National Textile University, Faisalabad

Department of Computer Science

Name	Nimra Fatima
Reg no	23-NTU-CS-1081
Section	BSCS 5th -B
Subject	Embedded&Iot Systems
Homework	1 (After-Mids.)
Submitted to	Sir Nasir
Submission date	17-12-25

Nimra Fatima

Homework

Part A: Short Questions

Question#1

ESP32 Webserver (Webserver.cpp)

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

Purpose of Webserver:

A web server is software or system that hosts websites and serves web pages to clients (like browsers) over the internet or network. When we write server (80) often in ESP32 or another web server it means:

- Creating a server instance that listens for incoming HTTP requests.

Representation of port 80:

Port 80 is the default port for HTTP (Hyper Text Transfer Protocol)

HTTP is the standard protocol used to transfer web Pages over the internet.

When we type URL like http:// so our browser is trying to connect with the port 80 which is the defaulter of web pages

----

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

Role of server. On ("/", handleRoot):

- Server. on is used to define a route for the web server that the server will respond to.
- "/" it means the root of website (main page).
- handleRoot is the function that runs when someone accesses this page. This function decides what messages, webpages, or data are sent back to the user.
- When a user opens the web page at server address, the server calls the handleRoot function to show the content.
- In short, server.on("/", handleRoot) defines and tells web server to run handleRoot function whenever the root URL ("/") is accessed.

----

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

Role of server. handleClient ():
This function checks for incoming client requests (like when someone visits web page) and processes them by calling appropriate handler function(handleRoot).

Inside the loop:

The loop () function in ESP32 runs repeatedly. Placing server. handleClient (); inside loop () ensures server continuously listens and responds to new requests.

When it is removed:
if server. handlClient (); is not called then server will not respond to any client requests. The webpage will not load, and your ESP32 will appear unresponsive to browser requests.

Server.handleClient(); keeps the server active and responsive. Without it, the server cannot interact with clients.

----

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

Server.send (200, “text/html”, html) is used to send a response from webserver to the client (browser).**200** that is the **HTTP** status code, which means request successful. **text/html** that tells the browser that the response content is an HTML webpage. **html** that is the actual HTML content (webpage) that will be displayed in the browser.

Therefore, this statement sends a successful HTML page from the ESP web server to the user’s browser.

----

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

Major difference is the **sensor data** that is obtained.

Last measured sensor values: it means that webpage shows sensor readings that were already taken earlier (stored in variables). This is faster and more stable because no new sensor readings are taken when page is loaded.

Fresh DHT reading: handleRoot () means sensor is read every time the webpage is opened and refreshed. This gives the latest values, but it can cause delays or inaccurate readings because DHT sensors need time between readings and do not work well if read too frequently.

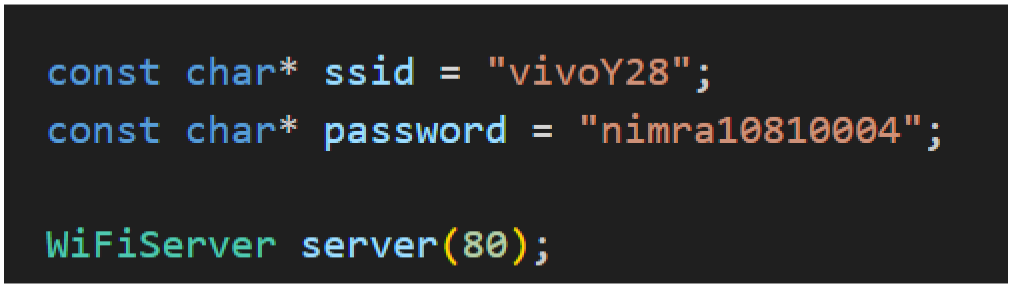
Therefore, sensor values that are last measured are faster and safer and fresh readings in handleRoot () is the latest data, but it may cause error or slow response.

