Name	:	Haider Ali
Registration #	:	23-NTU-CS-1254
Assignment #	:	2
Course Name	:	IoT and Embedded Systems
Semester	:	5th
Section	:	BSCS-A
Department	:	DCS

Assignment 2

Q1: ESP32 WebServer

Part A: Short Questions

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

- **Purpose:** This line instantiates an object named `WebServer server`. It initializes the ESP32 to listen for incoming HTTP requests on the specified port.
- **Port 80:** This sets the communication channel to TCP port 80. Port 80 is the standard default for HTTP web traffic. It allows users to access the server via a browser using just the IP address (e.g., `http://192.168.1.100`) without manually appending a port number like `:8080`.

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

- **Role:** This defines a specific route handler. It maps the root URL path (`/`) to the function `handleRoot()`.
- **Function:** When a client requests the root address, the server automatically triggers the `handleRoot` function to generate and send the HTML response.

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

- **Reason:** The `loop()` runs continuously. Placing `server.handleClient()` here ensures the ESP32 constantly checks for and processes incoming HTTP requests.
- **Consequence of Removal:** If removed, the server initiates but never listens for connections. Browsers attempting to connect will time out because the ESP32 is not actively handling the request client queue.

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

- **200:** The HTTP Status Code for "OK," indicating a successful request.
- **"text/html":** The MIME type indicating the content is an HTML document, ensuring the browser renders it as a webpage.
- **html:** The string variable containing the actual HTML code to be displayed.

5. What is the difference between displaying "last measured" values versus taking a fresh reading inside `handleRoot()`?

- **Last Measured (Current):** Displays data stored in global variables `(lastTemp, lastHum)` which only update when the physical button is pressed. If the button isn't pressed, the web data may be stale.

- **Fresh Reading:** If were called inside `readDHTValues()` `handleRoot()`, the sensor would measure *every* time the page loads. This guarantees real-time data for the web user regardless of the physical button state.

Part B: Long Question

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

The system operates as a hybrid node, providing both local (OLED) and remote (Web) monitoring. The workflow consists of four stages:

1. Network Initialization

- **Connection:** In , the ESP32 uses `setup()` `WiFi.begin(ssid, password)` to connect to the local network. It enters a blocking loop until the connection is established.
- **Addressing:** The router assigns an IP address via DHCP, which is printed to the Serial Monitor and OLED. This is the address the user visits to view the data.
- **Server Start:** The web server starts on Port 80 and registers the root path handler.

2. Sensor & Input Handling

- **Monitoring:** The `loop()` checks the physical button on GPIO 5.
- **Action:** Upon detecting a button press (edge detection):
    - The ESP32 queries the DHT22 sensor.
    - If valid, it updates the global `lastTemp` and `lastHum` variables.
    - The OLED display is immediately updated with these new values.

3. Request Processing

- **Listening:** `server.handleClient()` constantly monitors for incoming traffic.
- **Response:** When a browser hits the IP address:
    - The `handleRoot()` function is executed.
    - It dynamically constructs an HTML string, injecting the current values of `lastTemp` and `lastHum`.
    - If no data exists yet (fresh boot), it displays a placeholder message.

4. Client-Side features

- **Auto-Refresh:** The HTML includes a meta-tag: `<meta http-equiv='refresh' content='5'>`. This forces the browser to reload every 5 seconds, ensuring that if new sensor data is captured via the button, it appears on the screen automatically without manual refreshing.

5. Troubleshooting Common Issues

- **Sensor Failures:** Often caused by loose wiring or mismatching pin definitions in code.
- **Connection Timeout:** Usually due to the accessing device being on a different subnet/network than the ESP32.
- **Wi-Fi Failure:** ESP32 requires 2.4GHz Wi-Fi; it cannot connect to 5GHz bands.

Question-2: Blynk Cloud Interfacing (blynk.cpp)

Part A: Short Questions

1. What is the role of Blynk Template ID? Why must it match the cloud template?

- **Role:** The `BLYNK_TEMPLATE_ID` links the hardware to a specific project configuration in the Blynk Cloud (datastreams, widgets, UI).
- **Matching:** If the ID in firmware doesn't match the Cloud Console, the server rejects the connection, as it cannot map the data to the correct user interface.

