# National Textile University, Faisalabad


## Department of Computer Science

| | |
|---|---|
| **Name:** | **Fiza Mahboob** |
| **Class:** | **BSCS(A)** |
| **Semester:** | **5th** |
| **Registration No:** | **23-NTU-CS-1027** |
| **Course Name:** | **EMBEDDED IOT SYSTEM** |
| **Submitted To:** | **Sir Nasir Mahmood** |
| **Submisssion Date:** | **21 Dec,2025** |

# Homework-01 – After mid

## 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?

WebServer server(80); creates a web server object on the ESP32.
This allows the ESP32 to listen for HTTP requests from a web browser.

Port 80 is the default port for HTTP communication.
When we type the ESP32 IP address in a browser (for example 192.168.4.1), the browser automatically sends the request to port 80.
So, this line enables the ESP32 to act like a small web server that can display a webpage.

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

This statement tells the web server what to do when a client accesses the root URL (/) of the ESP32.

- / means the home page of the web server.
- handleRoot is a function that generates and sends the HTML webpage.

Whenever a user opens the ESP32 IP address in a browser, the function handleRoot() is automatically executed and the webpage with temperature and humidity data is displayed.

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

server.handleClient(); continuously checks if a web client (browser) has sent a request to the ESP32.

It is placed inside loop() because:

- loop() runs repeatedly.

- The ESP32 must always be ready to respond to new HTTP requests.

If this line is removed, the web server will stop responding.
Even though the ESP32 is connected to Wi-Fi, the webpage will not open or update because incoming requests are never processed.

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

This statement sends a complete HTTP response to the client's web browser:

- 200 → HTTP status code meaning “OK / Successful request”
- "text/html": Content-Type header telling the browser that the response contains HTML content
- html: The actual HTML string containing the webpage with temperature/humidity data

So, this statement delivers the dynamically created HTML page to the user’s browser.

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

- **Last measured values:**
  The webpage displays the most recent readings stored in lastTemp and lastHum. These values are updated only when the physical button is pressed.

- **Fresh DHT reading inside handleRoot():**
  If readDHTValues() is called inside handleRoot(), the sensor is read every time the webpage refreshes.

**Difference:**
Using last measured values avoids frequent sensor reads and keeps readings synchronized with the OLED display, while fresh readings on every request may overload the DHT sensor and cause unreliable results.

## Part B: Long Question

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

### System Overview:

This ESP32-based system creates a wireless temperature and humidity monitoring station that displays readings both locally on an OLED screen and remotely via a web interface. The system combines sensor reading, display output, and network communication into an integrated IoT solution.

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

**Connection Process:**

1. The ESP32 starts by loading the Wi-Fi library and credentials (ssid and password)
2. WiFi.begin(ssid, password) initiates connection to the specified network
3. The program enters a while loop that continuously checks WiFi.status() until it returns WL_CONNECTED
4. During connection, dots display on Serial Monitor, and status updates show on OLED
5. Once connected, the ESP32 obtains an IP address from the router via DHCP

**IP Address Assignment:**

- The router's DHCP server automatically assigns an available IP address
- This address is retrieved using WiFi.localIP() and displayed on Serial Monitor and OLED
- Users enter this IP address in any web browser to access the monitoring interface

## 2. Web Server Initialization and Request Handling

**Server Initialization:**

1. WebServer server(80) creates a web server instance on port 80
2. server.on("/", handleRoot) maps the root URL to our handler function
3. server.begin() activates the server to accept connections

Request Handling Flow:

1. User enters ESP32's IP address in browser
2. server.handleClient() in loop() catches the incoming request

3. Server checks if request is for root URL ("/")
4. handleRoot() function generates HTML response
5. server.send() transmits the webpage to browser
6. The server keeps listening for more requests

## 3. Button-Based Sensor Reading and OLED Update Mechanism

**Button Reading Logic:**

- Button pin uses INPUT_PULLUP mode (internal pull-up resistor)
- System detects falling edge (HIGH→LOW transition) for button press
- 50ms delay checks if button remains LOW, eliminating false triggers
- lastButtonState variable stores previous state for comparison

