National Textile University, Faisalabad

Department of Computer Science

Name	Rameen Fatima
Section	BSCS-B
Semester	5th
Registration no.	23-NTU-CS-1086
Course title	Embedded IoT systems
Submitted to	Sir Nasir
Submission date	16-11-2025

Assignment # 02

Question-1

ESP32 Webserver (webserver.cpp)

Part A: Short Questions

--- (1) ---

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

WebServer server (80); creates HTTP web server object on the ESP32 and configures it to listen for incoming connections on TCP port 80.

Purpose: It instantiates the Webserver from the ESP32 that you later configure with route handlers and start with server.begin(). After that, calling server.handleClient() in the loop processes incoming HTTP requests.

What port 80 represents:

Port 80 is the default TCP port for HTTP. When a browser visits a URL without an explicit port it automatically tries port 80. Using port 80 makes your ESP32 reachable via standard HTTP without needing: <port> in the URL. In short WebServer server (80); sets up the ESP32 to act as a standard HTTP server on the default port (80), which browsers use by default when no port is specified in the URL.

--- (2) ---

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

server.on("/", handleRoot); defines what happens when someone visits the root page (/) of the ESP32 web server.

Its role is to register a route for / and tell the server which function should respond to that request.

- The first part "/ " means the root URL path.
- The second part handleRoot is the function that will run when a request comes to that path.

server.on("/", handleRoot); links the root URL of your web server to the function that handles it, so when a browser sends an HTTP request to the ESP32's homepage, the webserver automatically calls the handleRoot () to send the page content.

--- (3) ---

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

It is kept inside the loop() so the ESP32 keeps checking for new HTTP requests and serves them continuously. The loop runs repeatedly; each time, handleClient();

- Looks for a new client connection,
- Reads the http request,
- Call your registered handlers,
- And send the response.

What If it is removed:

- The web server will not process any incoming requests.
- Your route functions like handleRoot will never be called.
- From the browser side, the page will keep loading and then fail because the ESP32 isn't reading or replying to request.

--- (4) ---

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

server.send(200, "text/html", html); builds and send the HTTP response back to the browser for the current request.

1. **200**: This is the http status code. It means “ok” (request succeeded).
    i. If something goes wrong, you might send other codes like 404 (not found).
2. **"text/html":** This is the Content-Type header.
    i. It tells the browser that the response body is HTML.
3. **html:** This is the response body.
    i. It should be a string containing your html markup (e.g., “<h1>welcome</h1>”).

In short, this code line sends successful response, tell the browser its HTML and deliver the HTML stored in the variable html.

--- (5) ---

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

Displaying last measured sensor values	Taking a fresh DHT reading inside handleRoot ()
Use the values that were already read earlier, no new sensor reading is taken inside handleRoot ().	Every time someone opens the page, the ESP32 reads the sensor again before sending the response.
Advantages	

• Faster response to web requests. • Avoids blocking delays. • Good for high traffic web pages or multiple clients.	• Always shows the latest real time value.
Disadvantages	
• Data might be slightly old.	• Slower page load. • Can cause timeouts if multiple requests come quickly. • More stress on the sensor and ESP32.

Part B: Long Question

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

1) Wi-Fi connection & IP address assignment:

First, the ESP32 connects to WI-FI using the SSID and password. After successful connection, the router gives the ESP32 an Ip address. This Ip is important because it allows us to access the ESP32's webserver from a bowser.

2) Webserver Initialization and Request Handling:

A webserver is started on port 80 (default HTTP port). Routes are defined, for example "/" for home page. When someone opens the esp32's IP in a browser, the server handles the request and sends back an HTML page. The function server.handleCLient() runs in the loop so the esp32 can continuously process incoming requests.

3) Button based sensor reading and OLED update:

A push button is connected to a GPIO pin with INPUT_PULLUP. When pressed, it triggers a fresh reading from the DHT sensor. The new temperature and humidity values are shown on the OLED display and also stored for web display. This makes the system interactive because the user can manually update readings.

