Question 1

Part A:

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

It creates an object of WebServer class on ESP32 named Server. This is responsible for handling incoming HTTP requests.

- *Port 80 is default port for HTTP communication, which permits us to access the ESP32 Web Page directly through a web browser with out any specified port number.*

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

It registers a handler function (handleRoot) to be to be executed whenever the web server receives an HTTP request for the root path ("/") of the server. It links a specific web request path to a function that generates and sends the web page.

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

It is important because it continuously checks and processes incoming HTTP requests from clients.

- ***Why in Loop****: Because the loop runs continuously and repeatedly; making it the perfect place to constantly monitor for new client connections and service any pending web requests.*
- ***What if removed***? If removed; the ESP32 will stop responding to any web request. The WebServer will be initialized but it will be non-functional; as it won't be checking or processing any client connection.

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

This step is crucial as it sends back generated WebPage to clients browser.

- ***200:*** *It is HTTP Success status code.*
- ***"text/html":*** *Content Type of Response; that is informing browser that the data being sent is an HTML document.*
- ***Html:*** It is dynamically generated WebPage content.

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

The choice determines how up to date the data on the webpage is.

Feature	*Last Measured Sensor Values*	*Fresh DHT Reading*
Data Source	*lastTemp and lastHum variables, updated only when the physical button is pressed.*	*Calls readDHTValues() every time the webpage is loaded or refreshed.*
Data Freshness	*Data is only as fresh as the last button press.*	*Data is always the most current reading at the time of the request.*
System Load	*Low. Sensor reading is decoupled from web request.*	***High**. Sensor reading adds latency to every web request, potentially delaying response.*
Use Case	*Suitable when readings are infrequent, or the sensor has a slow read-rate limit.*	*Suitable for fast updates where real-time display is critical.*

Part B:

Working of ESP32 Webserver-Based Temperature and Humidity Monitoring System

This system uses an ESP32 microcontroller to monitor temperature and humidity using a DHT22 sensor and display the data on both an OLED screen and a web browser.

- ***ESP32 Wi-Fi connection process and IP address assignment:***
  - ✓ ***Initialization:** The setup() function calls WiFi.begin(ssid, password).*
  - ✓ ***Connection:** The ESP32 attempts to connect to the WIFI network defined by the credentials(ssid, password).*
  - ✓ ***Waiting:** The code includes a while loop that pauses execution (delay (500);) util the WiFi.status() is WL_Connected.*
  - ✓ ***IP Assignment:** Once connected, ESP32 requests an IP address from the Wi-Fi router. The assigned local IP address is printed to the Serial Monitor and displayed on the OLED using WiFi.localIP().*

- ***Web server initialization and request handling:***

  - ✓ ***Initialization:*** *WebServer server(80); creates the server object on port 80.*
  - ✓ ***Route Mapping:*** *server.on("/", handleRoot); maps the root URL (/) to the handleRoot() function.*
  - ✓ ***Start Server:*** *server.begin(); starts the HTTP server, making it ready to listen for incoming connections.*
  - ✓ ***Request Processing:*** *In the loop(), server.handleClient(); constantly checks for and services any pending HTTP requests.*

- ***Button-based sensor reading and OLED update mechanism:***
  - ✓ ***Button Input:*** *The button is connected as active LOW using INPUT_PULLUP.*
  - ✓ ***Edge Detection:*** *The loop() function checks for a falling edge (lastButtonState == HIGH && currentButtonState == LOW). This detects when the button is physically pressed down.*
  - ✓ ***Action:*** *Upon a confirmed button press:*
    - o *readDHTValues() is called to read the DHT sensor and update the global lastTemp and lastHum variables.*
    - o *showOnOLED() is called to clear the OLED and display the newly updated temperature and humidity values.*

