INTERNET OF THINGS

UE22CS342AA3

ELECTIVE – 1

DATE : 17-11-2024

NAME : RAASHI BAFNA
SRN : PES2UG22CS422
SEM : 5 SEC : G

NAME : SHREYA SONI
SRN : PES2UG22CS535
SEM : 5 SEC : I

HACKATHON

REPORT

1) TITLE:

IoT Based Air Quality Monitoring System

Project Background:

In today’s era of rapid globalization, health has become an essential aspect of our lives. Various diseases, especially those related to respiratory issues such as asthma, cough, and other cardiovascular conditions, are increasingly being attributed to poor air quality. These health risks are further compounded by the negative impact of air pollution on the environment, contributing to global warming. The massive carbon emissions from vehicle combustion, industrial waste, uncontrolled burning of fossil fuels, and open-air burning in waste management sites have become key contributors to this issue.

Air Pollution in India

In 2019, a global air pollution survey revealed that 21 out of the 30 most polluted cities in the world were located in India, pushing the country to 5th place globally in terms of air pollution levels. According to data from IQAir.com, India's average Air Quality Index (AQI) stood at 152, with a PM2.5 concentration of 58.08μg/m³—five times higher than the World Health Organization's (WHO) recommended levels. While this represented some improvement from the previous year's figure of 72.54μg/m³, it still poses severe health risks to millions of people across the country. Approximately 50% of India’s pollution is attributed to industrial emissions, followed by 27% from vehicle emissions, 17% from crop burning, and 7% from domestic cooking.

The impact of this pollution is devastating. Over 2 million lives are lost each year in India due to conditions related to air pollution. In urban areas, industrial emissions and vehicular pollution dominate, while in rural areas, pollution mainly arises from the burning of organic materials used as cooking fuel. The practice of burning crop residue in fields, particularly during the autumn and winter months, exacerbates the issue. This method is cost-effective compared to alternatives like ploughing the residue back into the soil. Unfortunately, it often involves burning garbage alongside crop residue, further polluting the air. This

combination of factors has placed India as the third-largest emitter of greenhouse gases globally, behind China and the United States.

Sources of Air Pollution in India

The burning of fuel in domestic stoves, often made from wet mixtures of wood, dried leaves, hay, and animal dung, is a significant source of pollution. These fuels release pollutants five times more harmful than coal when burned. It is estimated that more than 100 million households use these stoves up to three times a day, every day of the week. In many remote areas, access to electricity or cleaner fuels is limited, which forces people to rely on these traditional stoves. Even in urban areas where electricity is available, these stoves remain in use due to cultural habits. It is believed that such stoves are responsible for 24% of urban air pollution in India.

The adulteration of fuels in vehicles also contributes to air pollution. In many parts of South Asia, including India, auto-rickshaws and taxis are often fueled with a mixture of cheaper ingredients, such as kerosene and lubricants, to reduce costs. This issue is exacerbated by India's tax system, which imposes high taxes on gasoline and diesel, making them prohibitively expensive for many low-income individuals. In contrast, kerosene, which is subsidized for use in cooking, carries a lower tax rate and is often mixed with higher-priced fuels to cut costs. The resultant pollution from this adulteration adds to the already severe levels of air pollution.

Urban traffic congestion also plays a significant role in worsening air quality. The lack of efficient intra-city highways and traffic management, combined with a high number of vehicles on the road, leads to constant idling in traffic jams, contributing to the pollution levels. Monitoring stations located near major intersections in cities like Delhi often report significantly higher pollution levels than in other areas.

Dust from construction activities, particularly during the dry season, and from natural sources such as nearby deserts, adds to the airborne particulate matter. During the winter months, the air quality in Delhi regularly falls into the "severe" category due to the practice of crop residue burning, which alone contributes to 32% of Delhi's PM2.5 levels. During the “Great Smog of Delhi” in November 2017, the Badarpur Thermal Power Station, a major contributor to pollution, was temporarily shut down. Although it produced only 8% of Delhi's electricity, it was responsible for up to 90% of the city's particulate emissions. It

was permanently shut down in late 2018 due to its significant environmental impact.

Thus,

India’s air pollution problem is multifaceted, with various sectors—industry, agriculture, transportation, and domestic fuel use—playing major roles in contributing to the country’s deteriorating air quality. The health impacts of air pollution are widespread, affecting millions of lives and contributing to rising mortality rates. While there have been efforts to address these issues, such as the Air (Prevention and Control of Pollution) Act of the 1980s, the lack of enforcement and the growing industrial and vehicular emissions have hindered their success. As awareness of the health risks associated with poor air quality grows, there is increasing pressure on the government and industries to implement stricter regulations and invest in cleaner technologies.

Problem Statement

Air pollution is a growing global concern, directly impacting public health and the environment. High levels of pollution, measured through the Air Pollution Index (API), can cause various respiratory and cardiovascular diseases. Current air quality monitoring systems are often expensive, large in size, and typically used by governmental bodies or large companies. These systems are not accessible to individual users or small communities, limiting their ability to monitor and respond to pollution levels in real-time.

This project aims to design a cost-effective, portable air quality monitoring system that provides users with real-time air quality data. By using sensors and Wi-Fi connectivity, the system will measure key pollutants and send the data to a cloud platform, such as ThingSpeak, for further analysis. Additionally, a machine learning (ML) model will be developed using the data collected on ThingSpeak to predict air quality trends and classify the pollution level. This will enable the prediction of future air quality, making it easier for users to anticipate and respond to potential health risks.

Project Scope

The scope of this project includes:

1. Sensor and Microcontroller Integration:
   Utilize the MQ135 air quality sensor and ESP8266 Wi-Fi module to monitor and collect real-time air quality data (gas concentration and particulate matter).

2. Cloud Data Storage and Visualization:

   Send data to ThingSpeak for storage and real-time visualization of air quality parameters, enabling easy access to historical data and trends.

3. Machine Learning Model Development:

   Train a ML model using data collected from ThingSpeak to predict air quality levels and classify them as normal, medium, or dangerous.

Objectives

The objectives of this project are:

1. Design a Low-Cost Air Quality Monitoring System:
   Develop an affordable, portable air quality monitoring system that integrates the MQ135 sensor with the ESP8266 Wi-Fi module.

2. Data Collection and Cloud Integration:
   Collect air quality data and transmit it to ThingSpeak for real-time monitoring, storage, and analysis of air quality trends.

3. Develop a Machine Learning Model:
   Train a ML model using historical air quality data to predict air quality levels and classify them as normal, medium, or dangerous.

4. Real-Time Prediction and Classification:
   Use the trained machine learning model to analyze real-time air quality data and predict potential hazards, providing users with timely alerts.

2) Hardware Components:

1. Arduino Uno

The Arduino Uno is a microcontroller board based on the ATmega328P. It is used to control all the components in the system, including reading data from the MQ135 Air Quality Sensor and controlling the LEDs, Buzzer, and LCD for feedback and display purposes.

Key Features:

- Microcontroller (ATmega328P): A reliable, widely-used microcontroller with 32 KB flash memory, 2 KB SRAM, and 1 KB EEPROM.
- Digital and Analog I/O Pins: Provides 14 digital I/O pins and 6 analog inputs, making it suitable for connecting a variety of sensors and components.
- USB Interface: The board can be easily programmed via the USB interface, making it beginner-friendly.
- Power Supply Options: Can be powered by USB or external power sources like a 9V battery or 5V adapter.
- Open-Source Platform: Being part of the Arduino ecosystem, it has a large community of developers and a wide range of available resources and libraries.

Justification:
The Arduino Uno is selected as the central microcontroller for this system due to its simplicity, flexibility, and compatibility with a wide range of sensors and peripherals. The ATmega328P is a widely-used and well-documented microcontroller, ensuring stability and ease of programming. Its digital and analog pins provide the necessary interfaces for connecting the MQ135 air quality sensor, LEDs, Buzzer, and LCD. The USB interface simplifies the setup and programming process, making it accessible for both beginners and professionals. Furthermore, Arduino's open-source platform ensures a wealth of community-driven support, libraries, and resources that can be leveraged for easy integration and troubleshooting. It is also cost-effective, which is crucial for maintaining the low-budget nature of the project.