4) Dynamic HTML webpage:

The ESP32 creates an HTML page that shows the latest temperature and humidity values. These values are inserted dynamically into the page before sending it to the browser. The page can also have a button or link to trigger a manual sensor read.

5) Purpose of Meta refresh:

The HTML page uses a meta refresh. This automatically reloads the page every 5 sec, so the user always sees updated sensor values without pressing refresh. It ’s a simple way to keep the data current.

6) Common issue and solutions:

Wi-Fi not connecting: Use correct SSID and password and 2.4 GHz network.

Slow or unresponsive page: Avoid delay (), use non-blocking timers.

DHT sensor shows NaN: Keep at least 2 sec between readings and check wiring.

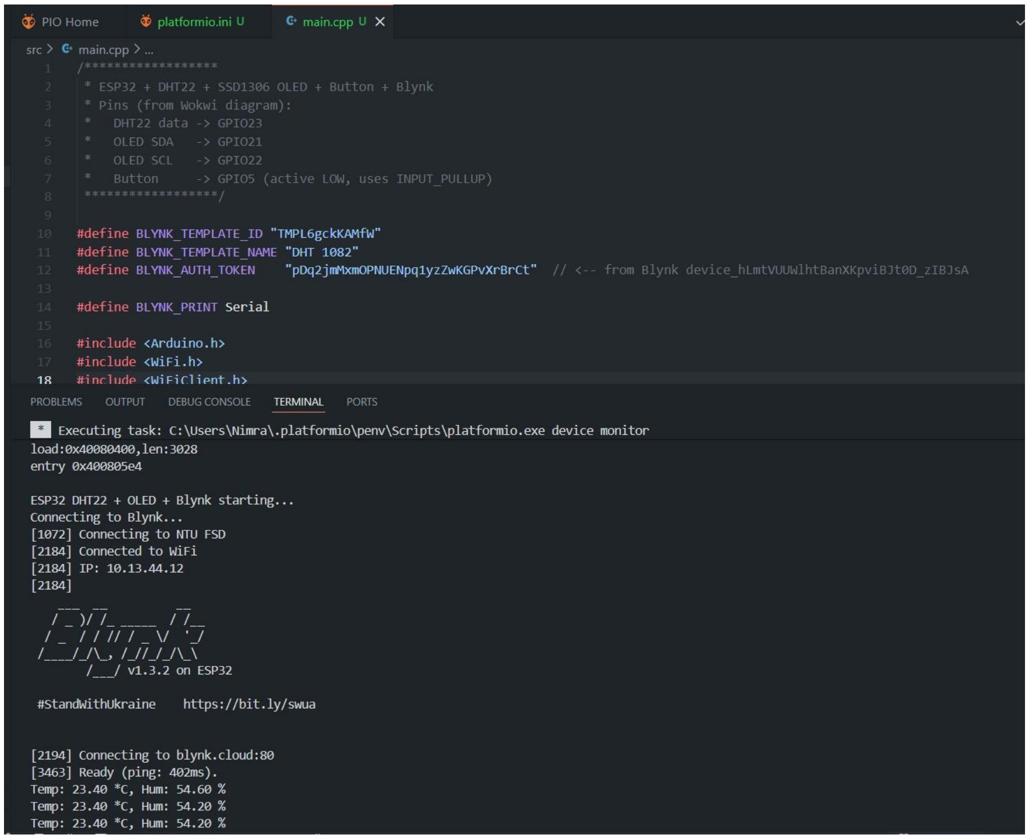
OLED not working: Verify I^{2}C pins and address.

Button misbehaves: Use INPUT_PULLUP and debounce the button.

ESP32 resets: Do not block the loop keep it responsive.

