KGiSL INSTITUTE OF TECHNOLOGY

(Approved By AICTE, New Delhi, Affiliate to Anna University

Recognized by UGC, Accredited by NBA(IT)

265, KGISL Campus, Thudiyalur Road, Saravanampatti, Coimbatore-641035**.)**

DEPARTMENT OF

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

NAAN MUDHALVAN - INTERNET OF THINGS

NOISE POLLUTION MONITORING

NAME: Lavanya R

REG NO: 711721243052

NM ID: au711721243052

TEAM MENTOR: Mr. Mohankumar M

TEAM EVALUATOR: Ms. Akilandeeshwari M

Phase 4: Development Part 2

Building a Noise pollution Monitoring system using IoT sensors and Raspberry Pi integration

PROBLEM STATEMENT

The IoT-Based Noise Pollution Monitoring System project aims to design, develop and deploy an innovative, cost-effective solution for monitoring and mitigating noise pollution in urban and residential environments. Noise pollution, often underestimated in its impact on public health and quality of life, is a growing concern in densely populated areas. This project seeks to address this issue by creating a comprehensive noise monitoring infrastructure that leverages the capabilities of the Internet of Things (IoT) technology.

Procedure:

Step 1: Define App Requirements:

- Objective: Start by clearly defining the purpose and objectives of your app. In this case, it's to provide real-time noise pollution monitoring in urban and residential environments.
- Feature List: Create a list of features that the app should have, such as real-time noise level display, historical data analysis, user authentication, and notifications.
- Platform Selection: Decide whether you want to develop the app for iOS, Android, or both. This choice will influence the technology stack you use.
- UI/UX Design: Create wireframes and mockups for the app's user interface. Design how users will interact with the monitoring data.

Step 2: Choose a Development Approach:

- Native Development: Opt for native app development if you have separate development teams for iOS and Android. You'll use platform-specific languages and tools like Swift, Objective-C for iOS, and Java or Kotlin for Android.
- Cross-Platform Development: Choose a cross-platform framework like Flutter (Dart), React Native (JavaScript), or Xamarin (C#) to build a single codebase that runs on both iOS and Android.

- Web-Based App: If you prefer a web-based solution, consider developing a progressive web app (PWA) that users can access through their mobile web browsers.

Step 3: Set Up the Development Environment:

- Install Development Tools: Install the required development tools, including Android Studio for Android, Xcode for iOS, or the IDE that corresponds to your chosen cross-platform framework.
- SDKs and Libraries: Download the necessary SDKs and libraries for the chosen platform or framework.

Step 4: App Development:

1. User Authentication:

- Implement user registration and login functionality using secure authentication methods.
- Integrate third-party authentication options like Google or Facebook login.

2. UI Design:

- Create the app's user interface, considering the display of real-time noise level data.
- Use appropriate design guidelines for the platform (Material Design for Android, Human Interface Guidelines for iOS).

3. Integration with IoT Devices:

- Develop the code to communicate with the IoT devices, like the ESP32.
- Use RESTful APIs or WebSockets to retrieve data from the IoT devices.

4. Real-time Data Display:

- Continuously update the app's user interface with real-time noise level and distance data.
- Use charting libraries or components for graphical representation of the data.

5. Data Logging:

- Implement data storage to record historical noise level and distance data.
- Store data in a database (e.g., Firebase, AWS, or a custom server) for later analysis.

Step 5: Testing and Debugging:

- Testing: Thoroughly test the app on different devices, screen sizes, and operating system versions.
- Debugging: Identify and resolve any issues or bugs during testing. Ensure the app functions as expected.

Step 6: Deployment

1. App Store Submission (for Native Apps):

- For native apps, prepare the app for submission to the Apple App Store and Google Play Store.
- Follow the guidelines and requirements for each platform.

2. Web Deployment (for Web-Based Apps):

- Host the web-based app on a web server.
- Ensure it's accessible through web browsers on mobile devices.

.

Step 7: User Documentation:

- Create user guides or documentation that explain how to use the app effectively.
- Provide clear instructions to help users understand and benefit from its features.

Step 8: Maintenance and Updates:

- Continuously monitor the app for issues and gather user feedback.
- Provide regular updates and improvements based on user needs and technological advancements.

Step 9: User Education:

- Promote the app to your target audience through marketing efforts.
- Educate users about the app's features and how it can improve their quality of life.

Step 10: Data Analysis:

- Develop data analysis tools to provide insights and trends based on the monitoring data collected by the app.
- This analysis can help users understand noise pollution patterns and take necessary actions.

APPLICATION DEVELOPMENT PROCEDURE:

Step 1: Setup Environment and Dependencies:

Before you begin, make sure you have the necessary environment and dependencies in place:

- Install Python (for the Flask part).
- Install Flask using `pip install Flask`.
- Have a Java development environment set up (for the Java Spring annotations part).
- For the JavaScript part, you don't need any special setup as it runs in web browsers.

Step 2: Code Organization:

You appear to have code snippets from different technologies mixed together. The Python code is for the Flask web server, the HTML is for the web page, and the JavaScript code is for the frontend (web page) functionality. The Java part seems to be a placeholder for a Spring Boot controller.

Organize your project like this:

```
project_folder/
    ├── app.py (Python code for Flask)
    ├── templates/
    │   ├── index.html (HTML template)
    ├── static/
    │   ├── script.js (JavaScript code)
    ├── src/
    │   ├── YourJavaController.java (Java Spring controller)
```

Step 3: Implement the Backend (Flask):

In your Python code (`app.py`), you have implemented the backend using Flask:

- Flask is set up and configured as a web application.
- A placeholder dictionary (`noise_data`) is used to represent noise monitoring data.
- The `/` route renders the HTML template (index.html) and provides the `noise_data` as context.
- The `/update` route receives data from a form and updates `noise_data`.

Step 4: Create HTML Templates (HTML):

In the `templates` folder, you have an HTML template (`index.html`):

- This template is for the web page that displays the monitoring data.
- It contains placeholders for location and noise level.
- The JavaScript functions in this template fetch data from the server and update the placeholders in real-time.

Step 5: Add JavaScript for Real-Time Updates (JavaScript):

You have JavaScript code inside your HTML template (`index.html`) in a `<script>` section:

- JavaScript functions are defined to fetch data from the server and update the web page in real-time.
- An interval-based approach is used to refresh the data every 5 seconds.
- Event listeners are added to submit form data via AJAX when the user updates the data.

Step 6: Java Spring Controller (Java):

You seem to have a Java Spring controller (`DataController.java`) included in the code, but it's disconnected from the Flask and HTML/JavaScript parts. Ensure your Spring Boot application properly configures this controller to provide data when the `/data` endpoint is accessed.

Step 7: Run Your Application:

- Start your Flask application by running `python app.py`.
- Ensure your Java Spring Boot application is configured correctly and running.
- Open a web browser and access your web application (usually at `http://localhost:5000` for Flask, and a different port for Spring Boot).

Step 8: Testing and Debugging:

- Test your application locally to ensure that data is displayed and updated correctly.
- Debug any issues in both the Flask backend and JavaScript frontend.

SOURCE CODE:

PYTHON:

```
from flask import Flask, render_template, request


app = Flask(__name__)


# Placeholder for monitoring data - in a real app, you would interface with IoT devices
noise_data = {
    "level": 0,
    "location": "Living Room"
}


@app.route('/')
def index():
    return render_template('index.html', data=noise_data)


@app.route('/update', methods=['POST'])
def update_data():
```

```
    # In a real app, you would update noise_data with data from IoT devices
    noise_data['level'] = request.form['noise_level']
    noise_data['location'] = request.form['location']
    return 'Data updated'

if __name__ == '__main__':
    app.run(debug=True)
```

HTML:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Noise Monitor</title>
</head>
<body>
    <h1>Noise Monitor</h1>
    <p>Location: <span id="location">Unknown</span></p>
    <p>Noise Level: <span id="noiseLevel">0</span> dB</p>
    <form id="updateForm">
        <input type="number" name="noiseLevel" placeholder="Enter noise level">
        <input type="text" name="location" placeholder="Enter location">
        <button type="submit">Update Data</button>
    </form>

    <script>
        // Function to update data in real-time
        function updateData() {
            fetch('/data')
                .then(response => response.json())
                .then(data => {
```

```
                document.getElementById('location').textContent = data.location;
                document.getElementById('noiseLevel').textContent = data.level + "
dB";
            });
        }

        // Periodically update data every 5 seconds
        setInterval(updateData, 5000);

        // Submit form data via AJAX
        const updateForm = document.getElementById('updateForm');
        updateForm.addEventListener('submit', function (event) {
            event.preventDefault();
            const formData = new FormData(updateForm);
            fetch('/update', {
                method: 'POST',
                body: formData
            });
        });
    </script>
</body>
</html>
```

CSS:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Noise Monitor</title>
</head>
<body>
```

```
    <h1>Noise Monitor</h1>
    <p>Location: <span id="location">Unknown</span></p>
    <p>Noise Level: <span id="noiseLevel">0</span> dB</p>
    <form id="updateForm">
        <input type="number" name="noiseLevel" placeholder="Enter noise level">
        <input type="text" name="location" placeholder="Enter location">
        <button type="submit">Update Data</button>
    </form>

    <script>
        // Function to update data in real-time
        function updateData() {
            fetch('/data')
                .then(response => response.json())
                .then(data => {
                    document.getElementById('location').textContent = data.location;
                    document.getElementById('noiseLevel').textContent = data.level + "
dB";
                });
        }

        // Periodically update data every 5 seconds
        setInterval(updateData, 5000);

        // Submit form data via AJAX
        const updateForm = document.getElementById('updateForm');
        updateForm.addEventListener('submit', function (event) {
            event.preventDefault();
            const formData = new FormData(updateForm);
            fetch('/update', {
                method: 'POST',
```

```
            body: formData
        });
    });
  </script>
</body>
</html>
```

JAVASCRIPT:

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class DataController {
  @GetMapping("/data")
  public NoiseData getData() {
    return noiseData;
  }
}
```

CONCLUSION:

In summary, developing a noise monitoring web app involves:

Planning: Clearly define project goals and create a roadmap.

Backend Setup: Build a secure backend for data storage and management.

Data Sources: Connect to sensors or APIs for real-time noise data.

User Interface: Design an intuitive UI for data display and visualization.

Deployment: Host the app and configure server settings.

Testing: Thoroughly test for functionality and security.

Documentation: Create user guides and comply with regulations.

Maintenance: Continuously monitor and update the app.

Expertise: Seek professional advice when needed.

Developing a noise monitoring app is a complex process, but it can lead to valuable insights for addressing noise pollution.