NAME: Talha mehmood

REG NO: 1286

QUESTION NO 1 (ESP32)

PART A (SHORT QUESTIONS)

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

Purpose: This line creates an "object" or an instance of the WebServer class named server. It tells the ESP32 to set aside resources to listen for incoming network requests.

Port 80: This is the standard **HTTP port**. When you type an IP address into a web browser (like 192.168.1.50), the browser automatically tries to connect via port 80. By setting the server to 80, you ensure that the user doesn't have to type a specific port number in the URL

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

This is a **route handler**. It links a specific URL path to a specific function in your code:

- **"/"**: This represents the "root" or home page of your website.
- **handleRoot**: This is the name of the function you wrote.
- **Role:** It tells the ESP32: *"If a user visits the main IP address of this device, immediately execute the handleRoot() function to decide what to show them."*

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

Why it's in loop(): The ESP32 cannot do two things at the exact same time. server.handleClient() is the command that tells the ESP32 to "stop for a millisecond and

check if anyone is trying to connect to the website." Because it is in the loop(), the ESP32 checks for new visitors thousands of times per second.

If removed: The web server will **stop responding**. Even if the ESP32 is connected to WiFi, it will never "hear" the browser's request. The browser will eventually time out and show a "Site cannot be reached" error

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

This statement is the final response sent back to the user's browser. It has three parts:

1. **200**: The HTTP status code for **"OK."** It tells the browser that the request was successful.
2. **"text/html"**: This is the **Content Type**. It tells the browser that the data being sent should be rendered as a webpage, not just plain text or an image.
3. **html**: This is the String variable containing the actual code (tags like <h2>, <p>, etc.) that makes up the visual webpage.

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

The main difference is below

- **Displaying last measured values:**
  The webpage shows the temperature and humidity that were previously read from the DHT sensor and stored in variables. This makes the page load very fast, but the data may be old if no new reading was taken recently.
- **Taking a fresh DHT reading:**
  The sensor is read every time handleRoot() runs (when the page is refreshed). This ensures the data is always up to date, but the webpage responds more slowly because a DHT sensor takes about 1–2 seconds to provide a reading.

Last measured values = fast response, possibly outdated data.
Fresh reading = accurate data, slower page loading.

PART B (LONG QUESTION)

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

ESP32 Wi-Fi connection process and IP address assignment

The system begins in setup() by initializing the Wi-Fi hardware.

- **The Process:** Using WiFi.begin(ssid, password), the ESP32 attempts to join the "Wokwi-GUEST" network. It runs a while loop that checks the connection status every 500ms.
- **IP Assignment:** Once connected, the local router assigns the ESP32 a unique **Local IP Address**. The code retrieves this via WiFi.localIP() and displays it on the **Serial Monitor** and the **OLED screen**. This IP address is the "home address" of your web server.

Web server initialization and request handling

The code sets up a server that listens for incoming communication.

- **Initialization:** The line WebServer server(80); sets the server to listen on **Port 80**, which is the default for all web browsers. In setup(), server.on("/", handleRoot) creates a "route" that links the home page to the handleRoot function.
- **Handling Requests:** In the loop(), server.handleClient() runs constantly. Whenever a user types the ESP32's IP into a browser, this function "catches" that request and tells the ESP32 to run the handleRoot logic to send back the webpage.

Button-based sensor reading and OLED update mechanism

Unlike systems that read data every second, this code is **event-driven** to save power and prevent sensor wear.

- **The Trigger:** The loop() monitors BUTTON_PIN (GPIO 5). When it detects a "falling edge" (the button is pressed to connect to GND), it triggers the reading.
- **Reading & Display:** The readDHTValues() function communicates with the **DHT22 sensor** (on GPIO 23) to fetch temperature and humidity. These are saved to global variables. Immediately after, showOnOLED() updates the **SSD1306 OLED** with the new values, providing instant local feedback.

Dynamic HTML webpage generation

When handleRoot() is called by a web request, it doesn't just send a fixed file; it generates a **Dynamic String**.

- ** assembly:** The ESP32 builds a String html starting with standard headers.
- **Data Insertion:** It uses if/else logic to check if lastTemp and lastHum have valid numbers. If they do, it converts the numbers to text and inserts them directly into the HTML code. This ensures the website always shows the **most recent button-triggered data**.
- **Sending:** server.send(200, "text/html", html) pushes this final webpage to the user's phone or computer.

Purpose of meta refresh in the webpage

Inside the HTML code, you have the tag: <meta http-equiv='refresh' content='5'>.

- **Automatic Updates:** Normally, a webpage is static; it won't change unless you manually click "refresh."
- **The Role:** This tag instructs the browser to automatically reload the page every **5 seconds**. This allows the user to see the new data on their phone shortly after pressing the physical button, without having to touch their screen.

Common issues in ESP32 webserver projects and their solutions

1. Sensor Communication Fault

- **Problem:** The screen displays **"DHT Error!"** or the Serial Monitor shows "Failed to read from DHT!"
- **Cause:** A physical break in the data line or insufficient power. The DHT22 is sensitive to wiring quality and requires a steady voltage to send digital data to **GPIO 23**.
- **Solution:** Inspect the jumper wires on Pin 23. Ensure the DHT22 is powered by the **3.3V** pin and shares a common Ground (GND) with the ESP32.

