National Textile University, Faisalabad

Department of Computer Science

Name:	Fareena Shahbaz
Class:	BSCS 5th _A
Registration No:	23-NTU-CS-1024
Course Name:	Embedded IoT Systems
Submitted To:	Sir Nasir Mahmood
Submission Date:	17/12/2025

HomeWork-1

Contents

Question-1 ESP32 Webserver

Part-A

Short Questions

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

WebServer server(80):

This will create a web server on the ESP32 that listens to incoming HTTP requests. The default port for communication using the HTTP protocol is Port 80. By employing this port, you can easily open the web page of the ESP32 using a browser with only the IP address being considered.

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

This statement defines a route for the web server. It tells the ESP32 that when a client opens the main page in a browser, the handleRoot() function should be executed. As a result, handleRoot() generates and sends the web page to the browser.

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

server.handleClient(); continuously checks for incoming browser requests. It is placed inside the loop() function so the ESP32 can respond to requests at any time.

If it is removed:

- The ESP32 will stop responding to browser requests
- The web page will not open or refresh

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

This statement sends the web page to the browser:

- **200** indicates that the request was successful.
- **"text/html"** tells the browser that the content is an HTML web page.
- **html** contains the actual web page data.

As a result, the browser displays the web page correctly.

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

- **Displaying last measured values:**
  - Uses previously stored temperature and humidity values.
  - Faster and more reliable.
  - Avoids unnecessary sensor reads.
- **Taking a fresh reading inside handleRoot():**
  - Reads the sensor every time the page is requested.
  - Slows down page loading.
  - May cause errors because DHT sensors cannot be read too frequently.

Part-B

Long Question

ESP32 Webserver-Based Temperature and Humidity Monitoring System

1. Wi-Fi Connection and IP Address:

The ESP32 connects to a Wi-Fi network using the provided SSID and password. After successful connection, it receives an IP address (static or dynamic), which is used to access the web server from a browser.

2. Web Server Initialization and Request Handling:

A web server is started on port 80. The ESP32 continuously checks for incoming browser requests and responds by sending the required web page.

3. Button-Based Sensor Reading and OLED Update:

A physical button is used to trigger temperature and humidity readings from the DHT sensor. When pressed, the ESP32 reads the sensor and updates the values on the OLED display.

4. Dynamic HTML Webpage Generation:

The ESP32 generates an HTML webpage dynamically. The webpage displays the latest stored temperature and humidity values and sends them to the browser.

5. Purpose of Meta Refresh:

The meta refresh tag automatically reloads the webpage after a fixed time interval, ensuring updated sensor values are shown without manual refresh.

6. Common Issues and Solutions:

- **Wi-Fi connection failure:**
  This can occur due to incorrect SSID or password.
  Solution: Verify Wi-Fi credentials and ensure the network is available.
- **Web page not loading:**
  Often caused by missing request handling or server polling.
  Solution: Ensure the server is properly started and client requests are handled continuously.
- **Slow or failed sensor readings:**
  DHT sensors are slow and can fail if read too frequently.
  Solution: Store sensor values and avoid reading the sensor on every web request.

- **IP address changes:**
  Dynamic IP assignment can cause difficulty in accessing the server.
  Solution: Use a static IP configuration.
- **Button bounce issues:**
  Rapid button presses may trigger multiple readings.
  Solution: Use a small debounce delay or edge detection.

Question-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 uniquely identifies the project template created on the Blynk Cloud. It defines the dashboard layout, widgets, and virtual pin mappings. The Template ID in the code must match the cloud template so that the ESP32 connects to the correct project and data is displayed properly.

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

The Blynk Template ID identifies the **project structure** on the Blynk Cloud. The Blynk Auth Token uniquely identifies and authenticates a specific device connected to that template.

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 both sensors use different data formats and specifications.
A main difference is that DHT22 has higher accuracy and a wider temperature range, while DHT11 has lower accuracy and limited range.

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

The Virtual Pins are software-defined pins used to send and receive data between the ESP32 and the Blynk Cloud. They are preferred because they allow cloud communication without being tied to physical GPIO pins and make the system more flexible and scalable.

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

BlynkTimer allows tasks to run at fixed time intervals without blocking the main program. Unlike **delay()**, it keeps the ESP32 responsive, ensuring continuous Wi-Fi connectivity and smooth communication with the Blynk Cloudrange.