Part B: Long Question

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

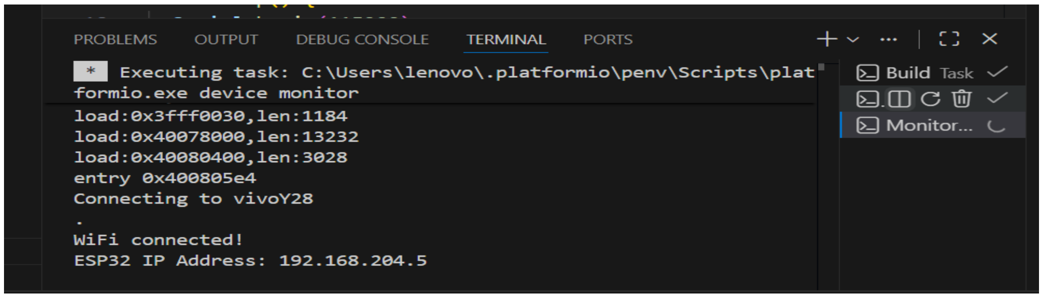
ESP32 Wi-Fi connection process and IP address assignment:

ESP32 connects to Wi-Fi network and using the provided SSID and password, after a successful connection, router assigns an IP address to ESP32 and through that IP address we access webpage from where the ESP32 detects reading from DHT sensor.



```
const char* ssid = "vivoY28";
const char* password = "nimra10810004";

WiFiServer server(80);
```

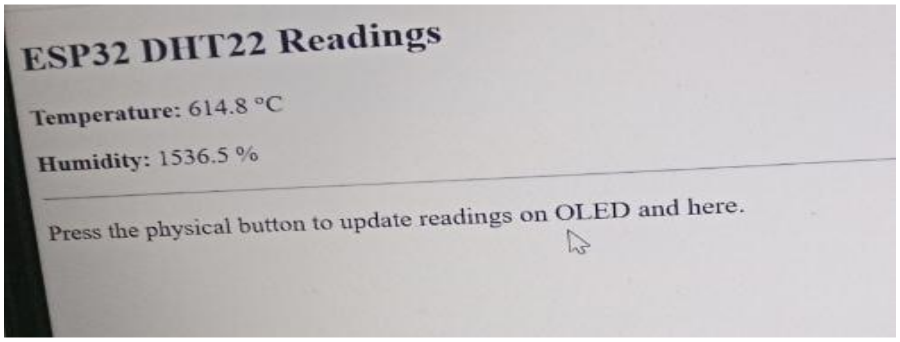
Connection and IP Address:



```
 *  Executing task: C:\Users\lenovo\.platformio\penv\Scripts\plat
formio.exe device monitor
load:0x3fff0030,len:1184
load:0x40078000,len:13232
load:0x40080400,len:3028
entry 0x400805e4
Connecting to vivoY28
.
WiFi connected!
ESP32 IP Address: 192.168.204.5
```

Web server initialization and request handling:

After connecting to ESP32 and getting IP address. ESP32 initializes webserver on port 80. server processes request and uses server. handleClient () and the we show that webpage through that IP address.



