

In-class Case Study 1

Instructor: Qasim Ali

Implementing and Demonstrating the IoT System

Group Name: Group B

Student Name: Govind Ram Gupta Belde

Objective

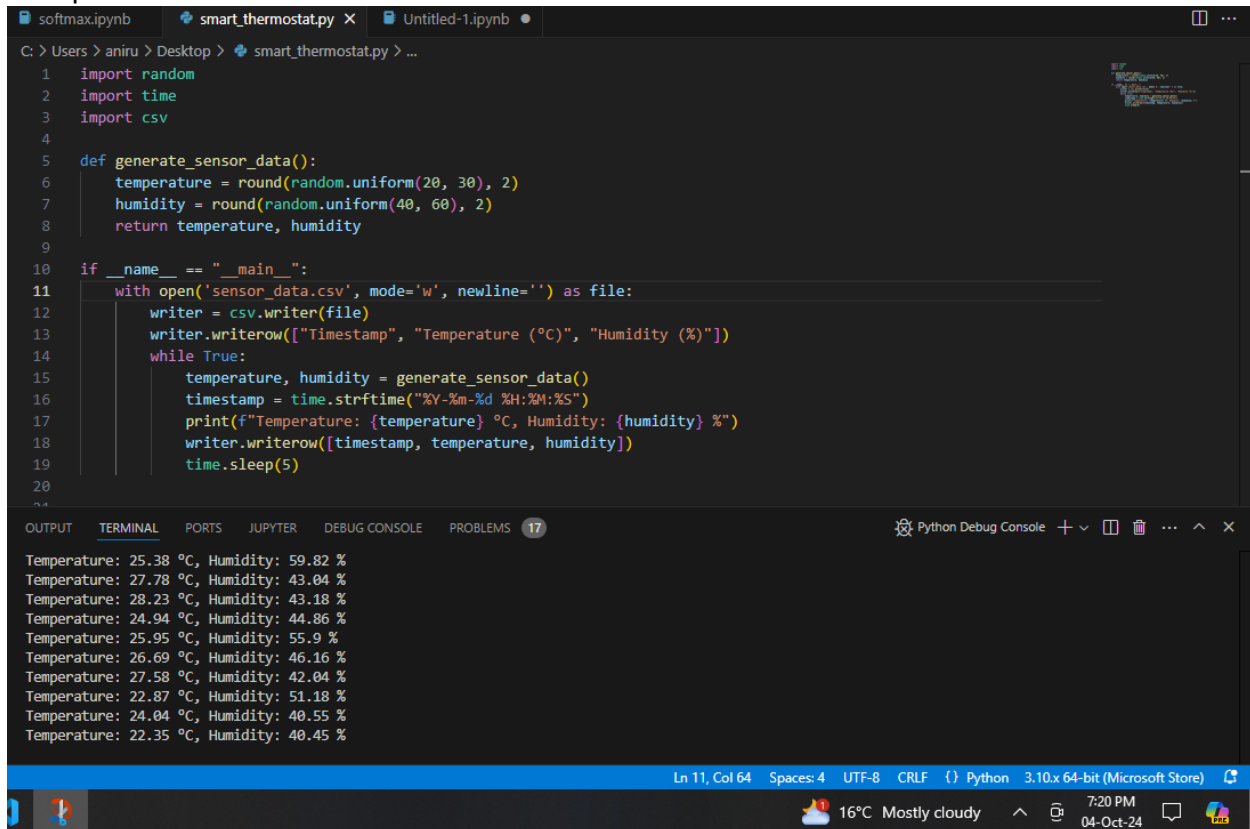
The objective of this assignment is to practically implement the designed IoT system using real or simulated IoT devices, and demonstrate data collection, processing, and visualization.

1. Implementation of IoT Devices

Device 1: Smart Thermostat

- Implementation: Create a Python script (`smart_thermostat.py`) that generates random temperature and humidity values.

