**IoT Noise Pollution Monitoring System Deployment**

**PHASE Ⅳ**

**Aim:**

The aim of this project is to develop and deploy an Internet of Things (IoT) solution to monitor noise pollution in public areas and make real-time noise level data accessible to the public through a platform or mobile app. This project aims to raise awareness of noise pollution issues and provide valuable data for noise control and urban planning.

**Project Team:**

|  |  |
| --- | --- |
| **Name** | **Mail-Id** |
| 1.Praveen Kumar.M | [praveenmurugan420@gmail.com](mailto:pushparajraje52141@gmail.com) |
| 2.Manobala.B | raviravi84104495[@gmail.com](mailto:srihariharan220@gmail.com) |
| 3.Siva.J | devilkingsiva15[@gmail.com](mailto:nishanthsaru.oto@gmail.com) |
| 4.Sanjay.R | [sanjeevsanjay947@gmail.com](mailto:navinnavin1909@gmail.com) |

**Project Overview:**

Noise pollution is a growing concern in urban environments, negatively impacting the health and well-being of residents. To address this issue, we propose the development of a noise pollution monitoring system using IoT sensors. The system will consist of sound sensors, Arduino boards, an ESP8266 module for data transmission, and a cloud platform for data storage and analysis.

**Key Components:**

* **Arduino Uno Module:** The Arduino Uno serves as the microcontroller for data collection and processing.
* **Sound Sensor Module (MO26):** This module is responsible for detecting sound and noise levels in the local environment.
* **ESP8266 Serial ESP 01 Wi-Fi Wireless Transceiver Module:** This Wi-Fi module facilitates wireless data transmission to the cloud.
* **Thingspeak Cloud:** We have chosen Thingspeak as the cloud platform for data storage and analysis.

**To Building a comprehensive noise pollution information platform and a mobile app for IoT-based noise pollution monitoring involves creating a system that receives, processes, and presents noise data collected from sensors. Here are the steps to continue building this project:**

* Define Objectives and Requirements:
* Determine the specific goals and objectives of the platform.
* Identify the target audience and their needs.
* Define the features and functionalities required, such as real-time data display, data visualization, maps, and user accounts.
* Choose Web Development Technologies:
* Select the web development technologies that best suit your project. Common technologies include HTML, CSS, JavaScript, and web frameworks like React, Angular, or Vue.js.
* Design User Interface (UI):
* Create a user-friendly and intuitive interface that allows users to interact with noise level data effectively.
* Design the layout, color scheme, and navigation elements.
* Data Sources and Integration:
* Identify the data sources for real-time noise level data. This may involve using IoT sensors, external APIs, or user-generated data.
* Set up mechanisms for real-time data integration, such as WebSocket or Server-Sent Events (SSE).
* Database Setup:
* Choose an appropriate database system to store historical noise level data. Options include MySQL, PostgreSQL, or NoSQL databases depending on the data structure and volume.
* Backend Development:
* Develop a server application to handle incoming data from IoT sensors. You can use frameworks like Flask, Django, or Node.js.Implement APIs for receiving and storing noise data, as well as retrieving historical data for analysis.Implement data validation and security measures to prevent unauthorized access and data tampering.
* Data Analysis:
* Develop algorithms to analyze and process the noise data. Calculate statistics, identify trends, and detect anomalies.Implement reporting mechanisms to generate insights and visualizations for different stakeholders.
* User Authentication:
* Implement user authentication and authorization mechanisms to control access to the platform.
* Define roles and permissions for different users, such as administrators, researchers, and the public.
* Alerting System:
* Set up an alerting system to notify relevant parties (e.g., local authorities) when noise levels exceed predefined thresholds. Data Storage:
* Continuously store and maintain historical noise data in the database for trend analysis and compliance reporting.
* Web Dashboard:
* Create a web-based dashboard for real-time monitoring and analysis of noise data. The dashboard should be user-friendly and visually appealing.Provide options for customizing charts, reports, and filtering data.
* User Authentication and Authorization:
* Implement user authentication to control access to the platform.
* Define user roles and permissions to restrict or grant access to specific features and data.
* Use technologies like OAuth or JWT for secure user authentication.
* Development and Testing:
* Develop the platform based on the requirements and design. Use the chosen web technologies and frameworks.
* Rigorously test the platform for functionality, performance, and responsiveness.
* Ensure cross-browser and cross-device compatibility.
* Real-Time Data Display:
* Implement the real-time data display feature to provide live noise level updates to users.
* Utilize JavaScript libraries for dynamic data visualizations, such as D3.js or Chart.js.
* User Feedback and Maintenance:
* Launch the platform and gather user feedback to monitor its performance.
* Continuously collect and analyze user input to make necessary improvements.
* Regularly maintain the platform by addressing issues and applying updates and bug fixes.

**To create a web platform that displays real-time noise level data, you can use web development technologies such as HTML, CSS, and JavaScript for the front-end, and a back-end framework (e.g., Node.js with Express) to handle data processing and serve the web application. Additionally, you can use a WebSocket connection for real-time updates. Here's a step-by-step guide:**

* **Set Up Your Development Environment:**
* Install Node.js and npm (Node Package Manager) on your server.
* Choose a code editor or integrated development environment (IDE) for web development.
* **HTML (index.html):**

Create the HTML structurefor your platform: html code

<!DOCTYPE html>

<html>

<head>

<title>Noise Pollution Platform</title>

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

</head>

`<body>

<header>

<h1>Noise Pollution Information Platform</h1>

</header>

<main>

<div id="map">

<!-- Map visualization goes here -->

</div>

`<div id="data">

<!-- Real-time data display goes here -->

</div>

</main>

<footer>

<p>&copy; 2023 Noise Pollution Platform</p>

</footer>

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

</body>

</html>

* **JavaScript (script.js):**

Use JavaScript to establish a WebSocket connection and update the real-time noise

data:

// Simulate real-Ɵme data updates (replace with actual data source)

funcƟon updateRealTimeData() {

const noiseDataElement = document.getElementById('noise-data');

setInterval(() => {

const randomNoiseLevel = Math.floor(Math.random() \* 101);

noiseDataElement.innerText = `Noise Level: ${randomNoiseLevel} dB`;

}, 5000); // Update every 5 seconds (adjust as needed)

}

// IniƟalize the real-Ɵme data display

funcƟon iniƟalizeRealTimeDataDisplay() {

updateRealTimeData();

}

// Wait for the document to be fully loaded before execuƟng JavaScript

document.addEventListener('DOMContentLoaded', () => {

iniƟalizeRealTimeDataDisplay();

});

* **Back-End (server.js):**

Create a Node.js server using Express and WebSocket to handle real-time data updates: javascript code

const axios = require('axios');

// Replace with your ThingSpeak Channel API Key and Field IDs const apiKey = 'YOUR\_API\_KEY';

const field1 = 1; // Replace with your Field ID const field2 = 2; // Replace with your Field ID

// ThingSpeak POST URL

const thingSpeakUrl = `https://api.thingspeak.com/update.json`;

// Data to be sent to ThingSpeak const data = {

api\_key: apiKey,

field1: 25, // Replace with your data field2: 50, // Replace with your data

};

// Function to send data to ThingSpeak

const sendDataToThingSpeak = async () => { try {

const response = await axios.post(thingSpeakUrl, data); if (response.status === 200) {

console.log('Data sent to ThingSpeak successfully.');

} else {

console.error('Failed to send data to ThingSpeak.');

}

} catch (error) {

console.error('Error sending data to ThingSpeak:', error.message);

}

};

// Call the function to send data to ThingSpeak sendDataToThingSpeak();

**Mobile App Development (iOS and Android):**

* Select Development Tools:
* Choose a cross-platform framework like React Native for efficient development. React Native allows you to build for both iOS and Android with a single codebase.
* UI Design:
* Follow platform-specific design guidelines for iOS (UIKit) and Android (Material Design) to create a user-friendly experience.
* Connect to the Platform:
* Implement API connections between the mobile app and the platform using libraries like Axios for making HTTP requests to your server.
* Real-Time Updates:
* Integrate real-time data updates using WebSocket technology in your mobile app. Libraries like React Native WebSocket make it easy to establish real-time connections.
* Display real-time noise data using charts and graphs.
* Provide access to historical noise data with customizable time frames.
* Include push notifications to alert users when noise levels exceed predefined limits.
* Allow users to customize notification settings.
* User Authentication:
* Use Firebase Authentication or a similar service for user authentication, allowing users to access their data securely.
* Enable users to export and share noise data and reports from the app.
* Utilize GPS and location services to provide location-specific noise data.
* Integration:
* Ensure seamless integration between the information platform and the mobile app to share data and provide a unified user experience.
* Development and Testing:
* Develop the mobile app using React Native, ensuring that it fetches and displays real-time noise level data. Test on both iOS and Android devices for performance and responsiveness.
* App Store Deployment:
* Prepare your app for deployment by following the guidelines for the Apple App Store and Google Play Store. Ensure that your app meets the submission requirements for each platform.

**Designing mobile apps for iOS and Android platforms that provide users with access to real-time noise level updates requires careful planning and a user-friendly interface. Below, I'll outline the design considerations for creating these mobile apps:**

**Mobile App(React Native – JavaScript):**

// App.js

import React, { useState, useEﬀect } from 'react'; import { View, Text, StyleSheet } from 'react-native';

const App = () => {

const [noiseLevel, setNoiseLevel] = useState(0);

// Simulate real-time updates (replace with actual data source) useEﬀect(() => {

const interval = setInterval(() => {

const randomNoise = Math.ﬂoor(Math.random() \* 101); setNoiseLevel(randomNoise);

}, 5000);

return () => clearInterval(interval);

}, []);

return (

<View style={styles.container}>

<Text style={styles.title}>Noise Level</Text>

<Text style={styles.noiseValue}>{noiseLevel} dB</Text>

</View>

);

};

const styles = StyleSheet.create({ container: {

ﬂex: 1,

justifyContent: 'center', alignItems: 'center',

},

title: { fontSize: 20,

},

noiseValue: { fontSize: 40, fontWeight: 'bold',

},

});

export default App;

**THANK YOU**