**College code:9512**

**College name:JP COLLEGE OF ENGINEERING**

**Department : ECE**

**Project\_code:Proj\_211934\_Team\_1**

**ENVIRONMENT MONITORING SYSTEM**

**T**EAM MEMBERS :

1. C. HEMAPRIYA (951221106013)
2. K. MUTHURAMA (951221106027)
3. N. MURUGESHWARI (951221106026)
4. A. ANISHA PUSHPAM(951221106004)
5. N. RAMYA (951221106032)

**Phase3**: IOT devices and then Developing a Python script on the IoT devices as per the project requirement.

1. Hardware Selection: Choose suitable IoT devices (e.g., temperature and humidity sensors) and microcontrollers (e.g., Raspberry Pi, Arduino) for your project.

2. \*Sensor Placement\*: Identify strategic locations within the parks for sensor deployment to collect relevant environmental data.

3. \*Connectivity\*: Ensure that the devices have access to the internet, either through Wi-Fi or cellular networks.

4. \*Programming\*: Develop a Python script for the IoT devices. You'll need to program the devices to read data from sensors and transmit it to a central monitoring platform. Consider using libraries like Adafruit IO, MQTT, or HTTP requests to send the data.

5. \*Monitoring Platform\*: Set up a monitoring platform that can receive and store the incoming data. This platform can be a cloud-based server or a local server, depending on your project's scale.

6. \*Data Storage\*: Decide how you want to store the data. You can use databases (e.g., MySQL, MongoDB) or cloud storage solutions (e.g., AWS S3, Google Cloud Storage).

7. \*Real-Time Data Processing\*: Implement real-time data processing to analyze and visualize the data. Tools like Grafana, InfluxDB, or custom web applications can be useful for creating dashboards.

8. \*Alerting System\*: Set up alerts based on predefined thresholds for environmental conditions. This can be done using notification services or email alerts.

9. \*Power Supply\*: Ensure that the IoT devices have a stable power supply. You might need to use batteries, solar panels, or a combination of power sources.

10. \*Security\*: Implement security measures to protect the devices and the data they transmit. Use encryption and access controls.

11. \*Scalability\*: Plan for the scalability of your system, especially if you intend to monitor multiple parks or expand the sensor network.

12. \*Testing and Calibration\*: Rigorously test the IoT devices, sensors, and the entire system. Calibrate the sensors if needed to ensure data accuracy.

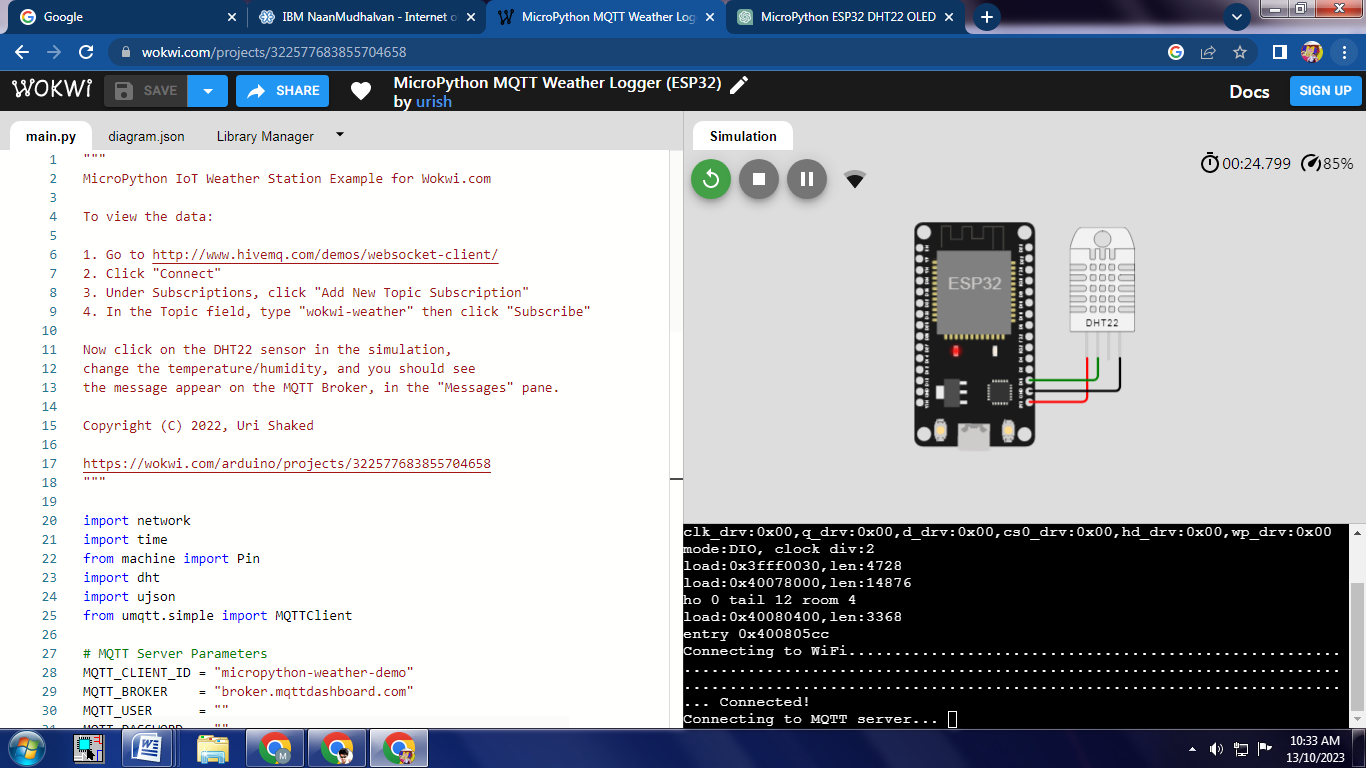
13. \*Maintenance and Updates\*: Create a plan for device maintenance and software updates, as IoT devices require ongoing care.

14. \*Compliance and Regulations\*: Ensure compliance with local regulations and privacy laws when collecting and storing environmental data.

15. \*User Interface\*: If the system is for public use, consider developing a user interface or mobile app for park visitors to access environmental data.

Below is a high-level outline of how you can approach this task

Connecting from micropython

To connect from a [MicroPython project](https://wokwi.com/projects/new/micropython-esp32" \t "_blank), use the following code:Once connected, you can use the [urequests library](https://mpython.readthedocs.io/en/master/library/mPython/urequests.html" \t "_blank) to send HTTP and HTTPS requests, and the [umqtt library](https://mpython.readthedocs.io/en/master/library/mPython/umqtt.simple.html" \t "_blank) to establish MQTT connections

**PYTHON SUBSCRIPT:**

import network

import time

From machine import Pin

Import dht

Import ujson

From umqtt.simple import MQTTClient

# MQTT Server Parameters

MQTT\_CLIENT\_ID = "micropython-weather-demo"

MQTT\_BROKER    = "broker.mqttdashboard.com"

MQTT\_USER      = ""

MQTT\_PASSWORD  = ""

MQTT\_TOPIC     = "wokwi-weather"

sensor = dht.DHT22(Pin(15))

print ("Connecting to Wi-Fi", end="")

sta\_if = network.WLAN(network.STA\_IF)

sta\_if.active(True)

sta\_if.connect('Wokwi-GUEST', '')

while not sta\_if.isconnected():

print(".", end="")

time.sleep(0.1)

print(" Connected!")

print("Connecting to MQTT server... ", end="")

client = MQTTClient (MQTT\_CLIENT\_ID, MQTT\_BROKER, user=MQTT\_USER, password=MQTT\_PASSWORD)

client.connect()

print ("Connected!")

prev\_weather = ""

while True:

Print ("Measuring weather conditions... ", end="")

sensor.measure()

message = ujson.dumps({

"temp": sensor. temperature (),

"Humidity": sensor. Humidity (),

})

if message != prev\_weather:

print("Updated!")

print("Reporting to MQTT topic {}: {}".format(MQTT\_TOPIC, message))

client.publish(MQTT\_TOPIC, message)

prev\_weather = message

else:

print("No change")

time.sleep(1)

**simulation output:**

(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:4728

load:0x40078000,len:14876

ho 0 tail 12 room 4

load:0x40080400,len:3368

entry 0x400805cc

Connecting to WiFi...................................................................................................................................................................................................................................................................................................................................................................................................................................................................................... Connected!

Connecting to MQTT server... Connected!

Measuring weather conditions... Updated!

Reporting to MQTT topic wokwi-weather: {"humidity": 40.0, "temp": 24.0}

Measuring weather conditions... No change

Measuring weather conditions... No change

Measuring weather conditions... No change

Measuring weather conditions... No change

Measuring weather conditions... No change

Measuring weather conditions... Updated!

Reporting to MQTT topic wokwi-weather: {"humidity": 80.5, "temp": 48.8}

Measuring weather conditions... Updated!

Reporting to MQTT topic wokwi-weather: {"humidity": 80.5, "temp": -13.8}

Traceback (most recent call last):

File "main.py", line 62, in <module>

File "umqtt/simple.py", line 134, in publish

OSError: [Errno 104] ECONNRESET

MicroPython v1.21.0 on 2023-10-05; Generic ESP32 module with ESP32

Type "help ()" for more information

**1.Arduino UNO:**

Similar to the ESP32, consider using a default naming scheme with unique identifiers if you have multiple device

2**. Sensor Naming**:

**DHT22 Sensor:**

Use a naming convention like “DHT22\_Park\_Sensor1” to specify the type of sensor, location (Park), and a unique identifier if you have multiple sensors.

**3. Connectivity Naming**:

**BLE:**

If you have multiple BLE devices, use clear and descriptive names, such as “BLE\_Device\_1.”

**Wi-Fi**:

It’s advisable to name Wi-Fi access points or devices with descriptive names based on their location or function in the park.

Wokwi simulates a WiFi network with full internet access. You can use the [ESP32](https://docs.wokwi.com/guides/esp32) together with the virtual WiFi to prototype IoT projects. Common use cases include:

* Connect to MQTT servers to send sensor data
* Query web services over HTTP, HTTPS, and web sockets
* Run an HTTP server inside the ESP32 and connect to it from your browser (requires the [Wokwi IoT Gateway](https://docs.wokwi.com/guides/esp32-wifi" \l "the-private-gateway))

**Connecting to the Wi-Fi**[**​**](https://docs.wokwi.com/guides/esp32-wifi#connecting-to-the-wifi)

The simulator provides a virtual Wi-Fi access point called **Wokwi-GUEST**.

It is an open access point - no password is required.

**Zigbee:**

If you have multiple Zigbee devices, use clear and descriptive names, such as “Zigbee\_Node\_Park\_Area.”

**4. Cloud Naming:**

**Beeceptor:**

You can keep the name **“Beeceptor**” as-is for this development and testing tool.

5**. Protocol Naming**:

**MQTT:**

Use standard MQTT topics and naming conventions for publishing and subscribing to data.

**HTTP:**

Specify clear and consistent API endpoints and URL paths.

**AMQP:**

Use well-defined queues and exchanges with appropriate names for AMQP communication.

6. **Data Naming**:

For data collected, follow a standardized naming format, such as “Data\_Temperature\_Humidity\_Park\_Location” to ensure data consistency.