2. ESP8266 Wi-Fi Module

The ESP8266 is a low-cost, Wi-Fi-enabled microcontroller used to connect the air quality monitoring system to the internet for real-time data transmission.

Key Features:

- Wi-Fi Connectivity: Provides easy access to the internet, enabling real-time data transmission to cloud platforms like ThingSpeak.
- Low Power Consumption: Ideal for battery-powered applications where energy efficiency is essential.
- GPIO and ADC Pins: Allows interfacing with various sensors and other components.
- Compact Size: Fits well in space-constrained projects.
- Multiple Communication Protocols: Supports UART, SPI, and I2C communication.

Justification:
The ESP8266 is chosen for its Wi-Fi capability, allowing seamless communication with cloud platforms for real-time air quality data monitoring. It offers low power consumption, which ensures the system operates efficiently without frequent charging, making it suitable for portable applications. Its compact size and multiple communication protocols make it versatile and easy to integrate into the system, while the large community support ensures a steady flow of resources and troubleshooting guides.

3. MQ135 Air Quality Sensor

The MQ135 is a gas sensor that detects a range of gases like ammonia, carbon dioxide, and smoke, which directly affect air quality.

Key Features:

- Multiple Gas Detection: Can detect ammonia, CO_2, benzene, and smoke.
- Analog Output: Provides a direct output to read air quality data.
- Calibration: Allows for sensitivity adjustments based on environmental conditions.
- Wide Detection Range: Can detect both low and high concentrations of gases.

Justification:
The MQ135 was chosen because it is specifically designed for air quality monitoring, making it ideal for detecting harmful gases. Its analog output is compatible with both the Arduino Uno and ESP8266, enabling smooth integration into the system. The calibration feature allows the sensor to be adjusted for accurate readings in different environments, while the wide detection range ensures it can capture varying levels of pollution.

4. LEDs (Green and Red)

The LEDs serve as a visual feedback mechanism, signalling whether the air quality is within acceptable limits or hazardous.

Key Features:

- Low Power Consumption: LEDs consume minimal power, ideal for energy-efficient systems.
- Clear Visual Indicators: Green for normal air quality, red for dangerous levels.
- Long Lifespan: LEDs have an extended operational life, reducing the need for replacements.
- Compact: Easily fits into the design without adding bulk.

Justification:
The LEDs were selected for their low power consumption, ensuring that the air quality monitoring system remains efficient, even if it is running continuously. The use of green and red LEDs offers an intuitive and easily understood visual feedback mechanism for users. Long lifespan and compact size make LEDs the perfect choice for this system, ensuring they are both cost-effective and practical.

5. Buzzer

The buzzer provides an auditory alert when the air quality exceeds safe levels.

Key Features:

- Auditory Alert: Emits a loud sound when air quality is dangerous.
- Low Power Consumption: Requires minimal energy.
- Simple Integration: Easily connected to the microcontroller.
- Effective Warning: Ensures users are alerted in noisy environments.

Justification:
The buzzer was selected for its ability to provide an auditory warning when air quality is hazardous, which is crucial in environments where users may not be looking at the system's visual feedback. The low power consumption ensures it does not significantly impact the overall energy use of the system. The simple integration with the microcontroller and its effectiveness in capturing attention makes the buzzer an essential component for user safety.

6. LCD Display (16x2)

The 16x2 LCD display is used to show the air quality data and system status, providing an easy-to-read interface for the user.

Key Features:

- Real-Time Display: Shows air quality data (PPM) and status messages.
- Low Power Consumption: Designed for energy-efficient usage.
- Clear and Readable: 16 characters across 2 lines ensure readable outputs.
- Simple to Interface: Works well with both Arduino Uno and ESP8266.
- Contrast Adjustment: The display's contrast can be adjusted using a potentiometer to ensure clear visibility in various lighting conditions.

Justification:
The 16x2 LCD display is ideal for this project because it offers a clear and readable interface for displaying real-time air quality readings and system status. This is crucial for users to monitor air quality levels without the need for an external device. Its low power consumption ensures that the system remains energy-efficient, while its simple integration with the Arduino Uno makes it easy to set up and use. The contrast adjustment potentiometer allows users to tweak the display visibility for different lighting conditions, ensuring that the readings are always visible. It provides a direct and user-friendly interface that enhances the usability of the air quality monitoring system.

7. Potentiometer for LCD Contrast

A potentiometer is used to adjust the contrast of the LCD display, allowing the user to fine-tune the display for optimal visibility.

Key Features:

- Adjustable Resistance: The potentiometer allows for manual adjustment of the contrast to suit different lighting environments.

- Simple Integration: Can be easily connected to control the LCD's contrast.
- Low Power: It doesn't consume much power, making it ideal for low-energy systems.
- Durability: Potentiometers are durable and reliable, ensuring long-term usability.

Justification:
The potentiometer is chosen for its ability to adjust the contrast of the LCD, ensuring that the information displayed is always clear, regardless of environmental lighting. This is particularly useful in systems that may be used in various indoor and outdoor environments. The low power consumption ensures the system remains efficient. The simple integration with the Arduino Uno and LCD provides flexibility in adjusting the display settings as needed without requiring additional components or complex coding.

8. Resistors

Resistors are used throughout the system to control the current flowing through various components, ensuring proper operation and protecting sensitive parts.

Key Features:
- Current Limiting: Resistors are used to limit the amount of current passing through LEDs, LCD backlight, and other components to prevent damage.
- Voltage Division: Resistors are used in circuits to divide voltage when needed, such as in the LCD contrast control circuit.
- Protective Function: They act as protective elements to ensure that components do not receive more current than they can handle.

Justification:
Resistors are crucial for protecting components and ensuring proper functionality. For LEDs, they are used to limit the current to prevent overloading and damage. In the LCD circuit, resistors are employed to divide voltage, especially when used with the potentiometer for adjusting the contrast. The use of resistors ensures the longevity and reliability of the components in the system by preventing them from receiving excessive current or voltage. Their cost-effectiveness and simplicity make them an essential component for this system, with minimal impact on overall system complexity.

3) Development boards specification

1. NodeMCU ESP8266 Specifications

The NodeMCU ESP8266 is a low-cost, open-source IoT platform based on the ESP8266 Wi-Fi module. It integrates a powerful microcontroller with built-in Wi-Fi capabilities, making it ideal for Internet of Things (IoT) applications.

Specifications

Feature	Details
Microcontroller	ESP8266 (Tensilica L106 32-bit RISC processor)
Operating Voltage	3.3V
Input Voltage	4.5V–10V
Flash Memory	4 MB (or 1 MB in some versions)
SRAM	80 KB
Clock Speed	80 MHz (can be overclocked to 160 MHz)
Digital I/O Pins	11 GPIO pins
PWM Pins	Available on GPIO pins
Analog Input Pins	1 (10-bit resolution, 0–3.3V range)
Wi-Fi Connectivity	802.11 b/g/n (2.4 GHz) with WPA/WPA2 encryption
UART Pins	RX/TX
Communication Interfaces	UART, SPI, I2C
Power Consumption	<215 mA during transmission
USB Connectivity	Micro USB for power and programming
Dimensions	Approximately 58 mm x 31 mm
Weight	8-10 g

Key Features

- Wi-Fi Connectivity: Supports 802.11 b/g/n standards for wireless communication. Allows for seamless connection to the internet or a local Wi-Fi network.
- Built-In Flash Memory: Offers 4 MB of flash memory for program storage, enabling larger programs or additional data.

- General-Purpose I/O (GPIO) Pins: Includes 11 GPIO pins that support PWM, I2C, SPI, and digital I/O functionalities.
- Analog Input Pin: One 10-bit ADC pin for measuring analog signals from sensors like MQ135 or potentiometers.
- Low Power Consumption: Optimized for IoT applications, where low power usage is crucial for battery-operated devices.
- Integrated USB Interface: Micro USB port for powering the board and programming via the Arduino IDE or Lua interpreter.
- Flexible Programming: Supports multiple programming platforms, including Arduino IDE, Lua, and MicroPython.

