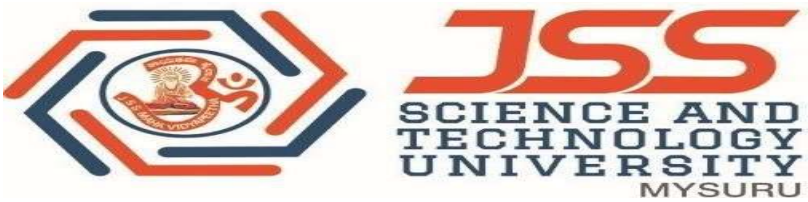
JSS MAHAVIDYAPEETHA

JSS Science and Technology University



"AirSphere"

Report for B.E Third Year Mini Project
On Database Management System

(Subject code: 22CS510)

Submitted by

Roll. No.	USN	NAME
55	02JST22UCS082	Purnaa L
04	01JST22UCS007	Aditi Katta
16	01JST22UCS067	Keerthana KV
30	01JST22UCS126	Raksha N Urs

Under the guidance of

Prof. Pooja JS

Assistant Professor

Dept. of CS&E

JSSSTU, Mysuru

ABSTRACT

AirSphere is an innovative web-based application designed to provide users with comprehensive air quality information while equipping administrators with robust tools for data management and user engagement. The platform integrates the World Air Quality Index (WAQI) API to offer real-time AQI metrics for cities worldwide. Users can view historical data, analyse trends through interactive graphs, and download reports in PDF format. A chatbot addresses user queries, while a dedicated contact form ensures effective communication. The admin dashboard provides features for managing city data, monitoring user engagement, resolving user queries, and analysing AQI trends with advanced metrics. Built using React for the front end and Java Spring Boot for the backend, with PostgreSQL as the database, AirSphere combines dynamic PDF generation, interactive analytics, and an efficient email notification system to deliver a seamless and intuitive experience.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

AirSphere is an advanced web-based application that combines real-time air quality monitoring, historical data analysis, and robust administrative tools to create a dynamic user experience. Designed for everyday users, AirSphere provides vital environmental insights through an intuitive interface and powerful backend technologies. The platform leverages React for its front end, Java Spring Boot for backend logic, and PostgreSQL for data management, ensuring efficiency, scalability, and user satisfaction. Key Features include:

User Features

- Real-time AQI Retrieval: Users can fetch their city's up-to-date Air Quality Index (AQI) data using the World Air Quality Index (WAQI) API. If the city is available, detailed AQI metrics are displayed; otherwise, users are notified that the city is unavailable.
- Historical Data Analysis: Users can select a city from a predefined list to view historical AQI data. The data can be sorted by date (ascending/descending) or pollution levels (most/least polluted). Users can also download the data as a PDF or analyse trends through graphical visualisations.
- Contact Form: A simple form allows users to submit their name, email, mobile number, and queries. Upon submission, users receive an automated email confirming that their query has been registered.
- Chatbot Assistance: Users can interact with a chatbot for instant answers to frequently asked questions, ensuring smooth and engaging user interaction.

Admin Features

- Data Management: Admins can add or view city details (e.g., name, description, author name) and upload historical AQI data, including metrics like ozone, NO_2, and SO_2 levels.
- Engagement Insights: The admin dashboard provides detailed analytics on user retention, new sign-ups, active users, churn rates, and session durations. Admins can visualise trends with interactive graphs or download reports in PDF format.
- Query Management: Admins can view, resolve, and mark user queries as resolved or unresolved for efficient query handling.

- Real-time AQI Analytics: Admins can monitor the national AQI, including average, maximum, minimum, median, and total AQI values, along with a standard deviation for comprehensive insights.

CHAPTER 2: REQUIREMENTS SPECIFICATION

The Requirements Specification chapter is essential for defining the functional, non-functional, and system requirements of the AirSphere project. It ensures a clear understanding of the system's expectations and serves as a blueprint for its design and development.

2.1 Introduction

This chapter defines the functional, non-functional, and system requirements for the AirSphere project. The purpose is to outline the core system objectives, constraints, and user expectations, ensuring that the system will meet its intended goals. The requirements have been compiled based on user and administrator perspectives to ensure the system is designed effectively and delivers the expected value.

2.2 Functional Requirements

2.2.1 User Features:

i. Fetch AQI Data

a. **Functionality**: Users can search for a city's current Air Quality Index (AQI) by entering its name.

b. Requirements:

- The system retrieves AQI data using WAQI API integration.
- City availability check: Notify the user if the requested city is unavailable or the data is inaccessible.

ii. View Historical AQI Data

a. Functionality: Users can select a city from a list to view its historical AQI data.

b. Requirements:

- Sorting options: Data should be sortable by date (ascending/descending) and by pollution levels (most/least polluted).
- Download option: Users can download historical data in PDF format for offline viewing.
- Graphical representation: Provide interactive graphs to visualise AQI trends over time.

iii. Contact Us Form

a. **Functionality**: Users can submit their name, email, mobile number, and a query or message.

b. **Requirements:**

- Upon submission, the system sends an automatic email acknowledgement to the user confirming the receipt of the query.
- The form must include validation checks to ensure proper input formatting.

iv. Chatbot Integration

a. Functionality: AirSphere integrates a chatbot to provide real-time assistance to users, answering frequently asked questions (FAQs) related to AQI and other related topics.

b. Requirements:

- The chatbot should deliver instant responses to users for a seamless user experience.
- The chatbot should be able to handle common queries such as city AQI status, how AQI is measured, and general pollution-related questions.
- The chatbot should have a learning mechanism to improve its responses based on user interactions.

2.2.2 Admin Features

i. Add and Manage City Data

a. Functionality: Admins can add new cities, update existing city information, and remove irrelevant cities from the system.

b. Requirements:

- Admins can input city details, including city name, description, and the author's name (to track data origin).
- Admins should be able to view, update, or delete city details from the system as required.
- Data validation should be in place to ensure consistency and accuracy of city data.

ii. Upload Historical AQI Data

a. **Functionality**: Admins can upload AQI data for cities, including pollutant levels (ozone, NO_2, SO_2, etc.) for specific periods.

b. Requirements:

- Provide an upload form where admins can enter or upload detailed AQI metrics in a predefined format.
- Include data validation to ensure the uploaded data is accurate and conforms to the system's format specifications.
- Support bulk data upload to streamline the process for cities with copious amounts of historical data.

iii. User Engagement Dashboard

a. **Functionality**: Admins can view key metrics related to user engagement on the platform, allowing for insights into user behaviour and platform usage trends.

b. Requirements:

- Display metrics such as user retention, churn rate, active users, and session durations.
- Represent data visually through graphs and charts to help admins identify trends and potential areas for improvement in user experience.
- Allow admins to filter the data by different periods or user demographics.

iv. Query Management

a. Functionality: Admins can view, resolve, and mark user-submitted queries as "resolved" or "unresolved" based on status.

b. Requirements:

- The system should maintain a query status table that allows admins to track unresolved queries, including query details and user contact information.
- Real-time notifications should alert admins of new or pending queries that need attention.
- Admins should be able to respond to or assign queries to appropriate departments or teams for resolution.

v. Real-time AQI Analytics

a. **Functionality**: Admins can monitor the national or city-level AQI metrics in real time, gaining insights into air quality across the regions.

b. Requirements:

- Provide national AQI statistics, including average, maximum, minimum, median, and total AQI values.
- Allow visualisations of the data through interactive graphs and charts.
- Provide options to export the data and analytics into various report formats, including PDF, Excel, and CSV.

vi. API Logs Monitoring

a. **Functionality**: Admins can monitor the performance of API calls, ensuring system reliability and tracking errors.

b. Requirements:

- Display API response codes such as 200 (success), 404 (not found), and 400 (bad request), along with timestamps and request details.
- Provide a detailed list of API logs to help troubleshoot any issues with data retrieval or system interactions.
- Allow admins to download and export API logs and performance metrics for further analysis.

2.3 Non-Functional Requirements

i. Performance:

- Ensure real-time responsiveness when fetching AQI data.
- Provide fast load times for historical data and interactive graphs.

ii. Scalability

- Support increasing user loads and larger datasets as the system grows.

iii. Reliability:

- Guarantee accurate AQI data retrieval from the WAQI API.
- Ensure the system is universally available with minimal downtime.

iv. Security:

- Protect sensitive user data (e.g., contact form submissions) using encryption.
- Authenticate admin access to secure data and feature management.

v. Usability:

- Design an intuitive interface for users and admins.
- Use clear visual cues for valid moves, sorting, and analytics.

vi. Maintainability:

- Structure the codebase for easy updates and debugging.
- Use modular design principles for efficient feature enhancements.

v. Compatibility:

- Ensure compatibility with major browsers and devices (desktop, mobile, tablet).
- Maintain cross-platform accessibility for diverse user bases.

2.3 System Requirements

i. Hardware Requirements:

For End Users:

- Device : PC, smartphone, or tablet.
- Internet connection for real-time data retrieval.

For Administrators:

- Device: PC or laptop with internet access.
- Storage Minimum 500MB free storage for accessing detailed analytics.

ii. Software Requirements:

- Frontend: React (with Node.js for package management).
- Backend: Java Spring Boot framework.
- Database: PostgreSQL for structured data management.
- Additional Tools: Recharts.js for graphical visualization, jsPDF for dynamic PDF generation, and an email service integrated with Spring Boot.

2.4 Constraints

- API Dependency: The system relies on the WAQI API for AQI data. Any downtime in the API service affects the system's functionality.
- Data Accuracy: The accuracy of AQI data depends on the source's reliability.
- Performance under Load: The system must handle high user loads during peak usage without degradation.
- Browser Compatibility: The system may behave inconsistently on outdated browser versions.

2.5 Assumptions

- Users have access to the internet to retrieve AQI data.
- Administrators are technically proficient enough to manage the admin dashboard.
- The WAQI API will continue providing free and reliable services.

By thoroughly defining these requirements, AirSphere ensures a robust foundation for its development, aligning functionality with user and admin expectations while adhering to technical and operational constraints.

CHAPTER 3: DETAILED DESIGN

3.1 Introduction

The Detailed Design chapter provides a comprehensive blueprint of the AirSphere system, outlining its architecture, components, and workflows. This design ensures the development process aligns with the defined requirements while maintaining scalability, maintainability, and performance.

3.2 Component Design

The website's frontend components are designed elegantly to facilitate smooth interfaces, dynamic updates, and easy usage. Some of the key components' design is as follows:

3.2.1 Frontend Components

User Components

i. Fetch AQI Module:

a. Input Field: Accepts user input for the city name.
b. Response Display: AQI data or an error message if the city is unavailable.
c. API Integration: Fetches data using the WAQI API.

ii. Historical AQI Module

a. City List: Displays a list of predefined cities for selection.
b. Data Table: Shows historical AQI data with sorting options (date, pollution level).
c. PDF Download: Allows users to export the data.
d. Graphical Analytics: Displays AQI trends using Recharts.js.

iii. Contact Form:

a. Input Fields: Name, email, mobile, and query.
b. Validation: Ensures required fields are filled before submission.
c. Acknowledgement: Sends an automated email confirmation using the backend.

iv. Chatbot:

a. Interactive Interface: Enables users to ask questions and receive instant responses.
b. Backend Connection: Retrieves answers to user queries.

Admin Components:

i. City Management Module:

a. Add City Form: Inputs for city name, description, and author name.
b. City List: Displays existing cities with options to edit or delete.
c. Confirmation Dialogs: Prompts for confirmation before deletions.

ii. Historical AQI Management Module:

a. Data Upload Form: Inputs for AQI metrics like ozone, NO_2, and SO_2.
b. Validation: Ensures correct data formats before submission.

iii. Engagement Dashboard:

a. Analytics Panels: Displays metrics like user retention, session durations, and churn rates.
b. Graphical Trends: Interactive charts to visualize engagement over time.

iv. Query Management Module:

a. Query List: Displays unresolved queries.
b. Resolution Status: Allows marking queries as resolved or unresolved.

v. API Logs Viewer:

a. Log Display: Lists response codes (200, 404, 400, etc.).
b. PDF Export: Generates downloadable reports of API performance.

3.3 Database Design

The database design for AirSphere is structured to ensure efficient data storage, retrieval, and integrity. It incorporates seven interconnected tables to manage various aspects of the system, from air quality metrics to user interactions and API logs. Each table is designed with specific attributes to support the functionalities defined in the requirements.

Table 1: airquality

This table stores detailed historical air quality data for various cities.

Column Name	Data Type	Constraints	Description
sl_no	INTEGER	Primary Key	Unique identifier for each record.
city	VARCHAR	Not Null	Name of the city.

no2	FLOAT		Level of nitrogen dioxide in the air.
so2	FLOAT		Level of sulfur dioxide in the air.
ozone	FLOAT		Ozone concentration.
co	FLOAT		Carbon monoxide levels.
aqi	FLOAT		Air Quality Index value.
verdict	VARCHAR		Description of air quality (e.g., Good, Poor).
date	DATE		Date of the air quality measurement.

Table 2: api_logs

This table logs API interactions to monitor system performance and reliability.

Column Name	Data Type	Constraints	Description
log_id	INTEGER	Primary Key	Unique identifier for each API log.
endpoint	VARCHAR	Not Null	API endpoint accessed.
request_ timestamp	TIMESTAMP		Timestamp of the API request.
response_ time	FLOAT		Time taken to respond to the API request.
status_ code	INTEGER		HTTP status code of the response.

Table 3: chatbot_query

This table manages user interactions with the chatbot.

Column Name	Data Type	Constraints	Description
query_id	INTEGER	Primary Key	Unique identifier for each chatbot query.
bot_response	TEXT		Response provided by the chatbot.
created_at	TIMESTAMP		Timestamp of when the query was created.
user_query	TEXT		Query submitted by the user.

Table 4: cities_added

This table stores information about cities added by administrators.

Column Name	Data Type	Constraints	Description
city_id	INTEGER	Primary Key	Unique identifier for each city.
city	VARCHAR	Not Null, Unique	Name of the city.
created_at	TIMESTAMP		Timestamp of when the city was added.
author	VARCHAR		Name of the admin who added the city.
description	TEXT		Description or details about the city.

Table 5: contact_us

This table handles user-submitted queries via the contact form.

Column Name	Data Type	Constraints	Description
contact_id	INTEGER	Primary Key	Unique identifier for each contact entry.
name	VARCHAR	Not Null	Name of the user submitting the query.
email	VARCHAR	Not Null	Email address of the user.
mobilenumber	VARCHAR		Mobile number of the user.
query	TEXT	Not Null	User's query text.

Table 6: user_engagement

This table tracks user engagement metrics for analysis.

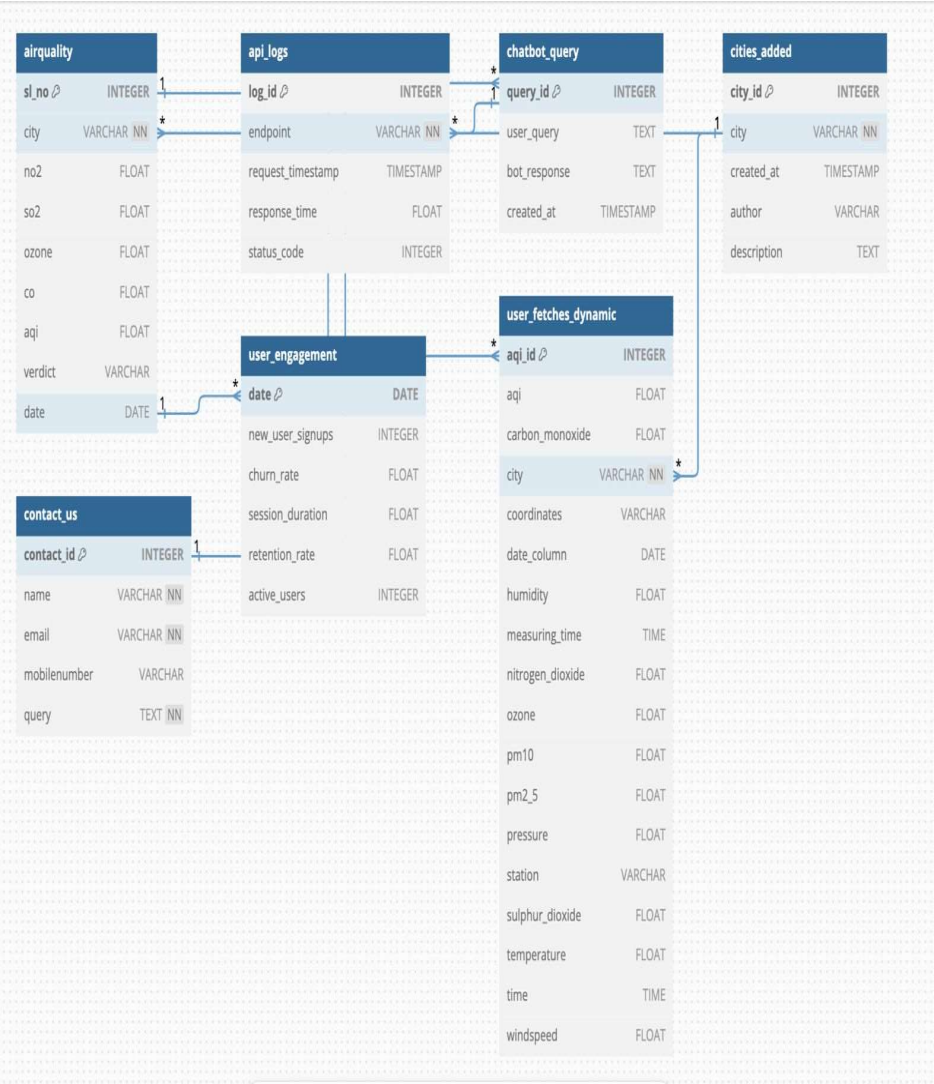
Column Name	Data Type	Constraints	Description
date	DATE	Primary Key	Date of the engagement metrics.
new_user_signups	INTEGER		Number of unaccustomed users who signed up.

churn_rate	FLOAT		Percentage of users who stopped using the system.
session_duration	FLOAT		Average session duration in minutes.
retention_rate	FLOAT		Percentage of returning users.
active_users	INTEGER		Number of active users on a given day.

