Wania Zanib

23-NTU-CS-1289

Homework-01 – After mid

Embedded IOT Systems Fall2025

BSAI 5th

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?

This creates a web server on the ESP32 that listens on port 80.
Port 80 is the default HTTP port, so users can open the webpage in a browser without writing a port number.

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

This line tells the ESP32 what to do when someone opens the main webpage ("/").When the browser requests the home page, the function handleRoot() is executed.

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

server.handleClient(); continuously checks for new client requests. If it is removed, the ESP32 will not respond to browser requests, and the webpage will not load.

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

- 200 → request successful
- "text/html" → response type is a web page
- html → the actual HTML content sent to the browser

This line sends the webpage to the user.

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

- **Last value** shows already stored data (faster, no delay)
- **Fresh reading** reads the sensor again (more accurate but slower)

Taking a fresh reading inside handleRoot() can slow the webpage.

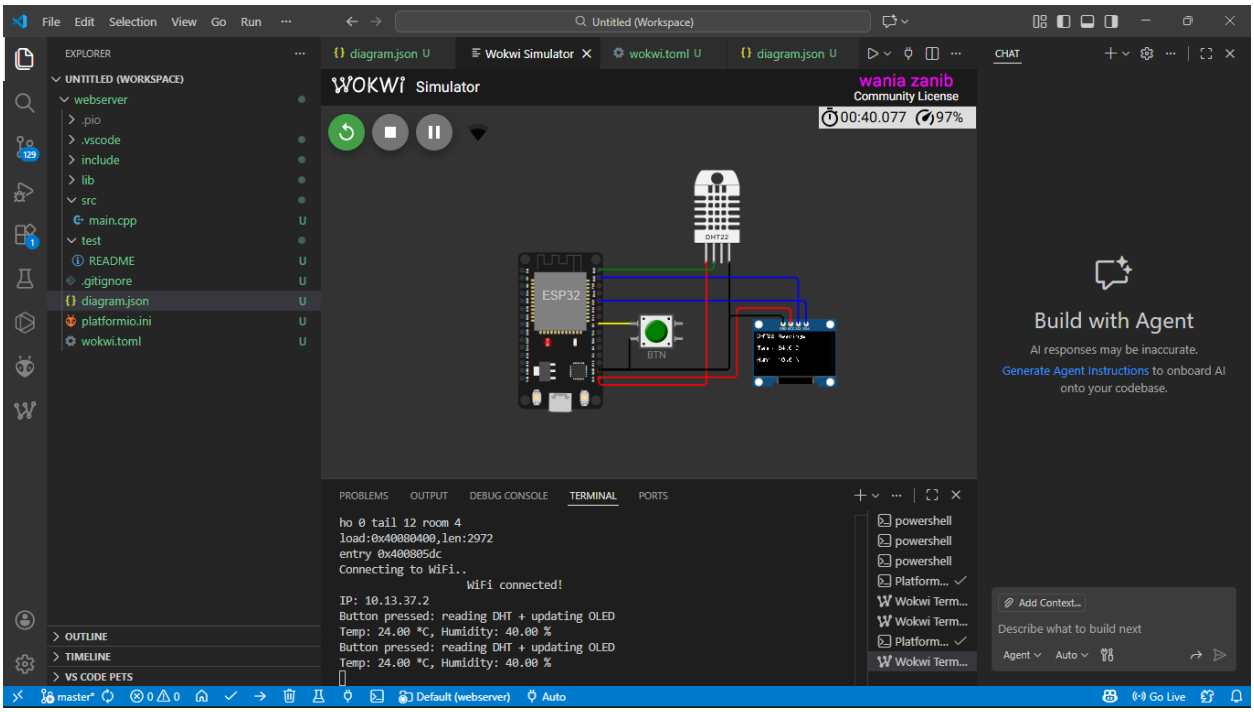
Part B: Long Question

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

First, the ESP32 connects to the Wi-Fi network using the SSID and password defined in the program. During this process, the ESP32 keeps checking the connection status until it is successfully connected. Once connected, the Wi-Fi router assigns a unique IP address to the ESP32. This IP address allows users to access the ESP32 web server through any web browser on the same network.

After establishing the Wi-Fi connection, the web server is initialized using server.begin() .This starts the HTTP server on port 80. The ESP32 then continuously listens for incoming browser requests. When a user enters the ESP32 IP address in the browser, the server identifies the request and responds using predefined routes such as /. Each route is linked to a specific function that handles the request and prepares the response.

