Air And Noise Pollution Monitoring Dashboard

Submitted in partial fulfillment of the requirements

for the degree of

Bachelor of Engineering

by

Roll No.	Name
55	Deviprasad Shetty
58	Rupam Singh
61	Ameya Tamanekar
63	Rajatkumar Yadav

Supervisor:

Asst. Prof. Tayyabali Sayyad

UNIVERSITY OF MUMBAI

Air And Noise Pollution Monitoring Dashboard

Submitted in partial fulfillment of the requirements

of the degree of

by

Roll No.	Name
55	Deviprasad Shetty
58	Rupam Singh
61	Ameya Tamanekar
63	Rajatkumar Yadav

Supervisor:

Asst. Prof. Tayyabali Sayyad

Department of Information Technology

Don Bosco Institute of Technology

An Autonomous Institute Affiliated to University of Mumbai

2025-2026

DON BOSCO INSTITUTE OF TECHNOLOGY

Vidyavihar Station Road, Mumbai - 400070

Department of Information Technology

CERTIFICATE

This is to certify that the project entitled **"Air And Noise Pollution Monitoring Dashboard"** is a bonafide work of

Deviprasad Shetty	**55**
Rupam Singh	**58**
Ameya Tamanekar	**61**
Rajatkumar Yadav	**63**

submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of **Undergraduate** in **Bachelor of Information Technology**

Date: / /

(Prof. Tayyabali Sayyad)
Supervisor

(Prof. Sunantha G.)
HOD, IT Department

(Dr. Sudhakar Mande)
Principal

DON BOSCO INSTITUTE OF TECHNOLOGY

Vidyavihar Station Road, Mumbai - 400070

Department of Information Technology

Project Report Approval

This project report entitled **"Air And Noise Pollution Monitoring Dashboard "** by **Rupam Singh , Deviprasad shetty , Ameya Tamanekar, RajatKumar Yadav** is approved for the degree of **Bachelor of Engineering in Information Technology**

(Examiner's Name and Signature)

1. ——————————

2. ——————————

(Supervisor's Name and Signature)

1. ——————————

(Chairman)

1. ——————————

Date:

Place:

DON BOSCO INSTITUTE OF TECHNOLOGY

Vidyavihar Station Road, Mumbai - 400070

Department of Information Technology

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea / data / fact / source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(————————————-)
(Signature)

(————————————-)
(Name of Student and Roll No.)

Date:

Abstract

An abstract is a brief summary of the most important points in a scientific paper/report. Abstracts enable professionals to stay current with the huge volume of scientific literature. Students have misconceptions about the nature of abstracts that may be described as the ”table of contents” or ”introduction” syndromes. There are several ways to tell if you’ve written an abstract or not.

An abstract is a brief synopsis or summary of the most important points that the author makes in the paper/report. It is a highly condensed version of the paper/report itself. After reading the abstract, the reader knows the main points that the authors have to make. The reader can then evaluate the significance of the paper and then decide whether or not she or he wishes to read the full paper/report. If one elects to read the full paper/report, further detail is given about each of the significant topics, but no new topics of importance are introduced. If one decides not to read the paper, that decision is based on a knowledge of the paper’s content.Although the abstract appears first in a paper/report, it is generally the last part written. Only after the paper has been completed can the authors decide what should be in the abstract and what parts are supporting detail.

Keywords: Air Quality Index (AQI), Noise Pollution, Spring Boot, REST API, Geospatial Mapping, Role-Based Access Control,

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

Currently, environmental data is often scattered across different platforms, generalized for entire cities rather than specific localities like railway stations, or completely inaccessible to the general public. There is a lack of a unified, user-friendly platform that visualizes both air quality and noise levels specifically for transit hubs, while also providing administrators with the tools to manage this data effectively.

The aim of this project, EcoGauge, is to develop a full-stack web application that serves as a centralized environmental dashboard. It intends to bridge the gap between complex environmental data and the daily commuter by providing real-time visualizations, interactive maps, and health-based rankings for railway stations.)

1.2 Scope of the Project

The scope of EcoGauge defines the boundaries of the system and the specific functionalities developed during the project lifecycle.

Geographical Scope: The system is designed to monitor stations across the three major railway lines of Mumbai: Western, Central, and Harbour lines.

Functional Modules:

- **User Module:** Provides access to live dashboards, interactive AQI/Noise maps using Leaflet.js, station rankings, and a feedback submission system.
- **Admin Module:** A secured dashboard allowing for the Create, Read, Update, and Delete (CRUD) of station data and the review of user feedback.
- **Security:** Implementation of robust authentication using Local (Email/-Password) and OAuth2 (Google) login mechanisms.
- **Data Parameters:** The system tracks specific metrics including AQI values (derived from PM2.5, PM10, NO2, O3) and Noise Levels (in decibels).

The project follows these development phases:

- Requirements Gathering Phase
- Requirements Analysis Phase
- System Design Phase

1.3 Current Scenario

In the current environmental monitoring landscape, citizens and administrators face a fragmented ecosystem of information regarding pollution levels in metropolitan transit hubs.

- **Generalization of Data:** Popular platforms like Google Weather, AccuWeather, or SAFAR-India typically provide a single Air Quality Index (AQI) reading for the entire city of Mumbai. This ignores the hyper-local variations that exist between highly congested transit hubs (e.g., Dadar, Kurla) and quieter suburban stations.
- **Absence of Noise Data:** While air quality data is somewhat accessible via general weather apps, real-time noise pollution metrics for public spaces are virtually non-existent for the general public. Commuters have no way to gauge decibel levels before entering a station.
- **Manual Administrative Processes:** Environmental officers often rely on legacy systems, disjointed databases, or manual spreadsheets to track pollution data. This lack of a centralized interface delays the dissemination of critical health warnings to the public and makes historical data analysis difficult.

1.4 Need for the Proposed System

The development of **EcoGauge** is driven by the urgent need to bridge the gap between complex environmental data and the daily commuter. The proposed system addresses the critical deficiencies in the current scenario through the following objectives:

- **Granular Monitoring:** Unlike existing solutions, this system provides station-specific data across the Western, Central, and Harbour lines. This allows commuters to know the specific conditions of their departure and arrival points.
- **Holistic Health Awareness:** By integrating both AQI and the Noise Pollution Index (NPI) into a single web dashboard, the system provides a complete picture of environmental health. This encourages commuters to take necessary precautions, such as wearing masks in high PM2.5 zones or using hearing protection in high-decibel areas.
- **Administrative Efficiency:** The system replaces manual tracking with a secured Admin Dashboard. This allows for the efficient creation, reading, updating, and deletion (CRUD) of station data, ensuring that the information displayed to the public is accurate and up-to-date.
- **Public Engagement:** The inclusion of a feedback mechanism creates a channel for users to report on-ground conditions, fostering a responsive and community-driven monitoring ecosystem.

1.5 Summary of the results and task completed

The development of the **EcoGauge** platform has been successfully executed, meeting the primary objectives defined during the requirements phase. The project has resulted in a fully functional web application that serves as a centralized dashboard for monitoring Air Quality Index (AQI) and Noise Pollution.

All planned modules, including the User Interface, Admin Dashboard, and Security Layer, have been implemented and tested. Crucially, the system successfully integrates with external environmental APIs to fetch real-time data for 30 monitoring stations across Mumbai's Western, Central, and Harbour railway lines, ensuring the dashboard displays the latest available metrics.

Key Results Obtained:

Table 1.1: Project Module Completion Summary

Module	Status	Key Features Achieved
Authentication	Completed	Implemented dual-layer security with Local Login (BCrypt encryption) and Google OAuth2 integration. Added Password Reset flow via Email.
Backend API	Completed	Developed 11 RESTful endpoints handling User data, Station metrics, and Feedback submission.
Data Management	Completed	**Successful integration of external API data ingestion.** The system fetches live environmental metrics and persists them into the H2 In-Memory Database for historical analysis.
User Interface	Completed	Deployment of responsive Dashboards, Interactive Maps (AQI/Noise), and Ranking pages using Leaflet.js and Chart.js.
Admin Controls	Completed	Successful implementation of CRUD operations allowing Admins to manage the fetched station data and review user feedback.

- **Real-time Data Integration:** The application automates the retrieval of complex environmental data (PM2.5, Noise dB) from external sources and renders it into intuitive gauges and maps.
- **Role-Based Access Control (RBAC):** The system correctly distinguishes between 'User' and 'Admin' roles, restricting sensitive operations (like modifying fetched data) to authorized personnel only.
- **Data Scalability:** The architecture successfully handles the ingestion and storage of high-frequency data points for 30 stations without performance degradation.

Table 1.2: Sample Real-time Environmental Data Fetched from API

Line	Station	Latitude	Longitude	AQI	Category	Dominant	PM2.5	PM10	NO_2	O_3
Western	Andheri	19.1190	72.8465	123	Poor	PM2.5	53.0	97.0	41.0	34.0
Western	Bandra	19.0560	72.8410	117	Poor	PM2.5	51.0	94.0	39.0	36.0
Harbour	Belapur CBD	19.0180	73.0300	90	Moderate	PM2.5	41.0	79.0	30.0	45.0
Western	Bhayandar	19.3000	72.8500	96	Moderate	PM2.5	44.0	83.0	32.0	43.0
Western	Borivali	19.2290	72.8570	102	Poor	PM2.5	46.0	86.0	34.0	42.0
Harbour	Chembur	19.0560	72.9000	123	Poor	PM2.5	53.0	97.0	41.0	34.0
Western	Churchgate	18.9283	72.8297	105	Poor	PM2.5	48.0	88.0	35.0	40.0
Central	CSMT	18.9402	72.8355	114	Poor	PM10	50.0	93.0	38.0	38.0

Chapter 2

Review of Literature

2.1 Summary of the investigation in the published papers

An extensive survey of existing literature and technical papers was conducted to understand the current state of environmental monitoring systems, web-based visualization, and role-based security. The following key studies were investigated to define the scope of EcoGauge:

- **Paper 1: "Real-time Air Quality Monitoring System using IoT and Cloud Computing" (IEEE, 2021).**
  Summary: This paper proposes a hardware-centric solution for collecting PM2.5 and CO levels using microcontroller sensors sending data to a cloud dashboard.
  Investigation Relevance: While the hardware implementation was robust, the investigation revealed a lack of a user-friendly frontend. The data was often displayed on LCD screens on-site rather than a centralized web dashboard accessible to the public. This identified the gap that EcoGauge aims to fill by focusing on the web-visualization layer.

- **Paper 2: "Web-based GIS for Noise Mapping in Urban Areas" (ACM, 2020).**
  Summary: This study focuses on visualizing noise pollution using Geographic Information Systems (GIS) to create heatmaps of urban noise.

Investigation Relevance: The study demonstrated the effectiveness of color-coded maps in raising public awareness. However, it relied on static maps generated once a month. EcoGauge improves upon this by using Leaflet.js to render interactive maps based on live data fetched from APIs.

- **Paper 3: "Role-Based Access Control (RBAC) in Enterprise Web Applications" (Springer, 2022).**
  Summary: A comparative study of security protocols for managing administrative vs. user privileges in Java applications.
  Investigation Relevance: The investigation highlighted the necessity of separating "Read-Only" access (for citizens) from "CRUD" access (for officials). This directly influenced the decision to implement Spring Security with distinct `ROLE_USER` and `ROLE_ADMIN` authorities.

2.2 Comparison between the tools / methods / algorithms

To ensure the optimal performance and scalability of EcoGauge, a comparative analysis was performed between the chosen technology stack (Spring Boot) and other popular web development frameworks.

Table 2.1: Comparison of Technology Stacks for Environmental Dashboards

Parameter	Proposed System (EcoGauge)	Traditional/Alternative Stacks (e.g., MERN)
Backend Architecture	**Spring Boot (Java):** Offers robust dependency injection, strict type-checking, and built-in multi-threading, which is ideal for processing synchronous API calls from multiple stations.	**Node.js (JavaScript):** Event-driven architecture is fast but can become unmanageable with complex business logic and lacks the strict structure of Java needed for enterprise security.
Data Storage	**H2 Database (SQL):** Relational, ACID-compliant structure ensures that Station Data (Latitude, Longitude, AQI) maintains strict integrity.	**MongoDB (NoSQL):** Flexible schema is good for unstructured data but risks data inconsistency when managing relational entities like Users and their Feedback.
Security Implementation	**Spring Security:** Provides a comprehensive, configurable security chain for OAuth2 and RBAC out-of-the-box.	**Passport.js / JWT:** Requires significant manual configuration and boilerplate code to achieve the same level of granular security.
Map Visualization	**Leaflet.js:** Open-source, lightweight, and capable of rendering custom markers without API costs.	**Google Maps API:** Powerful but incurs high costs for high-traffic public dashboards.

2.3 Algorithm(s) with example

EcoGauge employs specific algorithms for data processing (Backend) and geospatial searching (Frontend).

Algorithm 1: Environmental Category Classification

This algorithm runs on the backend server immediately after fetching data from the external API. It classifies raw AQI numerical values into human-readable categories (Good, Moderate, Poor, etc.) before storing them in the H2 database.

Pseudo Code (Java Logic):

```
FUNCTION CategorizeAQI(value):
    IF value <= 50 RETURN "Good"
    ELSE IF value <= 100 RETURN "Moderate"
    ELSE IF value <= 150 RETURN "Poor"
    ELSE IF value <= 200 RETURN "Unhealthy"
```

```
    ELSE IF value <= 300 RETURN "Severe"
    ELSE RETURN "Hazardous"
END FUNCTION
```

Analysis:

- **Time Complexity:** $O(1)$ - The logic consists of a fixed set of conditional checks, executing in constant time regardless of the input size.
- **Space Complexity:** $O(1)$ - Requires negligible auxiliary space.

Algorithm 2: Geospatial Proximity Search (Haversine)

This algorithm runs on the client-side (Frontend) when a user clicks "Detect My Location". It calculates the distance between the user's GPS coordinates and all stored stations to find the nearest monitoring point.

Pseudo Code (Haversine Formula):

```
FUNCTION FindNearestStation(userLat, userLon, stationList):
    minDistance = INFINITY
    nearestStation = NULL

    FOR EACH station IN stationList:
        dLat = RADIANS(station.lat - userLat)
        dLon = RADIANS(station.lon - userLon)

        a = SIN(dLat/2)^2 + COS(userLat) * COS(station.lat) * SIN(dLon/2
        c = 2 * ATAN2(SQRT(a), SQRT(1-a))
        distance = EARTH_RADIUS * c

        IF distance < minDistance:
            minDistance = distance
            nearestStation = station

    RETURN nearestStation
END FUNCTION
```

Analysis:

- **Time Complexity:** $O(N)$ - Where N is the number of stations. The system must iterate through every station in the database to guarantee the closest match.
- **Space Complexity:** $O(1)$ - Only stores variables for the minimum distance found so far.

Algorithm 2: Geospatial Proximity Search (Haversine)

This algorithm runs on the client-side (Frontend) when a user clicks "Detect My Location". It calculates the distance between the user's GPS coordinates and all stored stations to find the nearest monitoring point.

Pseudo Code (Haversine Formula):

```
FUNCTION FindNearestStation(userLat, userLon, stationList):
    minDistance = INFINITY
    nearestStation = NULL

    FOR EACH station IN stationList:
        dLat = RADIANS(station.lat - userLat)
        dLon = RADIANS(station.lon - userLon)

        a = SIN(dLat/2)^2 + COS(userLat) * COS(station.lat) * SIN(dLon/2
        c = 2 * ATAN2(SQRT(a), SQRT(1-a))
        distance = EARTH_RADIUS * c

        IF distance < minDistance:
            minDistance = distance
            nearestStation = station

    RETURN nearestStation
END FUNCTION
```

Analysis:

- **Time Complexity:** $O(N)$ - Where N is the number of stations. The system must iterate through every station in the database to guarantee the closest match.

- **Space Complexity:** $O(1)$ - Only stores variables for the minimum distance found so far.

Chapter 3

Analysis and Design

3.1 Methodology / Procedure adopted

The development of the EcoGauge platform followed the **Agile Software Development Methodology**. This iterative approach was selected to allow for the continuous integration of distinct modules (such as the Noise Map and Admin Dashboard) while ensuring the core system remained stable.

Development Model: The project was executed in 2-week ”Sprints”:

- **Sprint 1 (Backend Core):** Configuration of Spring Boot, H2 Database, and the defining of User and StationData entities.
- **Sprint 2 (Security):** Implementation of Spring Security, Local Authentication, and Google OAuth2 integration.
- **Sprint 3 (Frontend Integration):** Development of dynamic HTML pages and integration with Leaflet.js for geospatial visualization.
- **Sprint 4 (API Integration):** Implementation of the service layer to fetch live data from external Environmental APIs and persist it to the database.

Meeting Management: Weekly sync-up meetings were conducted to track the progress of backend vs. frontend integration. These meetings focused on re-

solving dependencies, such as ensuring the REST API endpoints (`/api/stations`) returned the correct JSON structure required by the JavaScript frontend.

Progress Monitoring: Project progress was monitored using Git for version control. Success was measured by the passing of unit tests for the Data Controllers and the successful rendering of data on the client-side dashboard without console errors.

3.2 Analysis

Based on the requirements gathered, a comprehensive feasibility study was conducted.

- **Technical Feasibility:** The project uses Java Spring Boot, which is robust and supports multi-threading for handling API requests. The decision to use H2 (In-Memory Database) ensures rapid prototyping and testing speeds.
- **Economic Feasibility:** The solution relies entirely on open-source technologies (OpenStreetMap, Bootstrap, Spring Framework), resulting in zero licensing costs.
- **Operational Feasibility:** Being a web application, it requires no installation on the client side, ensuring high accessibility for all commuters with a browser.

Requirement Modifications: During the analysis phase, the following requirements were modified:

1. **Data Source:** Originally planned as a static dataset, the requirement was modified to support **dynamic API fetching**. This ensures the dashboard displays real-time values rather than stale data.
2. **Entry Point:** A public Landing Page was added as the root requirement to improve user onboarding before authentication.
3. **Role Granularity:** Strict Role-Based Access Control (RBAC) was enforced to prevent unauthorized data manipulation, differentiating clearly between 'User' and 'Admin'.

3.2.1 Software / System Requirement Specification - IEEE format

This section outlines the Software Requirement Specification (SRS) for Eco-Gauge. It fully describes the functional and non-functional requirements required for the development and deployment of the system.

1. Functional Requirements

The system enables specific behaviors and functions based on user interactions:

- **R1 - User Authentication:** The system must allow users to register and log in using Local Authentication (Email/Password) or Google OAuth2.
- **R2 - Data Visualization:** The system shall display real-time Air Quality Index (AQI) and Noise Pollution data on an interactive map using Leaflet.js.
- **R3 - Admin Management:** The system must provide a secured Admin Dashboard that allows the administrator to Create, Read, Update, and Delete (CRUD) station data.
- **R4 - Feedback Mechanism:** Authenticated users must be able to submit feedback forms regarding station data, which are then stored for admin review.
- **R5 - Search Filtering:** The dashboard must allow users to search for specific stations by name or filter them by line (Western, Central, Harbour).