In conclusion, the ESP32 connects to Wi-Fi, starts a webserver and serves dynamic page showing temperature and humidity. A button lets the user take fresh readings, which are displayed on the OLED and updated on the webpage. Meta refresh keeps the page updated automatically. Common problems like Wi-Fi issues, sensor errors etc are solved by proper configuration and non-blocking design.

Serial Monitor Output:



```cpp
/*****************
 * ESP32 + DHT22 + SSD1306 OLED + Button + Blynk
 * Pins (from Wokwi diagram):
 *   DHT22 data -> GPIO23
 *   OLED SDA   -> GPIO21
 *   OLED SCL   -> GPIO22
 *   Button     -> GPIO5 (active LOW, uses INPUT_PULLUP)
 *****************/

#define BLYNK_TEMPLATE_ID "TMPL6gckKAMfW"
#define BLYNK_TEMPLATE_NAME "DHT 1082"
#define BLYNK_AUTH_TOKEN    "pDq2jmMxmOPNUENpq1yzZwKGPvXrBrCt"  // <-- from Blynk device_hLmtVUUWlhtBanXKpviBJt0D_zIBJsA

#define BLYNK_PRINT Serial

#include <Arduino.h>
#include <WiFi.h>
#include <WiFiClient.h>
```

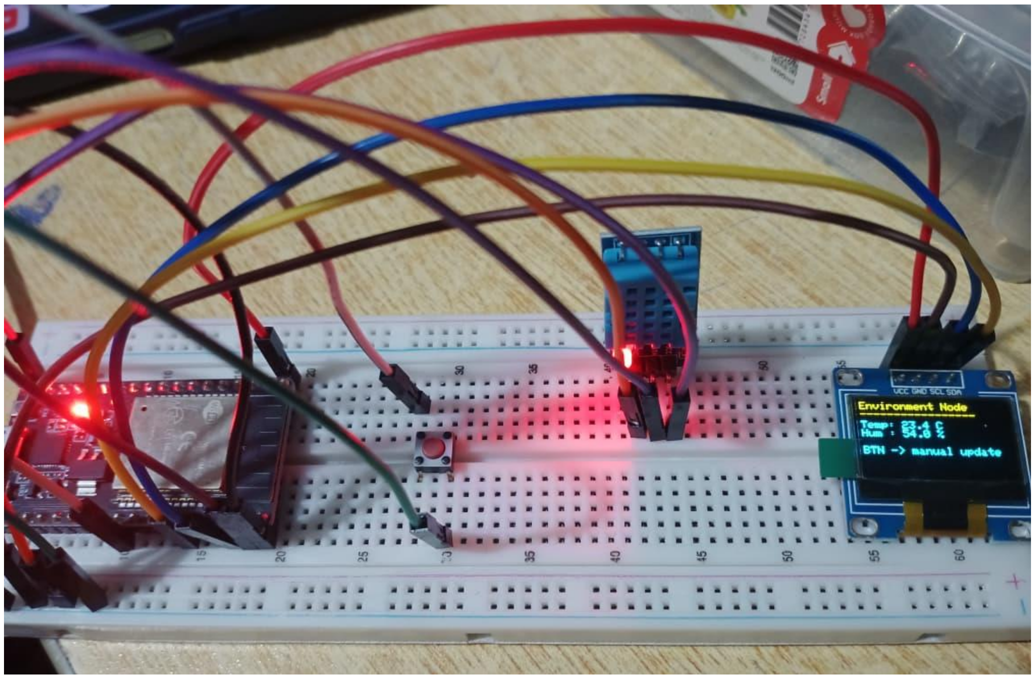
```
 *  Executing task: C:\Users\Nimra\.platformio\penv\Scripts\platformio.exe device monitor
load:0x40080400,len:3028
entry 0x400805e4

ESP32 DHT22 + OLED + Blynk starting...
Connecting to Blynk...
[1072] Connecting to NTU FSD
[2184] Connected to WiFi
[2184] IP: 10.13.44.12
[2184]
    ___  __          __
   / _ )/ /_ _____  / /__
  / _  / / // / _ \/  '_/
 /____/_/\_, /_//_/_/\_\
        /___/ v1.3.2 on ESP32

 #StandWithUkraine    https://bit.ly/swua


[2194] Connecting to blynk.cloud:80
[3463] Ready (ping: 402ms).
Temp: 23.40 *C, Hum: 54.60 %
Temp: 23.40 *C, Hum: 54.20 %
Temp: 23.40 *C, Hum: 54.20 %
```