A push button is connected to the ESP32 to control when sensor data is read. When the button is pressed, the ESP32 reads temperature and humidity values from the DHT sensor. These readings are then processed and displayed on the OLED screen, allowing real-time monitoring directly on the device without using a browser.



The webpage displayed in the browser is generated dynamically using HTML stored in string variables. The ESP32 inserts the latest temperature and humidity values into the HTML code before sending it to the client. This ensures that the webpage always shows updated sensor data instead of static information.

To keep the webpage updated automatically, a meta refresh tag is included in the HTML code. This tag reloads the webpage after a fixed time interval, such as every few seconds. As a result, the user can see updated sensor values without manually refreshing the page.

Common Issues and Their Solutions

- **Wrong Wi-Fi credentials:** Double-check the SSID and password. Incorrect credentials will prevent the ESP32 from connecting to the network.
- **Webpage not loading:** Ensure that server.handleClient() is placed inside the loop() function so the ESP32 can process incoming requests.
- **OLED not updating:** Verify the I2C address, wiring connections, and display initialization code.
- **Slow response:** Avoid taking frequent DHT sensor readings inside the web request handler, as this can delay page loading.

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?

Template ID links the ESP32 project with the correct Blynk Cloud template**.** If it does not match, the device will not connect.

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

	Template ID	Auth Token
Purpose	Identifies the IoT project on Blynk Cloud	Authorizes a specific physical device
Used For	Linking ESP32 to a predefined project template	Allowing the ESP32 to securely connect to the cloud
Scope	Same for all devices using the same template	Unique for each individual device
Provided By	Generated when a template is created in Blynk Cloud	Generated when a device is added to the template
Security Role	Defines project structure, not device security	Acts as a security key for device authentication
Without It	Device cannot link to the correct cloud project	Device cannot connect to Blynk Cloud
Example Use	Ensures correct datastreams and widgets are used	Prevents unauthorized devices from sending data

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

The DHT11 and DHT22 sensors use similar communication protocols, but their data formats and precision are different so if we use DHT22 code with a DHT11 sensor, the ESP32 will interpret the data incorrectly, resulting in wrong temperature and humidity readings.

Feature	DHT11	DHT22
Temperature Range	0°C – 50°C	-40°C – 80°C
Humidity Range	20% – 80%	0% – 100%
Data Format	8-bit values	16-bit values

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

Virtual Pins in Blynk are software-based pins that allow ESP32 to communicate with the Blynk app without using the physical GPIO pins. We can send or receive data like sensor readings, control LEDs, or motors through virtual pins.

Why preferred over physical pins:

- They allow unlimited controls without worrying about the hardware pin limits.
- Enable cloud communication, so the device can send or receive data from anywhere.
- Reduce hardware complexity and make the app more flexible.

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

BlynkTimer is used to run tasks repeatedly without blocking the rest of the code. Unlike delay(), which pauses the whole program for a given time, BlynkTimer allows the ESP32 to handle multiple things at once, like reading sensors, updating the Blynk app, and controlling devices simultaneously.

Benefits:

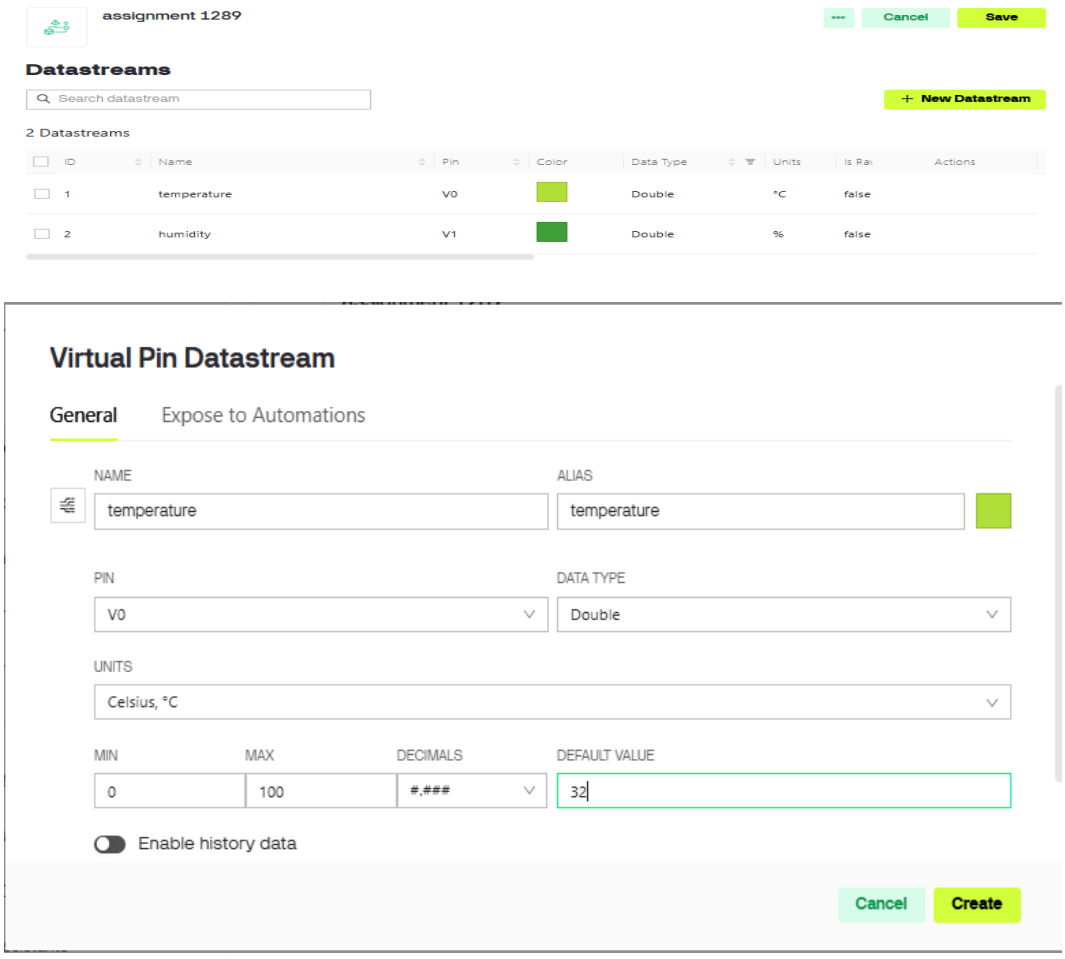
- Non-blocking: The program keeps running while waiting.
- Better for real-time IoT applications.
- Can schedule multiple functions with different intervals easily.