2. Website Accessibility Error

- **Problem:** The browser shows **"Site cannot be reached"** or a "Connection Timed Out" message.
- **Cause:** A network mismatch. Your phone/PC and the ESP32 are not on the same "sub-network." If your phone is using 4G/5G data or a different Wi-Fi, it cannot access the ESP32's private IP.
- **Solution:** Confirm both devices are connected to **"Wokwi-GUEST."** Double-check that the IP address typed into the browser matches the one shown on the OLED.

3. Empty Data Display

- **Problem:** The web server works, but it says **"No valid data yet."**
- **Cause:** The sensor hasn't been triggered. In your code, readDHTValues() only runs inside the button-press logic. On startup, the variables are "empty" (NAN).
- **Solution:** Press the physical button on **GPIO 5** once. This populates the variables, and the webpage will show the data on its next auto-refresh.

4. OLED Initialization Failure

- **Problem:** The display is frozen on **"Booting..."** or is completely blank.
- **Cause:** I^2C communication breakdown. This happens if the SDA/SCL wires are swapped or if the OLED hardware address is not 0x3C.
- **Solution:** Check the I2C wiring. If the wiring is correct but the screen is blank, change the address in display.begin() from 0x3C to 0x3D.

5. System Latency (Lag)

- **Problem:** The webpage is slow to load, or the button press doesn't register immediately.
- **Cause:** "Blocking" code in the loop(). If you add long delay() functions, the ESP32 stops running server.handleClient(), making it ignore web requests for several seconds.
- **Solution:** Ensure the loop() stays fast. Avoid adding any delay() larger than 50ms. Use the millis() function if you need to create longer timing intervals.

QUESTION NO 2 (BLYNK CLOUD INTERFACING)

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** (TMPL6UCKQ_P6D in your code) acts as the primary blueprint or "ID Card" for your device. It tells the Blynk Cloud exactly how to treat the incoming data from the ESP32.

- **Why it must match:** It defines the **Dashboard layout**, the **Virtual Pin** assignments, and the **Data Types** (like Celsius vs. Fahrenheit). If the code uses a different ID than the Cloud, the server will not know how to visualize the data, and your mobile/web dashboard will remain disconnected or empty.

2. Differentiate between Blynk Template ID and Blynk Auth Token

Blynk Template ID

The Template ID identifies a specific device template in the Blynk platform. It defines the device type, hardware, datastreams, and dashboard layout. When a device connects to Blynk, the Template ID tells Blynk **which template configuration to use**.

Blynk Auth Token

The Auth Token is a unique security key assigned to each device. It is used to **authenticate and authorize** the device so it can connect to the Blynk cloud, send sensor data, and receive commands

In short:

Template ID → Identifies the device model/template.

Auth Token → Authenticates a specific device and allows secure communication.

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

Using DHT22 code with a DHT11 sensor produces incorrect readings because the two sensors have different **data formats and timing protocols**. The code written for DHT22 expects the DHT22's specific signal timing and resolution, which the DHT11 cannot match

DHT22:

DHT22 has a wider range and higher resolution (−40 to 80 °C, 0.1 °C resolution; 0–100% RH, 0.1% resolution

DHT11:

DHT11 has a smaller range and lower resolution (0–50 °C, 1 °C resolution; 20–80% RH, 1% resolution).

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

Virtual Pins (like V0 and V1 in your code) are digital "channels" used to send data between the hardware and the app without being tied to a physical wire on the ESP32.

Why they are preferred over physical GPIO pins:

- **Flexibility:** You can send complex data like strings, colors, or decimals that a physical "High/Low" pin cannot handle.
- **Abstraction:** You can change your physical wiring (e.g., move the DHT sensor from GPIO 23 to 14) without needing to change your Blynk Mobile Dashboard layout.
- **Hardware Safety:** It prevents the Cloud from accidentally toggling a pin that might be connected to sensitive hardware.

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

1. delay()

- Pauses the entire program for a set amount of time.
- **Effect:** During this pause, the ESP32 cannot do anything else (no sensor reading, no Blynk communication).

2. BlynkTimer

- Schedules functions to run at specific intervals **without stopping** the main program.
- **Effect:** The ESP32 can continue other tasks while the timer triggers periodic functions.

3. Why use BlynkTimer instead of delay()

- Keeps the device **responsive**.
- Allows **multitasking** (reading sensors, sending data, controlling devices simultaneously).
- Makes the program **efficient and smooth**

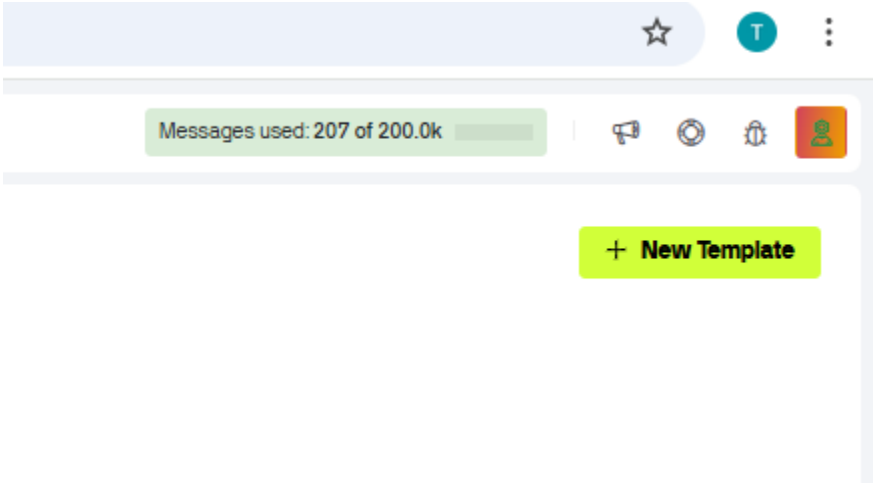
PART B(LONG QUESTION)

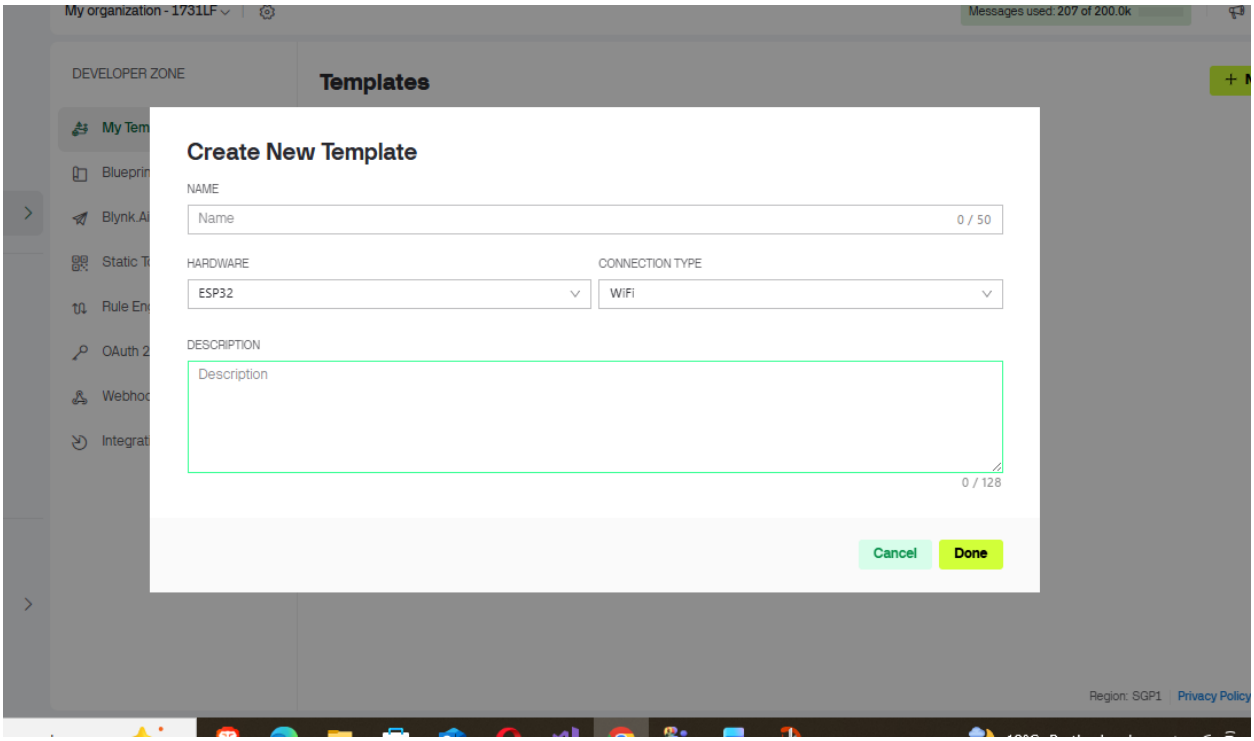
Explain the complete workflow of interfacing ESP32 with Blynk Cloud to display temperature and humidity values.

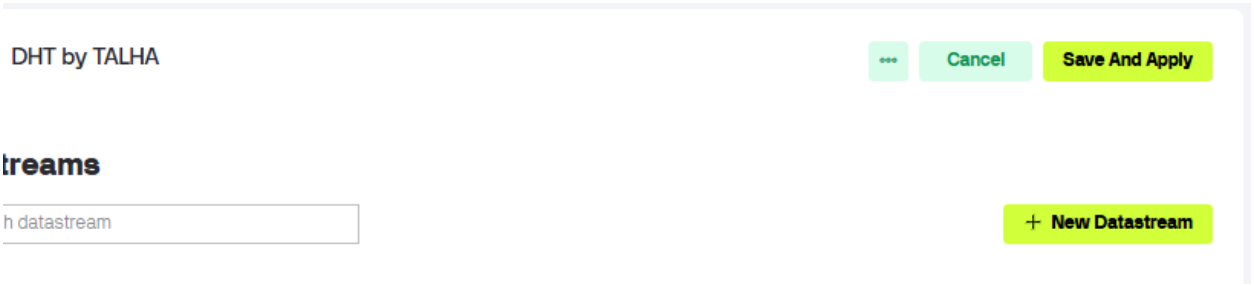
Creation of Blynk Template and Datastreams

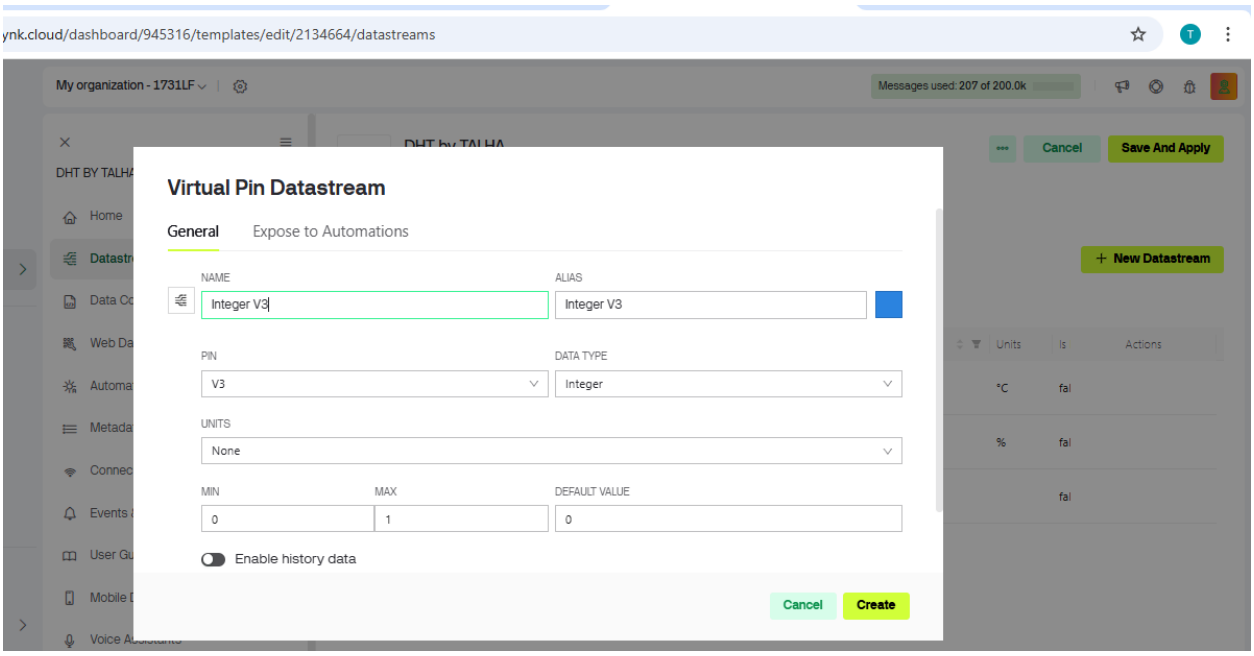
To connect hardware to Blynk, you must first create a **Template** in the Blynk Console, which acts as a blueprint for the device.