Why NodeMCU ESP8266 is a Good Choice

- Cost-Effective Solution: Highly affordable for projects requiring wireless connectivity, making it accessible for developers and students.
- Compact Size: Small dimensions make it easy to integrate into portable or space-constrained designs.
- Wi-Fi Capability: Built-in Wi-Fi eliminates the need for an external module, simplifying IoT project designs.
- IoT-Oriented Features: Provides a direct way to send data to cloud platforms like ThingSpeak, making it ideal for air quality monitoring or similar IoT projects.
- Versatility: Supports integration with various sensors and modules, including gas sensors (e.g., MQ135), LEDs, buzzers, and LCDs.

2. Arduino Uno Specifications

The Arduino Uno is one of the most popular development boards, widely used for beginners and advanced users alike. It is based on the ATmega328P microcontroller and offers an easy-to-use hardware and software interface.

Specifications

Feature	Details
Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (Recommended)	7-12V

Input Voltage (Limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (0.5 KB used by bootloader)
SRAM	2 KB
EEPROM	1 KB
Clock Speed	16 MHz
USB Connectivity	USB Type-B for programming and power
Communication Interfaces	UART, SPI, I2C
Dimensions	68.6 mm x 53.4 mm
Weight	Approximately 25 g

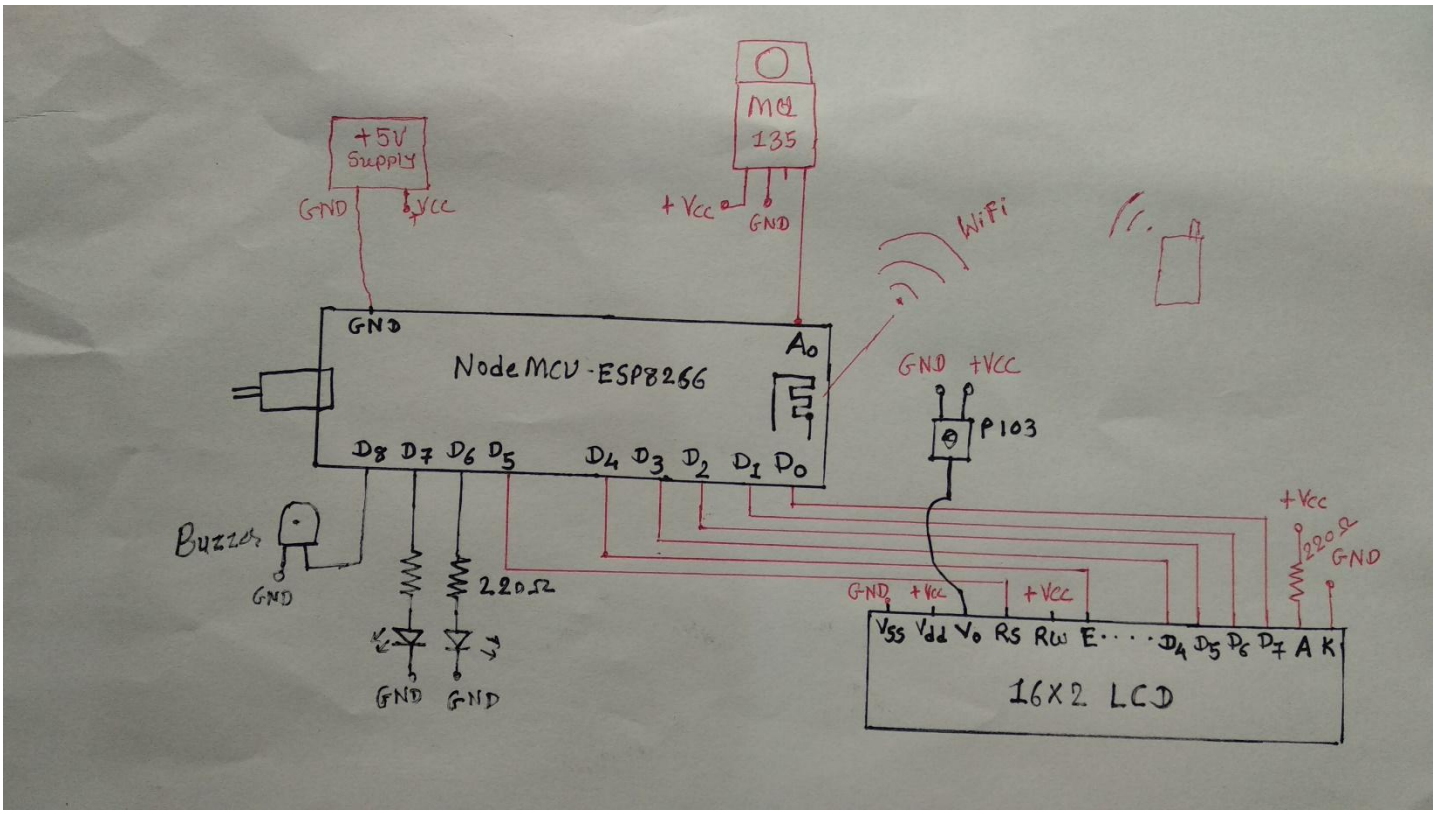
Key Features

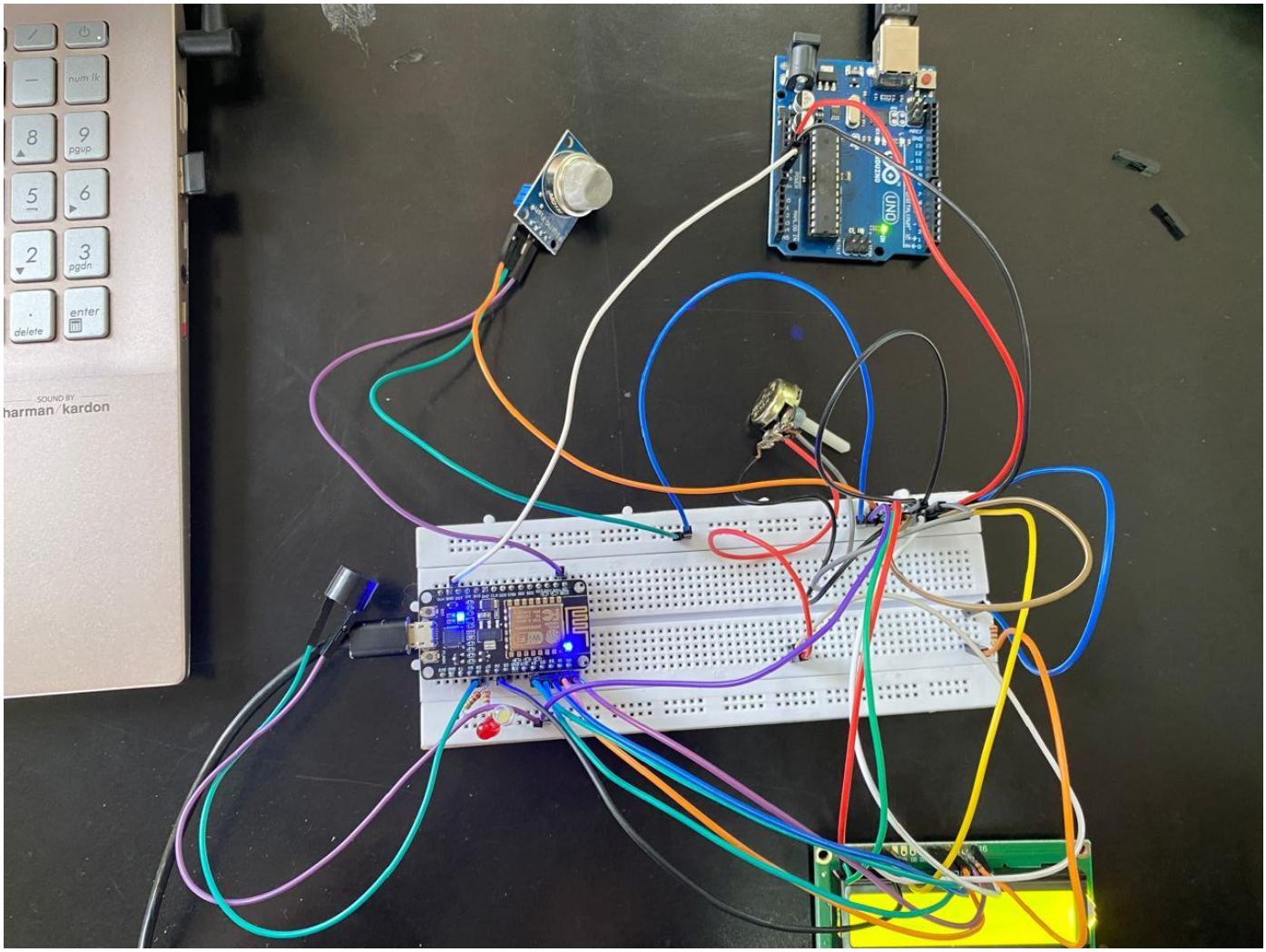
- Microcontroller ATmega328P:
  - Low-power 8-bit microcontroller with 16 MHz clock speed.
  - Ideal for handling basic processing and interfacing with sensors and actuators.
- Digital and Analog I/O Pins:
  - 14 digital pins and 6 analog input pins for interfacing with a wide variety of peripherals.
  - Provides flexibility for PWM outputs (ideal for motor control or LED dimming).
- Onboard Power Management:
  - Operates at 5V but supports input voltages between 7–12V, making it versatile for different power sources.
- Built-in Communication Support:
  - Supports UART, SPI, and I2C communication protocols for seamless connectivity with sensors, displays, and other devices.
- Flash Memory and EEPROM:
  - 32 KB of Flash memory for code storage and 1 KB of EEPROM for storing non-volatile data.
- Ease of Programming:
  - Uses the Arduino IDE, which provides a beginner-friendly platform for writing and uploading code via USB.