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

- **Template ID:** Identifies the *Model/Design* of the device (e.g., "Weather Station"). Shared by all devices using that firmware.
- **Auth Token:** Identifies the *Specific Device Instance* (e.g., "Living Room Unit"). It is the unique security key permitting that specific board to connect.

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

- **Reason:** The DHT11 and DHT22 use different timing protocols for signal transmission. If the code expects DHT22 (high resolution) but receives DHT11 signals, parsing fails.
- **Result:** This leads to `NAN` (Not A Number) or erratic values because the data bits are decoded incorrectly.
- **Hardware Diff:** DHT22 has higher accuracy and a wider temp range (-40 to 80°C) compared to the DHT11 (0 to 50°C).

4. What are Virtual Pins? Why are they preferred over physical GPIOs?

- **Definition:** Virtual Pins (V0, V1) are logical channels for data exchange, not physical wires.
- **Preference:** They allow sending processed data (integers, floats, strings) rather than simple HIGH/LOW voltage states. They also decouple the app interface from the physical wiring logic.

5. What is the purpose of instead of `BlynkTimer` `delay()`?

- **Purpose:** `BlynkTimer` is non-blocking. It allows the main loop to continue running while waiting for an interval.

- **Necessity:** Using `delay()` pauses the processor completely. This stops `Blynk.run()`, causing the "Heartbeat" to fail and the device to disconnect from the cloud.

Part B: Long Question

Explain the workflow of interfacing ESP32 with Blynk Cloud for temperature/humidity monitoring.

The integration involves Cloud Configuration, Firmware Setup, and Data Transmission.

1. Cloud Configuration (Dashboard) Before coding, the project is defined in the Blynk Console:

- **Template:** Created to generate the `BLYNK_TEMPLATE_ID`.
- **Datastreams:** Virtual pins are assigned to data types.
  - **V0:** Data type `Double` (for Temperature).
  - **V1:** Data type `Double` (for Humidity).
- These mappings ensure `Blynk.virtualWrite(V0, value)` sends data to the correct widget.

2. Credentials Setup The firmware requires three keys to authenticate:

- **Template ID & Name:** Categories the device type.
- **Auth Token:** The unique password for the specific board. This is sent on boot to authorize the connection.

3. Sensor Initialization

- The code initializes the sensor using `DHT dht(DHTPIN, DHTTYPE);`.
- **Critical Check:** The `DHTTYPE` (e.g., DHT22) must match the physical sensor. A mismatch (e.g., using a DHT11) results in failed data parsing and `NAN` errors.

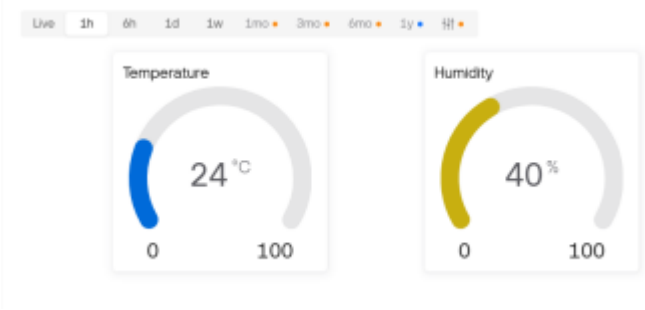
4. Data Transmission Loop To prevent server flooding, data is sent periodically, not continuously:

- **Timer:** A calls the `BlynkTimer` `periodicSend` function every 5 seconds.
- **Manual Trigger:** A physical button on GPIO 5 can also force an immediate read.
- **Pushing Data:** Inside the function, data is pushed using:
  - `Blynk.virtualWrite(V0, t);` (Sends temp to V0)
  - `Blynk.virtualWrite(V1, h);` (Sends humidity to V1)

5. Troubleshooting

- **Device Offline:** Usually caused by blocking delays in the loop preventing `Blynk.run()` from maintaining the heartbeat.
- **WiFi Fail:** Incorrect credentials or attempting to connect to a 5GHz network.
- **NAN Values:** Loose wiring or incorrect Sensor Type definition in the code.

Blynk Dashboard:



Mobile app: