Project Definition:

The project involves setting up IoT devices to monitor environmental conditions in public parks, including temperature and humidity. The primary objective is to provide real-time environmental data to park visitors through a public platform, enabling them to plan their outdoor activities accordingly. This project includes defining objectives, designing the IoT sensor system, developing the environmental monitoring platform, and integrating them using IoT technology and Python.

ENVIRONMENTEL MONITERING:

Environmental monitoring refers to the tools and techniques designed to observe an environment, characterize its quality, and establish environmental parameters, for the purpose of accurately quantifying the impact an activity has on an environment

OBJECTIVES 1.

Real-Time Environmental Monitoring:

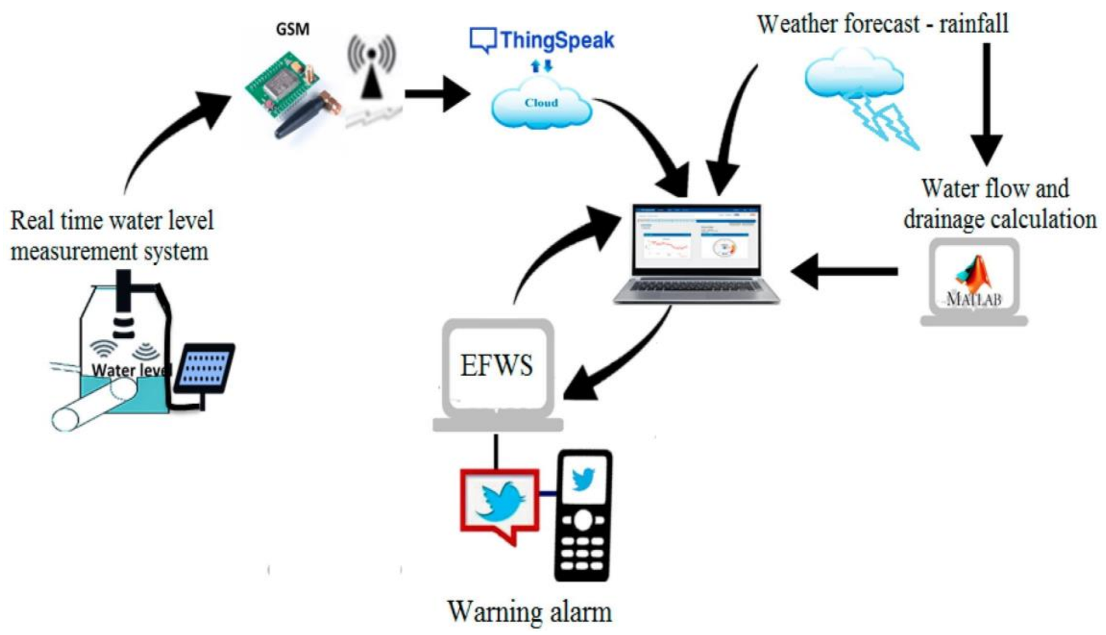
➢ Objective: Implement a system for real-time environmental monitoring within the park.

➢ **Key Results:**

o Install weather stations, air quality sensors, and water quality sensors at strategic locations.

o Develop a centralized monitoring dashboard accessible to park management for real-time data analysis.

o Alert park authorities and visitors about extreme weather conditions or environmental hazards


GSM
ThingSpeak
Cloud
Weather forecast - rainfall
Water flow and drainage calculation
MATLAB
Real time water level measurement system
Water level
EFWS
Warning alarm

2.Aiding Park Visitors in Activity Planning:

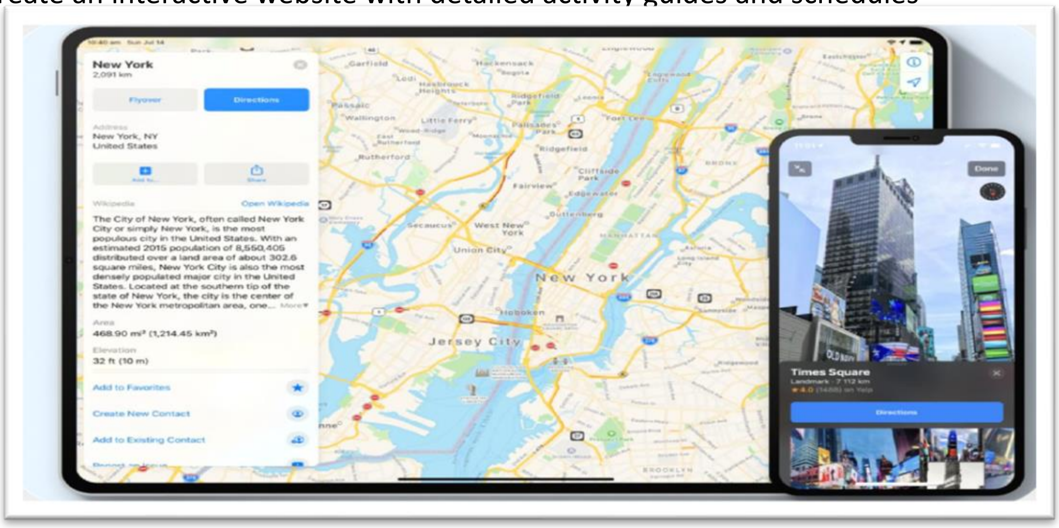
➢ **Objective:** Provide park visitors with tools and information to plan their activities effectively.

➢ **Key Results:**

• Develop a user-friendly mobile app with a park map, activity recommendations, and real-time updates.

• Integrate the app with GPS technology to help visitors navigate the park and locate amenities

. • Create an interactive website with detailed activity guides and schedules



3.Promoting Outdoor Experiences:

➢ **Objective**: Encourage park visitors to engage in outdoor activities and connect with nature.

➢ **Key Results:**

• Organize outdoor events such as guided hikes, bird-watching excursions, and stargazing nights.

• Establish designated outdoor education areas for workshops and nature-related programs.

• Install signage and information boards highlighting the park's natural features and history.

4.Enhancing Visitor Satisfaction:

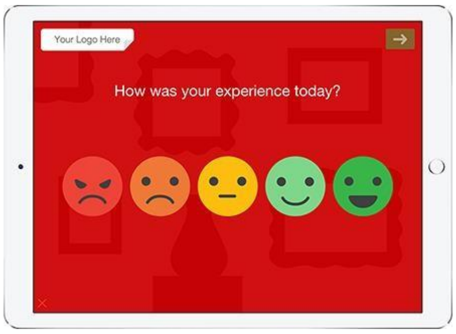
➢ **Objective**: Improve overall visitor satisfaction and park experience.

➢ **Key Results:**

• Collect visitor feedback through surveys, comment boxes, and online reviews.

• Use feedback to make improvements, such as adding more amenities, enhancing safety measures, and addressing concerns.

• Implement a responsive customer service system to address visitor inquiries and issues promptly.



5.Data-Driven Decision-Making:

➢ **Objective**: Utilize data collected from various sources to make informed decisions. ➢ Key Results:

• Analyse visitor data to identify peak times and popular activities, allowing for better resource allocation.

• Evaluate environmental data to develop strategies for conservation and sustainability.

• Collaborate with local businesses and organizations based on visitor trends and preferences

6.Community Engagement and Education:

➢ **Objective**:

Foster a sense of community and environmental stewardship among park visitors.

➢ **Key Results:**

• Establish partnerships with schools and community groups for educational programs and volunteer opportunities.

• Create interpretive signage and displays that educate visitors about the park's ecosystems and history.

• Host regular community meetings to gather input and involve residents in park planning.

Plan the deployment of IoT sensors in public parks

1.Define Objectives:

Identify the specific environmental parameters you want to monitor in the park, such as temperature, humidity, air quality, soil moisture, or noise levels.

2. Sensor Selection:

Choose appropriate IoT sensors for temperature and humidity monitoring. Consider factors like sensor accuracy, range, power source, and connectivity options (e.g., Wi-Fi, LoRa, cellular). Plan the deployment of IoT sensors in public parks

3. Sensor Placement:

Identify strategic locations within the park for sensor placement. Consider areas where environmental conditions are most critical or where visitor comfort is a priority.

4.Connectivity Infrastructure:

Establish a robust and reliable connectivity infrastructure to transmit data from sensors to a central data hub or cloud server. This may involve setting up Wi-Fi access

5. Power Supply:

•Determine the power source for the IoT sensors. Options include battery-powered sensors, solar panels, or wired connections.

•Consider the maintenance requirements associated with each power source option

6.Data Collection and Storage:

•Choose a data collection and storage solution that can handle the volume of sensor data. Cloud-based platforms are often suitable for scalability and data accessibility**.**

7. Data Visualization and Analysis:

•Develop a user-friendly interface or dashboard for visualizing real-time sensor data . Implement data analysis tools to identify trends and anomalies in the collected data.

8.Alerts and Notifications:

•Set up alerting mechanisms to notify park management or visitors in the event of extreme environmental conditions. Alerts can be sent via email.

9.Maintenance and Calibration:

•Establish a regular maintenance schedule to ensure sensors remain functional. This includes battery replacement (if applicable) and sensor calibration.

10.Public Engagement:

- Promote awareness of the IoT monitoring system among park visitors. Display information about the system on signage or through the park's website.

11.Data Sharing:

Consider sharing anonymized environmental data with research institutions, local authorities, or educational programs to contribute to broader environmental monitoring efforts.

12.Continuous Improvement:

Regularly review the monitoring system's performance and assess whether it meets its objectives. Make adjustments and improvements as needed.

Design a web-based platform to display real time environmental data to the public

1. Define Objectives:

•Identify the specific environmental data to be displayed, such as temperature, humidity, air quality, and more. •Determine the scope and objectives of the platform, including its target audience (e.g., park visitors, local residents, researchers).

2. Data Sources:

•Identify the sources of real-time environmental data, including sensors deployed in the park or data from external sources (e.g., weather services, air quality monitoring agencies).
•Ensure data quality, accuracy, and reliability from these sources. 3. User Interface (UI) Design:
•Develop a user-friendly and visually appealing interface that is accessible on various devices (desktops, tablets, smartphones).
•Create a clean and intuitive design with easy navigation and clear data presentation.
•Use responsive design principles to adapt to different screen sizes

. 4. Data Visualization:
•Choose appropriate data visualization techniques to represent environmental data effectively. Examples include charts, graphs, maps, and gauges.
•Implement real-time updates to ensure users see the most current data.
•Provide historical data access for trend analysis and comparisons.

5. Map Integration:
•If relevant, integrate interactive maps to display geographical data, such as temperature variations across the park.
•Include markers or overlays to show sensor locations or specific environmental conditions in different park areas.

6. User Interactivity:
•Allow users to customize their experience by selecting which environmental parameters they want to view.
•Implement features like zooming, panning, and tooltips for an interactive experience.
•Provide options for users to set alerts or notifications based on predefined thresholds (e.g., high temperature warnings).

7. Accessibility and Inclusivity:
•Ensure the platform complies with web accessibility standards (e.g., WCAG) to accommodate users with disabilities.
•Design for inclusivity by making data and features accessible to a diverse audience.

8. Data Privacy and Security:
•Implement robust data security measures to protect sensitive information and user privacy.
•Clearly communicate data usage and privacy policies to users.

9. Real-Time Updates:
•Establish a data acquisition and processing pipeline to handle real-time data from sensors or external sources.
•Implement mechanisms for automatic data refresh at defined intervals (e.g., every few minutes).

10. User Engagement:
Encourage user engagement by providing educational content related to the displayed environmental data. This can include explanations of data trends or tips on outdoor activities. - Offer a feedback mechanism for users to report issues or provide suggestions for platform improvement.

11. Mobile Responsiveness:

Ensure the platform is mobile-responsive to accommodate users accessing it from smartphones and tablets while in the park.

12. Performance Optimization:

Optimize the platform's performance to handle concurrent user access, especially during peak times.

13. Testing and Feedback:

- Conduct thorough testing, including usability testing, to identify and address any issues or usability concerns. - Gather feedback from potential users and stakeholders to make necessary improvements.

14. Launch and Promotion:

Launch the platform and promote it through various channels, such as the park's website, social media, and local media outlets.

- Consider an official launch event or outreach campaign to raise awareness.

<u>IOT devices send data to the environmental monitoring platform</u>:

1.Sensor Data Collection:

➢ IoT devices equipped with environmental sensors (e.g., temperature, humidity, air quality) collect data from the park's surroundings.

2.Data Processing on IoT Devices:

➢ Process and preprocess data on IoT devices as needed, including calibration, filtering, or aggregation to reduce the volume of data transmitted.

3.Data Transmission Protocols:

➢ Choose appropriate data transmission protocols based on factors like data volume, device power constraints, and network availability. Common options include MQTT, HTTP, CoAP, or Lora WAN.

4.Connectivity Options:

➢ Select the most suitable connectivity options for IoT devices:

➢ Wi-Fi: If park areas have Wi-Fi coverage.

➢ Cellular: For areas without Wi-Fi coverage, providing broader connectivity.

➢ LoRa or other LPWAN technologies: Ideal for low-power, long-range communication in outdoor settings.

➢ Bluetooth or Zigbee: Suitable for short-range communications, such as within a local sensor network.

5.Security Measures:

➢ Implement robust security measures to protect data during transmission, including encryption and authentication.

➢ Utilize secure communication channels, such as VPNs or encrypted tunnels, to enhance data security.

6.Data Aggregation and Packaging:

➢ Aggregate data from multiple IoT devices into packets for efficient transmission.

➢ Include essential metadata, such as device ID, timestamps, and data type, in each packet.

7.Data Routing and Integration Layer:

➢ Set up a data routing and integration layer to receive incoming data from IoT devices.

➢ Implement routing rules to direct data to the appropriate destinations within the environmental monitoring platform.

8.Cloud or Server Integration:

➢ Integrate the data routing and integration layer with cloud services or a dedicated server hosting the environmental monitoring platform.

➢ Establish data storage solutions (e.g., databases, data lakes) to store and manage incoming data efficiently

Innovation:

- MQ135 sensor [interference with Arduino] – To check the daily air quality around the park with specifications of detecting **Air Humidity, Ammonia gas, Smoke** and **Other toxic Gases**.

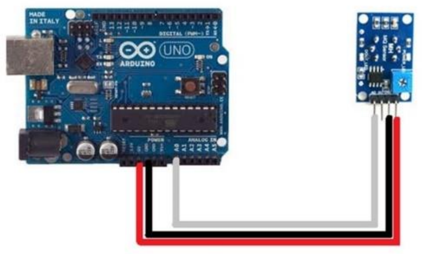
MQ135 Air Sensor with Arduino:

- MQ135 Sensor has high sensitivity to *ammonia gas, sulphide, benzene series stream, also can monitor smoke and other toxic gases well.*
- It can detect kinds of toxic gases and is a kind of low-cost sensor for kinds of applications.

Feature: It has good sensitivity to toxic gas in wide range, and has advantages such as long lifespan, low cost and simple drive circuit &etc.

Main application: It is widely used in domestic gas alarm, industrial gas alarm and portable gas detector.

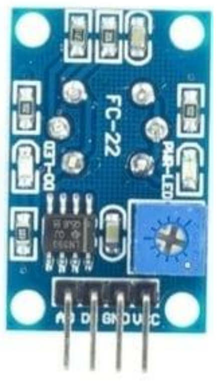
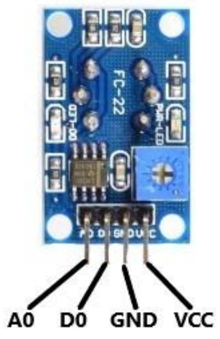
Circuit Diagram:



Arduino Specification:

- A0 – Analog output of the Sensor
- D0 – Digital output of the sensor
- GND – Ground
- VCC – 5V

-

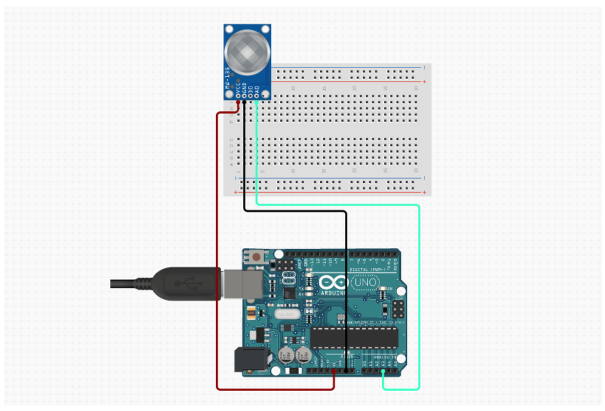



Components Required:

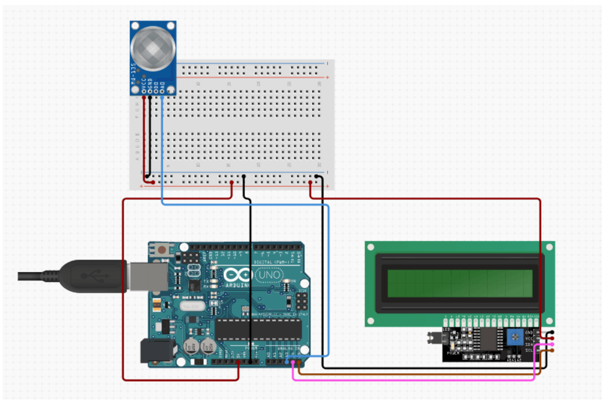
- Arduino Uno
- USB Cable
- LCD Display screen
- Gas sensor – MQ-135
- Bread Board
- Jumper wires Pack

Connections:

Step 1: MQ-135 sensor with Arduino.

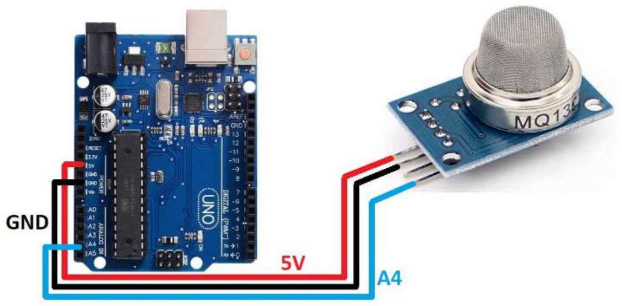


Step 2: MQ-135 Gas Sensor with LCD and Arduino.



Pin connection of MQ135 Air Quality Sensor with Arduino:

- Connect the VCC pin of the sensor to the 5V pin of the Arduino UNO board.
- Connect the GND pin of the sensor to the GND pin of the Arduino UNO board.
- Connect the AO pin of the sensor to the A4 pin of the Arduino UNO board.



Wiring MQ-135 sensor with Arduino UNO:

Step 1: First, place a 16×2 LCD on the breadboard, then connect A to +5V with a 220-ohm resistor and K to Ground. To vary the contrast of a 16×2 LCD, connect the VO to the middle pin of the potentiometer and VDD to +5V, VSS & RW to Ground. Also, provide +5V and Ground to the potentiometer.

Step 2: The MQ-135 module connects to the A0 pin of an Arduino Uno and connects GND to Ground, providing +5V to VCC.

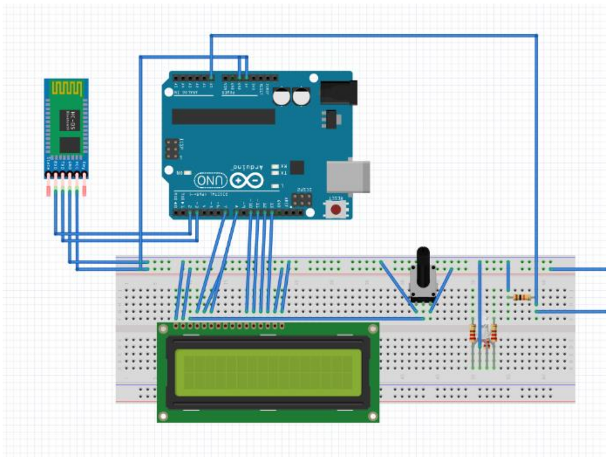
Step 3: Connect Anode (+) of Green LED to digital pin 8 of Arduino; Blue LED to digital 9 pin of Arduino and Red LED to digital pin 10 of Arduino and all LEDs Cathode (-) Ground with 220-ohm resistor.

Step 4: Connect the buzzer's positive terminal to digital pin 11 of Arduino and the negative terminal to the ground.

Water Quality sensor:

- This method uses Arduino to monitor the quality of water, and can be modified to monitor various aspects such as the temperature of the water and its surrounding, the turbidity of the water (how clean the water is) as well as the PH levels of the Water.
- This system monitors all of these aspect and finally when all checks have been completed, the information or data can be sent to the user.

Circuit Diagram:



Components Required:

- Arduino UNO
- Resistor 330 ohm
- Jumper Wires
- RGB Diffused Common Cathode
- Rotary potentiometer

- Alphanumeric LCD
- HC-05 Bluetooth Module
- Resistor

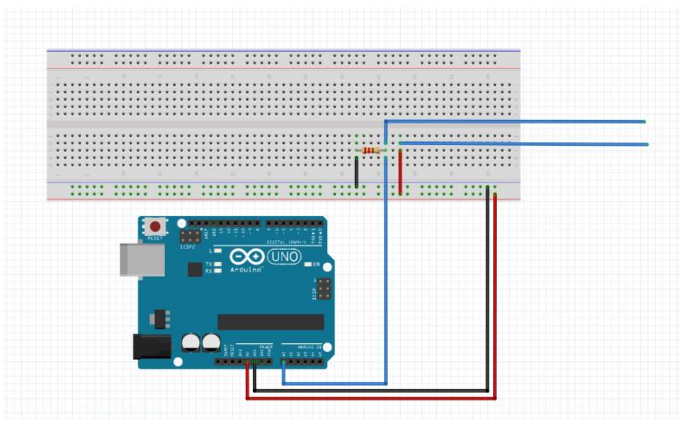
Pin Connections:

With Arduino:

Step 1: Connect 5V of Arduino to one power rail of the breadboard

Step 2: Connect the ground of Arduino to the other power rail of the breadboard

Step 3: Connect one end of a 1k-ohm resistor to the ground and the other end to the breadboard. Connect the analog pin A0 on the Arduino to the resistor. Finally, connect a wire to the resistor and another wire to 5V. Connect the free ends of these wires to the crocodile clips.



With LCD Display:

Step 1: Connect VSS pin to the ground rail

Step 2: Connect VDD pin to 5V rail

Step 3: Connect V0 to the centre pin of the potentiometer

Step 4: Connect ends of the potentiometer to 5V and ground

Step 5: Connect RS pin to Arduino pin 7

Step 6: Connect R/W pin to the ground rail

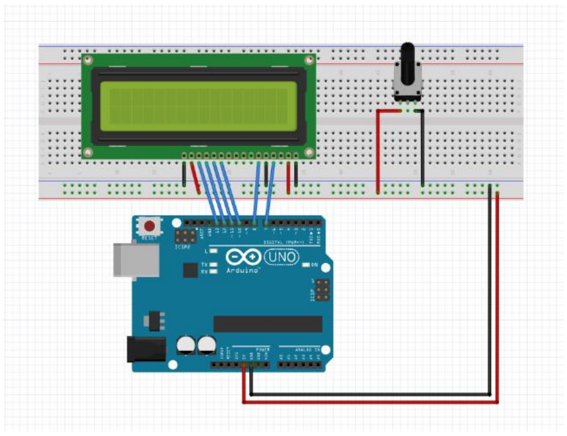
Step 7: Connect E pin to Arduino pin 8

Step 9: Connect D4 to Arduino pin 10

Step 10: Connect D5 to Arduino pin 11

Step 11: Connect D6 to Arduino pin 12

Step 12: Connect D7 to Arduino pin 13



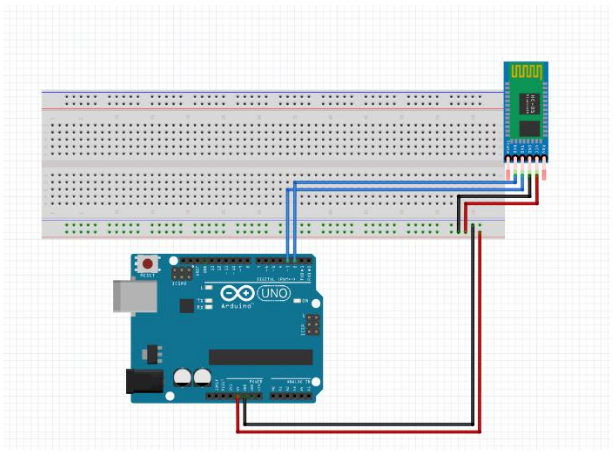
HC-05 Bluetooth Module:

Step 1: Connect VCC pin to 5V rail

Step 2: Connect GND pin to ground

Step 3: Connect TX pin to Arduino pin 3 (Serves as RX)

Step 4: Connect RX pin to Arduino pin 2 (Serves as TX)



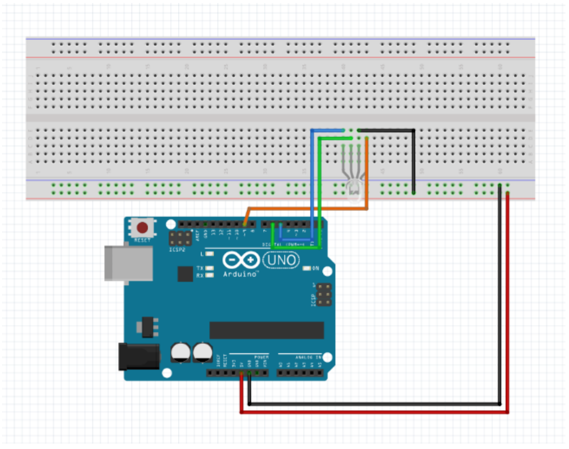
With RGB LED:

Step 1: Connect the common cathode (longest pin) to ground

Step 2: Connect the red pin (right of cathode pin) to PWM pin 9 on Arduino through a 330-ohm resistor

Step 3: Connect green pin (left of cathode pin) to PWM pin 6 on Arduino through a 330-ohm resistor

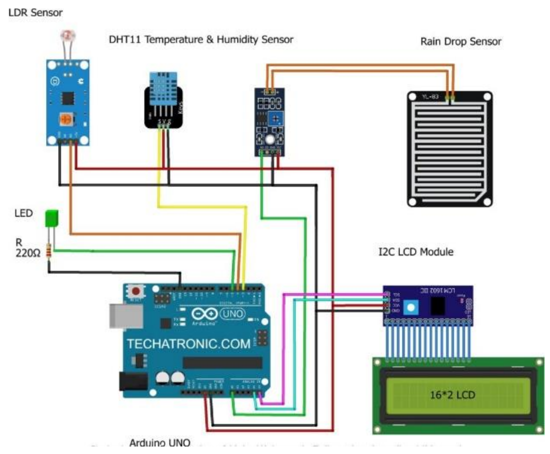
Step 4: Connect blue pin (extreme left) to PWM pin 5 on Arduino through a 330-ohm resistor



Weather Monitoring System with Arduino:

- Weather monitoring system using Arduino can help us to monitoring the temperature and humidity by using one sensor.
- The DH11 Sensor is easily capable to measure the temperature and humidity.
- The Arduino uno help to calculate & display the information to the display.
- nrf have good range for data transmission that’s why we are using nrf module for communication.
- At the side of transmitter there are dht11 sensor which gives the output to the transmitter microcontroller and transmitter microcontroller of **weather monitoring system** send this information to the receiver via transmitter nrf.

Circuit Diagram:



Components Required:

- Arduino UNO
- LCD Display
- DHT11 Sensor
- Rain Sensor
- LDR Sensor
- Breadboard
- Jumper wires
- LED
- 220-ohm Resistor

Connection:

Step 1: Connect the LCD I2C (SCL) –> A5

Step 2: Connect the LCD I2C (SDA) –> A4

Step 3: Connect the RAIN SENSOR (A0) –> A0

Step 4: Connect the DHT11 (OUT) –> PIN 3

Step 5: Connect the LDR MODULE (D0) –> PIN 4

Step 6: Connect the GREEN LED (ANODE) –> PIN 5

AI BASED DATA SET:

Weather Data:

Meteorological Data: Collect historical and real-time weather data, including temperature, humidity, wind speed, wind direction, atmospheric pressure, and precipitation.

Solar Radiation Data: Gather data related to solar radiation, UV levels, and solar exposure.

Air Quality Data: Include measurements of pollutants such as PM2.5, PM10, CO2, NO2, and O3.

Rainfall Data: Data on the timing, intensity, and duration of rainfall or snowfall events.

Geospatial Data:

GIS Data: Geographic Information System (GIS) data, such as maps, land use data, and terrain information.

GPS Data: Collect data related to the location of monitoring stations or weather sens

Sensor Data:

Weather Station Data: Data from weather stations that include temperature sensors, anemometers, barometers, and rain gauges.

Air Quality Sensor Data: Data from air quality monitoring sensors, which measure air pollutants and meteorological parameters.

Satellite Imagery: Utilize satellite images to gather data on cloud cover, surface temperature, and more.

Historical Data:

Past weather records, ideally spanning several years, to enable long-term trend analysis.

Crowdsourced Data:

Data collected from public sources, such as social media posts or citizen science initiatives.

User-Generated Data:

Data collected from mobile apps or devices used by the public for weather-related information sharing.

Annotations:

Annotate the data with labels such as weather conditions (e.g., clear, cloudy, rainy), air quality levels (e.g., good, moderate, unhealthy), and geospatial coordinates.

Images and Videos:

Images or videos capturing real-time weather conditions can be valuable for training AI models.

Natural Language Data:

Textual data from news articles, weather reports, or social media that describe weather events and conditions.

PYTHON SCRIPT:

```
import Adafruit_DHT
import requests
import time
# Sensor setup (DHT22)
DHT_SENSOR = Adafruit_DHT.DHT22
DHT_PIN = 4  # GPIO pin on the Raspberry Pi

# Server URL for sending data (optional)
SERVER_URL = "https://yourserver.com/upload_data"
# Main loop
while True:
    try:
        # Read sensor data
        humidity, temperature = Adafruit_DHT.read_retry(DHT_SENSOR,
DHT_PIN)
        if humidity is not None and temperature is not None:
            # Print data to the console
            print(f"Temperature: {temperature:.2f}°C, Humidity: {humidity:.2f}%")
            # Send data to a server (optional)
            if SERVER_URL:
                data = {"temperature": temperature, "humidity": humidity, "location":
"Park Name"}
                response = requests.post(SERVER_URL, json=data)
                if response.status_code == 200:
                    print("Data sent to the server successfully.")
                else:
                    print("Failed to send data to the server.")
        else:
            print("Failed to retrieve data from the sensor.")
        time.sleep(60)  # Wait for 60 seconds before reading data again
    except KeyboardInterrupt:
        print("Exiting the program.")
        break
```

ARDUINO BASED SENSOR CODE:

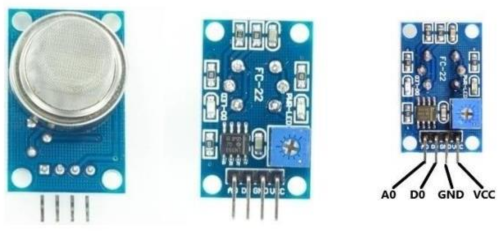
```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP085_U.h>
#include <Adafruit_DHT.h>
#include  <LiquidCrystal.h>
#define DHTPIN 7
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
Adafruit_BMP085_Unified bmp = Adafruit_BMP085_Unified(10085);
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  Serial.begin(9600);
  lcd.begin(16, 2); // Initialize the LCD
  if (!bmp.begin()) {
    Serial.println("Could not find a valid BMP085 sensor, check wiring!");
    while (1) {}
  }
}
void loop()
 {
  sensors_event_t event;
  bmp.getEvent(&event);
  float temperature;
  float pressure;
  float humidity = dht.readHumidity();
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Temperature: ");
  if (isnan(humidity))
 {
    lcd.print("Failed to read DHT");
  }
else
 {
    temperature = event.temperature;
    lcd.print(temperature, 1);
    lcd.print("C");
    lcd.setCursor(0, 1);
    lcd.print("Humidity: ");
    lcd.print(humidity);
    lcd.print("%");
```

```
}
delay(2000);
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Pressure: ");
lcd.print(event.pressure, 2);
lcd.print("hPa");
lcd.setCursor(0, 1);
lcd.print("Altitude: ");
lcd.print(bmp.readAltitude(1013.25));
lcd.print("m");
delay(2000);
```

IOT DEVICE:

Arduino Specification:

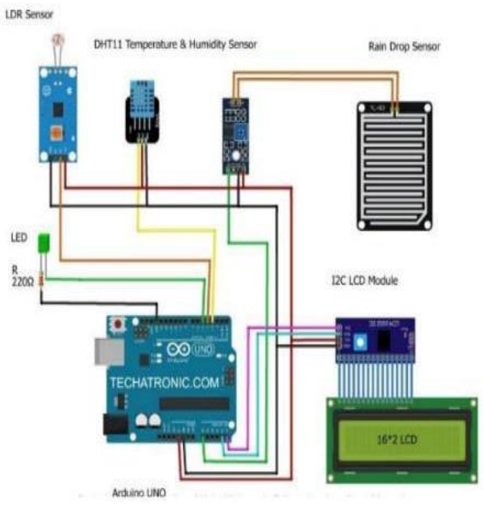
- A0 – Analog output of the Sensor
- D0 – Digital output of the sensor
- GND – Ground
- VCC – 5V



Components Required:

- Arduino Uno
- USB Cable
- LCD Display screen
- Gas sensor – MQ-135
- Bread Board
- Jumper wires Pack

Circuit Diagram:



Overview:

The use of IoT in an EMS allows for the monitoring and control of various environmental parameters, such as air quality, water quality, energy consumption, waste management, and more. Sensors and devices can be deployed to collect data on these parameters, which is then transmitted to a central system for analysis and action.

The real-time nature of IoT data allows organizations to respond quickly to environmental incidents or deviations from set targets. For instance, if a sensor detects a sudden increase in air pollution levels, an alert can be generated, enabling immediate action to be taken to mitigate the issue.

In summary, incorporating IoT into an EMS provides organizations with the ability to collect and analyze real-time environmental data, leading to improved environmental performance, resource efficiency, and sustainability. It allows for proactive environmental management, reduces costs, and enhances overall operational effectiveness.

VARIOUS TOOLS USED FOR THIS PROJECT :

Sensors: IoT environmental monitoring systems rely on a variety of sensors designed to measure parameters like temperature, humidity, air quality, water quality, soil moisture, radiation levels, and more. These sensors are strategically deployed to collect real-time data.

IoT Devices: These are the hardware components that house sensors, process data, and facilitate communication. Devices like Raspberry Pi, Arduino, or specialized IoT modules are commonly used to collect data from sensors and transmit it to central systems.

Communication Networks: Data from sensors is transmitted using various communication protocols, such as Wi-Fi, Bluetooth, LoRaWAN, Zigbee, or cellular networks. The choice of network depends on the specific application's range and data transfer requirements.

IoT Platform: Data collected by sensors is sent to an IoT platform or cloud service, such as AWS IoT, Google Cloud IoT, or Microsoft Azure IoT. These platforms provide storage, data processing, real-time monitoring, and visualization tools.

Data Processing and Analytics: The collected data is processed and analyzed to derive valuable insights. Advanced analytics techniques may be used to detect trends, anomalies, and patterns within the data.

Creating a real-time Environment Management platform involves a combination of front end and back end technologies. Here’s a simplified outline using C and C++ and phython programing with wifi connection for the front end and Node.js for the back end:

Phython:

```
import network
import time
from machine import Pin,ADC
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))
MQ7=ADC(Pin(35))
MQ8=ADC(Pin(32))
button=Pin(34,Pin.IN)
led=Pin(33,Pin.OUT)
min_rate=0
max_rate=4095

print("Connecting to WiFi", 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:
  CO_sensor=(MQ7.read())*100/(max_rate)
  print("CO Sensor value: " + "%.2f" % CO_sensor +"%")
  Hydrogen_sensor=(MQ8.read())*100/(max_rate)
  print("Soil Sensor value: " + "%.2f" % Hydrogen_sensor +"%")
  button_value=button.value()
  if button_value == True:
    led.value(000)
    print("It's Raining")
  else:
    led.value(0)
  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)
```

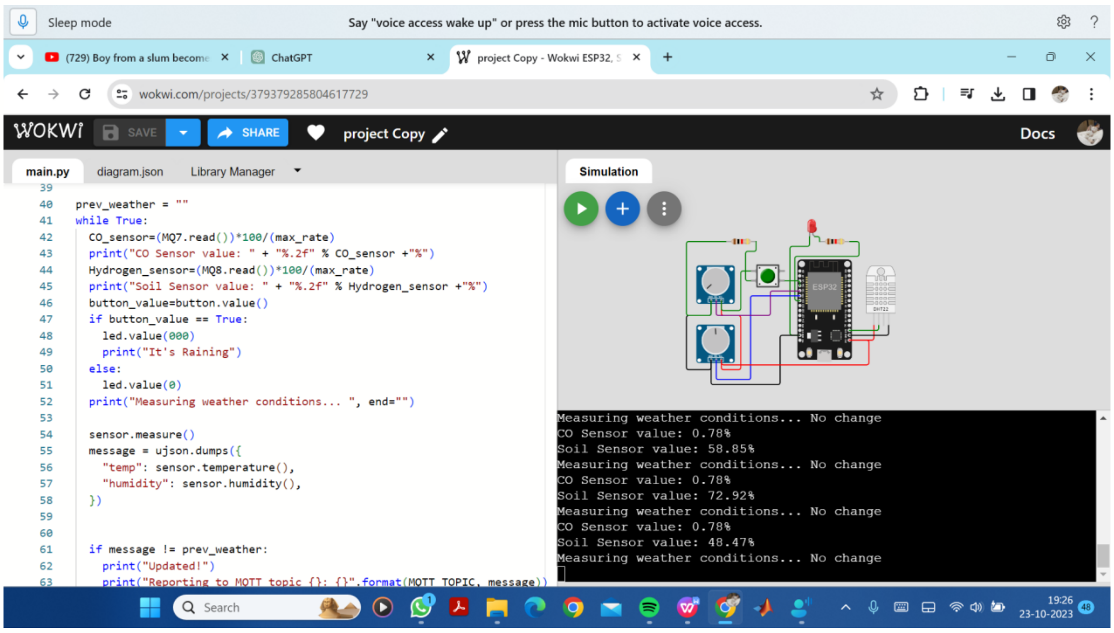
C++:

```
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <ESP8266WiFi.h>
#include <Adafruit_MQTT.h>
#include <Adafruit_MQTT_Client.h>

// Replace these with your Wi-Fi credentials.
const char* WIFI_SSID = "YourWiFiSSID";
const char* WIFI_PASS = "YourWiFiPassword";

// Replace with your Adafruit IO credentials.
#define ADAFRUIT_IO_USERNAME "YourAdafruitUsername"
#define ADAFRUIT_IO_KEY "YourAdafruitAIOKey"

// Define the DHT sensor.
#define DHT_PIN 2          // The pin where your DHT sensor is connected.
#define DHT_TYPE DHT22     // DHT sensor type (DHT11, DHT22, AM2302, etc.)

DHT dht(DHT_PIN, DHT_TYPE);

WiFiClient client;

Adafruit_MQTT_Client mqtt(&client, "io.adafruit.com", 1883, ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY);

// Define MQTT feeds.
Adafruit_MQTT_Publish temperature = Adafruit_MQTT_Publish(&mqtt, ADAFRUIT_IO_USERNAME "/feeds/temperature");
Adafruit_MQTT_Publish humidity = Adafruit_MQTT_Publish(&mqtt, ADAFRUIT_IO_USERNAME "/feeds/humidity");

void setup() {
  Serial.begin(115200);

  // Connect to Wi-Fi.
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi...");
  }

  Serial.println("Connected to WiFi");

  // Connect to Adafruit IO.
  mqtt.connect();
  Serial.println("Connected to Adafruit IO");
}
void loop() {
  // Read temperature and humidity data from the DHT sensor.
  float temperatureValue = dht.readTemperature();
  float humidityValue = dht.readHumidity();
  // Publish data to Adafruit IO.
  if (!isnan(temperatureValue)) {
    temperature.publish(temperatureValue);
    Serial.print("Temperature: ");
    Serial.println(temperatureValue);
  } else {
    Serial.println("Failed to read temperature");
  }

  if (!isnan(humidityValue)) {
    humidity.publish(humidityValue);
    Serial.print("Humidity: ");
    Serial.println(humidityValue);
  } else {
    Serial.println("Failed to read humidity");
  }

  delay(60000); // Delay for 60 seconds (adjust as needed).
}
```

C program:

```
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <ESP8266WiFi.h>
#include <Adafruit_MQTT.h>
#include <Adafruit_MQTT_Client.h>

// Replace these with your Wi-Fi credentials.
const char* WIFI_SSID = "YourWiFiSSID";
const char* WIFI_PASS = "YourWiFiPassword";

// Replace with your Adafruit IO credentials.
#define ADAFRUIT_IO_USERNAME "YourAdafruitUsername"
#define ADAFRUIT_IO_KEY "YourAdafruitAIOKey"

// Define the DHT sensor.
#define DHT_PIN 2           // The pin where your DHT sensor is connected.
#define DHT_TYPE DHT22      // DHT sensor type (DHT11, DHT22, AM2302, etc.)

DHT dht(DHT_PIN, DHT_TYPE);

WiFiClient client;

Adafruit_MQTT_Client mqtt(&client, "io.adafruit.com", 1883, ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY);

// Define MQTT feeds.
Adafruit_MQTT_Publish temperature = Adafruit_MQTT_Publish(&mqtt, ADAFRUIT_IO_USERNAME "/feeds/temperature");
Adafruit_MQTT_Publish humidity = Adafruit_MQTT_Publish(&mqtt, ADAFRUIT_IO_USERNAME "/feeds/humidity");

void setup() {
  Serial.begin(115200);

  // Connect to Wi-Fi.
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi...");
  }

  Serial.println("Connected to WiFi");

  // Connect to Adafruit IO.
  mqtt.connect();
  Serial.println("Connected to Adafruit IO");
}

void loop() {
  // Read temperature and humidity data from the DHT sensor.
  float temperatureValue = dht.readTemperature();
  float humidityValue = dht.readHumidity();

  // Publish data to Adafruit IO.
  if (!isnan(temperatureValue)) {
    temperature.publish(temperatureValue);
    Serial.print("Temperature: ");
    Serial.println(temperatureValue);
  } else {
    Serial.println("Failed to read temperature");
  }
  if (!isnan(humidityValue)) {
    humidity.publish(humidityValue);
    Serial.print("Humidity: ");
    Serial.println(humidityValue);
  } else {
    Serial.println("Failed to read humidity");
  }
  delay(60000); // Delay for 60 seconds (adjust as needed).
}
```

Result:

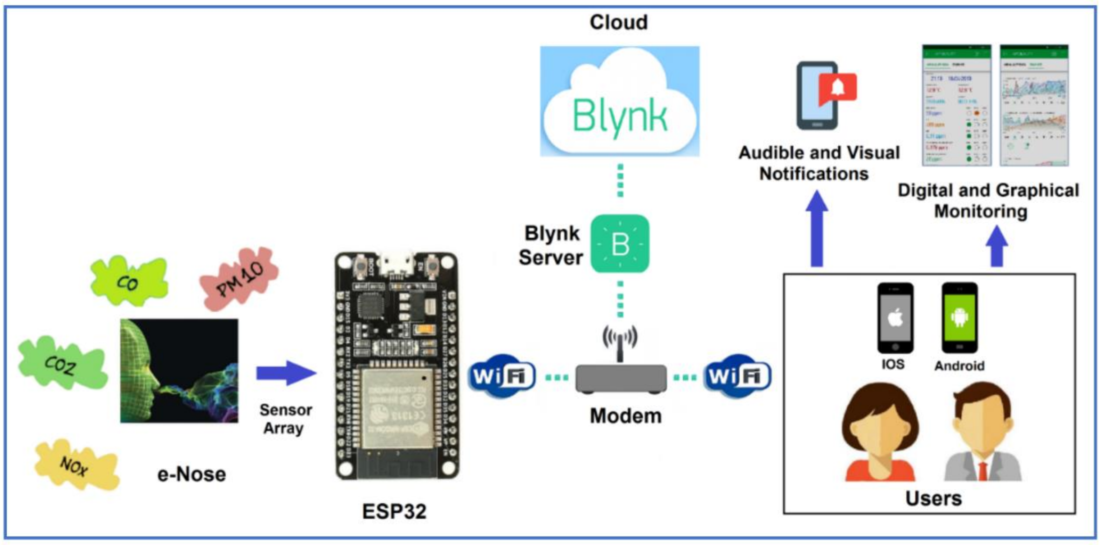


```
prev_weather = ""
while True:
  CO_sensor=(MQ7.read())*100/(max_rate)
  print("CO Sensor value: " + "%.2f" % CO_sensor +"%")
  Hydrogen_sensor=(MQ8.read())*100/(max_rate)
  print("Soil Sensor value: " + "%.2f" % Hydrogen_sensor +"%")
  button_value=button.value()
  if button_value == True:
    led.value(000)
    print("It's Raining")
  else:
    led.value(0)
  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))
```

```
Measuring weather conditions... No change
CO Sensor value: 0.78%
Soil Sensor value: 58.85%
Measuring weather conditions... No change
CO Sensor value: 0.78%
Soil Sensor value: 72.92%
Measuring weather conditions... No change
CO Sensor value: 0.78%
Soil Sensor value: 48.47%
Measuring weather conditions... No change
```

Sleep mode
Say "voice access wake up" or press the mic button to activate voice access.
(729) Boy from a slum become
ChatGPT
project Copy - Wokwi ESP32,
wokwi.com/projects/379379285804617729
WOKWI
SAVE
SHARE
project Copy
Docs
main.py
diagram.json
Library Manager
Simulation
ESP32
DHT22
Search
19:26
23-10-2023

VIEW MODEL FOR ENVIRONMENTAL MONITERING:



OVERALL DETAILS ABOUT ENVIRONMENTAL MONITORING:

Weather Monitoring:

This involves the collection of data related to weather conditions such as temperature, humidity, precipitation, wind speed and direction, atmospheric pressure, and solar radiation. Accurate weather data is essential for forecasting, climate studies, and providing timely information to park visitors and local communities.

Air Quality Monitoring:

Monitoring air quality involves measuring concentrations of pollutants like particulate matter (PM2.5 and PM10), ozone (O3), nitrogen oxides (NOx), sulfur dioxide (SO2), carbon monoxide (CO), and volatile organic compounds (VOCs). This helps assess air quality and its impact on human health and the environment.

Water Quality Monitoring:

It includes testing water bodies within the park, such as rivers, lakes, and ponds, for parameters like water temperature, pH, turbidity, dissolved oxygen, nutrient levels, and the presence of contaminants like heavy metals and pathogens. This ensures the health of aquatic ecosystems and water resources.

Ecological Monitoring:

Monitoring the health and biodiversity of plant and animal species in the park is crucial. This can involve tracking population sizes, migration patterns, and the impact of environmental changes on local flora and fauna.

Soil and Groundwater Monitoring:

Assessing soil quality and groundwater conditions helps ensure the stability of the ecosystem, identify contamination sources, and protect the local environment.

Remote Sensing:

Modern technology, including satellite imagery and drones, can be used to gather data from the park's surroundings, offering a broader perspective on environmental conditions and changes.

Data Analysis and Reporting:

The collected data is analysed and used to generate reports, which can inform decision-making, assist in the management of the park, and provide the public with information on current environmental conditions.

Public Awareness and Education:

Environmental monitoring in a weather-checking park can also serve as an educational tool for park visitors, helping them understand the importance of preserving natural resources and contributing to scientific research.

Conclusion:

In conclusion, an Environmental Monitoring System using the Internet of Things (IoT) represents a transformative and highly valuable technology for addressing a wide range of environmental challenges. This system harnesses the power of interconnected sensors, devices, and data analytics to collect, manage, and analyze environmental data in real-time.