National Textile University, Faisalabad

Department of Computer Science

Name:	Hafsa Shafique
Class:	BSCS 5th _A
Registration No:	23-NTU-CS-1031
Course Name:	Embedded IoT
Submitted To:	Sir Nasir
Submission Date:	21/12/2025

Home Task:

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?

The line of code “WebServer server(80);” is employed to make a web server on the ESP32 to listen to incoming HTTP requests. The number 80 stands to indicate the default communication port of a website. This implies whenever a user visits a website with the IP address of the ESP32, the internet request will be received on this particular port automatically.

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

The "server.on('/', handleRoot);" line specifies how the server should handle requests when the root URL is navigated. It associates the request for the main web page with the "handleRoot()" function such that every time a user accesses the IP address of the ESP32 board with a web browser, the "handleRoot()" function is triggered.

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

The reason why the `server.handleClient();` function call is embedded in the `loop()` function is to allow the ESP32 to listen to client requests repeatedly. Removing this line will cause the server to stop listening to browser requests, preventing the webpage from loading successfully.

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

Inside the `handleRoot()` function, the line `server.send(200, "text/html", html);` sends the response to the client. The code 200 signifies that the request was successful, while text/html signifies that the retrieved result is of html format, and the html variable holds the webpage details.

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

Showing the last measured sensor readings means that the webpage displays readings that had been already measured and recorded, and this enhances responsiveness and accuracy. Reading the new DHT sensor reading within the `handleRoot()` function means that the sensor is read each time the webpage is refreshed, resulting in delays and inaccuracies due to the minimum time interval between readings for DHT sensors.

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?**

   Blynk Template ID: The Blynk Template ID represents a unique identifier for the device template that has been generated within the Blynk Cloud system. This defines interactions between the ESP32 device and its corresponding dashboard, widgets, and data organization structure on Blynk Cloud.

2. Difference between Blynk Template ID and Blynk Auth Token

The Template ID represents the type or structure of the device used by Blynk, such as widgets, virtual pins, or others

An Auth Token is an exclusive security token that is given to each device separately and is used by the device for authentications and secure communications with the Blynk server

3. Why DHT22 code is giving inaccurate results with a DHT11 sensor

The improper use of code from a DHT22 sensor on a DHT11 sensor leads to inaccurate information, considering that both sensors read information based on different formats and scales.

In one of the main differences, DHT22 offers more accuracy and a larger range, but DHT11 supports lower accuracy and range. Because both sensors process data in their own formulas, incorrect code results in incorrect values.

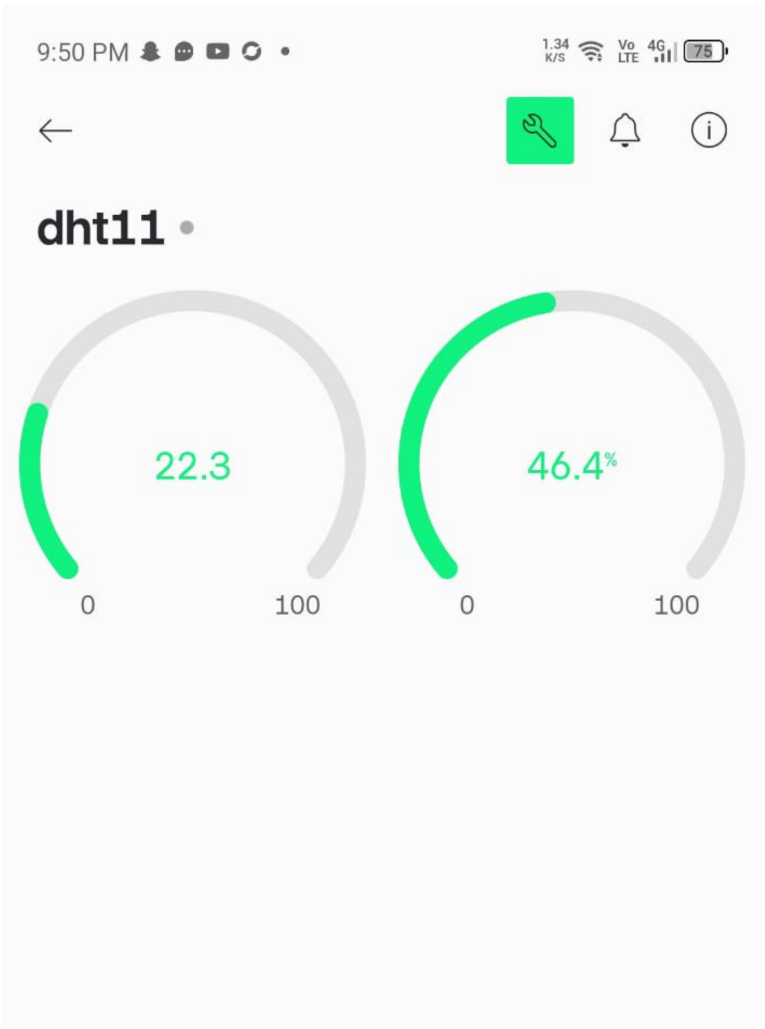
4. **Virtual Pins in Blynk and their significance Virtual Pins:**