Output:



```

C: > Users > aniru > Desktop > smart_thermostat.py > ...
1 import random
2 import time
3 import csv
4
5 def generate_sensor_data():
6     temperature = round(random.uniform(20, 30), 2)
7     humidity = round(random.uniform(40, 60), 2)
8     return temperature, humidity
9
10 if __name__ == "__main__":
11     with open('sensor_data.csv', mode='w', newline='') as file:
12         writer = csv.writer(file)
13         writer.writerow(["Timestamp", "Temperature (°C)", "Humidity (%)"])
14         while True:
15             temperature, humidity = generate_sensor_data()
16             timestamp = time.strftime("%Y-%m-%d %H:%M:%S")
17             print(f"Temperature: {temperature} °C, Humidity: {humidity} %")
18             writer.writerow([timestamp, temperature, humidity])
19             time.sleep(5)
20

```

OUTPUT

```

Temperature: 25.38 °C, Humidity: 59.82 %
Temperature: 27.78 °C, Humidity: 43.04 %
Temperature: 28.23 °C, Humidity: 43.18 %
Temperature: 24.94 °C, Humidity: 44.86 %
Temperature: 25.95 °C, Humidity: 55.9 %
Temperature: 26.69 °C, Humidity: 46.16 %
Temperature: 27.58 °C, Humidity: 42.04 %
Temperature: 22.87 °C, Humidity: 51.18 %
Temperature: 24.04 °C, Humidity: 40.55 %
Temperature: 22.35 °C, Humidity: 40.45 %

```

Ln 11, Col 64 Spaces: 4 UTF-8 CRLF {} Python 3.10.x 64-bit (Microsoft Store)

16°C Mostly cloudy 7:20 PM 04-Oct-24

Device 2: Smart Light Bulb

- Implementation: Use a Raspberry Pi with a connected LED. Write a Python script (`smart_light.py`) to control the LED.

AISC2010 – Programming and Deployment of IOT Devices

```
C: > Users > aniru > Desktop > samrt_light.py > ...
1 import Mock.GPIO as GPIO # Use Mock GPIO for non-RPi environment
2 import time
3
4 # GPIO pin connected to the LED
5 LED_PIN = 18
6
7 # Set up GPIO mode and the LED pin
8 GPIO.setmode(GPIO.BCM)
9 GPIO.setup(LED_PIN, GPIO.OUT)
10
11 def control_light(state):
12     """Turn the light ON or OFF by controlling the LED."""
13     GPIO.output(LED_PIN, state)
14
15 if __name__ == "__main__":
16     try:
17         while True:
18             # Example: Simulating light on/off cycles
19             control_light(True) # Turn on the light
20             print("Light On")
21             time.sleep(5) # Keep it on for 5 seconds
22             control_light(False) # Turn off the light
23             print("Light Off")
24             time.sleep(5) # Keep it off for 5 seconds
25     except KeyboardInterrupt:
```

OUTPUT TERMINAL PORTS DEBUG CONSOLE PROBLEMS (20)

Light On
Light Off
Light On
Light Off
Light On

Ln 10, Col 1 Spaces: 4 UTF-8 CRLF () Python 3.10.11 64-bit (Microsoft Store)

Device 3: Smart Door Lock

- Implementation: Use a Raspberry Pi with a servo motor. Write a Python script (`smart_lock.py`) to control the servo motor.

```
C: > Users > aniru > Desktop > SmartDoorLock.py > ...
1 import Mock.GPIO as GPIO # Use Mock GPIO for non-RPi environment
2 import time
3
4 # GPIO pin connected to the servo motor
5 SERVO_PIN = 17
6
7 # Set up GPIO mode and the servo pin
8 GPIO.setmode(GPIO.BCM)
9 GPIO.setup(SERVO_PIN, GPIO.OUT)
10
11 # Set PWM frequency to 50Hz (standard for servos)
12 pwm = GPIO.PWM(SERVO_PIN, 50)
13
14 def control_lock(lock):
15     """Lock or unlock the door by controlling the servo."""
16     pwm.start(7.5) # Initialize the servo position
17     if lock:
18         # Position the servo to 0 degrees (locked position)
19         pwm.ChangeDutyCycle(2.5) # Adjust based on your servo's needs
20         print("Door Locked")
21     else:
22         # Position the servo to 90 degrees (unlocked position)
23         pwm.ChangeDutyCycle(12.5) # Adjust based on your servo's needs
24         print("Door Unlocked")
25     time.sleep(1) # Allow the servo to reach its position
```