Table 7: user_fetches_dynamic

This table records dynamic AQI data fetched by users.

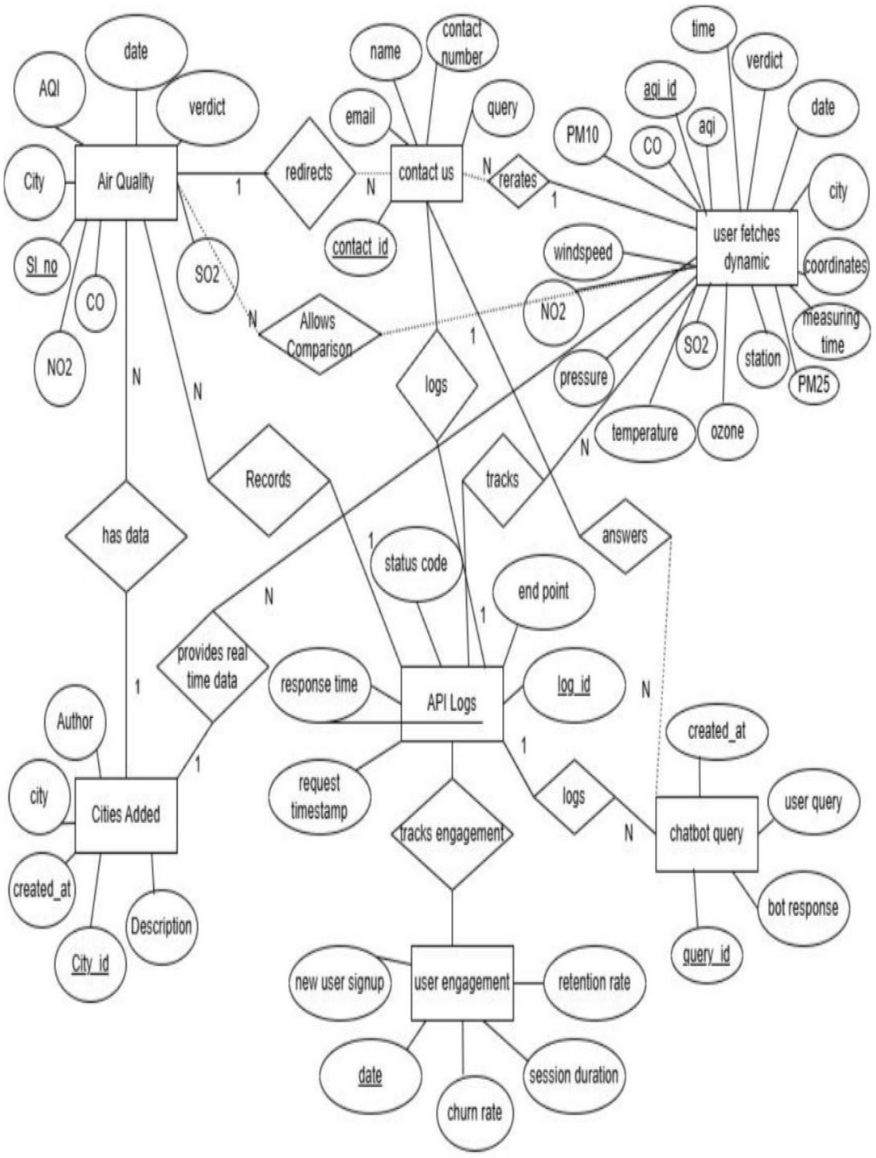
Column Name	Data Type	Constraints	Description
aqi_id	INTEGER	Primary Key	Unique identifier for each AQI entry.
aqi	FLOAT		Air Quality Index value.
carbon_monoxide	FLOAT		Carbon monoxide levels.
city	VARCHAR	Not Null	Name of the city.
coordinates	VARCHAR		Geographical coordinates of the station.
date_column	DATE		Date of the AQI reading.
humidity	FLOAT		Humidity level at the station.
measuring_time	TIME		Time of the AQI reading.
nitrogen_dioxide	FLOAT		Nitrogen dioxide levels.
ozone	FLOAT		Ozone concentration.
pm10	FLOAT		Particulate matter (PM10) levels.
pm2_5	FLOAT		Particulate matter (PM2.5) levels.
pressure	FLOAT		Atmospheric pressure.
station	VARCHAR		Station name or identifier.
sulphur_dioxide	FLOAT		Sulfur dioxide levels.
temperature	FLOAT		Temperature at the station.
time	TIME		Specific time of the reading.
windspeed	FLOAT		Wind speed at the station.



AirSphere's DBML Schema

3.3.1 ER Diagram

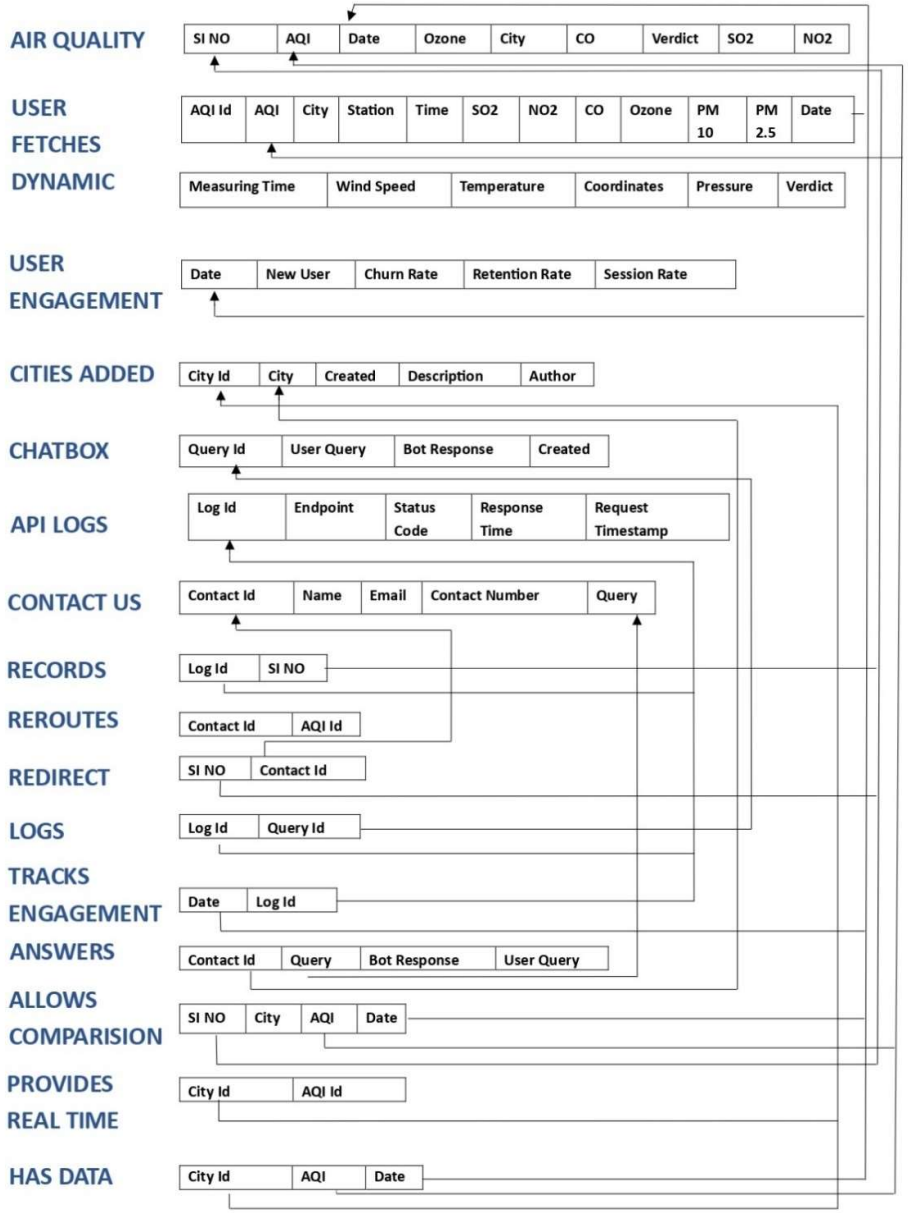
An entity relationship diagram (ERD) is a visual representation of how entities in a database are connected. ERDs are a type of flowchart that uses symbols to show the relationships between entities, attributes, and keys.



ER Diagram for AirSphere

3.3.2 Schema Diagram

This database design ensures that the AirSphere system operates efficiently, maintains data integrity, and supports both user and admin functionalities seamlessly.



Schema Diagram for AirSphere

3.4 Workflow Design

3.4.1 Fetch AQI Workflow:

- The user enters a city name.
- The front end sends a request to the WAQI API through the backend.
- The backend processes the response and returns AQI metrics or an error message.
- The front end displays the data to the user.

3.4.2 Historical Data Workflow:

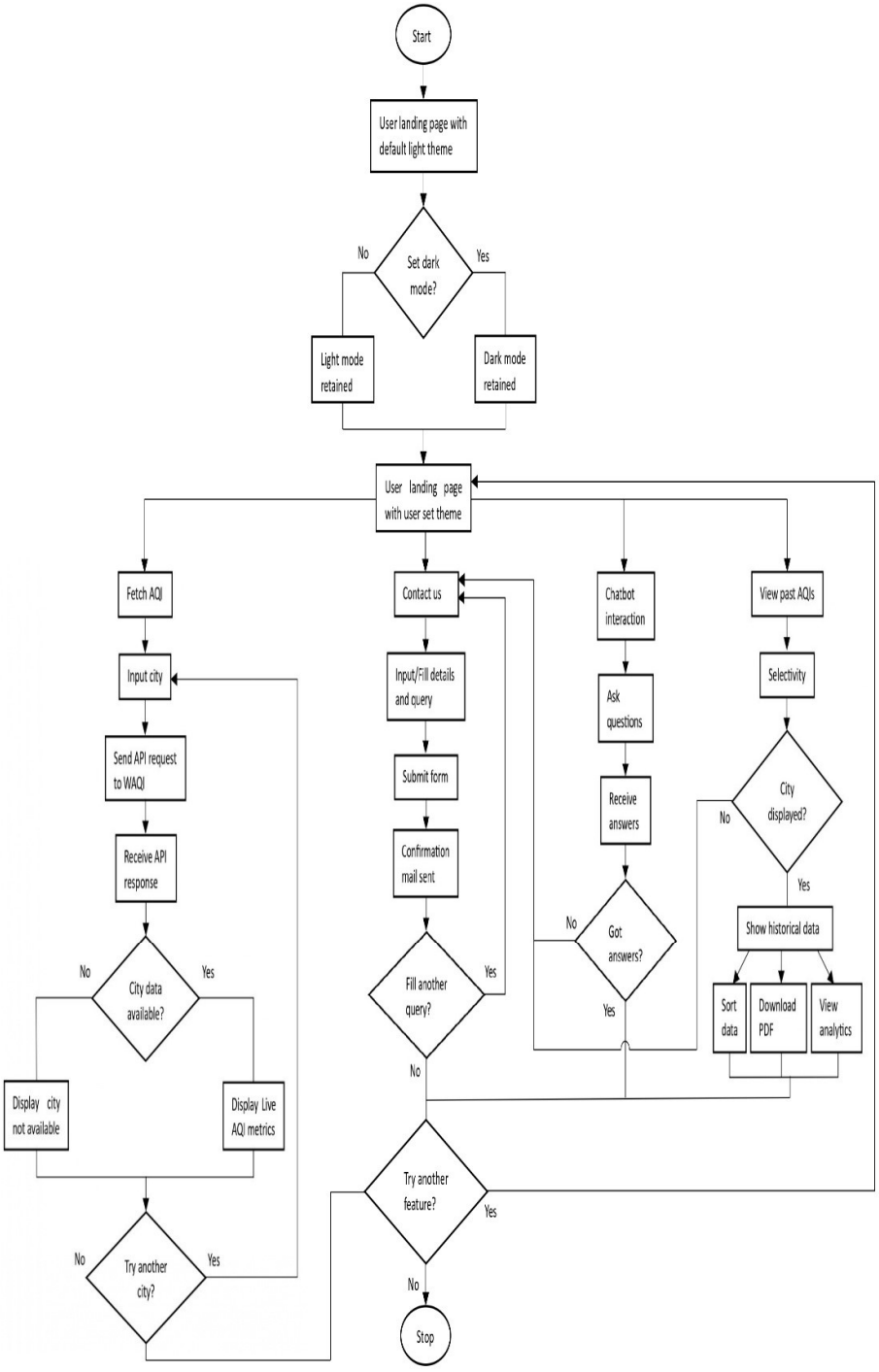
- The user selects a city from the list.
- The front-end requests historical data from the backend.
- The backend retrieves data from the database and returns it to the frontend.
- The user views data, sorts it, downloads a PDF, or analyses it via graphs.

3.4.3 Contact Us Workflow:

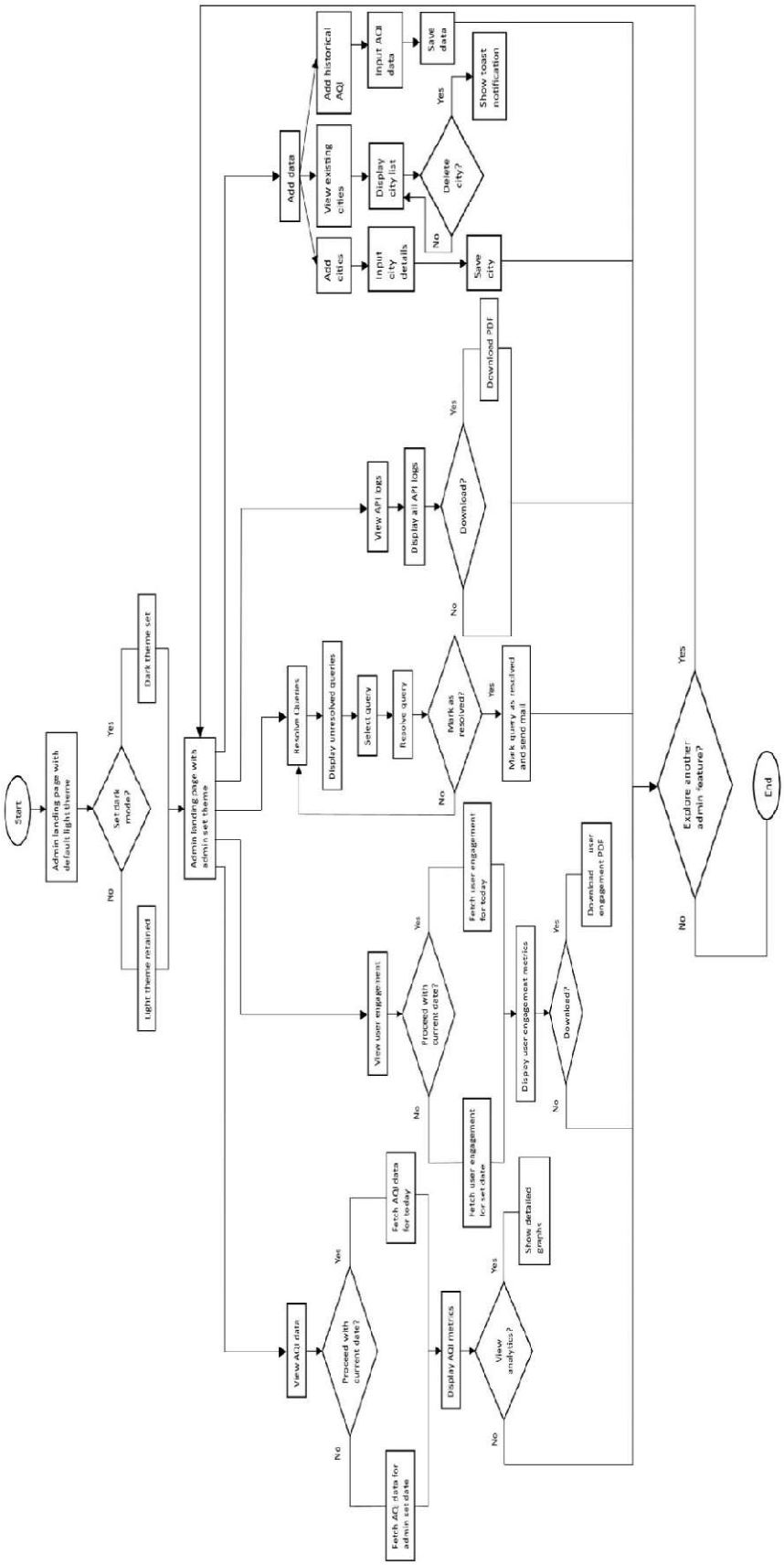
- The user fills in the form and submits it.
- The backend validates the input and stores the query in the database.
- The email service sends an acknowledgement to the user.
- Admin reviews the query and updates its status in the Query Management Module.

3.4.4. Admin Workflow:

- City Management: Admin adds, edits, or deletes city information.
- Historical AQI Data: Admin uploads AQI metrics for a city.
- Dashboard: Admin views and downloads engagement statistics.
- API Logs: Admin monitors API performance and downloads reports.
- Query Resolution: Admin views unresolved queries, resolves them, and updates their status.



User Flow Chart for AirSphere



Admin Flow Chart for AirSphere

3.5 Graphical User Interface Design (GUI)

The Graphical User Interface (GUI) design for the AirSphere project is centred around providing a seamless, intuitive, and visually appealing user experience. The design will prioritise simplicity, accessibility, and functionality across user and admin interfaces, ensuring the system is easy to navigate and responsive and provides interactive data visualisations. Below is a detailed breakdown of the GUI design for each page and section.