Why Arduino Uno is a Good Choice

- Ease of Use: Suitable for beginners, with a simple interface and robust online documentation and tutorials.
- Wide Compatibility: Supports a variety of sensors, modules, and shields (e.g., Wi-Fi, LCD, motor drivers).
- Reliable Performance: Stable and predictable operation, thanks to its 16 MHz clock and efficient ATmega328P microcontroller.
- Versatility: Works with both analog and digital sensors, making it suitable for numerous applications, including IoT, robotics, and home automation.

4) Circuit diagram





Explanation:

1. NodeMCU (ESP8266):

- Acts as the central controller for the system.
- It processes the analog signal from the MQ-135 sensor and transmits data over Wi-Fi to a cloud platform (such as ThingSpeak).
- Pins D0 to D8 are used to control other components.

2. MQ-135 Gas Sensor:

- Connected to the A0 (Analog Input) pin of the NodeMCU.
- Detects air pollutants and sends an analog signal proportional to the concentration of gases (e.g., CO2, NH3, etc.).
- VCC is connected to the +5V supply, and GND is connected to the common ground.

3. 16x2 LCD Display with Potentiometer (P103):

- Displays real-time air quality readings and status messages.
- Connections:
- RS (Register Select): Connected to D5 of NodeMCU.
- E (Enable): Connected to D4.
- D4 to D7 (Data Lines): Connected to D3, D2, D1, D0.
- Potentiometer:
- Used to adjust the contrast of the LCD. One side is connected to VCC (+5V), the other to GND, and the wiper pin to the LCD's Vo pin.

4. Green and Red LEDs:

- Indicate air quality levels.
- Green LED (D6): Lights up for medium air quality.
- Red LED (D7): Lights up for air quality level danger.
- Each LED is connected via a 220Ω resistor to limit current and prevent damage.

5. Buzzer:

- Sounds an alarm when the air quality crosses a dangerous threshold.
- Connected to D8 pin and GND.

Arduino:

Powers the NodeMCU, sensor, LCD, and other components.
Wi-Fi Communication:

The NodeMCU connects to the internet via Wi-Fi, sending sensor data to the ThingSpeak IoT platform.

Working of the Circuit

1. MQ-135 Sensor:

- Continuously measures the air quality and sends an analog voltage to the NodeMCU through pin A0.
- The analog reading is processed in the microcontroller to calculate the air quality level in parts per million (PPM).

2. NodeMCU ESP8266:

- Based on the PPM values:
- Displays air quality on the LCD display.
- Activates the appropriate LED (Green for good quality, Red for bad quality).
- Sounds the buzzer if air quality exceeds dangerous levels.
- Sends the processed air quality data to the ThingSpeak cloud for remote monitoring.

3. LCD with Potentiometer:

- The potentiometer allows users to adjust the contrast of the LCD for better readability.
- Displays the real-time air quality in PPM and status messages such as "Normal," "Medium," or "Danger."

4. ThingSpeak:

- The NodeMCU uploads the air quality data to the ThingSpeak platform, where users can visualize, analyze, and monitor the data from anywhere.

Key Features of the Circuit:

- Real-time air quality monitoring with visual and audible alerts.
- LCD display for local data representation.
- LEDs and buzzer for quick alert mechanisms.
- Wi-Fi-enabled IoT platform integration (ThingSpeak) for remote monitoring.
- Cost-effective and energy-efficient design.

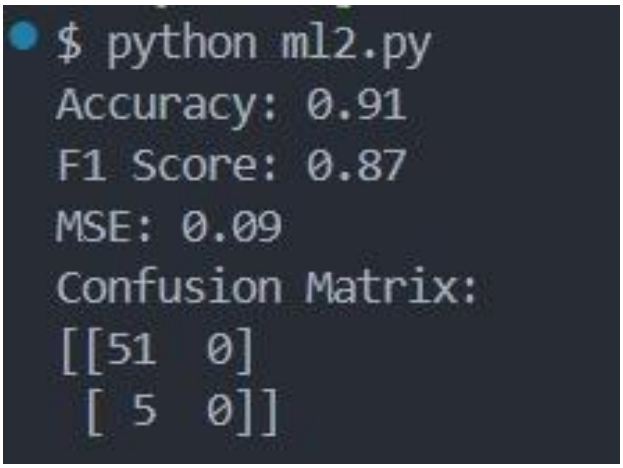
5) Predictive analysis

ML MODEL:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, f1_score, accuracy_score,
mean_squared_error
import numpy as np

# Load Data
df = pd.read_csv('feeds.csv')
df['datetime'] = pd.to_datetime(df['created_at'])
df['hour'] = df['datetime'].dt.hour
df['day'] = df['datetime'].dt.day
df['month'] = df['datetime'].dt.month

# Ensure 'field1' is string and extract numeric values
df['field1'] = df['field1'].astype(str)  # Convert to string
df['ppm'] = pd.to_numeric(df['field1'].str.extract(r'(\d+)')[0],
errors='coerce')

# Remove rows with NaN values after extraction
df.dropna(subset=['ppm'], inplace=True)

# Convert ppm to categorical (you can adjust thresholds as needed)
df['ppm_category'] = pd.cut(df['ppm'], bins=[0, 100, 150, 200, np.inf],
labels=['Good', 'Moderate', 'Unhealthy', 'Very Unhealthy'])

# Split Data
X = df[['hour', 'day', 'month']]
y = df['ppm_category']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Train Model
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# Predict and Evaluate
y_pred = model.predict(X_test)

# Metrics
accuracy = accuracy_score(y_test, y_pred)
```

```
f1 = f1_score(y_test, y_pred, average='weighted')  # Weighted F1 for
imbalanced datasets
mse = mean_squared_error(pd.factorize(y_test)[0], pd.factorize(y_pred)[0])  #
Encode categories numerically for MSE
conf_matrix = confusion_matrix(y_test, y_pred)

# Print Metrics
print(f"Accuracy: {accuracy:.2f}")
print(f"F1 Score: {f1:.2f}")
print(f"MSE: {mse:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
```

OUTPUT :



We machine learning to classify air quality levels based on historical data. It employs a Decision Tree Classifier and evaluates its performance using various metrics.

STEPS:

1. Data Preprocessing:
   - Data Loading: The script loads data from a CSV file (feeds.csv) into a Pandas DataFrame.

- Datetime Parsing: Extracts time-based features like hour, day, and month from the created_at column.
- PPM Conversion: Converts the air quality data (field1) into numeric values representing PPM (Parts Per Million).
- Categorization: Categorizes PPM values into four air quality categories: Good , Moderate , Unhealthy , Very Unhealthy
- Missing Data Handling: Drops rows with invalid or missing PPM values.

2. Feature Selection:
   - Uses hour, day, and month as the independent features (X).
   - Air quality categories (ppm_category) act as the target variable (y).

3. Splitting Data:
   - Divides the dataset into training (80%) and testing (20%) subsets using train_test_split.

4. Model Training:
   - Uses a Decision Tree Classifier from sklearn to train on the features (X_train) and target labels (y_train).

5. Evaluation:
   Predicts the air quality category for the test dataset (y_test) and calculates:
   - Accuracy: Measures overall performance.
   - F1 Score: Balances precision and recall, especially useful for imbalanced datasets.
   - MSE (Mean Squared Error): Quantifies the prediction error.
   - Confusion Matrix: Displays the model's classification performance per category.

Justification for Choices:

1. Decision Tree Classifier:

A Decision Tree Classifier is particularly suitable for this problem due to its characteristics that align well with the nature of air quality classification. Below are detailed reasons why it is a good and appropriate choice:

1. Interpretability and Transparency

- Decision trees provide a visual and logical representation of the decision-making process, making them easy to interpret. For air quality classification, where thresholds play a crucial role, a decision tree can directly map these thresholds as simple decision rules.

2. Suitability for Categorical and Numerical Data
   - Decision Trees work well with both numerical and categorical features, such as air quality levels (categorical target) and temporal features like numerical inputs. Decision Trees are a natural fit without needing excessive preprocessing like feature scaling or normalization.
3. Handling of Non-Linear Relationships:
   - Decision Trees inherently capture non-linear patterns in data through branching. Air quality data often shows non-linear trends.
   - For example: During peak hours, air quality may drastically deteriorate (sharp boundary).Seasonal changes (e.g., pollution spikes in winter due to smog) create non-linear dependencies between the month feature and air quality. The Decision Tree can effectively learn these relationships without requiring explicit feature transformations.
4. Efficiency and Speed
   - Decision Trees are computationally efficient, especially for small-to-medium-sized datasets. Air quality datasets, such as the one derived from the feeds.csv file, are relatively small in size. A Decision Tree is lightweight, fast to train, and doesn't require high computational resources, making it a practical choice for real-time or low-latency applications like IoT-based air quality monitoring.
5. Robustness to Outliers
   - Decision Trees are robust to outliers because splits are based on thresholds rather than numerical calculations like means or standard deviations. Air quality sensors may occasionally report erroneous values (spikes in PPM due to sensor glitches). The Decision Tree can handle such outliers without significant degradation in performance.
6. Aptness for Feature Importance
   - Decision Trees provide a measure of feature importance, identifying which features contribute the most to predictions. For this project, understanding the influence of hour, day, and month on air quality can provide actionable insights:
   - E.g., "Air quality is worse in the afternoon" (hour influence).
   - E.g., "Air pollution peaks in winter" (month influence). These insights can guide environmental policies or sensor placement.
7. Adaptability for Threshold-Based Problems

- Decision Trees are ideal for problems where the solution depends on well-defined thresholds. Air quality classification relies heavily on thresholds defined by health standards (e.g., AQI thresholds for Good, Medium, etc.). A Decision Tree maps these thresholds into its structure, ensuring accurate and logical predictions.

8. Low Risk of Overfitting with Pruning
    - Pruning methods (like setting a maximum tree depth) mitigate the risk of overfitting in Decision Trees.In this project, where training data may be limited, pruning ensures the model generalizes well to unseen data while maintaining simplicity.

2. Features (hour, day, month):

- Time-based factors can significantly impact air quality, e.g., industrial emissions peak during working hours or seasons.
- These features allow the model to detect patterns and periodicity in air quality variations.

3. PPM Categorization:
- Transforming numeric PPM data into categorical air quality levels simplifies interpretation and aligns with real-world classification.
- The air quality index is widely represented using such categories (Good, Medium, etc.).

4. Metrics (Accuracy, F1, MSE, Confusion Matrix):
- A combination of metrics gives a holistic view of the model's performance.
- Accuracy: Measures overall correctness.
- F1 Score: Mitigates imbalances between categories.
- MSE: Indicates how far off predictions are numerically.
- Ensures the model is evaluated fairly across all performance aspects.

Analysis of Results:

Performance Metrics:

- Accuracy (91%): Indicates that the model is highly accurate in predicting the correct air quality category.
- F1 Score (0.87): Reflects strong performance across all categories, with balanced precision and recall.
- MSE (0.09): Low error suggests the model's predictions closely match the actual values.
- Confusion Matrix:
    - True Positives (51): The model correctly predicted the majority of cases.
    - False Negatives (5): A small number of instances were misclassified, possibly at category boundaries.
    - No False Positives for Some Classes: Indicates the model’s strong ability to avoid over-prediction for certain categories.

6) Cloud platform used:

ThingSpeak is an Internet of Things (IoT) platform that enables users to collect, store, visualize, and analyze data from sensors or IoT devices in real-time. It acts as a bridge between devices and the cloud, allowing seamless data transmission, monitoring, and management. ThingSpeak is widely used for applications such as environmental monitoring, energy management, industrial automation, and smart home projects.

Developed by MathWorks, ThingSpeak is tightly integrated with MATLAB, a powerful tool for numerical computation and analytics, which allows users to perform in-depth analysis and develop machine learning models based on the collected data. Its RESTful API support makes it highly compatible with a wide range of devices and programming platforms.

Key Features of ThingSpeak:

- Real-Time Data Logging: Collects data from IoT devices at regular intervals.
- Data Visualization: Offers charts and plots to visualize sensor data on dashboards.
- Cloud Storage: Stores sensor readings securely in the cloud.
- MATLAB Integration: Enables advanced analytics and predictive modelling using MATLAB tools.
- Alerts and Notifications: Configures triggers and alerts based on thresholds.
- APIs for Communication: Supports RESTful APIs for data communication between devices and the platform.
- Multi-Channel Support: Allows multiple data streams to be sent to and managed from a single channel.
- Open-Source Accessibility: Freely available for personal use with options for commercial licenses.

How the Code Uses ThingSpeak:

- Upload Real-Time Air Quality Data: Data from the MQ135 sensor, such as the Air Quality Index (AQI), is uploaded to the ThingSpeak cloud via Wi-Fi using the ESP8266 module.

- Visualization of Data: Once uploaded, the data is displayed graphically on ThingSpeak's dashboard. Users can monitor trends in air quality over time.
- Storage for Further Analysis: ThingSpeak stores the data, which can later be used for advanced analytics or training an ML model.
- Access via API: The platform provides an API for retrieving the stored data for additional processing or visualization.

The following steps occur in the code:

- Wi-Fi Connection: The ESP8266 module connects to the Wi-Fi network using the credentials provided in the code.
- Data Upload: The AQI values are formatted and sent to ThingSpeak's server using its API key and RESTful HTTP protocol.

Justification:

ThingSpeak was chosen for this project due to its unique features and advantages:

- Ease of Use: ThingSpeak’s user-friendly interface makes it simple to set up and integrate with IoT devices, especially with platforms like Arduino and ESP8266.
- Real-Time Monitoring: It provides a straightforward way to monitor real-time air quality data from anywhere in the world, as long as there is internet connectivity.
- Visualization Capabilities: The in-built plotting and dashboard tools allow the user to see trends in air quality without the need for additional programming or tools.
- Data Accessibility: The stored data can be accessed anytime via ThingSpeak’s APIs.
- Free and Open-Source: For non-commercial purposes, ThingSpeak provides free accounts with enough functionality for small-scale projects like this one, keeping the project cost-effective.

7)Security measures taken care

- Wi-Fi Security
  - Encrypted Communication via Wi-Fi:
    - The NodeMCU ESP8266 connects to a secured Wi-Fi network using WPA/WPA2 encryption. This is a robust wireless security protocol, ensuring that unauthorized devices cannot intercept the communication between the ESP8266 and the router.

  - Code Implementation:
    - const char *ssid = "";   // Enter your WiFi Name
    - const char *pass = ""; // Enter your WiFi Password
    - WiFi.begin(ssid, pass);

  - Security Risk Mitigated:
    - Prevents unauthorized access to the network and ensures the air quality data sent to ThingSpeak remains confidential.

- API Key Authentication for Cloud Communication
  - API Key Implementation:
    - The connection to the ThingSpeak server requires a valid API key to authenticate the client and ensure only authorized devices can write data to the designated channel.

  - Code Implementation:
    - String apiKey = "FEXSPSVQVOF61UUR"; // ThingSpeak Write API Key
    - client.print("X-THINGSPEAKAPIKEY: " + apiKey + "\n");

  - Security Risk Mitigated:
    - Unauthorized data entry into your ThingSpeak channel is prevented. Only clients with the correct API key can send data.

- Controlled Communication Protocol
  - HTTP Requests:

        - The data is sent to ThingSpeak using HTTP POST requests. HTTP itself ensures structured communication between the device and the server.

    - Code Implementation:
        - client.print("POST /update HTTP/1.1\n");
        - client.print("Host: api.thingspeak.com\n");
        - client.print("Connection: close\n");

    - Security Risk Mitigated:
        - Minimizes malformed communication and ensures the server can recognize legitimate data packets.

- Device Behavior and Communication Monitoring
    - Serial Monitor Debugging:
        - The code uses the serial monitor to output important status messages (e.g., Wi-Fi connection status, air quality levels, ThingSpeak server response). This allows the developer to monitor the device's functioning in real-time and identify potential issues quickly.
    - Code Implementation:
        - Serial.print("Air Quality: ");
        - Serial.println(ppm);
        - .println("Waiting...");

    - Security Risk Mitigated:
        - Helps in diagnosing miscommunication, firmware bugs, or incorrect sensor readings.

- Logical Error Prevention
    - Range-Based Threshold Logic:
        - The code carefully segregates air quality levels into normal, medium, and danger zones. This prevents logical errors in alert triggering and ensures the accuracy of the warning system.

    - Code Implementation:
        - if (ppm <= 130)
        - {
        - digitalWrite(gled, LOW);

        - digitalWrite(rled, LOW);
        - digitalWrite(buz, LOW);
        - }

    - Risk Mitigated:
        - Avoids false alarms or missed alerts due to incorrect conditional checks.

8) Experiment components and results:

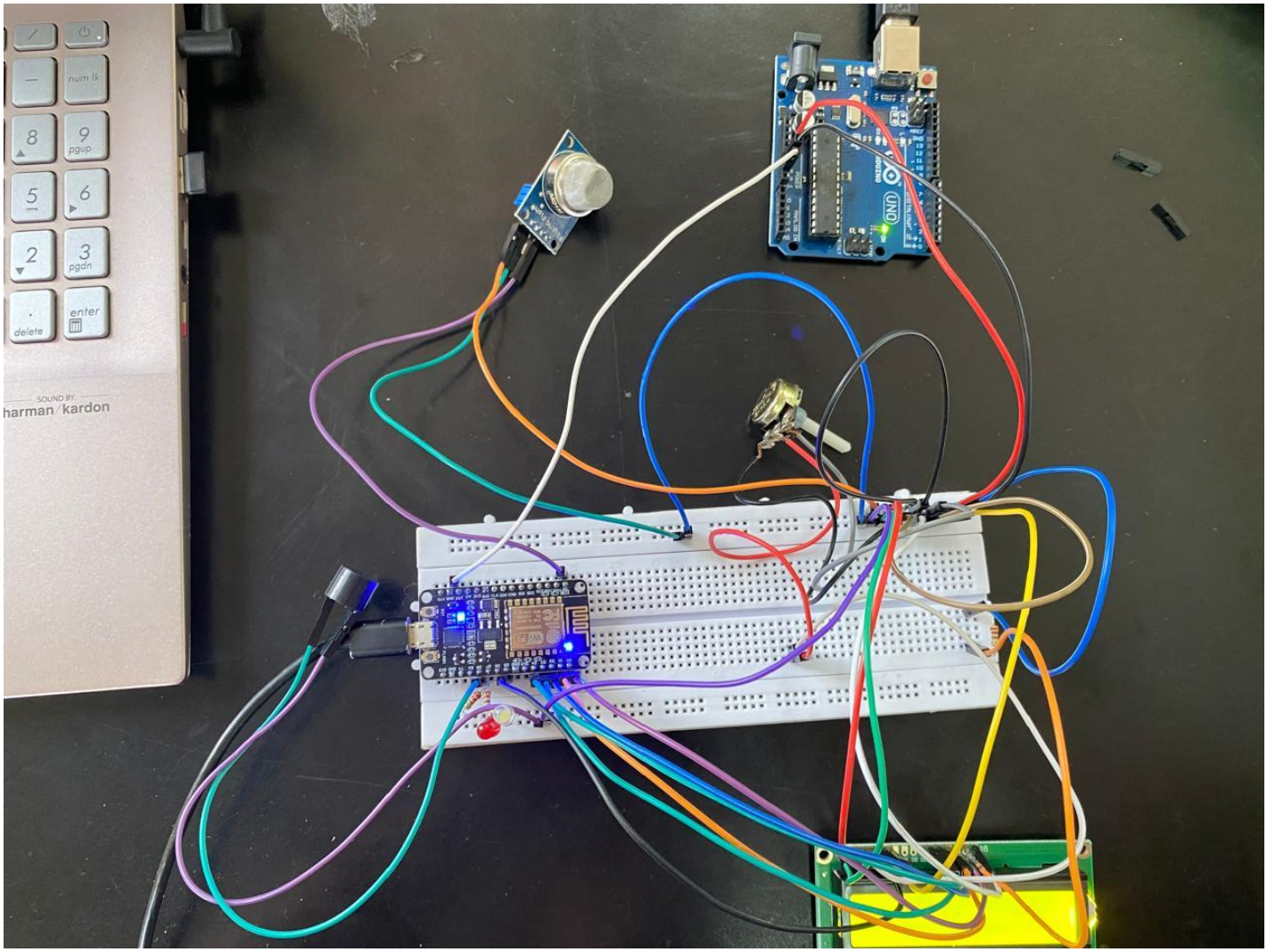
Arduino Code:

```
#include <ESP8266WiFi.h>
#include <LiquidCrystal.h>  // include the library code:

// initialize the library by associating any needed LCD interface pin
// with the arduino pin number it is connected to
const int rs = D5, en = D4, d4 = D3, d5 = D2, d6 = D1, d7 = D0;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

const int aqsensor = A0;  //output of mq135 connected to A0 pin of ESP-8266

int gled = D6;  //green led connected to pin 6
int rled = D7;  //red led connected to pin 7
int buz = D8;   //buzzer led connected to pin 8

String apiKey = "FEXSPSVQVOF61UUR";     //  Enter your Write API key here
const char *ssid =  "POCO";     // Enter your WiFi Name
const char *pass =  "6a6ccd88dfc7"; // Enter your WiFi Password
const char* server = "api.thingspeak.com";
WiFiClient client;

void setup() {
  // put your setup code here, to run once:
  pinMode (gled,OUTPUT);     // gled is connected as output from ESP-8266
  pinMode (aqsensor,INPUT); // MQ135 is connected as INPUT to ESP-8266
  pinMode (rled,OUTPUT);
  pinMode (buz,OUTPUT);

  Serial.begin (115200);      //begin serial communication with baud rate of 115200

  lcd.clear();              // clear lcd
  lcd.begin (16,2);         // consider 16,2 lcd

  Serial.println("Connecting to ");
  lcd.print("Connecting...");
  Serial.println(ssid);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED)
  {
```

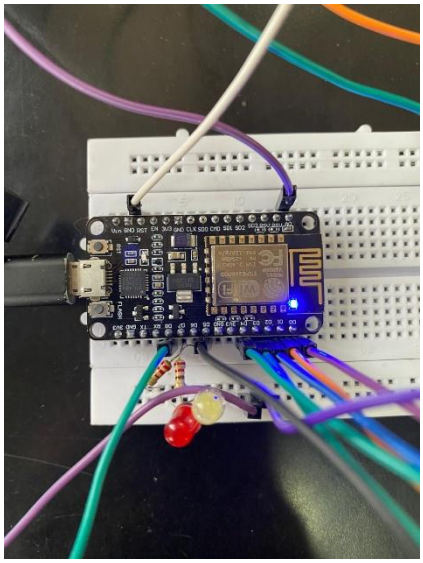
```
39)    delay(1000);
40)    Serial.print(".");
41)    lcd.print(".");
42)  }
43)  Serial.println("");
44)  Serial.println("WiFi connected");
45)  Serial.print("IP Address: ");
46)  delay(5000);
47)  Serial.println(WiFi.localIP());
48)  delay(5000);
49)}
50)
51)void loop() {
52)  // put your main code here, to run repeatedly:
53)
54)    int ppm = analogRead(aqsensor); //read MQ135 analog outputs at A0
   and store it in ppm
55)
56)  Serial.print("Air Quality: ");  //print message in serail monitor
57)  Serial.println(ppm);            //print value of ppm in serial monitor
58)
59)  lcd.setCursor(0,0);             // set cursor of lcd to 1st row and
   1st column
60)  lcd.print("Air Quality: ");      // print message on lcd
61)  lcd.print(ppm);                 // print value of MQ135
62)  delay(1000);
63)
64)  if (client.connect(server,80))
65)                       {
66)                              String postStr = apiKey;
67)                              postStr +="&field1=";
68)                              postStr += String(ppm);
69)                              postStr += "\r\n\r\n";
70)
71)                              client.print("POST /update HTTP/1.1\n");
72)                              client.print("Host: api.thingspeak.com\n");
73)                              client.print("Connection: close\n");
74)                              client.print("X-THINGSPEAKAPIKEY:
   "+apiKey+"\n");
75)                              client.print("Content-Type: application/x-
   www-form-urlencoded\n");
76)                              client.print("Content-Length: ");
77)                              client.print(postStr.length());
78)                              client.print("\n\n");
79)                              client.print(postStr);
80)                              Serial.print("Air Quality: ");
81)                              Serial.print(ppm);
82)                              Serial.println(" PPM.Send to Thingspeak.");
```

```
                        }
  if(ppm <= 130)
  {
    digitalWrite(gled,LOW);   //jump here if ppm is not greater than
  threshold and turn off gled
    digitalWrite(rled,LOW);
    digitalWrite(buz,LOW);   //Turn off Buzzer
    lcd.setCursor(1,1);
    lcd.print ("AQ Level Normal");
    Serial.println("AQ Level Normal");
  }
  else if (ppm > 130 && ppm < 250)
  {
    digitalWrite(gled,HIGH);   //jump here if ppm is not greater than
  threshold and turn off gled
    digitalWrite(rled,LOW);
    digitalWrite(buz,LOW);   //Turn off Buzzer
    lcd.setCursor(1,1);
    lcd.print ("AQ Level Medium");
          Serial.println("AQ Level Medium");
       }
       else
       {
         lcd.setCursor(1,1);         //jump here if ppm is greater than
  threshold
         lcd.print("AQ Level Danger!");
         Serial.println("AQ Level Danger!");
         digitalWrite(gled,LOW);
         digitalWrite(rled,HIGH);
         digitalWrite(buz,HIGH);     //Turn ON Buzzer
       }

               client.stop();
               Serial.println("Waiting...");
               delay(1000);

     }
```

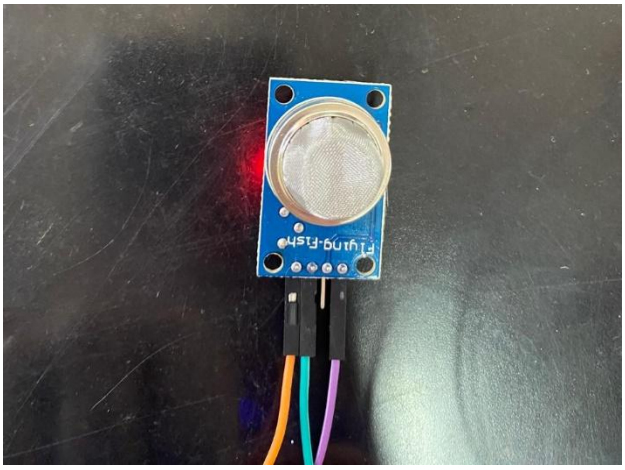
CIRCUIT DIAGRAM:



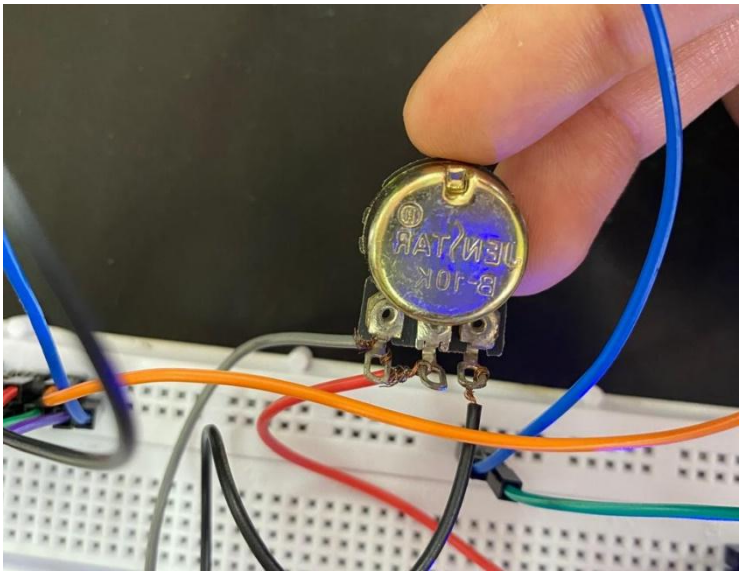
NODEMCU ESP8266



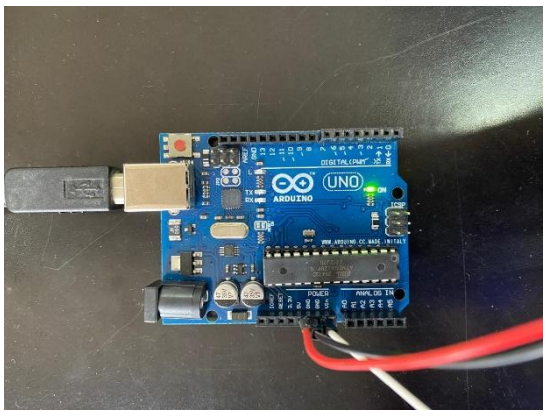
MQ - 135 SENSOR :



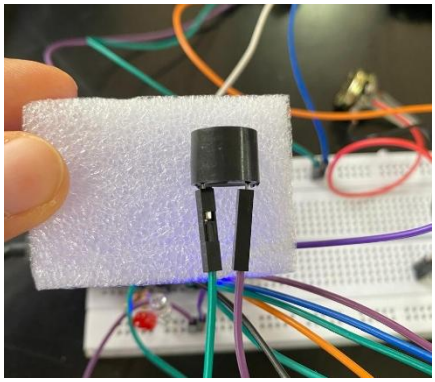
POTENTIOMETER :



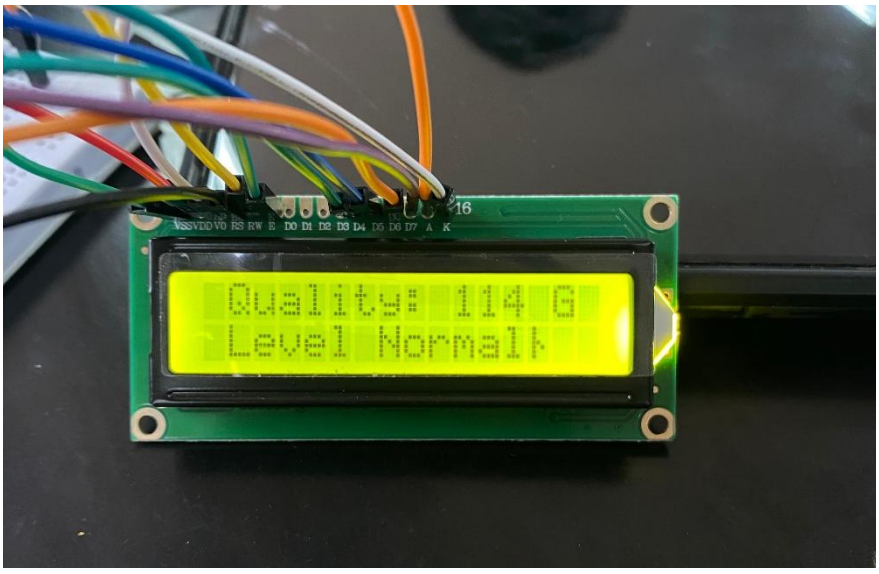
ARDUINO UNO :



BUZZER :



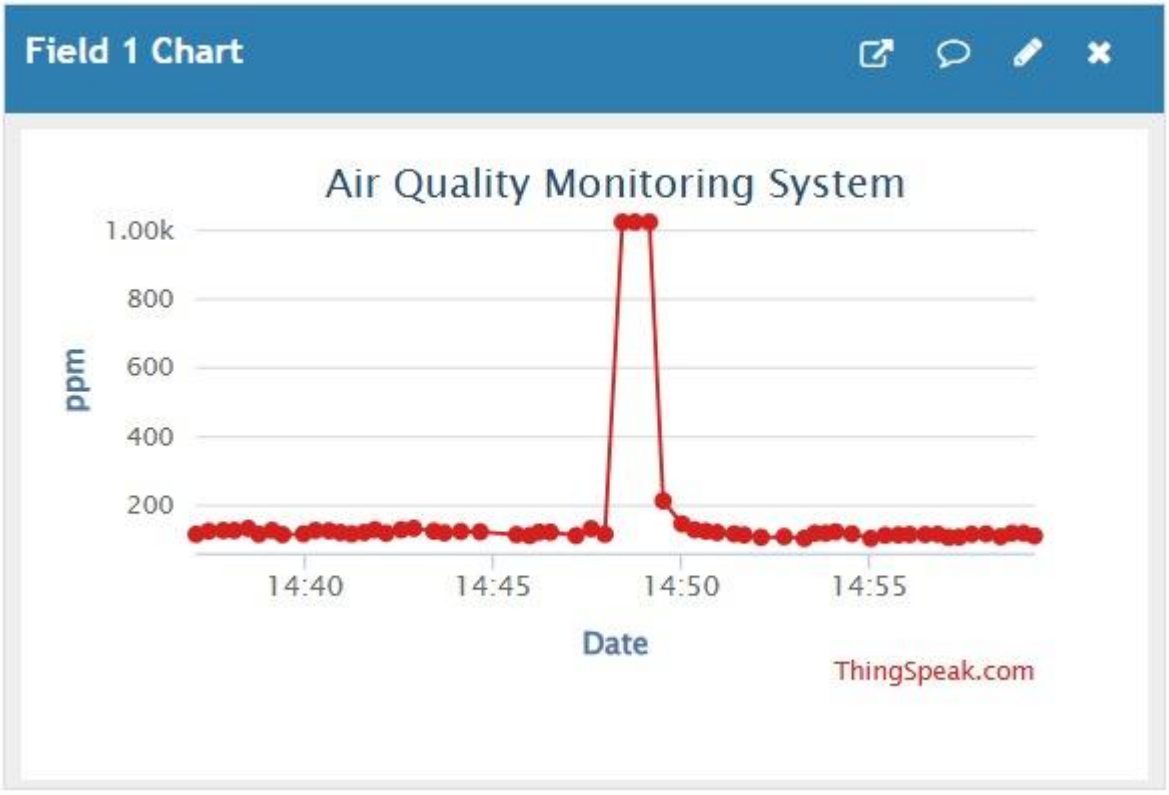
AIR QUALITY NORMAL :

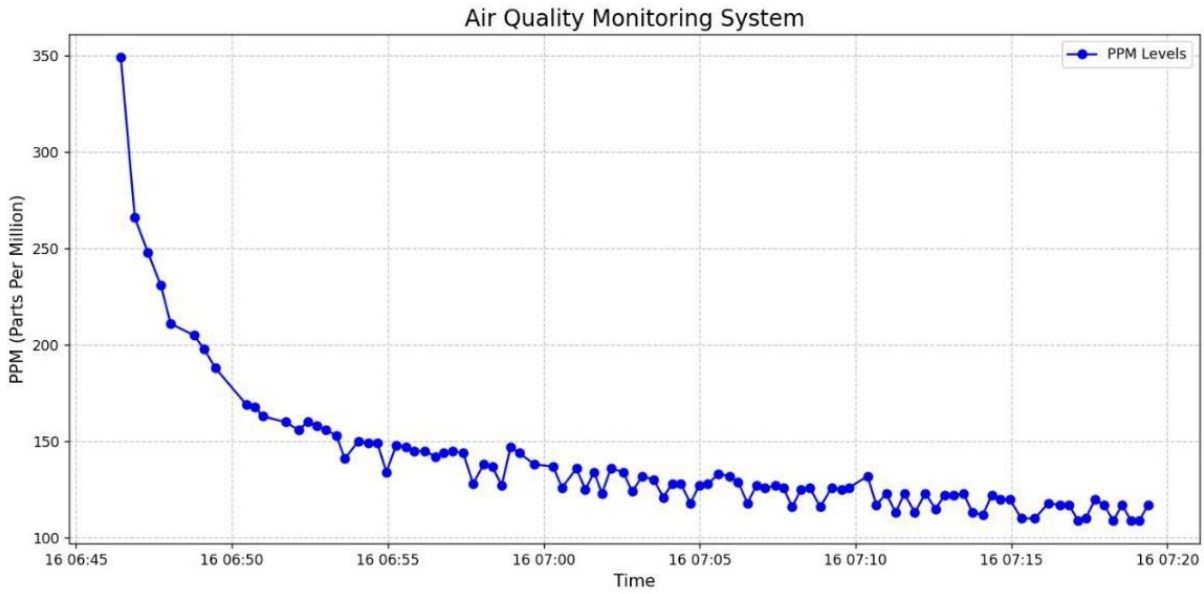


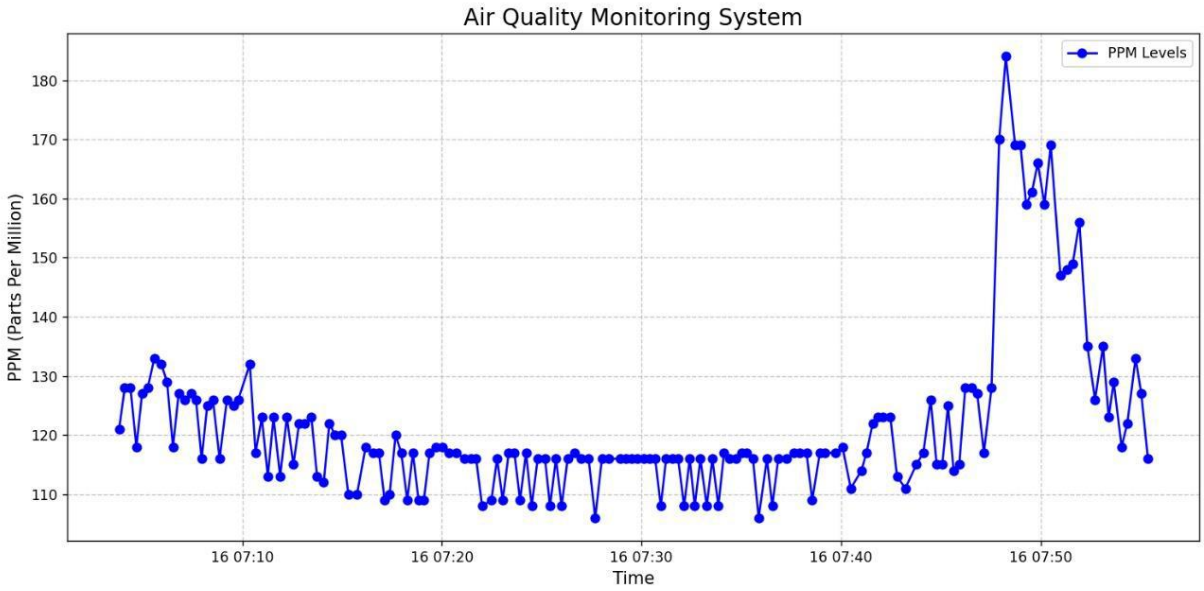
AIR QUALITY MEDIUM :

a

SAMPLE GRAPH FOR READINGS FOR A PARTICULAR TIME INTERVAL FROM CLOUD






Air Quality Monitoring System
PPM Levels
PPM (Parts Per Million)
180
170
160
150
140
130
120
110
16 07:10
16 07:20
16 07:30
16 07:40
16 07:50
Time