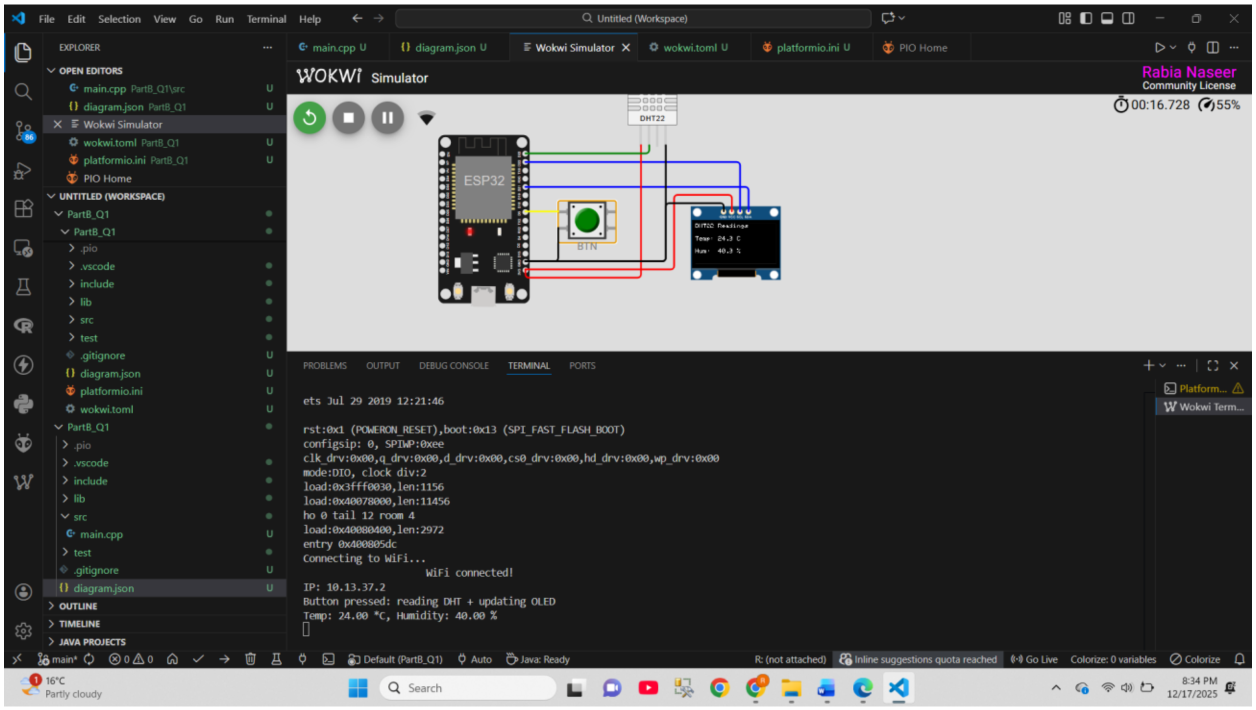
- ***Dynamic HTML webpage generation:***
  - ✓ *The handleRoot() function constructs the webpage as a String variable named html.*
  - ✓ *The measured values (lastTemp, lastHum) are dynamically inserted into the HTML content using html += String(lastTemp, 1).*
  - ✓ *The complete HTML string is then sent to the client's browser using server.send(200, "text/html", html).*

- ***Purpose of meta refresh in the webpage:***
  It instructs the client's web browser to automatically refresh the webpage every 5 seconds. This ensures that the user's browser periodically fetches the latest sensor data from the ESP32 server without the user having to manually click the refresh button.
- ***Common issues in ESP32 webserver projects and their solutions:***

Issue	***Solution***
WiFi not connecting	*Check ssid/password & signal strength*
DHT sensor returning NaN	*Add delays and avoid frequent reads.*

