**AIR QUALITY MONITORING USING IOT**

**(AQM)**

**Problem definition:**

**Objective:** To set up an IoT device to measure air quality parameters and create a platform that provides real-time air quality information to the public for raising awareness and its impact on public health.

**Design thinking:**

**1. Select Air Quality Sensors:**

- Choose appropriate air quality sensors based on the parameters you want to measure (e.g., PM2.5, PM10, CO2, CO, NO2, SO2, O3, temperature, humidity, VOCs).

- Ensure the selected sensors are accurate, reliable, and suitable for long-term deployment.

**2. IoT Device Design:**

- Develop or select IoT devices that can host the chosen sensors and communicate data to a central server or cloud platform.

- Consider power sources (battery or solar), data transmission protocols (Wi-Fi, cellular, Lora WÀN), and enclosure design for outdoor or indoor placement.

**3. Data Collection and Transmission:**

- Integrate the selected sensors into the IoT devices to collect air quality data.

- Implement data transmission mechanisms to send data to a central server or cloud platform. Ensure secure and reliable data transfer.

**4. Cloud Platform or Server:**

- Set up a cloud platform or server to receive, store, and process air quality data. Implement a database to manage historical data records.

**5. Data Processing and Analysis:**

- Develop data processing algorithms to calculate air quality indices (e.g., AQI) based on collected sensor data.

- Analyze historical data to identify trends and patterns.

**6. Real-Time Visualization:**

- Create a web-based dashboard or mobile app that displays real-time air quality data in an easy-to-understand format.

- Include charts, maps, and other visualizations to engage users.

**7. Public Accessibility:**

- Make the air quality data publicly accessible through the dashboard or app.

- Provide user-friendly features for searching, filtering, and exploring data.

**8. Alerts and Notifications:**

- Implement alerting mechanisms to notify users when air quality levels exceed predefined thresholds.

- Offer options for users to receive alerts via email, SMS, or mobile app notifications.

**9. Community Engagement:**

- Encourage community involvement by allowing users to report air quality concerns or contribute data from their own devices.

- Promote community-driven initiatives to address air quality issues.

**10. Data Sharing:**

- Consider sharing anonymized and aggregated air quality data with local environmental agencies, research institutions, and government bodies to support policy decisions and research.

**11. Scalability:**

- Design the system to scale as more IoT devices are deployed to monitor air quality in additional locations.

**12. Maintenance and Calibration:**

- Establish a maintenance schedule for sensor calibration and device upkeep to ensure data accuracy.

**13. Security and Privacy:**

- Prioritize security measures to protect user data and the integrity of the air quality monitoring system.

- Address privacy concerns by collecting only necessary data and implementing privacy policies.

**14. Outreach and Promotion:**

- Promote the availability of air quality data through social media, local news, and community outreach programs.

- Collaborate with environmental organizations and health agencies to raise awareness.