Part-B

Long Question

Workflow of Interfacing ESP32 with Blynk Cloud

1. Blynk Template and DataStream Creation

A Blynk template is first created on the Blynk Cloud to define the project structure.
Data streams are added to the template using Virtual Pins (such as V0 for temperature and V1 for humidity). These datastreams receive sensor data from the ESP32 and display it on the dashboard.

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

The **Template ID** links the ESP32 to the correct Blynk project and must match the cloud template.
The **Template Name** helps identify the project on the Blynk platform.
The **Auth Token** uniquely authenticates the ESP32 device and allows secure communication with the Blynk Cloud.

3. Sensor Configuration Issues (DHT11 vs DHT22)

DHT11 and DHT22 sensors have different accuracy, ranges, and data formats.
Using incorrect sensor code leads to wrong readings. Therefore, the correct sensor type must be defined in the program.

4. Sending Data Using Blynk.virtualWrite()

After reading temperature and humidity from the sensor, the ESP32 sends data to the Blynk Cloud using Blynk.virtualWrite().

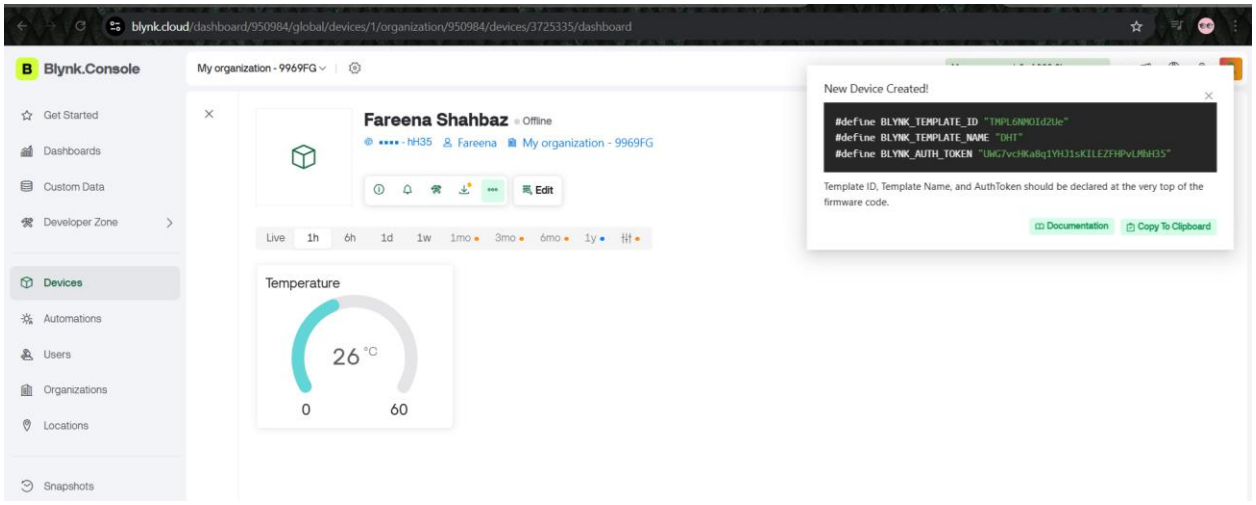
Each value is transmitted through assigned Virtual Pins, enabling real-time display on the Blynk dashboard.

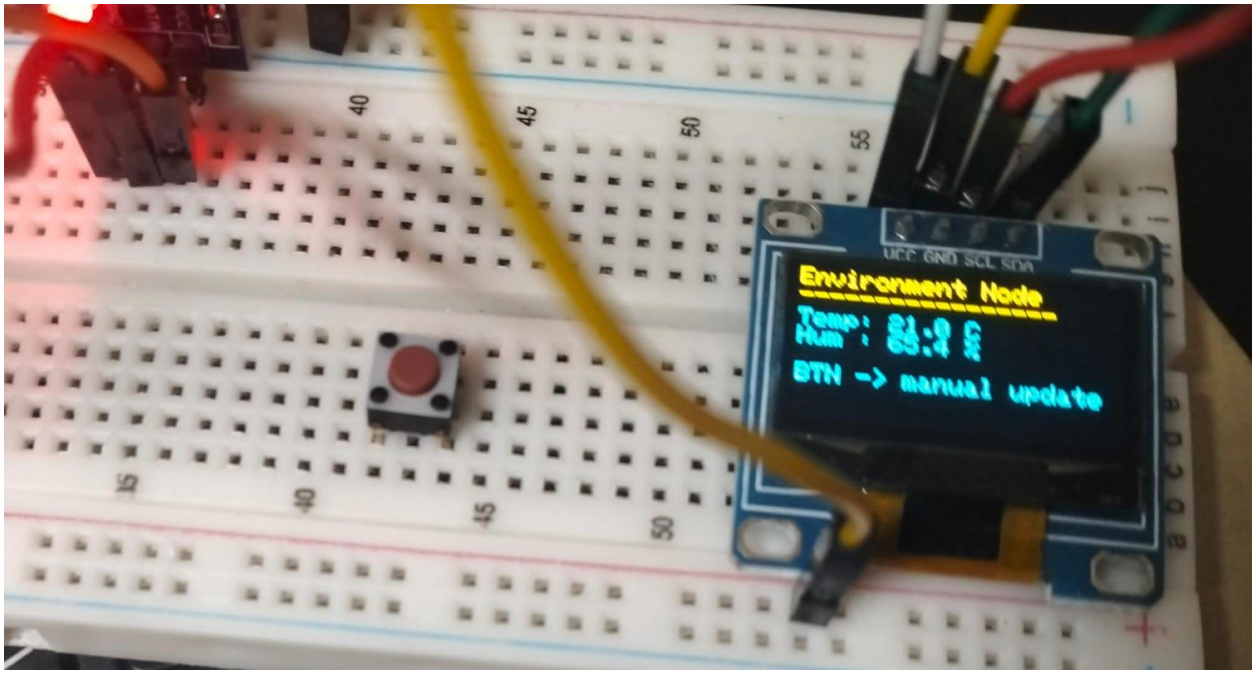
5. Common Problems and Solutions

Common issues include Wi-Fi connection failure, incorrect Template ID or Auth Token, wrong virtual pin mapping, and sensor misconfiguration.
These issues can be resolved by verifying network credentials, matching Blynk IDs, correctly assigning virtual pins, and using the proper sensor type.

Conclusion

Proper template setup, correct authentication details, accurate sensor configuration, and correct virtual pin usage enable reliable temperature and humidity monitoring using ESP32 and Blynk Cloud.





Virtual Pin Datastream

NAME: Humidity

ALIAS: Humidity

PIN: V1

DATA TYPE: Double

UNITS: Percentage, %

MIN	MAX	DECIMALS	DEFAULT VALUE
0	100	#.##	40

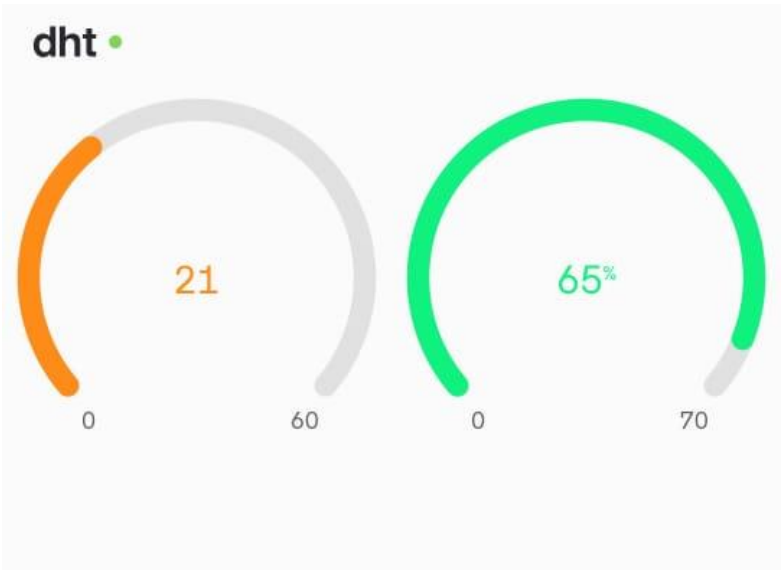
Enable history data

Close

Datastreams

Search datastream

ID	Name	Pin	Color	Data Type	Units	Is Raw	Min
1	Temprature	V0		Double	°C	false	0
2	Humidity	V1		Double	%	false	0



Github link:

https://github.com/Fareena184/Embedded-IoT-Systems-1024.git