3.5.1 User Interface

i. Homepage

The homepage serves as the entry point for users and should offer easy access to the primary features such as AQI data, historical data, and chatbot assistance.

ii. Navigation Bar:

- A sticky header navigation bar will allow easy access to the following:
- Logo: Positioned on the left, leading back to the homepage.
- Links: Clear links to Historical Data, Contact Us, and Chatbot Assistance for quick access to respective pages.
- Responsive Design: The navigation bar adjusts seamlessly across mobile, tablet, and desktop devices.

iii. Key Features Section:

A section on the homepage will highlight key features, such as:

- Fetch AQI Data: Descriptive call-to-action (CTA) prompting users to enter a city name.
- View Historical Data: A visually engaging button linking to the historical data page.
- Chatbot: An AI-powered chatbot icon positioned at the bottom right of the screen for quick access to instant assistance.
- Responsive Elements: All elements on the homepage should be resized dynamically based on the screen size for optimal use on desktop and mobile devices.

iv. Historical Data Page

The Historical Data page allows users to explore and download AQI data over time. This page should have a clean, organised layout with multiple filtering and sorting options.

a. City Selection Carousel:

- A carousel menu will allow users to select a city from a list or search for one. Once selected, AQI historical data for that city is displayed. The carousel will support animation, helping users quickly locate the desired city.
- A loading spinner will appear while the data for the selected city is being fetched.

b. Data Display:

- Sortable Table: Below the city selection, a table displays the historical AQI data. Columns will include the date, AQI value, and pollutant levels (e.g., ozone, NO_2, SO_2). Users can sort the table by each column (ascending/descending).

c. Graphical Analysis:

- Interactive line charts or bar graphs will visualize AQI trends over time. Users can hover over chart elements to view detailed data for specific dates or periods.
- The graphs allow users to zoom in/out, explore data ranges, and toggle specific pollutants to compare trends.

d. Download Options:

- A Download button will allow users to download the AQI data in PDF format. This button should be placed prominently above the table or within the page's footer.
- The button's icon should be visually aligned with a PDF file symbol to ensure clarity.

e. Analytics Toggle:

- An analytics toggle will allow users to switch between a tabular view and a graphical view of the data.

v. Contact Us Page

- The Contact Us page is designed to collect user queries and provide instant email feedback. Form Fields: The contact form will include the following fields:
    - Name: Input field for the user's full name.
    - Email: Input field for the user's email address.
    - Mobile Number: Input field for the user's mobile number (with input validation for proper format).
    - Query: A text area where users can type their message or question.

  The fields should be clearly labelled, and placeholders should be provided to guide users on the expected input format.
- Submit Button: A Submit button will appear beneath the form, with clear validation. The button should be disabled until all required fields are correctly filled out. An email confirmation will automatically be sent to the user upon successful submission.
- Confirmation Message: After form submission, a confirmation toast will appear, confirming the submission and displaying a message like "Your query has been submitted." The user will also get an automatic email acknowledgement.
- Error Handling: If there are any issues with the form submission, a red error message will be displayed next to the respective field, indicating what needs to be corrected.

vi. Chatbot Assistance

- The chatbot serves as a virtual assistant, helping users instantly with AQI-related queries and general information.
- Positioning: A chatbot icon will appear at the top right of the screen, in the navbar, ensuring that it does not obstruct the main content. The icon will be designed as a friendly, simple chatbot symbol.
- Chat Window: Clicking on the icon will open a pop-up chat window. The window should have:
    - A text input box at the bottom for the user to type their query.
    - A typing indicator when the chatbot is processing the user's request.
    - Instant responses with relevant AQI information.

3.5.2 Admin Interface

The Admin Interface will provide a dashboard and various management tools to interact with the system and monitor user activity.

i. Dashboard

The Admin Dashboard should present a summary of key metrics in a visually engaging way.

- Metrics Overview: Key metrics such as user retention, active users, and session duration will be represented using interactive charts and graphs. Bar charts and pie charts will display distribution metrics like the number of active users per city or the average AQI readings.
- Graphical Representation: The metrics should be updated in real time, with graphs adjusting dynamically to represent changes in user behaviour or AQI statistics.
- Navigation Sidebar: A vertical sidebar should be included on the left for easy navigation between different admin sections such as City Management, Query Management, AQI Analytics.

ii. City Management

Admin functionality to manage cities involves adding, editing, and deleting city data.

- City List: The list of cities will be presented in a data table with columns like city name, description, and the date it was added. Each city entry will have delete buttons.
- City Data Entry Form: A form will be provided to add or edit city details. The form will include fields for the city name, description, and author's name. A Save button will allow the admin to commit changes.

iii. Query Management

Admin tools for managing and resolving user queries will be displayed in a query management table.

- Query Table: Each query will be listed with columns for the user's name, query subject, submission date, and status (resolved or unresolved).
- Search and filtering options will be included to help admins find specific queries quickly.
- Resolve Query: Admins will have the option to mark queries as resolved or unresolved. Once marked resolved, the query will be

moved to a separate section and archived.

3.5.3 Overall Design Considerations

- Responsive Design: The entire UI should be designed to be fully responsive across a range of devices and screen sizes, from desktop to mobile, ensuring usability on smartphones and tablets.
- Color Scheme: The color scheme should reflect the project's theme (environmental focus). Soft greens, blues, and earth tones can be used to convey cleanliness, nature, and sustainability.
- Typography: Clear and readable fonts such as Roboto and Playwrite NL Guides should be used, with appropriate contrast, to ensure accessibility and readability.
- Consistent Visual Style: Consistency in buttons, icons, and layouts across pages will ensure the UI is cohesive and user-friendly.

This detailed GUI design ensures that users and administrators can easily interact with the system while leveraging advanced visualizations, intuitive forms, and dynamic features for a seamless experience.

3.6 Security Design

i. Authentication and Authorization:
   - Secure access for administrators.
   - Role-based access control to restrict admin-only features.

ii. API Security:
   - Validation of input data to prevent malicious requests.
   - Rate limiting to prevent API abuse.

iii. Database Security:
   - Regular backups and secure access controls.
   - Protection against SQL injection through parameterized queries.

By detailing the system architecture, components, workflows, and security measures, this chapter ensures a clear roadmap for the development and deployment of AirSphere.

CHAPTER 4: IMPLEMENTATION

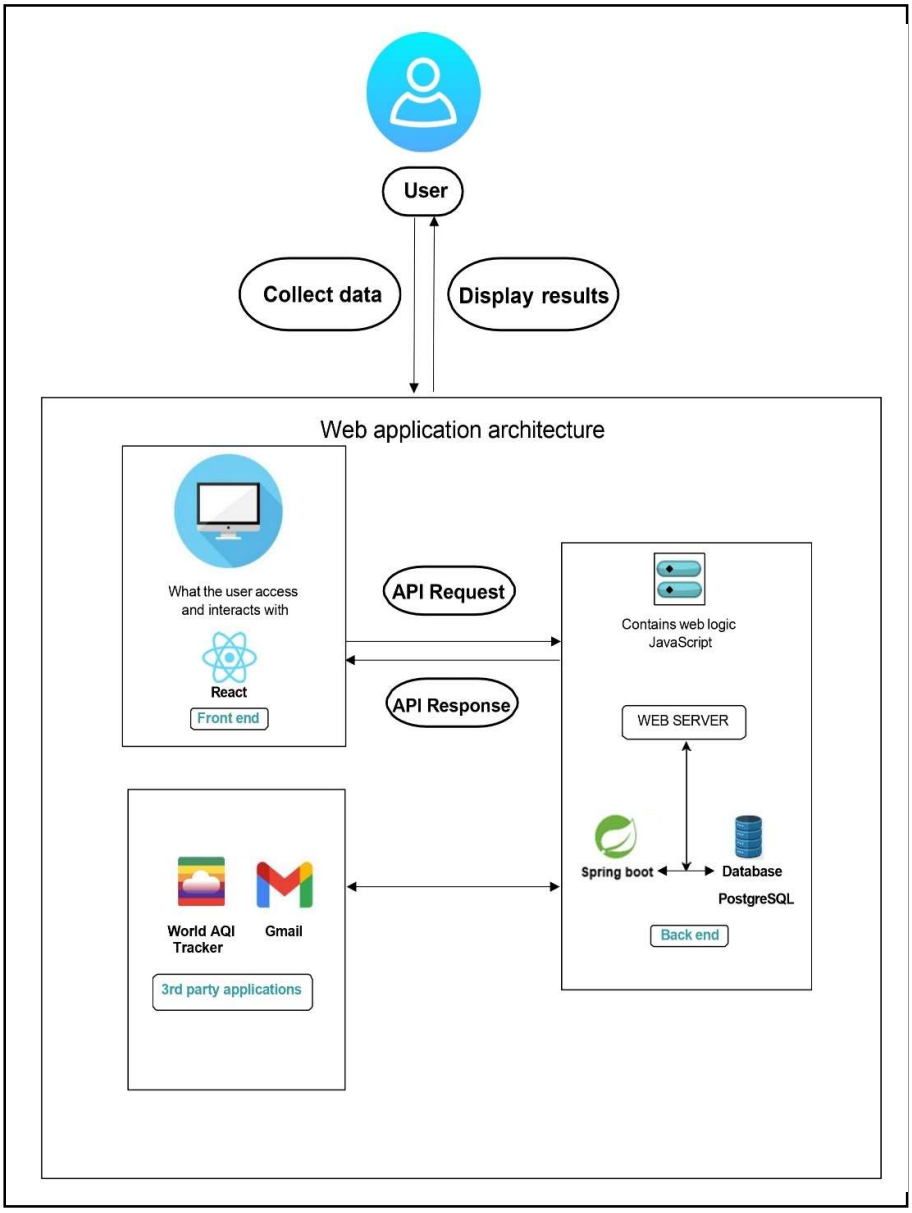
4.1 Introduction

The AirSphere platform integrates React and Spring Boot backend to deliver real-time AQI data, historical insights, and interactive features. This section outlines how the front and back end communicate and work together, discussing key concepts like API endpoints, annotations, integration flow, and email functionality.

4.2 System Architecture

The AirSphere platform follows a three-tier architecture, with distinct layers for the frontend, backend, and database.

- Frontend: The frontend uses React, a popular JavaScript library for building dynamic user interfaces. React allows the creation of reusable components and ensures a responsive, fluid user experience. It communicates with the backend via RESTful APIs, ensuring data is fetched asynchronously for real-time updates.
- Backend: The backend is implemented with Java Spring Boot, a powerful framework that simplifies the development of robust, secure, and scalable web applications. The backend is the intermediary between the front end and the database, processing business logic, handling API requests, and managing authentication and user data.
- Database: The database for AirSphere is PostgreSQL, a relational database management system (RDBMS) known for its reliability and support for complex queries. It stores city data, AQI metrics, user queries, and system logs with a well-structured schema that ensures efficient data retrieval and scalability.
- Third-party Integrations:
    - WAQI API: Used to fetch real-time AQI data for cities worldwide.
    - PDF Generation Library: Integrated into the backend to generate downloadable reports in PDF format for historical AQI data.
    - Email Notification System: Utilises a third-party service (SMTP) to send email confirmations and notifications.



Architectural Diagram of AirSphere

4.3 Backend Implementation (Spring Boot)

The backend of AirSphere is powered by Spring Boot, providing robust and scalable architecture. It utilises various Spring annotations to map HTTP requests to appropriate methods in the controllers, enabling the front end to fetch data, submit queries, and manage city information. The backend integrates with external APIs (such as the WAQI API) and provides endpoints for CRUD operations on AQI data, city management, and user queries.

4.3.1 Key Annotations in the Backend

- @RestController: Marks the class as a controller where every method

returns a JSON response, enabling the creation of RESTful web services.

- @RequestMapping: Specifies the base path for API endpoints, allowing for the consolidation of similar routes under a common URL prefix.
- @GetMapping, @PostMapping, @PutMapping, @DeleteMapping: Map HTTP GET, POST, PUT, and DELETE requests to specific handler methods, enabling CRUD operations tailored to different HTTP methods.
- @CrossOrigin: Enables Cross-Origin Resource Sharing (CORS), allowing the backend to accept requests from a different domain or port, which is essential for frontend-backend communication in distributed systems.
- @Autowired: Injects dependencies automatically by Spring's dependency injection mechanism, facilitating the creation of loosely coupled, easily testable components.
- @Override: Used to indicate that a method is overriding a method from a superclass, providing custom behavior while maintaining a clear inheritance structure.
- @Configuration: Indicates that the class contains Spring Bean definitions and configurations, enabling the customization of application context setup.

4.3.2 Key API Endpoints

- /api/aqi: Retrieves current AQI data for a specified city.
- /api/aqi/{id}: Fetches detailed AQI data for a specific record using its unique identifier.
- /api/aqi/chatbot: Manages queries and responses for the chatbot, enabling dynamic user interactions related to air quality.
- /api/aqi/admin/add-cities: Allows administrators to add new cities along with their AQI data, extending the system’s coverage.
- /api/aqi/contact-us: Submits user feedback or inquiries from the contact form, aiding in user support and engagement.
- /api/historical-data: Provides historical AQI data for a city within a specified date range, supporting trend analysis and historical insights.
- /api/historical-data/{id}: Retrieves detailed historical AQI data for a

specific record by its unique identifier.

- /api/sql/aqi: Returns comprehensive AQI metrics for dashboard visualization, including statistics on air quality and user engagement.
- /api/sql/aqi/{id}: Fetches specific AQI metrics or engagement data for a particular record, useful for detailed analysis or reporting.

4.3 Spring Email Integration

The Spring Boot backend uses Spring Mail to handle email notifications for various system processes, such as acknowledging user queries, notifying administrators about new queries, or sending reports. This integration is crucial for ensuring effective user communication and maintaining an engaged experience. Email Features include:

- User Query Acknowledgement: When a user submits a query through the contact form (via the /api/aqi/contact-us endpoint), the backend triggers an email to acknowledge receipt of the query. This ensures that users are informed that their issue is being processed.
- Admin Notifications: The system can also send email notifications to administrators when new queries are submitted, providing real-time updates about user interactions.
- Error and System Alerts: Email notifications can be set up for system errors or performance issues, allowing administrators to be promptly notified and take corrective actions.

Spring Mail Integration: The Spring Mail integration is set up in the backend to send emails using SMTP or an external mail service provider like Gmail. It involves configuring an SMTP server in the application properties, which is then used by the backend to send emails.

4.4 Frontend Implementation (React)

The front end of AirSphere is developed using React, providing a dynamic and interactive user interface. The platform uses React hooks like useState, useEffect, and useRef to manage the state, handle side effects, and interact with the DOM. The front end communicates with the backend using RESTful APIs. Key Aspects of Frontend Implementation:

- API Communication: The front end makes HTTP requests (GET,

POST, etc.) to the backend API endpoints for fetching AQI data, submitting user queries, and managing historical data. These requests are made asynchronously using Axios or Fetch.

- State Management: React's useState and useEffect hooks manage dynamic state and trigger API calls. For example, when a city is selected, the corresponding AQI data is fetched from the backend.
- Dynamic UI Updates: As data is fetched or changed (e.g., the user submits a form), the state is updated and React automatically re-renders the UI components.
- Recharts.js: Interactive graphs are implemented to visualise AQI trends and data over time.

4.5 Frontend-Backend Communication

The front end communicates with the backend using RESTful APIs, where the React application sends HTTP requests to specific API endpoints mapped in the Spring Boot backend.

- Base URL: https://localhost:8080/api/
- GET /api/aqi/{city}: Retrieves the current AQI data for the specified city, providing real-time air quality information.
- POST /api/aqi/contact-us: Submits user feedback or inquiries from the contact form, storing or forwarding the messages for administrative action.
- GET /api/historical-data: Fetches historical AQI data for a specified city and date range, allowing users to analyze past air quality trends.
- POST /api/aqi/admin/add-cities: Enables administrators to add new cities with their corresponding AQI data, extending the application's geographical coverage.
- GET /api/sql/aqi: Returns AQI metrics and user engagement data for dashboard reports, aiding in monitoring and analysis.
- GET /api/aqi/chatbot: Processes and responds to chatbot queries, offering automated answers to common AQI-related questions.
- GET /apilogs: Retrieves a list of all API logs, providing detailed records of system activities and interactions.
- GET /apilogs/{id}: Fetches a specific API log entry by its unique ID, offering detailed information about that event.

- POST /apilogs: Adds a new API log entry, documenting specific events or transactions within the system.
- PUT /apilogs/{id}: Updates an existing API log identified by its unique ID, allowing for corrections or additional information.
- DELETE /apilogs/{id}: Deletes a specific API log by its ID, removing it from the system to manage log data effectively.
- GET /api/users: Retrieves a list of all users in the system, providing user details such as roles and activity status.
- GET /api/users/{id}: Fetches details of a specific user by their ID, providing personalized data and account information.
- POST /api/users: Creates a new user account, adding them to the system with specified details and permissions.
- PUT /api/users/{id}: Updates information for an existing user by their ID, allowing changes to roles or personal data.
- DELETE /api/users/{id}: Deletes a user account by ID, removing their access and associated data from the system.

The backend processes these requests, interacts with the database (PostgreSQL) and external APIs (like WAQI), and returns the required data in JSON format, which is then consumed by the front end.

4.6 Error Handling and Validation

The backend and frontend work together to handle errors and provide a smooth user experience.

- Backend Error Handling: Spring Boot provides centralised exception handling via @ControllerAdvice. Custom exceptions can be created for specific error scenarios, such as invalid city data or missing information. This ensures that the front end receives a consistent response format.
- Frontend Error Handling: The front end uses useState to manage error messages. These messages are displayed to users when API calls fail (e.g., incorrect city name, no data available).
- Input Validation: Spring Boot handles input validation at the backend using annotations like @Valid and @NotNull, ensuring data integrity.

The AirSphere platform seamlessly integrates a React frontend with a Spring Boot backend, creating a responsive and interactive system for monitoring air quality. Both systems interact efficiently through well-defined RESTful API endpoints, enabling the front end to fetch real-time AQI data, display historical information, and manage user queries. The backend, built with Spring Boot, handles data processing, storage, and integration with external APIs. At the same time, the front end provides a dynamic interface using React hooks to manage the state and communicate with the backend. Spring Mail integration adds a vital communication layer, ensuring users and administrators stay informed via email notifications. This integration ensures that AirSphere delivers a reliable, intuitive, and scalable solution for air quality monitoring.

CHAPTER 5: TESTING

5.1 Introduction

Testing plays a vital role in verifying the functionality, performance, and security of the AirSphere system. This chapter outlines the testing approach used to validate the various features and workflows of the AirSphere application. It includes the User Interface (UI) and Admin Interface testing to ensure that all critical user interactions, backend processes, and API responses meet the expected behaviour.

5.2 Testing Overview

The testing approach involves several categories:

- Unit Testing: Individual components and methods are isolated to ensure correctness.
- Integration Testing: Verify the integration of multiple components and ensure proper data flow between the front and back end.
- End-to-end Testing: Simulating real-world user scenarios ensures the overall system works as intended.
- Regression Testing: Ensuring recent changes do not break existing features.
- User Acceptance Testing (UAT): Verifying that the application meets user requirements and business needs.

5.3 User Interface Test Cases

Test Case No	Test Case Description	Expected Result	Test Case Result	Passed (Yes/No)
TC1	User enters a valid city name in AQI Fetch workflow	Valid API response with AQI data for the entered city	Available	Yes
TC2	User enters a non-existent city name in	Invalid API response: City Not Found	Not Available	Yes

	AQI Fetch workflow			
TC3	User submits contact form with all fields filled	Form is successfully submitted, user receives a confirmation email	Submitted	Yes
TC4	User submits contact form with missing fields	Error message displayed indicating missing required fields	Error Returned	Yes
TC5	User selects a city and fetches historical AQI data	Valid API response: Historical data displayed	Data Available	Yes
TC6	User selects a city and requests historical data with no results	API returns an empty dataset or error if city data is missing	No Data Available	Yes
TC7	User clicks on the “Download Historical Data” button	Historical AQI data is successfully downloaded as a PDF	Download Successful	Yes
TC8	User interacts with the AQI Chatbot and asks for AQI info	Chatbot provides relevant and accurate AQI information	Response Received	Yes
TC9	User asks a non-AQI related question to the chatbot	Chatbot returns a message indicating it cannot answer	Unsupported Query	Yes

TC10	User views the homepage and checks responsiveness	Homepage elements adjust dynamically based on device size	Responsive Display	Yes
TC11	User interacts with the city selection carousel	Carousel displays city options correctly, users can select a city	Carousel Functionality	Yes
TC12	User sorts historical AQI data in ascending order	Table is sorted correctly based on AQI value	Sorted Table	Yes
TC13	User interacts with the interactive AQI graph	Graph displays AQI data trends, allows zooming, and toggles pollutants	Graph Interaction Successful	Yes

5.4 Admin Interface Test Cases

Test Case No	Test Case Description	Expected Result	Test Case Result	Passed (Yes/No)
TC1	Admin adds a new city with valid AQI data	City is successfully added and visible in the city list	City Added	Yes
TC2	Admin adds a new city with missing AQI data	Error message: Missing AQI data	Error Returned	Yes

TC3	Admin updates an existing city's AQI data	Valid API response: City AQI data updated	City Updated	Yes
TC4	Admin deletes a city from the city list	City is successfully removed from the system	City Deleted	Yes
TC5	Admin uploads new AQI data for a city	New AQI data is successfully uploaded and saved in the database	Data Uploaded	Yes
TC6	Admin views and downloads dashboard metrics	Dashboard metrics are displayed correctly, download options work	Metrics Displayed	Yes
TC7	Admin views city data in City Management page	City data is displayed in a sortable table with correct details	City Data Displayed	Yes
TC8	Admin deletes an API log by its ID	The API log is successfully deleted from the system	Log Deleted	Yes
TC9	Admin views all unresolved queries in Query Management	List of unresolved queries is displayed, sorted by status	Queries Displayed	Yes
TC10	Admin resolves a query and marks it as resolved	Query status is updated and moved to the resolved section	Query Resolved	Yes
TC11	Admin accesses the	Resolved queries are archived and	Query Archived	Yes

	query management page and resolves queries	removed from unresolved list		
TC12	Admin views and monitors API logs	API logs are displayed, showing all events and logs correctly	Logs Displayed	Yes
TC13	Admin accesses and views a specific API log by ID	Log data for the specified ID is shown correctly	Log Retrieved	Yes
TC14	Admin edits city information and saves the changes	Changes are saved and displayed correctly in the city list	Data Saved	Yes
TC15	Admin resolves a query and sends a confirmation email	Email is sent successfully to the user with query resolution	Email Sent	Yes

These test cases are designed to thoroughly check the functionality of both the user interface and Admin Interface, ensuring that every workflow and the feature works as expected.

5.5 Unit Testing

Unit tests verify that individual methods and components perform as expected. In this phase, we focus on:

- Backend Unit Tests: Using tools like JUnit and Mockito to test service methods, controllers, and repositories.
- Frontend Unit Tests: Using tools like Jest and React Testing Library to test React components, ensuring they render correctly and respond to user interactions.

5.6 Integration Testing

Integration testing ensures that multiple components work together as expected. Key integrations tested include:

- Backend API Endpoints: Verifying the functionality of API endpoints (e.g., fetching AQI data, submitting contact queries, etc.) using tools like Postman or integration tests in Spring Boot.
- Frontend-Backend Integration: Ensuring that frontend React components correctly fetch data from backend APIs and display it to users.

5.7 Regression Testing

After each update, regression testing ensures that previously working features continue functioning as expected, especially after bug fixes or enhancements.

- Frontend Regression: Ensures the UI components, such as AQI data display, graphs, and chatbots, work after latest changes.
- Backend Regression: Ensures API endpoints and services, such as AQI data fetching and user query handling, continue to work after updates.

5.7User Acceptance Testing (UAT)

UAT ensures that the system meets user expectations and business requirements. End-users test the following scenarios:

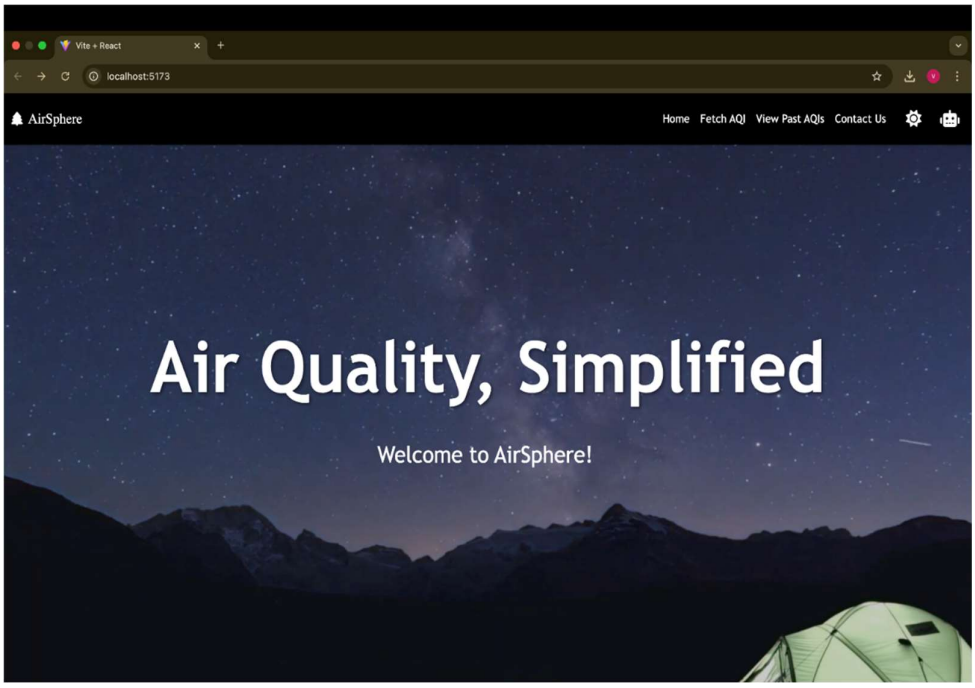
- Testing Real-World Use Cases: Simulating scenarios where users request AQI data, view historical data, interact with the chatbot, and submit queries via the contact form.

- User Feedback: Gathering feedback from real users regarding the usability and accessibility of the application to ensure that it is intuitive and user-friendly.

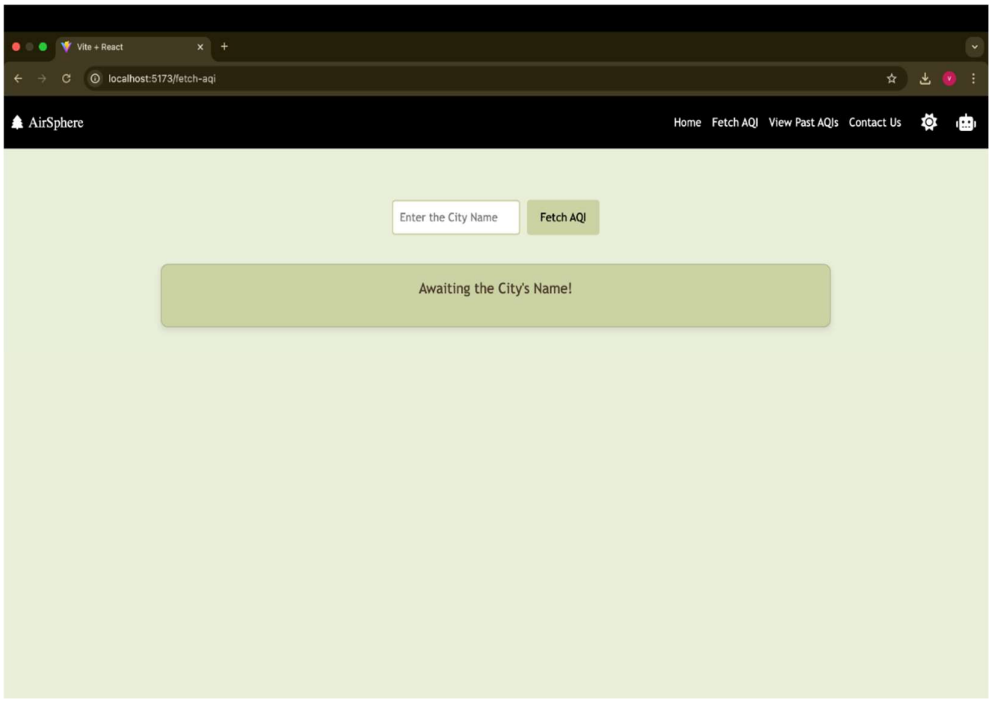
Testing for the AirSphere project covers all critical aspects, from unit testing individual components to ensuring end-to-end functionality through integration and UAT. The application has passed rigorous testing, ensuring it meets functional, performance, and usability standards for users and administrators.

CHAPTER 6: SNAPSHOTS

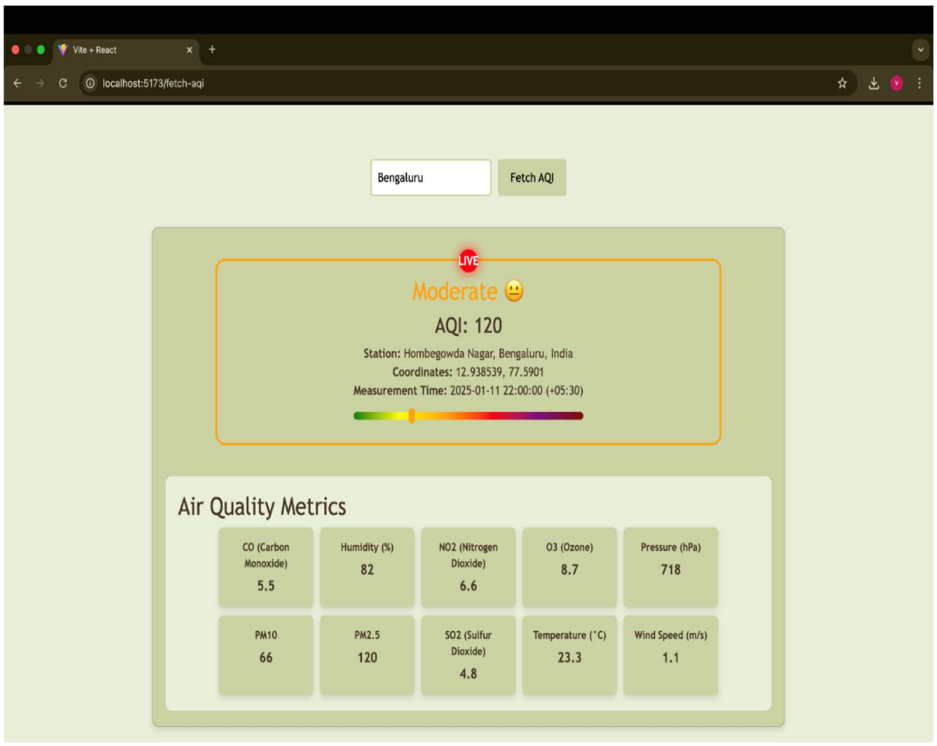
6.1 User Slides



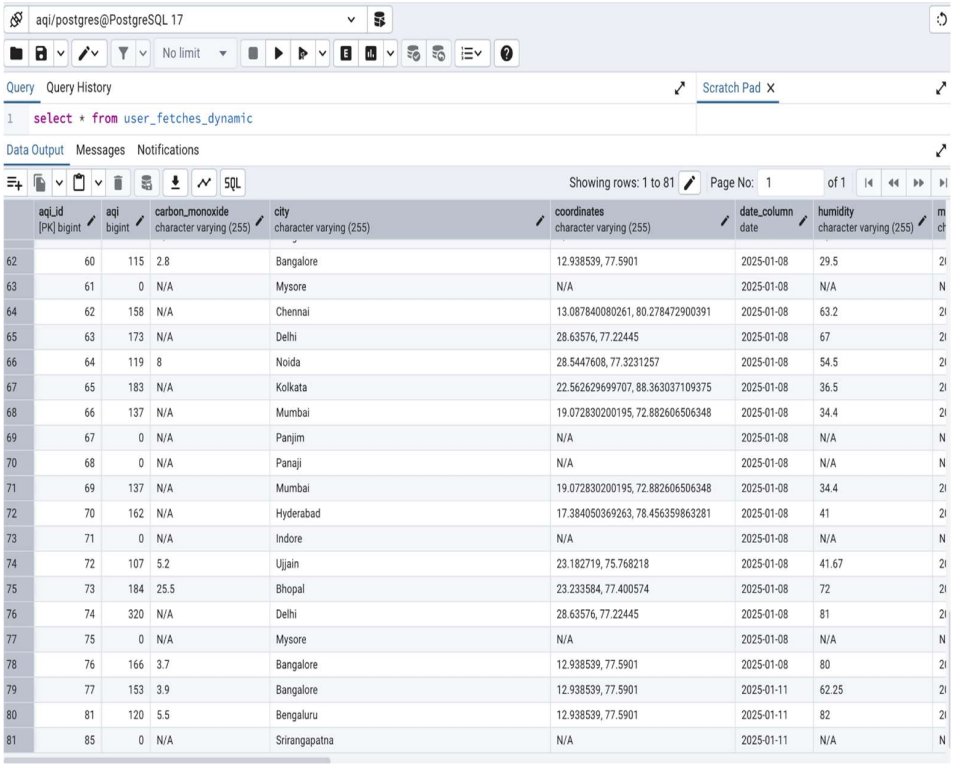
I. Landing Page



II. Fetch Live AQI page

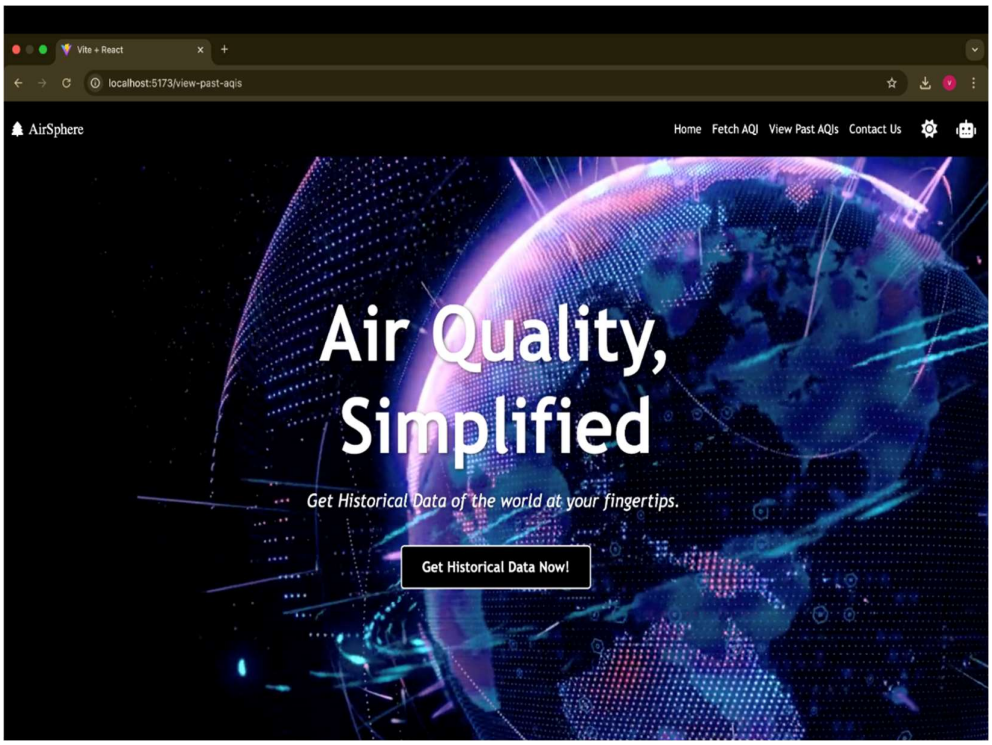


III.When WAQI returns a valid response, with available city data

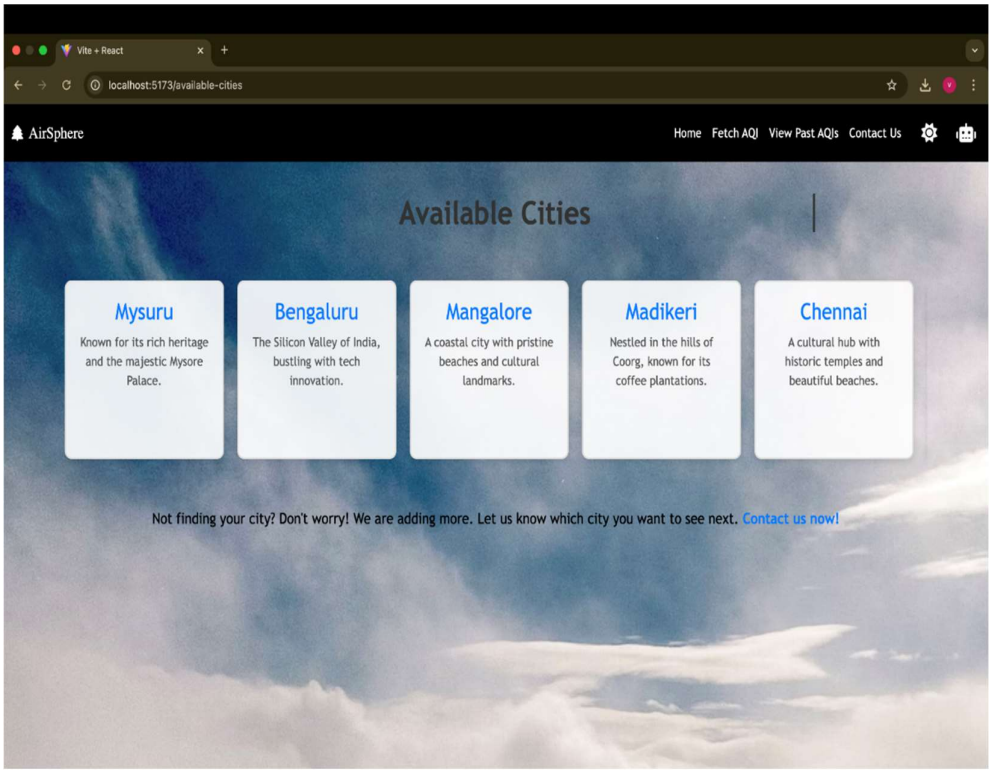


	aqi_id [PK] bigint	aqi bigint	carbon_monoxide character varying (255)	city character varying (255)	coordinates character varying (255)	date_column date	humidity character varying (255)	m ch
62	60	115	2.8	Bangalore	12.938539, 77.5901	2025-01-08	29.5	2(
63	61	0	N/A	Mysore	N/A	2025-01-08	N/A	N
64	62	158	N/A	Chennai	13.087840080261, 80.278472900391	2025-01-08	63.2	2(
65	63	173	N/A	Delhi	28.63576, 77.22445	2025-01-08	67	2(
66	64	119	8	Noida	28.5447608, 77.3231257	2025-01-08	54.5	2(
67	65	183	N/A	Kolkata	22.562629699707, 88.363037109375	2025-01-08	36.5	2(
68	66	137	N/A	Mumbai	19.072830200195, 72.882606506348	2025-01-08	34.4	2(
69	67	0	N/A	Panjim	N/A	2025-01-08	N/A	N
70	68	0	N/A	Panaji	N/A	2025-01-08	N/A	N
71	69	137	N/A	Mumbai	19.072830200195, 72.882606506348	2025-01-08	34.4	2(
72	70	162	N/A	Hyderabad	17.384050369263, 78.456359863281	2025-01-08	41	2(
73	71	0	N/A	Indore	N/A	2025-01-08	N/A	N
74	72	107	5.2	Ujjain	23.182719, 75.768218	2025-01-08	41.67	2(
75	73	184	25.5	Bhopal	23.233584, 77.400574	2025-01-08	72	2(
76	74	320	N/A	Delhi	28.63576, 77.22445	2025-01-08	81	2(
77	75	0	N/A	Mysore	N/A	2025-01-08	N/A	N
78	76	166	3.7	Bangalore	12.938539, 77.5901	2025-01-08	80	2(
79	77	153	3.9	Bangalore	12.938539, 77.5901	2025-01-11	62.25	2(
80	81	120	5.5	Bengaluru	12.938539, 77.5901	2025-01-11	82	2(
81	85	0	N/A	Srirangapatna	N/A	2025-01-11	N/A	N

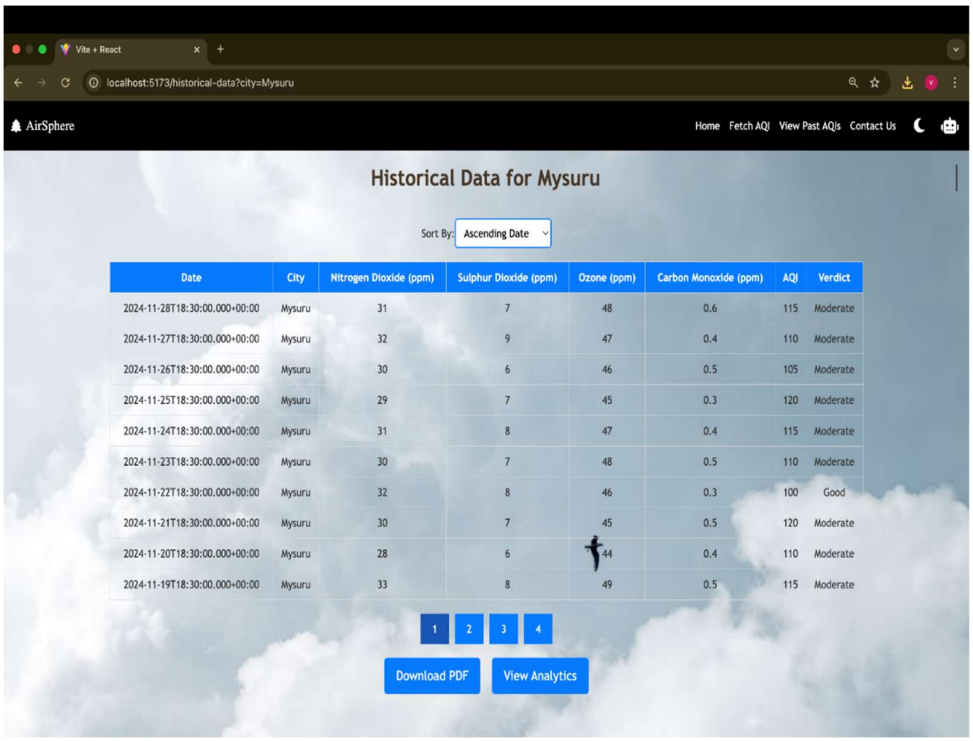
IV. Fetches by the user recorded dynamically in the database entries



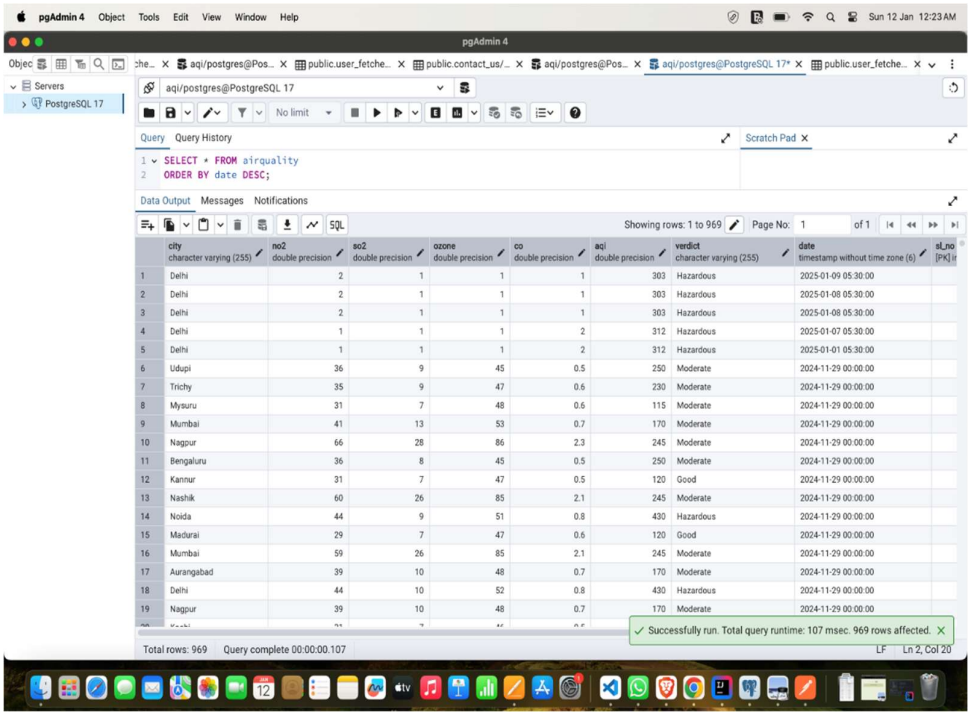
V. View Historical AQI Data



VI. View available cities with a stunning carousel.

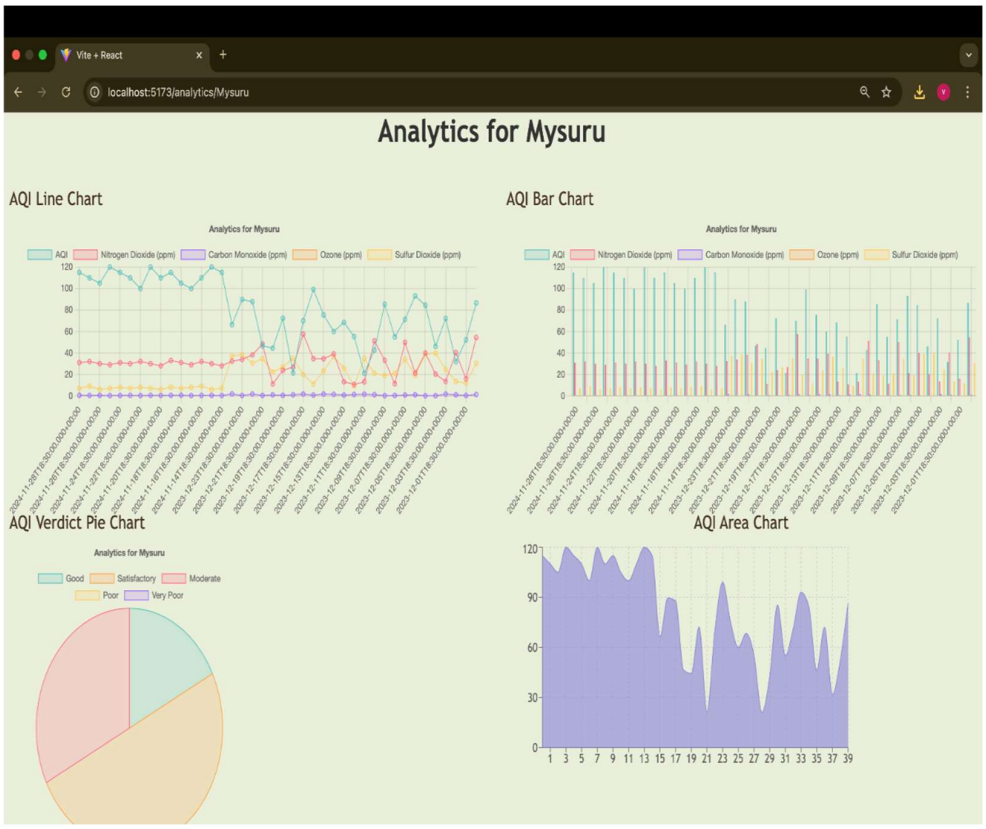


VII. View Historical Data with pagination options, Download PDF, and View Analytics Options. Sort the data as most polluted, least polluted, ascending date, descending date.

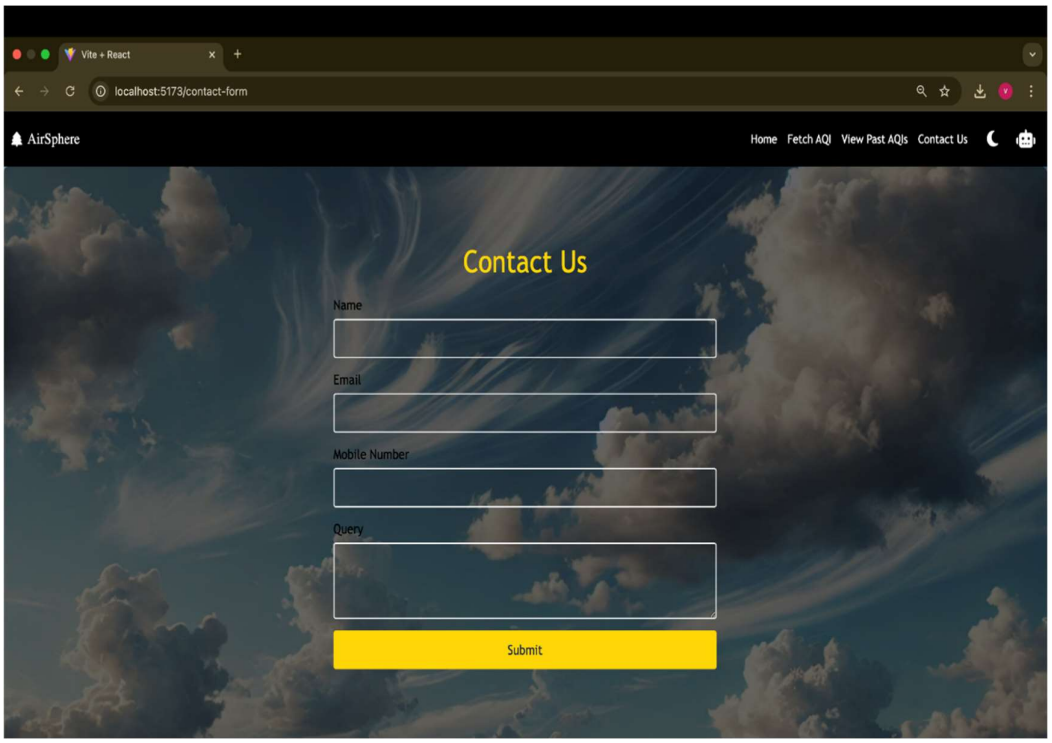


	city	no2	so2	ozone	co	aqi	verdict	date	sl_no
1	Delhi	2	1	1	1	303	Hazardous	2025-01-09 05:30:00	
2	Delhi	2	1	1	1	303	Hazardous	2025-01-08 05:30:00	
3	Delhi	2	1	1	1	303	Hazardous	2025-01-08 05:30:00	
4	Delhi	1	1	1	2	312	Hazardous	2025-01-07 05:30:00	
5	Delhi	1	1	1	2	312	Hazardous	2025-01-01 05:30:00	
6	Udupi	36	9	45	0.5	250	Moderate	2024-11-29 00:00:00	
7	Trichy	35	9	47	0.6	230	Moderate	2024-11-29 00:00:00	
8	Mysuru	31	7	48	0.6	115	Moderate	2024-11-29 00:00:00	
9	Mumbai	41	13	53	0.7	170	Moderate	2024-11-29 00:00:00	
10	Nagpur	66	28	86	2.3	245	Moderate	2024-11-29 00:00:00	
11	Bengaluru	36	8	45	0.5	250	Moderate	2024-11-29 00:00:00	
12	Kannur	31	7	47	0.5	120	Good	2024-11-29 00:00:00	
13	Nashik	60	26	85	2.1	245	Moderate	2024-11-29 00:00:00	
14	Noida	44	9	51	0.8	430	Hazardous	2024-11-29 00:00:00	
15	Madurai	29	7	47	0.6	120	Good	2024-11-29 00:00:00	
16	Mumbai	59	26	85	2.1	245	Moderate	2024-11-29 00:00:00	
17	Aurangabad	39	10	48	0.7	170	Moderate	2024-11-29 00:00:00	
18	Delhi	44	10	52	0.8	430	Hazardous	2024-11-29 00:00:00	
19	Nagpur	39	10	48	0.7	170	Moderate	2024-11-29 00:00:00	

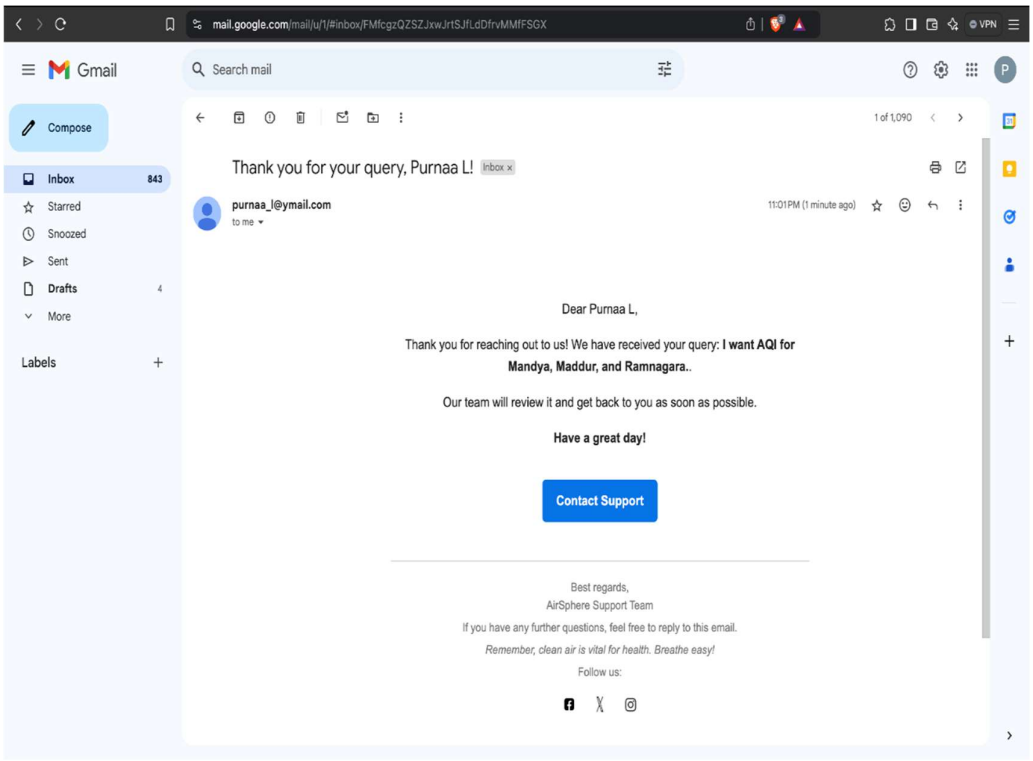
VIII. Vast amount of Historical Data (969 entries) is stored in the database. This is dynamically fetched by Spring Boot, the data is filtered based on the city selected by the user and displayed in the front end.



IX. Dynamic Analysis of the Data with interactive graphs and charts, using recharts.js

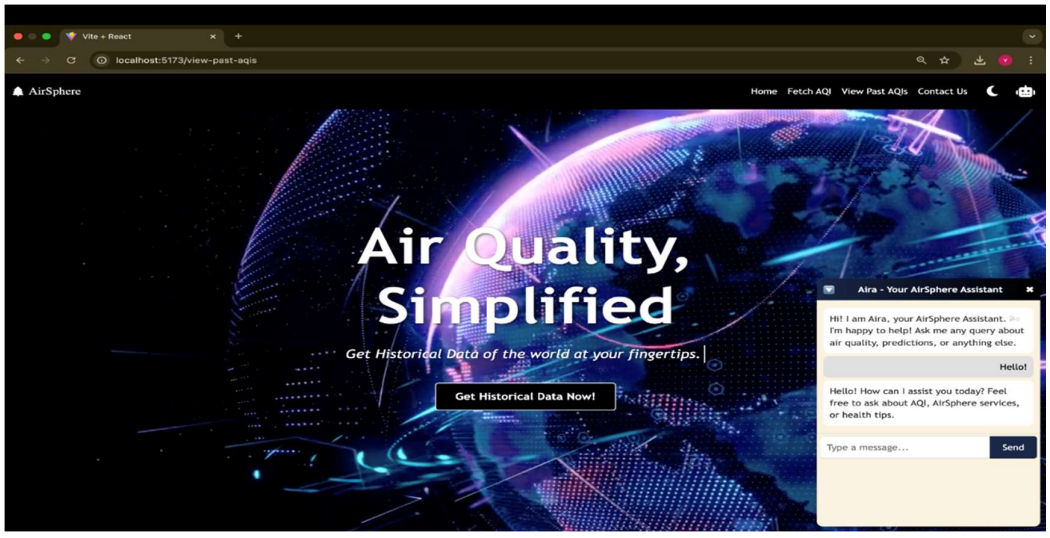


X. Contact Us Form

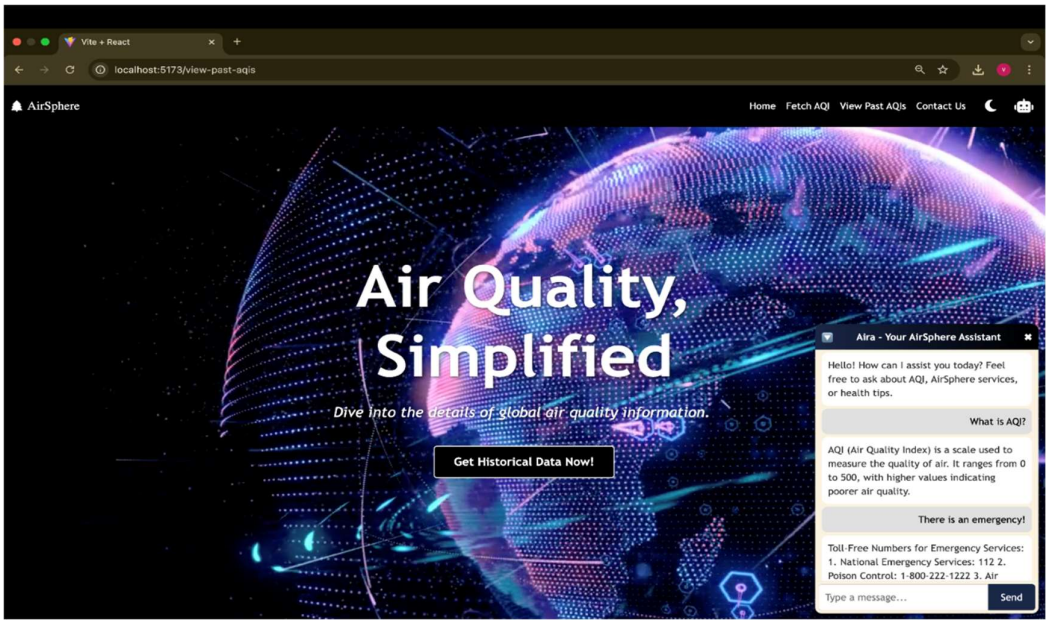


XI. Automatic Mail Sender by Spring Email using SMTP services.

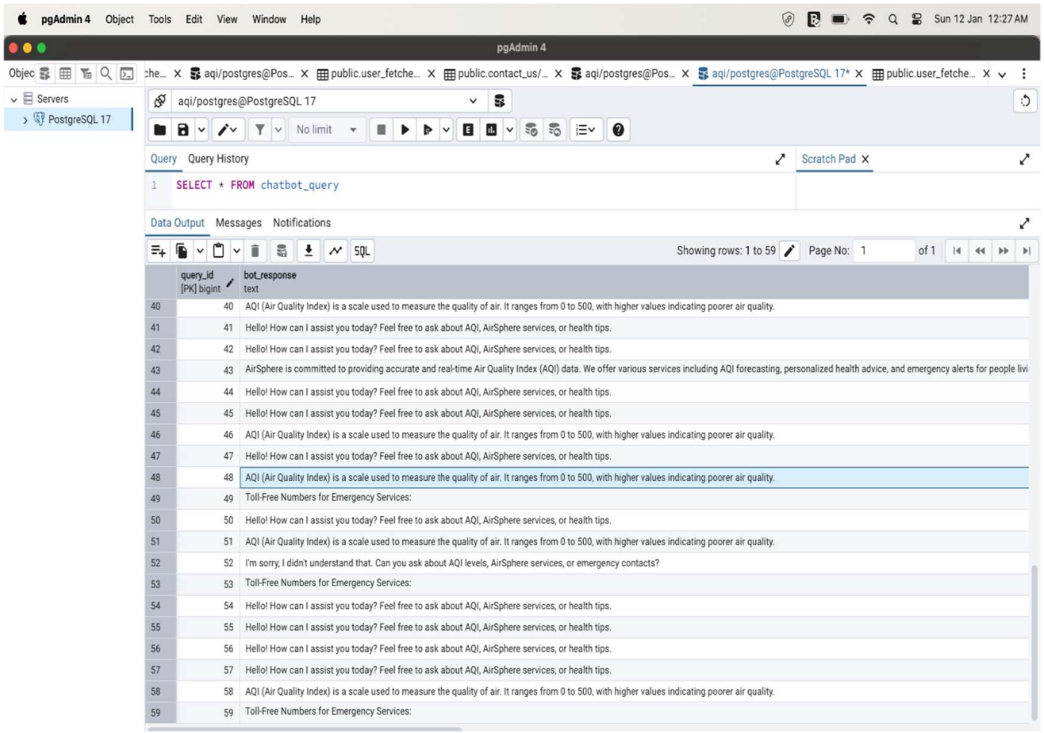
XII. Contact Us table dynamically updated as and when the user submits query



XIII. Custom Chatbot integration

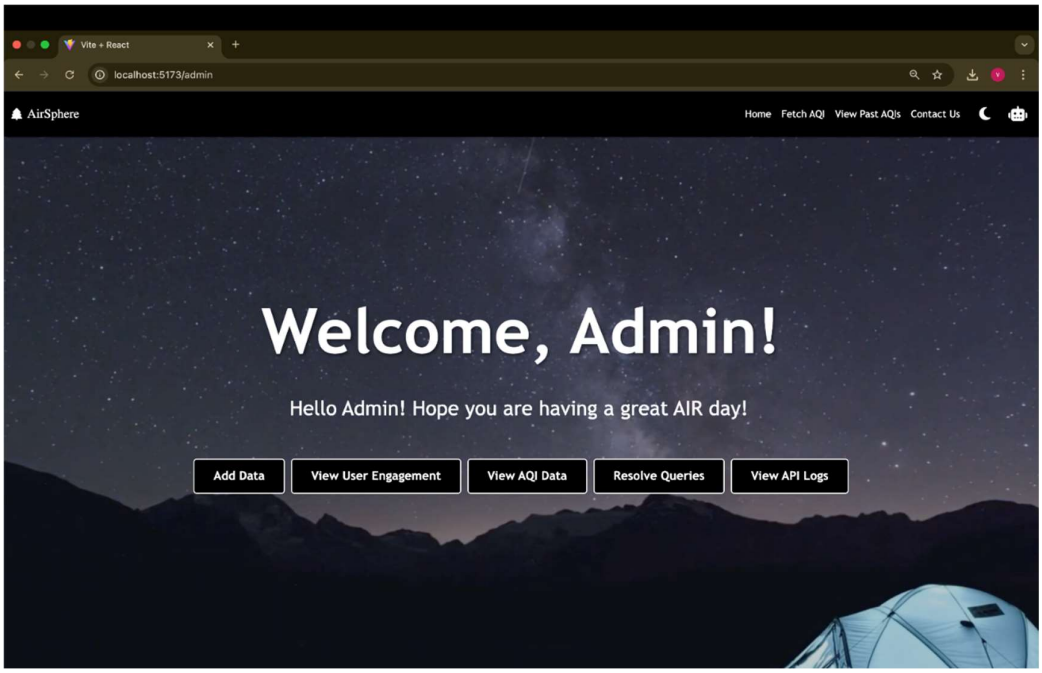


XIV. Custom Chatbot answering queries and FAQs.

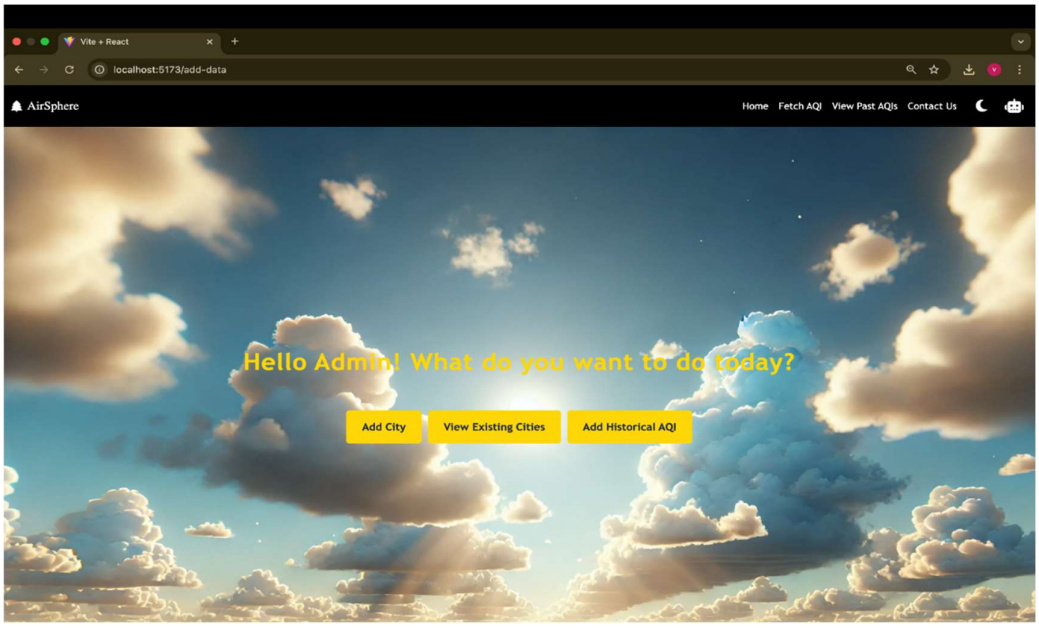


XV. Chatbot query table dynamically updated based on the queries asked by the user in the chatbot. Denoted here by entries of query_id 57, 58, 59