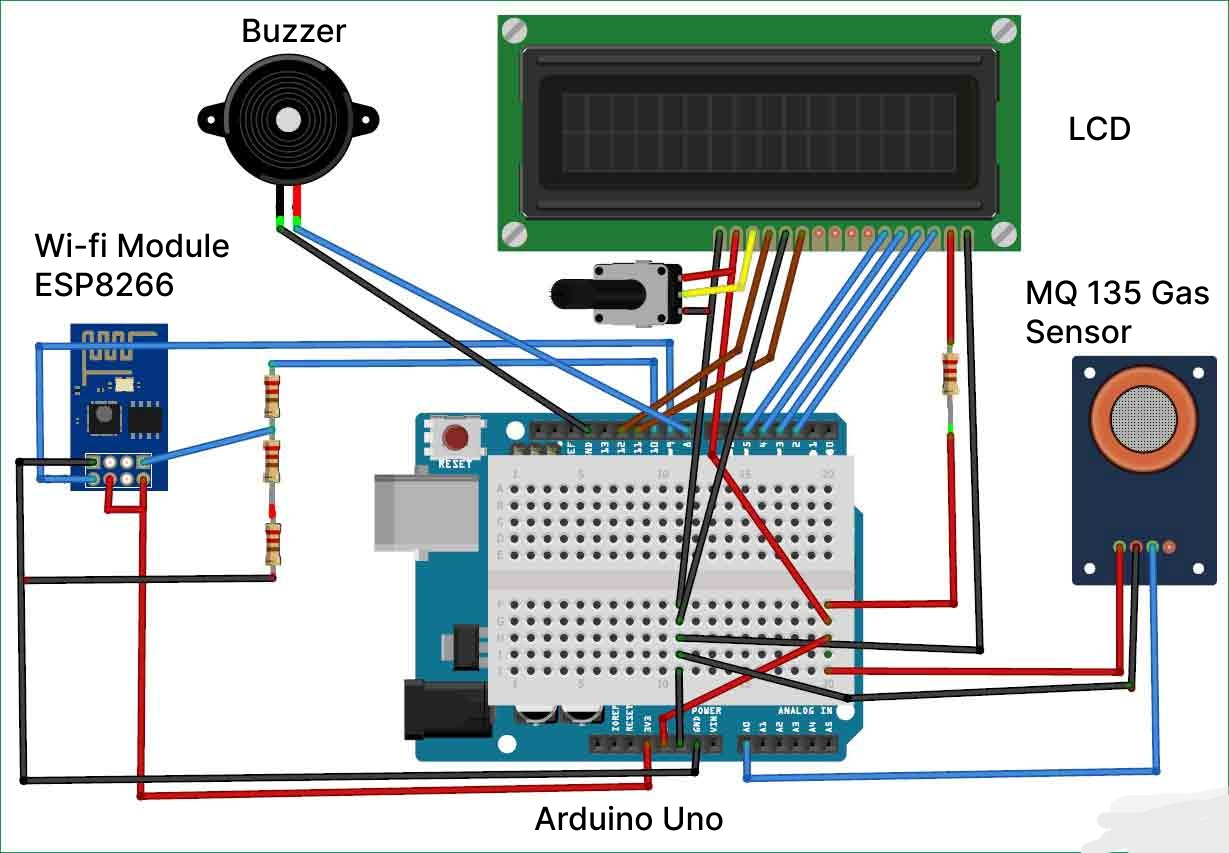
In conclusion, implementing this system may require collaboration with environmental agencies, local government, non-profit organizations, and community volunteers. By making air quality data easily accessible and engaging the public, you can contribute to raising awareness and driving positive changes in air quality management and public health.

**AIR QUALITY MONITORING USING IOT (PHASE 2)**

**DESIGN:**

In this project we made an IoT Based Air Pollution MonitoringSystem in which we will monitor the Air Quality over a webserver using internet and will trigger an alarm when the air quality goes down beyond a certain level, means when there are sufficient amount of harmful gases are present in the air like CO2, smoke, alcohol, benzene and NH3. It will show the air quality in PPM on the LCD and as well as on webpage so that we can monitor it very easily.

The block diagram below is the general design of our solution.



**Figure 1-Block Diagram of Air Quality Monitoring using Arduino**

**STEPS TO CREATE THE AIR QUALITY MONITORING IOT PROJECT:**

Creating an IoT-based air pollution monitoring system using an Arduino Uno involves several steps. Here's a high-level overview of the process:

**1. Gather Components:**

* Arduino Uno board
* Air quality sensor (e.g., MQ series)
* Wi-Fi module (e.g., ESP8266)
* Power supply
* Jumper wires
* Breadboard or custom PCB (Printed Circuit Board)

**2. Assemble Hardware:**

* Connect the air quality sensor to the Arduino Uno using the appropriate pins and power supply.
* Connect the Wi-Fi module to the Arduino Uno.

**3. Set Up Arduino IDE:**

* Install the Arduino IDE on your computer.
* Install the necessary libraries for the air quality sensor and the Wi-Fi module.

**4. Write Arduino Code:**

* Write code for the Arduino Uno to read data from the air quality sensor.
* Code should also include Wi-Fi connectivity to send data to a server or cloud platform.
* Ensure that your code is well-documented and handles error conditions gracefully.

**5. Create IoT Platform Account:**

* Sign up for an IoT platform or cloud service, such as AWS IoT, Google Cloud IoT, or Thing Speak.

**6. Set Up IoT Platform:**

* Create a new IoT device in your platform account.
* Retrieve API keys and credentials for your device.

**7. Configure Wi-Fi Module:**

* Program the Wi-Fi module with your network credentials and the IoT platform's server information.

**8. Test and Debug:**

* Upload the Arduino code to your Arduino Uno.
* Test the system to ensure its reading air quality data and sending it to the IoT platform correctly.
* Debug any issues you encounter.

**9. Data Visualization:**

* Set up data visualization and monitoring tools on your IoT platform. Create dashboards to display air quality data in real-time.

**10. Power Supply and Enclosure:**

* Make sure your system is powered properly. Consider placing the components in an enclosure for outdoor use.

**11. Deploy:**

* Install the monitoring system in the location where you want to measure air quality.

**12. Maintenance:**

* Regularly check and calibrate the air quality sensor. Monitor the system for any issues and perform updates as needed.

**WORKING EXPLANATION:**

* The MQ135 sensor can sense NH3, NOx, alcohol, Benzene, smoke, CO2 and some other gases, so it is perfect gas sensor for our Air Quality Monitoring Project.When we will connect it to Arduino then it will sense the gases, and we will get the Pollution level in PPM (parts per million).
* MQ135 gas sensor gives the output in form of voltage levels and we need to convert it into PPM. So for converting the output in PPM, here we have used a library for MQ135 sensor, it is explained in detail in “Code Explanation” section below.
* Sensor was giving us value of 90 when there was no gas near it and the safe level of air quality is 350 PPM and it should not exceed 1000 PPM.
* When it exceeds the limit of 1000 PPM, then it starts cause Headaches, sleepiness and stagnant, stale, stuffy air and if exceeds beyond 2000 PPM then it can cause increased heart rate and many other diseases.
* When the value will be less than 1000 PPM, then the LCD and webpage will display “Fresh Air”.
* Whenever the value will increase 1000 PPM, then the buzzer will start beeping and the LCD and webpage will display “Poor Air, Open Windows”.
* If it will increase 2000 then the buzzer will keep beeping and the LCD and webpage will display “Danger! Move to fresh Air”.

**FEATURES OF THE AIR QUALITY MONITOR:**

**1. Sensor Array:**

* Air quality monitoring systems typically consist of a variety of sensors that measure different air pollutants. These sensors can include those for measuring particulate matter (PM), gases (e.g., CO, NO2, SO2, and O3), volatile organic compounds (VOCs), and more.

**2. Data Logger:**

* A data logger or data acquisition system is used to collect and store data from the sensors over time. It records measurements at regular intervals and can store this data for analysis and reporting.

**3. Real-time Monitoring:**

* Many systems provide real-time monitoring capabilities, allowing users to access current air quality information on-site or remotely. This feature is crucial for immediate responses to air quality issues.

**4. Communication Interface:**

* Air quality monitoring systems often include communication interfaces to transmit data to a central server or cloud platform. Common communication methods include Ethernet, Wi-Fi, cellular, or satellite connectivity.

**5. Data Visualization:**

* To make the data more understandable, these systems often include data visualization tools, such as charts, graphs, and dashboards, which can display air quality trends and anomalies over time.

**6. Alarms and Alerts:**

* Air quality monitoring systems can be programmed to send alerts or alarms when pollutant levels exceed predetermined thresholds. These alerts can be sent via email, SMS, or other communication methods.

**7. Historical Data Analysis:**

* Systems usually store historical data, allowing for trend analysis, pattern recognition, and long-term assessments of air quality.

**8. Geographic Information System (GIS) Integration:**

* Some systems incorporate GIS to provide spatial information about air quality, which can be valuable for understanding regional variations in pollution.

**9. Weather Monitoring:**

* Weather conditions can significantly affect air quality. Some systems include weather monitoring sensors (e.g., temperature, humidity, wind speed, wind direction) to help correlate air quality data with meteorological factors.

**10. Calibration and Maintenance Alerts:**

* Sensors need regular calibration and maintenance to ensure accurate measurements. Monitoring systems may include features to schedule and track calibration and maintenance tasks.

**11. Remote Control and Configuration:**

* Users can often remotely configure and manage the monitoring system, adjusting settings, sensor calibrations, and data reporting parameters as needed.

**12. Data Storage and Backup:**

* Systems typically offer data storage options, including data backup to prevent data loss in case of system failure.

**13. Compliance Reporting:**

* Many air quality monitoring systems are used for regulatory compliance. They may include features to generate compliance reports and export data in standardized formats for regulatory authorities.

**14. Multiple Sensor Compatibility:**

* Some systems support a range of sensor types and brands, allowing for flexibility and customization based on specific monitoring needs.

**15. User Access Control:**

* To protect the integrity of the system and its data, access control and user authentication features may be included.

**16. Mobile Access:**

* Some systems offer mobile apps or mobile-responsive web interfaces to allow users to monitor air quality data on smartphones or tablets.

**17. Data Integration:**

* For comprehensive environmental monitoring, systems can integrate air quality data with data from other environmental sensors, such as water quality or noise level sensors.

**18. Remote Diagnostics:**

* Remote diagnostics can help identify and address system issues, reducing downtime and maintenance costs.

The specific features and capabilities of an air quality monitoring system can vary depending on the application, budget, and environmental conditions of the location where it is deployed. These systems are essential for assessing air quality, managing pollution, and making informed decisions to protect public health and the environment.

#include "MQ135.h"

#include <SoftwareSerial.h>

#define DEBUG true

SoftwareSerial esp8266(9,10); // This makes pin 9 of Arduino as RX pin and pin 10 of Arduino as the TX pin

const int sensorPin= 0;

int air\_quality;

#include <LiquidCrystal.h> 

LiquidCrystal lcd(12,11, 5, 4, 3, 2);

void setup() {

pinMode(8, OUTPUT);

lcd.begin(16,2);

lcd.setCursor (0,0);

lcd.print ("Air Quality Monitoring ");

lcd.setCursor (0,1);

lcd.print ("Sensor Warming ");

delay(1000);

Serial.begin(115200);

esp8266.begin(115200); // your esp's baud rate might be different

  sendData("AT+RST\r\n",2000,DEBUG); // reset module

  sendData("AT+CWMODE=2\r\n",1000,DEBUG); // configure as access point

  sendData("AT+CIFSR\r\n",1000,DEBUG); // get ip address

  sendData("AT+CIPMUair\_quality=1\r\n",1000,DEBUG); // configure for multiple connections

  sendData("AT+CIPSERVER=1,80\r\n",1000,DEBUG); // turn on server on port 80

pinMode(sensorPin, INPUT);        //Gas sensor will be an input to the arduino

lcd.clear();

}

void loop() {

MQ135 gasSensor = MQ135(A0);

float air\_quality = gasSensor.getPPM();

if(esp8266.available()) // check if the esp is sending a message 

  {

    if(esp8266.find("+IPD,"))

    {

     delay(1000);

     int connectionId = esp8266.read()-48; /\* We are subtracting 48 from the output because the read() function returns the ASCII decimal value and the first decimal number which is 0 starts at 48\*/ 

     String webpage = "<h1>IOT Air Pollution Monitoring System</h1>";

       webpage += "<p><h2>";   

       webpage+= " Air Quality is ";

       webpage+= air\_quality;

       webpage+=" PPM";

       webpage += "<p>";

     if (air\_quality<=1000)

{

  webpage+= "Fresh Air";

}

else if(air\_quality<=2000 && air\_quality>=1000)

{

  webpage+= "Poor Air";

}

else if (air\_quality>=2000 )

{

webpage+= "Danger! Move to Fresh Air";

}

webpage += "</h2></p></body>"; 

     String cipSend = "AT+CIPSEND=";

     cipSend += connectionId;

     cipSend += ",";

     cipSend +=webpage.length();

     cipSend +="\r\n";

     sendData(cipSend,1000,DEBUG);

     sendData(webpage,1000,DEBUG);

     cipSend = "AT+CIPSEND=";

     cipSend += connectionId;

     cipSend += ",";

     cipSend +=webpage.length();

     cipSend +="\r\n";

     String closeCommand = "AT+CIPCLOSE="; 

     closeCommand+=connectionId; // append connection id

     closeCommand+="\r\n";

     sendData(closeCommand,3000,DEBUG);

    }

  }

lcd.setCursor (0, 0);

lcd.print ("Air Quality is ");

lcd.print (air\_quality);

lcd.print (" PPM ");

lcd.setCursor (0,1);

if (air\_quality<=1000)

{

lcd.print("Fresh Air");

digitalWrite(8, LOW);

}

else if( air\_quality>=1000 && air\_quality<=2000 )

{

lcd.print("Poor Air, Open Windows");

digitalWrite(8, HIGH );

}

else if (air\_quality>=2000 )

{

lcd.print("Danger! Move to Fresh Air");

digitalWrite(8, HIGH);   // turn the LED on

}

lcd.scrollDisplayLeft();

delay(1000);

}

String sendData(String command, const int timeout, boolean debug)

{

    String response = ""; 

    esp8266.print(command); // send the read character to the esp8266

    long int time = millis();

    while( (time+timeout) > millis())

    {

      while(esp8266.available())

      {

        // The esp has data so display its output to the serial window 

        char c = esp8266.read(); // read the next character.

        response+=c;

      }  

    }

    if(debug)

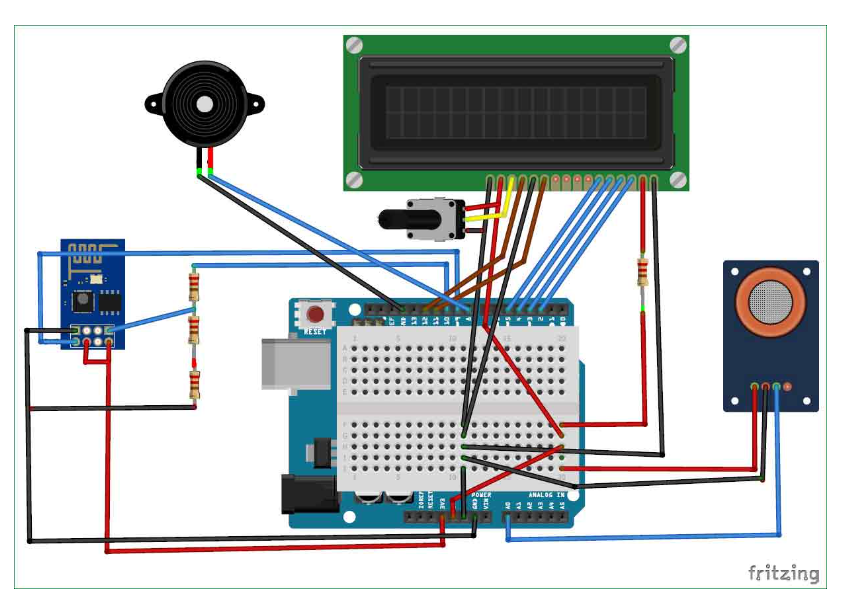
    {

      Serial.print(response);

    }

    return response;

}



Creating a real-time air quality monitoring platform using web development technologies (HTML, CSS, and JavaScript) requires both a frontend for displaying the data and a backend for receiving and processing data from IoT devices. A basic example is provided below of how to create such a platform:

**Frontend (HTML, CSS, and JavaScript):**

1. Create the HTML structure for the user interface.
2. Design the layout and elements to display air quality data.
3. Use JavaScript to fetch and display real-time data from the backend.
4. Update the UI periodically to reflect the latest air quality information.
5. Style the platform using CSS for a user-friendly interface.

Here's a simple example of a frontend using HTML, JavaScript, and CSS:

<!DOCTYPE html>

<html>

<head>

<title>Air Quality Monitor</title>

<link rel="stylesheet" type="text/css" href="styles.css">

</head>

<body>

<h1>Air Quality Monitor</h1>

<div id="airQualityData">

<p>Air Quality Index: <span id="aqiValue">Loading...</span></p>

<p>Location: <span id="location">Loading...</span></p>

<p>Last Updated: <span id="lastUpdated">Loading...</span></p>

</div>

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

</body>

</html>

**Backend (Server for Receiving Data):**

1. Set up a backend server to receive and process data from IoT devices.
2. Use a database (e.g., MongoDB or MySQL) to store historical air quality data.
3. Define API endpoints to receive and provide air quality data.
4. Store incoming data from IoT devices in the database.
5. Implement a mechanism to send real-time data to the frontend.

Here's a simplified example of a Node.js backend:

const express = require('express');

const app = express();

const bodyParser = require('body-parser');

const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost/air\_quality\_db', { useNewUrlParser: true, useUnifiedTopology: true });

const AirQualityData = mongoose.model('AirQualityData', {

aqi: Number,

location: String,

timestamp: Date

});

app.use(bodyParser.json());

// API endpoint to receive data from IoT devices

app.post('/api/air-quality-data', (req, res) => {

const { aqi, location } = req.body;

const airQualityData = new AirQualityData({

aqi,

location,

timestamp: new Date()

});

airQualityData.save()

.then(() => {

// Send the newly received data to connected clients

// You can use WebSockets or another real-time communication method here

})

.catch(err => {

console.error(err);

res.status(500).send('Error saving data');

});

});

app.listen(3000, () => {

console.log('Server is running on port 3000');

});

The style.css file used in Frontend:

/\* styles.css \*/

body {

font-family: Arial, sans-serif;

background-color: #f0f0f0;

}

h1 {

color: #333;

}

#airQualityData {

border: 1px solid #999;

padding: 10px;

background-color: #fff;

}

#aqiValue {

font-size: 24px;

color: #0074cc;

}

/\* Add more CSS rules for your specific styling needs \*/