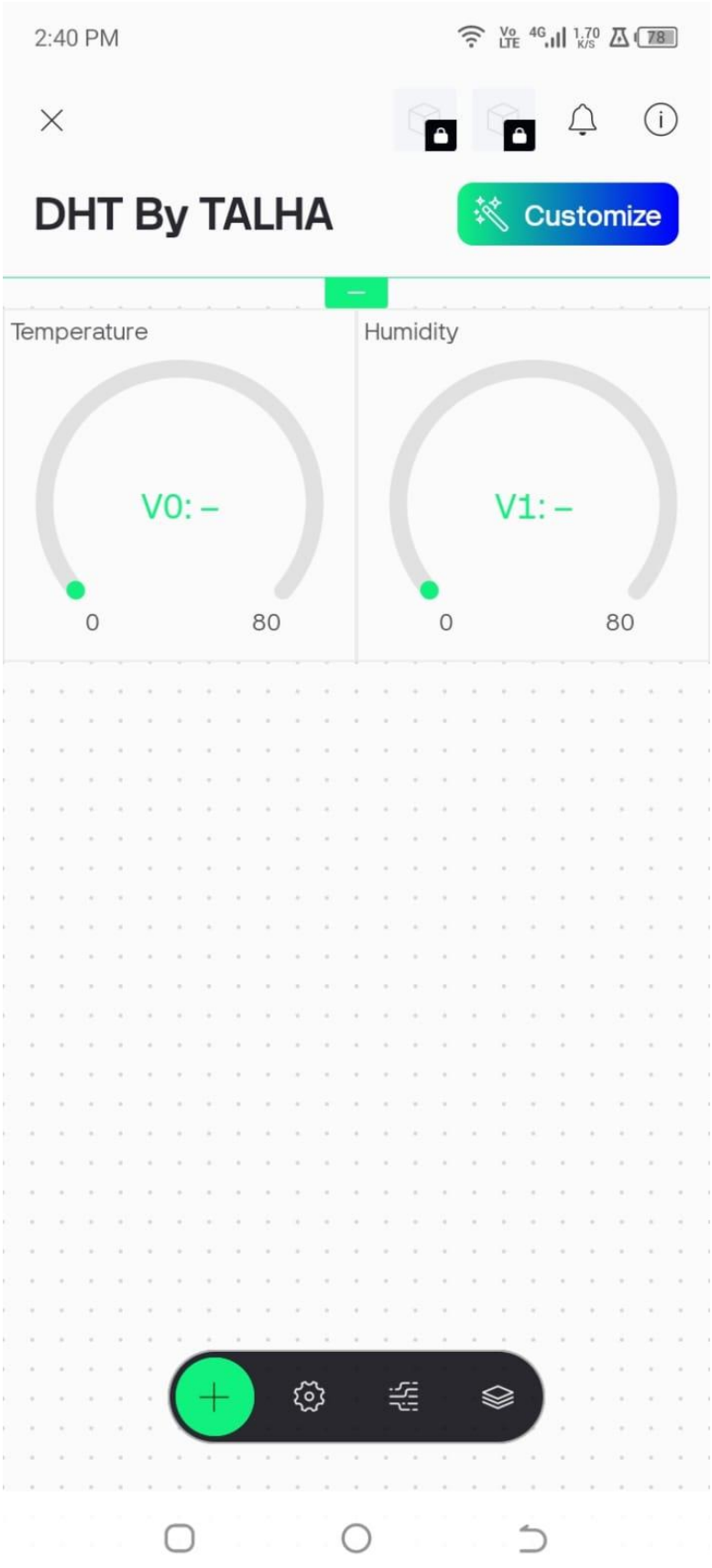
- **Template Creation:** In the Developer Zone, a new template (e.g., "DHT by TALHA") is created to define the hardware (ESP32) and connection type (Wi-Fi).
- **Datastreams:** You must set up **Virtual Pin Datastreams** to act as digital channels for data. Based on the code, you would create:
    - **V0:** Named "Temperature," configured as a "Double" data type to handle decimal values.
    - **V1:** Named "Humidity," also configured as a "Double."




My organization - 1731LF
Messages used: 207 of 200.0k
DEVELOPER ZONE
Templates
Create New Template
NAME
Name
0 / 50
HARDWARE
ESP32
CONNECTION TYPE
WiFi
DESCRIPTION
Description
0 / 128
Cancel
Done
Region: SGP1
Privacy Policy


DHT by TALHA
Cancel
Save And Apply
treams
h datastream
New Datastream


ynk.cloud/dashboard/945316/templates/edit/2134664/datastreams
My organization - 1731LF
Messages used: 207 of 200.0k
DHT BY TALHA
Home
Datastr
Data Co
Web Da
Automa
Metada
Connec
Events
User Gu
Mobile
Voice A
DHT by TALHA
Cancel
Save And Apply
+ New Datastream
Units
Is
Actions
°C
fal
%
fal
fal
Virtual Pin Datastream
General
Expose to Automations
NAME
Integer V3
ALIAS
Integer V3
PIN
V3
DATA TYPE
Integer
UNITS
None
MIN
0
MAX
1
DEFAULT VALUE
0
Enable history data
Cancel
Create


2:40 PM
DHT By TALHA
Customize
Temperature
V0: –
0
80
Humidity
V1: –
0
80

Role of Template ID, Template Name, and Auth Token

These three unique identifiers are required at the very top of your code for the ESP32 to authenticate with the Blynk Cloud.

- **Template ID (BLYNK_TEMPLATE_ID):** This is the primary identifier that tells the Blynk server which dashboard layout and datastream configuration to use for the device.
- **Template Name (BLYNK_TEMPLATE_NAME):** This is the human-readable name of the project used for organization within the Blynk console.
- **Auth Token (BLYNK_AUTH_TOKEN):** This is a unique "security key" assigned to an individual physical device. It ensures that data from your ESP32 goes to your specific account and project.

Sensor configuration issues (DHT11 vs DHT22)

The code is specifically configured for the **DHT22** sensor (#define DHTTYPE DHT22).

- **Incompatibility:** Using a DHT11 sensor with DHT22 code will result in incorrect readings or "NaN" (Not a Number) errors because the sensors use different data formats and timing protocols.
- **Key Differences:** The DHT22 is more precise, providing decimal values and a wider temperature/humidity range, whereas the DHT11 typically only provides whole numbers.

Sending data using Blynk.virtualWrite()

In the readAndDisplayAndSend() function, data is pushed to the cloud using the Blynk.virtualWrite() command.

- **Functionality:** This command sends the value stored in a variable directly to a specific Virtual Pin in the Blynk Cloud.
- **Implementation:** The code uses Blynk.virtualWrite(V0, t); to send temperature and Blynk.virtualWrite(V1, h); to send humidity. This allows the values to bypass physical pins and interact directly with the software widgets on your dashboard.

Common problems faced during configuration and their solutions

Connection Failure – ESP32 cannot connect to Wi-Fi or Blynk Cloud.
Solution: Check SSID, Wi-Fi password, and Auth Token; make sure they are correct.

Incorrect Sensor Readings – DHT11/DHT22 gives wrong values.
Solution: Use the correct sensor type in the code (#define DHTTYPE DHT11 or DHT22).

Data Not Displaying in App – Virtual Pins don't show data.
Solution: Ensure Virtual Pins in the code match the datastreams in the Blynk template (e.g., V1 → Temperature, V2 → Humidity).

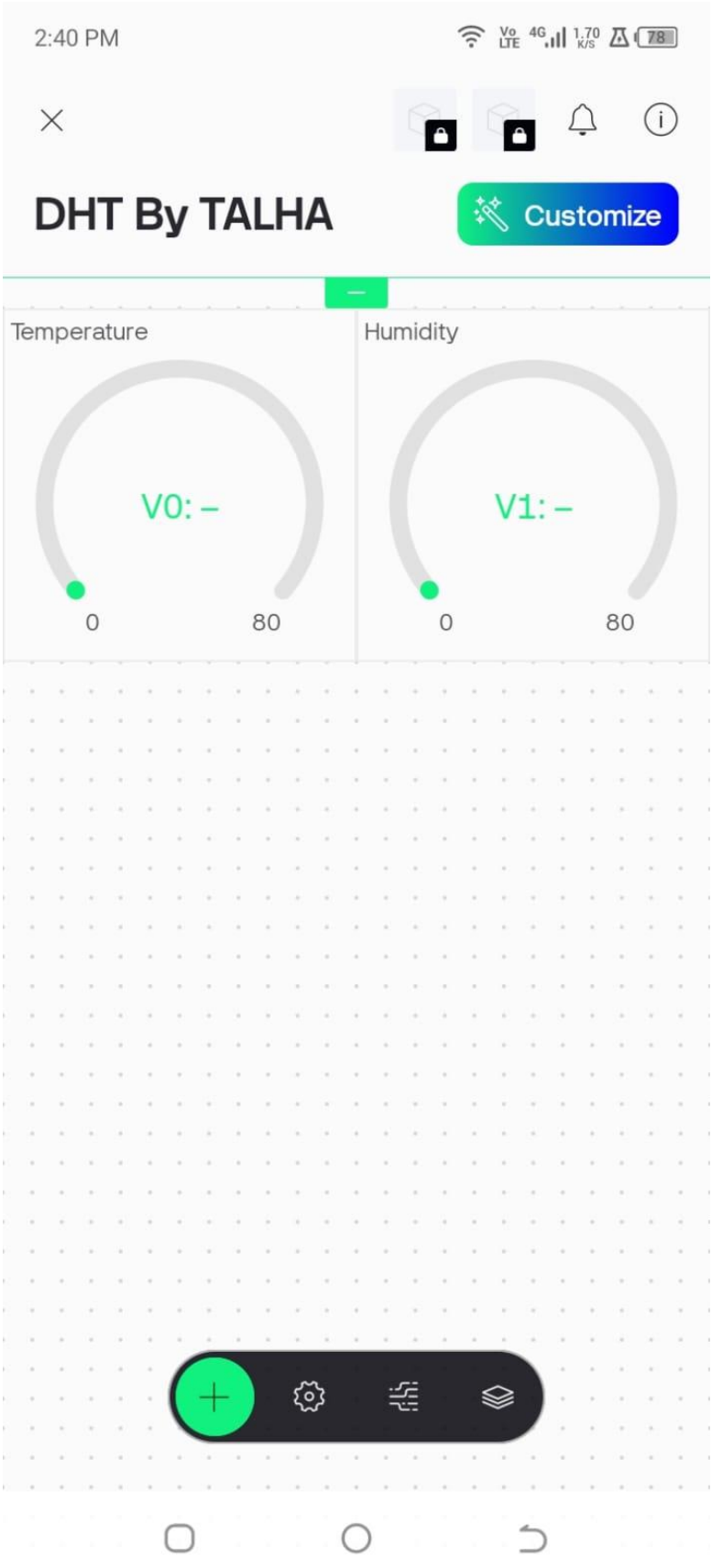
ESP32 Restarting or Freezing – Device keeps resetting.
Solution: Avoid long delay() calls; use BlynkTimer to send data periodically; ensure stable power supply (3.3V for ESP32).