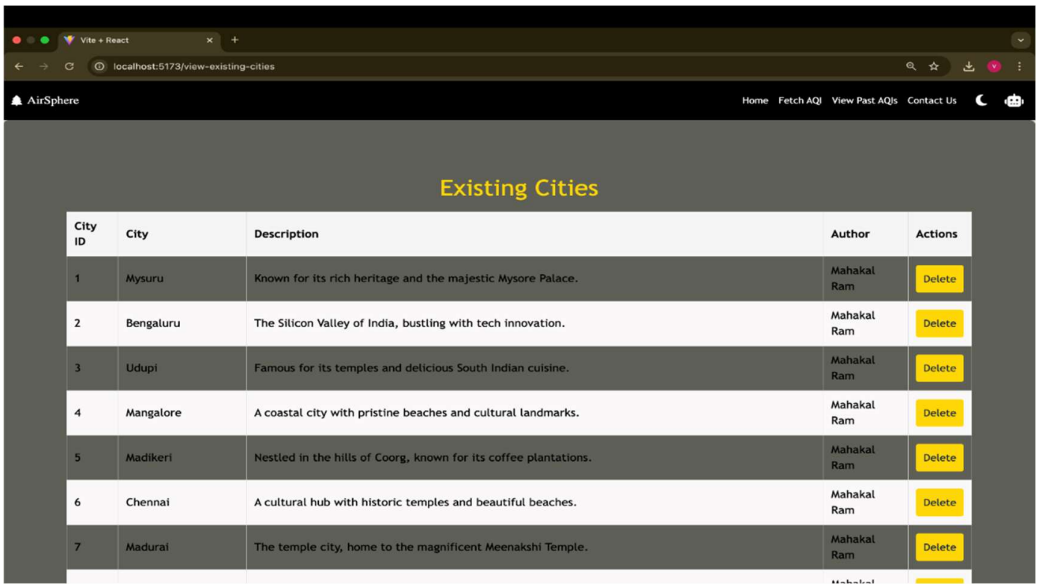
6.2 Admin Slides



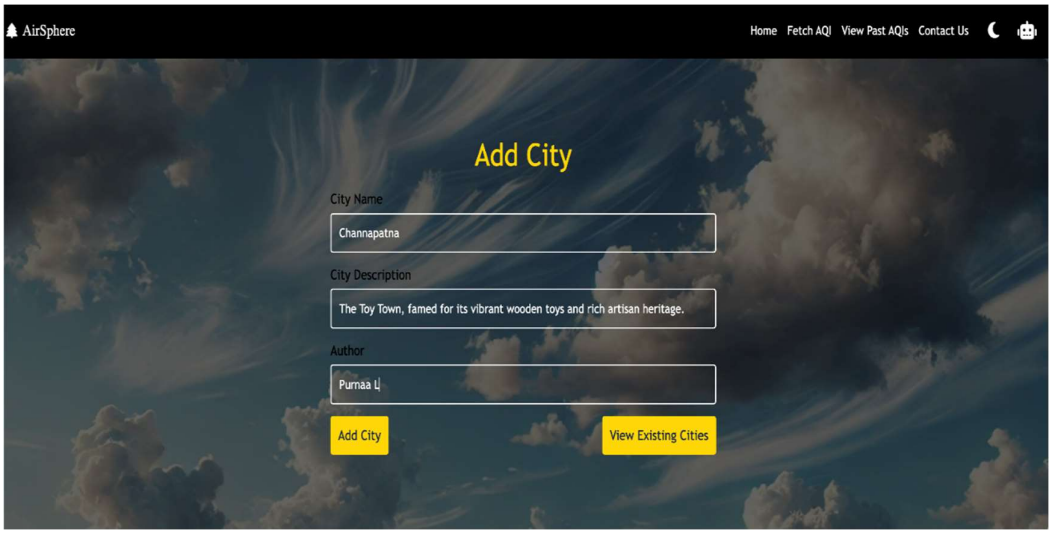
I. Admin Landing Page



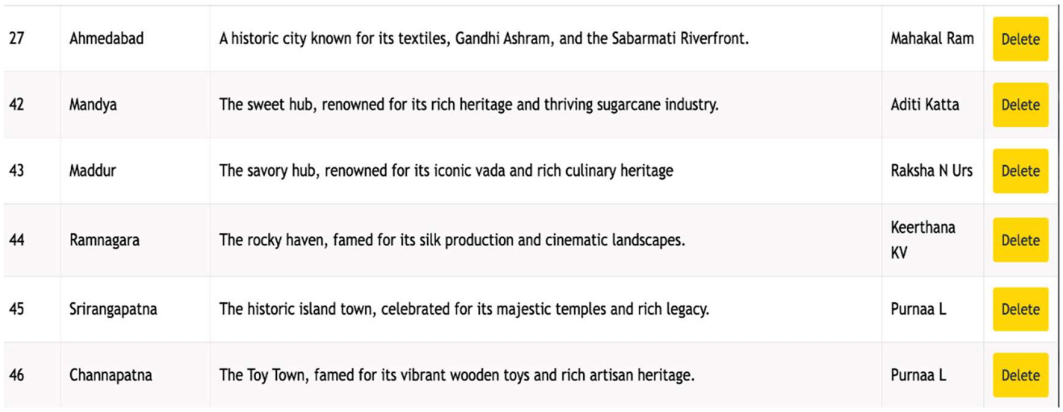
II. Add Data Option



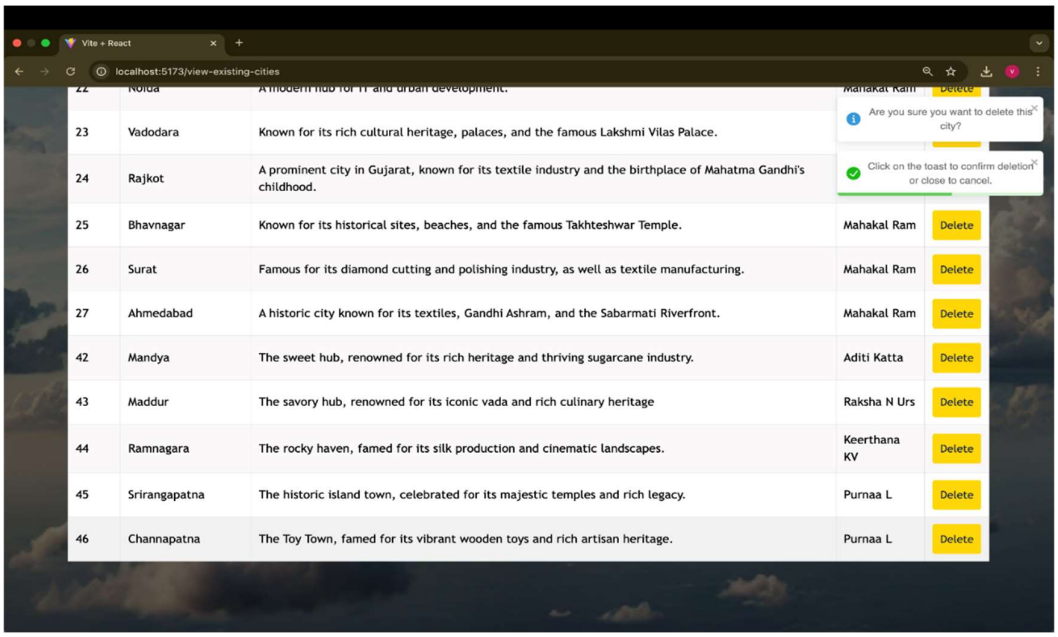
III. View Existing Cities



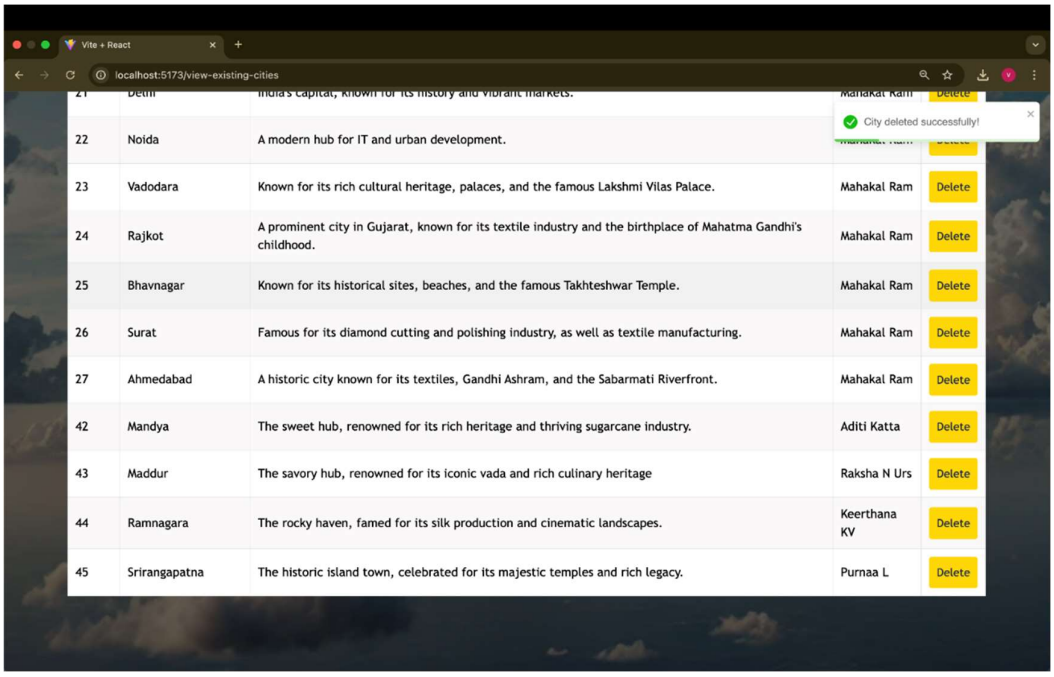
IV. Add City Option



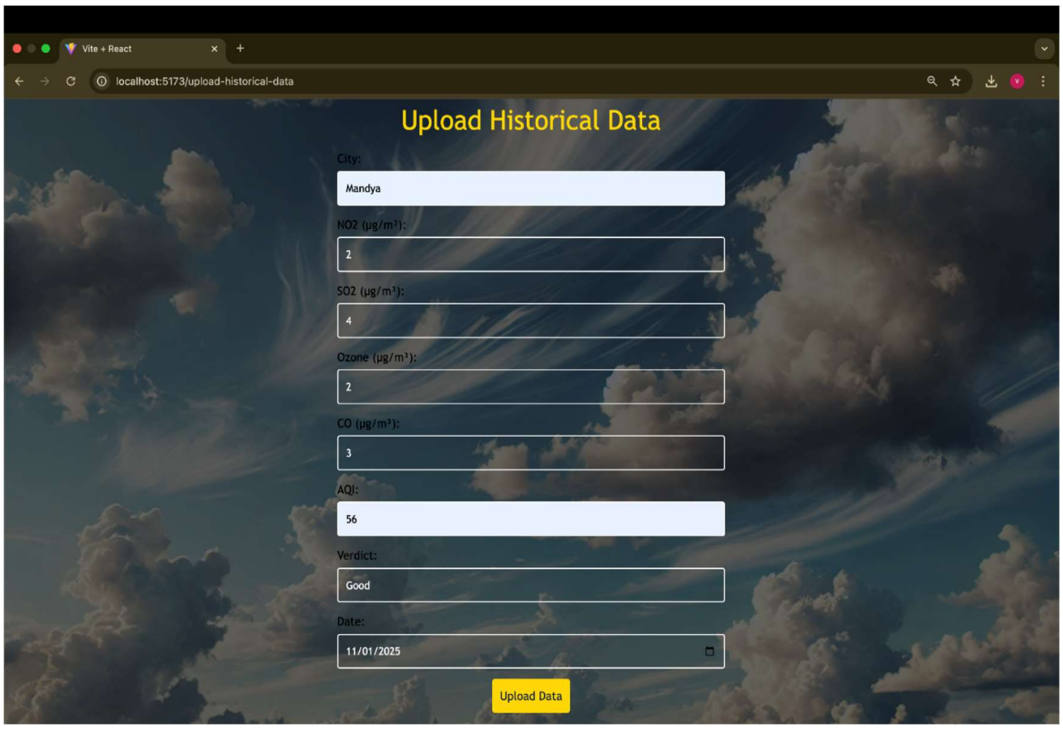
V. Updated City List



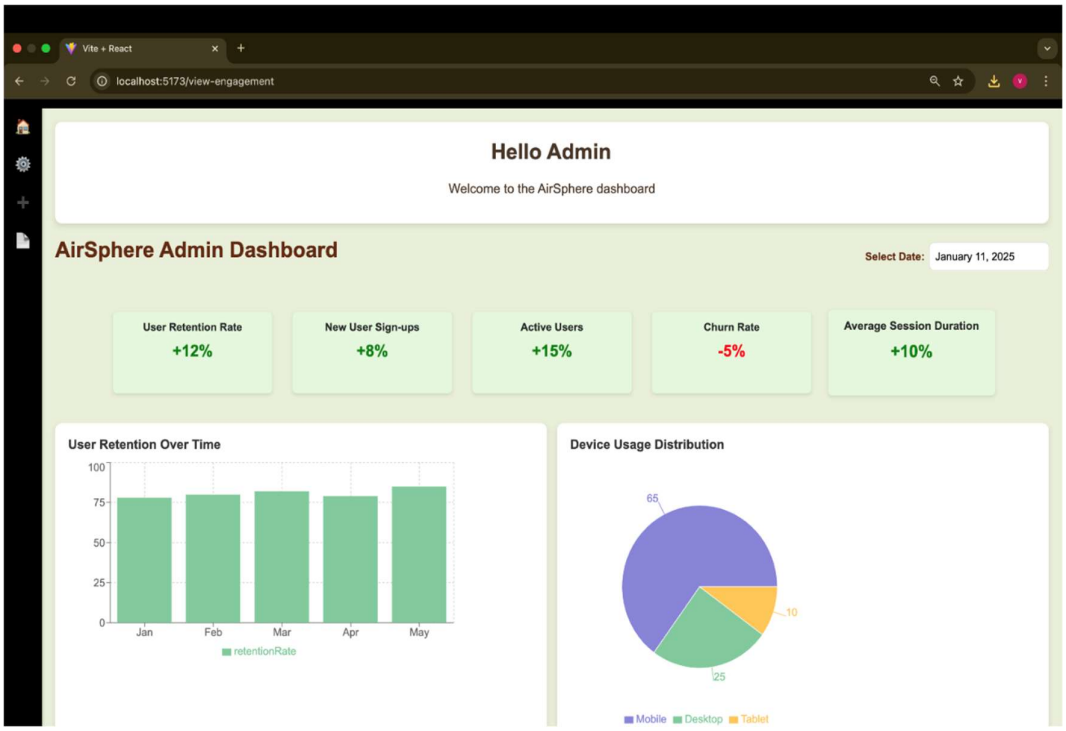
VI. Prompting the Delete action on ’Channapatna’ city



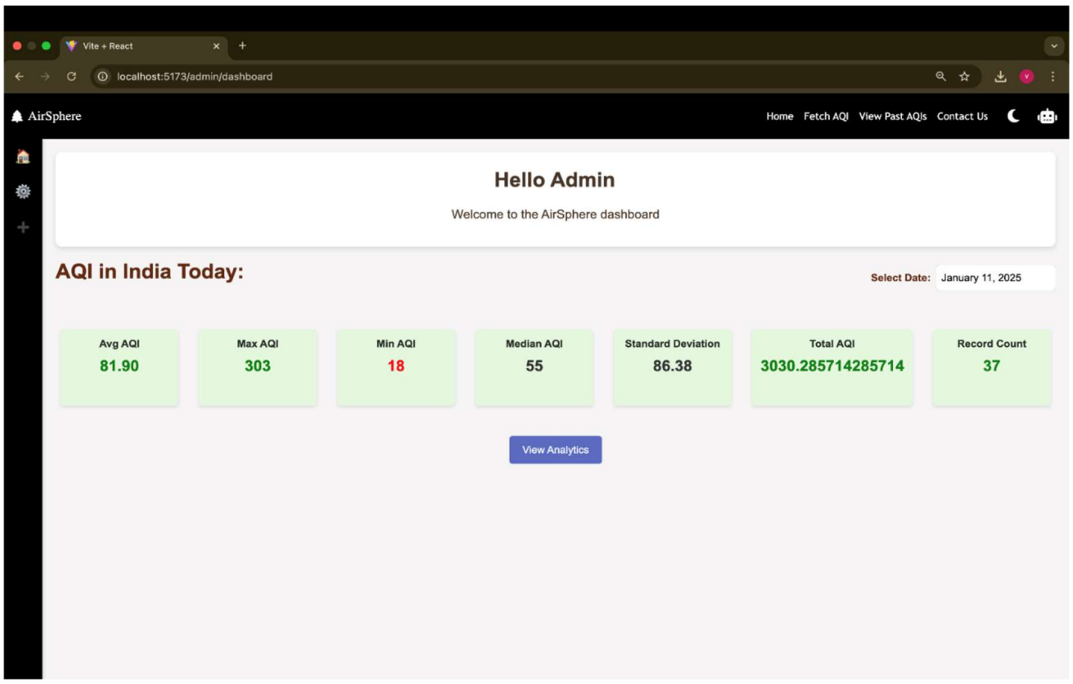
VII. Successful Deletion of the city



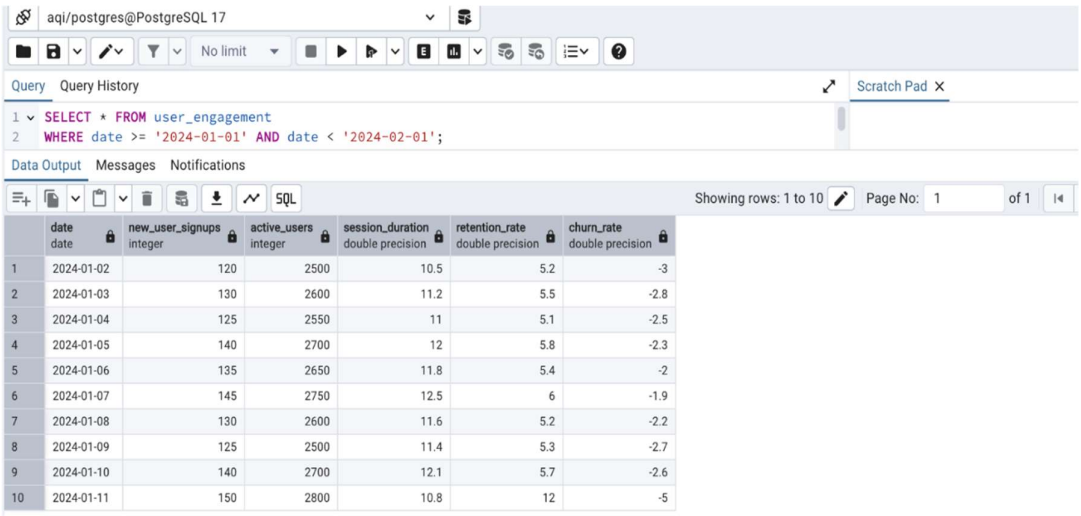
VIII. Upload Historical Data



IX. User Engagement Analytics

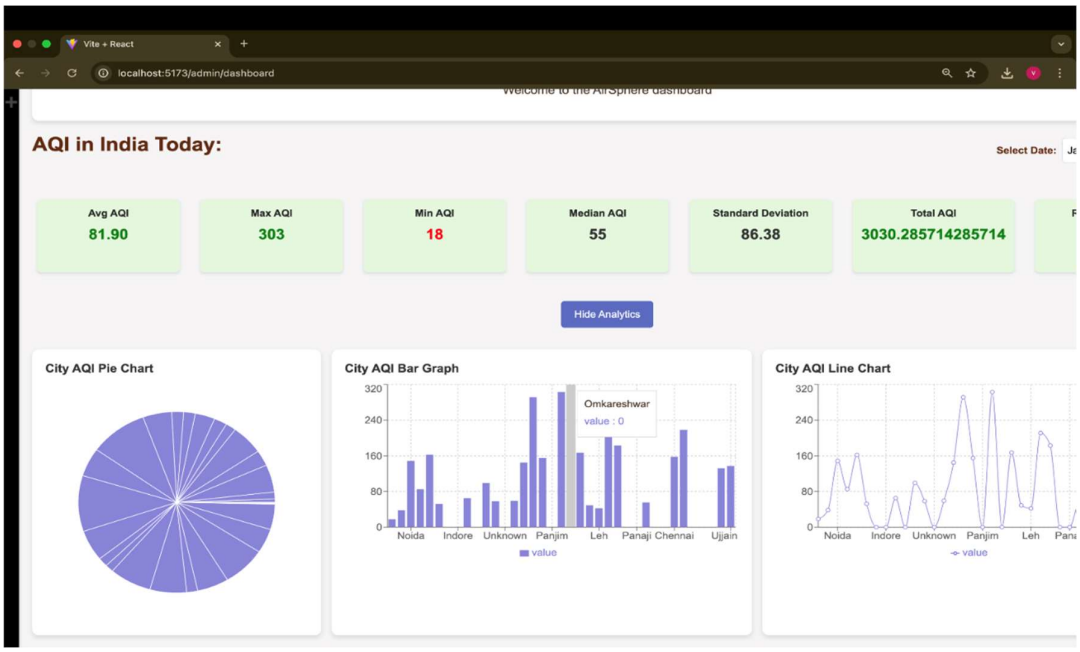


X. Live AQI Analytics for the selected date

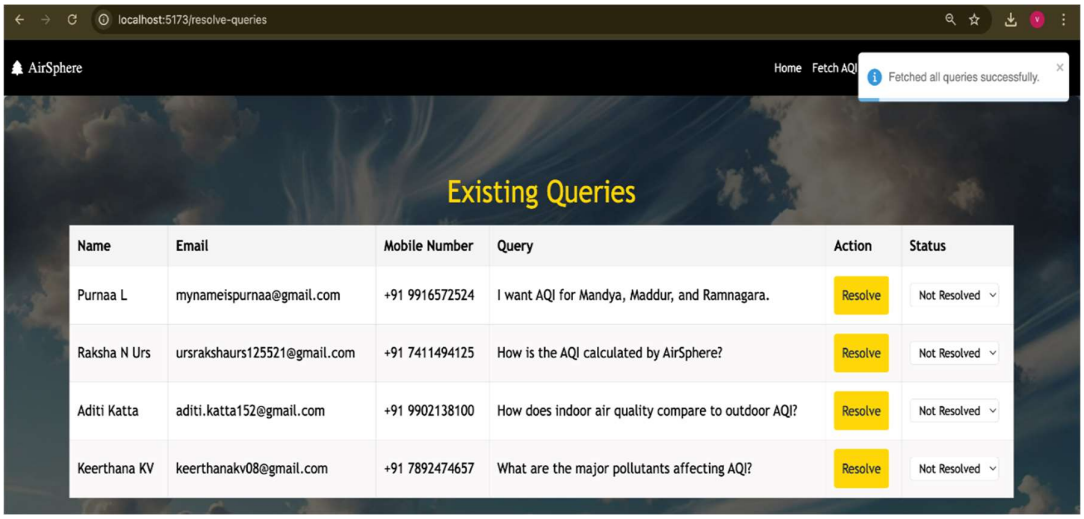


	date date	new_user_signups integer	active_users integer	session_duration double precision	retention_rate double precision	churn_rate double precision
1	2024-01-02	120	2500	10.5	5.2	-3
2	2024-01-03	130	2600	11.2	5.5	-2.8
3	2024-01-04	125	2550	11	5.1	-2.5
4	2024-01-05	140	2700	12	5.8	-2.3
5	2024-01-06	135	2650	11.8	5.4	-2
6	2024-01-07	145	2750	12.5	6	-1.9
7	2024-01-08	130	2600	11.6	5.2	-2.2
8	2024-01-09	125	2500	11.4	5.3	-2.7
9	2024-01-10	140	2700	12.1	5.7	-2.6
10	2024-01-11	150	2800	10.8	12	-5

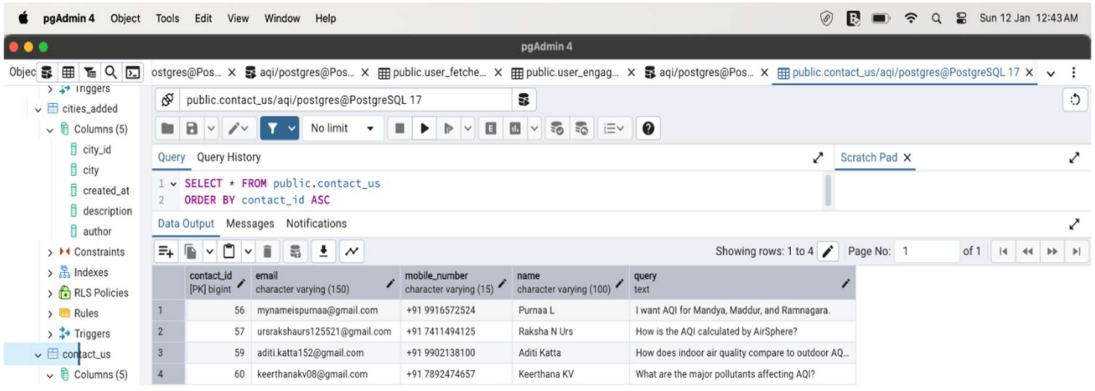
XI. User Engagement data in the database. Fetched dynamically and displayed in the front end based on the selected date.



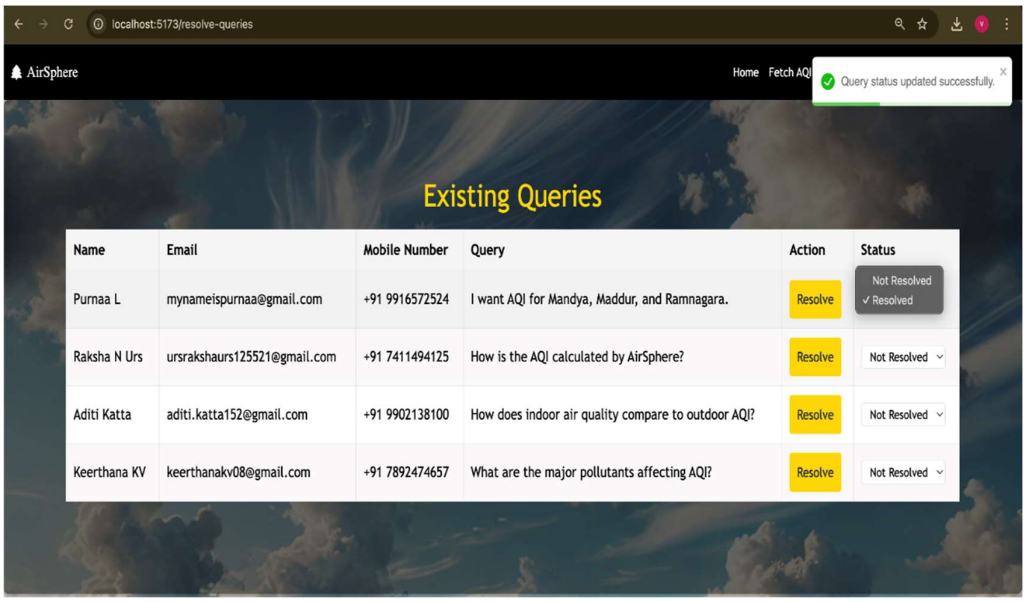
XII. Live Graphical Analysis using Interactive charts



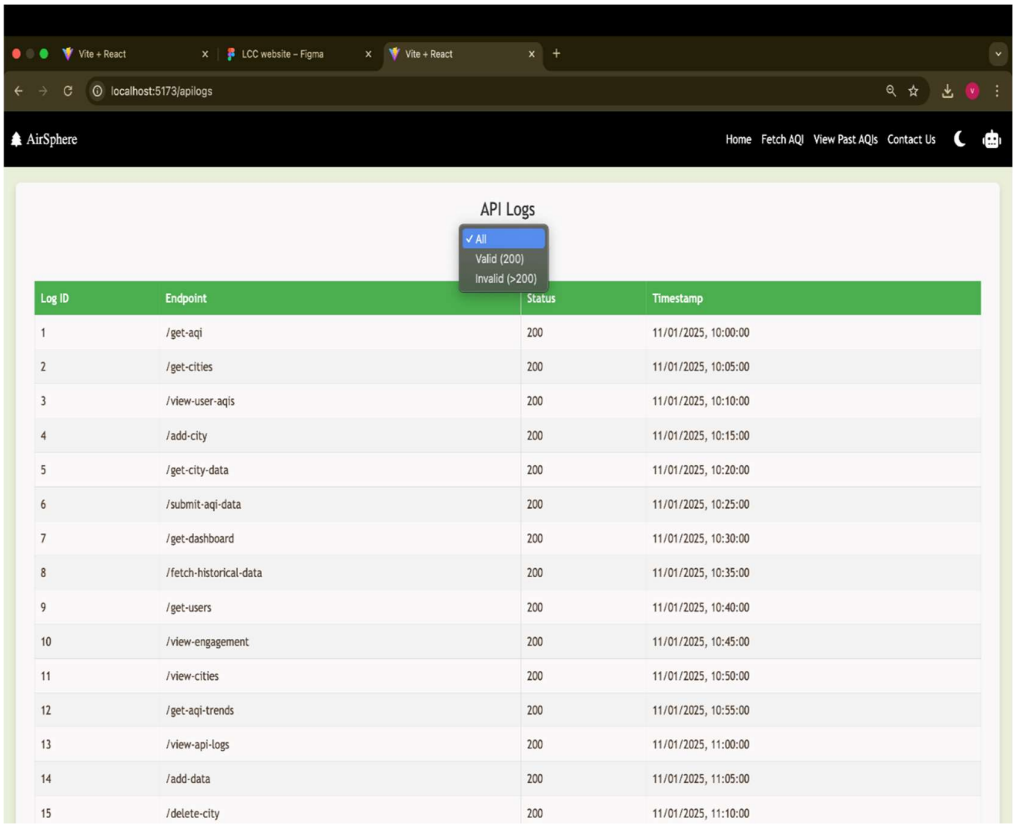
XIII. View Existing Queries



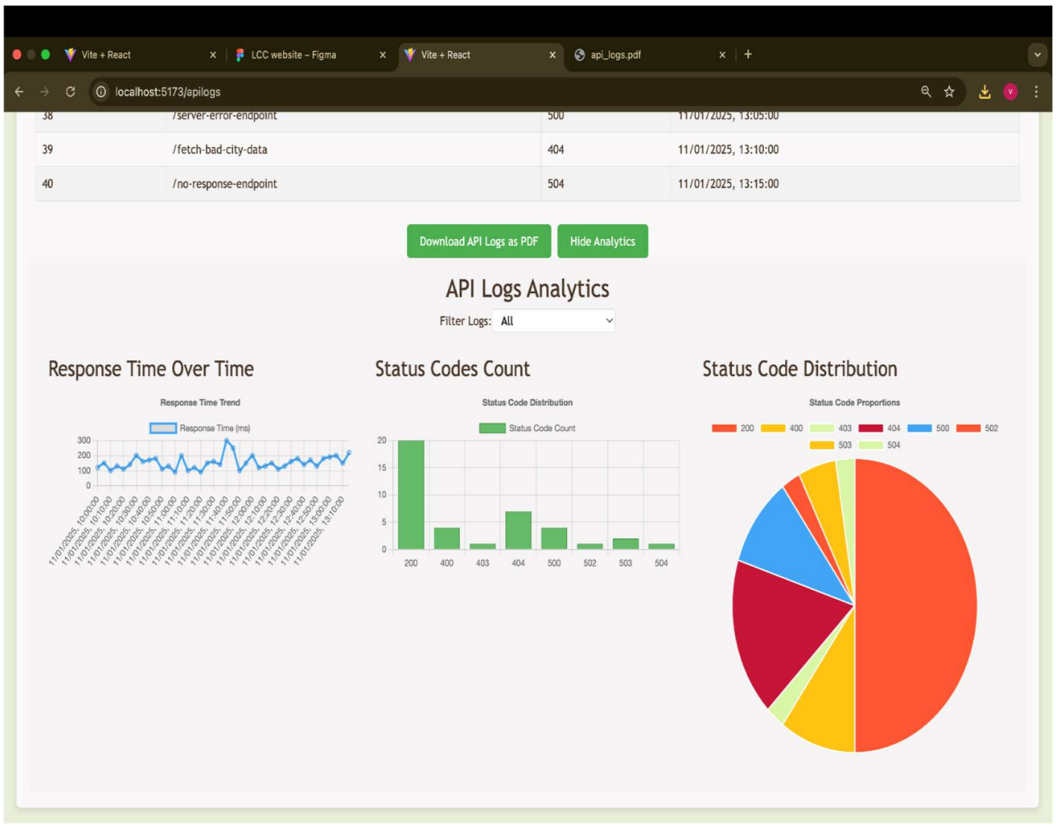
XIV. Queries updated in the database as and when a user submits a query



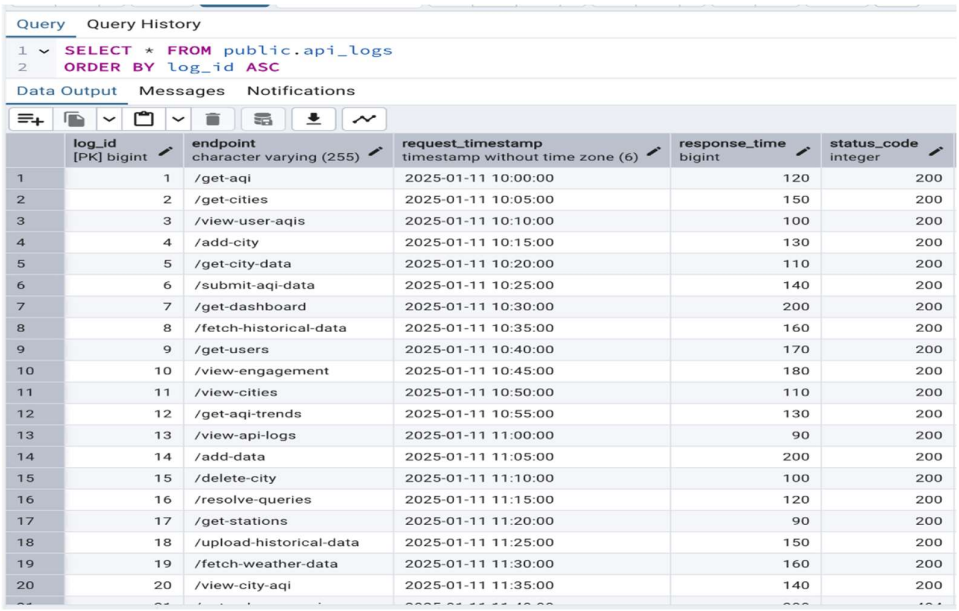
XV. Dynamically Update Queries as resolved/not resolved.



XVI. View API Logs with filter options for valid/invalid responses



XVII. View Analytics Option



Query Query History

```
SELECT * FROM public.api_logs
ORDER BY log_id ASC
```

Data Output Messages Notifications

	log_id [PK] bigint	endpoint character varying (255)	request_timestamp timestamp without time zone (6)	response_time bigint	status_code integer
1	1	/get-aqi	2025-01-11 10:00:00	120	200
2	2	/get-cities	2025-01-11 10:05:00	150	200
3	3	/view-user-aqis	2025-01-11 10:10:00	100	200
4	4	/add-city	2025-01-11 10:15:00	130	200
5	5	/get-city-data	2025-01-11 10:20:00	110	200
6	6	/submit-aqi-data	2025-01-11 10:25:00	140	200
7	7	/get-dashboard	2025-01-11 10:30:00	200	200
8	8	/fetch-historical-data	2025-01-11 10:35:00	160	200
9	9	/get-users	2025-01-11 10:40:00	170	200
10	10	/view-engagement	2025-01-11 10:45:00	180	200
11	11	/view-cities	2025-01-11 10:50:00	110	200
12	12	/get-aqi-trends	2025-01-11 10:55:00	130	200
13	13	/view-api-logs	2025-01-11 11:00:00	90	200
14	14	/add-data	2025-01-11 11:05:00	200	200
15	15	/delete-city	2025-01-11 11:10:00	100	200
16	16	/resolve-queries	2025-01-11 11:15:00	120	200
17	17	/get-stations	2025-01-11 11:20:00	90	200
18	18	/upload-historical-data	2025-01-11 11:25:00	150	200
19	19	/fetch-weather-data	2025-01-11 11:30:00	160	200
20	20	/view-city-aqi	2025-01-11 11:35:00	140	200

XVIII. Database consisting of the API logs. Dynamically updated, fetched, and elegantly displayed in the front end

CHAPTER 7: CONCLUSION

AirSphere represents a significant advancement in monitoring and interacting with air quality data. By integrating powerful real-time APIs and a well-designed user interface, AirSphere enables users to access up-to-date AQI (Air Quality Index) information for cities worldwide. This platform is not just about providing information; it empowers individuals, organizations, and governmental bodies to make informed decisions based on accurate and current air quality data. The system's architecture ensures seamless communication between the front and back end, leveraging advanced technologies such as Spring Boot and ReactJS. The backend handles processing AQI data from external APIs, storing historical data, and performing analytics. At the same time, the front end offers an interactive experience with dynamic charts, graphs, and easy-to-use forms. This dual real-time data display and historical analysis approach benefits users who want to track air quality trends, compare data over time, or be informed about the air they breathe.

One of the standouts features of AirSphere is the Chatbot integration, which acts as a virtual assistant to help users with common queries related to air quality. Whether a user is concerned about specific pollutant levels or requires tips for improving air quality, the chatbot provides instant responses, enhancing user engagement and making the platform more accessible.

AirSphere also serves a crucial role for administrators. Through the administrative dashboard, city management, and user engagement monitoring, administrators can ensure the smooth operation of the platform. They can add new cities, update AQI metrics, resolve user queries, and generate reports. This makes AirSphere a highly adaptable tool, capable of supporting diverse needs, from individual users to large-scale urban management.

The platform's ability to track user engagement—such as new sign-ups, active users, session durations, and retention rates—gives administrators valuable insights into how the platform is utilized. This data can be used to refine user experiences, optimize platform features, and better tailor communications to the needs of different user groups.

AirSphere is designed to be scalable, flexible, and accessible. Its responsive design ensures that users can access AQI information easily across devices—whether on desktop or mobile—making it an ideal solution for today's fast-paced, mobile-first world. The emphasis on accessibility means that users from different regions with diverse needs can easily interact with the platform.

In conclusion, AirSphere is more than just an air quality monitoring tool; it is a comprehensive platform that educates, engages, and empowers individuals and organizations to take meaningful actions towards improving air quality and public health. With its intuitive user interface, powerful backend analytics, and continuous improvements in real-time data processing, AirSphere plays a key role in promoting awareness, encouraging proactive environmental decisions, and contributing to a healthier planet.

CHAPTER 8: FUTURE ENHANCEMENTS

As AirSphere continues to evolve, several opportunities exist to enhance its functionality, improve user experience, and expand its impact. Below are some potential future enhancements that could be considered:

- **Real-time Air Quality Forecasting:** Integrate machine learning models to predict air quality trends and forecasts, providing users with current data and forecasts for the next 24-48 hours. This would allow users to plan their activities based on future air quality conditions, particularly in pollution-prone areas.
- **Mobile App Development:** While AirSphere is optimized for desktop and mobile web usage, developing a dedicated mobile app for iOS and Android would improve user experience, allow for push notifications on air quality updates, and facilitate location-based alerts for users on the go.
- **Integration with IoT Sensors:** By integrating with Internet of Things (IoT) sensors in cities, businesses, or homes, AirSphere could provide even more granular and real-time air quality data. This would allow users to monitor air quality in specific locations, such as their home or office, and receive personalized alerts.
- **Personalized Health Alerts:** Implement personalized health alerts for users with respiratory conditions such as asthma or COPD. Based on the user's preferences and health information (with their consent), AirSphere could send alerts when AQI levels reach harmful thresholds, advising them to take precautionary measures.
- **Community-based Reporting:** Enable users to contribute to the system by reporting air quality in their local areas. This feature could crowdsource data from sensors and user reports, improving the accuracy of air quality information in regions that may lack official monitoring stations.
- **Gamification and Challenges:** Introduce gamification elements to encourage users to take actions that reduce their carbon footprint or improve air quality, such as planting trees or reducing car usage. Users could earn points or badges for sustainable activities, fostering greater

environmental consciousness.

- **AI-Driven Chatbot Improvements:** While the current chatbot serves as a helpful virtual assistant, further enhancements could be made by implementing natural language processing (NLP) and machine learning to allow the chatbot to understand a broader range of queries and provide more specific and context-aware responses.
- **Enhanced Historical Data Analysis:** Implement more advanced visualization tools like heat maps, trend graphs, and detailed pollutant breakdowns. Users could track historical trends over extended periods, such as months or years, to gain deeper insights into long-term air quality patterns and their potential impact on health.
- **Multi-Language Support:** To further expand AirSphere's user base, introduce multi-language support for users across the globe. This will make the platform more accessible to non-English speaking populations and foster greater international adoption.

By incorporating these enhancements, AirSphere can continue to grow, providing richer, more personalized user experiences and contributing to a global effort to improve air quality and health outcomes. These future upgrades could significantly increase the platform's utility, engagement, and relevance in the evolving landscape of environmental monitoring.

CHAPTER 8: REFERENCES

WEBSITES

1.https://react.dev/

2.https://react.dev/reference/react/hooks

3.https://react.dev/reference/react-dom

4.https://spring.io/

5.https://www.postman.com/

6.https://waqi.info/

7.https://www.postgresql.org/docs/

8. https://docs.spring.io/spring-boot/index.html