OLED Display:



Question-2 Blynk Cloud Interfacing (blynk.cpp)

Part-A: Short Questions

--- (1) ---

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 is a unique identifier that links your ESP32 device to a specific template you created in the blynk cloud.

Role in an ESP32 IoT Project:

- It tells the blynk platform which dashboard layout, data streams (virtual pins), and settings belong to your device.
- When your ESP32 connects to blynk cloud, the template id ensures the device uses the correct configuration for widgets, pins and logic defined in that template.

Why must it match the cloud template:

- If the template id in code does not match the one in blynk cloud:
    - The device cannot associate with the correct dashboard.
    - Data streams and widgets will not work properly.
    - The blynk app or web dashboard will show “device not in template” or fail to display data.
- Matching ensures seamless communication between hardware and the cloud dashboard. If it does not match, our IoT project won’t sync with intended interface.

--- (2) ---

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

Blynk Template ID	Blynk Auth Token
Purpose	
Identifies the template in Blynk Cloud that defines the dashboard layout, widgets, and data streams.	A unique key for each device to authenticate and securely connect to blynk cloud.
Scope	
• Defines what the device should display. • Common for all devices using the same template.	• Confirms which device is allowed to connect. • Specific to one device.
Function	
Links your device to the correct UI and configuration in the cloud.	Ensures that only authorized devices can send/receive data for that account.
Example	
#define BLYNK_TEMPLATE_ID "TMPL12345etc"	#define BLYNK_AUTH_TOKEN "abcd1234efgh567"

--- (3) ---

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 requirements. The library or code expects a specific protocol for reading temperature and humidity, and if the wrong sensor type is configured, the data bits are interpreted incorrectly.

One Key Difference:

- **DHT11:**
    - Temperature range: **0°C to 50°C**, accuracy ±2°C

    - Humidity range: **20%–90%**, accuracy ±5%
    - **Resolution:** DHT11 sends data with lower resolution, Integer only (no decimals).

- **DHT22:**
    - Temperature range: **-40°C to 80°C**, accuracy ±0.5°C
    - Humidity range: **0%–100%**, accuracy ±2–5%
    - **Resolution:** DHT22 sends data with higher resolution, Decimal values (higher precision)

If we use DHT22 code for a DHT11 sensor, the extra bits expected by the code will not exist, causing garbage values or NaN.

--- (4) ---

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

Virtual pins:

These are software defined channels in the Blynk platform that allow us to send and receive data between our ESP32 and the Blynk Cloud or mobile app. They are not tied to any physical GPIO pin on the ESP32. Instead, they act as logical identifiers for data streams.

Why are they preferred over physical GPIO pins:

- We can map any sensor or variable to a virtual pin without worrying about hardware limitations.
- Virtual pins are designed for data exchange with the Blynk Cloud, making it easy to update dashboards, graphs, and widgets.
- They avoid direct hardware control over the internet, which can be slow or unsafe.

- Multiple devices and dashboards can share virtual pins without changing physical wiring.
- Reduces risk of exposing direct GPIO control to the internet.

--- (5) ---

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

The purpose of using BlynkTimer instead of delay () in ESP32 IoT applications is to keep the device responsive and connected to the blynk Cloud while performing periodic tasks.

Why not use delay ():

- Delay(ms) blocks the CPU for the specified time.
- During this blocking period:
  - The ESP32 cannot process Blynk commands.
  - Cloud connection may time out.
  - Web server or other tasks stop responding.
- This leads to disconnection, missed updates, and poor performance.