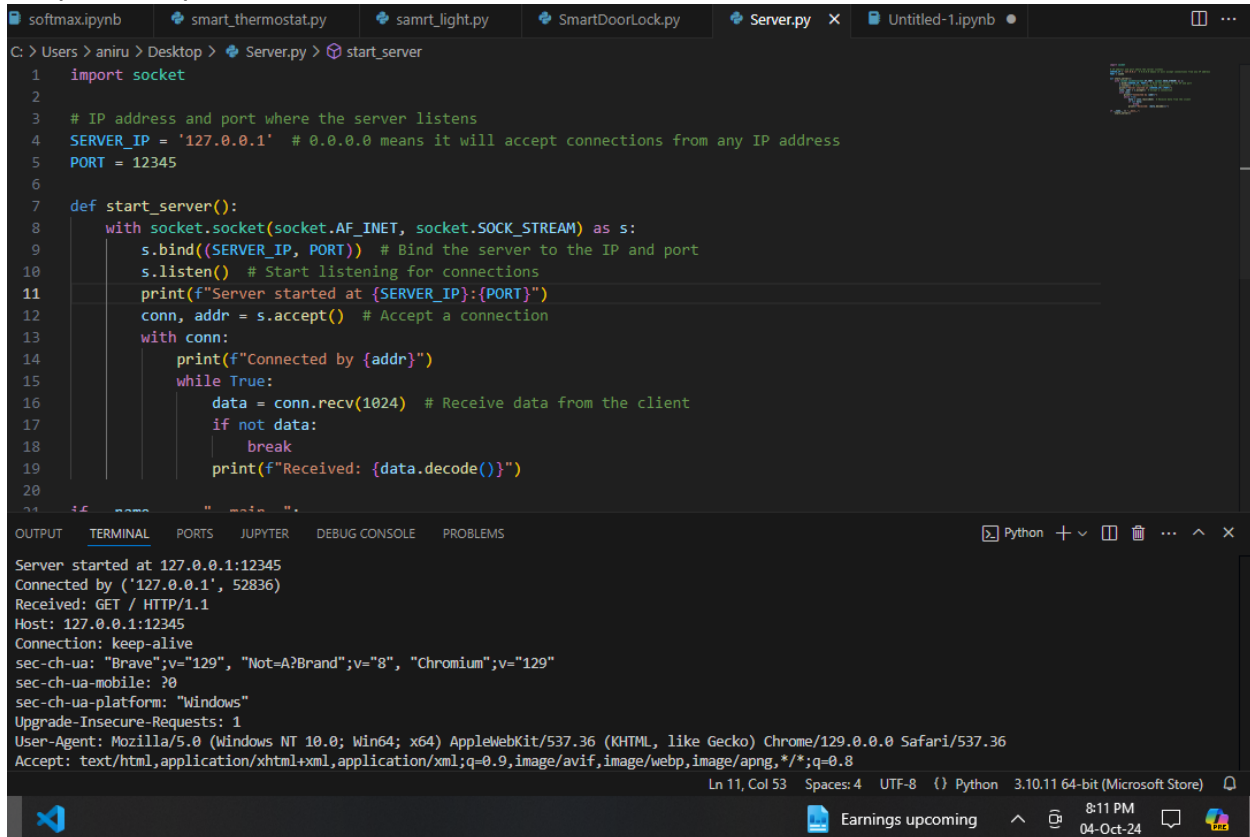
OUTPUT TERMINAL PORTS DEBUG CONSOLE PROBLEMS

Door Unlocked
Door Locked
Door Unlocked
Door Locked
Door Unlocked

Ln 41, Col 19 Spaces: 4 UTF-8 () Python 3.10.11 64-bit (Microsoft Store)

2. Network Connectivity

- Setup: Use Python libraries like `socket` to implement network connectivity. Here's a simple example for socket communication:



```

C:\Users\aniru\Desktop> Server.py > start_server
1  import socket
2
3  # IP address and port where the server listens
4  SERVER_IP = '127.0.0.1' # 0.0.0.0 means it will accept connections from any IP address
5  PORT = 12345
6
7  def start_server():
8      with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
9          s.bind((SERVER_IP, PORT)) # Bind the server to the IP and port
10         s.listen() # Start listening for connections
11         print(f"Server started at {SERVER_IP}:{PORT}")
12         conn, addr = s.accept() # Accept a connection
13         with conn:
14             print(f"Connected by {addr}")
15             while True:
16                 data = conn.recv(1024) # Receive data from the client
17                 if not data:
18                     break
19                 print(f"Received: {data.decode()}")
20
21 if __name__ == "__main__":
22     start_server()

```

OUTPUT TERMINAL PORTS JUPYTER DEBUG CONSOLE PROBLEMS

```

Server started at 127.0.0.1:12345
Connected by ('127.0.0.1', 52836)
Received: GET / HTTP/1.1
Host: 127.0.0.1:12345
Connection: keep-alive
sec-ch-ua: "Brave";v="129", "Not=A?Brand";v="8", "Chromium";v="129"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/129.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8

```

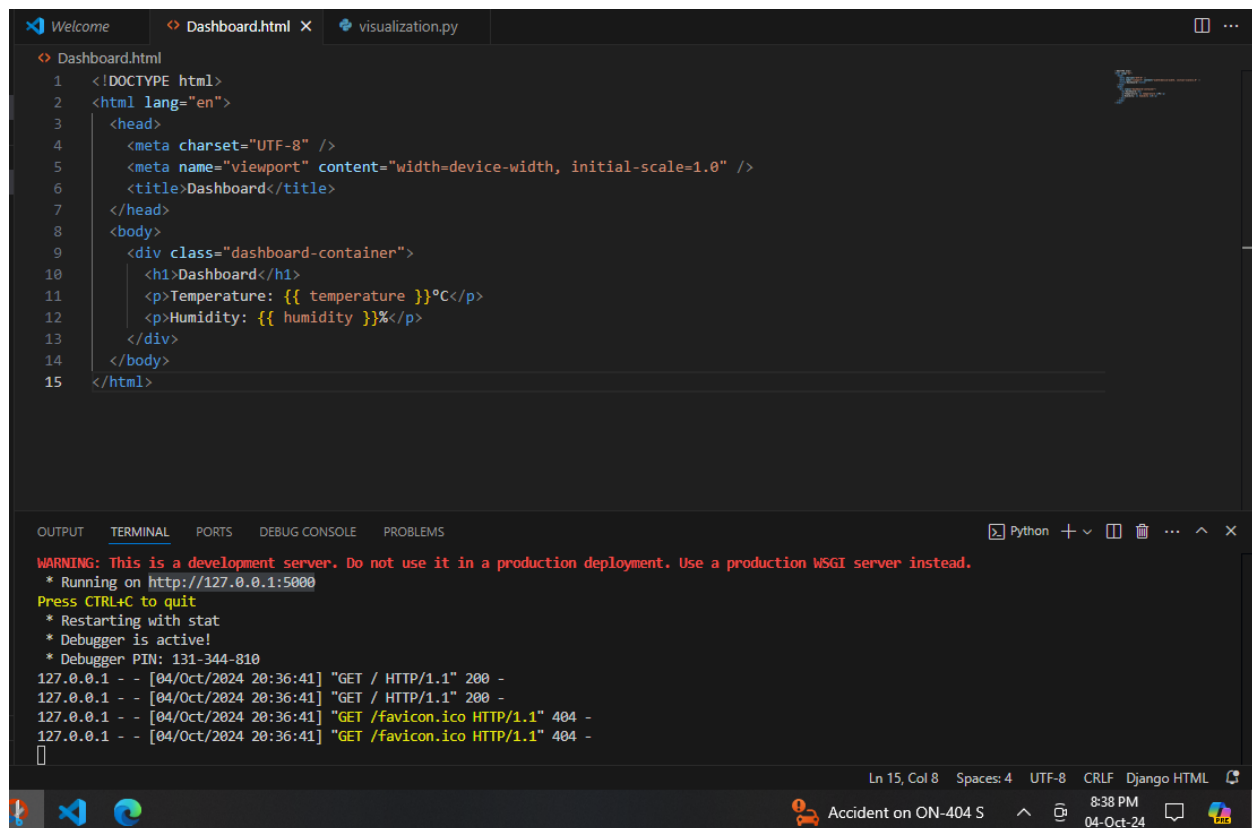
Ln 11, Col 53 Spaces: 4 UTF-8 Python 3.10.11 64-bit (Microsoft Store)

Earnings upcoming 8:11 PM 04-Oct-24

3. Data Collection and Processing

- Data Collection: Modify the device scripts to send data over MQTT using Python libraries like `paho-mqtt`.
- Data Processing: Use Python libraries like `pandas` or `numpy` to process the collected data locally or on a cloud platform.

AISC2010 – Programming and Deployment of IOT Devices



The screenshot shows a code editor with two tabs: 'Dashboard.html' and 'visualization.py'. The 'Dashboard.html' tab is active, displaying the following HTML code:

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <title>Dashboard</title>
7   </head>
8   <body>
9     <div class="dashboard-container">
10      <h1>Dashboard</h1>
11      <p>Temperature: {{ temperature }}°C</p>
12      <p>Humidity: {{ humidity }}%</p>
13    </div>
14  </body>
15 </html>
```

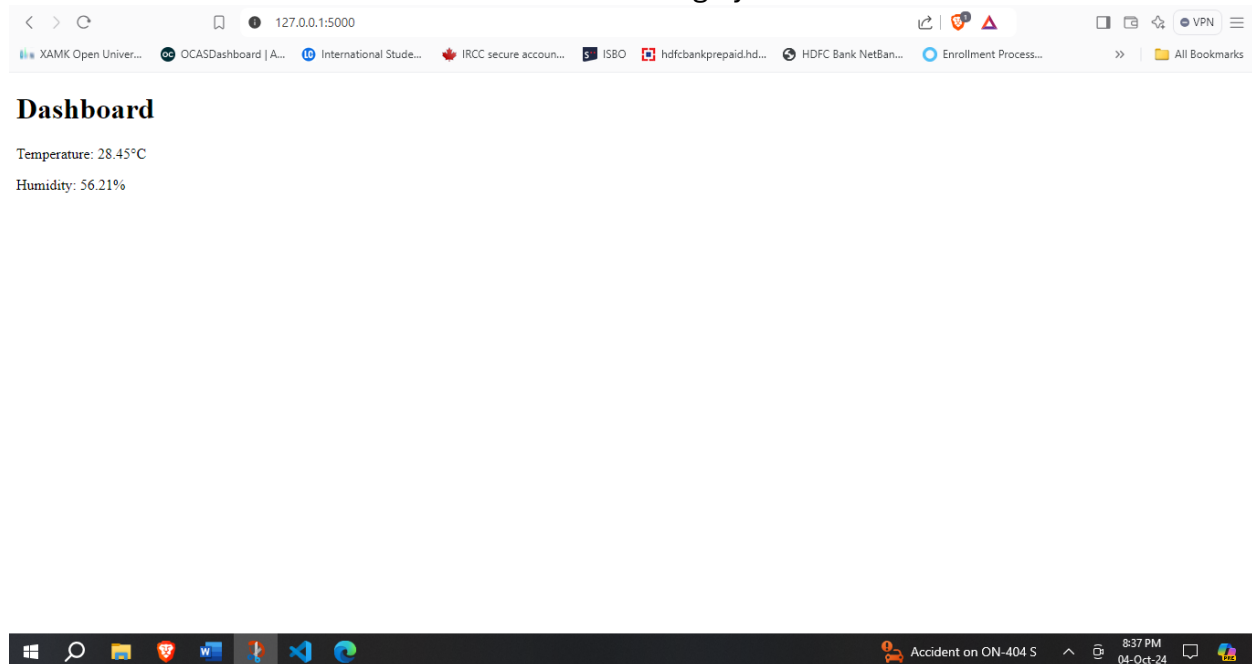
Below the code editor is a terminal window with the following output:

```
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 131-344-810
127.0.0.1 - - [04/Oct/2024 20:36:41] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Oct/2024 20:36:41] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Oct/2024 20:36:41] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [04/Oct/2024 20:36:41] "GET /favicon.ico HTTP/1.1" 404 -
```

The terminal window also shows the status bar at the bottom: 'Ln 15, Col 8 Spaces: 4 UTF-8 CRLF Django HTML'.

4. Data Visualization

- Dashboard: Create a web-based dashboard using Python web frameworks like Flask or



5. Documentation and Demonstration

1. Creating a Web-Based Dashboard with Flask

We built a simple web-based dashboard using **Flask** to display dynamic temperature and humidity values. Flask was used to create a web server, and an HTML file was used as a template to render the data dynamically.

- We created a project structure with a templates folder to store HTML files, and a Python file (app.py) to handle the web server and routing.
- The Flask app uses the random.uniform() function to generate random temperature and humidity values, and it uses render_template() to pass the values to the HTML file.
- In the templates folder, we created dashboard.html to display the data using **Jinja2** syntax ({{ temperature }}, {{ humidity }}).
- We ran the Flask server locally, which allowed us to view the dashboard at <http://127.0.0.1:5000/>.

2. Setting Up MQTT Communication with paho-mqtt

We extended the project by introducing **MQTT** for device communication using the **paho-mqtt** Python library. This setup allows the devices to send sensor data (temperature and humidity) to a broker.

- We created a client that publishes sensor data to an MQTT broker (e.g., HiveMQ).
- The device sends data to a specific topic (iot/smart_thermostat) at regular intervals.
- Another script subscribes to the MQTT topic and collects the data for further processing.

3. Data Processing with Pandas and NumPy

We used **Pandas** and **NumPy** to process the data collected via MQTT.

- The collected sensor data is saved to a CSV file using Pandas.
- We used Pandas to generate basic statistics for temperature and humidity and visualized the data with **matplotlib**.