**Update Sequence When Button Pressed:**

1. Button press detected via state change
2. readDHTValues() reads temperature and humidity from DHT22 sensor
3. Valid readings update lastTemp and lastHum global variables
4. showOnOLED() displays updated values on the screen
5. Webpage automatically shows new values on next refresh/visit

## 4. Dynamic HTML Webpage Generation

HTML Construction Process:
The handleRoot() function builds the webpage as a single string:

1. **HTML Structure:** Starts with <!DOCTYPE html> and basic page structure
2. **Dynamic Content:** Conditionally includes:
    - Error message if no valid data exists
    - Current temperature and humidity values when available
3. **Data Formatting:** Converts float values to strings with one decimal place

4. **Visual Elements**: Includes headings, paragraphs, and horizontal rules for readability
5. **User Instructions:** Explains how to update readings via physical button

Key Feature: The HTML is generated on-demand each time the page loads, ensuring the latest lastTemp and lastHum values display.

## 5. Purpose of Meta Refresh in Webpage

The line <meta http-equiv='refresh' content='5'> implements automatic page refreshing:

- content='5' refreshes page every 5 seconds
- Users see updated sensor readings without manually reloading
- Provides near real-time monitoring experience
- Only updates display; sensor reading still requires button press

This creates a semi-live dashboard where new button-press readings automatically appear in the browser within 5 seconds.

## 6. Common Issues in ESP32 Webserver Projects and Their Solutions

| **Issue** | **Solution** |
|---|---|
| Webpage not loading | Ensure server.handleClient() is in loop() |
| Wrong IP address | Check Serial Monitor for correct IP |
| DHT sensor errors | Avoid frequent readings, add delays |
| OLED not displaying | Check I²C address and wiring |
| Wi-Fi not connecting | Verify SSID and password |

## Conclusion:

This ESP32 webserver system successfully integrates Wi-Fi, a DHT22 sensor, OLED display, and a web interface.
It provides efficient sensor reading using a physical button, reliable web access through

HTTP, and clear data visualization on both hardware and browser, making it a practical and well-structured IoT monitoring project.

# 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 uniquely identifies a specific project template created on the Blynk Cloud.
It links the ESP32 firmware with the correct dashboard layout, widgets, and datastreams.

It must match the cloud template because:

- The Blynk server uses this ID to verify which project the device belongs to.
- If the Template ID does not match, the ESP32 will connect to Wi-Fi but will not sync data with the Blynk dashboard**.**

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

| Feature | Template ID | Auth Token |
|---|---|---|
| What it is | Blueprint for device type | Unique password for each device |
| Purpose | Identifies the project template | Authenticates a specific device |
| Scope | Same for all devices using the template | Unique for each device |
| Cloud Role | Links firmware to dashboard | Grants permission to send/receive data |
| Security | Project-level | Device-level security key |
| Changes | Rarely changes | Generate new if compromised |

In short, **Template ID defines "what project"**, while **Auth Token defines "which device"**.

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

**Why incorrect readings occur:**
DHT22 code with DHT11 sensor produces wrong values because different libraries interpret sensor data differently. The DHT22 sensor returns 16-bit values with different scaling, while DHT11 uses 8-bit values. Using DHT22 code assumes higher precision and different data processing than DHT11 provides.

**Key difference:**

- DHT11: Lower precision (±2°C accuracy), 8-bit data, limited range (20-90% humidity)
- DHT22: Higher precision (±0.5°C accuracy), 16-bit data, wider range (0-100% humidity)

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

**Virtual Pins** are **software-defined channels** that act as communication bridges between your ESP32 and Blynk Cloud. Unlike physical pins connected to actual hardware, virtual pins exist only in code and cloud configuration.

**They are preferred because:**

- They are not tied to actual hardware pins
- They allow flexible data mapping (sensor values, buttons, graphs)
- They work reliably over the internet

**In the code:**

- V0 → Temperature
- V1 → Humidity

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

BlynkTimer allows tasks to run periodically without blocking the program.

