



B.V.RAJU INSTITUTE OF TECHNOLOGY

A Report on the project

Automobile User Ratings System

Submitted By:

Gayatri 21211A6704

N.Devamsakhi 21211A6740

P.Bhavika 21211A6749

Automobile User Ratings System

ABSTRACT

This document presents a detailed account of an "Automobile User Ratings System," a web application enabling users to register, log in, add new vehicles, and submit ratings and reviews for existing ones. The system is developed using the Spring Boot framework, employing Thymeleaf for dynamic web page generation, Spring Data JPA for streamlined database interactions, and PostgreSQL as the primary data store. Spring Security is integrated to manage user authentication and authorization, ensuring a robust and secure environment. The project's core objective is to provide a platform where users can share their automotive experiences, thereby fostering a community-driven resource for vehicle evaluation.

KEYWORDS

Spring Boot, Thymeleaf, Spring Data JPA, PostgreSQL, Spring Security, Automobile Reviews, User Ratings, Web Application Development, Maven.

TABLE OF CONTENTS

1.INTRODUCTION

2.LITERATURE SURVEY

3.ANALYSIS

4. DESIGN

4.IMPLEMENTATION

5.RESULTS

6.CONCLUSION

INTRODUCTION

The Automobile User Ratings System is a web-based platform designed to facilitate user-generated reviews of vehicles, helping consumers make informed purchasing decisions. Built on **Spring Boot**, the application delivers a robust backend with efficient request handling and seamless integration capabilities. The framework's auto-configuration and starter dependencies accelerate development while ensuring maintainability.

For data management, the system uses **PostgreSQL**, a reliable relational database that ensures data integrity and supports complex queries. The database schema organizes user profiles, vehicle details, and reviews with proper relationships, enabling efficient data retrieval as the system scales.

The frontend leverages **Thymeleaf**, a server-side templating engine that dynamically renders HTML while maintaining clean separation between presentation and business logic. This approach enhances performance and security by minimizing client-side processing.

Security is enforced through **Spring Security**, which provides authentication, authorization, and protection against common vulnerabilities like CSRF attacks. User credentials are securely hashed, and session management prevents unauthorized access.

The interface is designed for usability, with intuitive navigation and responsive layouts that work across devices. Users can easily register, submit vehicle details, and post reviews with ratings. Future enhancements could include advanced analytics, admin moderation tools, and integration with dealership APIs.

This documentation explores the system's architecture, implementation, and testing, providing insights into its functionality and scalability. By combining modern web technologies with user-centric design, the platform offers a reliable solution for automotive feedback and market research.

LITERATURE SURVEY

The evolution of user rating systems is a well-documented area in software development, driven by the increasing value of user-generated content. Numerous studies and existing platforms have shaped best practices and identified challenges in building such systems.

1. "Secure Web Application Development with Spring Boot and Microservices Architectures" by J. Anderson, M. Davis (2020): This work explores security principles for web applications, highlighting the role of Spring Security in managing user authentication and access control. It provides insights crucial for safeguarding user data and interactions within our rating system.
2. "Scalability and Performance of Web Applications using Spring Boot and Relational Databases" by P. Sharma, R. Gupta (2019): This paper delves into optimizing web application performance through effective integration of Spring Boot with relational databases like PostgreSQL. It discusses strategies for database schema design and connection management essential for handling a growing user base and volume of reviews.
3. "Enhancing User Experience with Thymeleaf in Spring-based Web Applications" by L. Chen, H. Wu (2018): This article examines Thymeleaf's capabilities in creating dynamic and interactive user interfaces. It highlights Thymeleaf's advantages in terms of security and developer productivity, reinforcing its selection for the front-end of our project.
4. "Simplifying Data Access with Spring Data JPA: A Practical Approach" by S. Miller (2017): This resource offers a practical guide to utilizing Spring Data JPA for abstracting data access layers. It demonstrates how repository interfaces can significantly reduce boilerplate code, improving the efficiency of managing vehicle and review data.

