IOT PHASE 5

AIR QUALITY MONITORING

INTRODUCTION:

Air pollution has been causing serious problems in our modern society and the regulations on the emission standards of pollutants are becoming more severe year by year. It is well-known that carbon monoxide and nitrogen oxides are a prominent part of the polluting agents of the atmosphere and they are among the most dangerous chemical species for human health present in the atmosphere.

Their detection is therefore of great interest in various fields like air quality control, combustion processes and engine emissions. This is the reason why increasing requirements are being made for monitoring the gas pollution in urban area, fixing the concentration alarm threshold at about 13 ppm for CO and about 0.1 ppm for NO_2.

Moreover, in the last years SnO_2 gas sensors have been subjected to extensive research and development. It has been demonstrated that the sensing characteristics of a semiconductor gas sensor can be improved by several factors like the decreasing of crystallite size, the valency control of SnO_2 or the addition of noble metal catalysers .

In the last case Pd is often used as catalytic element; it is known to modify surface reactions which are responsible for the electrical signal of SnO_2. The objective of this paper is to find the best response to CO and NO_2 gases given by pure and Pd-modified SnO_2 sensors prepared by means of the sol–gel process and then report about the application of sol–gel technique for preparing integrated gassensors.

OBJECTIVES:

Air quality monitoring aims to protect public health by tracking and managing harmful pollutants, enforce environmental regulations, inform policymaking, support research, provide early warnings during pollution events, and raise public awareness about air quality issues. It's crucial for ensuring clean air and a healthier environment.

PROJECT DEVELOPMENT PART 2

Building a project by developing Air Quality Monitoring platforms and mobile app using Html java script etc…

AIM:

The aim of creating an app for air quality monitoring using HTML, CSS, and JavaScript is to provide users with a convenient way to access and monitor real-time air quality information.

This app will enable users to stay informed about their surrounding air quality and take necessary precautions if required.

ALGORITHM:

Here's a high-level algorithm for developing the app:

1. User Interface:

   - Design and create the user interface using HTML and CSS.

   - Include elements like maps, charts, and data visualizations to display air quality information effectively.

2. Data Integration:

   - Research and choose reliable air quality data sources, such as APIs provided by government agencies or environmental organizations.

   - Implement JavaScript code to fetch air quality data from the chosen API(s).

   - Process and sanitize the data to extract relevant information like pollutant levels, air quality index, location, and timestamp.

3. Data Presentation and Visualization:

   - Utilize JavaScript libraries or frameworks like Chart.js or D3.js to create visual representations of air quality data in the form of charts, graphs, or maps.

   - Display the real-time or historical air quality information to the user in a visually appealing and easy-to-understand manner.

4. Location Services:

   - Use JavaScript's Geolocation API or relevant plugins to retrieve the user's current location.

