ℕ𝕆𝕀𝕊𝔼 ℙ𝕆𝕃𝕃𝕌𝕋𝕀𝕆ℕ 𝕄𝕆ℕ𝕀𝕋𝕆ℝ𝕀ℕ𝔾

❤️PROJECT OBJECTIVES❤️

Real-time Noise Pollution Monitoring:

Implement a system capable of continuously measuring ambient noise levels in real-time.

Ensure accuracy, precision, and reliability of noise measurements to provide reliable data for analysis.

Public Awareness:

Develop an interface (e.g., mobile app or web dashboard) for users to access and understand noise pollution data.

Provide visualizations and user-friendly information to raise public awareness about noise levels in specific areas.

Noise Regulation Compliance:

Integrate features to monitor and compare noise levels against local or national regulations and standards.

Generate alerts or notifications when noise levels exceed prescribed limits, facilitating compliance with noise regulations.

Improved Quality of Life:

Aim to reduce noise pollution's impact on the community's quality of life.

Provide actionable insights or recommendations based on noise data to mitigate sources of excessive noise

❤️ＩＯＴ ＤＥＳＩＧＮ❤️

Hardware Components:

Microcontroller: You'll use a microcontroller (e.g., Arduino, Raspberry Pi, etc.) as the core of the system. This microcontroller will handle data acquisition, processing, and communication.

Microphone: A high-quality microphone (e.g., electret condenser microphone) is essential for accurately capturing ambient noise levels.

Amplifier (optional): Depending on the sensitivity of the microphone and the noise levels you want to measure, you might include an amplifier circuit to boost the audio signal.

Analog-to-Digital Converter (ADC): The microcontroller's built-in or an external ADC will convert the analog audio signal from the microphone into digital data for processing.

Display (optional): You can add a display (e.g., OLED, LCD) to show real-time noise levels or other relevant information.

WiFi Module (optional): If you want to enable wireless communication for real-time monitoring, you'll integrate a WiFi module (e.g., ESP8266 or ESP32).

Software and Functionality:

Microcontroller Firmware:

You'll write firmware to control the microcontroller, manage sensor inputs, and handle data processing.

The firmware will configure the ADC to sample audio data from the microphone.

Audio Processing:

Apply algorithms to process the raw audio data. For example, you might calculate decibel levels (in dB) or perform frequency analysis.

Real-time Monitoring :

If you've integrated a WiFi module, set up communication protocols (e.g., HTTP or MQTT) to send data to a server or cloud platform for real-time monitoring

❤️ ＮＯＩＳＥ ＰＯＬＬＵＴＩＯＮ ＩＮＦＯＲＭＡＩＯＮ❤️

❤️ ＰＬＡＴＦＯＲＭ❤️

Set Up Django Web Application:

Step 1: Install Django: Use pip to install Django, a high-level Python web framework.

Step 2: Create a New Django Project: Use the django-admin command to create a new project. This will serve as the backend for your web application.

Step 3: Design Models: Define Django models to represent data structures related to noise measurements. This may include tables for measurements, locations, users, etc.

Step 4: Create Views and Templates: Define views to handle HTTP requests and render templates to generate HTML pages. These templates will serve as the frontend of your web application.

Step 5: Implement RESTful APIs: Create APIs using Django REST framework to allow communication between the Django backend and your React Native app.

Step 6: Set Up a Database: Configure a database (e.g., SQLite, PostgreSQL) to store measurement data.

Step 7: Implement User Authentication (if needed): Set up user authentication for secure access to the Django site.

Step 8: Test the Django Site: Run the development server and test the Django site to ensure it's functioning as expected

Set Up React Native App:

Step 1: Install React Native: Set up a new React Native project using the command-line tool react-native-cli.

Step 2: Design Components: Create React Native components to handle the user interface of your mobile app. This may include screens for viewing noise data, user authentication, settings, etc.

Step 3: Set Up Navigation: Implement navigation between different screens using a navigation library like react-navigation.

Step 4: Connect to Django Backend: Use Axios or another HTTP client library to make API requests to your Django backend, allowing the React Native app to fetch and display data.

Step 5: Implement User Authentication (if needed): Set up user authentication within the React Native app to allow users to log in and access their accounts.

Step 6: Test the React Native App: Run the app in an emulator or on a physical device to ensure it functions correctly and communicates with the Django backend.

Integrate Django Backend with IoT Device:

Step 1: Configure API Endpoints: In your Django project, create API endpoints to receive data from the IoT device. This could be in the form of HTTP POST requests.

Step 2: Process and Store Data: Implement logic to process incoming data, validate it, and store it in the database.

❤️ ＩＮＴＥＧＲＡＴＩＯＮ ＡＰＰＲＯＡＣＨ （ＨＴＴＰ） ❤️

Audio Capture:

Use the microphone to capture audio. Depending on your microcontroller and hardware setup, you'll need to employ appropriate libraries or drivers to interface with the microphone.

Audio Encoding:

The captured audio will need to be encoded into a format suitable for transmission over the network. Common audio formats include WAV, MP3, or even raw PCM data.

Establish an HTTP Connection:

Use the microcontroller's WiFi chip to establish an HTTP connection with the target application. This will likely involve creating an HTTP POST request to send the audio data.

Stream Audio Data:

Continuously send chunks of audio data to the application in real-time. This could be achieved by breaking the audio into smaller packets and sending them over the established HTTP connection.

Receive and Process Audio Data:

On the application side, you'll need a server capable of receiving HTTP requests, extracting the audio data, and processing it. This could involve decoding the audio, storing it, or processing it in real-time.