**Air Quality Monitoring(Full submission)**

Certainly! AQM-IOT stands for Air Quality Monitoring in the Internet of Things (IoT) context. It refers to the use of IoT devices and technology to monitor and manage air quality. Here's a full description of AQM-IOT:

**Title**: **Air Quality Monitoring in the Internet of Things (AQM-IOT)**

**Introduction:**

Air Quality Monitoring in the Internet of Things (AQM-IOT) is a cutting-edge approach to assessing and managing air quality in both indoor and outdoor environments. It leverages the power of IoT devices and technology to collect real-time data on various air pollutants, enabling more informed decisions for public health, environmental conservation, and overall well-being.

**Key Components of AQM-IOT:**

1. **Sensor Nodes: AQM-IOT**: It relies on a network of specialized sensor nodes strategically placed in different locations. These nodes are equipped with sensors capable of measuring a wide range of air quality parameters, including particulate matter (PM2.5 and PM10), carbon dioxide (CO2), carbon monoxide (CO), ozone (O3), nitrogen dioxide (NO2), sulfur dioxide (SO2), and volatile organic compounds (VOCs).

2. **Data Connectivity**: IoT technology facilitates seamless data transmission from sensor nodes to central databases or cloud platforms. This connectivity allows for real-time monitoring and analysis, ensuring that air quality data is accessible to relevant stakeholders.

3. **Data Analysis**: The collected data is processed and analyzed using sophisticated algorithms and machine learning techniques. This analysis provides insights into pollutant concentrations, trends, and potential health risks. It can also identify pollution sources and patterns.

4. **Visualization and Alerts**: AQM-IOT systems often include user-friendly interfaces, such as web dashboards or mobile apps, that display air quality information in an easily understandable format. Alerts and notifications are generated when air quality exceeds predefined thresholds, enabling timely responses.

5. **Integration with Environmental Agencies**: Many AQM-IOT projects collaborate with environmental agencies, allowing them to access and utilize the collected data for regulatory compliance and policy decisions.

**Benefits of AQM-IOT**:

1. **Improved Public Health**: Real-time air quality information helps individuals make informed decisions about outdoor activities and can mitigate the health risks associated with poor air quality.

2. **Environmental Conservation**: AQM-IOT data aids in the identification of pollution sources, enabling authorities to take corrective actions to reduce emissions and improve environmental sustainability.

3. **Data-Driven Policies**: Governments and local authorities can use AQM-IOT data to formulate and implement policies aimed at reducing air pollution and protecting public health.

4. **Research and Innovation**: Researchers can access vast datasets to conduct studies on air quality trends, pollution's impact on health, and the development of new mitigation technologies.

5. **Community Engagement**: AQM-IOT systems often engage the community by providing access to real-time data, fostering awareness and citizen involvement in air quality improvement efforts.

**Challenges and Future Developments:**

1. **Data Privacy and Security**: Protecting sensitive air quality data from cyber threats is a significant challenge that AQM-IOT systems must address.

2. **Data Accuracy**: Ensuring the accuracy and calibration of sensors is crucial for reliable air quality assessments.

3. **Scalability**: Expanding AQM-IOT networks to cover larger geographical areas requires careful planning and investment.

4. **Integration with Smart Cities**: The integration of AQM-IOT into broader smart city initiatives is an exciting avenue for future development.

In conclusion, AQM-IOT is a transformative approach to air quality monitoring that harnesses the capabilities of IoT technology. It plays a vital role in safeguarding public health, preserving the environment, and shaping data-driven policies for cleaner and healthier communities. As technology advances and AQM-IOT networks expand, the potential for positive impacts on air quality and human well-being is boundless.

Here are high-level algorithm steps for Air Quality Management (AQM) in IoT:

**Algorithm:**

**Step1:**

- IoT sensors collect air quality data, including parameters like PM2.5, PM10, CO2 levels, temperature, and humidity.

**Step2:**

- Transmit collected data to a central IoT platform or cloud for processing and analysis.

**Step3:**

- Clean and preprocess the data, removing outliers and ensuring accuracy.

**Step4:**

- Analyze the air quality data to detect trends, anomalies, and potential pollution events.

**Step5:**

- Set threshold values for various pollutants and continuously monitor the data against these thresholds.

**Step6:**

6.1: Generate alerts or notifications when air quality parameters exceed predefined thresholds.

6.2: Create visual representations of air quality data for real-time monitoring and historical analysis.

**Step7:**

7.1: Trigger actions based on air quality alerts, such as activating air purifiers, adjusting ventilation systems, or notifying authorities.

7.2: Continuously collect and analyze data to refine threshold values and improve the accuracy of alerts.

**Step8:**

- Develop a user-friendly interface for users to access real-time air quality information and receive alerts.

**Step9:**

- Store historical air quality data for long-term analysis, reporting, and trend identification.

**Step10:**

- Integrate the AQM system with other smart devices and services for holistic air quality management.

**Step11:**

11.1: Implement robust security measures to protect air quality data from unauthorized access and tampering.

11.2: Ensure compliance with environmental regulations and generate reports for regulatory authorities.

11.3: Regularly maintain and calibrate IoT sensors to ensure data accuracy and system reliability.

**Step12:**

12.1: Design the system to scale efficiently as the number of IoT sensors and monitored locations increase.

12.2: Optimize power consumption of IoT devices to prolong battery life and reduce environmental impact.

12.3: Gather feedback from users and stakeholders to make continuous improvements to the AQM system.

**Program:**

import time

import csv

#Stimulated air quality sensor function

def read\_air\_quality\_sensor():

#Replace this with actual sensor data

#Return values like(PM2.5, PM10, CO2)

Return(25, 15, 400)

#Function to log data to a CSV file

def log\_to\_csv(data, filename):

with open(filename, ‘a’, newline=”)as file:

writer=csv.write(file)

writer.writerow(data)

#Main monitoring loop

def main():

while true:

#Read air quality sensor data

air\_quality\_data=read\_air\_quality\_sensor()

#Get current timestamp

Timestamp=timie.strftime(‘%Y-%m-%d%H:%M:%S’)

#Combine data with timestamp

data\_with\_time=(timiestamp)+air\_quality\_data

#Log data to CSV file

log\_to\_csv(data\_with\_time, ‘air\_quality\_data.csv’)

#Print the data(can replace this with data virtualization)

print(data\_with\_time)

#Sleep for a specified interval

time.stamp(600)

if name\_==’\_\_main\_\_’:

main()

**A real-time air quality monitoring system can significantly raise public awareness about air quality and its health impacts in several ways:**

**1. Immediate Access to Information:** Real-time monitoring systems provide instant access to current air quality data. People can check the air quality in their area at any time, leading to increased awareness about the air they breathe on a daily basis.

**2. Visibility of Air Pollution Trends:** Continuous monitoring allows the public to observe patterns and trends in air quality. Spikes in pollution levels, especially during specific times of the day or under certain weather conditions, can be easily noticed. This visibility can highlight the sources of pollution and emphasize the need for specific actions.

**3. Alerts and Notifications:** Real-time systems can send alerts and notifications to the public when air quality reaches unhealthy levels. These alerts can be delivered through mobile apps, social media, or other communication channels, prompting people to take precautions such as staying indoors, wearing masks, or avoiding outdoor activities.

**4. Educational Outreach:** By providing real-time data, monitoring systems can educate the public about the various pollutants present in the air and their health effects. Understanding the correlation between air quality and health issues can motivate individuals and communities to advocate for cleaner air policies.

**5. Policy Advocacy:** Access to real-time data empowers communities to advocate for stricter environmental regulations and policies. Public awareness can lead to increased pressure on authorities to enforce existing regulations or introduce new measures to curb pollution.

**6. Behavioral Changes:** When individuals are aware of poor air quality, they are more likely to make behavioral changes such as using public transportation, carpooling, or reducing energy consumption. These changes, when adopted collectively, can lead to a significant reduction in pollution levels.

**7. Community Engagement:** Real-time monitoring systems foster community engagement by encouraging discussions and actions related to air quality. Communities can organize awareness campaigns, workshops, and clean-up initiatives, promoting a sense of shared responsibility for air quality improvement.

**8. Health Impact Awareness:** Real-time monitoring systems can link air quality levels to specific health conditions. For instance, during high pollution days, information about the increased risk of respiratory issues, allergies, or heart problems can be disseminated, prompting vulnerable individuals to take extra precautions.

**9. Research and Data Analysis:** Real-time data collected over time can be valuable for scientific research. Researchers can analyze this data to identify long-term trends, pollution sources, and health impacts, providing evidence for policy changes and public awareness campaigns.

**Objectives:**

1. Define the Problem:

* Determine what air quality parameters you want to monitor (e.g., PM2.5, CO2, humidity).
* Define the scope of the project (e.g., monitoring indoor air quality in homes or offices).

2. Data Collection:

* Set up IoT sensors to collect air quality data.
* Store the data in a database for further analysis.

3. Data Preprocessing:

* Clean the data by handling missing values and outliers.
* Convert raw sensor data into meaningful features.
* Timestamp data for time-based analysis.

4. Feature Engineering:

* Extract relevant features from the raw data (e.g., hourly averages, daily averages).
* Consider adding additional features like weather conditions, time of day, or occupancy status if available.

5. Data Splitting:

* Split the dataset into training and testing sets (typically 80/20 or 70/30 ratio).

6. Model Selection:

* Choose appropriate machine learning models for the task (e.g., regression models for numerical prediction).
* Common models include Linear Regression, Random Forest, or Gradient Boosting.

7. Model Training:

* Train the selected models using the training dataset.
* Tune hyperparameters using techniques like grid search or random search.

8. Model Evaluation:

* Evaluate the models using appropriate metrics (e.g., Mean Absolute Error, Root Mean Squared Error) on the test dataset.
* Compare the performance of different models to choose the best one.

9. Deployment:

* Deploy the trained model on an IoT device or a cloud server to make real-time predictions.

10. Monitoring and Maintenance:

* Set up monitoring mechanisms to track the performance of the deployed model over time.
* Implement regular maintenance to update the model if new data suggests a change in the underlying patterns.

**Dataset:** <https://www.kaggle.com/datasets/fedesoriano/air-quality-data-set/>

**Reference for Iot:** <https://www.researchgate.net/figure/Block-diagram-of-proposed-IOT-based-Air-Pollution-Monitoring-System_fig1_342338791>

**Sample Python Code (Using Scikit-Learn):**

#Here's an example code snippet demonstrating steps 3 to 7 using Python and Scikit-Learn for a regression problem (predicting air quality index):

import pandas as pd

# Load air quality data from CSV file

def load\_air\_quality\_data(air\_quality\_data):

try:

df = pd.read\_csv(air\_quality\_data)

return df

except FileNotFoundError:

print("Error: File not found.")

return None

except pd.errors.EmptyDataError:

print("Error: The CSV file is empty.")

return None

# Function to check air quality

def check\_air\_quality(pm25, co2, humidity):

if 0 < pm25 <= 12:

pm25\_status = "Good"

elif 12 < pm25 <= 35.4:

pm25\_status = "Moderate"

elif 35.4 < pm25 <= 55.4:

pm25\_status = "Unhealthy for Sensitive Groups"

elif 55.4 < pm25 <= 150.4:

pm25\_status = "Unhealthy"

elif 150.4 < pm25 <= 250.4:

pm25\_status = "Very Unhealthy"

else:

pm25\_status = "Hazardous"

if 0 < co2 <= 400:

co2\_status = "Good"

elif 400 < co2 <= 1000:

co2\_status = "Moderate"

else:

co2\_status = "Poor"

if 0 < humidity <= 50:

humidity\_status = "Good"

elif 50 < humidity <= 70:

humidity\_status = "Moderate"

else:

humidity\_status = "Poor"

return pm25\_status, co2\_status, humidity\_status

# Main function

def main():

file\_path = 'air\_quality\_data.csv' # Path to the CSV file

air\_quality\_data = load\_air\_quality\_data(file\_path)

if air\_quality\_data is not None:

for index, row in air\_quality\_data.iterrows():

pm25 = row['PM2.5 (µg/m³)']

co2 = row['CO2 (ppm)']

humidity = row['Humidity (%)']

pm25\_status, co2\_status, humidity\_status = check\_air\_quality(pm25, co2, humidity)

print(f"Timestamp: {row['Timestamp']}")

print(f"PM2.5 Status: {pm25\_status}")

print(f"CO2 Status: {co2\_status}")

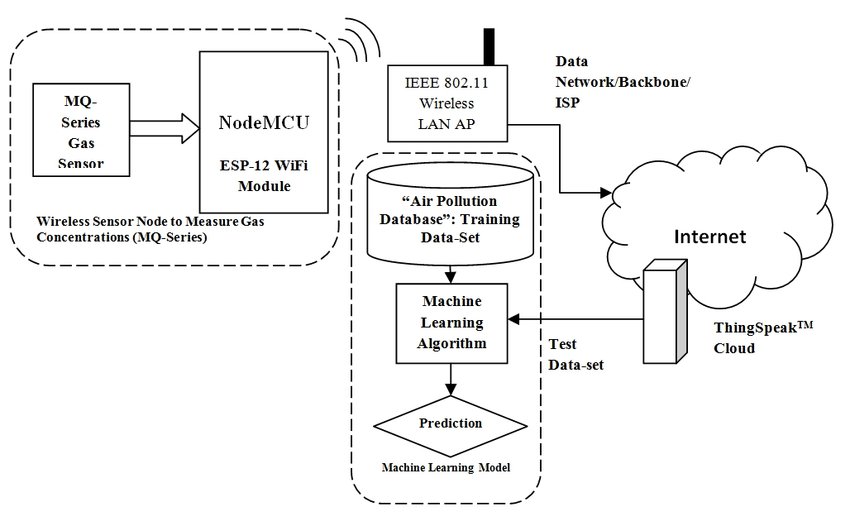
print(f"Humidity Status: {humidity\_status}")

print("-" \* 40)

if \_\_name\_\_ == "\_\_main\_\_":

main(

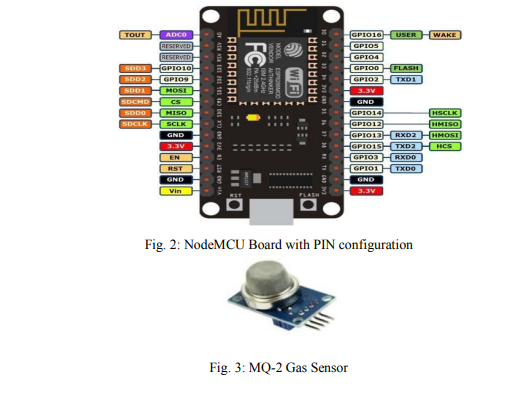
**Block Diagram:**

****

**IoT prototype(setup):**

* IoT mainly deals with connecting smart devices (embedded electronics devices) to internet by harnessing the advantage of OSI layered Architecture.
* In the context of this work we propose a cluster of Air Quality Monitoring Sensor motes, which are used to measure the concentration of Air pollutants in the air.
* All the Air Sensors are interfaces with a tiny embedded platform equipped with network connectivity and are interconnected to internet making it a global network of connected things .
* We have mainly used the NodeMCU which is an open source development boards with ESP8266-12E chips. MQ-2 Gas Sensor is used to collect gas concentration measurements.
* This sensor data would be captured and sent to the ThingSpeak cloud for IoT based data acquisition.

**IoT devices:**

****

**Analysis o/p:**

* From the context of this work here we have discussed the performance of the different models, after the data preprocessing and feature extraction.
* Especially first part of this section deals with the data that has been generated by our device and the next part is the extension of our model on the open-source dataset.
* Here are the insights and analytics we got from the data that we generated

**Conclusion:**

* Air quality is a critical issue that straightforwardly influences human wellbeing.
* Air quality information are gathered remotely from checking bits that are outfitted with a variety of vaporous also, meteorological sensors.
* This information are investigated and utilized as a part of anticipating fixation estimations of contaminations utilizing savvy machine to machine stage.
* The stage comprises of a ML-based calculations to construct the estimating models by training from the gathered information.
* However we can conclude that we can use gradient boosting method for prediction, preferably XGBoost because of its level-wise approach and helps in building a model which has low bias and low variance.
* ARIMA performs pretty well as a forecasting model, what can be used as to make a daily forecast just like regular weather forecasting.

**HTML:**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<title>Air Pollution Tracker</title>

<meta name="description" content="">

<link rel="icon" href="back/My Post (1).jpg" sizes="32x32">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="style.css">

<link rel="stylesheet"

href="https://fonts.googleapis.com/css?family=Montserrat|Concert One|Prompt">

<script src="script.js" defer></script>

</head>

<body>

<video autoplay muted loop id="vid">

<source src="back/Lake Kindness Quote Serene & Calm Zoom Background (1).mp4" type ="video/mp4">

</video>

<div class="app-wrap">

<header>

<h1>Air Quality Index Tracker</h1>

<input type="text" placeholder="Enter a Location..." class="search-box" />

</header>

<div class="location">

<div class="city">Enter a Location</div>

<div class="date">Current Date</div>

</div>

<div class="coordinates">

<div class="lat">Lat.</div>

<span id="lat-value">TBD</span>

<div class="lon">Long.</div>

<span id="lon-value">TBD</span>

</div>

<div class="weather">

<span id="temp">TBD&deg;C</span>

</div>

<hr>

<div class="parameters">

Air Quality Parameters

</div>

<hr>

<div class="air-quality">

<div class="air-parameters">

<div class="o3">Ozone(o3)</div>

<hr>

<div class="co">CO</div>

<hr>

<div class="s02">SO2</div>

<hr>

<div class=n02>NO2</div>

<hr>

<div class="pm10">PM10</div>

<hr>

<div class="pm25">PM2.5</div>

</div>

<div class = "last-update">

<div class="o3-time">last-update</div>

<hr>

<div class="co-time">last-update</div>

<hr>

<div class="s02-time">last-update</div>

<hr>

<div class=n02-time>last-update</div>

<hr>

<div class="pm10-time">last-update</div>

<hr>

<div class="pm25-time">last-update</div>

</div>

<div class="parameters-options">

<div class="o3-value">TBD µg/m³</div>

<hr>

<div class="co-value">TBD µg/m³</div>

<hr>

<div class="so2-value">TBD µg/m³</div>

<hr>

<div class="no2-value">TBD µg/m³</div>

<hr>

<div class="pm10-value">TBD µg/m³</div>

<hr>

<div class="pm25-value">TBD µg/m³</div>

</div>

</div>

</div>

</body>

</html>

**CSS:**

\*, \*::after, \*::before{

margin: 0px;

padding: 0px;

box-sizing: border-box;

}

body{

font-family: sans-serif;

display: flex;

justify-content: center;

align-items: center;

font-family: Work Sans, Geneva;

}

.app-wrap{

z-index: 1;

margin-top: 2rem;

display: flex;

flex-direction: column;

min-height: 80vh;

background-image: linear-gradient(to bottom, rgb(255, 216, 250), rgb(255, 255, 255));

max-width: 100%;

width:1200px;

box-shadow: inset 0 0px 10px #333;

border-radius: 10px ;

opacity: 0.8;

}

header{

display: grid;

justify-content: center;

align-items: center;

text-align: center;

padding: 10px;

font-family: Montserrat;

}

header input{

width: 100%;

align-items: center;

padding: 10px 15px;

border: none;

outline: none;

background-color: rgb(255, 255, 255);

color: #202020;

border-radius: 16px 0 16px 0;

border-bottom: 3px solid rgb(158, 158, 158);

margin-top: 10px;

font-size: 20px;

font-weight: 300;

}

header input :focus{

background-color: rgba(255,255,255,0.8);

}

.location{

display: flex;

flex-direction: column;

justify-content: center;

align-items: center;

}

.location .city{

font-weight: bolder;

font-size: 1.5rem;

font-family: Verdana;

margin-bottom: 10px;

}

.location .date{

font-weight: bold;

font-style: italic;

}

.coordinates{

margin: 10px;

display: flex;

justify-content: space-between;

}

.coordinates .lat{

font-weight: normal;

font-size: 1.5rem;

}

.coordinates .lon{

font-weight: normal;

font-size: 1.5rem;

}

#lat-value{

font-family: Montserrat;

font-size: 1.3rem;

font-weight: bold;

}

#lon-value{

font-family: Montserrat;

font-size: 1.3rem;

font-weight: bold;

}

.weather{

text-align: center;

margin: 20px;

font-weight: bold;

color: rgb(0, 255, 213);

font-size: 50px;

text-shadow: 2px 5px rgb(77, 77, 77);

font-family: Concert One;

}

.air-quality{

margin: 10px;

display: flex;

flex-direction: row;

justify-content: space-between;

}

.parameters{

margin: 10px;

font-size: 20px;

color: crimson;

font-weight: bold;

display: flex;

justify-content: center;

align-items: center;

}

.air-parameters{

font-weight: bold;

font-family: Prompt;

font-size: 1.5rem;

display: flex;

flex-direction: column;

justify-content: space-around;

}

.parameters-options{

font-weight: bold;

font-family: Geneva;

display: flex;

flex-direction: column;

justify-content: space-between;

}

.last-update{

display: flex;

flex-direction: column;

justify-content: space-between;

font-style: oblique;

font-weight: lighter;

font-size:12px;

}

#vid{

position: fixed;

min-width: 100%;

min-height: 100%;

}

**JavaScripit:**

const api={

key:"f6bf196015464b5fb8273e5522911cfc",

base:"https://api.openweathermap.org/data/2.5/",

base1: "https://api.openaq.org/v1/measurements"

}

let o3V = document.querySelector('.air-parameters .o3')

let coV = document.querySelector('.air-parameters .co')

let so2V = document.querySelector('.air-parameters .s02')

let no2V = document.querySelector('.air-parameters .n02')

let pm10V = document.querySelector('.air-parameters .pm10')

let pm25V = document.querySelector('.air-parameters .pm25')

let o3T = document.querySelector('.last-update .o3-time')

let coT = document.querySelector('.last-update .co-time')

let so2T = document.querySelector('.last-update .s02-time')

let no2T = document.querySelector('.last-update .n02-time')

let pm10T = document.querySelector('.last-update .pm10-time')

let pm25T = document.querySelector('.last-update .pm25-time')

const searchbox=document.querySelector('.search-box');

searchbox.addEventListener('keypress',setQuery);

function setQuery(evt)

{

if(evt.keyCode==13)

{

getResults(searchbox.value);

//console.log(searchbox.value);

}

}

function getResults(query)

{

try{

fetch(`${api.base}weather?q=${query}&units=metric&APPID=${api.key}`)

.then(weather=>{

return weather.json();

}).then(displayResults);

} catch{

alert('City Not Found')

}

}

function displayResults\_aq(ap)

{

console.log(ap)

let o3 = document.querySelector('.parameters-options .o3-value');

o3.innerText = `${ap.results[0].value} µg/m³`;

let co = document.querySelector('.parameters-options .co-value')

co.innerText = `${ap.results[1].value} µg/m³`;

let so2 = document.querySelector('.parameters-options .so2-value')

so2.innerText = `${ap.results[2].value} µg/m³`;

let no2 = document.querySelector('.parameters-options .no2-value')

no2.innerText = `${ap.results[3].value} µg/m³`

let pm10 = document.querySelector('.parameters-options .pm10-value')

pm10.innerText = `${ap.results[4].value} µg/m³`;

let pm25 = document.querySelector('.parameters-options .pm25-value')

pm25.innerText = `${ap.results[5].value} µg/m³`;

o3V.innerText = `${ap.results[0].parameter}`;

coV.innerText = `${ap.results[1].parameter}`;

so2V.innerText = `${ap.results[2].parameter}`;

no2V.innerText = `${ap.results[3].parameter}`;

pm10V.innerText = `${ap.results[4].parameter}`;

pm25V.innerText = `${ap.results[5].parameter}`;

o3T.innerText = `${ap.results[0].date.local}`;

coT.innerText = `${ap.results[1].date.local}`;

so2T.innerText = `${ap.results[2].date.local}`;

no2T.innerText = `${ap.results[3].date.local}`;

pm10T.innerText = `${ap.results[4].date.local}`;

pm25T.innerText = `${ap.results[5].date.local}`;

}

function displayResults(weather)

{

try{

fetch(`${api.base1}?coordinates=${weather.coord.lat},${weather.coord.lon}`)

.then(ap =>{

return ap.json();

}).then(displayResults\_aq)

} catch{

alert('Enter More Precise Location');

}

console.log(weather);

let lat = document.getElementById('lat-value');

lat.innerText = `${weather.coord.lat}`;

let lon = document.getElementById('lon-value');

lon.innerText = `${weather.coord.lon}`;

let temp = document.getElementById('temp');

temp.innerText =`${weather.main.temp}°C`;

let location = document.querySelector('.location .city')

location.innerText = `${weather.name},${weather.sys.country}`

let date = new Date();

let datenow = document.querySelector('.location .date')

datenow.innerText = dateBuilder(date);

function dateBuilder(d)

{

let months=[

"January","February","March","April","May","June","July","August","September",

"October","November","December",

];

let days=[

"Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"

];

let day=days[d.getDay()];

let date=d.getDate();

let month=months[d.getMonth()];

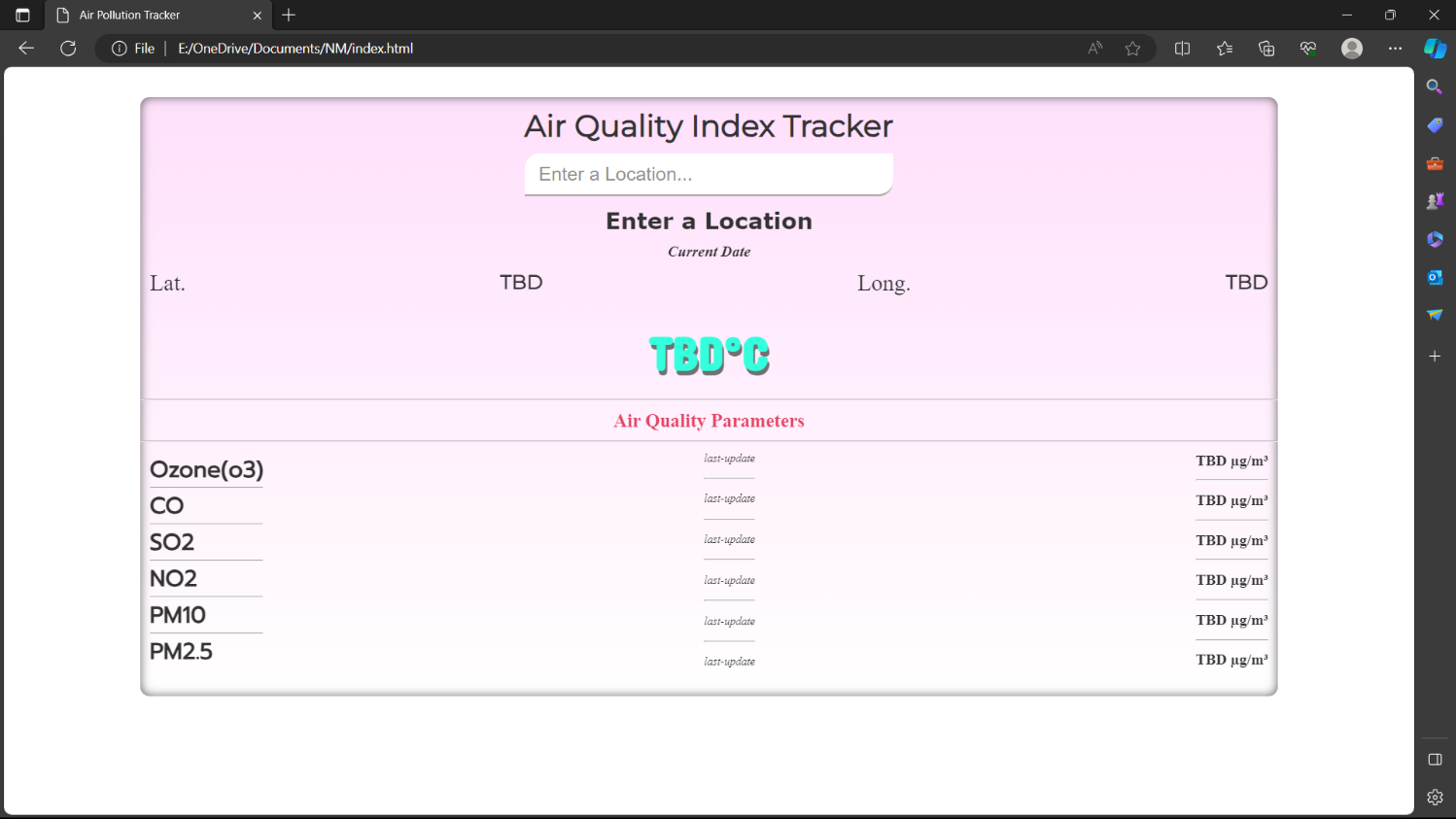
let year=d.getFullYear();

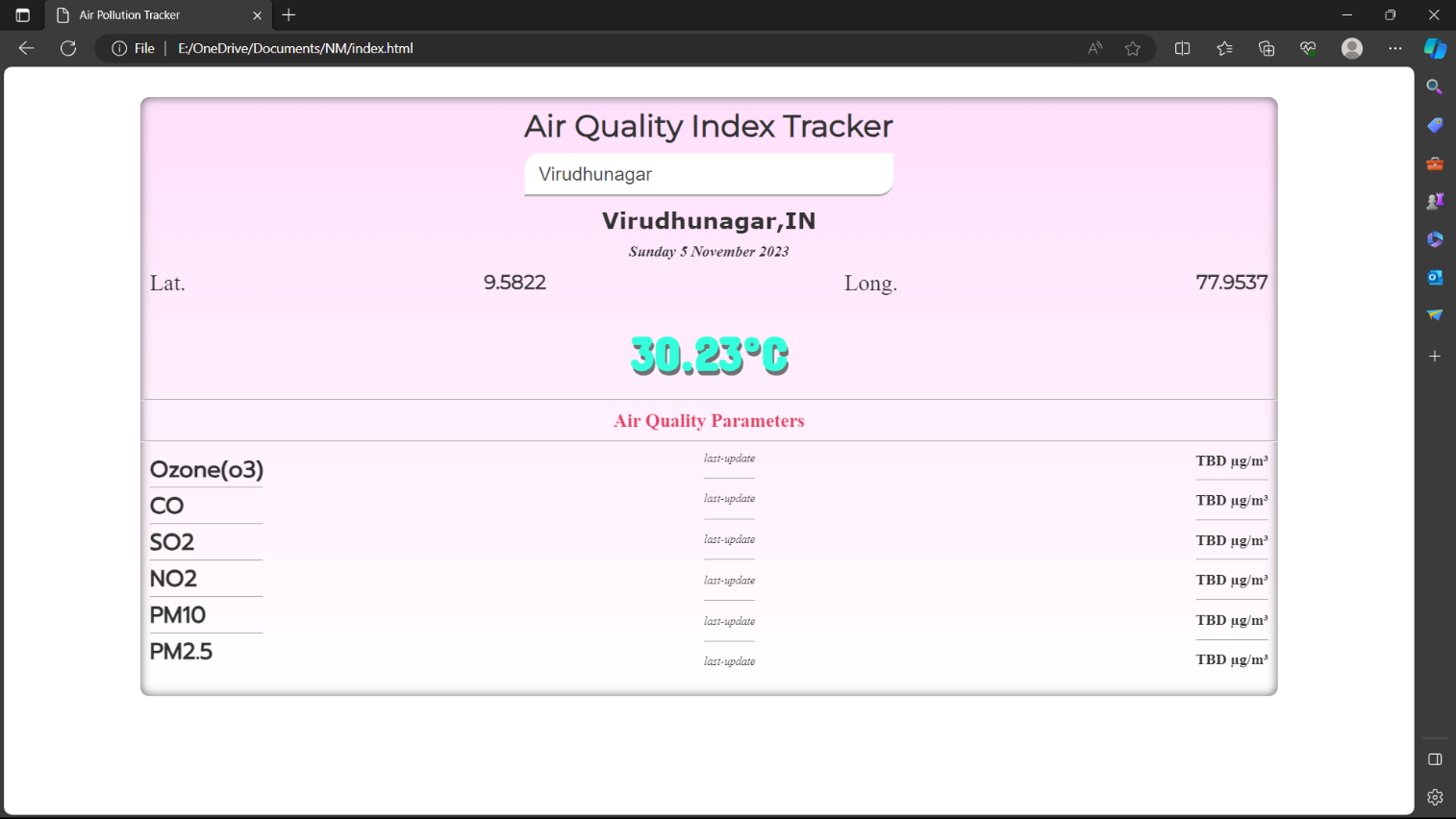
return `${day} ${date} ${month} ${year}`;

}

}

**Output:**

****

****