**Project: Environment monitoring in park**

**Project definition:**

Monitoring the environment in a park using IoT (Internet of Things) technology can provide real-time

data and insights, making it more efficient and effective.

**Air Quality Monitoring**:

Deploy IoT sensors to measure air quality parameters such as particulate matter (PM2.5 and PM10),

ozone (O3), nitrogen dioxide (NO2), and carbon monoxide (CO).

Transmit data from sensors to a central system via wireless networks.

Analyze the data in real-time and provide alerts or warnings to park authorities and visitors when air

quality reaches unhealthy levels

**Water Quality Testing:**

InStall IoT-enabled water quality sensors in various water bodies within the park.

These sensors can continuously measure parameters like pH, turbidity, dissolved oxygen, and

temperature.

Transmit data wirelessly to a central monitoring station.

Set up automated alerts for deviations from acceptable water quality standards.

**Temperature and Humidity Monitoring**:

Deploy IoT temperature and humidity sensors at different locations in the park.

Collect and transmit data in real-time to monitor temperature variations and humidity levels.

Use this data to predict weather changes and assess their impact on the park environment.

**Biodiversity Surveys:**

Use IoT-enabled cameras and acoustic sensors to monitor wildlife and record sounds.

- Implement machine learning algorithms to identify and classify species based on images and audio

data.

- Gather valuable information on the park’s biodiversity and ecosystem health.

**Noise Pollution Monitoring:**

Place IoT noise sensors at strategic locations to continuously measure noise levels.

- Transmit data

wirelessly to a central database.

- Generate noise pollution reports and visualize trends over time.

**Weather Monitoring:**

Deploy IoT weather stations to measure parameters like precipitation, wind speed, and temperature.

- Transmit data to a central weather database for analysis.

- Provide real-time weather updates to park authorities and visitors.

**Soil Health Assessment:**

Install soil moisture and nutrient sensors in different areas of the park.

- Use IoT technology to collect and transmit soil data, including moisture levels and nutrient content.

- Monitor soil health and provide recommendations for park maintenance and conservation efforts.

**Remote Sensing and GIS:**

Combine IoT data with Geographic Information Systems (GIS) to create maps and visualizations of

environmental conditions.

- Monitor changes in land use, vegetation, and ecological features over time.

**Visitor Impact Studies:**

Use IoT-connected trail counters, waste bins, and visitor counters to track visitor impact.

- Collect data on trail usage, waste generation, and visitor density.

- Optimize park management and develop strategies to minimize environmental impact.

**Early Warning Systems:**

Implement IoT-based early warning systems for natural disasters, such as wildfire detection sensors and

flood level sensors.

- Send real-time alerts to park authorities and emergency services when potential threats are detected.

**Energy Efficiency:**

Implement IoT-based early warning systems for natural disasters, such as wildfire detection sensors and

flood level sensors.

- IoT to optimize energy consumption in park facilities, such as lighting and HVAC systems, by

adjusting them based on occupancy and weather conditions.

IoT-enabled environmental monitoring not only provides valuable data for park management but also

enhances visitor safety and engagement by offering real-time information about park conditions. It can

play a significant role in preserving the natural beauty and ecological balance of the park while

promoting sustainable practices.

**Abstract:**

Air quality, water pollution, and radiation pollution are major factors that pose challenges in the

environment. Suitable monitoring is necessary so that the world can sustainable growth, by maintaining

a healthy society. In recent years, the environment monitoring has turned into a smart environment

monitoring (SEM) system, with the advances in the internet of things (IoT) and the development of

modern sensors. Under this scenario, the present mar aims to accomplish a critical review of

noteworthy contributions and research studies c that involve monitoring of air quality, water quality,

radiation pollution, and agricultures The review is divided on the basis of the purposes where SEM

methods are applied, and the purpose is further analyzed in terms of the sensors used, machine learning

techniques in and classification methods used. The detailed analysis follows the extensive review with

suggested major recommendations and impacts of SEM research on the basis of discussion res research

trends analyzed. The authors have critically studied how the advances in sensor ted IoT and machine

learning methods make environment monitoring a truly smart monitoring Finally, the framework of

robust methods of machine learning; denoising methods and devel of suitable standards for wireless

sensor networks (WSNs), has been suggested.

For an environmental monitoring project in a park using ESP32 and Arduino UNO microcontrollers,

here’s a more detailed breakdown:

**Microcontrollers:**

**1. ESP32:**

As previously mentioned, the ESP32 is a powerful microcontroller with built-in Wi-Fi and

Bluetooth capabilities, making it suitable for IoT projects. It can collect data and send it to a

cloud platform.

**2. Arduino UNO:**

The Arduino UNO is a simple microcontroller suitable for basic tasks. In this project, it can be

used for simple data collection tasks or as a backup device.

**Sensor:**

For monitoring temperature and humidity in an outdoor park environment, you can use a sensor like the

DHT22 or DHT11, as previously described in the technical specs section.

**Connectivity:**

You can utilize multiple connectivity options for redundancy and flexibility:

**BLE (Bluetooth Low Energy):**

This can be used for short-range communication, such as configuring or troubleshooting devices

in the park.

**Wi-Fi:**

Use Wi-Fi to connect your devices to the internet and post data to a cloud platform. This allows

remote access to the data.

**Zigbee:**

If your park covers a large area, Zigbee can be useful for creating a mesh network for

communication between devices, especially in areas with limited Wi-Fi coverage.

**Cloud**:

You mentioned using Beeceptor for data handling and testing. While Beeceptor is a great choice for

development and testing, for a production environment, you might consider a more robust cloud

platform like AWS, Azure, Google Cloud, or a dedicated IoT platform to securely store and analyze your

data.

**Protocols:**

Your chosen protocols are suitable for IoT data communication:

**MQTT:**

Ideal for efficient and lightweight communication between devices and the cloud, especially

when dealing with intermittent connections or low bandwidth.

**HTTP:**

Useful for sending data to web servers or APIs. It’s a standard protocol for web communication.

**AMQP:**

A more advanced protocol suitable for complex IoT scenarios that require advanced message

handling and queuing.

**Microcontroller Naming:**

**ESP32:**

You can keep the default naming, but it’s advisable to add a unique identifier for each ESP32 device if

you have multiple.

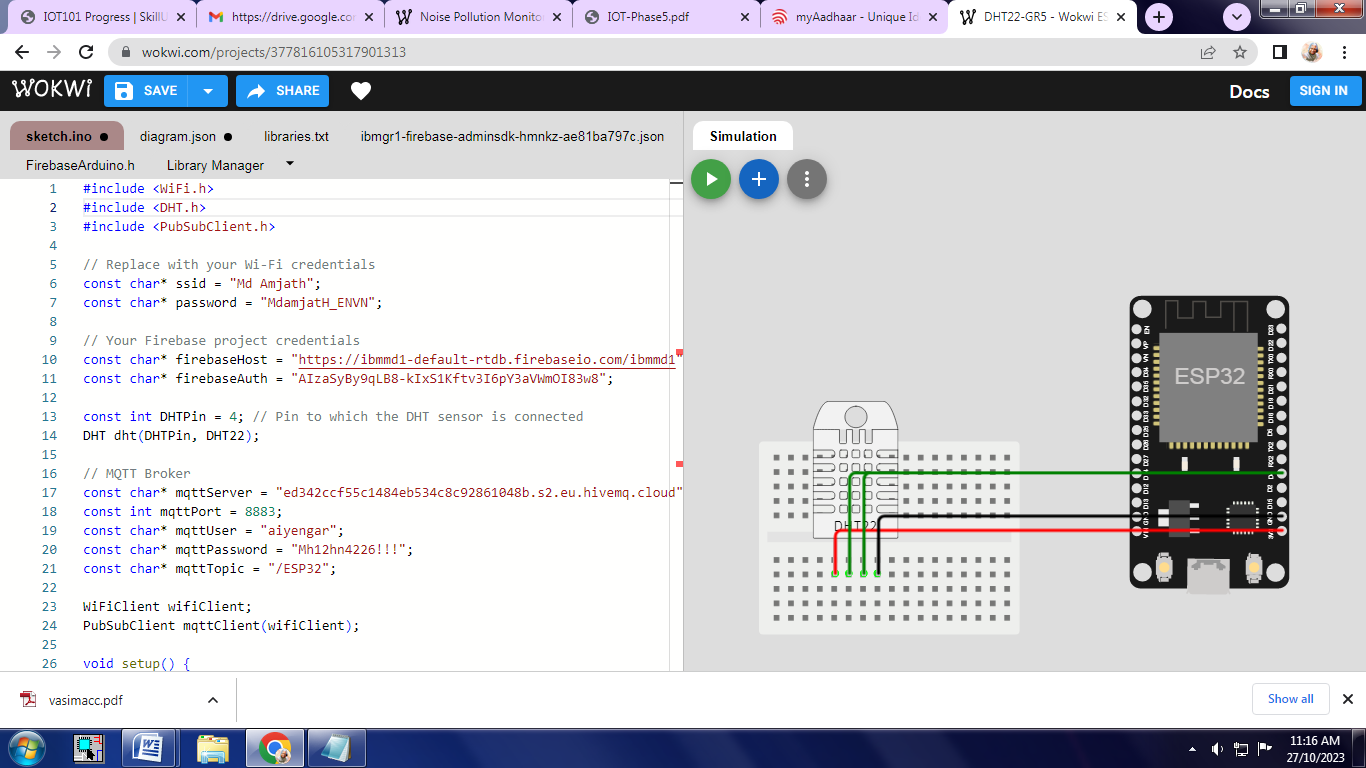
Connecting from MicroPython ​

To connect from a  MicroPython project , use the following code:Once connected, you

can use the  urequests library  to send HTTP and HTTPS requests, and the  umqtt

library  to establish MQTT connections

**Circuit Diagram:**



**Python Code:**

#include <WiFi.h>

#include <DHT.h>

#include <PubSubClient.h>

// Replace with your Wi-Fi credentials

const char\* ssid = "Md Amjath";

const char\* password = "MdamjatH\_ENVN";

// Your Firebase project credentials

const char\* firebaseHost = "https://ibmmd1-default-rtdb.firebaseio.com/ibmmd1";

const char\* firebaseAuth = "AIzaSyBy9qLB8-kIxS1Kftv3I6pY3aVWmOI83w8";

const int DHTPin = 4; // Pin to which the DHT sensor is connected

DHT dht(DHTPin, DHT22);

// MQTT Broker

const char\* mqttServer = "ed342ccf55c1484eb534c8c92861048b.s2.eu.hivemq.cloud"; // Replace with your MQTT broker address

const int mqttPort = 8883;

const char\* mqttUser = "Amjath.md";

const char\* mqttPassword = "Mdamjath786";

const char\* mqttTopic = "/ESP32";

WiFiClient wifiClient;

PubSubClient mqttClient(wifiClient);

void setup() {

**Serial**.begin(115200);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL\_CONNECTED) {

    delay(1000);

**Serial**.println("Connecting to WiFi...");

  }

**Serial**.println("Connected to WiFi");

  // Initialize Firebase

  Firebase.begin(firebaseHost, firebaseAuth);

  // MQTT setup

  mqttClient.setServer(mqttServer, mqttPort);

  mqttClient.setCallback(mqttCallback);

  // Connect to MQTT broker

  if (mqttClient.connect("ESP32Client", mqttUser, mqttPassword)) {

**Serial**.println("Connected to MQTT broker");

  }

}

void loop() {

  float humidity = dht.readHumidity();

  float temperature = dht.readTemperature();

  if (!isnan(humidity) && !isnan(temperature)) {

**Serial**.print("Temperature: ");

**Serial**.print(temperature);

**Serial**.print(" °C, Humidity: ");

**Serial**.print(humidity);

**Serial**.println(" %");

    // Firebase

    Firebase.pushFloat("/environment/temperature", temperature);

    Firebase.pushFloat("/environment/humidity", humidity);

    // MQTT publish

    String tempPayload = String(temperature);

    String humidityPayload = String(humidity);

    mqttClient.publish((String(mqttTopic) + "/temperature").c\_str(), tempPayload.c\_str());

    mqttClient.publish((String(mqttTopic) + "/humidity").c\_str(), humidityPayload.c\_str());

  } else {

**Serial**.println("Failed to read from DHT sensor");

  }

  mqttClient.loop();

  delay(60000); // Send data every 60 seconds

}

void mqttCallback(char\* topic, byte\* payload, unsigned int length) {

  // Handle MQTT subscription messages if needed

}

**Diagram Code:**

{

  "version": 1,

  "author": "Md Amjath",

  "editor": "wokwi",

  "parts": [

    { "type": "wokwi-breadboard-mini", "id": "bb1", "top": 56.2, "left": -136.8, "attrs": {} },

    { "type": "wokwi-esp32-devkit-v1", "id": "esp", "top": -43.3, "left": 110.2, "attrs": {} },

    { "type": "wokwi-dht22", "id": "dht1", "top": 29.1, "left": -101.4, "attrs": {} }

  ],

  "connections": [

    [ "esp:TX0", "$serialMonitor:RX", "", [] ],

    [ "esp:RX0", "$serialMonitor:TX", "", [] ],

    [ "dht1:VCC", "bb1:5b.g", "", [ "$bb" ] ],

    [ "dht1:SDA", "bb1:6b.g", "", [ "$bb" ] ],

    [ "dht1:NC", "bb1:7b.g", "", [ "$bb" ] ],

    [ "dht1:GND", "bb1:8b.g", "", [ "$bb" ] ],

    [ "dht1:VCC", "esp:3V3", "red", [ "v0" ] ],

    [ "dht1:GND", "esp:GND.1", "black", [ "v-38.4", "h268.8" ] ],

    [ "dht1:SDA", "esp:D4", "green", [ "v-67.2", "h288.1" ] ],

    [ "dht1:NC", "esp:D4", "green", [ "v-67.2", "h278.5" ] ]

  ],

  "dependencies": {}

}

**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: {&quot;humidity&quot;: 40.0, &quot;temp&quot;: 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: {&quot;humidity&quot;: 80.5, &quot;temp&quot;: 48.8}

Measuring weather conditions... Updated!

Reporting to MQTT topic wokwi-weather: {&quot;humidity&quot;: 80.5, &quot;temp&quot;: -13.8}

Traceback (most recent call last):

File &quot;main.py&quot;, line 62, in &lt;module&gt;

File &quot;umqtt/simple.py&quot;, line 134, in publish

OSError: [Errno 104] ECONNRESET

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

Type &quot;help ()&quot; for more information