**Using delay():**

- Freezes the ESP32
- Breaks Blynk communication
- Causes disconnection issues

**BlynkTimer:**

- Keeps Blynk cloud connection alive
- Allows multitasking (button, OLED, cloud updates)
- Is essential for real-time IoT applications

## Part-B: Long Question

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

## System Overview

This system creates an IoT monitoring solution where ESP32 reads environmental data from a DHT sensor, displays it locally on an OLED screen, and transmits it to Blynk Cloud for remote monitoring via smartphone app. The system supports both manual (button-press) and automatic (timed) data transmission.

### 1. Creation of Blynk Template and Datastreams

**Template Creation Process:**

1. **Login to Blynk Console**: Access developers.blynk.io with your credentials
2. **New Template**: Click "New Template" and provide basic information
3. **Define Hardware**: Select ESP32 as the microcontroller platform
4. **Set Connection Type:** Choose Wi-Fi for wireless communication

**Datastreams Configuration:**

1. **Add Datastream**: Click "Add Datastream" for each data type
2. **Temperature Datastream:**

- Name: "Temperature"
- Pin: Virtual Pin V0
- Data Type: Double (for decimal values)
- Units: °C or °F
- Min/Max Range: Set appropriate limits (-40 to 80°C for DHT22)

3. **Humidity Datastream:**
    - Name: "Humidity"
    - Pin: Virtual Pin V1
    - Data Type: Double
    - Units: %
    - Range: 0-100%

Template Saving: After configuration, the template is saved and assigned a Template ID that must be copied into your ESP32 code.

## 2. Role of Template ID, Template Name, and Auth Token

**Template ID (e.g., "TMPL6UCKQ_P6D"):**

- Primary Purpose: Links code to specific cloud template configuration
- When ESP32 connects, it tells Blynk Cloud: "I'm using template configuration TMPL6UCKQ_P6D"
- Cloud knows what virtual pins to expect and how to process data
- Defined at top: #define BLYNK_TEMPLATE_ID "TMPL6UCKQ_P6D"

**Template Name (e.g., "dht"):**

- Primary Purpose: Human-readable identifier for developers
- Helps organize multiple templates in Blynk Console
- Appears in dashboard lists, helps differentiate between project types

- #define BLYNK_TEMPLATE_NAME "dht"

**Auth Token (e.g., "S2P_gLWBb5E-DINBtsXmlOcndOq6xJnU"):**

- Primary Purpose: Unique authentication key per physical device
- Like a password that proves "This specific ESP32 is authorized"
- Automatically created when you add a device to template
- Must be kept secret - if exposed, others can send data to your account
- Location in Code: #define BLYNK_AUTH_TOKEN "S2P_gLWBb5E-DINBtsXmlOcndOq6xJnU"

## 3.Sensor Configuration Issues (DHT11 vs DHT22)

| **Aspect** | **DHT11** | **DHT22** |
|---|---|---|
| **Code Definition:** | #define DHTTYPE DHT11 | #define DHTTYPE DHT22 |
| **Library Usage:** | Same DHT library, different type constant | Same library, different constant |
| **Data Precision:** | Integers only | Decimal values |
| **Reading Range:** | More limited | Wider range |
| **Timing:** | Different reading intervals | 2-second minimum between reads |

**Common Configuration Issues:**

1. **Incorrect Type Definition:** Using DHT22 in code with DHT11 hardware
    - **Symptom:** Readings are wildly incorrect or show decimals where none exist
    - **Solution:** Ensure DHTTYPE matches physical sensor
2. **Library Compatibility Issues:**
    - **Issue:** Some libraries handle DHT11/DHT22 differently
    - **Solution:** Use standard DHT.h library from Adafruit

3. **Reading Interval Problems:**
    - **Issue:** Reading too frequently (both sensors need 2+ seconds)
    - **Solution:** Implement minimum delay between reads (as done with button debounce)

## 4. Sending Data Using Blynk.virtualWrite()

After **reading the sensor:**

Blynk.virtualWrite(V0, t);

Blynk.virtualWrite(V1, h);

**This sends:**

- Temperature to V0
- Humidity to V1

Blynk Cloud then updates the dashboard widgets in real time.

