AIR DATA MANAGEMENT SYSTEM

GROUP 77 FINAL PROJECT REPORT

UNIVERSITY OF SOUTHERN CALIFORNIA

Zihan Mai
Hengxuan Wang

TEAM MEMBERS

Sr No.	Name
1	Zihan Mai
2	Hengxuan Wang
Group	77

PROJECT DETAILS

Name	**Air Data Management System**
About	**Interactive Platform for Real-Time Air Quality Visualization and Management**
Youtube link	**https://youtu.be/fYsMYpnkXYo**
Google Drive Link	

INTRODUCTION

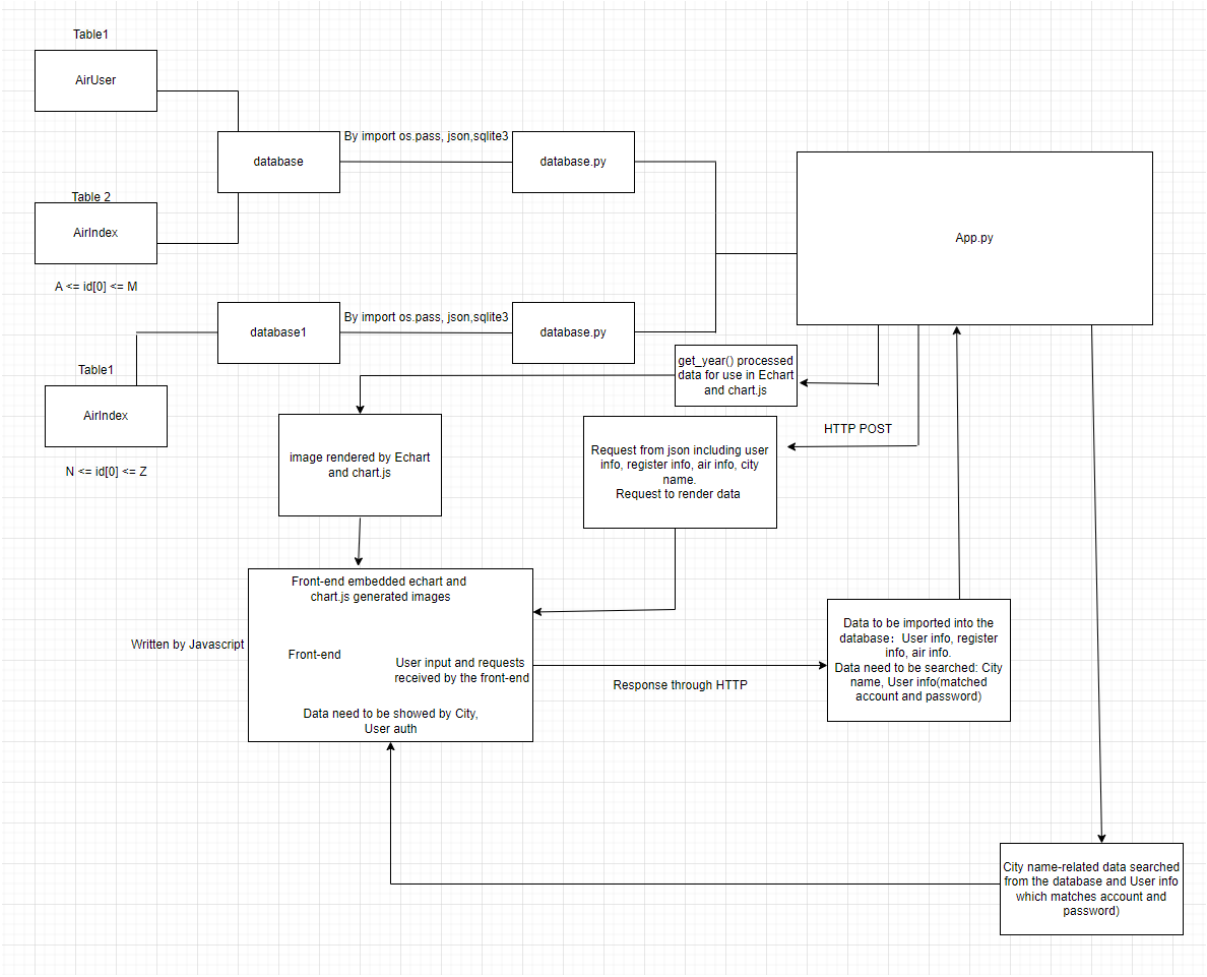
This project involves the development of a web-based administration panel designed to manage and display air quality data across various geographic locations. The primary objective is to provide a user-friendly interface that allows administrators to visualize trends, update and retrieve data, and manage user access effectively. To achieve this, we employed Vue.js, a progressive JavaScript framework known for its adaptability and component-based architecture, enabling a dynamic and responsive user experience. For data visualization, we use echart to make horizontal comparisons between cities in the same year, and We use chart js to conduct virtcal comparisons based on different years in the same city. The backend data management is handled through a SQLite database, chosen for its lightweight nature and ease of integration with the Flask web framework used to develop the backend logic.

PLANNED IMPLEMENTATION

TIMELINE

Week	Dates	Tasks
Week 1	Jan 8 - Jan 14	Collect requirements through piazza Define project scope and objectives.
Week 2	Jan 15 - Jan 21	Decide on the tech stack. Initialize project repositories, define directory structure, and set up development environments.
Week 3	Jan 22 - Jan 28	Backend Development Start designing system architecture and database schema.
Week 4	Jan 29 - Feb 4	Project Proposal Report
Week 5	Feb 5 - Feb 11	Develop RESTful APIs for user management and data interaction.
Week 6	Feb 12 - Feb 18	Frontend Development Begin
Week 7	Feb 19 - Feb25	Set up Flask server and integrate initial routes.
Week 8	Feb 26 - Mar 3	Mid Progress Report
Week 9	Mar 4 - Mar 17	Implement interactive elements in the frontend Setup SQLite database and integrate with Flask.
Week 10	Mar 18 - Mar 24	Integrate frontend with backend Ensure data flow through APIs and adjust based on frontend needs.
Week 11	Mar 25 - Mar 31	Implement login and user authentication system. Integrate ECharts for dynamic data visualization
Week 12	Apr 1 – Apr 7	Start unit testing and integration testing of APIs. Enhance and refine user interfaces and user experience.
Week 13	Apr 8 – Apr 14	Prepare the Demo Presentation
Week 14	Apr 15 - Apr 21	Project Demo Presentation
Week 15	Apr 22 – Apr 26	BUFFER TIME

ARCHITECTURE DESIGN



IMPLEMENTATION

FUNCTIONALITIES

The Air Data Management System is equipped with a range of functionalities designed to enhance user experience, provide robust data management, and deliver powerful data visualization capabilities. These functionalities are built to facilitate easy access, manipulation, and understanding of air quality data across various dimensions.

- User Account Management
  - Registration and Login: The application allows users to register and log in, storing user data securely in the SQLite database. During registration, if a user account does not already exist, a new user is added to the AirUser table with a unique identifier generated by uuid.uuid4(), along with the provided account details and a default role. This process ensures that each user has a unique presence in the system. All user data is stored in the database's AirUser.
  - User Authentication: For login, the application checks the AirUser table for matching account and password combinations, ensuring secure access to the user's data and functionalities within the application.
- Data Handling and CRUD Operations

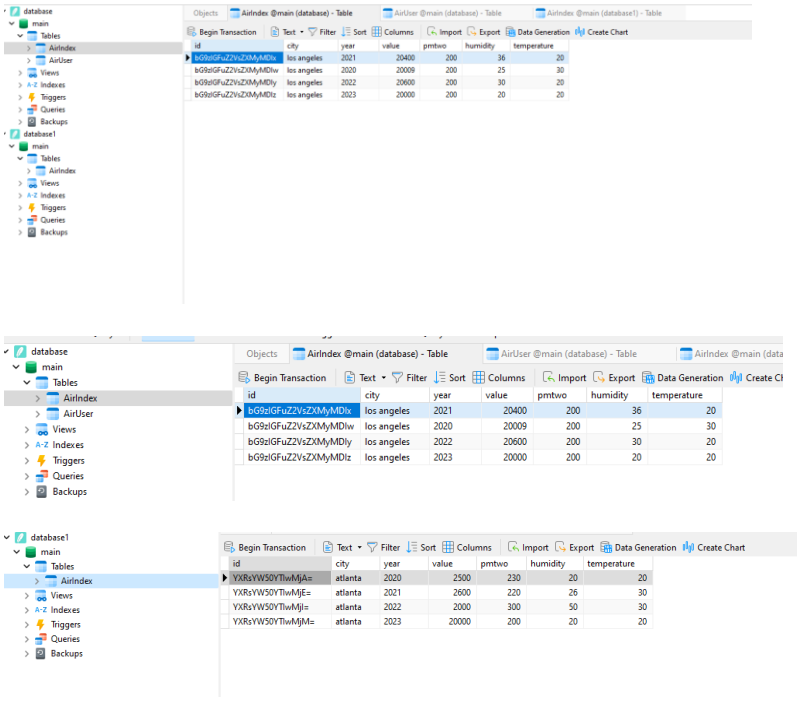
  - Data Storage: Utilizing two separate SQLite databases, the application manages air quality data and user account information efficiently to prevent performance bottlenecks caused by large datasets in a single database.
  - Create, Update, and Delete Operations: The backend provides comprehensive functionalities for managing air data, allowing users to add new records, update existing entries, and delete outdated information from the database. These operations are facilitated through easy-to-use API endpoints that handle data transactions securely and efficiently.
- Data Retrieval
  - By combining the name of the city with the year and generating a unique id via base64, it is sorted according to the first letter of the id. A-M are grouped in database and N-Z are grouped in database1. In this way we can efficiently and accurately search for the data that the user needs.
- Dynamic Data Visualization
  - Interactive Charts with ECharts and Chart.js: The application leverages both ECharts and Chart.js to render visual data effectively.
  - Horizontal Comparisons: ECharts is used to compare air quality across different cities within the same year, enabling users to visualize and analyze regional air quality trends.
  - Vertical Comparisons: Chart.js is utilized to conduct comparisons based on different years for the same city, providing insights into how air quality has changed over time in a specific location.
  - Data Preparation for Visualization: A getYear function is implemented to prepare and format data specifically for ECharts, ensuring the data structure meets the requirements of the visualization library. In order to cope with the different data output of chart.js, we have made corresponding adjustments in new.js.
- Responsive User Interface
  - Dynamic Content Generation: The frontend dynamically generates content such as buttons for each city using JavaScript, which when clicked, display corresponding air quality data charts in the mainChartBox. This interactive element enhances user engagement and makes data exploration intuitive.
  - Real-Time Data Updates: Utilizing AJAX calls through api.js, the application fetches updated data from the backend, ensuring that the displayed information is current and accurate.

TECH STACK

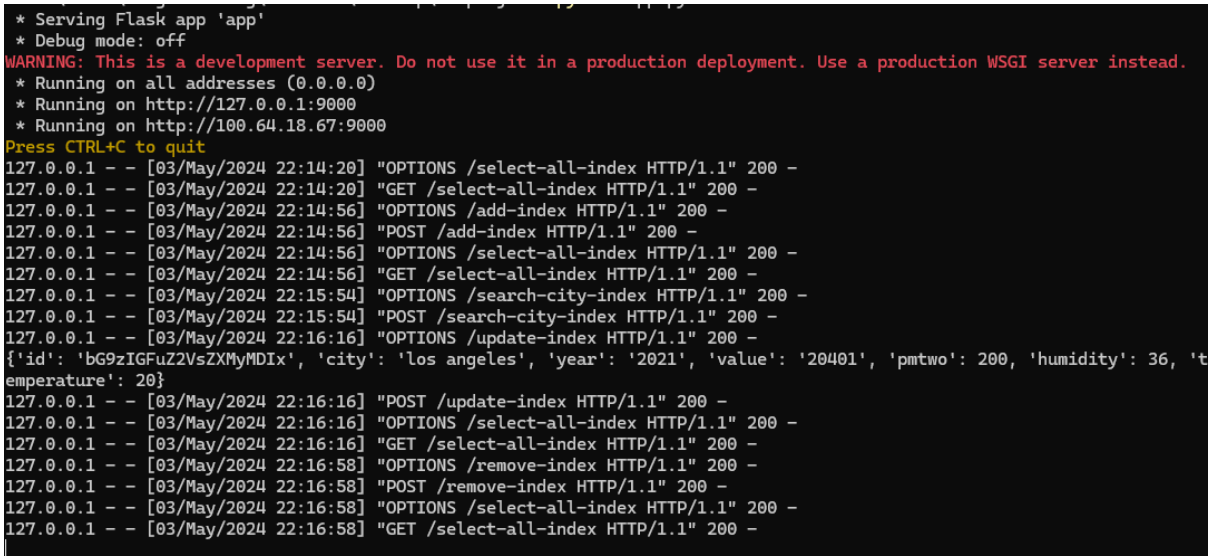
TECH NAME	DESCRIPTION
PYTHON	• Popular programming language with a large community and extensive library support • Serves as the primary programming language for the backend, chosen for its readability and extensive support for web development • Provides many built-in data structures and data manipulation capabilities. • Supports both object-oriented and functional programming paradigms. • Good for scripting, automation, and building web applications
FLASK	• Lightweight and versatile web framework written in Python, is used to handle backend server tasks • Provides the flexibility needed for rapid development without enforcing stricter application structure, making it ideal for projects requiring a custom solution.
SQLITE3	• Lightweight, file-based database engine that supports all essential SQL operations without requiring a separate server process.