ESP32 DHT22 Readings

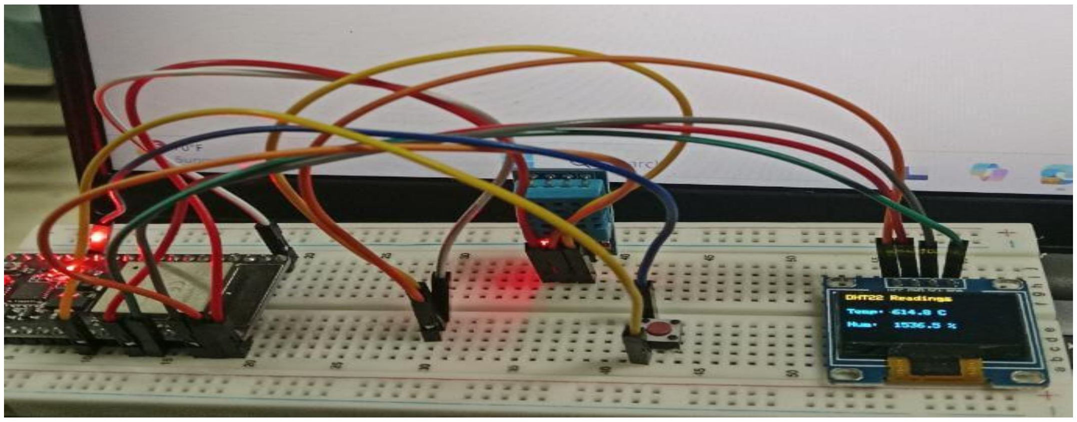
Temperature: 614.8 °C

Humidity: 1536.5 %

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

Button-based sensor reading and OLED update mechanism:

When button is pressed then a trigger generates the ESP32 reads the DHT sensor values and fresh values ae then shown on OLED.



Dynamic HTML webpage generation:

Dynamic HTML webpage generation means the ESP32 creates webpage content at runtime. Sensor values are then inserted into HTML code. The values are shown on the webpage of the corresponding values through DHT sensor.

Purpose of meta refresh in the webpage:

Purpose of meta refresh tag in webpage is to automatically reload web page. This allows the temperature and humidity values to update automatically without user manually refreshing page.

Common issues in ESP32 webserver projects and their solutions:

Common issues are Wi-Fi connection error.

Connecting to that specific network causes delays and shows the diamond shape continuously.

Due to weak connection, it does not open the webpage.

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?**

Role of Blynk Template ID:

A unique identifier for specific template that we created in Blynk. It ensures the ESP32 communicates correctly with app, mapping its sensors and controls to widgets defined in the template.

Match cloud template:

If ESP32's template ID does not match, cloud cannot associate the device with correct template. That leads to errors in data display, control and synchronization between hardware and Blynk app.

----

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

Blynk Template ID	**Blynk Auth Token**

Identifies the **template** in the Blynk Cloud that defines the device's structure, widgets, and pins.	Authenticates a **specific device** to connect to the Blynk Cloud
Template level (applies to all devices using that template)	Device-level (unique for each device)

Template-ID is the blueprint that a device must follow.

Auth Token is the device allowed to Blynk Cloud.

----

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

Because both of two sensors have different protocols and different timing. **DHT22** expects a faster response and higher resolution while DHT11 **is** slower and less precise.

Key difference:

- **DHT11:** Measures temperature 0–50°C, humidity 20–80%, with lower accuracy.
- **DHT22:** Measures temperature −40–80°C, humidity 0–100%, with higher accuracy.

So, the code designed for DHT22 cannot correctly interpret DHT11's data.

----

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

Virtual Pins in Blynk are software defined pins used to send or receive data between **ESP32** and **Blynk Cloud/App.** They don't correspond to any physical GPIO's pin on the device.

Preference:

Because it provides flexibility we don't worry about the limitations of wires and it separates the cloud communication separate from physical hardware, avoiding conflicts.it is ideal for sending sensor readings, controlling widgets, or storing values in cloud.

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

Blynk Timer is used to run functions at regular intervals without blocking main program, unlike delay (), which stops all other code while waiting.

1. **Non-blocking:** Lets the ESP32 handle multiple tasks simultaneously, like reading sensors, updating the display, and communicating with Blynk.
2. **Efficient timing:** Functions can be scheduled to run periodically without pausing the program.
3. **Smooth operation:** Ensures real-time data updates and responsiveness of the IoT system.

In short: BlynkTimer allows timed actions while keeping the ESP32 responsive, whereas delay () would pause everything.