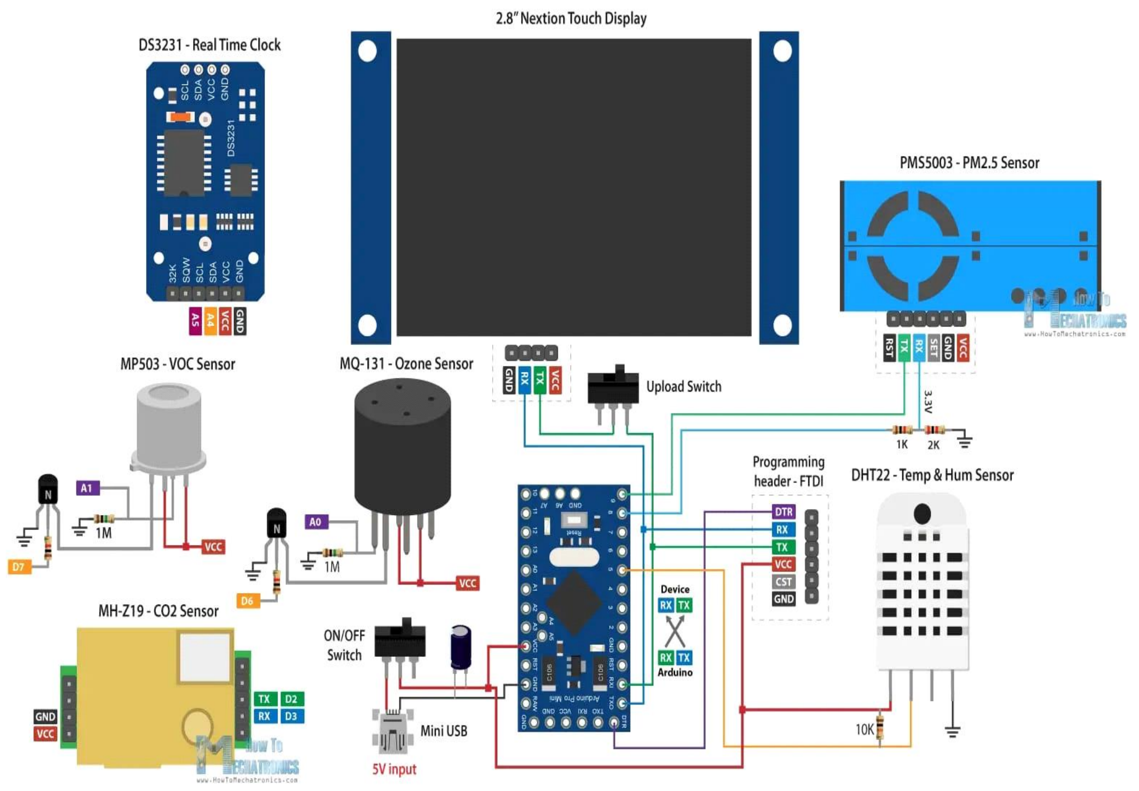
   - Integrate the obtained location coordinates with air quality data requests to provide customized information based on the user's location.

5. Notifications and Alerts:

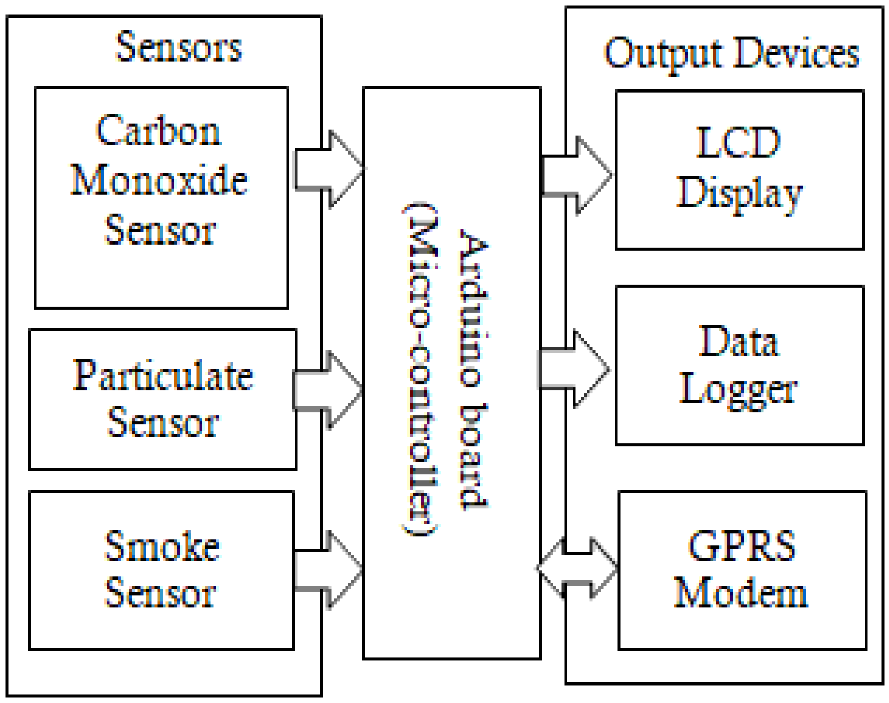
- Implement push notifications or in-app alerts to inform users about significant changes in air quality, such as high pollution levels or health warnings.