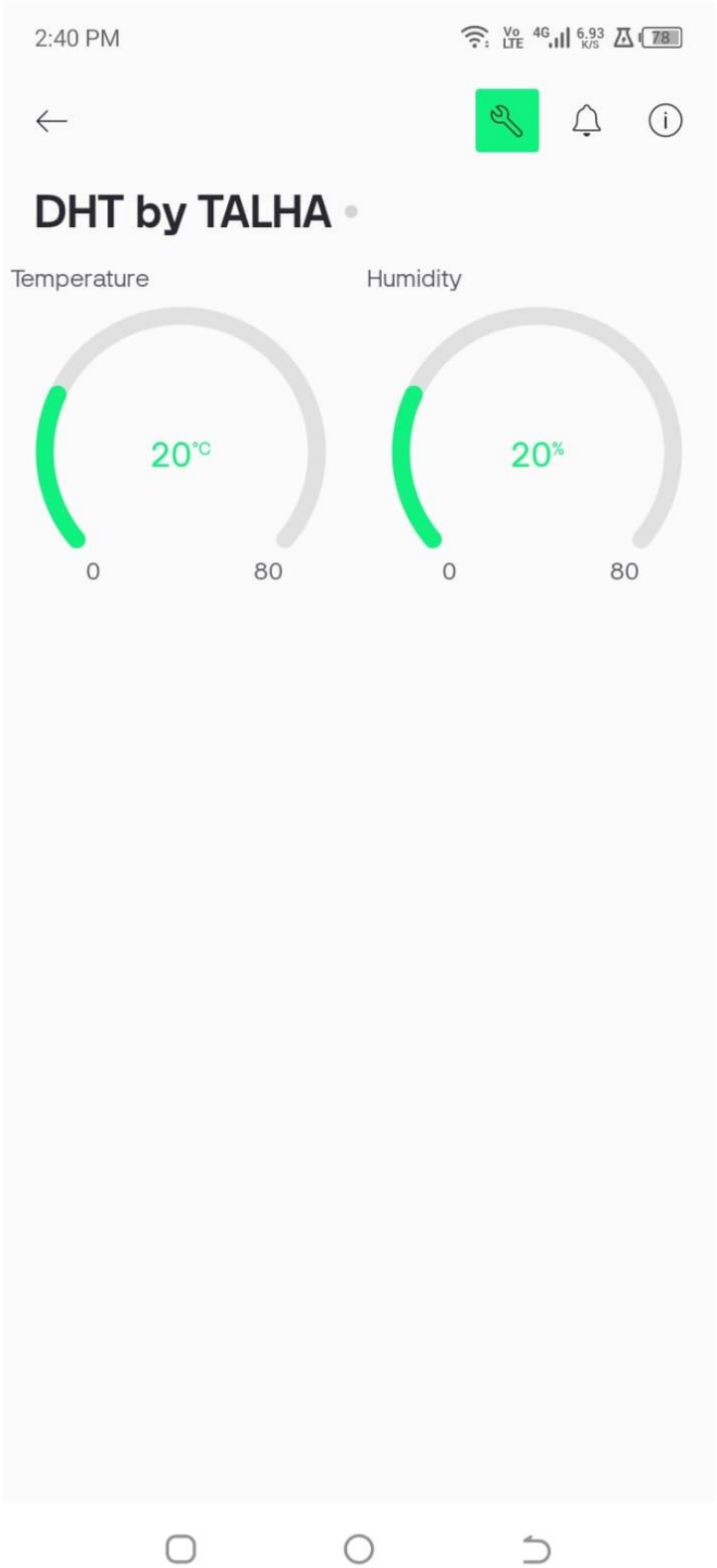
Cloud / Internet Issues – Data fails to update.
Solution: Check Wi-Fi connection and Blynk server status; make sure internet is stable.

Timing Errors – Sensor readings fail or are unstable.
Solution: Follow proper sensor library usage and respect timing requirements; do not overload code with blocking functions.

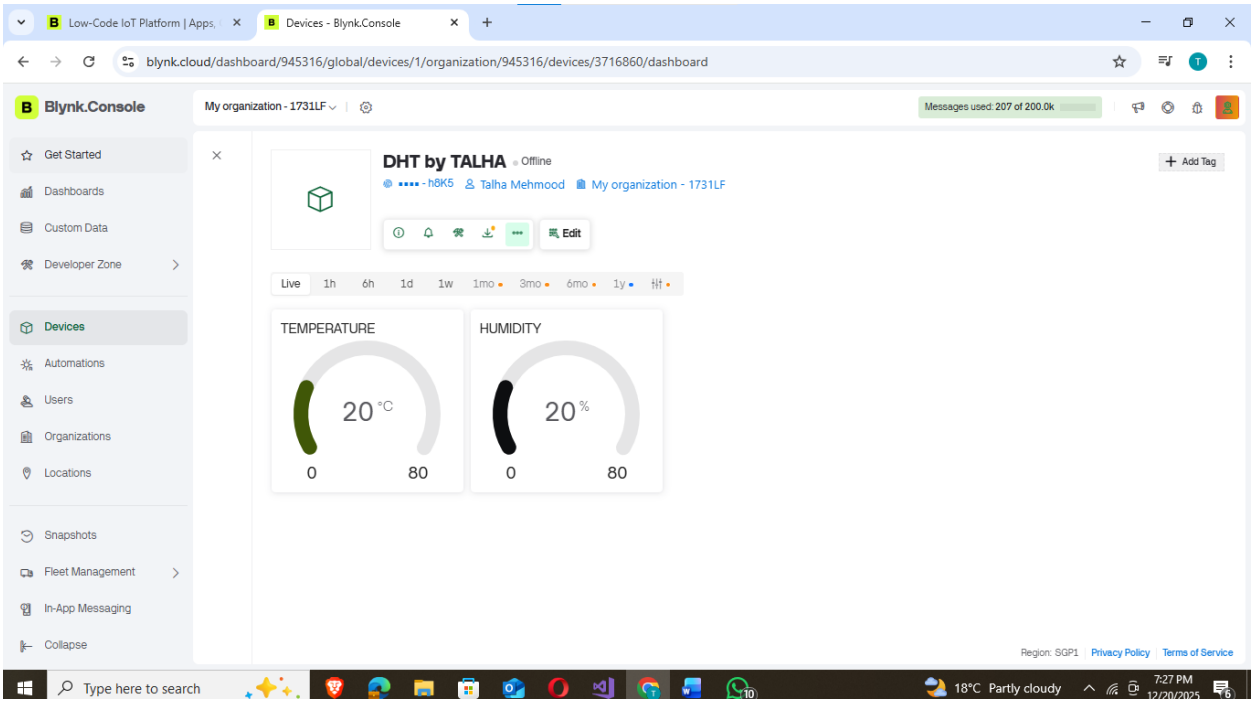
Auth Token Misuse – Device cannot send data.
Solution: Use the correct Auth Token generated for the device; do not share it with other projects.