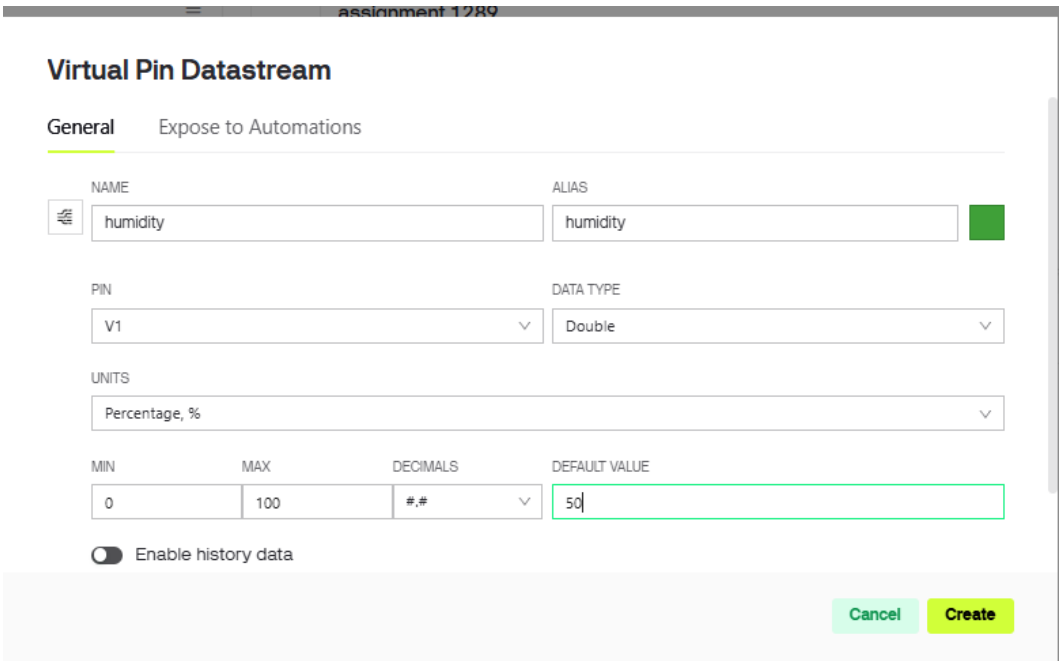
Part-B: Long Question

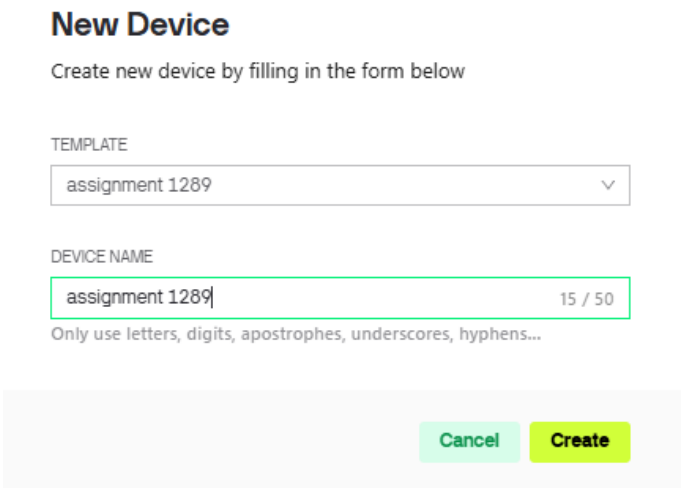
Explain the complete workflow of interfacing ESP32 with Blynk Cloud to display temperature and humidity values.

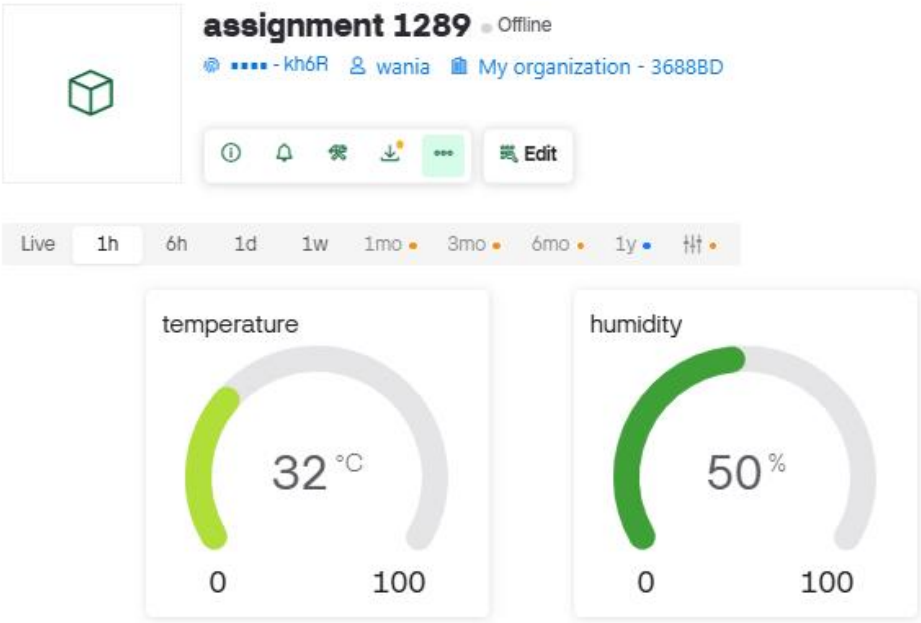
1. **Creating a Blynk Template and Datastreams**
    - **Step 1:** Log in to the Blynk IoT platform and create a new Device Template for ESP32.
    - **Step 2:** Add Datastreams in the template. Datastreams are channels to send or receive data between the hardware and the app.These datastreams are then linked to Virtual Pins on the ESP32.

Blynk Cloud (Web):

