

Traffic Collision Record Web Application

Table of Contents

1. **Introduction**
2. **System Overview**
3. **Functionalities**
 - 3.1. User Authentication
 - 3.2. Collision Record Management
 - 3.3. CSV File Import
4. **Technologies Used**
 - 4.1. Backend: Spring Boot
 - 4.2. Frontend: React with Vite
 - 4.3. Styling: Tailwind CSS
 - 4.4. Database: MySQL
5. **Backend Validation**
6. **Project Workflow**
 - 6.1. Setup
 - 6.2. API Development
 - 6.3. Frontend Development
 - 6.4. UI Styling
 - 6.5. Testing and Validation
7. **System Architecture**
8. **Challenges Faced**
9. **Lessons Learned**
10. **Conclusion**

1. Introduction

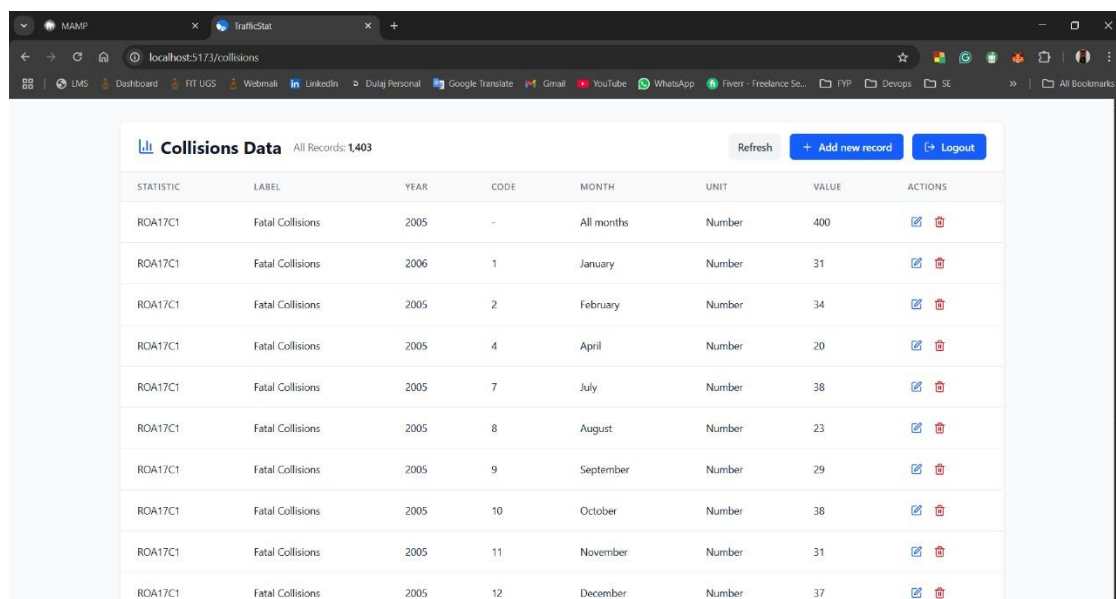
Road safety and traffic monitoring are becoming more and more important components of urban infrastructure in today's fast-paced society. Accidents and collisions are regrettable but frequent events, and public safety analysis, policymaking, and prevention tactics can all benefit from efficient data management of these incidences. Using a contemporary tech stack—Spring Boot for backend services, React with Vite for a quick frontend experience, MySQL for the relational database, and Tailwind CSS for responsive styling—I created a Traffic Collision Record Web Application in order to achieve this goal. Features like user authentication and complete CRUD operations for traffic collision record management are included in the system. A CSV file is used to load collision data initially, and data integrity is guaranteed by backend validations.

This report details the system design, implementation, features, and technologies used, along with challenges faced and future improvements.

2. System Overview

The application is a full-stack web solution that is straightforward but efficient, enabling users to safely log in and handle traffic collision reports. Using Tailwind CSS for a unified and contemporary appearance, the user interface is meant to be simple and easy to use. A MySQL database contains the CRUD (create, read, update, and delete) collision entries that users may access after logging in.

The system's main objective is to provide users with a simple and safe interface for managing traffic collision data.



The screenshot displays a web browser window with the URL `localhost:5173/collisions`. The application interface includes a header with the title "Collisions Data" and a subtitle "All Records: 1,403". There are three buttons in the top right: "Refresh", "+ Add new record", and "Logout". Below the header is a table with the following columns: STATISTIC, LABEL, YEAR, CODE, MONTH, UNIT, VALUE, and ACTIONS. The table contains 11 rows of data, all for "Fatal Collisions" in the year 2005, with values ranging from 20 to 400. Each row has edit and delete icons in the ACTIONS column.

STATISTIC	LABEL	YEAR	CODE	MONTH	UNIT	VALUE	ACTIONS
ROA17C1	Fatal Collisions	2005	-	All months	Number	400	Edit Delete
ROA17C1	Fatal Collisions	2006	1	January	Number	31	Edit Delete
ROA17C1	Fatal Collisions	2005	2	February	Number	34	Edit Delete
ROA17C1	Fatal Collisions	2005	4	April	Number	20	Edit Delete
ROA17C1	Fatal Collisions	2005	7	July	Number	38	Edit Delete
ROA17C1	Fatal Collisions	2005	8	August	Number	23	Edit Delete
ROA17C1	Fatal Collisions	2005	9	September	Number	29	Edit Delete
ROA17C1	Fatal Collisions	2005	10	October	Number	38	Edit Delete
ROA17C1	Fatal Collisions	2005	11	November	Number	31	Edit Delete
ROA17C1	Fatal Collisions	2005	12	December	Number	37	Edit Delete

3. Functionalities

3.1. User Authentication

- Users can sign in with their username and password on a login page.
- Before being stored, passwords are safely hashed using industry-standard encryption.
- A session or token is produced to authenticate subsequent requests after a successful login.
- Spring Security (or its equivalent secure handling in Spring Boot) is used to build the backend authentication mechanism.

3.2. Collision Record Management

- The application's CRUD functionality for traffic collisions is its primary feature.
- Users can:
 - Add new collision records via a form.
 - View existing records in a table format.
 - Edit specific fields in existing records.
 - Delete records if necessary.
- To guarantee data integrity, every operation such as date formats, mandatory fields, numerical ranges, etc. is verified on the backend.

3.3. CSV File Import

- A CSV file called collisions.csv is used to import the first collision data into the database.
- The file is read by a parser, which then updates MySQL's collisions database.
- One collision incidence is represented by each row in the CSV file.

4. Technologies Used

4.1. Backend: Spring Boot

- The popular Java-based technology Spring Boot is utilised to construct the application's backend.
- The MySQL database is accessed by ORM (Object-Relational Mapping) using Spring Data JPA.
- The purpose of REST APIs is to manage CRUD tasks.
- To enforce data integrity standards, validation annotations are employed, such as `@NotNull`, `@Size`, `@Pattern`, etc.

4.2. Frontend: React with Vite

- React is used for component-based programming when building the frontend.
- The build tool for quick development and hot-reloading is Vite.
- `useState` and `useEffect` hooks are used to manage state in responsive and interactive pages.
- HTTP queries to the backend APIs are made using `Axios`.

4.3. Styling: Tailwind CSS

- Tailwind CSS, a utility-first CSS framework, is used for styling.
- It offers a uniform layout for all pages, including data tables and login windows.
- The user experience is improved by custom responsiveness and hover effects.

4.4. Database: MySQL

- The relational database used by the application is MySQL.
- There are two main tables:
 - 01. users: keeps track of access roles and user credentials.
 - 02. collisions: records the location, timing, cause, severity, and other facts of every traffic collision.
- Constraints and indexes are added for dependability and performance.

5. Backend Validation

Backend validations are essential to preserving the system's accuracy and security. The following validation methods are used by the application:

Field Requirements:

- Required fields can't be empty.
- The length and content of strings are verified.
- Dates undergo parsing and format validation.

Error Handling:

- When validation fails, custom error messages are sent back.
- HTTP status codes and error messages are used to structure all API responses.

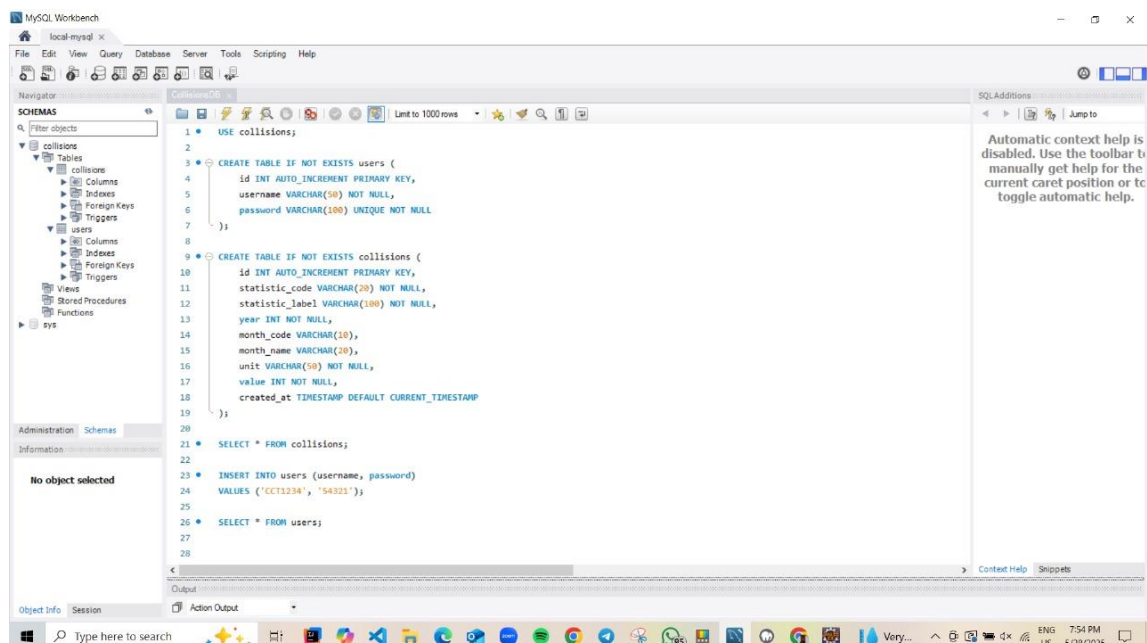
Validations for security:

- ORM and parameterised queries help prevent SQL injection.
- Before being stored in the users table, passwords are hashed.

6. Project Workflow

6.1. Setup

- Make tables and a MySQL database.

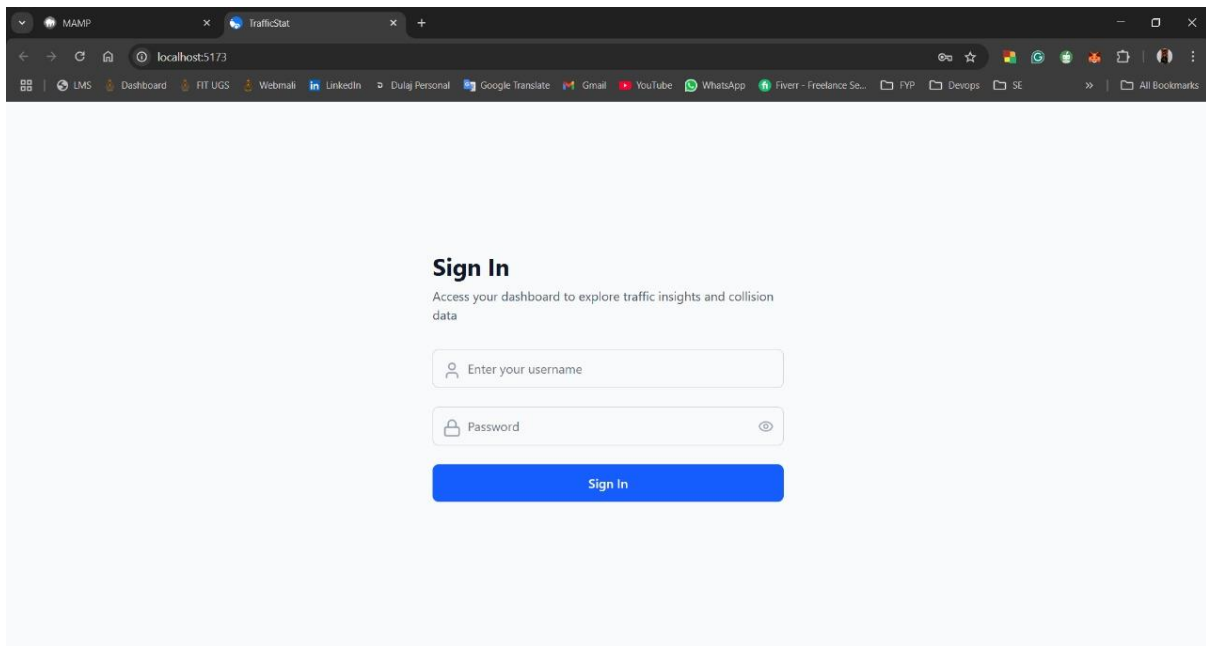


6.2. API Development

- Create endpoints for error handling, record management, and login.

6.3. Frontend Development

- Configure the environment for Vite plus React.
- Make a page:
 - The login page.
 - dashboard.
 - interface for managing collision records.
- Use Axios to integrate with the backend.



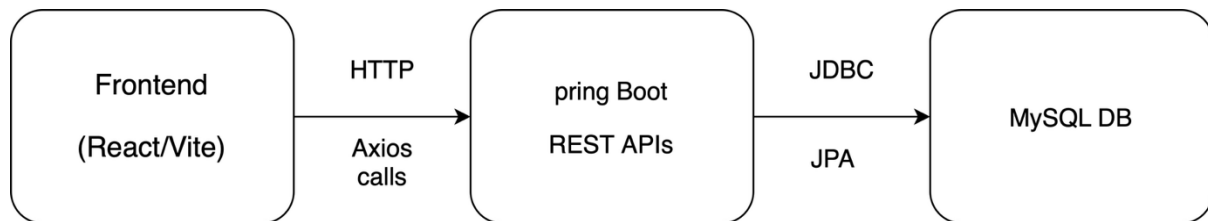
6.4. UI Styling

- Use Tailwind CSS when creating responsive elements.
- Make sure the layout is tidy and reliable.

6.5. Testing and Validation

- Manual testing of all CRUD operations.
- Check login, record addition, update, and deletion with various inputs.
- Validate form inputs before submission.

7. System Architecture



- The frontend sends requests via Axios.
- The backend (Spring Boot) handles authentication and CRUD operations.
- Data is stored and retrieved from the MySQL database.

8. Challenges Faced

1. Data loading and CSV parsing
 - A few CSV files contained missing fields or were distorted.
 - Fallback processing and logging for corrupt entries were put into place as a solution.
2. Consistency in Validation
 - ensuring alignment between backend and frontend validation.
 - Solution: To stop tampering, extra attention was paid to backend validation.
3. Problems with CORS
 - Throughout development, the Spring backend and React frontend operated on separate ports.
 - Solution: To enable cross-origin requests, Spring Boot enabled CORS.
4. Management of Sessions
 - putting in place safe session-based authentication without a whole OAuth framework.
 - Solution: Made use of the fundamental session management technique provided by Spring Security.
5. UI that is responsive
 - At first, it was challenging to make tables and forms mobile-friendly.
 - Using Tailwind's responsive classes was the solution.

9. Lessons Learned

1. Permissions and Roles for Users
 - For more regulated access, add admin and viewer roles.
2. Pagination and Search
 - Use pagination and collision record filtering to increase usability.
3. Analytical Graphics
 - Show graphs and charts to analyse collision trends visually.
4. CSV Export Feature
 - For reporting purposes, users can export collision data back to CSV using the CSV Export Feature.
5. Integration and Unit Tests
 - To make sure the code is stable, create test cases with Mockito and JUnit.
6. Authentication Through Tokens
 - Use JWT-based authentication for a stateless, more secure method.

10. Conclusion

One useful tool for efficiently organising traffic accident data is the Traffic Collision Record Web Application. Despite having a straightforward structure, it exemplifies several essential full-stack web development concepts, such as responsive user interfaces, secure login systems, reliable backend services, and structured database design.

This project's construction gave me extensive exposure to contemporary web technology and practical software design techniques. This system can be developed into a comprehensive traffic data management platform for safety departments or municipalities with the inclusion of new features and improvements.