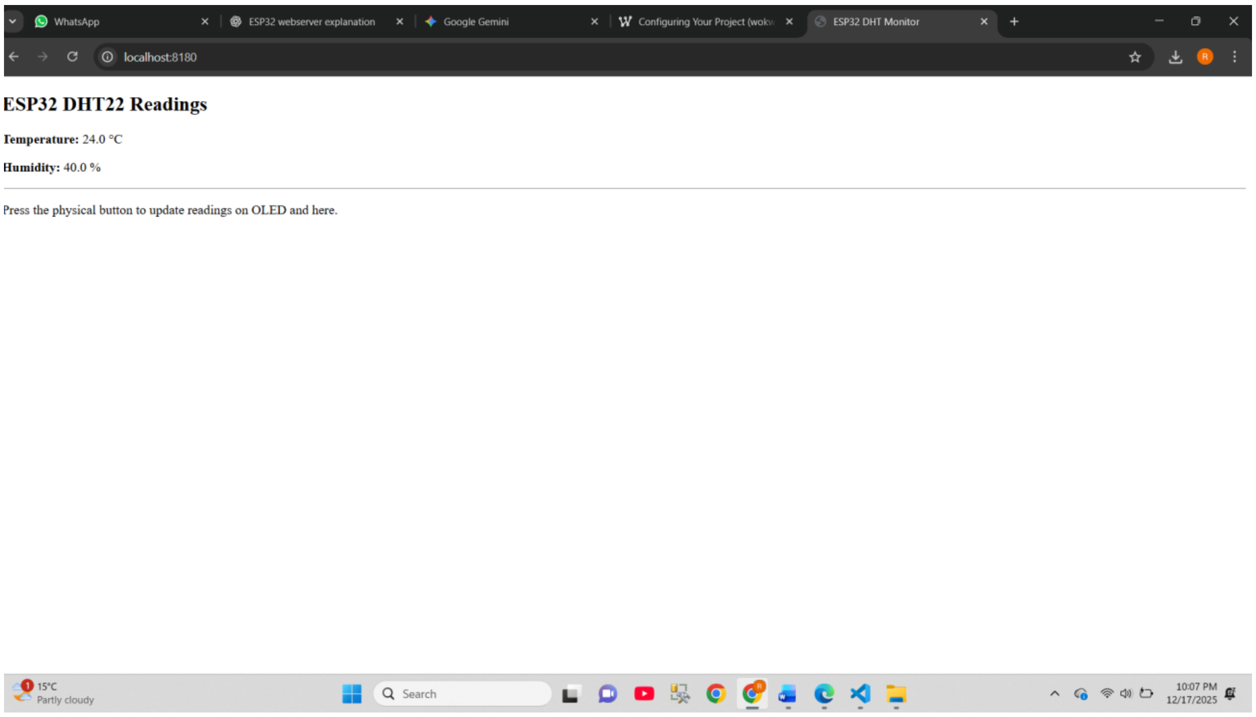
Web Page not loading	*Ensure server.handleClient() is in loop.*
OLED not displaying	*Verify I2C address and wiring*

- ***Conclusion:*** *The ESP32 web server system efficiently combines Wi-Fi communication, sensor interfacing, OLED display, and web technologies to create a reliable IoT monitoring solution.*



```
ets Jul 29 2019 12:21:46

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:1156
load:0x40078000,len:11456
ho 0 tail 12 room 4
load:0x40080400,len:2972
entry 0x400805dc
Connecting to WiFi...
                    WiFi connected!
IP: 10.13.37.2
Button pressed: reading DHT + updating OLED
Temp: 24.00 *C, Humidity: 40.00 %
```



ESP32 DHT22 Readings

Temperature: 24.0 °C

Humidity: 40.0 %

Press the physical button to update readings on OLED and here.

Question 2

Part A:

1. ***What is the role of Blynk Template ID in an ESP32 IoT project? Why must it match the cloud template?***

It is a unique identifier that links ESP32 device to a specific template created in the Blynk Cloud.

- ***Why must match?*** *If the on-device Template ID does not match the one on the cloud, the Blynk will not recognize the device and ESP32 will fail to correctly display data on dashboard. It ensures that the device configuration, data streams, and widgets match the cloud setup.*

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

FEATURE	*TEMPLATE ID*	*AUTH TOKEN*
PURPOSE	*Identifies the Project template in blynk Cloud.*	*Acts as a unique security key or password that*

		authenticates a specific device to connect to Blynk servers.
UNIQUENESS	*Unique to the Template*	*Unique to the Device*

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

They use different timing protocols for data transmission, and the DHT library uses these defined types to correctly decode the signal.