Part-B: Long Question

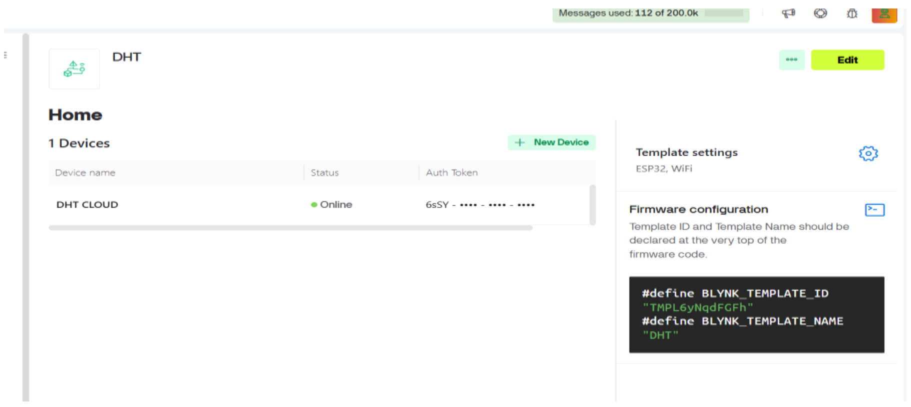
Explain the complete workflow of interfacing ESP32 with Blynk Cloud to display temperature and humidity values.

- **Creation of Blynk Template and DataStream:**

Blynk template defines structure of IoT device in Blynk Cloud.

Blynk Template:

Login to Blynk.io after creation of account click new template, enter name, device type, connection type and add some widgets save the template it will generate Template ID and Auth Token.



Device name	Status	Auth Token
DHT CLOUD	● Online	6sSY - •••• - •••• - ••••

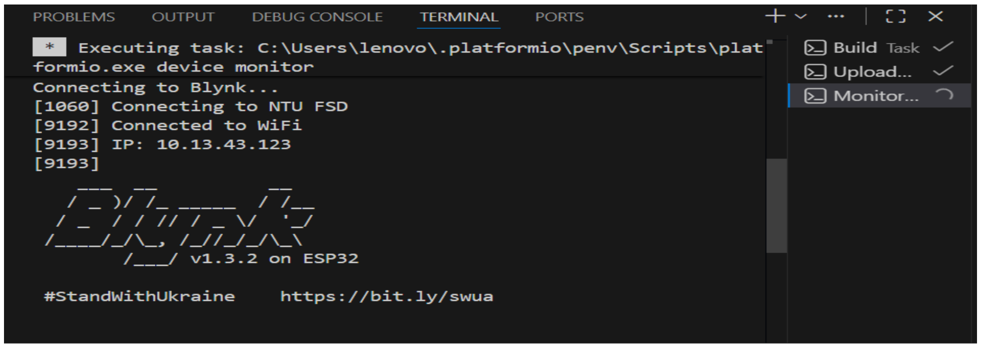
Template settings
ESP32, WiFi

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

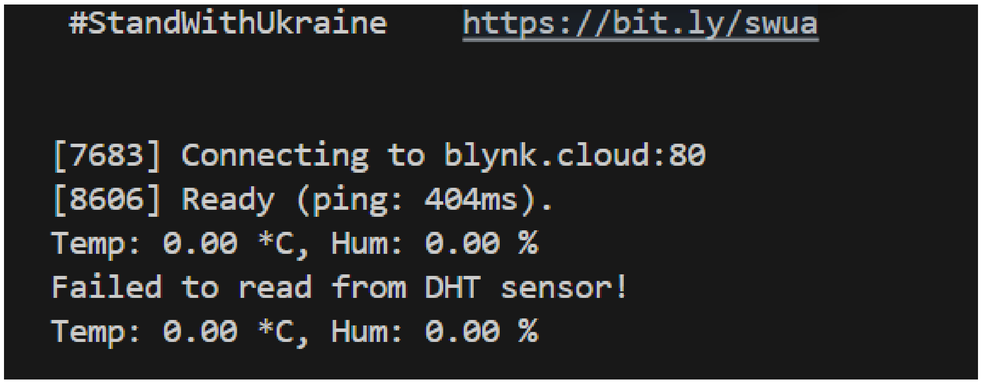
```
#define BLYNK_TEMPLATE_ID
"TMPL6yNqdFGFh"
#define BLYNK_TEMPLATE_NAME
"DHT"
```