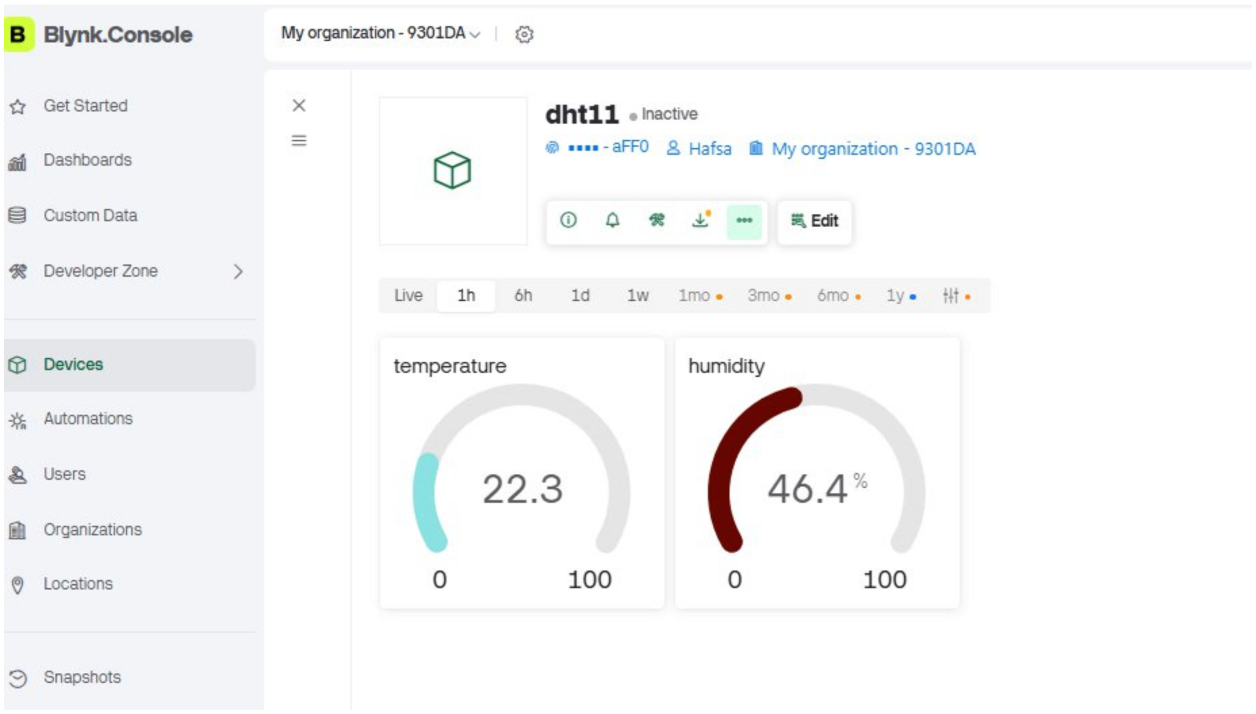
They are software pins which enable the transfer of data from the ESP32 to the Blynk Cloud and vice versa. They are not physical pins. These are preferred over the other types because they enable flexible communication on the cloud without relying on the restrictions imposed by the GPIO layout on the microcontroller hardware.

5. **Reason for Using BlynkTimer Instead of delay()**

The reason It provides a facility for timers using BlynkTimer for performing timed tasks without suspending the program. However, delay() prevents the ESP32 from performing many other critical tasks, such as WLAN communication and cloud update. BlynkTimer ensures smooth multitasking, better performance, and network connection in Internet of Things projects.

Part-B: Long Question





1. A Blynk Template is first created on the Blynk Cloud to lay down the structure of the device. Data streams for temperature and humidity are created and assigned to virtual Pins, which are then linked to widgets on the Blynk Dashboard.

2. It refers to the linking of the ESP32 firmware to the proper cloud template and should be named the same as the template developed in Blynk.

The ‘Template ID’ is only a naming convention of the template in the dashboard.

The ‘Auth Token’ is a token specific to the ESP32, and it is a separate token developed and is required to communicate with the Blynk cloud securely.

3. It is necessary to correctly use the temperature-humidity sensor in code. There are differences in accuracy, scale, and data representation between a DHT11 temperature-humidity sensor and a DHT22 temperature-humidity sensor.

4. Data readings from temperatures and humidity are performed by ESP32 and then uploaded to the Blynk cloud via `Blynk.virtualWrite()` function calls. They are uploaded to each respective virtual pin, thus updating real-time displays on the widget.

5. BlynkTimer helps send sensor data periodically without lagging the execution of programs. This ensures precise Wi-Fi and cloud connectivity.

6. Typical errors: entering a Template ID/Auth Token incorrectly, entering Wi-Fi details incorrectly, corresponding errors in virtual pins and datastreams, and connectivity mistakes involving sensors. These are corrected by checking credentials for validity, aligning datastreams and virtual pins, choosing a sensor correctly, and checking connectivity.