- *DHT22 provides higher accuracy and a wider range while DHT11 provides a lower accuracy and limited range.*

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

Virtual Pins(Vo, V1) are software-based pins that act as a flexible interface for exchanging any type of data between ESP32 and blynk cloud.

They are preferred over GPIO because:

- ***Flexibility:*** *They can send or receive any data(integers, strings, floats) and are not limited to the digital HIGH/LOW state of a physical pin.*
- ***Decoupling:*** *They decouple the widget's function on the dashboard from the physical hardware pin, allowing pins to be repurposed without changing the cloud setup.*
- ***Bidirectional:*** *They support both sending data(using Blynk.virtualWrite()) and receiving data(using BLYNK_WRITE) handlers.*

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

- ***Problem with delay():*** *The ESP32's loop() must constantly execute Blynk.run() to maintain the connection to the Blynk cloud and process incoming/outgoing messages. A standard delay(5000) would freeze the program for 5 seconds, causing the device to disconnect from the cloud or miss requests.*
- ***Advantages of BlynkTimer:*** *It allows non-blocking execution.*

- ✓ *Keeps WIFI and cloud communication active.*
- ✓ *Prevents freezing of ESP32.*
- ✓ *Essential for real time IOT applications.*

Part B:

workflow of interfacing ESP32 with Blynk Cloud

This project sends temperature and humidity data from an ESP32 to the Blynk Cloud for real-time monitoring.

- ***Creation of Blynk Template and Datastreams:***
  - ✓ ***Template:*** *On the Blynk web dashboard, a template is created. This template defines the structure for all devices of this type. This step generates the Template ID (BLYNK_TEMPLATE_ID).*
  - ✓ ***DataStreams:*** *Within the template, DataStreams are created to map to the Virtual Pins used in the code.*
    - **V0 (Temperature):** *Type typically set to Double or Float, Unit Celsius.*
    - ***V1 (Humidity):*** *Type typically set to Double or Float, Unit Percentage %.*
- ***Role of Template ID, Template Name, and Auth Token:***
  - ✓ ***Template ID:*** *Connects device firmware to cloud template.*
  - ✓ ***Template Name:*** *Used for human-readable identification on the cloud platform.*
  - ✓ ***Auth Token:*** *Assigned to specific device created under the template. It is used in Blynk.begin() to authenticate the ESP32 instance with the Blynk Cloud. In short, it's a secure key allowing device-cloud communication.*
- ***Sensor configuration issues:***
  - ✓ *Selecting wrong sensor type (DHT11 vs DHT22) causes errors.*
  - ✓ *Correct library and sensor definition ensure accurate readings.*

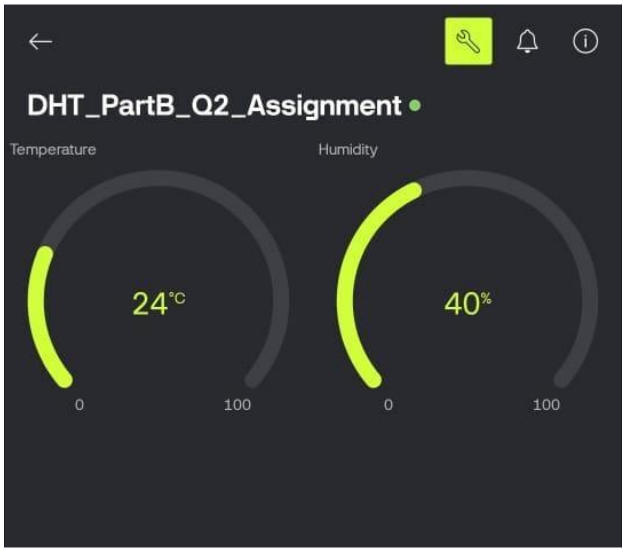
- ***Sending data using Blynk.virtualWrite():***
  - ✓ ***Reading:*** *The readAndDisplayAndSend() function first reads the sensor values (t and h).*
  - ✓ ***Transmission:*** *The data is sent to the cloud using:*
    - *Blynk.virtualWrite(V0, t)*
    - *Blynk.virtualWrite(V1, h)*