DataStream:

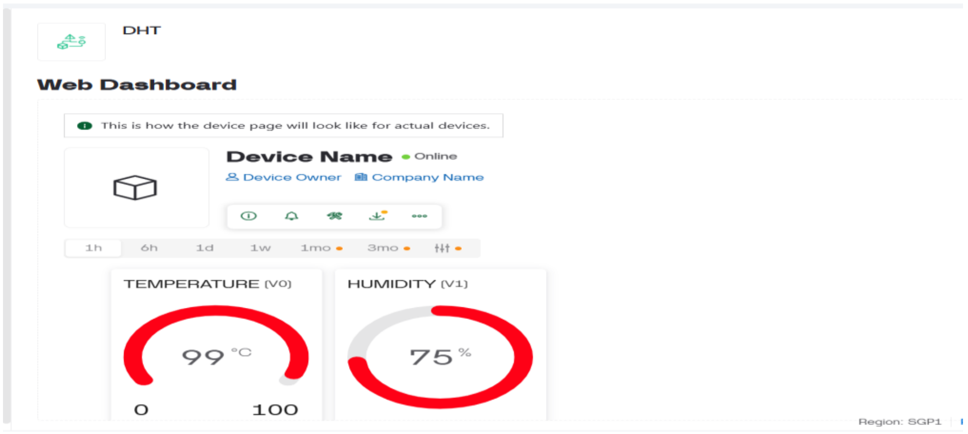
Open template in Blynk Console then click data stream, choose a virtual pin and set data type and then save it.

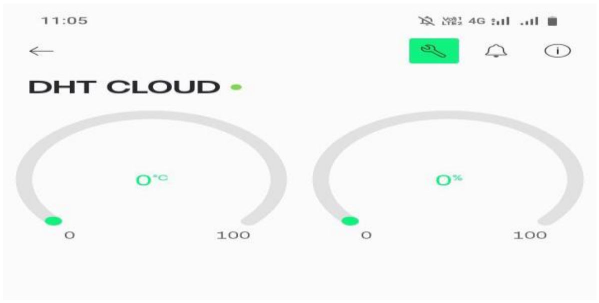


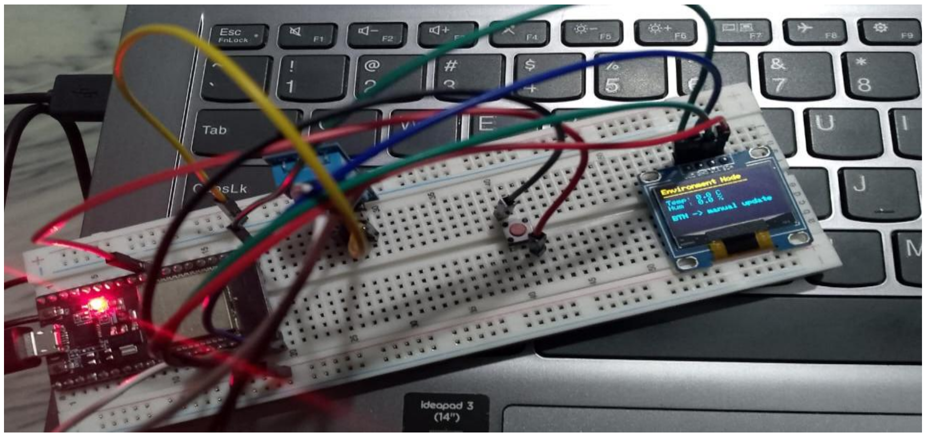
```
 *  Executing task: C:\Users\lenovo\.platformio\penv\Scripts\plat
formio.exe device monitor
Connecting to Blynk...
[1060] Connecting to NTU FSD
[9192] Connected to WiFi
[9193] IP: 10.13.43.123
[9193]
    ___  __          __
   / _ )/ /_ _____  / /__
  / _  / / // / _ \/  '_/
 /____/_/\_, /_//_/_/\_\
        /___/ v1.3.2 on ESP32

 #StandWithUkraine    https://bit.ly/swua
```



```
  #StandWithUkraine    https://bit.ly/swua


 [7683] Connecting to blynk.cloud:80
 [8606] Ready (ping: 404ms).
 Temp: 0.00 *C, Hum: 0.00 %
 Failed to read from DHT sensor!
 Temp: 0.00 *C, Hum: 0.00 %
```







- **Role of Template ID, Template Name, and Auth Token:**

Template ID: A unique identifier for the device template in the Blynk Cloud; links your ESP32 to the correct template structure and widgets.

Template Name: A readable name for the template in the Blynk Console; helps you identify the template easily.

Auth Token: A unique key for each device that authenticates it with the Blynk Cloud, allowing secure communication and data exchange.

- **Sensor configuration issues (DHT11 vs DHT22)**

Timing difference: DHT22 sends data faster and with higher resolution, while DHT11is slower

Accuracy and range: DHT11: Measures temperature 0–50°C, humidity 20–80%, with lower accuracy. **DHT22:** Measures temperature −40–80°C, humidity 0–100%, with higher accuracy

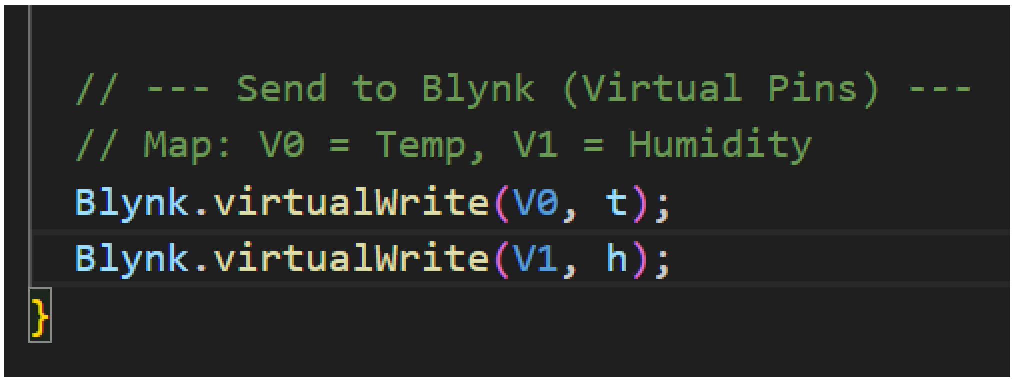
Code Compatibility: code written for DHT22 may not work correctly with DHT11 because of differences in data format and response time.

Sending data using Blynk.virtualWrite():

Blynk.virtualWrite() is used to send data from your ESP32 to a Blynk app widget via a virtual pin.

Blynk.virtualWrite(Vx, t);

Vx corresponds to virtual pin(V0,V1) and 't' means value/data that has to send.



```
  // --- Send to Blynk (Virtual Pins) ---
  // Map: V0 = Temp, V1 = Humidity
  Blynk.virtualWrite(V0, t);
  Blynk.virtualWrite(V1, h);
}
```

Common problems faced during configuration and their solutions:

- Common problems are it does not connect to Wi-Fi: correct SSID and password
- Device cannot recognize: Ensure Template ID
- Do not connect errors: Ensure wiring
- Wrong sensor reading: correct wiring and working devices should be used.
