing the specific ESP32 board to authenticate and write data to the cloud dashboard.

3. Sensor Configuration and Data Reading

In the firmware, the DHT sensor is initialized using the correct pin and type (e.g., #define DHTTYPE DHT22) to avoid sensor configuration issues. The function **dht.readTemperature()** and **dht.readHumidity()** captures the environmental data into float variables.

4. Sending Data via Virtual Pins

To send this data to the cloud, the command Blynk.virtualWrite(pin, value) is used.

- Blynk.virtualWrite(V0, t): It sends the temperature value to the V0 datastream.
- Blynk.virtualWrite(V1, h): It sends the humidity value to the V1 datastream.
- This updates the associated widgets on the Blynk Mobile App and Web Dashboard instantly.

5. Common Problems and Solutions

- **"Offline" Device:** Often caused by using delay () inside the loop(), which blocks the Blynk.run() maintenance process.
- *Solution:* Use BlynkTimer for periodic actions.
- **Invalid Auth Token:** If the token doesn't match the one in the cloud, the device will be stuck in a "Connecting" state.
- *Solution:* Double-check the token string in the code.
- **Incorrect Values:** Caused by defining the wrong sensor type (DHT11 vs DHT22).
- *Solution:* Ensure DHTTYPE matches the physical sensor used.