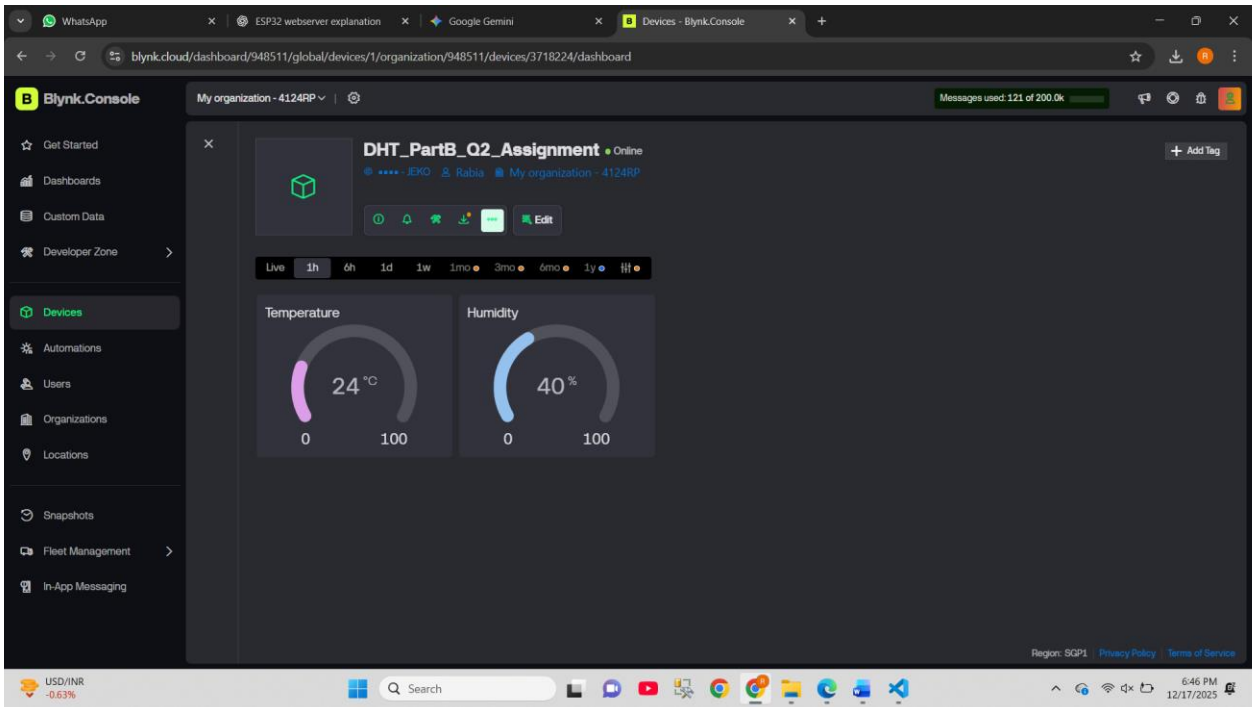
- ✓ ***Timing:*** *The transmission is triggered either periodically by BlynkTimer(every 5 seconds) or manually by a button press.*
- ✓ ***Display:*** *The Blynk cloud then pushes this data to the configured widgets(Gauge) on Blynk Web or mobile dashboard.*

- ***Common problems faced during configuration and their solutions:***

Problem	*Solution*
Device Offline	*Check Auth Token and WiFi*
No data on app	*Verify virtual pins*
Frequent Disconnection	*Avoid delay(), use Timers*
Wrong values	*Confirm sensor type*