AIR QUALITY MONITORING REQUIREMENTS:

- Accurate sensors
- Real-time data processing
- Data logging
- Alerting system
- Remote access
- Calibration
- Data visualization
- Geospatial info
- Weather integration
- Networked sensors
- Power options
- Portability
- Data sharing
- AQI integration



BLOCK DIAGRAM:



DEVELOPMENT OF AN APP:

To create an Android platform that provides users with access to air quality monitoring, you can develop a mobile application using JavaScript, HTML, and CSS with a hybrid framework like Ionic or React Native. Here's a general outline of the development process:

1. Project Setup:

- Install the necessary software development kits (SDKs) and tools for Android app development.
- Set up a project in your preferred integrated development environment (IDE).

2. UI Design:

- Use HTML and CSS to design the user interface (UI) of the application.
- Consider incorporating elements like maps, charts, and data visualization to display air quality information effectively.
- Ensure the UI is intuitive, user-friendly, and responsive across different screen sizes.

3. Air Quality Data Integration:

- Research reliable air quality data sources, such as APIs provided by government agencies or environmental organizations.
- Implement JavaScript code to fetch air quality data from the chosen API(s).

- Process and sanitize the data to extract relevant information like pollutant levels, air quality index, location, and timestamp.

4. Data Presentation and Visualization:

- Utilize JavaScript libraries or frameworks like Chart.js or D3.js to create visual representations of air quality data in the form of charts, graphs, or maps.
- Display the real-time or historical air quality information to the user in a visually appealing and easy-to-understand manner.

5. Location Services:

- Use JavaScript's Geolocation API or relevant plugins to retrieve the user's current location.
- Integrate the obtained location coordinates with air quality data requests to provide customized information based on the user's location.

6. Notifications and Alerts:

- Implement push notifications or in-app alerts to inform users about significant changes in air quality, such as high pollution levels or health warnings.

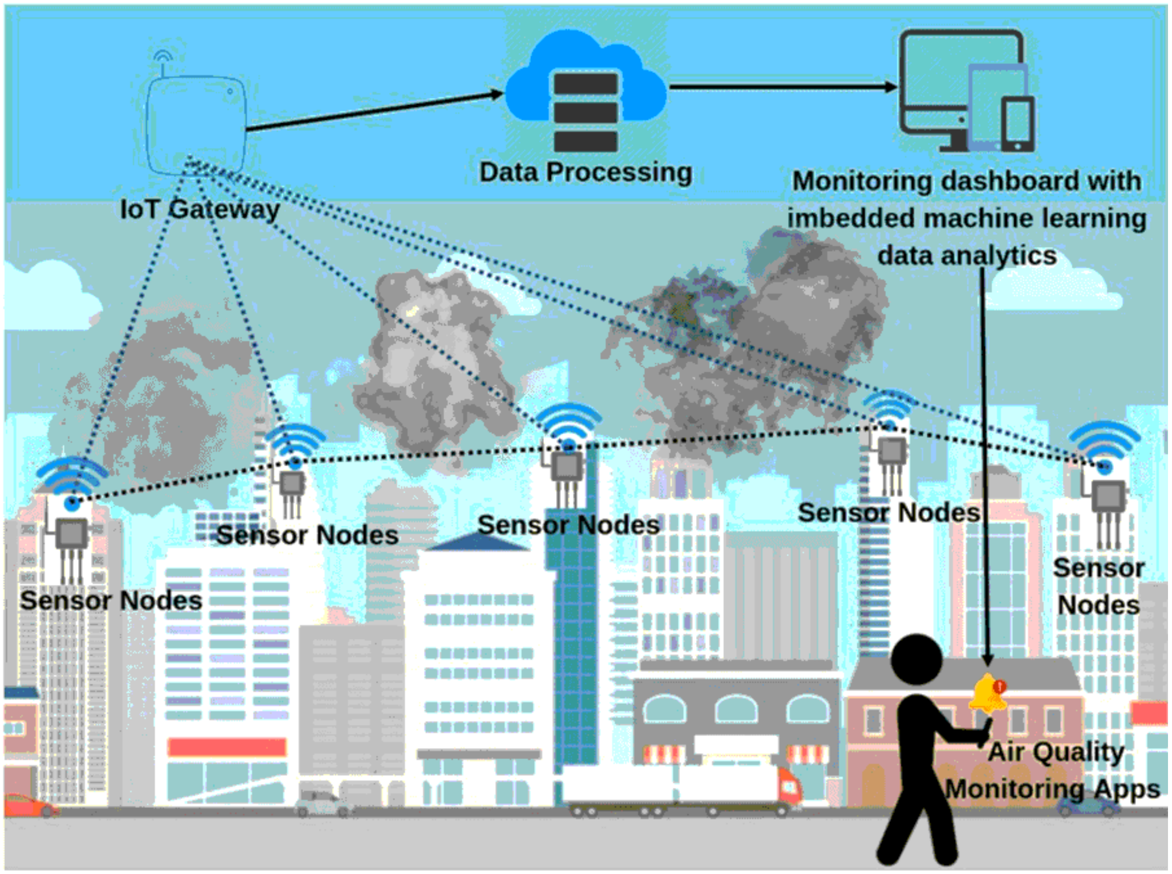
7. Testing and Deployment:

- Thoroughly test the application on various Android devices and emulator configurations to ensure functionality and compatibility.
- Sign the APK file and prepare it for deployment on the Google Play Store or other distribution platforms.

8. Continuous Maintenance:

- Regularly update the application to keep up with changes in air quality data sources, API updates, or user feedback.

- Address bug fixes, implement new features, and enhance the overall performance and user experience.


Data Processing
IoT Gateway
Monitoring dashboard with imbedded machine learning data analytics
Sensor Nodes
Sensor Nodes
Sensor Nodes
Sensor Nodes
Sensor Nodes
Air Quality Monitoring Apps

PROGRAM:(Using HTML, CSS, and JavaScript):

Here's a program using HTML, CSS, and JavaScript to create a simple app for checking the quality of air and keeping our environment dust -

free so that every habitat can breathe free and can be aware of the quality of the air now a days.

HTML CODE:

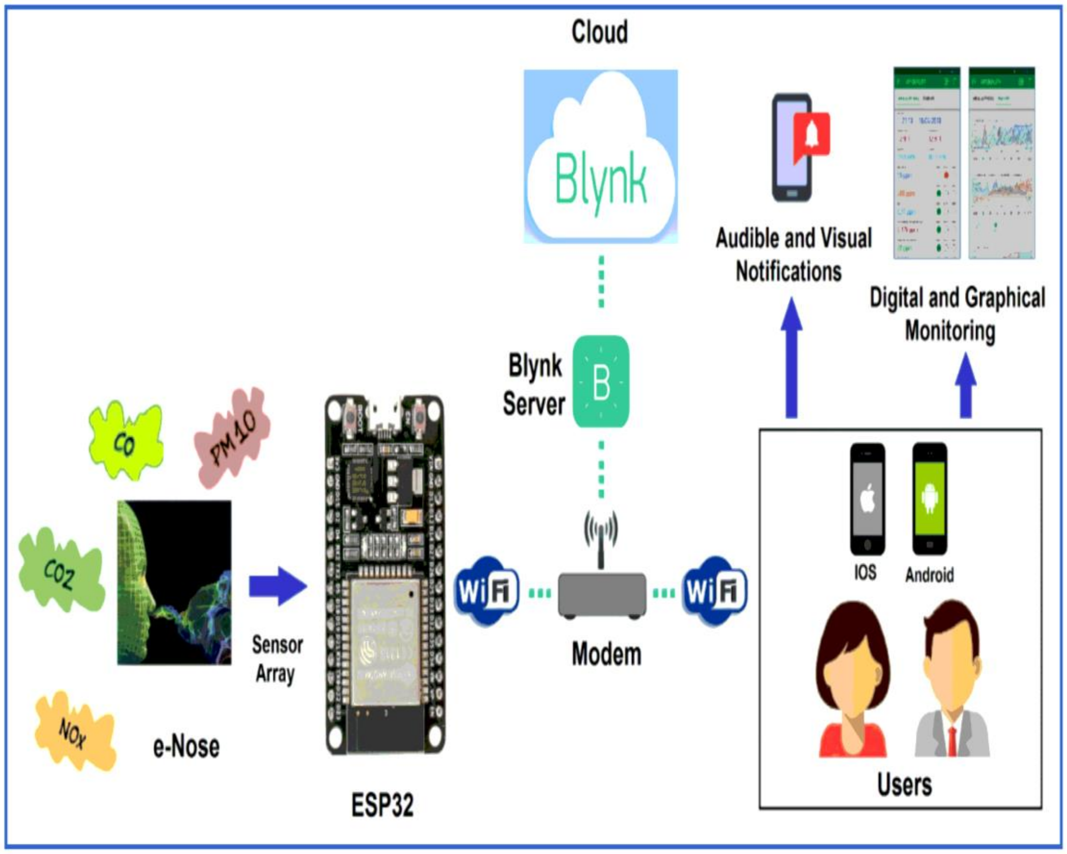
Certainly! Here's an example of html code for creating an app for air quality monitoring:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Air Quality Monitoring App</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <header>
    <h1>Air Quality Monitoring App</h1>
  </header>

  <section id="mapSection">
    <div id="map"></div>
  </section>
  <section id="dataSection">
    <h2>Air Quality Information</h2>
    <ul id="dataList"></ul>
  </section>

  <script src="script.js"></script>
</body>
</html>
```

CSS (styles.css):

```css
/* Add your styles here */
body {
  font-family: Arial, sans-serif;
}

header {
  background-color: #333;
  color: #fff;
  padding: 20px;
}
```

```
#map {
  width: 100%;
  height: 400px;
}

#dataList {
  list-style-type: none;
  padding: 0;
  margin: 0;
}
#dataList li {
  padding: 10px;
  border-bottom: 1px solid #ccc;
}
#dataList li:last-child {
  border-bottom: none;}
```



JavaScript (script.js):

```
// Sample data for demonstration purposes
const airQualityData = [
  { city: 'New York', aqi: 52 },
  { city: 'London', aqi: 40 },
  { city: 'Beijing', aqi: 168 },
  { city: 'Tokyo', aqi: 32 }
];

// Function to render the air quality data
function renderAirQualityData() {
  const dataList = document.getElementById('dataList');

  airQualityData.forEach(data => {
    const listItem = document.createElement('li');
    listItem.innerHTML = `${data.city}: ${data.aqi}`;
    dataList.appendChild(listItem);
  });
}

// Initialize the map and render air quality data on page load
window.addEventListener('load', () => {
  // Code to initialize and display the map goes here
  // ...

  renderAirQualityData();
});
```

EXPLANATION OF CODE:

In the above code, we have the HTML structure for the app as explained before. We also have some basic CSS styles to make it visually appealing.

The JavaScript code includes a sample data array (`airQualityData`) for demonstration purposes. You can replace it with actual data from an API or any other source. The `renderAirQualityData()` function dynamically generates a list of air quality data based on the provided array.

Please make sure to include appropriate links to the CSS and JavaScript files in your HTML file.

RASPBERRY PI PROGRAM:

Air quality monitoring using a Raspberry Pi can involve various sensors. One common sensor used for air quality monitoring is the MQ series gas sensor. Below is a sample Python code for monitoring air quality using the MQ-135 gas sensor with a Raspberry Pi. This code reads the analog output of the sensor and estimates the air quality in terms of the air quality index (AQI):

python

```
import RPi.GPIO as GPIO
import time

# Define the GPIO pin where the analog output of the MQ-135 sensor is connected
MQ_PIN = 0

# Set the GPIO mode to BCM
GPIO.setmode(GPIO.BCM)

def read_mq():
```

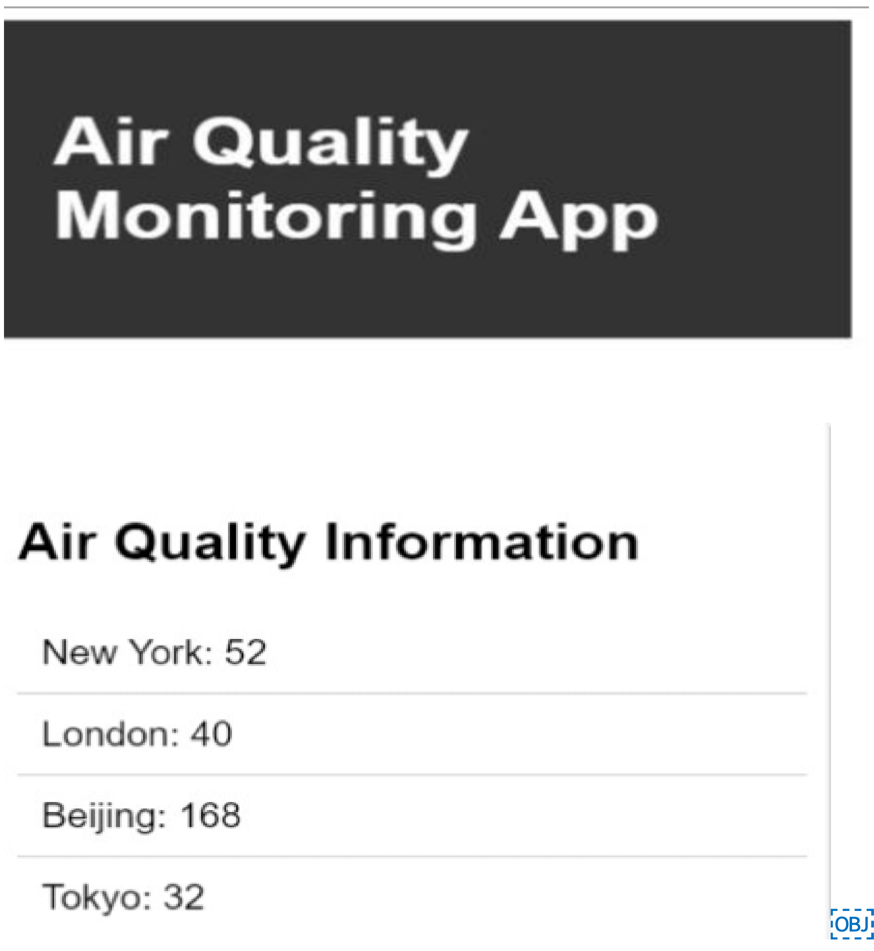
```
try:
    while True:
        # Open a file to store the sensor values
        with open("air_quality_log.txt", "a") as log_file:
            # Initialize GPIO
            GPIO.setup(MQ_PIN, GPIO.IN)
            time.sleep(2)  # Allow the sensor to warm up

            # Read the analog value from the sensor
            value = 0
            GPIO.setup(MQ_PIN, GPIO.OUT)
            GPIO.output(MQ_PIN, GPIO.LOW)
            time.sleep(0.1)
            while GPIO.input(MQ_PIN) == GPIO.LOW:
                continue
            start_time = time.time()
            while GPIO.input(MQ_PIN) == GPIO.HIGH:
                continue
            end_time = time.time()

            # Calculate sensor resistance and air quality index
            pulse_duration = end_time - start_time
```

```
                ratio = pulse_duration / 2 / 30  # 30 is a typical value for clean air
                sensor_value = (1 - ratio) * 10000

                # Log the sensor value
                log_file.write(f"{time.ctime()}: Sensor Value: {sensor_value:.2f}\n")
                print(f"{time.ctime()}: Sensor Value: {sensor_value:.2f}")

                # Adjust this threshold according to your specific sensor and air quality standards
                if sensor_value > 300:
                    print("Air quality is poor")
                else:
                    print("Air quality is good")

                time.sleep(60)  # Read the sensor every minute

    except KeyboardInterrupt:
        GPIO.cleanup()

if __name__ == "__main__":
    read_mq()
```

This code initializes the GPIO, reads the analog output from the MQ-135 sensor, calculates the sensor resistance, and estimates air quality based on a threshold value (300 in this example). The sensor values are logged to a file for reference.

Remember to connect the MQ-135 sensor properly to your Raspberry Pi and adjust the threshold value based on your specific sensor and air quality standards. You may also want to use more advanced calibration techniques for accurate results.

OVERALL OUTPUT:



Air Quality Monitoring App

Air Quality Information

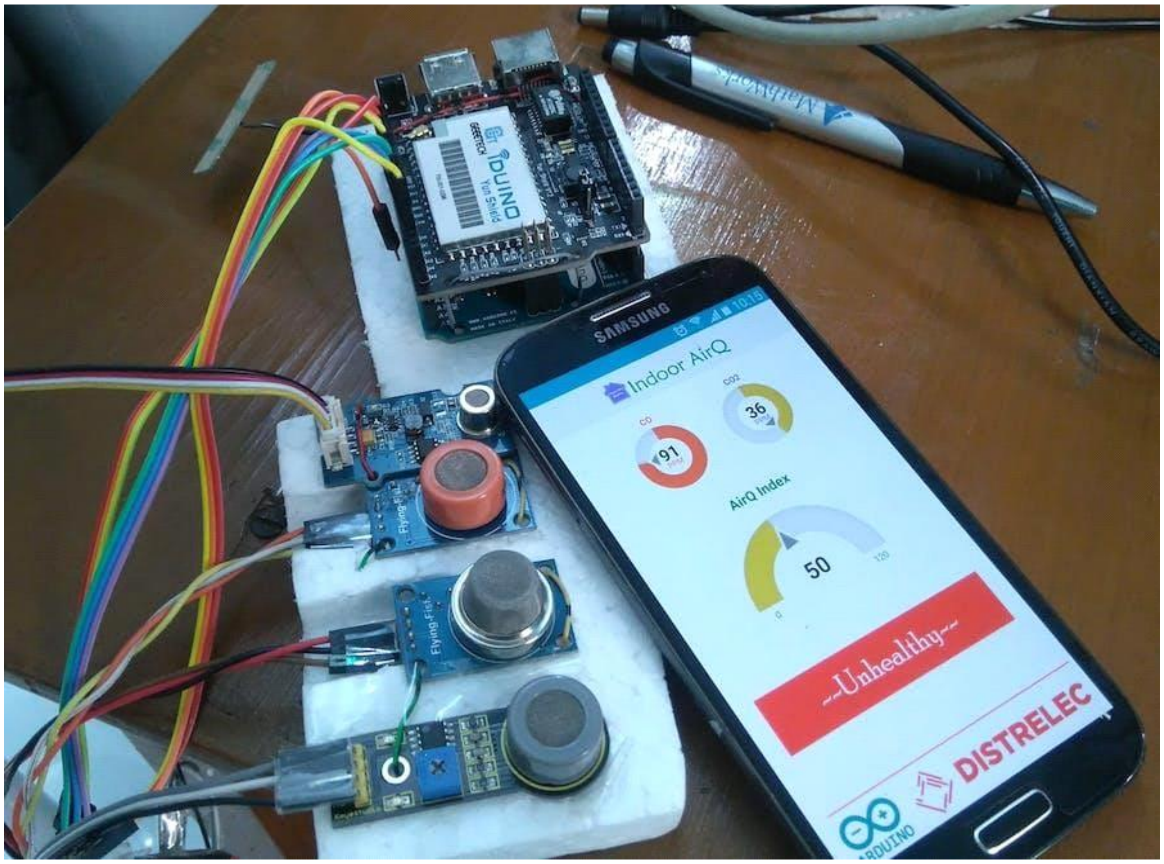
New York: 52

London: 40

Beijing: 168

Tokyo: 32

SAMPLE OUTPUT:



AIR QUALITY MONITOR NEEDS:

1. Environmental protection
2. Public health
3. Industrial compliance
4. Traffic management
5. Agriculture
6. Indoor air quality

7. Research

8. Emergency response

9. Smart cities

10. Weather forecasting

11. Emission reduction

12. Real estate choices

13. Asthma and allergy management

14. Energy efficiency

KEY FEATURES:

1. Sensor Integration: IoT devices incorporate a range of sensors to measure various air pollutants such as particulate matter (PM2.5, PM10), volatile organic compounds (VOCs), nitrogen dioxide (NO2), carbon monoxide (CO), and more.

2. Real-time Data: Continuous and real-time data collection, which enables immediate awareness of air quality changes.

3. Data Transmission: Wireless connectivity for transmitting data to a central server or cloud platform for remote monitoring and analysis.

4. Geographic Mapping: Integration with GPS or location-based services to provide spatial information and map air quality variations.

5. Mobile Accessibility: Access to air quality information through mobile apps or web interfaces, allowing users to check air quality in real time.

6. Historical Data Storage: Storage of historical air quality data for trend analysis and long-term monitoring.

7. Alerts and Notifications: Automated alerts and notifications sent to users when air quality levels exceed predefined thresholds, allowing for timely action.

8. Integration with Weather Data: Combining air quality data with weather conditions to provide a more comprehensive picture of environmental factors affecting air quality.

9. User-Friendly Interface: Intuitive interfaces for easy understanding of air quality data, with visualization tools such as charts and graphs.

10. Scalability: The ability to scale the monitoring network up or down to cover a wide range of areas and locations.

11. Environmental Parameters: Monitoring additional environmental parameters like temperature, humidity, and wind speed for a more accurate assessment of air quality.

12. Data Analytics: Advanced data analytics and machine learning algorithms to identify trends, correlations, and potential sources of pollution.

13. Historical Reporting: Generating reports and summaries of historical data for regulatory compliance and public information.

14. Cost-Effectiveness: Efficient power management and cost-effective hardware to ensure sustainability.

15. Customization: Flexibility to customize the system to suit specific monitoring needs, whether in urban areas, industrial zones, or indoor environments.

ADVANTAGES:

- Provides users with easy access to real-time air quality information
- Helps users make informed decisions about outdoor activities and health precautions
- Raises awareness about air pollution and its impact on health
- Encourages individuals to take steps towards reducing pollution and improving air quality

DISADVANTAGES:

- Reliance on third-party air quality data sources, which may not always be completely accurate or up-to-date
- May require a stable internet connection to fetch and update air quality data
- Limited functionality compared to specialized air quality monitoring devices or official monitoring systems

CONCLUSION:

It's important to note that the actual program implementation might involve more detailed coding examples and specific libraries, APIs, or frameworks depending on your requirements and chosen development approach.