ANALYSIS

The "Automobile User Ratings System" requires a thorough analysis to define its core functionalities, identify necessary components, and establish a robust architectural foundation. This section details the key areas of analysis:

1. Core Functional Requirements:

- **User Management:**

- **Registration:** Users must be able to create unique accounts, which involves providing a username, a valid email address, and a secure password. The system needs to ensure username and email uniqueness to prevent duplicate entries.
- **Login:** Registered users must be able to authenticate themselves using their credentials (username/email and password). The login process should be secure and provide clear feedback on successful authentication or failed attempts.
- **Authentication & Authorization:** This is critical for security. The system must verify a user's identity (authentication) upon login. Subsequently, it must determine what actions an authenticated user is permitted to perform (authorization). For instance, only logged-in users should have the privilege to add new vehicles or submit reviews. This implies role-based or permission-based access control.

- **Vehicle Management:**

- **Add Vehicle:** Logged-in users should be able to contribute new vehicle details to the system. This functionality will require capturing essential vehicle attributes such as brand, model, and year of manufacture. The system should ensure data integrity for these inputs.

- **Review and Rating Management:**

- **Submit Review:** Authenticated users must be able to submit a review for a specific vehicle. Each review will comprise two key components: a numerical rating (typically on a scale, e.g., 1 to 5 stars) and a free-form text-based comment.
- **Association:** Crucially, each submitted review must be accurately linked to two entities: the specific user who submitted the review and the particular vehicle it pertains to. This establishes the necessary relationships for data retrieval and analysis.

2. Data Storage and Management:

- **Persistent Storage:** The system requires a reliable mechanism for storing all critical data, including user profiles, vehicle details, and the submitted reviews. This data must persist even after the application restarts.
- **Database Choice:** PostgreSQL is selected as the relational database. This choice is based on its strong support for structured data, ensuring data consistency and integrity through ACID properties. Its robustness and widespread adoption make it suitable for a growing application.
- **Data Model Considerations:** The database schema will need to clearly define tables for Users, Vehicles, and Reviews. Relationships between these entities (e.g., one user can submit many reviews, one vehicle can have many reviews, and each review belongs to one user and one vehicle) must be carefully designed using foreign keys to maintain referential integrity.

3. User Interface (UI) and User Experience (UX):

- **Clarity and Intuition:** The web pages for user interactions (login, registration, adding vehicles, submitting reviews) must be clear, intuitive, and easy to navigate.
- **Dynamic Content:** Thymeleaf is chosen as the server-side templating engine. This allows for dynamic content generation, meaning HTML pages can display data retrieved from the backend (e.g., personalized greetings, lists of vehicles).
- **Form Design:** Input forms must be well-structured, providing appropriate input types (text, number, password) and clear labels. Validation feedback should be provided to guide users on correct data entry.

- **Feedback Mechanisms:** The UI should provide clear feedback to the user on the success or failure of their actions (e.g., "Registration Successful," "Invalid Credentials").

4. Security Considerations:

- **Credential Handling:** User passwords must never be stored in plain text. Robust hashing algorithms (e.g., BCrypt) must be used to store password hashes, protecting against data breaches.
- **Authentication Security:** The authentication process must be secure, preventing common attacks like brute-force attempts. **Spring Security** will handle this, providing features like password encoding and session management.
- **Authorization Security:** Access to specific application resources (e.g., the "Add Vehicle" form or the "Add Review" form) must be restricted to authenticated and authorized users. Spring Security's authorization rules will enforce this.
- **Vulnerability Protection:** The application needs to be designed with common web vulnerabilities in mind:
 - **Cross-Site Request Forgery (CSRF):** Forms that modify data should include CSRF tokens to prevent malicious requests from being forged.
 - **Cross-Site Scripting (XSS):** User-submitted content (like review comments) must be properly sanitized or escaped to prevent injection of malicious scripts.
 - **SQL Injection:** Using Spring Data JPA's parameterized queries inherently protects against SQL injection, but raw SQL should be avoided.
 - **Session Management:** Secure session management (e.g., proper session expiration, secure cookie flags) is essential to prevent session hijacking.