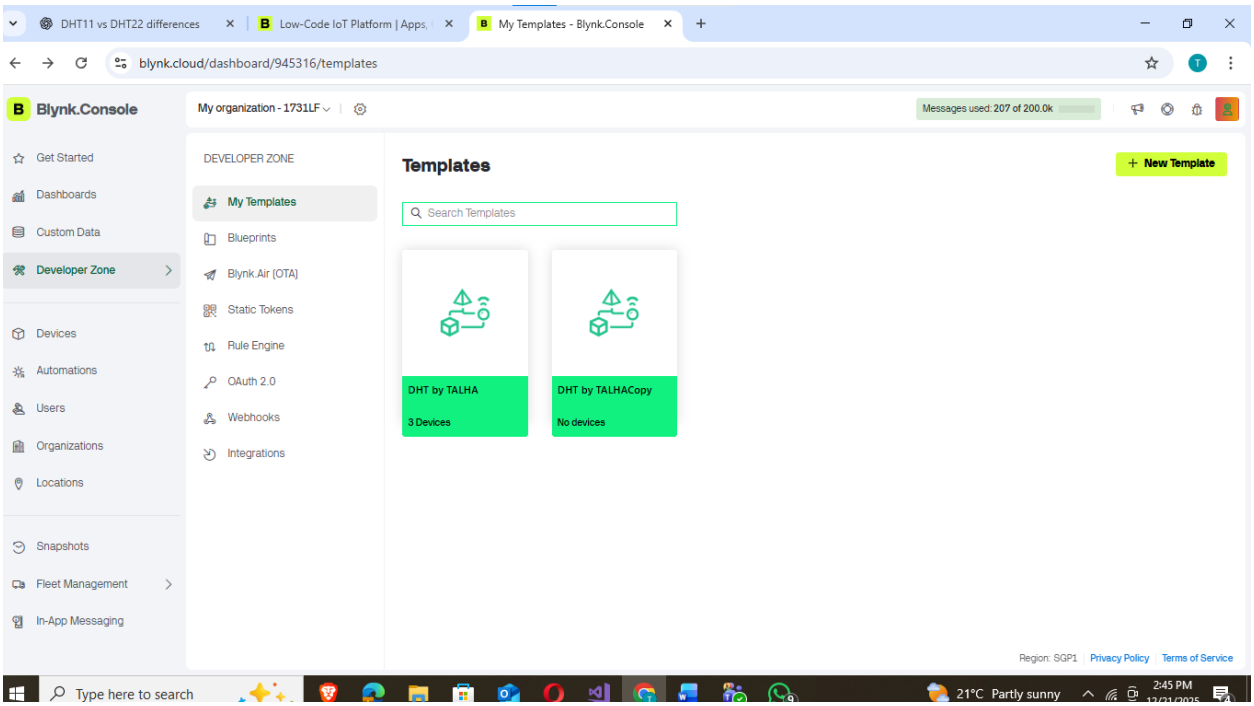
MOBILE DASHBOARD


2:40 PM
DHT By TALHA
Customize
Temperature
V0: –
0
80
Humidity
V1: –
0
80


2:40 PM
DHT by TALHA
Temperature
20°C
0
80
Humidity
20%
0
80

WEBDASHBOARD




DHT11 vs DHT22 differences
Low-Code IoT Platform | Apps
My Templates - Blynk.Console
blynk.cloud/dashboard/945316/templates
Blynk.Console
My organization - 1731LF
Messages used: 207 of 200.0k
Get Started
Dashboards
Custom Data
Developer Zone
Devices
Automations
Users
Organizations
Locations
Snapshots
Fleet Management
In-App Messaging
DEVELOPER ZONE
My Templates
Blueprints
Blynk.Air (OTA)
Static Tokens
Rule Engine
OAuth 2.0
Webhooks
Integrations
Templates
New Template
Search Templates
DHT by TALHA
3 Devices
DHT by TALHACopy
No devices
Region: SGP1
Privacy Policy
Terms of Service
Type here to search
21°C Partly sunny
2:45 PM