COLLEGE CODE: 5113
COLLEGE NAME: Kingston Engineering College
DOMAIN: Internet of Things
PROJECT TITLE: Noise Pollution Monitoring System

PROJECT MEMBERS:	NAN MUDHALVAN Id
D Navin(Leader)	511321104062
D Namdev	511321104060
B Purushoth	511321104076
H Sai Jaswanth	511321104083
T Vamsi Krishna	511321104108

Monitoring noise pollution with Arduino and sound devices can be a valuable project, helping to collect data for environmental purposes or personal use. Below is a development plan for such a project:

1. Define Project Objectives:

- Clearly state the goals of your noise pollution monitoring system. Consider what you want to achieve and the specific data you wish to collect.

2. Select Hardware and Components:

- Choose the right sound sensor or microphone for your project. Some popular options include electret condenser microphones or sound level sensors.
- You will also need an Arduino board (e.g., Arduino Uno, Arduino Nano) to process data.
- Depending on the project's requirements, consider other components like an SD card module for data storage, an LCD screen for real-time display, and a power source.

3. Software Development:

- Write the Arduino code to interface with the sound sensor/microphone and collect noise data.

- You may need to use libraries for signal processing or noise analysis.

- Implement data logging functions to store noise level data.

4. Data Processing:

- Consider including data preprocessing steps if required. This can include filtering, averaging, or other data manipulation to get more accurate readings.

5. Calibration:

- Calibrate your sound sensor to real-world noise levels to ensure accurate measurements.

6. Data Storage:

- Implement data storage options, such as writing data to an SD card, uploading to a cloud server, or displaying on a local screen.

7. Real-Time Display:

- If you want real-time monitoring, connect an LCD or LED display to show noise levels.

8. Data Analysis:

- Develop software for data analysis to identify patterns and trends in noise pollution data over time.

9. Connectivity:

- Add Wi-Fi or Bluetooth modules to transmit data to a remote server for centralized monitoring and analysis.

10. Power Supply:

- Ensure that you have a reliable power source for your Arduino and sensors. Battery or solar power options can be considered for remote and outdoor deployments.

11. Housing and Weatherproofing:

- If the system will be deployed outdoors, consider weatherproofing and enclosure options to protect the components.

12. Testing and Calibration:

- Test your system in various environments to ensure accurate and reliable noise measurements.

- Recalibrate as needed to maintain data accuracy.

13. Data Visualization:

- If desired, create a data visualization dashboard, which could include graphs, maps, or a web-based interface to view the noise pollution data.

14. Maintenance and Monitoring:

- Develop a plan for maintaining and monitoring the system, including periodic calibration and checking for hardware issues.

15. Data Usage:

- Determine how you will use the collected data. Will you share it with the public, local authorities, or for personal analysis?

16. Legal and Ethical Considerations:

- Be aware of any legal and ethical considerations related to collecting noise data, especially in public spaces.

Python program

```
import sounddevice as sd
import numpy as np
import csv
import time

# Define the parameters
duration = 3600  # Recording duration in seconds (1 hour)
sample_rate = 44100  # Sampling rate in Hz
channels = 1  # Mono audio

# Create an empty list to store the sound level data
sound_levels = []

# Function to calculate the decibel level
def calculate_decibel_level(audio_data):
    rms = np.sqrt(np.mean(audio_data**2))
    decibel_level = 20 * np.log10(rms)
    return decibel_level

# Callback function for recording audio
def audio_callback(indata, frames, time, status):
    if status:
        print("Error:", status)

    decibel_level = calculate_decibel_level(indata)
    sound_levels.append(decibel_level)

# Start recording
with sd.InputStream(callback=audio_callback, channels=channels,
                    samplerate=sample_rate):
    print(f"Recording for {duration} seconds...")
    sd.sleep(duration * 1000)  # Sleep for the recording duration

# Save the recorded data to a CSV file
timestamp = time.strftime("%Y%m%d%H%M%S")
filename = f"sound_levels_{timestamp}.csv"

with open(filename, 'w', newline='') as csvfile:
    csv_writer = csv.writer(csvfile)
    csv_writer.writerow(['Time', 'Decibel Level (dB)'])
    for i, level in enumerate(sound_levels):
        csv_writer.writerow([i / sample_rate, level])

print(f"Data saved to {filename}")
```