Why use BlynkTimer:

- Executes functions at regular intervals without freezing the main loop.
- Allows the ESP32 to:
  - Maintain Wi-Fi and Blynk connectivity.
  - Handle multiple tasks (sensor reading, UI updates, etc.) smoothly.
- Works like a lightweight task scheduler.

Part B: Long Question

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

1) Creation of Blynk Template and DataStream:

In the Blynk cloud dashboard, firstly create a new template for project. Then define DataStream for the values we want to send, such as:

- V0 for temperature
- V1 for humidity

These virtual pins act as logical channels for data exchange between ESP32 and the Blynk app/dashboard.

2) Role of template ID, Template name and Auth token:

Template ID:

Uniquely identifies the template in blynk cloud. It ensures that ESP32 uses the correct dashboard layout and datastreams.

Template name:

A human readable name for your template. It is for organization and display purposes.

Auth token:

A unique key for each device under the template. It authenticates the ESP32 with Blynk cloud and ensures secure communication.

If any of these do not match the cloud configuration, the device will fail to connect or display data correctly.

3) Sensor configuration issues(DHT11 vs DHT22):

Problem:

Using DHT22 code with a DHT11 sensor causes incorrect readings because:

DHT11 sends integer values only, while DHT22 sends decimal values with a different data format.

Solution:

Always set the correct sensor type in code.

Also ensure proper wiring and respect the minimum interval between readings.

4) **Sending data using Blynk.virtualwrite():**

After reading temperature and humidity from the sensor

- Use Blynk.virtualwrite (V0, temperature);
- Use Blynk.virtualwrite (V1, humidity);

This sends the values to the corresponding virtual pins in the blynk cloud, updating the dashboard widgets in real time.

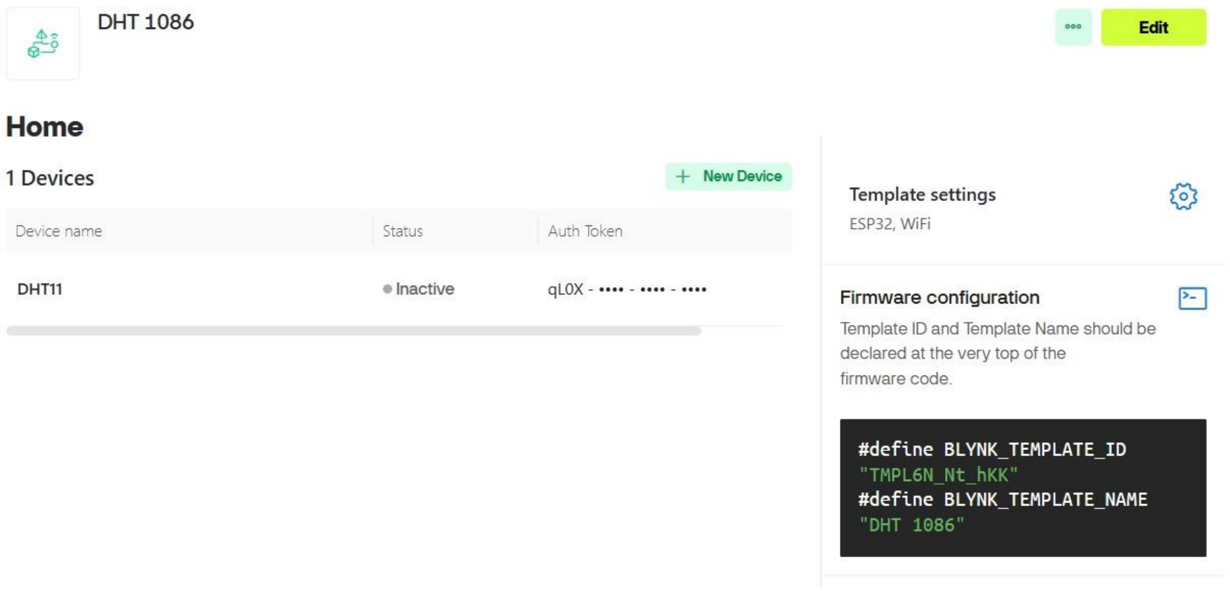
5) **Common problems faced during configuration and their solutions:**

Problem	Cause	Solution
Device not connecting	Wrong auth token or WiFi credentials	Verify token from blynk dashboard and SSID/password.
Wrong values or NaN	Incorrect sensor type or timing	Use correct DHT type and allow 2s between reads.
Dashboard not updating	Wrong virtual pin mapping	Match widget pins with code (V0 for temp, V1 for humidity)
Frequent disconnections	Using delay () in loop	Replace delays with BlynkTimer for non-blocking tasks.
App shows “device not in template”	Template ID mismatch	Ensure Template ID in code matches cloud template.

Conclusion:

The ESP32 connects to Wi-Fi and authenticates with Blynk cloud using the Auth Token. It reads temperature and humidity from the DHT sensor, then send these values to the cloud using virtual pins. The Blynk dashboard displays the data in real time. Correct template setup, proper sensor configuration, and non-blocking code are essential for smooth operation.

Blynk template:



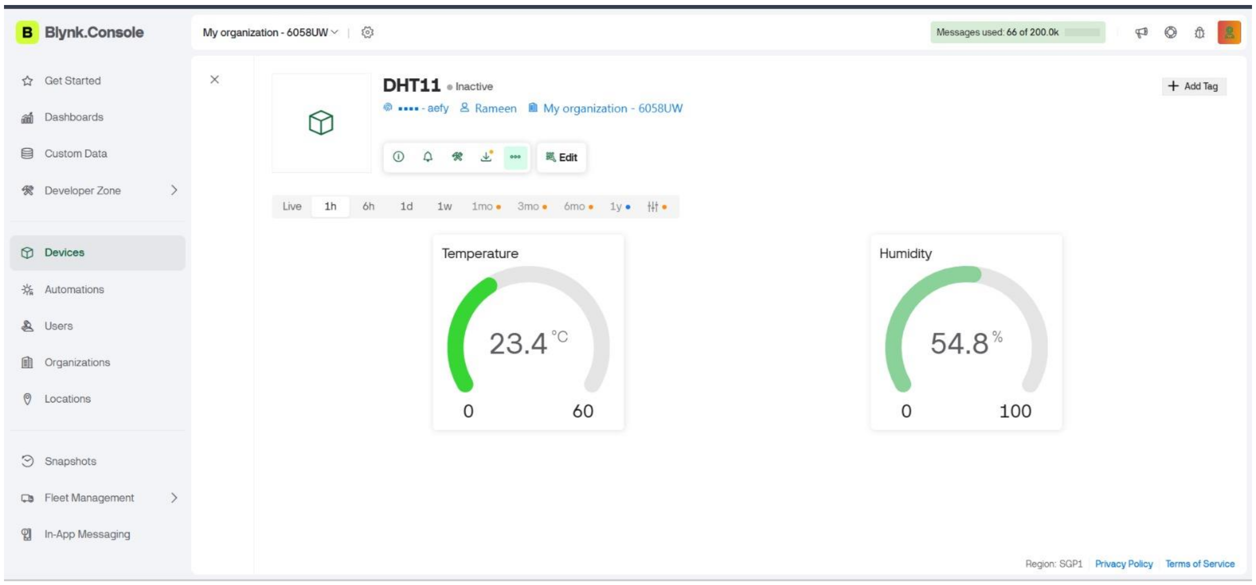
1 Devices

Device name	Status	Auth Token
DHT11	● Inactive	qL0X - •••• - •••• - ••••

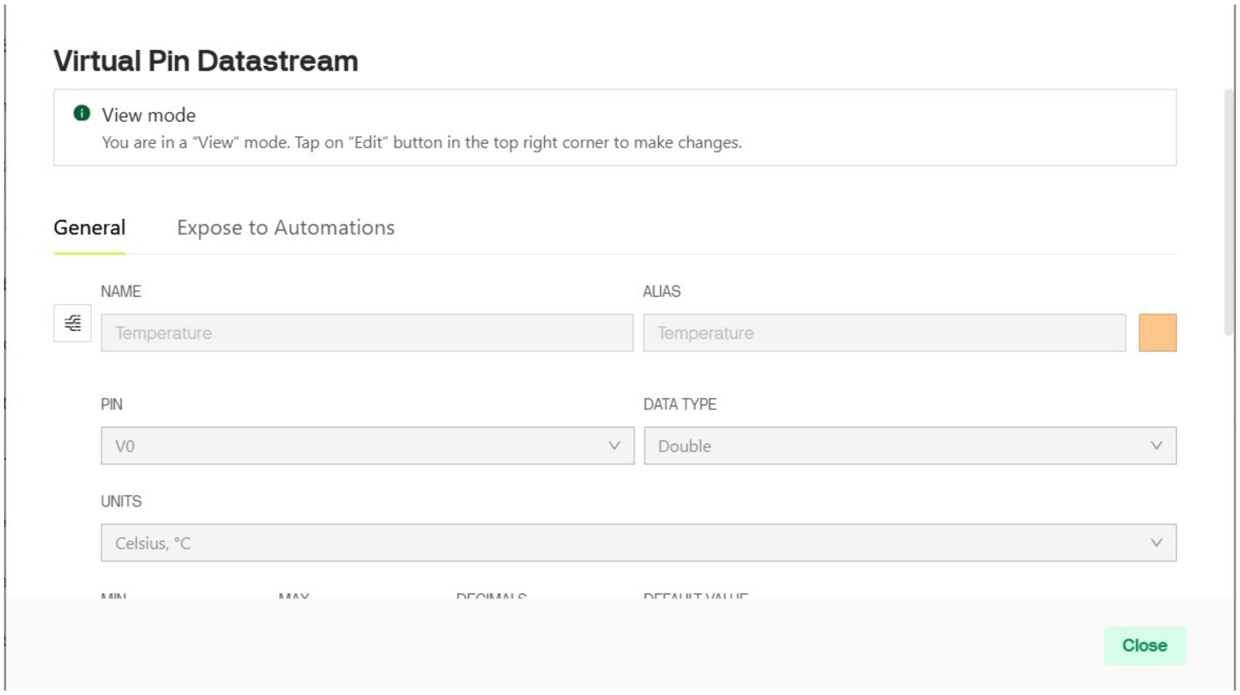
Template ID and Template Name should be declared at the very top of the firmware code.

```
#define BLYNK_TEMPLATE_ID
"TMPL6N_Nt_hKK"
#define BLYNK_TEMPLATE_NAME
"DHT 1086"
```

Web dashboard:



DataStream for temperature:



Virtual Pin Datastream

View mode

You are in a "View" mode. Tap on "Edit" button in the top right corner to make changes.

General | Expose to Automations

NAME: Temperature

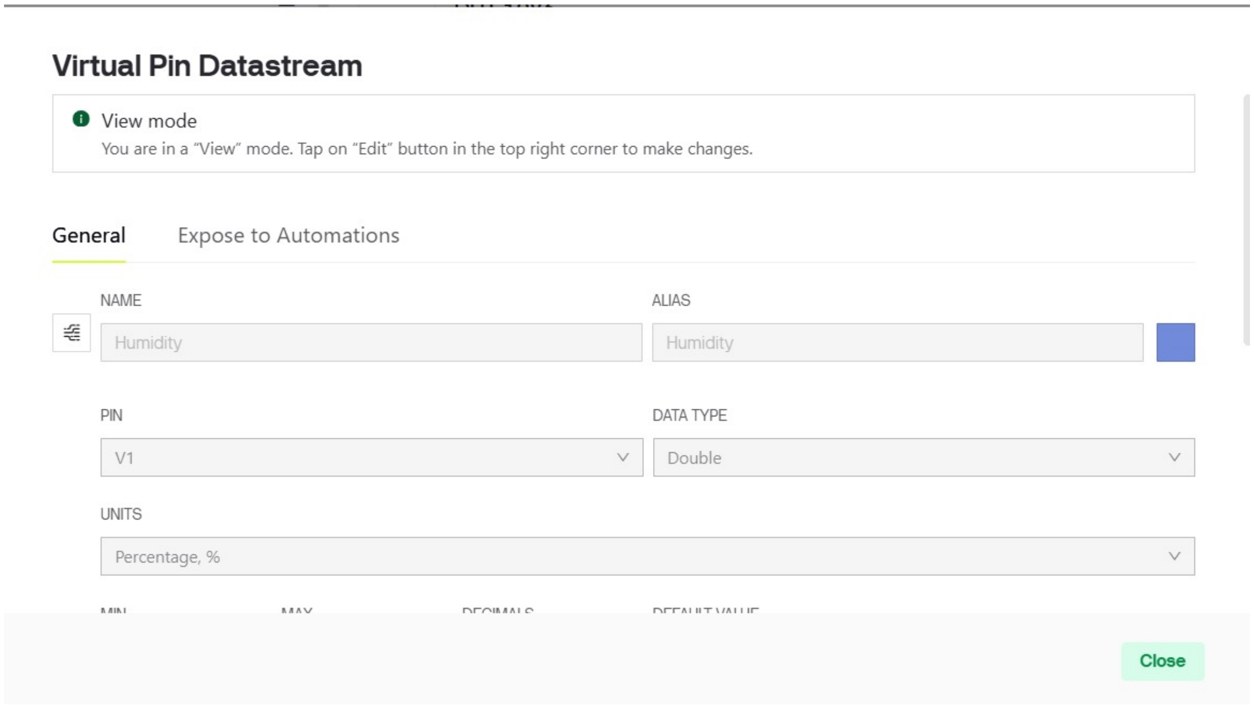
ALIAS: Temperature

PIN: V0

DATA TYPE: Double

UNITS: Celsius, °C

DataStream for humidity:



Virtual Pin Datastream

View mode

You are in a "View" mode. Tap on "Edit" button in the top right corner to make changes.

General | Expose to Automations

NAME: Humidity

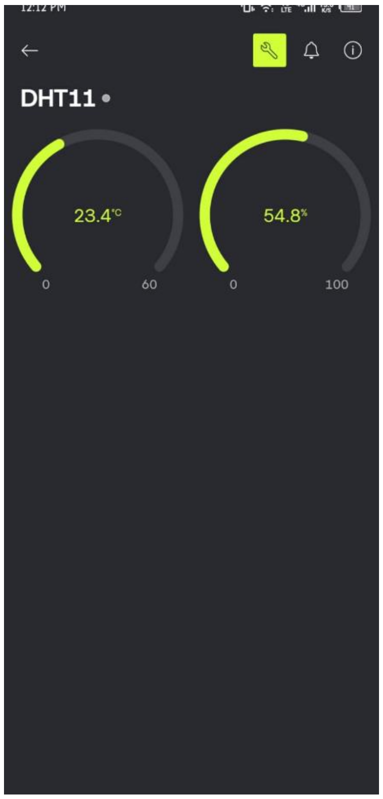
ALIAS: Humidity

PIN: V1

DATA TYPE: Double

UNITS: Percentage, %

Blynk Mobile App:



Github link:

https://github.com/Rameen-Fatima67/Embedded-iot--23-NTU-CS-1086-Rameen