- ***Conclusion:*** *Blynk Cloud provides an efficient platform for remote monitoring of IoT devices. Proper configuration of templates, data streams, and firmware ensures smooth and reliable operation.*

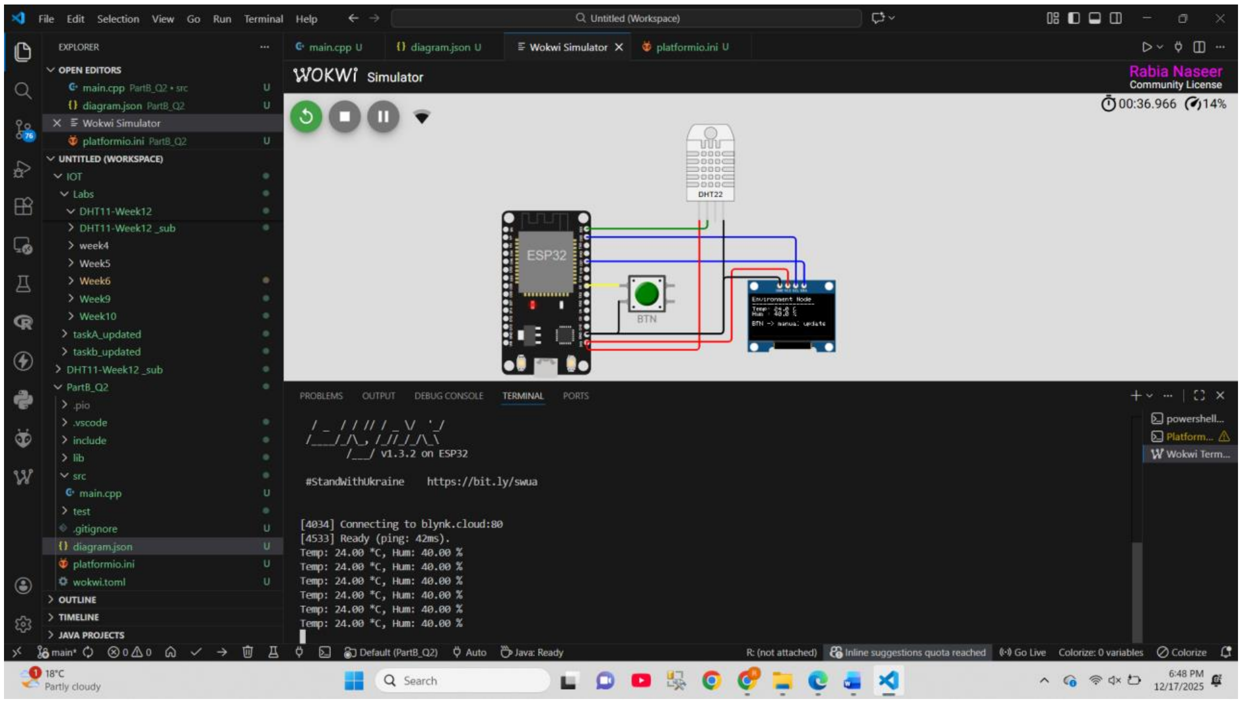




DHT_PartB_Q2_Assignment • Online

JEKO Rabia My organization - 4124RP

Temperature	Humidity
24 °C (0–100)	40 % (0–100)



```
  / _  / / // / _ \/  '_/
 /____/_/\_, /_//_/_/\_\
        /___/ v1.3.2 on ESP32

 #StandWithUkraine    https://bit.ly/swua


[4034] Connecting to blynk.cloud:80
[4533] Ready (ping: 42ms).
Temp: 24.00 *C, Hum: 40.00 %
Temp: 24.00 *C, Hum: 40.00 %
Temp: 24.00 *C, Hum: 40.00 %
Temp: 24.00 *C, Hum: 40.00 %
Temp: 24.00 *C, Hum: 40.00 %
Temp: 24.00 *C, Hum: 40.00 %
```