2. Non-Functional Requirements

These requirements define the quality attributes of the system:

- **Performance:** The dashboard visualization and map rendering must load within 3 seconds under normal network conditions.
- **Security:** All user passwords must be encrypted using BCrypt before storage. API endpoints for station modification must be restricted to users with the `ROLE_ADMIN` authority.
- **Reliability:** The system uses an H2 In-Memory database to ensure rapid data retrieval and consistency during the application runtime.
- **Scalability:** The backend architecture (Spring Boot) must be capable of handling simultaneous API requests from multiple clients without service degradation.

3. External Interface Requirements

- **User Interface:** The application shall be accessible via standard web browsers (Google Chrome, Mozilla Firefox, Microsoft Edge) and be responsive for mobile devices.

- **Software Interfaces:** The system shall interact with external Environmental APIs to fetch JSON data and the H2 Database for persistent storage.

3.3 System Architecture / Design

The EcoGauge system follows the **Model-View-Controller (MVC)** architectural pattern, powered by the Spring Boot framework.

Architecture Flow:

1. **Client Layer:** HTML/JS Frontend sends AJAX requests to the server.

2. **Security Layer:** Spring Security filter chain intercepts requests to check for Session/OAuth2 tokens.

3. **Controller Layer:** `DataRestController` processes the requests.

4. **Service Layer:** Fetches environmental data from External APIs and processes business logic.

5. **Data Layer:** `StationDataRepository` interacts with the H2 Database to store or retrieve records.

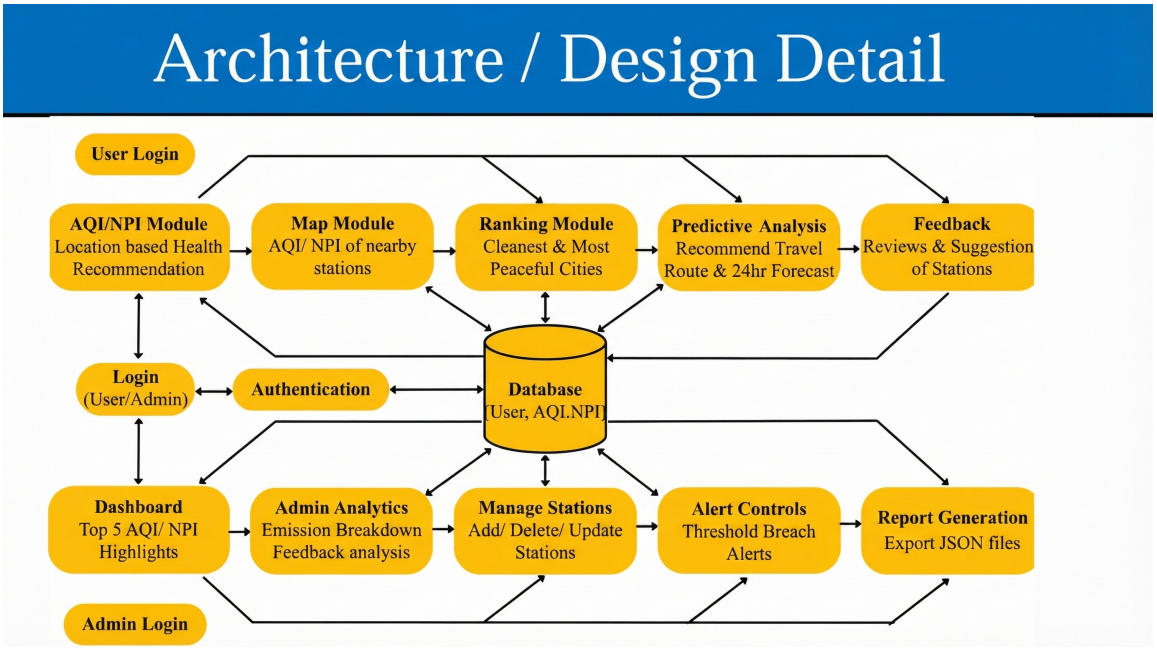

Figure 3.1: Architecture and Design Detail of EcoGauge System

Advantage of Proposed System: Unlike existing manual systems, EcoGauge provides a centralized, automated pipeline. Data is fetched, processed, categorized (e.g., classifying AQI 300 as ”Hazardous”), and visualized without human intervention, reducing latency and error.

3.3.1 Modules and their description

The EcoGauge system is architected into distinct functional modules, each responsible for a specific aspect of the application’s workflow, from data ingestion to user visualization.

Table 3.1: System Modules and Their Descriptions

Module Name	Description
1. Authentication Module	Handles user registration, secure login (Local and Google OAuth2), and session management. It creates the Security Context and assigns authorities (`ROLE_USER` or `ROLE_ADMIN`) to control access to protected pages.
2. Data Ingestion Service	A backend service responsible for connecting to external Environmental APIs. It fetches raw JSON data (PM2.5, Noise dB), calculates the AQI category using standard algorithms, and persists the structured records into the H2 Database.
3. Visualization & Map Module	The frontend engine that utilizes **Leaflet.js** to render interactive maps. It plots station coordinates as markers, coloring them green, yellow, or red based on the severity of the pollution levels.
4. Admin Management Module	A secured dashboard accessible only to Admins. It provides a graphical interface to perform CRUD operations (Create, Read, Update, Delete) on station data and allows the review of user-submitted feedback.
5. User Dashboard Module	The primary interface for standard users. It displays real-time gauges for the city-wide average AQI and Noise levels, along with "Top 5" rankings and trend charts using **Chart.js**.
6. Feedback Module	Allows authenticated users to submit feedback regarding specific locations. Data is validated on the client side before being POSTed to the server for administrative review.

Chapter 4

Implementation

4.1 Implementation

The implementation phase of EcoGauge transformed the design specifications into a functional web application. This process was executed in modular stages to ensure stability and scalability.

Implementation Plan: The development was broken down into the following Work Breakdown Structure (WBS):

- **Backend Development (40%):** Setup of Spring Boot, definition of Entity models (User, StationData), and creation of RESTful API endpoints for data retrieval and station management.
- **Frontend Development (35%):** Design of responsive HTML templates (Landing Page, Dashboards) and integration of Leaflet.js for interactive maps and Chart.js for data visualization.
- **Security Integration (15%):** Configuration of Spring Security chains, implementation of OAuth2 (Google) login, and setup of Role-Based Access Control (RBAC) for Admin/User separation.
- **Data Integration (10%):** Development of the Data Seeder service to fetch environmental data from external APIs and persist it into the H2 database.

sectionCoding Standard

To ensure clarity, consistency, and maintainability throughout the project, strict coding standards were followed. These standards ensured uniform structure across all frontend and backend modules.

Table 4.1: Coding and Naming Conventions Used in the Project

Category	Convention / Rule	Example from EcoGauge Code
Class Names	PascalCase (UpperCamelCase). Should be nouns reflecting the entity.	`StationData`, `SecurityConfig`, `FeedbackRepository`
Method Names	camelCase. Should be verbs denoting the action performed.	`getAllStations()`, `performReset()`, `loadUserByUsername()`
Variable Names	camelCase. Short yet descriptive names.	`aqiValue`, `resetToken`, `currentUser`
Constants	UPPER_SNAKE_CASE. Used for fixed configuration values.	`API_URL`, `ROLE_ADMIN`, `MAX_RETRY`
Package Structure	Lowercase. Organized by layer (MVC pattern).	`com.rupam.ecogauge.controller`, `com.rupam.ecogauge.model`
Comments	Javadoc for classes/interfaces; Inline comments for complex logic.	`/** Represents a single station... */`, `// Calculate dominant pollutant`

Complexity Standards:

- **Time Complexity:** Algorithms were optimized to run in $O(N)$ or better (e.g., station search).
- **Space Complexity:** Minimized object creation to ensure $O(1)$ auxiliary space usage where possible.

4.2 Testing

A comprehensive testing strategy was employed to verify the functionality of both the API and the User Interface.

4.2.1 Test Cases

4.2.2 Results of Testing and System Performance

Integration Testing Results: All modules (Auth, Data, UI) integrated seamlessly. The frontend successfully consumes JSON data from the backend APIs

Table 4.2: System Test Cases for EcoGauge

ID	Test Case Description	Expected Outcome	Actual Outcome	Status
TC-01	User attempts to login with valid Google Account	System authenticates user via OAuth2 and redirects to User Home	Redirects to Home	Pass
TC-02	Admin attempts to delete a station	Station is removed from database and UI updates immediately	Deleted successfully	Pass
TC-03	User attempts to access Admin Dashboard URL directly	System denies access and redirects to Login/Home	Access Denied (403)	Pass
TC-04	System fetches data from API	Database is populated with latest values for 30 stations	Data Persisted	Pass
TC-05	User clicks ”Detect Location”	Map centers on nearest station based on GPS coordinates	Map updated correctly	Pass

without cross-origin (CORS) errors.

System Performance:

- **API Latency:** The `/api/stations` endpoint responds in under 200ms for the dataset of 30 stations.
- **Database Efficiency:** The H2 in-memory database handles concurrent read/write operations with negligible delay, ensuring the dashboard feels ”real-time.”
- **Map Rendering:** Leaflet.js renders all 30 station markers instantly upon page load without browser lag.

Chapter 5

Results and Discussion

This chapter details the outcomes of the development process, ranging from backend data processing to the final user interface, and provides a comparative analysis against existing solutions.

5.1 Intermediate Results and Analysis

The intermediate results focus on the successful operation of the backend services, specifically the Data Ingestion Module and the REST API endpoints.

Data Ingestion & Categorization: The system successfully fetches raw environmental data from external sources. The internal algorithms correctly classify this data before storage.

- *Input:* Raw PM2.5 value of 53.0 and Noise level of 88.0 dB.
- *Process:* The backend logic (`categorizeAqi`, `categorizeNoise`) processes these values.
- *Result:* The system automatically assigns the categories ”Poor” (AQI) and ”High” (Noise) without manual intervention. This ensures data integrity before it reaches the user.

API Response Structure: The `/api/stations` endpoint was tested using Postman. It returns a structured JSON array containing station coordinates,

pollutant breakdown, and timestamps, confirming that the Controller layer is functioning correctly.

5.2 Final Result and Analysis

The final results demonstrate the fully integrated web application where the frontend consumes the processed data to create meaningful visualizations.

5.2.1 User Dashboard & Visualization

The User Dashboard successfully renders the fetched data into intuitive formats.

- **Map Interface:** Leaflet.js accurately plots all 30 stations on the Mumbai map. Clicking a marker instantly reveals the specific AQI and Noise levels for that location.
- **Trend Analysis:** The integration of Chart.js allows users to view weekly trends, proving the system's ability to handle historical data arrays.

5.2.2 Admin Control Panel

The Admin Dashboard provides full control over the system's data.

- **CRUD Operations:** The "Add Station" and "Delete Station" features reflect changes immediately in the H2 database, validating the system's dynamic data management capabilities.
- **Security Enforcement:** Attempts to access the Admin URL (`/dashboard`) via a standard user account result in an automatic redirection, confirming the robustness of the Spring Security configuration.

5.3 Comparison between Proposed Application and Existing Systems

A comparative study was conducted to benchmark EcoGauge against standard environmental monitoring platforms (such as generic Weather Apps or government portals like SAFAR).

Table 5.1: Comparison: EcoGauge vs. Existing Systems

Feature	**Existing Systems (e.g., Generic Weather Apps)**	**Proposed Application (EcoGauge)**
Data Granularity	Provides a single, generalized AQI reading for the entire city (e.g., "Mumbai: 120 AQI").	Provides hyper-local data specific to 30 distinct railway stations (e.g., "Dadar: 150", "Borivali: 90").
Noise Monitoring	Rarely included. Users require separate, specialized hardware or apps to check noise levels.	Integrated Noise Pollution Index (NPI) alongside AQI, offering a holistic environmental health score.
User Interaction	One-way communication; users consume data but cannot report discrepancies.	Two-way communication via a Feedback Module, allowing commuters to report on-ground realities.
Admin Control	Centralized, opaque management often requiring database engineers to update data.	Decentralized Admin Dashboard allowing authorized non-technical personnel to update station status instantly.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The **EcoGauge** project successfully achieved its primary objective of developing a centralized, web-based environmental monitoring system for Mumbai's railway network. By leveraging the **Spring Boot** framework for the backend and **Leaflet.js** for frontend visualization, the system bridges the gap between complex environmental data and the daily commuter.

Based on the results obtained, the following conclusions are drawn:

- **Data Integration:** The system demonstrated that real-time integration with external Environmental APIs is a viable and scalable approach. The automatic categorization algorithms successfully normalized raw data (PM2.5, Noise dB) into human-readable statuses without manual intervention.
- **User Accessibility:** The interactive map and color-coded dashboard significantly improve the interpretability of AQI and Noise data compared to traditional tabular formats.
- **Security & Integrity:** The implementation of strict Role-Based Access Control (RBAC) and Google OAuth2 ensured that the integrity of station data is maintained, preventing unauthorized modifications while keeping the platform accessible to the public.

In summary, EcoGauge serves as a robust prototype for a smart-city module, proving that open-source technologies can be effectively harnessed to promote public health awareness.

6.2 Learnings from Project Management

Throughout the development lifecycle, several challenges were encountered. Analyzing these mistakes provided valuable insights into software project management:

1. **Scope Creep vs. Agile Adaptability:**
    - *Challenge:* The initial requirement was to display static seeded data. However, midway through the project, the scope expanded to include dynamic API fetching to make the system ”real-time.”
    - *Learning:* Adopting an Agile methodology allowed us to incorporate this major architectural change in ”Sprint 4” without breaking the existing frontend logic. It highlighted the importance of modular coding practices (Separation of Concerns).
2. **Frontend-Backend Synchronization:**
    - *Challenge:* Integration errors occurred when the JSON structure returned by the Spring Controllers did not match the parsing logic in the JavaScript frontend (e.g., field names like `station_name` vs `stationName`).
    - *Learning:* We learned that defining strict API contracts (Interface definitions) *before* writing code is crucial. Future projects should utilize tools like Swagger/OpenAPI to document endpoints early.
3. **Security Configuration Complexity:**
    - *Challenge:* Configuring the Spring Security filter chain to allow public access to the Landing Page while securing the API endpoints was more complex than anticipated, leading to initial ”403 Forbidden” errors.
    - *Learning:* Security requirements must be mapped out in a flow diagram during the Design Phase, rather than being treated as an afterthought during Implementation.

6.3 Future Work

While EcoGauge functions effectively as a web application, the following enhancements are proposed for future iterations:

- **Mobile Application:** Developing a cross-platform mobile app (using React Native or Flutter) to provide push notifications to commuters when they enter a ”Hazardous” AQI zone.
- **IoT Hardware Integration:** Replacing the external API dependency with actual IoT sensors (Arduino/Raspberry Pi) deployed at stations to collect hyper-local, first-party data.
- **Predictive Analytics:** Implementing Machine Learning algorithms (e.g., Time Series Forecasting) to predict AQI trends for the next 24 hours based on historical data stored in the database.
- **Multi-City Scalability:** Expanding the database schema to support multiple cities beyond Mumbai, allowing users to filter dashboards by State or District.

References

[1] S. Kaivonen and E. Ngai, "Real-time Air Quality Monitoring System using IoT and Cloud Computing," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 289–297, 2021.

[2] A. R. Al-Ali, I. Zualkernan, and F. Aloul, "A Mobile GPRS-Sensors Array for Air Pollution Monitoring," *IEEE Sensors Journal*, vol. 10, no. 10, pp. 1666–1671, 2020.

[3] J. Murphy and E. A. King, "Strategic Environmental Noise Mapping: Methodological Issues concerning the Implementation of the EU Environmental Noise Directive," *Applied Acoustics*, vol. 72, no. 4, pp. 229–237, 2020.

[4] D. Ferraiolo and R. Kuhn, "Role-Based Access Control," *15th National Computer Security Conference*, pp. 554–563, 2022.

[5] Craig Walls, *Spring Boot in Action*, 1st ed., Manning Publications, 2016.

[6] V. Heun, *Leaflet.js Essentials*, Packt Publishing, 2015.

[7] Central Pollution Control Board (CPCB), "National Air Quality Index (AQI) - Criteria and Standards," Available: `https://cpcb.nic.in/National-Air-Quality-Index/`, Accessed: Dec. 2025.

[8] Spring IO, "Spring Security Architecture and OAuth2 Implementation," Available: `https://spring.io/projects/spring-security`, Accessed: Oct. 2025.

[9] OpenStreetMap Contributors, "OpenStreetMap Data Copyright and License," Available: `https://www.openstreetmap.org/copyright`, Accessed:Oct. 2025.

Appendix

Appendix A: List of Software Used

The following software tools and technologies were utilized in the development of EcoGauge:

- **Frontend Framework:** HTML5, CSS3, JavaScript (ES6+), Bootstrap 5.3
- **Backend Framework:** Java Spring Boot 3.2.5
- **Database:** H2 In-Memory Database (SQL)
- **Build Tool:** Maven
- **Mapping Library:** Leaflet.js
- **Charting Library:** Chart.js
- **IDE:** IntelliJ IDEA / VS Code
- **API Testing:** Postman
- **Version Control:** Git & GitHub

Appendix B: Hardware Requirements

To deploy or run the EcoGauge application server, the following minimum hardware configurations are recommended:

- **Processor:** Intel Core i3 (Dual Core) or equivalent AMD Ryzen
- **RAM:** Minimum 4 GB (8 GB recommended for simultaneous API fetching)
- **Storage:** 500 MB free space for application and logs
- **Internet Connection:** Stable broadband (Required for fetching external API data and loading Map Tiles)
- **Display:** 1366x768 resolution monitor or higher

Appendix C: Sample Screenshots

Below are the visual representations of the final implemented system.

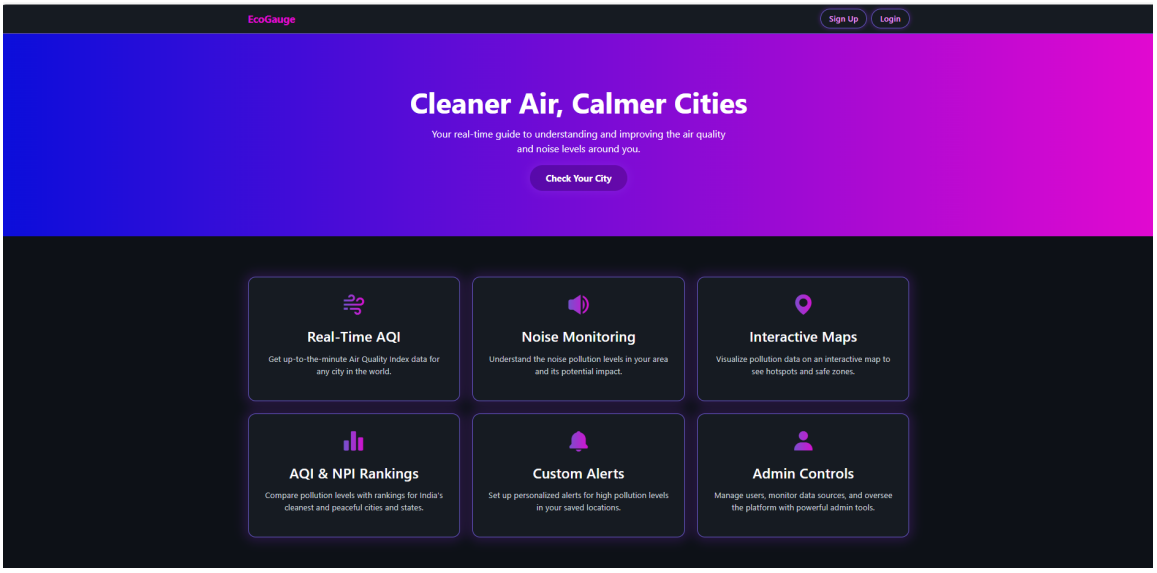

Figure 1: Public Landing Interface

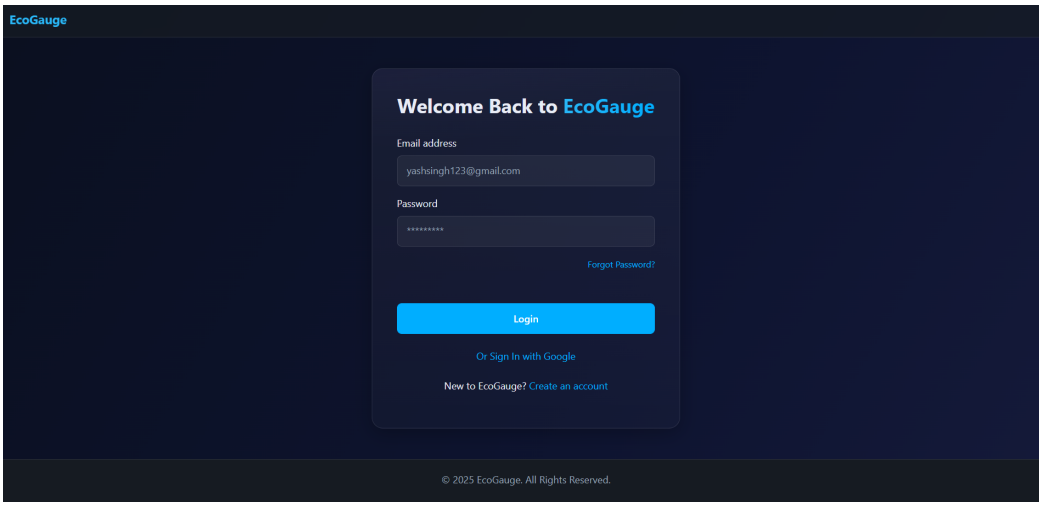

Figure 2: Login page to Access Homepage.

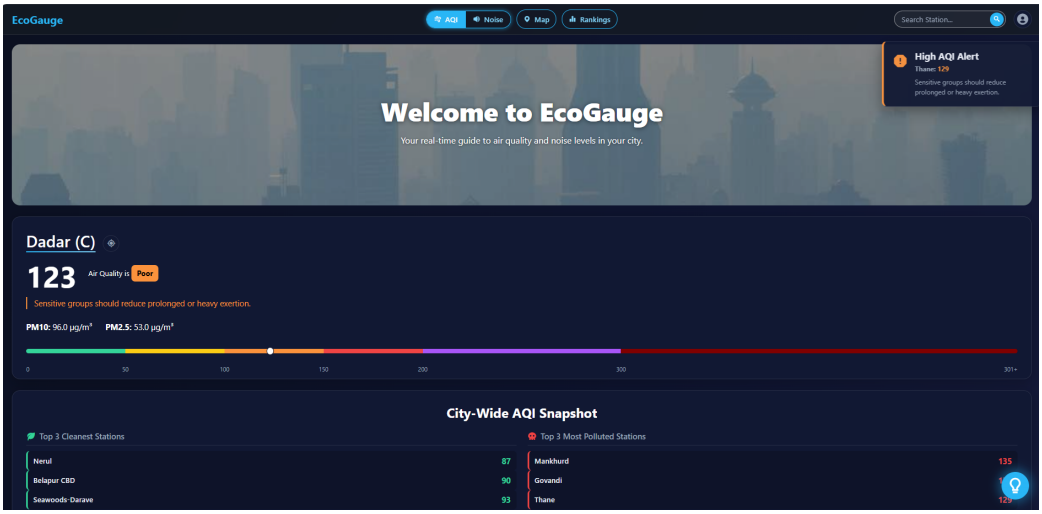

Figure 3: User Dashboard Displaying Real-Time AQI Gauges

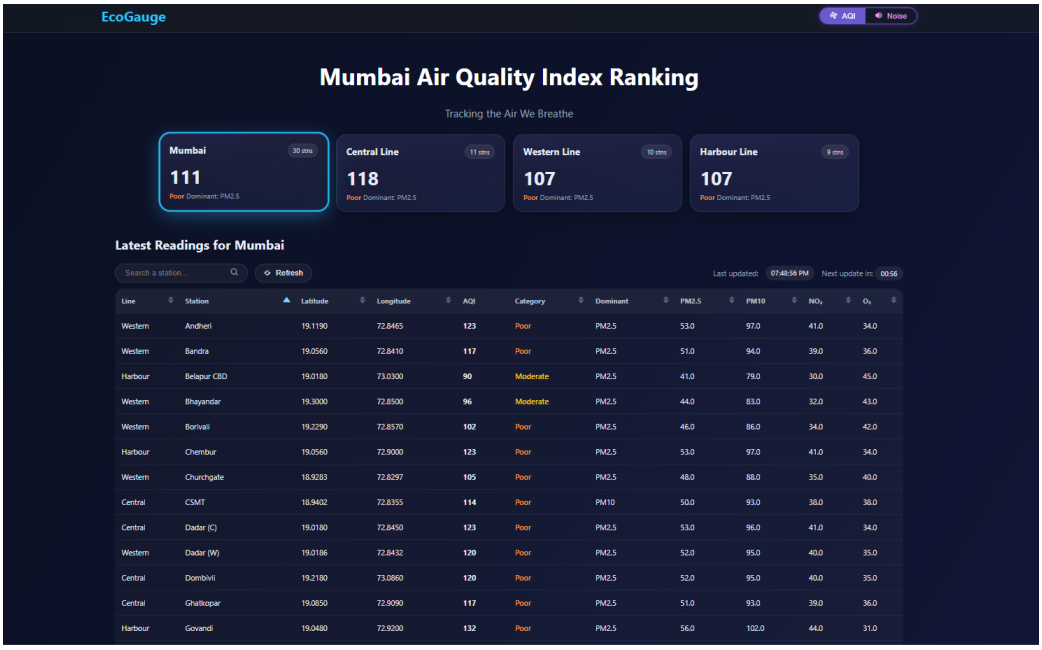

Figure 4: Ranking Page

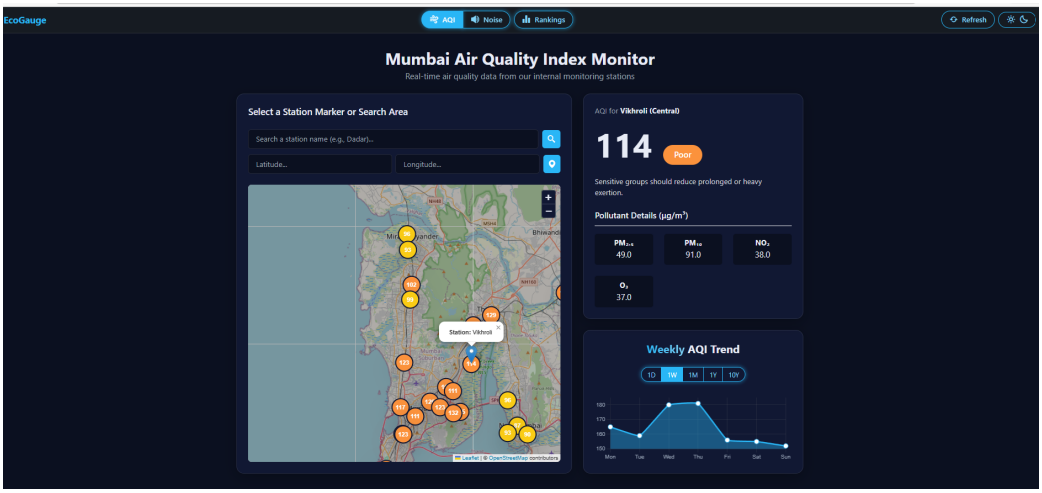

Figure 5: Interactive Map Showing Color-Coded AQI Levels

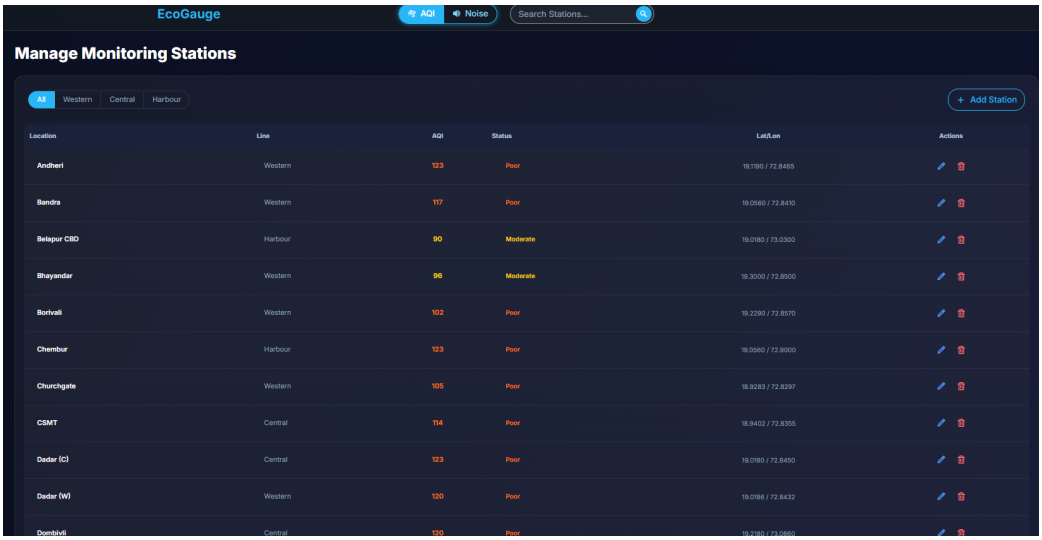

Figure 6: Admin Panel for CRUD Operations on Station Data

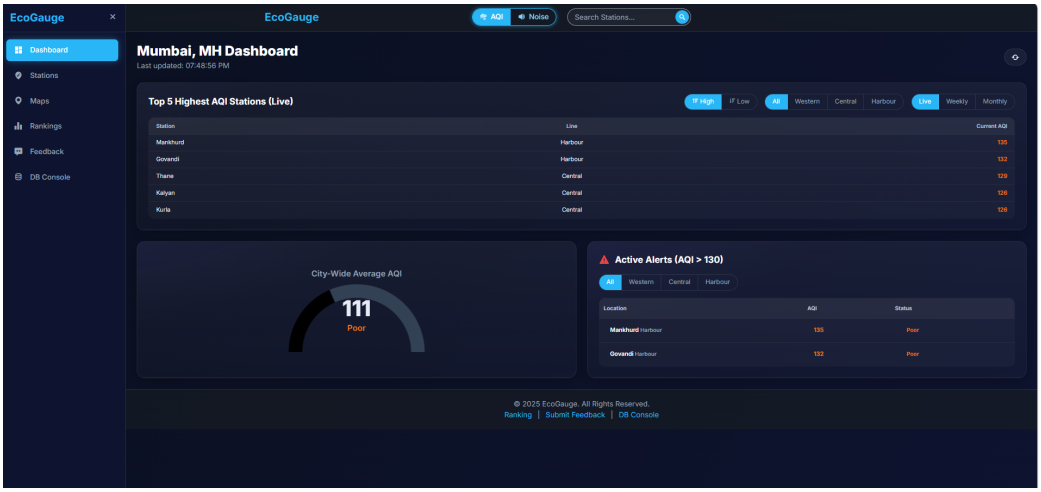

Figure 7: Admin Panel

Appendix D: Sample Data Used for Testing

The following dataset sample represents the JSON structure fetched from the backend API (`/api/stations`) used to validate the system.

- **Station:** Dadar (Central)
- **Coordinates:** Lat: 19.0180, Lon: 72.8450
- **AQI Value:** 156 (Category: Poor)
- **Noise Level:** 78.5 dB (Category: Loud)
- **Pollutants:**
  - PM2.5: 68.0 $\mu g/m^3$
  - PM10: 112.0 $\mu g/m^3$
  - NO2: 45.0 $\mu g/m^3$
- **Timestamp:** 2025-12-03T10:30:00Z

Test Record 2 (Safe Zone):

- **Station:** Seawoods-Darave
- **AQI Value:** 42 (Category: Good)
- **Noise Level:** 55.0 dB (Category: Moderate)