	• Perfect for development and smaller-scale applications where simplicity and minimal configuration are preferred. • Handles all data storage, retrieval, and manipulation tasks, ensuring efficient data operations directly integrated with Flask.
JAVASCRIPT	• The scripting language for the frontend • Enables interactive web pages and is fundamental for integrating dynamic content and handling user interactions efficiently.
VUE.JS	• A progressive JavaScript framework • Used for building user interfaces. • Vue.js is particularly favored for its adaptability and the powerful ecosystem it supports, including tools for routing and state management, which are essential for a single-page application
ECHARTS, CHART.JS	• A JavaScript library for data visualizationIt • offers a rich array of chart types that are both interactive and customizable • meeting our needs for sophisticated data visualization capabilities.

IMPLEMENTATION

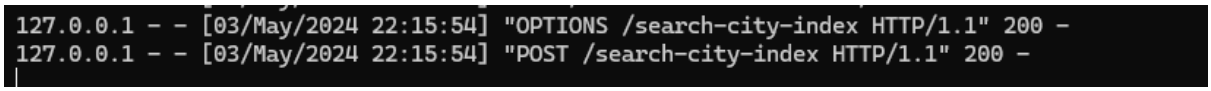


The above screenshot are showing the data in database and database1.

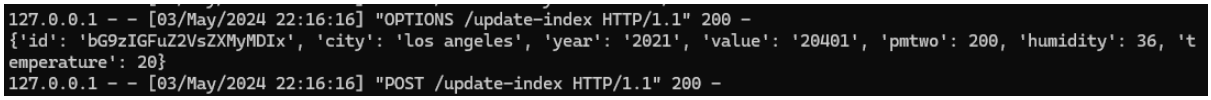


This screenshot shows the feedback for all the data operations in databases

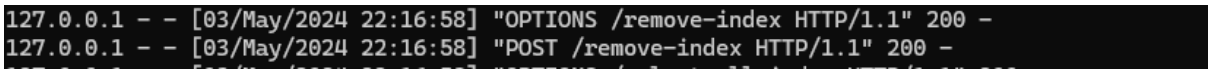
This screenshot shows the successful addititon of new data to the databases



This screenshot shows the successful searching of one city in databases



This screenshot shows the successful updating new data to databases.



This screenshot shows the successful deletion of data in databases.

```
127.0.0.1 - - [03/May/2024 22:18:00] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2024 22:18:00] "GET /main.css HTTP/1.1" 304 -
127.0.0.1 - - [03/May/2024 22:18:00] "GET /static/js/new.js HTTP/1.1" 304 -
127.0.0.1 - - [03/May/2024 22:18:00] "GET /static/js/chart/chart.js HTTP/1.1" 304 -
127.0.0.1 - - [03/May/2024 22:18:00] "GET /static/js/api.js HTTP/1.1" 304 -
2021los angeles
2021atlanta
2020los angeles
2020atlanta
2022los angeles
2022atlanta
2023los angeles
2023atlanta
127.0.0.1 - - [03/May/2024 22:18:00] "GET /get-year HTTP/1.1" 200 -
2021los angeles
2021atlanta
2020los angeles
2020atlanta
2022los angeles
2022atlanta
2023los angeles
2023atlanta
127.0.0.1 - - [03/May/2024 22:18:01] "GET /get-year HTTP/1.1" 200 -
```

This screenshot shows what happens when we call chart.js and echart for rendering.

```
PS C:\Users\Hengxuan Wang\OneDrive\Desktop\airproject\air-admin\air-admin> npm run dev

> vue-admin-template@4.4.0 dev
> vue-cli-service serve

 INFO  Starting development server...
98% after emitting CopyPlugin

 DONE  Compiled successfully in 2937ms


  App running at:
  - Local:   http://localhost:8080
  - Network: unavailable

  Note that the development build is not optimized.
  To create a production build, run npm run build.
```

This screenshot shows a successfully launch of our web application.

LEARNINGS OUTCOMES

OUTCOMES

- Advanced Frontend Development with Vue.js
  - Outcome: Gained proficiency in using Vue.js for creating dynamic user interfaces, learning to integrate and manage Vue components, directives, and lifecycle hooks effectively.
  - Skills Acquired: In-depth understanding of the Vue.js ecosystem, including Vue Router for SPA routing and Vuex for state management across components.
- Advanced Frontend Development with Vue.js
  - Outcome: Enhanced knowledge of building and deploying server-side applications using Flask, handling routes, requests, and responses efficiently.
  - Experience in integrating Flask with SQLite for data persistence, performing CRUD operations, and understanding Flask's application context and request lifecycle.
- Database Management with SQLite

- Outcome: Learned to design and manage lightweight databases with SQLite, understanding the nuances of SQL for querying and updating data.

CHALLENGES

- Data Handling and Visualization Complexity
  - The initial design for managing and visualizing air quality data proved inadequate due to the complexity of dynamic data handling and real-time updates.
  - Implemented a relational model with SQLite, which initially simplified CRUD operations but complicated real-time data aggregation and querying.
  - Found difficulties in real-time visualization which required fetching, processing, and displaying data simultaneously without significant delays.
- Integration of Mock Data with Frontend Development
  - Integrating mock data seamlessly with frontend development initially led to discrepancies between simulated and actual application behavior.
  - Used Mock.js extensively, which while helpful, sometimes masked issues that would only appear with real data interactions.
  - Encountered issues where mock data did not fully represent the complexity or scale of production data, leading to unanticipated bugs during later testing phases.
- Timeline Catchup & Mitigation
  - Encountered several unexpected challenges that required re-prioritizing tasks and adjusting the project timelineUsed Mock.js extensively, which while helpful, sometimes masked issues that would only appear with real data interactions.
  - Integration issues between frontend components and the backend data flow caused delays.

INDIVIDUAL CONTRIBUTIONS

The project responsibilities were shared equally between the two team members, ensuring a collaborative and balanced workload. Each member brought unique skills and perspectives to the project, contributing to various aspects of the application.

Hengxuan Wang Contributions:

1. Frontend Development: Took the lead in implementing the Vue.js frontend. This involved setting up the project structure, integrating ECharts and Chart.js for the data visualization components, and ensuring responsiveness across different devices. Also handled the integration of Vuex for state management to maintain a reactive and consistent user interface.
2. Database Design and Integration: Worked on designing the SQLite database schema and integrating it with the Flask backend. This task involved defining the tables, relationships, and ensuring efficient data retrieval methods were in place for the dynamic data visualization needs of the project.
3. Testing and Optimization: Focused on the initial rounds of performance testing, particularly in optimizing the frontend interactions and ECharts visualization performance under various data loads.

Zihan Mai Contributions:

1. Backend Development and API Design: Was primarily responsible for setting up the Flask backend, designing RESTful APIs, and handling server-side logic including user authentication, data processing, and CRUD operations for air quality data.

2. Mock Data Implementation: Took charge of implementing and managing the Mock.js setup, ensuring the frontend could be developed and tested independently of the backend. This involved creating realistic mock data and responses, which were crucial for early-stage frontend development.
3. Documentation and Reporting: Handled the majority of the project documentation, including the setup instructions, API usage guides, and the final project report. Also contributed to refining the testing strategies and implementing additional backend tests to ensure API reliability.

CONCLUSION

The development of the AIR DATA MANAGEMENT SYSTEM Dashboard has successfully demonstrated the ability to integrate modern web technologies to create a comprehensive platform for managing and visualizing air quality data. Throughout the project, we have leveraged the strengths of Vue.js, Flask, SQLite, Chart.js and ECharts to deliver a responsive and interactive application that meets the needs of users seeking real-time environmental data insights. This project not only addressed the initial objectives set forth but also provided a robust framework for handling complex data interactions and visualizations efficiently.

Our collaborative approach ensured that all aspects of the project were covered comprehensively, from frontend design and interaction to backend processing and database management. The challenges faced during development were turned into learning opportunities, enhancing our problem-solving skills and deepening our understanding of application development in a real-world context.

FUTURE STEPS

Looking ahead, there are several avenues for further development and enhancement of the Air Data Management System

1. Real Backend Integration: While the mock backend served its purpose for development, integrating a real backend will enable the application to handle live data streams and interact with actual databases. This will improve data accuracy and allow for real-time updates from various data sources.
2. Advanced Data Analytics Features: Implementing more advanced analytics capabilities, such as predictive modeling and trend analysis, could provide users with more in-depth insights into air quality trends and help in making informed decisions.
3. User Customization Options: Enhancing the dashboard to allow users to customize their views and save specific data filters or visualization settings would make the application more user-friendly and tailored to individual needs.