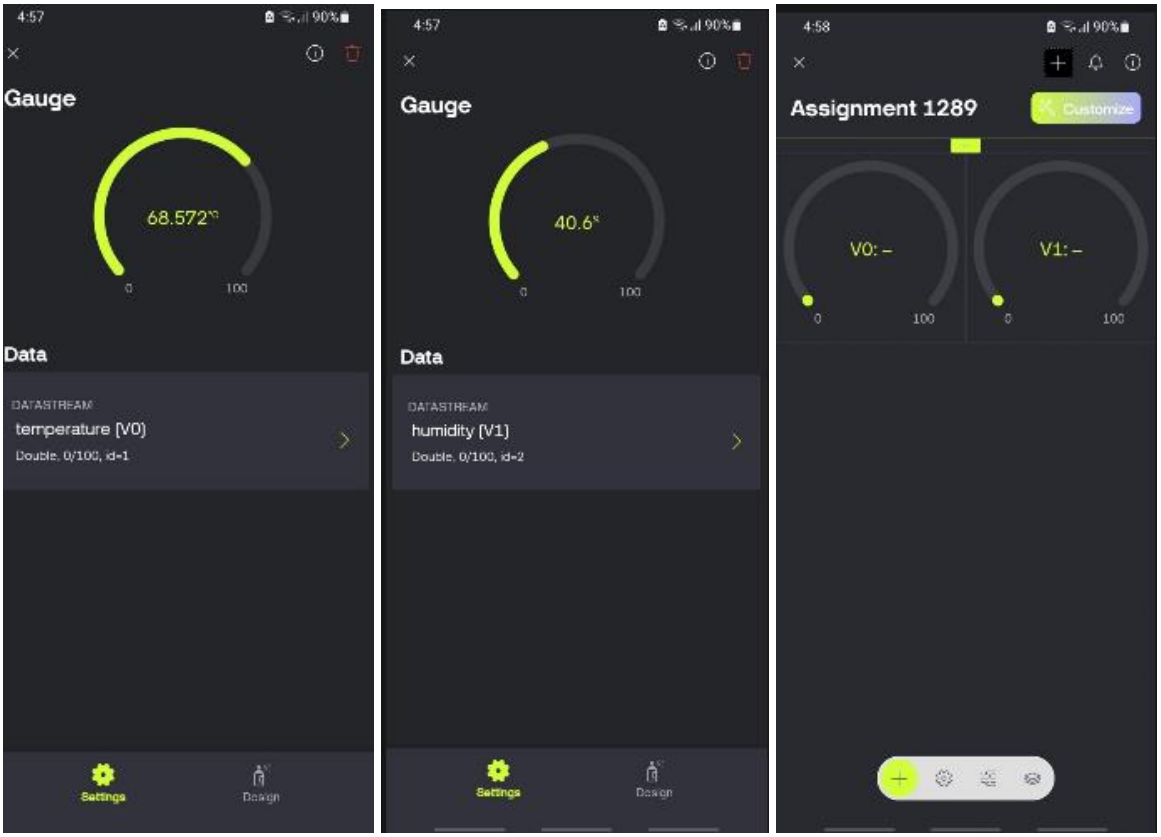



assignment 1289
Virtual Pin Datastream
General
Expose to Automations
NAME
humidity
ALIAS
humidity
PIN
V1
DATA TYPE
Double
UNITS
Percentage, %
MIN
0
MAX
100
DECIMALS
#.#
DEFAULT VALUE
50
Enable history data
Cancel
Create


New Device
Create new device by filling in the form below
TEMPLATE
assignment 1289
DEVICE NAME
assignment 1289
15 / 50
Only use letters, digits, apostrophes, underscores, hyphens...
Cancel
Create


assignment 1289
Offline
- kh6R
wania
My organization - 3688BD
Edit
Live
1h
6h
1d
1w
1mo
3mo
6mo
1y
temperature
32 °C
0
100
humidity
50 %
0
100

Mobile App:



2. **Role of Template ID, Template Name, and Auth Token**
    - **Template ID:** A unique identifier for device template. The ESP32 uses it to know which template it belongs to on Blynk Cloud.
    - **Template Name:** Just a name for the template. Useful for organizing multiple devices.
    - **Auth Token:** A unique key that allows ESP32 to connect securely to the Blynk Cloud. Without it, the device cannot send or receive data.

3. **Sensor Configuration Issues (DHT11 vs DHT22)**
    - Using the wrong code for the sensor can give wrong readings.
    - **DHT11**: Lower temperature and humidity range, less accurate.
    - **DHT22**: Wider range, more accurate.

4. **Sending Data using Blynk.virtualWrite()**

After reading values from the DHT sensor, the ESP32 sends the data to Blynk Cloud using virtual pins:

float temperature = dht.readTemperature();

```
float humidity = dht.readHumidity();

Blynk.virtualWrite(V1, temperature);
Blynk.virtualWrite(V2, humidity);
```

The Blynk app automatically updates the corresponding widget connected to the virtual pin.

5. **Common Problems and Solutions**

Problem	Cause	Solution
ESP32 not connecting to Blynk	Wrong Auth Token, Wi-Fi issue	Check Auth Token, SSID, and password
Wrong sensor readings	Using DHT22 code with DHT11	Match the code to the correct sensor type
Widgets not updating	Wrong virtual pin linked	Verify widget is linked to correct virtual pin
App shows disconnected	Poor internet or firewall	Check internet, disable firewall, or restart router
ESP32 freezing or delayed updates	Using delay() instead of BlynkTimer	Use BlynkTimer for non-blocking code