Assignment: IOT
Name: MUHAMMAD ABDULLAH QADRI Roll No: 1266

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?

- **Purpose of `WebServer server (80);`**: This line initializes the web server object. The number 80 specifies the TCP Port (the "door") the server uses to listen for incoming HTTP requests from web browsers.
- **Port 80**: This is the global standard default port for HTTP (Hypertext Transfer Protocol) services. If an IP address is typed into a browser without a specific port, the browser automatically attempts to connect via Port 80.

---

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

- **Role**: This function acts as a **Router**, linking a specific web address path to a specific function in the code.
- **"/" (The Root)**: This represents the home page. When a user requests only the ESP32's IP address, they are accessing this root path.
- **handleRoot**: This is the **Callback Function** that the Arduino code must execute when a request for the "/" path is received.

---

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

- **Why it is in `loop()`**: This function must run constantly to check the network for new incoming requests from clients (like browsers) and handle them immediately. Placing it in the `loop()` ensures no requests are missed.
- **If it is removed**: The web server will fail to function. The ESP32 will not detect connection attempts, and the browser will eventually display a "Connection Timeout" error.

---

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

- **200**: The HTTP Status Code for "OK," meaning the request was successful and content is being sent.
- **"text/html"**: The Content Type, informing the browser that the data is in HTML format so it can be displayed correctly.
- **html**: The String Variable containing the complete, dynamically created HTML code, including the current sensor values.

---

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

Feature	Method A: Using Stored Variables	Method B: Real-Time Sensor Sampling
Data Retrieval	Accesses information already saved in global variables from the last background update.	Executes a physical scan of the DHT sensor at the exact moment the page is requested.
Efficiency	**High Speed**: The response is nearly instantaneous since the data is ready in memory.	**Low Speed**: The process is delayed because the sensor requires several hundred milliseconds to complete a reading.
System Impact	**Optimized**: Keeps the web server highly responsive and prevents lag during user interaction.	**Risk Factor**: Can lead to system crashes or timeouts if multiple users access the site simultaneously.

Part B: Long Question

---

1. ESP32 Wi-Fi Connection Process and IP Address Assignment

- **Wi-Fi Authentication**: The ESP32 initiates a connection to the local access point by providing the predefined SSID and security credentials.
- **IP Allocation**: Upon successful authentication, the local router assigns a unique Private IP Address to the ESP32.
- **Network Identification**: This specific IP address serves as the destination URL that a user must enter into their browser to access the monitoring interface.

---

2. Web Server Initialization and Request Handling

- **Service Activation**: The HTTP service is launched on the standard Port 80 through the `server.begin()` command.
- **Endpoint Mapping**: Using `server.on()`, the system establishes a "listening rule" that links the root URL ("/") to the `handleRoot` function.
- **Continuous Monitoring**: The `server.handleClient()` method resides in the main execution loop to intercept and process incoming data requests from clients in real-time.

---

3. Button-based Sensor Reading and OLED Update Mechanism

- **Data Sampling Triggers**: The firmware is programmed to sample the environment either through a physical button interrupt or an automated internal timer.
- **Sensor Polling**: Once the trigger occurs, the DHT22 retrieves the current thermal and moisture levels.
- **OLED Synchronization**: The retrieved data is instantly transmitted to the integrated OLED screen for local viewing.
- **Variable Storage**: These updated values are cached in global variables, ensuring they are ready for the web server to distribute

---

4. Dynamic HTML Webpage Generation

- **HTML Construction**: Within the `handleRoot` function, the ESP32 assembles a temporary HTML document stored as a string.
- **Live Data Injection**: The system embeds the most recent sensor variables directly into the HTML structure.
- **Variable Content**: This ensures that every time a user accesses the site, the content is updated based on the latest hardware state.
- **Data Transmission**: The finalized document is dispatched to the client's browser via the `server.send()` protocol

---

5. Purpose of Meta Refresh in the Webpage

- **Automatic Updates**: The HTML header includes a "Meta Refresh" instruction.
- **Function**: This command prompts the browser to re-request the page at a fixed interval, such as every 5 seconds.
- **Necessity**: Because the ESP32 operates as a passive server, this tag ensures the dashboard stays current without requiring the user to manually reload the browser.

---

6. Common Issues in ESP32 Webserver Projects and their Solutions

- **`Problem: High Latency or Freezing:`** `This is frequently caused by using delay() functions, which halt all CPU processes, including the web server.`
- **Fix**: Replace all delays with non-blocking logic, such as the `millis()` function or dedicated timers.
- **Problem: Unstable Network Link**: Environmental factors or router timeouts can cause the ESP32 to drop from the Wi-Fi.
- **Fix**: Implement a routine in the loop to monitor the connection status and execute a reconnection sequence if the signal is lost.

Question 2: Blynk Cloud Interfacing (blynk.cpp)

Part A: Short Questions

1. The Function of the Blynk Template ID

- Project Identifier: The Template ID acts as a primary digital signature or serial number for your IoT project.
- **Cloud Synchronization**: It informs the ESP32 about the specific architectural layout and

dashboard configuration it must link to on the Blynk Cloud.
- **Connection Requirement**: Precise matching is mandatory; if the ID is incorrect, the Cloud cannot route data packets, and the hardware will fail to establish a connection.

---

2. Comparison: Template ID vs. Auth Token

Feature	Blynk Template ID	Blynk Auth Token
Identification Level	Identifies the **project category** (e.g., "Solar Monitor"). It is identical for every device within that specific project type.	Identifies a **unique hardware unit** (e.g., "Device 01"). Every individual ESP32 receives its own distinct token.
Primary Objective	Synchronizes the firmware with the correct project dashboard and data structures on the Cloud.	Serves as a digital key to **authenticate and log in** a specific device to the Blynk server.

---

3. Technical Conflicts: Using DHT22 Code with a DHT11 Sensor

- Protocol Mismatch: DHT sensors operate on strict, time-sensitive communication protocols.
- **Data Misinterpretation**: If the ESP32 is programmed for the DHT22 format while using a DHT11, it will misinterpret the incoming data bits, leading to corrupted readings.
- **Core Differences**:**DHT11**: Limited to whole-number precision and a narrower detection range.
- **DHT22**: Provides high-accuracy decimal readings and operates across a significantly wider environmental range.

4. The Utility of Virtual Pins in Blynk

- Logical Channels: Unlike physical GPIOs, Virtual Pins (V0, V1, etc.) are software-based communication channels.
- **Data Mailboxes**: They function as organized digital mailboxes for the exchange of information between the hardware and the app.
- **Key Advantages**:**Versatility**: They are not restricted to binary signals and can transmit complex data types like floats, strings, or system status updates.
- **Bi-directional Flow**: They support a dual-path system where the ESP32 sends sensor data to the Cloud, and the Cloud sends control signals back to the ESP32.

---

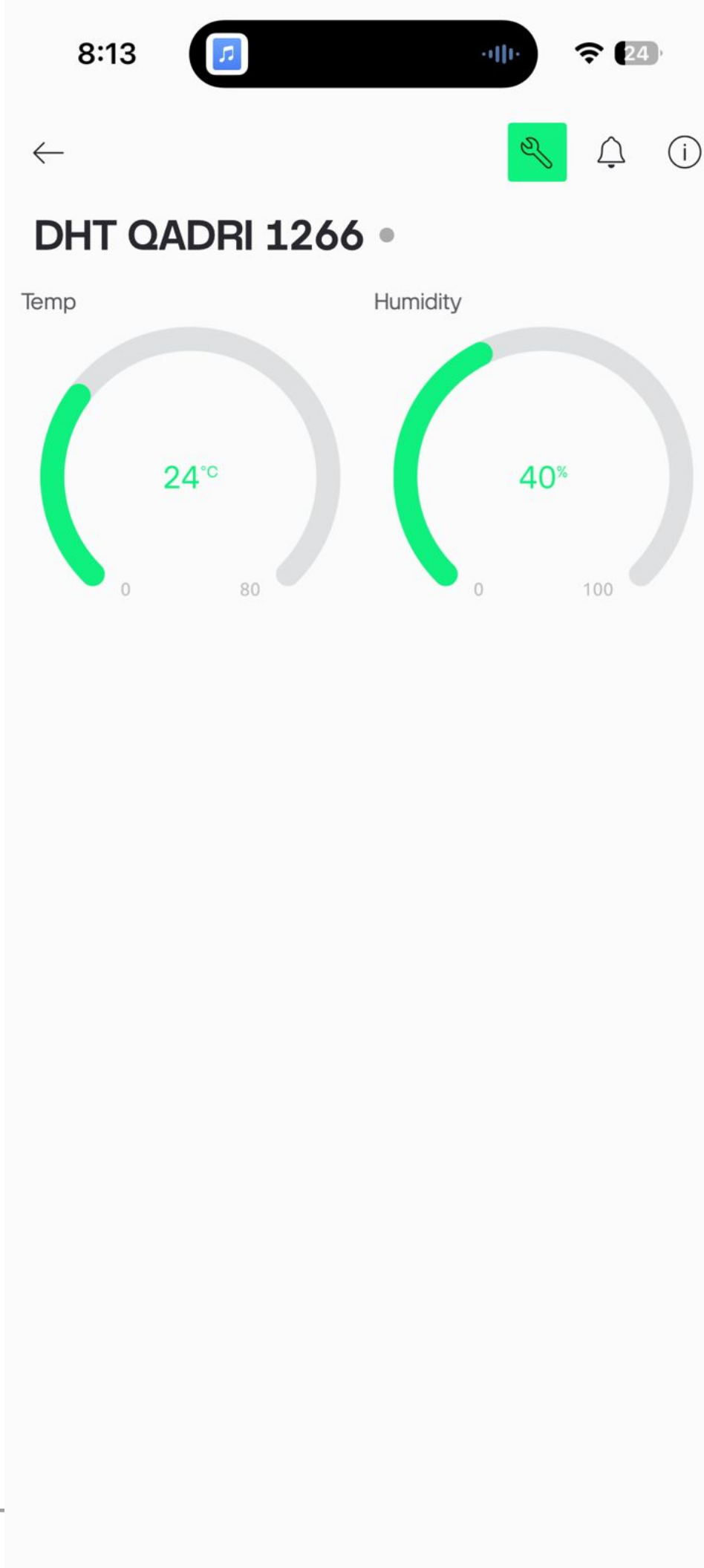
5. Operational Benefits of BlynkTimer over `delay()`

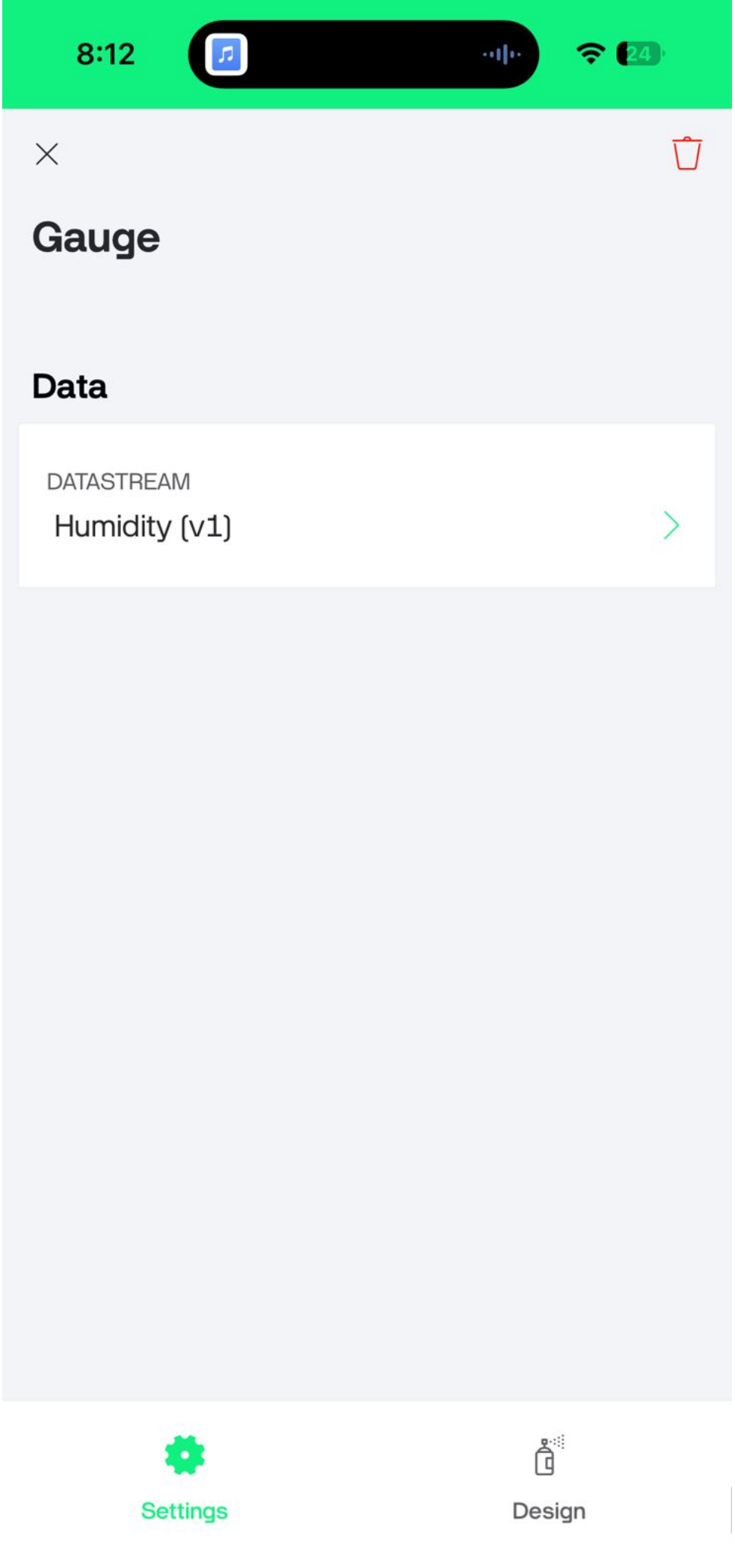
- **The `delay()` Limitation**: Using `delay()` creates a "blocking" state that halts all CPU activity, causing Wi-Fi checks to fail and the Blynk cloud connection to drop.

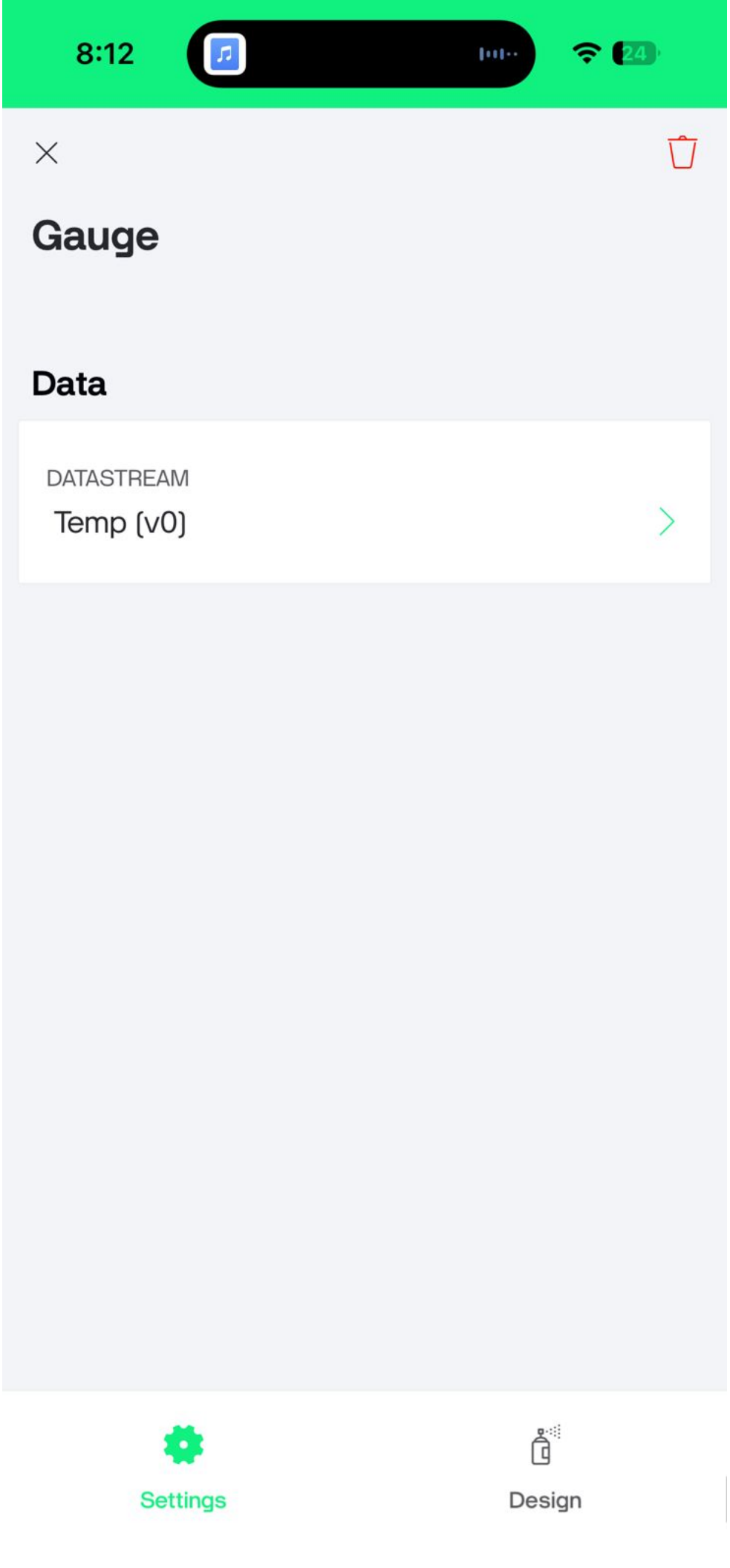
- **The BlynkTimer Solution**: This is a non-blocking scheduler designed for multi-tasking.
- **Interval Execution**: It allows specific tasks (like sensor reading) to run at set times (e.g., every 5 seconds) without pausing the rest of the code.
- **Continuous Connectivity**: While the timer waits, the `loop()` continues to execute `Blynk.run()`, ensuring the device remains online and responsive to app commands.

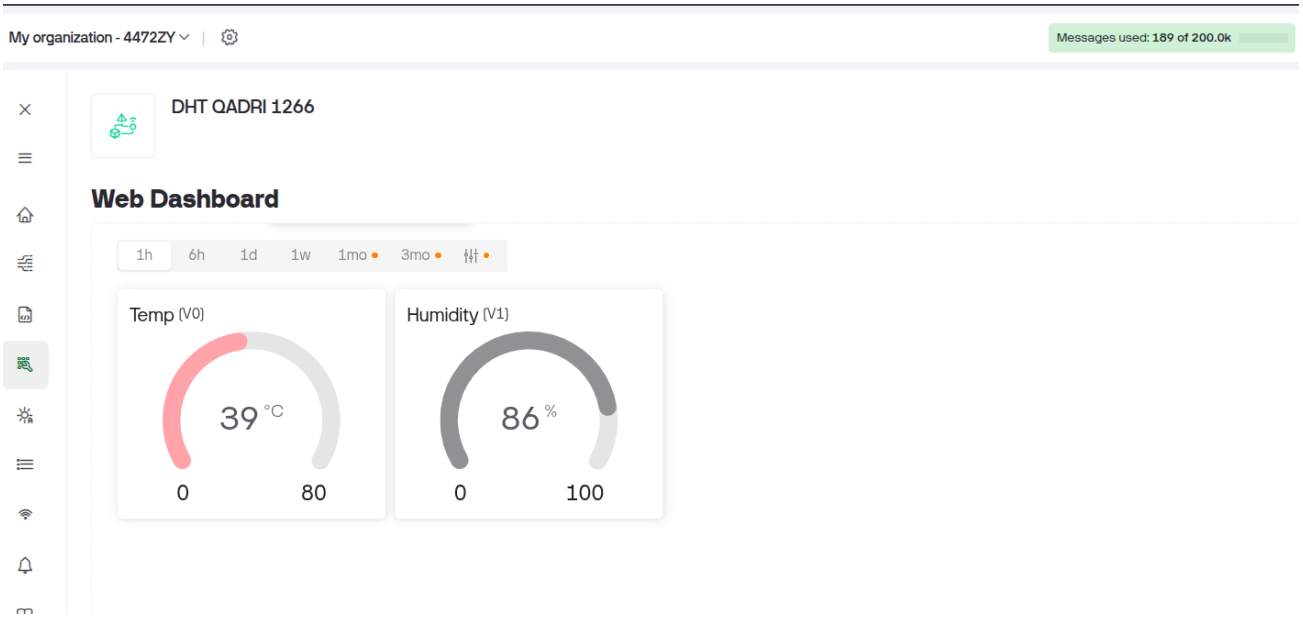
Part B: Long Question – Complete Workflow

1. **Create Template + Datastreams:**


8:13
24
DHT QADRI 1266
Temp
24°C
0
80
Humidity
40%
0
100


8:12
24
Gauge
Data
DATASTREAM
Humidity (v1)
Settings
Design


8:12
24
Gauge
Data
DATASTREAM
Temp (v0)
Settings
Design



2. Project Authentication: Template ID, Name, and Auth Token

- Credential Generation: Upon successfully registering a new device in the Blynk Console, the system generates a unique Template ID, Template Name, and Authentication Token.
- **Firmware Integration**: These credentials must be precisely copied into the ESP32 source code.
- **Cloud Handshake**: This step is critical as it serves as the digital handshake, ensuring the hardware logs into the correct project and establishes a secure link with the Blynk Cloud.

3. Hardware Configuration: Resolving Sensor Discrepancies

- Defining Sensor Type: It is mandatory to specify the exact hardware model (DHT11 vs. DHT22) within the code's definitions.
- **Data Accuracy**: Selecting the **incorrect sensor type** will lead to communication errors or the transmission of distorted, inaccurate environmental data to the dashboard.

---

4. `Data Transmission via Blynk.virtualWrite()`

- **Scheduled Execution**: Rather than sending data constantly, a **BlynkTimer** is used to trigger the reading function at a fixed interval, such as every few seconds6.
- **Sensor Acquisition**: During each interval, the function polls the DHT22 for current temperature (T) and humidity (H) metrics7.
- **Cloud Pushing**:
    - The command Blynk.virtualWrite(V0, T); dispatches the temperature reading to the cloud via Virtual Pin 08888.
    - Similarly, Blynk.virtualWrite(V1, H); transmits the humidity level to the cloud using Virtual Pin 19999.

5. Interface Visualization and Dashboard Setup

- **Widget Configuration**: Users can customize their Blynk dashboard on both web and mobile platforms by adding visual elements like **Gauges**, **Level Sliders**, or **Real-time Charts**.
- **Datastream Linking**: These widgets must be mapped to the corresponding Virtual Pins (e.g., linking a Gauge to V0 for Temperature).
- **Real-time Monitoring**: Once the ESP32 begins its broadcast, the dashboard widgets reflect the sensor changes instantly, providing a live monitoring experience.

---

6. Debugging and Maintenance

- **Symptom: Device Status "Offline"**:
- **Root Cause**: Usually results from incorrect Wi-Fi credentials (SSID/Password) or mismatched Auth Tokens.

- **Resolution**: Verify all credentials for typographical errors and confirm that `Blynk.run()` is present and unobstructed in the main `loop()`.

- **Symptom: Stale or Frozen Data**:

- **Root Cause**: Occurs when the data transmission loop is broken or timed incorrectly.

- **Resolution**: Adjust the **BlynkTimer** frequency to a reasonable rate (e.g., 5 seconds) and ensure the `Blynk.virtualWrite()` logic is being reached by the processor.