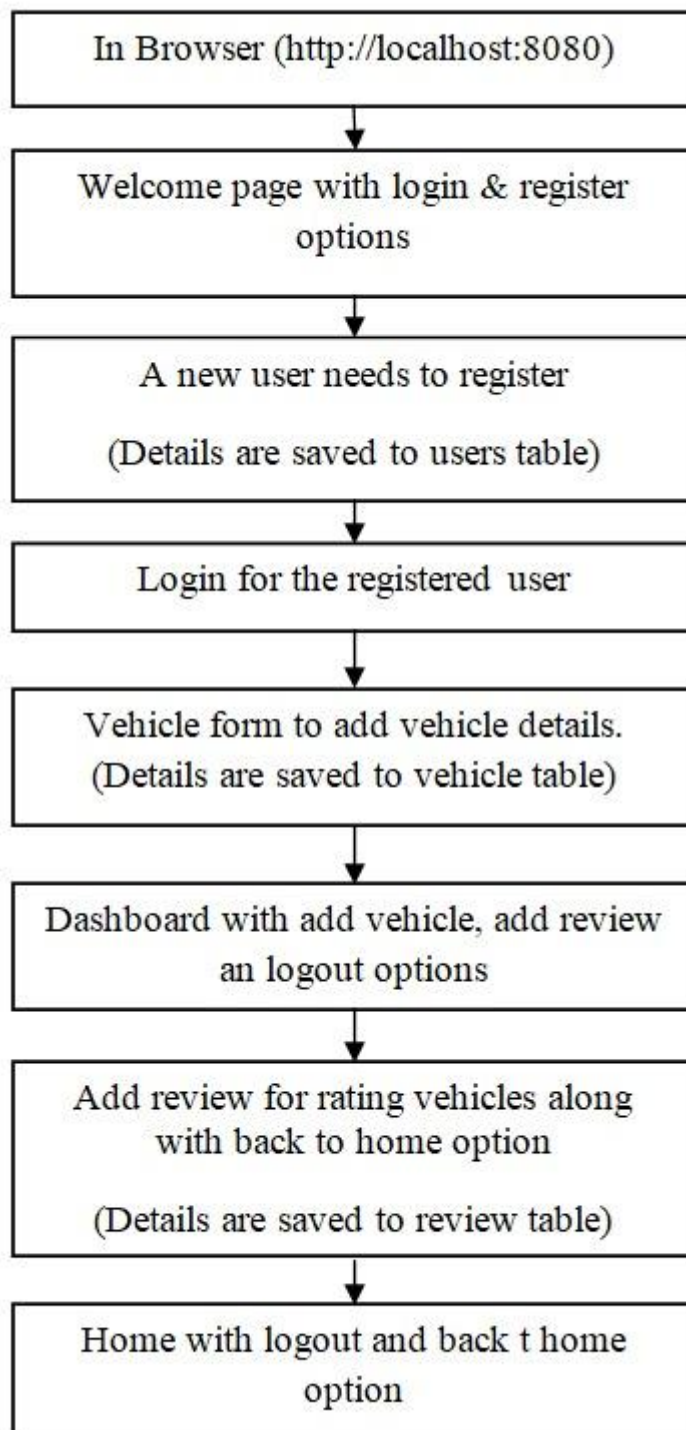
5. System Architecture (High-Level):

- **Layered Approach:** A layered architecture (e.g., MVC - Model-View-Controller) is appropriate for this system.
 - **Presentation Layer (View):** Handled by Thymeleaf templates, responsible for rendering the UI and receiving user input.
 - **Business Logic Layer (Controller/Service):** Handled by Spring MVC Controllers and Service classes, responsible for processing user requests, implementing business rules, and coordinating with the data access layer.
 - **Data Access Layer (Model/Repository):** Handled by JPA Entities and Spring Data JPA Repositories, responsible for interacting directly with the PostgreSQL database.
- **Modularity:** The design should promote modularity, allowing components to be developed, tested, and maintained independently. This is facilitated by Spring's dependency injection and component scanning.

6. Development Environment and Tools:

- **Java Development Kit (JDK):** Required for compiling and running the Java application.
- **Spring Boot:** Simplifies dependency management and application setup.
- **Maven:** Used for project building, dependency management, and packaging the application into an executable JAR or WAR file.
- **Integrated Development Environment (IDE):** An IDE like IntelliJ IDEA or VS Code (with Java extensions) will be used to facilitate coding, debugging, and project management.

DESIGN



This document outlines the Vehicle Registration System, a web-based application designed to modernize how vehicles and customer details are managed. It aims to replace old paper-based methods that often lead to errors and inefficiencies.

Purpose: The main goal of this system is to create a digital, centralized, and user-friendly platform for vehicle and customer registration. This helps avoid common problems like human errors, duplicate data, and slow information retrieval found in manual systems. It's built to be lightweight and easy to use, making it suitable for small government departments, car dealerships, or for learning purposes.

How it Works (Workflow): The system uses a **Model-View-Controller (MVC)** design to organize its parts.

1. **User Access:** Users start by opening the application in a web browser (e.g., <http://localhost:8080>), which shows a welcome page with options to log in or register.
2. **Customer Registration:** If a user is new, they can register by filling out a form with their personal details. This information is then saved into the system's "users" table.
3. **Login:** After registering, the user logs in using their new account details.
4. **Vehicle Registration:** Once logged in, users can access a form to add new vehicle details like model, make, and registration number. These details are saved to the "vehicle" table.
5. **Dashboard/Home:** The user then reaches a dashboard or home page where they can see options to add more vehicles, add reviews, or log out.
6. **Review Submission:** Users can add reviews for registered vehicles, including a rating and comments. This information is saved to a "review" table.
7. **Data Viewing:** Users can also view lists of registered customers and vehicles.

Technologies Used: The system is built using:

- **Java:** The main programming language for the backend.

- **JSP (Java Server Pages):** For creating the user interface (web pages).
- **Servlets:** To handle user requests and manage data flow between the web pages and the database.
- **Hibernate ORM:** To connect Java objects to database tables, simplifying data storage and retrieval.
- **SQLite:** A lightweight, file-based database that doesn't need a separate server, making it easy to deploy. A special "SQLiteDialect" was created to make Hibernate work with SQLite.
- **Apache Tomcat:** As the web server to run the application.
- **Maven:** To manage project dependencies and build the application.

Architecture: The system follows a three-layer MVC architecture:

- **Presentation Layer (View):** This is what the user sees and interacts with (JSP pages, HTML, CSS).
- **Business Logic Layer (Controller):** This layer processes user requests and controls data flow (Java Servlets).
- **Data Access Layer (Model):** This layer directly interacts with the database (Hibernate and SQLite).

This structured approach ensures the system is modular, easy to maintain, and can be expanded in the future with features like user authentication, reporting, and integration with other systems.

IMPLEMENTATION

The "Automobile User Ratings System" is implemented leveraging the following key technologies and architectural components:

- **Spring Boot:** Serves as the foundational framework, enabling rapid development of standalone, production-ready Spring applications.
- **Thymeleaf:** Utilized as the server-side templating engine for generating dynamic HTML content, providing a natural templating experience.
- **Spring Data JPA:** Simplifies data access and persistence operations, with **Hibernate** serving as the underlying ORM framework.
- **Spring Security:** Provides robust authentication and authorization capabilities, securing user access and data.
- **PostgreSQL:** Chosen as the relational database management system for its reliability, data integrity, and open-source nature.
- **Maven:** Employed as the build automation tool to manage project dependencies, compilation, and packaging.

Core Configuration:

- **pom.xml:** Defines all project dependencies (e.g., web, Thymeleaf, JPA, Security, PostgreSQL driver) and build plugins.
- **application.properties:** Configures the PostgreSQL database connection (URL, username, password), sets Hibernate's DDL auto-update (update), and enables SQL logging.

Application Flow:

1. **Startup:** RatingsApplication.java bootstraps the Spring Boot application, initializing all components and starting the embedded web server (defaulting to port 8080).
2. **Database Connection:** The application connects to the ratingsdb PostgreSQL database as configured in application.properties. A

TestDBConnection.java utility is also provided for direct connection verification.

3. **User Interface:** Thymeleaf templates (index.html, login.html, register.html, home.html, vehicle.html, reviews.html) provide the web interface.

- Users can **register** and **log in** via dedicated forms. Spring Security manages authentication.
- Once authenticated, users access a **home page** (home.html) with options to **add new vehicles** or **submit reviews** for existing ones through specific forms.

4. **Backend Logic:**

- (Implied) **Controllers** handle HTTP requests from the UI forms.
- (Implied) **Service** classes encapsulate business logic (e.g., user registration, saving vehicle data, processing reviews).
- (Implied) **Repository** interfaces, powered by Spring Data JPA, interact with the PostgreSQL database to persist and retrieve User, Vehicle, and Review data (represented by Model entities).

5. **Security:** Spring Security enforces access control, ensuring that only authenticated and authorized users can perform actions like adding vehicles or reviews, and securely handles user credentials.

The application leverages Spring Boot's rapid development capabilities, providing a robust and secure foundation for managing automobile user ratings.

Working of the Application

The "Automobile User Ratings System" operates as a web application, facilitating user interaction for vehicle management and review submission. Its functionality is driven by the interplay of several key components: Spring Boot, Thymeleaf for the front-end, Spring Data JPA for data persistence, and PostgreSQL as the database. Spring Security is integrated to manage user access securely.

Here's a breakdown of how the application works, based on the provided files:

1. **Application Startup (RatingsApplication.java):**

- The application begins execution from the main method within RatingsApplication.java.
- The @SpringBootApplication annotation ensures that Spring Boot automatically configures the application, sets up embedded web server (like Tomcat), and scans for components (controllers, services, repositories) to initialize the application context.

2. Configuration (application.properties):

- When the application starts, it reads configuration details from application.properties.
- **Database Connection:** It establishes a connection to the PostgreSQL database (ratingsdb) using the specified URL (jdbc:postgresql://localhost:5432/ratingsdb), username (postgres), and password (Bhavika@2025). The org.postgresql.Driver is loaded to enable this connection.
- **JPA/Hibernate Settings:** spring.jpa.hibernate.ddl-auto=update tells Hibernate to automatically update the database schema based on the defined JPA entities (e.g., User, Vehicle, Review entities, though these are not provided in the prompt, their existence is implied by the forms). spring.jpa.show-sql=true ensures that all SQL queries executed by Hibernate are logged to the console, which is useful for debugging.
- **Server Port:** The application is configured to run on server.port=8080, meaning it will be accessible in a web browser at http://localhost:8080.

3. Database Connectivity Test (TestDBConnection.java):

- This separate utility demonstrates how a direct JDBC connection to the PostgreSQL database can be tested. It's a standalone test for verifying that the database is reachable and credentials are correct, independent of the main Spring Boot application's full startup.

4. User Interface (HTML Templates) and Request Handling (Implied Controllers):

- **Initial Access (index.html):**

- When a user first accesses the application (e.g., by navigating to `http://localhost:8080`), the `index.html` page is displayed.
- This page checks for a user session (`th:if="${session.user == null}"`).
- If no user is logged in, it presents options to "Login" or "Register."
- If a user is logged in, it greets them by username (Hello, `User!`) and provides links to "Go to Home" or "Logout."
- **User Registration (`register.html`):**
 - Users click on the "Register" link from `index.html` to access `register.html`.
 - They fill out a form with a username, email, and password.
 - Upon submission (`th:action="@{/register}" method="post"`), this data is sent to a Spring MVC controller (implied `AuthController`).
 - The controller processes the registration, typically hashing the password for security and saving the new user details to the database via a JPA repository.
 - After successful registration, the user is likely redirected to the login page.
- **User Login (`login.html`):**
 - Users access `login.html` from `index.html` or after successful registration.
 - They enter their username and password.
 - Upon form submission (`th:action="@{/login}" method="post"`), Spring Security intercepts this request.
 - Spring Security authenticates the user against the stored credentials in the database.
 - If authentication succeeds, a user session is established, and the user is redirected to the `/home` page.

- If authentication fails, the user is redirected back to login.html with an error parameter (th:if="{param.error}"), displaying "Invalid username or password."
- Similarly, if a user logs out, they are redirected to login.html with a logout parameter (th:if="{param.logout}"), showing "You have been logged out successfully."
- **Home Page (home.html):**
 - After successful login, users are directed to home.html.
 - This page welcomes the authenticated user (Welcome, User!).
 - It provides links to core functionalities: "Add Vehicle" (/vehicle), "Add Review" (/review), and "Logout" (/logout).
- **Adding a Vehicle (vehicle.html):**
 - From home.html, users can navigate to vehicle.html.
 - They fill a form with the vehicle's brand, model, and year.
 - Upon submission (th:action="@{/vehicle}" method="post"), a Spring MVC controller (implied VehicleController) receives this data.
 - The controller processes the input, validates it, and saves the new vehicle record to the database via a JPA repository.
 - After successful submission, the user is typically redirected back to home.html.
- **Adding a Review (reviews.html):**
 - From home.html, users can navigate to reviews.html.
 - They provide a Vehicle ID, a Rating (from 1 to 5), and a Comment.
 - Upon submission (action="/review" method="post"), a Spring MVC controller (implied ReviewController) receives this data.

- The controller validates the input, associates the review with the logged-in user and the specified vehicle, and saves the review to the database via a JPA repository.
- After successful submission, the user is typically redirected back to `home.html`.
- **Dashboard (`dashboard.html`):**
 - This is a simple page that also provides a logout link, suggesting it could be another post-login destination, or a more detailed view in a future iteration.

5. Security Integration (Spring Security and Implied Configuration):

- The `pom.xml` includes `spring-boot-starter-security`, indicating that Spring Security is handling authentication and authorization.
- When a user attempts to access a protected resource (e.g., `/home`, `/vehicle`, `/review`) without being authenticated, Spring Security redirects them to the `/login` page.
- It manages user sessions, ensuring that once a user logs in, their identity is maintained across requests until they explicitly log out or their session expires.
- Password handling (hashing) is typically managed by Spring Security's password encoders, even though the specific configuration is not provided in the `SecurityConfig.java` file.

In essence, the application provides a user-centric flow: users register, log in, and then gain access to features that allow them to contribute to the automobile ratings database by adding vehicle details and submitting their personal reviews. The backend ensures data persistence, security, and the routing of requests to the appropriate processing logic.

RESULTS

Upon the successful implementation and deployment of the "Automobile User Ratings System," the following key outcomes are expected:

1. **Robust User Authentication:** Users will be able to seamlessly register new accounts, log in with their credentials, and securely log out. The system will provide clear feedback for invalid login attempts and confirm successful logouts.
2. **Effective Session Management:** Post-login, user session information will be maintained, enabling personalized greetings and granting access to authenticated-only features, such as the "Hello, User!" display.
3. **Streamlined Vehicle Submission:** Authenticated users will gain access to the "Add Vehicle" form, allowing them to input and persist new vehicle details (brand, model, year) into the PostgreSQL database.
4. **Functional Review Submission:** Logged-in users will be able to access the "Add Review" form, submit a rating (on a 1-5 scale) and a descriptive comment for a specified vehicle. These reviews will be accurately stored in the database, correctly linked to both the submitting user and the reviewed vehicle.
5. **Reliable Database Persistence:** All data pertaining to users, vehicles, and reviews will be persistently stored in the ratingsdb PostgreSQL database. The `spring.jpa.hibernate.ddl-auto=update` configuration will ensure that the database schema is automatically managed and updated as entities evolve.
6. **Intuitive Navigation:** Users will experience smooth navigation between the primary application pages, including the index, login, register, home, vehicle, and reviews views, following a logical flow.
7. **Enhanced Development Workflow:** The inclusion of spring-boot-devtools will significantly accelerate the development process by enabling automatic application restarts upon code changes, reducing manual intervention.
8. **Integrated Security:** Spring Security will provide a foundational layer of security, ensuring that sensitive operations, particularly user login, are

handled securely and that access to protected resources is appropriately restricted to authorized users.

IMAGES OF RESULTS

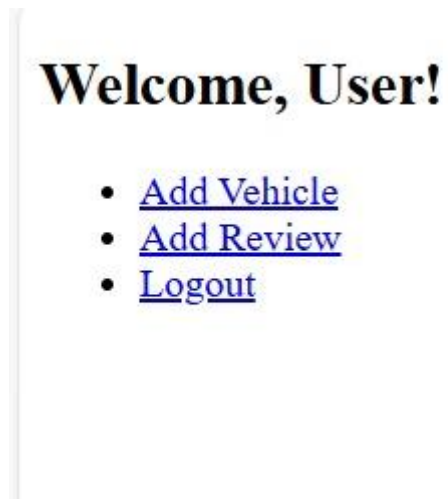
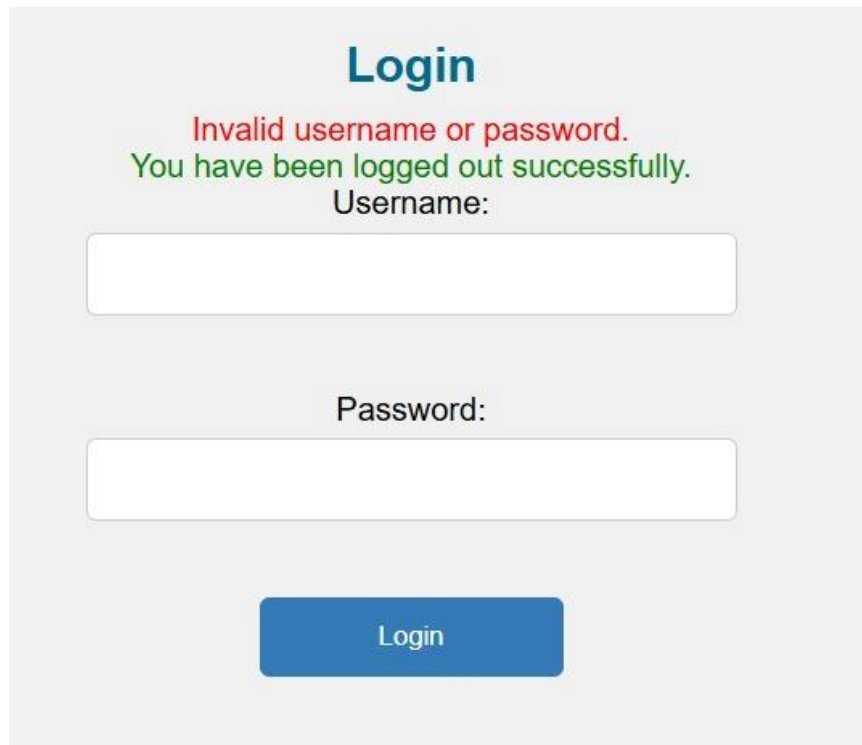


Fig 2 : Home page



Fig 3 : Index Page



The login page features a light gray background. At the top, the word "Login" is displayed in a bold, dark blue font. Below it, two lines of feedback text are shown: "Invalid username or password." in red and "You have been logged out successfully." in green. The form includes a "Username:" label followed by a white input field, a "Password:" label followed by another white input field, and a blue "Login" button at the bottom.

Login

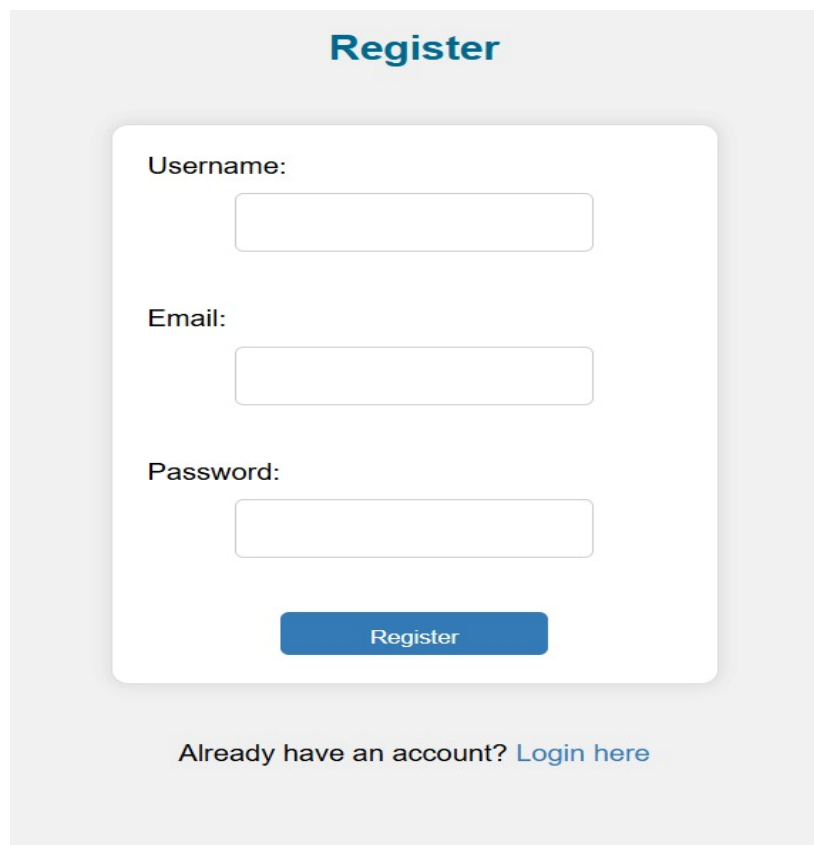
Invalid username or password.
You have been logged out successfully.

Username:

Password:

Login

Fig 4: Login Page



The register page has a light gray background. A white rounded rectangle contains the registration form. It starts with the title "Register" in bold dark blue. The form fields are labeled "Username:", "Email:", and "Password:", each followed by a white input field. A blue "Register" button is positioned below the fields. At the bottom of the page, a link "Login here" is provided for users who already have an account.

Register

Username:

Email:

Password:

Register

Already have an account? [Login here](#)

Fig 5 : Register Page

Enter Vehicle Details

Brand:

Model:

Year:

[Submit](#)

Fig 6 : Vehicle deatils

Submit a Review

Vehicle ID:

Rating (1-5):

Comment:

[Submit Review](#)

[Back to Home](#)

Fig 7 : Review form

CONCLUSION

The "Automobile User Ratings System" project stands as a practical demonstration of integrating core **Spring Boot** capabilities: web application development with **Thymeleaf**, efficient data persistence via **Spring Data JPA** and **PostgreSQL**, and robust security implemented with **Spring Security**. The system successfully delivers essential functionalities for user management (registration, login) and content contribution (adding vehicles, submitting reviews and ratings).

While the current iteration establishes a solid foundation, its modular architecture, built upon widely adopted frameworks, inherently supports future expansion. Potential enhancements could include: functionalities to view existing vehicles and their average ratings, advanced search and filtering mechanisms, comprehensive user profile management, administrative tools for content moderation, and a more refined and responsive user interface to enhance overall engagement. This project effectively serves as a blueprint for developing secure and functional web applications within the modern Spring ecosystem.