## 5. Common Problems and Solutions

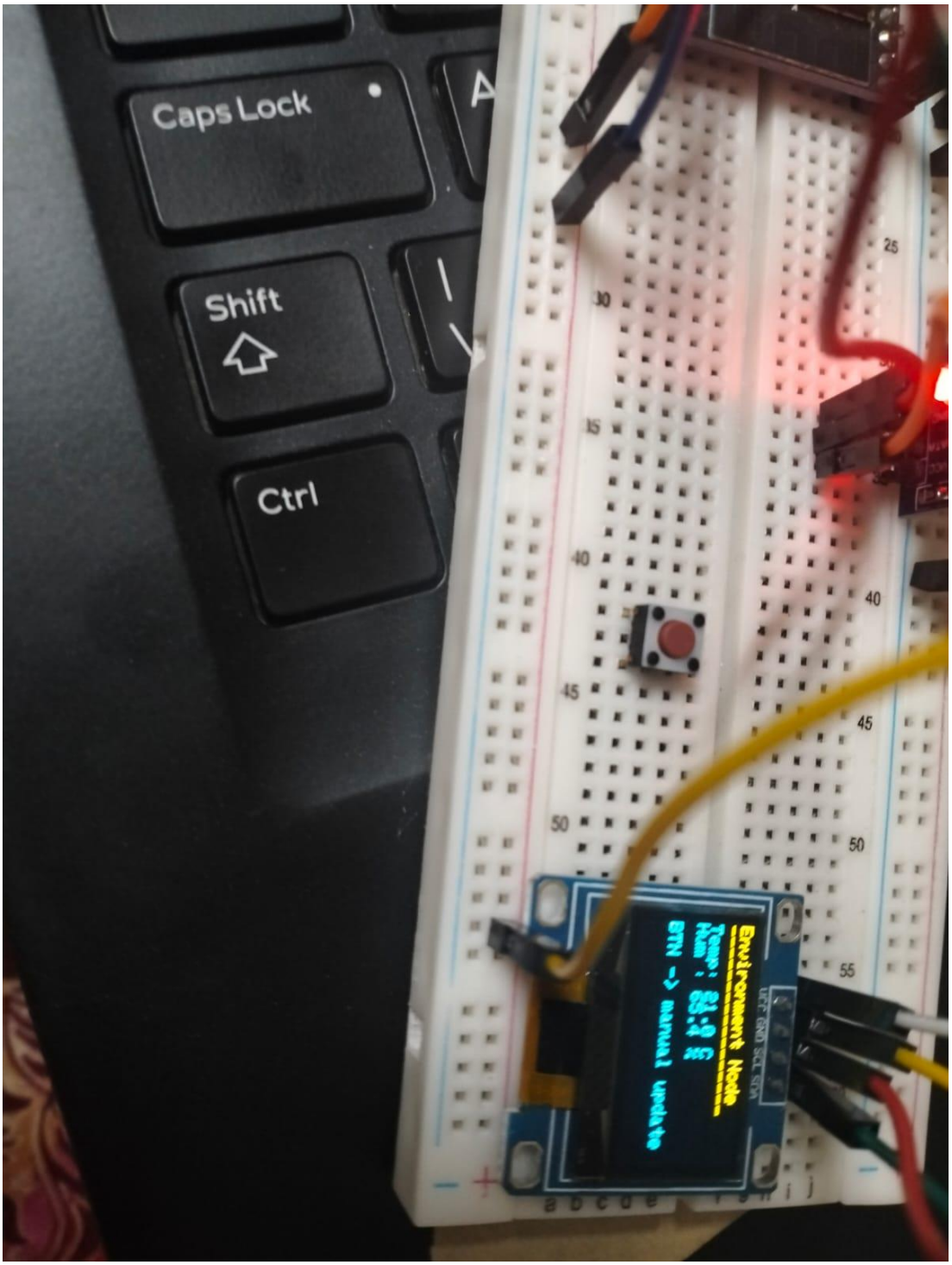
| **Problem** | **Symptoms** | **Solution** |
|---|---|---|
| **Connection Failures** | ESP32 doesn't connect to Blynk | 1. Verify WiFi credentials<br>2. Check Blynk server status<br>3. Ensure correct Auth Token |
| **Incorrect Template ID** | Data appears wrong or doesn't update | 1. Copy Template ID from Blynk Console<br>2. Ensure no typos in #define |
| **Auth Token Issues** | "Invalid Auth Token" error | 1. Generate new token in Blynk Console<br>2. Update code with new token |
| **Virtual Pin Mismatch** | Data goes to wrong widget | 1. Check pin mapping in template<br>2. Verify Blynk.virtualWrite() uses correct pins |

| | | |
|---|---|---|
| **Data Not Updating in App** | App shows old values | 1. Check ESP32 serial monitor for transmission confirmation<br>2. Verify Blynk.run() in loop()<br>3. Check internet connection |
| **Button Multiple Triggers** | Single press sends multiple updates | 1. Implement debouncing (as done with 50ms check)<br>2. Use interrupt with debounce capacitor |

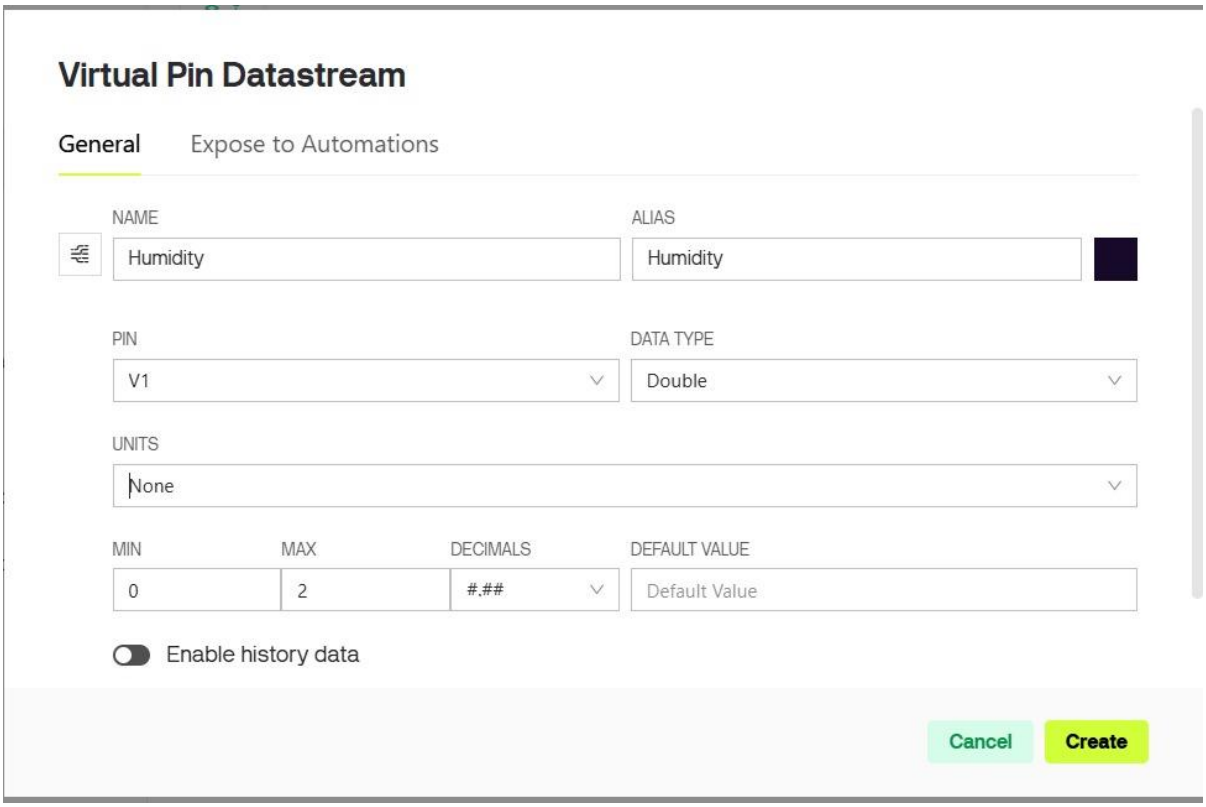
## Conclusion:

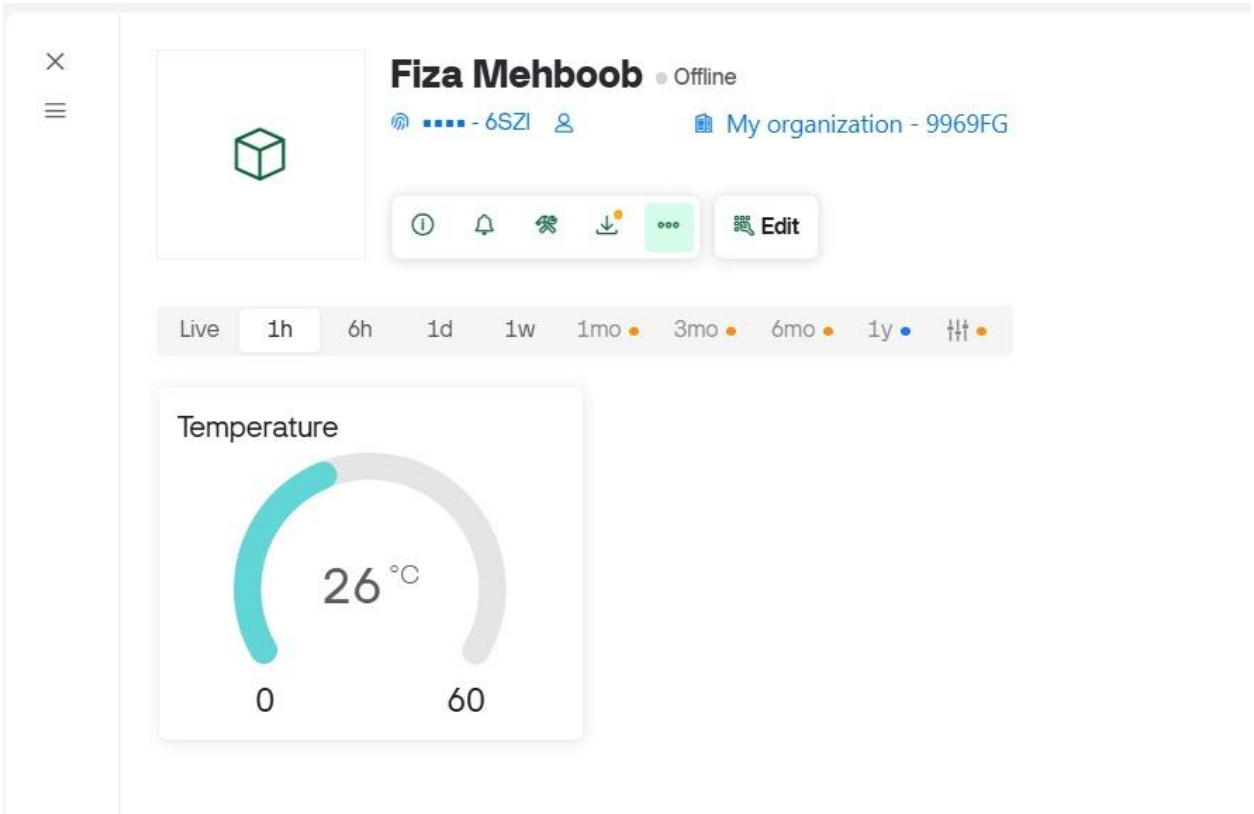
This ESP32-Blynk system demonstrates effective cloud-based IoT communication. By using Blynk templates, virtual pins, timers, and proper sensor configuration, the system provides reliable real-time temperature and humidity monitoring on both hardware and mobile dashboard.

## ESP Work:


Caps Lock
Shift
Ctrl
Environment Node
Temp: 29.9 C
Hum: 89.0 %
BTN -> manual update

# Web Interface:



## Mobile App: