
# IT 309 SOFTWARE ENGINEERING

## PROJECT DOCUMENTATION

CAR RENTAL

Prepared by:
**Mustafa Ajanović**
**Amir Bašović**
**Arnela Ombaša**

Proposed to:

**Nermina Durmić, Assist. Prof. Dr.**
**Naida Fatic, Teaching Assistant,**
**Mirza Krupić, Teaching Assistant**

June, 2024

TABLE OF CONTENTS

# 1. Introduction

## 1.1. About the Project

The Car Rental System is a comprehensive web application developed to facilitate the online booking and management of rental cars. This project allows users to browse, book, and manage rental cars effortlessly. Admins have the capability to manage the car inventory, ensuring that the system is always up-to-date with the latest information. The project emphasizes security, usability, and performance, ensuring a seamless experience for both users and administrators.

The application features user-friendly interfaces for both customers and administrators. Customers can easily search for available cars, filter results based on various criteria such as price, and view detailed information about each car, including specifications, availability, and pricing. The booking process is streamlined, allowing users to reserve cars for specific durations.

Administrators have access to a robust backend interface where they can manage car listings, update car details, and handle user accounts. The system supports adding, editing, and deleting car information, making it easy to keep the inventory current. Advanced security measures, including JWT authentication ensure that user data is protected and transactions are secure.

The Car Rental System also includes several non-functional requirements aimed at enhancing the user experience. These include quick loading times, responsive design for compatibility with various devices, and intuitive navigation to help users easily find the information they need. The application leverages modern web technologies to deliver a fast, reliable, and secure service.

## 1.2. High-level Plan

**Product Roadmap**

| **Date** | May, 2024 | June, 2024 |
|---|---|---|
| **Name** | Backend focus - Car Rental 1.0 | Frontend focus - Car Rental 1.1 |
| **Goal** | Establish a robust backend with essential functionalities and a secure API. | Optimize the user interface for better interaction and increase overall system usability. |
| **Features** | - Register user<br>- Login and logout<br>- Browse cars<br>- View car details<br>- Add, edit, and delete cars (Admin)<br>- Book car<br>- Implement JWT for security<br>- Set up database schemas | - Refine login and registration UI/UX to streamline user access.<br>- Develop an interactive car browsing experience with filters and detailed views.<br>- Create a user dashboard for managing personal info and bookings<br>- Implement a responsive design for cross-device compatibility.<br>- Integrate 2FA setup and payment UI within the user profile for enhanced security and usability. |
| **Metrics** | - Number of registered users<br>- API response times<br>- Number of transactions processed<br>- Successful implementation of security measures | - Reduction in average time taken for users to complete the login and registration process.<br>- Increase in the number of users utilizing filters and viewing car details.<br>- User engagement rate with the dashboard features (e.g., number of bookings managed, reviews written).<br>- Percentage of sessions without cross-device compatibility issues.<br>- Uptake rate of 2FA among users and successful transactions rate post-payment UI update. |

**Release Plan**

# Release 1: Backend Focus - Car Rental 1.0 (May, 2024)

The goal for this release is to establish a robust backend with all the essential functionalities and a secure API. This involves setting up the core services that enable the system's primary operations.

**Features:**

- **User Registration:** Implement user sign-up with data validation and storage in the database.
- **Login and Logout:** Create authentication processes, including secure session management.
- **Browse Cars:** Develop an interface to list available cars with filtering options.
- **View Car Details:** Provide detailed information for each car, including availability status.
- **Add, Edit, and Delete Cars (Admin):** Allow administrators to manage the car inventory through CRUD operations.
- **Book Car:** Enable customers to reserve cars with a date and time selection.
- **JWT for Security:** Implement JSON Web Tokens for secure and scalable user authentication.
- **Database Schemas:** Design and deploy the database schemas for storing user data, car inventory, bookings, and transactions.

**Metrics:**

- **User Registration:** Track the number of users who sign up.
- **API Performance:** Monitor API response times to ensure system responsiveness.
- **Transactions:** Count the number of transactions processed to measure the uptake of the service.
- **Security:** Confirm the successful implementation of security measures

## Release 2: Frontend Focus - Car Rental 1.1 (June, 2024)

The objective for this release is to optimize the user interface for improved interaction and overall system usability. This focuses on enhancing the customer experience.

**Features:**

- **UI/UX Refinements:** Improve the user interface for the login and registration processes to be more intuitive.
- **Interactive Car Browsing:** Enhance the car browsing experience with better interactivity and detailed views.
- **User Dashboard:** Develop a dashboard for users to manage personal information, bookings.
- **Responsive Design:** Ensure that the application is fully responsive for cross-device compatibility.
- **2FA Integration:** Add two-factor authentication for increased security and integrate it with the user profile.
- **Payment UI:** Streamline the payment user interface within the user profile to make transactions seamless and secure.

**Metrics:**

- **Time Efficiency:** Measure the reduction in the average time taken for users to complete the login and registration.
- **Engagement:** Monitor the increase in the number of users utilizing new features like filters and car details.
- **User Dashboard Interaction:** Gauge user engagement with the dashboard by tracking activities such as bookings and reviews written.
- **Session Quality:** Assess the percentage of sessions without cross-device compatibility issues.
- **Security Adoption:** Evaluate the uptake rate of 2FA amongst users and its correlation with successful transactions.

## 1.3. Project Requirements

### Functional Requirements

**Login**

- As a user, I want to log into my account so that I can access personalized features.
- Acceptance Criteria:
    - The login form is accessible from the homepage and any page via a direct link or a dedicated login button.
    - Upon entering incorrect login credentials, the system provides a clear, non-generic error message that does not specify whether the email or password was incorrect, for security purposes.
    - Successful login redirects the user to the homepage.
    - After logging in, the user's session remains active until they manually log out or the session times out after a predefined period of inactivity.

**Logout**

- As a logged-in user, I want to log out of my account to ensure my account is secure.
- Acceptance Criteria:
    - The logout option is easily accessible in the user interface, requiring no more than two clicks from any page.
    - Upon initiating the logout process, the system immediately invalidates the user's session token, ensuring no further transactions can be processed without re-authentication.
    - The system provides visual confirmation of successful logout, such as a message or redirection to a logged-out state page.
    - After logging out, the user's sensitive information is cleared from the client-side, preventing data leakage.

**Register**

- As a new user, I want to create an account to use the car rental services.
- Acceptance Criteria:
    - User can register using email, password, and required personal information.
    - The system validates data and shows errors for invalid inputs.

**Explore Cars**

- As a user, I want to view a list of available cars so I can choose one to rent.
- Acceptance Criteria:
    - The system presents a user-friendly interface for browsing cars, with options to view them as a list or grid, and detailed cards for each car that include at least a photo, model, and price.
    - Users can access detailed pages for each car with a single click, which provides comprehensive details including specifications, availability, and additional images.
    - The car browsing feature is optimized for performance, ensuring that car images and information load quickly, even on slower internet connections.

**Add Cars**

- As an admin, I want to add new cars to the system to update our inventory.
- Acceptance Criteria:
    - The admin interface includes a clear and intuitive form for adding new cars, with fields for all relevant details such as make, model, year, price, images, and categories.
    - The system supports bulk uploads or an efficient method for adding multiple cars, reducing the time and effort required to update the inventory.
    - Upon submission, the system performs comprehensive validation checks, and any errors in the submitted data are clearly communicated back to the admin for correction.
    - Once a car is successfully added, the admin receives immediate visual confirmation, and the car appears in the relevant listings without requiring a page refresh.

**Delete Cars**

- As an admin, I want to delete cars from the system to keep the inventory updated.
- Acceptance Criteria:
    - Admin can delete a car from the inventory.
    - System asks for confirmation before deletion.

**View Car Details**

- As a user, I want to view details of cars so I can decide which car to rent.
- Acceptance Criteria:
    - Users can click on a car to view its detailed information.
    - The details page includes price, model, specifications, and availability.

**Book Car**

- As a user, I want to book a car for a specific duration.
- Acceptance Criteria:
    - The booking process is streamlined, requiring minimal steps to select dates, enter personal information, and confirm the booking.
    - Real-time availability checks are performed as the user selects dates, with immediate feedback if the car is not available, suggesting alternative dates or cars.
    - Before finalizing the booking, users are presented with a clear summary of their booking details, including the total cost, rental terms, and cancellation policy.

**View Personal Info**

- As a user, I want to view my personal information to verify my account details.
- Acceptance Criteria:
    - Users can access a profile page showing their personal information.
    - The profile page includes name, email, phone number, and booking history.

**Edit Personal Info**

- As a user, I want to edit my personal information to keep my account up-to-date.
- Acceptance Criteria:
    - Users can edit their personal information through the profile page.
    - System validates changes and updates information upon submission.

**Change Password**

- As a user, I want to change my password for security reasons.
- Acceptance Criteria:
    - Users can change their password from the profile settings.
    - System requires current password verification for the change.

**Edit Rental Data**

- As a user, I want to modify my rental booking details in case of changes in my plans.
- Acceptance Criteria:
    - Users can access their booking details and make changes.
    - System updates the booking details and confirms the changes.

**Update Car Details**

- As an admin, I want to update the details of cars in the inventory to ensure the information is accurate and up-to-date.
- Acceptance Criteria:
    - Admins have access to an edit form for each car in the inventory, allowing them to update information such as car model, price, availability, and specifications.
    - The system validates input data to ensure it meets the required format and standards, providing error messages for invalid inputs.
    - Changes are reflected immediately in the system upon submission, ensuring that users see the most current information.
    - The system logs changes made to car details, including what was changed and the timestamp of the change, to maintain a history of updates for audit purposes.

**Add New Location**

- As an admin, I want to add new locations to the system to expand the rental service coverage.
- Acceptance Criteria:
        - The admin interface includes a form for adding new locations with fields for the location name, address, and contact information.
        - The system validates the input data to ensure it meets the required format and standards, providing error messages for invalid inputs.
        - Upon successful addition, the new location is immediately available in the system and visible to users when searching for cars.

**Delete Location**

- As an admin, I want to delete locations from the system to keep the service area updated and remove outdated or closed locations.

- Acceptance Criteria:
    - Admins can select a location to delete from the list of existing locations.
    - The system prompts for confirmation before completing the deletion to prevent accidental removal.
    - Once confirmed, the location is removed from the system, and users no longer see it in searches or listings.

**Get All Users**

- As an admin, I want to view a list of all registered users to manage user accounts and monitor activity.
- Acceptance Criteria:
    - The admin interface provides a comprehensive list of all registered users, including details such as name, email, registration date, and account status.
    - The system allows sorting and filtering of the user list based on different criteria (e.g., registration date, account status).
    - Admins can access detailed profiles of individual users, including booking history and account actions.

**Get All Bookings**

- As an admin, I want to view all bookings to monitor rental activity and manage reservations.
- Acceptance Criteria:
    - The admin interface provides a list of all bookings, including details such as user name, car model, rental dates, and booking status.
    - Admins can access detailed information about individual bookings, including payment status, rental duration, and user contact information.

## Non-Functional Requirements

**User-Friendly Navigation**

- As a user, I expect the website to provide clear and intuitive navigation, allowing me to easily find and access different sections and functionalities without confusion.
- Acceptance Criteria:

    - Key functionalities are accessible within 2-3 clicks from the homepage.
    - Navigation menus are clearly labeled and logically organized.

**Quick Loading**

- As a user, I expect the website pages to load quickly, minimizing wait times and enhancing my overall experience.
- Acceptance Criteria:
    - Pages load within 2-3 seconds on standard internet connections.
    - Loading time is consistently fast across all pages of the website.

**Responsive Design**

- As a user, I expect the website to be easily accessible and navigable on any device, whether I'm using a desktop, tablet, or smartphone.
- Acceptance Criteria:
    - The website adjusts its layout and elements based on the device's screen size.
    - Interactive elements are easily clickable and navigable on touch devices.

**Security with JWT**

- As a user, I expect my data and sessions to be secure, utilizing modern security standards to protect against unauthorized access and data breaches.
- Acceptance Criteria:
    - The system uses JWT for authentication, ensuring secure transmission of data.
    - Sessions are securely managed, with tokens properly expiring and being invalidated upon logout or expiration.

## 1.4. UML diagrams

- **Activity Diagrams**

Login

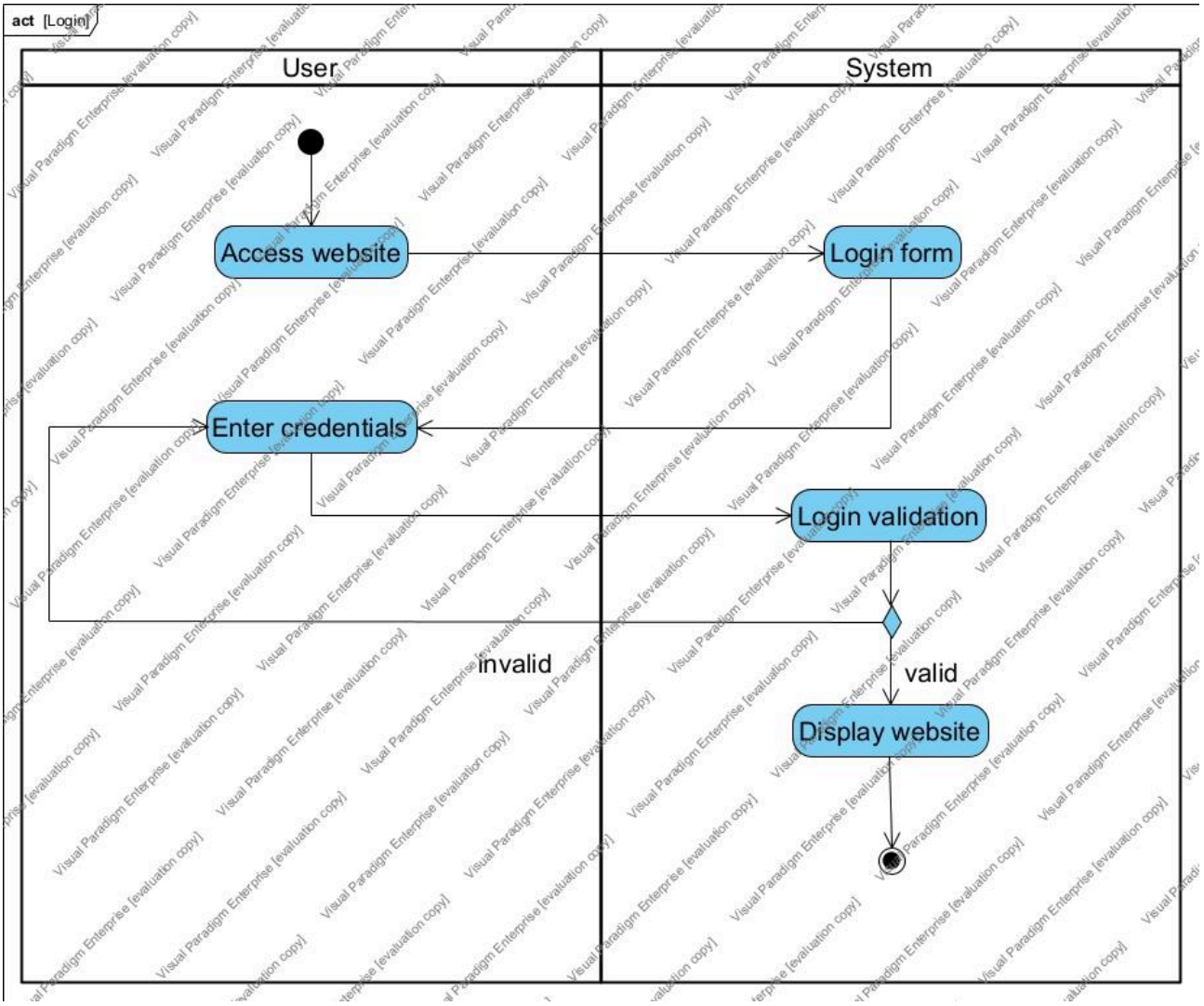


*Figure 1: Activity Diagram for Login functionality*

Explore/Browse Cars

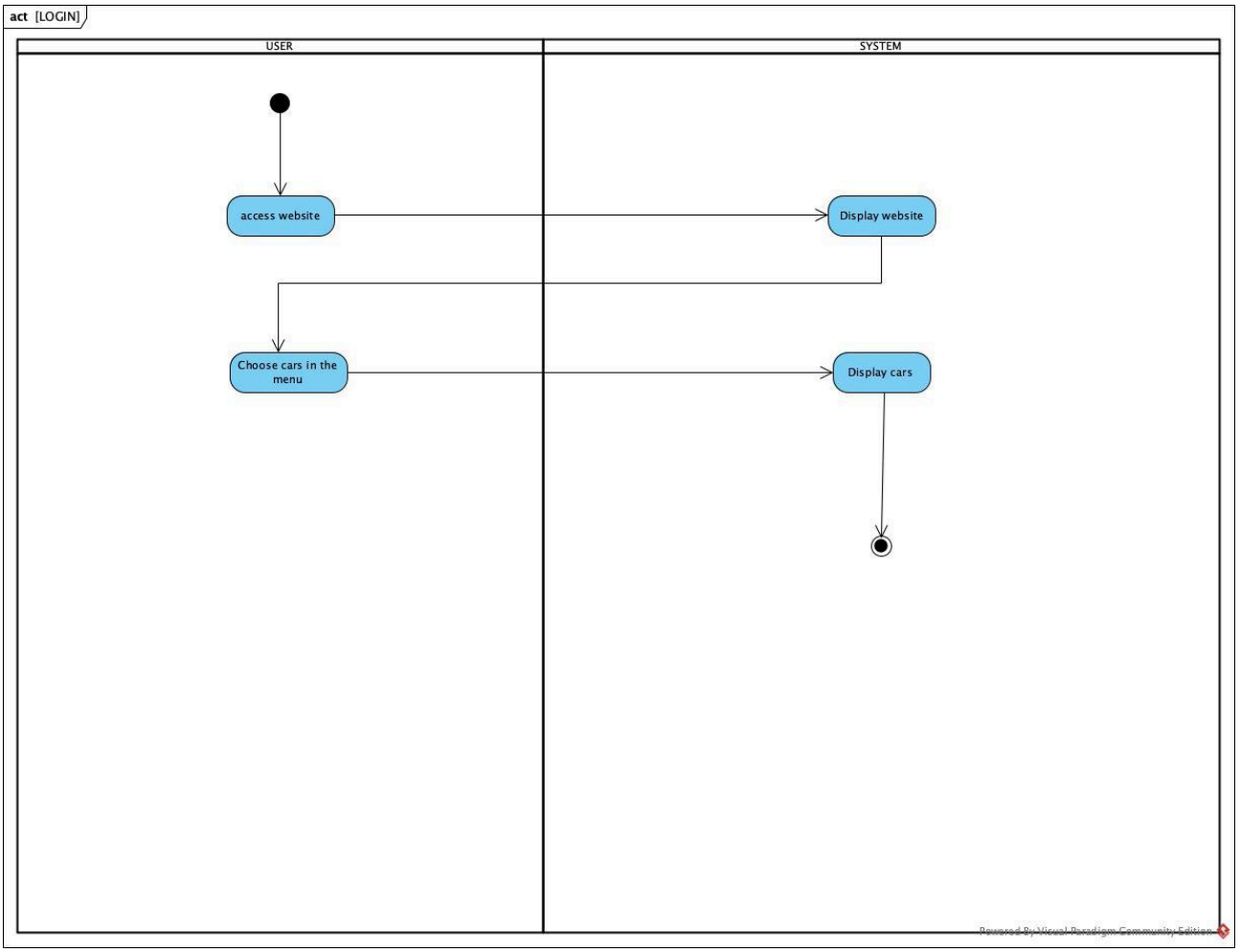


*Figure 2: Activity Diagram for Explore/Browse Cars functionality*

Rent a car

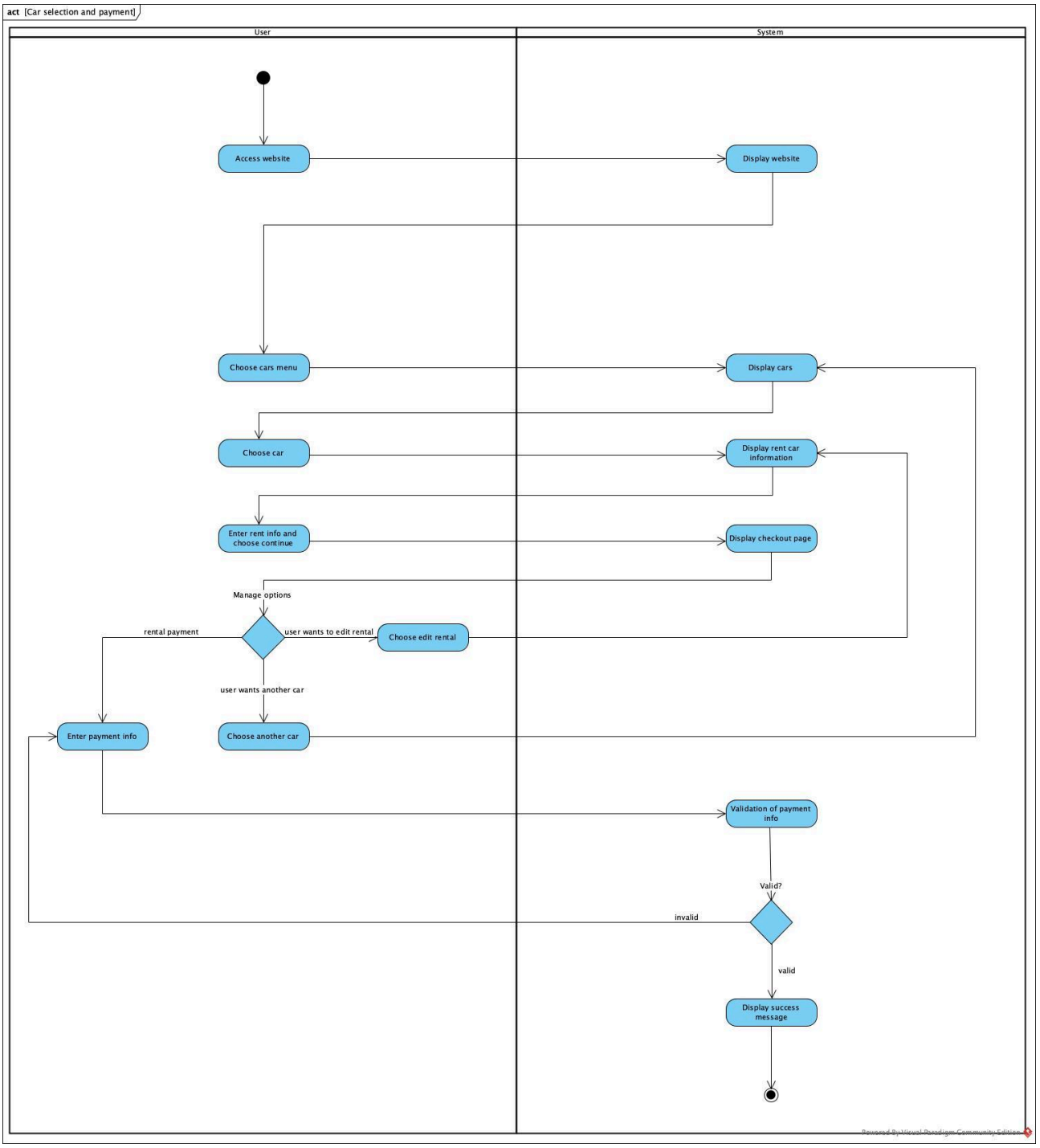


*Figure 3: Activity Diagram for Rent a Car functionality*

## Sequence Diagrams

Login

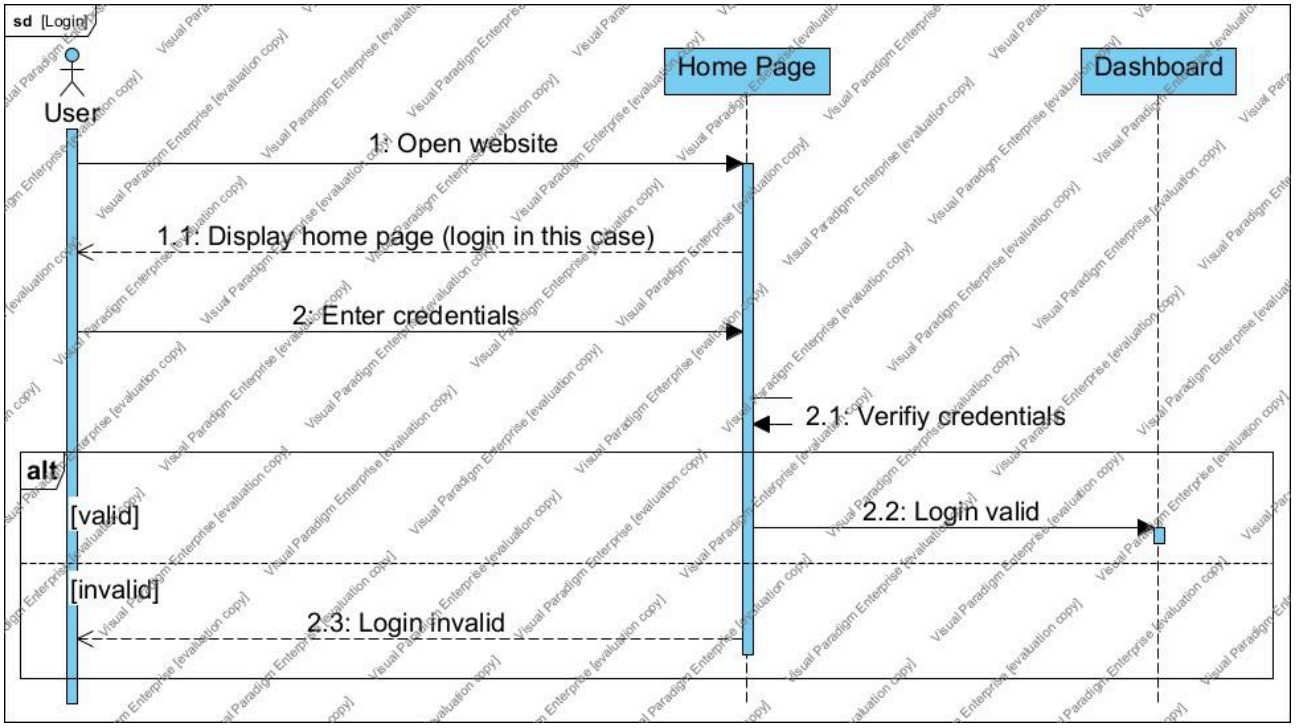


*Figure 4: Sequnce Diagram for Login functionality*

Browse/Explore Cars

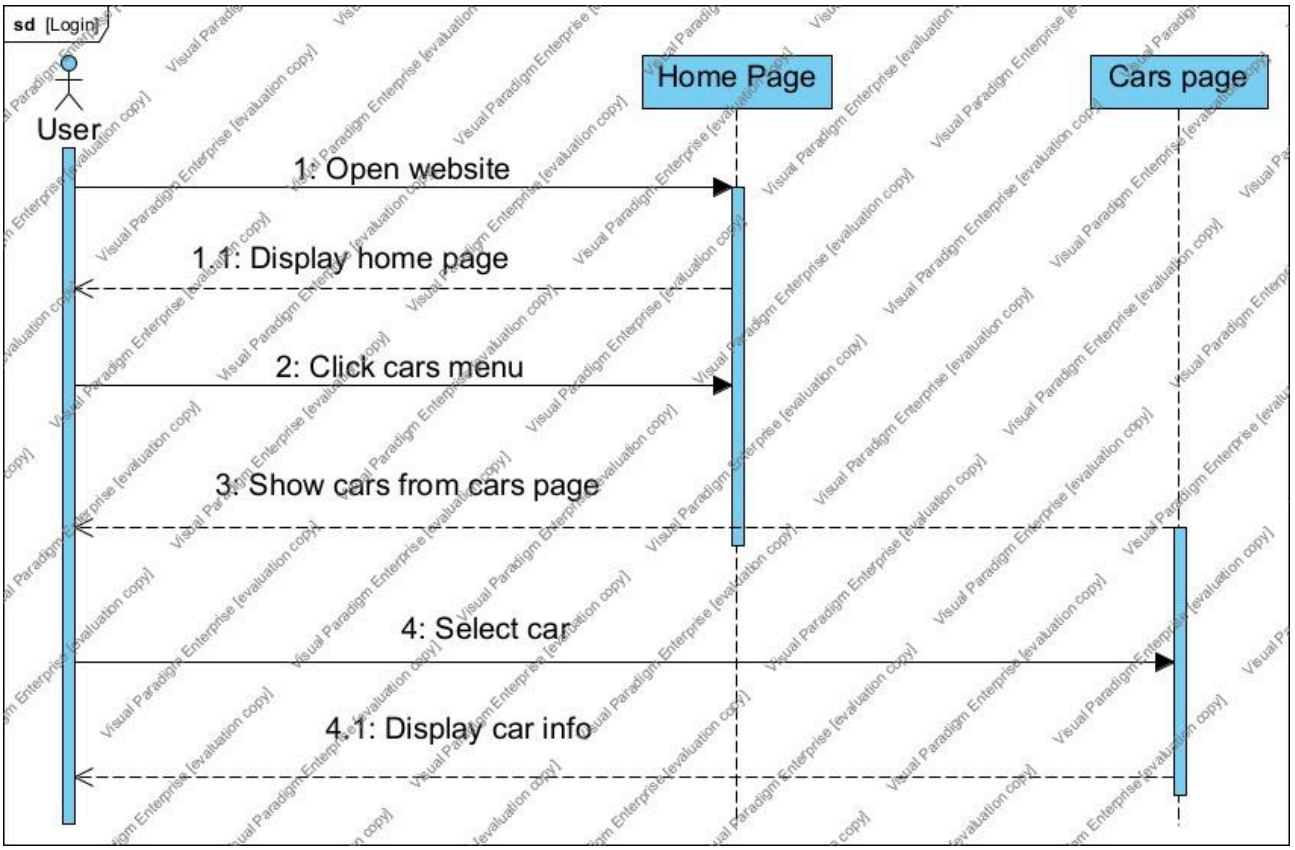


*Figure 5: Sequence Diagram for Explore/Browse Cars functionality*

Rent a car

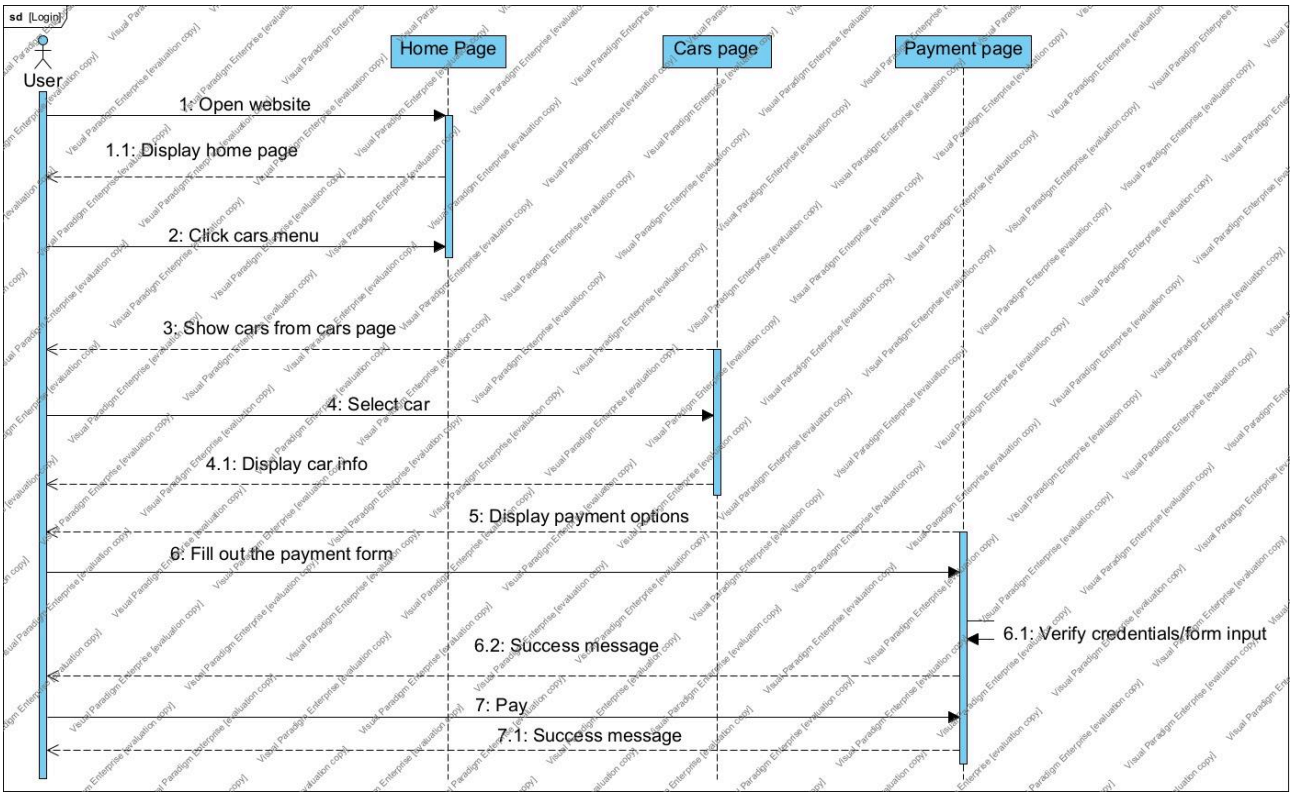


*Figure 6: Sequence Diagram for Rent a Car functionality*

## Class Diagram

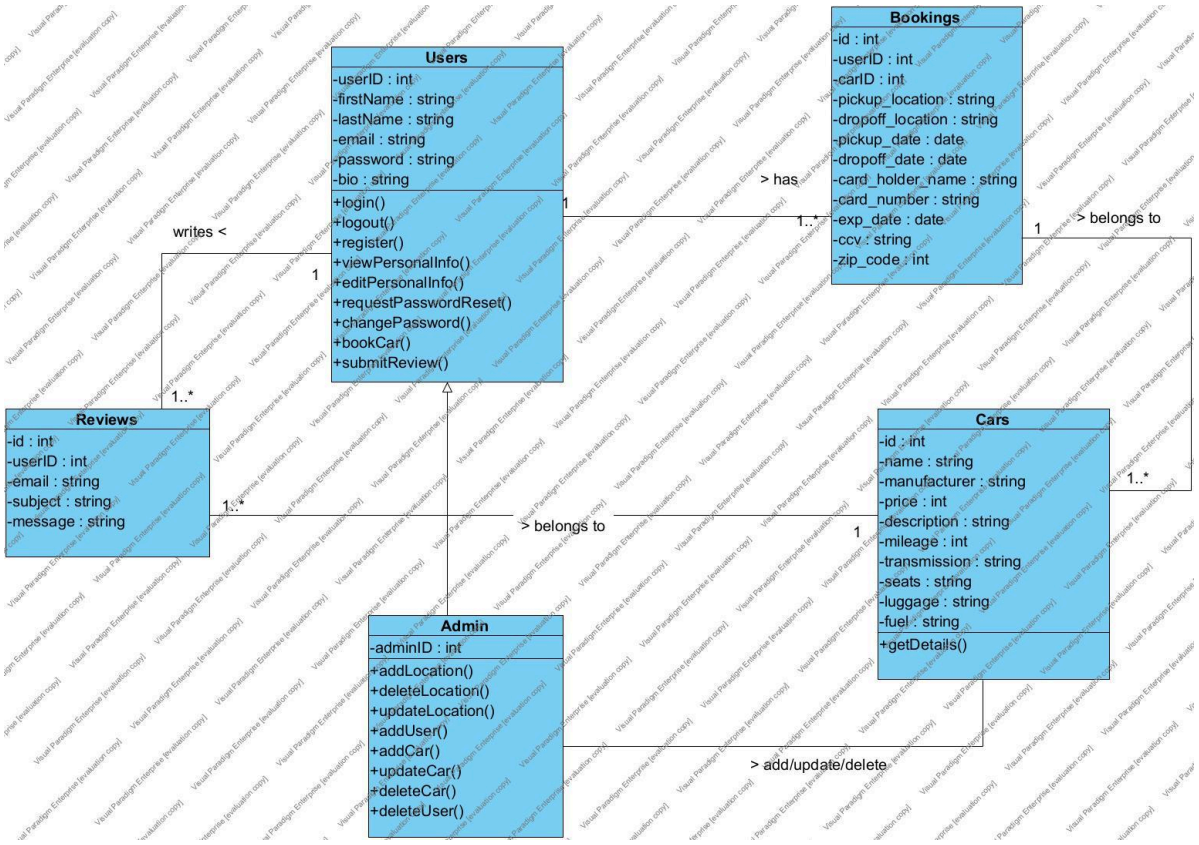


*Figure 7: Class Diagram*

# 2. Project Structure

## 2.1. Technologies

The Car Rental System utilizes a range of technologies to ensure robust functionality, security, and a seamless user experience. Below is a detailed overview of the technologies used for the backend, frontend, and database, as well as other tools integrated into the project.

**Backend:**

- **Flight PHP:** A micro-framework that provides a simple and lightweight structure for building RESTful APIs. It is used for handling server-side logic, routing, and integrating with the database.

**Frontend:**

- **HTML:** Used for structuring the web pages and creating the basic layout of the application.

- **JavaScript:** Provides interactivity and dynamic content updates on the client side. It is essential for handling user actions and making asynchronous requests to the backend.
- **CSS:** Responsible for styling the HTML elements and ensuring a visually appealing interface.
- **SCSS:** A preprocessor scripting language that extends CSS with variables, nested rules, and functions, making the styling process more efficient and maintainable.

**Database:**

- **MySQL:** A relational database management system used to store and manage all application data, including user information, car inventory, bookings, and more. It ensures data integrity and supports complex queries for efficient data retrieval.

**Additional Technologies and Tools:**

- **Bootstrap:** A front-end framework used to design responsive and mobile-first web pages. It helps in creating a consistent and attractive UI with minimal effort.
- **jQuery:** A fast and concise JavaScript library that simplifies HTML document traversal, event handling, and animation.
- **Composer:** A dependency manager for PHP that allows the installation and management of libraries and packages required by the project.
- **phpMyAdmin:** A free tool written in PHP, intended to handle the administration of MySQL over the web. It provides a user-friendly interface for managing the database.

## 2.2. Architectural Pattern

The Car Rental System follows a layered architectural pattern. This pattern is chosen for its clear separation of concerns, making the system more manageable, scalable, and maintainable. Below is an explanation of the architectural pattern and the rationale behind its use, along with a screenshot of the project file organization.

**Layered Architecture**

The layered architecture pattern, also known as the n-tier architecture, divides the application into distinct layers. Each layer has a specific responsibility and communicates with adjacent layers through well-defined interfaces. In this project, the main layers are:

1. **Presentation Layer:** Handles the user interface and user interactions. In this project, it includes HTML, JavaScript, CSS, and SCSS files.

2. **Business Logic Layer (Service Layer):** Contains the core application logic. This layer processes the data and makes decisions. It includes service classes that perform operations such as booking cars, managing users, etc.
3. **Data Access Layer (DAO Layer):** Responsible for interacting with the database. It includes classes that handle CRUD operations on the database.
4. **Routing Layer:** Manages the endpoints and routes of the application. It includes route definitions that map URLs to specific functionalities.
5. **Backend Framework Layer:** Utilizes Flight PHP to handle server-side logic and routing.

The layered architecture was chosen for the following reasons:

- **Separation of Concerns:** Each layer has a distinct responsibility, making the application easier to develop, test, and maintain.
- **Scalability:** Allows for the independent scaling of layers. For example, the data access layer can be optimized separately from the business logic layer.
- **Maintainability:** Changes in one layer (e.g., changing the database) do not affect other layers, reducing the impact of modifications and easing maintenance.
- **Reusability:** Components in each layer can be reused across different parts of the application or even in different projects.

**Project Structure**

**Presentation Layer:**

The presentation layer in our project is responsible for handling the user interface and user interactions. It includes all the components necessary for displaying the data and collecting user input. The main technologies used in the presentation layer are HTML, JavaScript, CSS, and SCSS.

- **assets:**
    - **css:** Contains CSS files for styling the application.
    - **fonts:** Contains font files used in the application.
    - **images:** Contains image files used in the application.
    - **js:** Contains JavaScript files for client-side logic and interactions.
    - **scss:** Contains SCSS files, which are preprocessed into CSS for styling.

**Key Files:**

- **admin.html:** The HTML file for the admin interface.
- **index.html:** The HTML file for the main user interface.
- **pages:** Contains additional HTML files for other parts of the application.
- **login:** Contains the HTML file for the login page and possibly other login-related resources.

The backend-related files in our project are organized as follows:

- **backend:**
  - **public:** Contains public-facing scripts and configuration files.
  - **rest:** This directory includes subdirectories for:
- **dao:** Data Access Objects, responsible for database interactions.
- **routes:** PHP files that define the application's routing logic.
- **services:** Service classes that contain the business logic.
  - **config.php:** Configuration file for database and other settings.

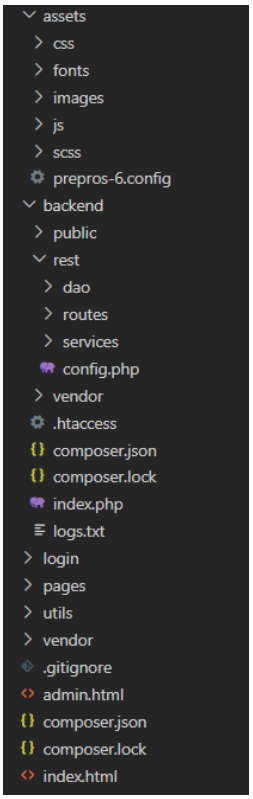


*Figure 8: Project Structure*

## 2.3. Database Entities

The database schema consists of the following entities:

**Users**

Description: This table stores information about the users who can book cars and write reviews.

Fields:

- `userID (int)`: Unique identifier for each user.

- `firstName (string)`: First name of the user.

- `lastName (string)`: Last name of the user.

- `email (string)`: Email address of the user.

- `password (string)`: Encrypted password for the user's account.

- `bio (string)`: A short biography or additional information about the user.

**Cars**

Description: This table stores information about the cars available for booking.

Fields:

- `carID (int)`: Unique identifier for each car.

- `name (string)`: Name or title of the car.

- `manufacturer (string)`: Manufacturer of the car.

- `price (int)`: Rental price of the car.

- `description (string)`: Description of the car including features and conditions.

- `mileage (int)`: Mileage of the car.

- `transmission (string)`: Type of transmission (e.g., automatic, manual).

- `seats (string)`: Number of seats available in the car.

- `luggage (string)`: Luggage capacity of the car.

- `fuel (string)`: Type of fuel the car uses.

**Bookings**

Description: This table stores information about the bookings made by users.

Fields:

- `id (int)`: Unique identifier for each booking.

- `userID (int)`: Identifier linking to the user who made the booking.

- `carID (int)`: Identifier linking to the car being booked.

- `pickup_location (string)`: Location where the car will be picked up.

- `dropoff_location (string)`: Location where the car will be dropped off.

- `pickup_date (date)`: Start date of the booking.

- `dropoff_date (date)`: End date of the booking.

- `card_holder_name (string)`: Name on the credit card used for booking.

- `card_number (string)`: Credit card number.

- `exp_date (date)`: Expiration date of the credit card.

- `ccv (string)`: CCV of the credit card.

- `zip_code (int)`: Billing zip code of the credit card.

**Reviews**

Description: This table stores reviews written by users for cars.

Fields:

- `id (int)`: Unique identifier for each review.

- `userID (int)`: Identifier linking to the user who wrote the review.

- `email (string)`: Email of the user for contact purposes.

- `subject (string)`: Subject or title of the review.

- `message (string)`: Detailed text content of the review.

## 2.4. Design Patterns

In the development of the Rent-A-Car software system, we utilized several design patterns Below is detailed each pattern, where it is applied, and how it benefits the project.

**Data Access Object (DAO) Pattern**

**Usage:** Implemented in all DAO classes (AuthDao.class.php, BookingDao.class.php, CarDao.class.php, etc.).
**Path:** backend/dao/BaseDao.class.php
**Description:** The DAO pattern provides an abstraction layer between the business logic and data access logic in the application. It encapsulates access to data sources which enables loose coupling between the data access operations and the business logic. Each DAO class extends BaseDao.class.php which provides common CRUD operations, thus minimizing code redundancy.
**Why:** This pattern is crucial for isolating the database interaction from the business services, ensuring that changes to the database access logic do not affect the business rules or logic. For example, CarDao.class.php extends BaseDao.class.php and inherits methods like getById or delete, which are then customized for car-specific operations.

**Dependency Injection**
**Usage:** Visible in service classes, such as `CarService.class.php` , BookingService.class.php, etc.
**Path:** backend/services/CarService.class.php
**Description:** Dependency Injection is employed to manage class dependencies. For instance, CarService and BookingService are injected with their respective DAOs. This pattern allows classes to be more independent of their dependencies and simplifies unit testing and maintenance.
**Why:** This approach is used to decouple the service layers from their dependencies on DAOs, enhancing the flexibility and testability of the system. For example, in CarService.class.php, the CarDao is injected into the service through the constructor, allowing for easier changes and better manageability.

**Singleton Pattern**
**Usage:** The Singleton pattern is applied in the configuration management (Config.php).
**Path:** backend/config.php
**Description:** We use the Singleton pattern to manage database and application configurations, ensuring that there is a single instance of the configuration throughout the application. This approach helps in managing shared resources efficiently, such as database connections.
**Why:** Ensuring that configuration settings such as database credentials are loaded and maintained from a single point prevents inconsistencies across the application and reduces memory usage, as shown in Config.php where database settings are fetched statically.

**Factory Method**
**Usage:** This pattern can be inferred from the instantiation of DAOs in service constructors like in CarService.
**File Path:**`backend/services/CarService.class.php`
**Description:** The Factory Method pattern is subtly used to create objects of DAO classes within service classes without specifying the exact class of object that will be created. This makes the system more adaptable to changes with less impact on the clients.
**Why:** It abstracts the instantiation process, allowing the CarService to remain agnostic of the concrete DAO implementations it uses. This is evident in how the CarService initializes CarDao , facilitating the addition of new DAOs without altering service logic.

**Facade Pattern**
**Usage:** Seen in the service layer classes (CarService, BookingService, etc.).
**Path:** backend/services/CarService.class.php
**Description:** The Facade pattern simplifies the interface presented to clients. It provides a unified interface to a set of interfaces in a subsystem. For example, the CarService class provides a simple

interface to perform operations on car data which internally uses more complex logic in the DAO layer.
**Why:** This pattern is applied to shield client layers from the complexities of the underlying DAO operations. CarService offers methods like getCarById, which internally may involve complex SQL queries handled by CarDao, thereby simplifying client interactions.

**Template Method**
**Usage:** Demonstrated in the BaseDao class with methods like add, delete, update.
**File Path:** backend/dao/BaseDao.class.php
**Description:** The Template Method pattern defines the skeleton of an algorithm in an operation, deferring some steps to subclasses. `BaseDao` provides the template method for CRUD operations while allowing subclasses to define certain details.
**Why:** This pattern ensures that common database operations are executed in a standardized manner while still allowing for specific details to be handled differently by each DAO subclass. For instance, the update method in BaseDao sets up the SQL update query framework, but the actual columns and values are specified by the subclasses tailored to specific database tables.

## 2.5. Project Functionalities and Screenshots

### Registration

The registration process allows users to create a new account by providing the necessary information such as email, password, name and surname. Once the user submits the registration form, the application validates the input data and creates a new user account in the system.

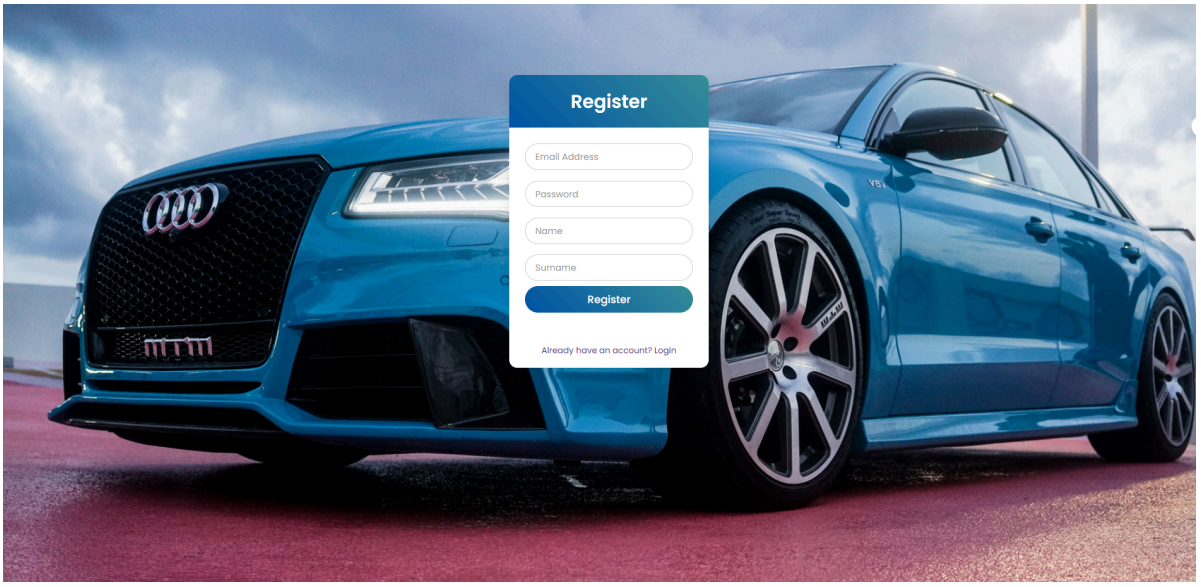


*Figure 9: Registration Screen*

### Login

The login functionality enables registered users to authenticate themselves and gain access to the application's features. Users can enter their email and password credentials, which are

then verified against the stored user information. Upon successful login, the user is granted access to the application's main interface.

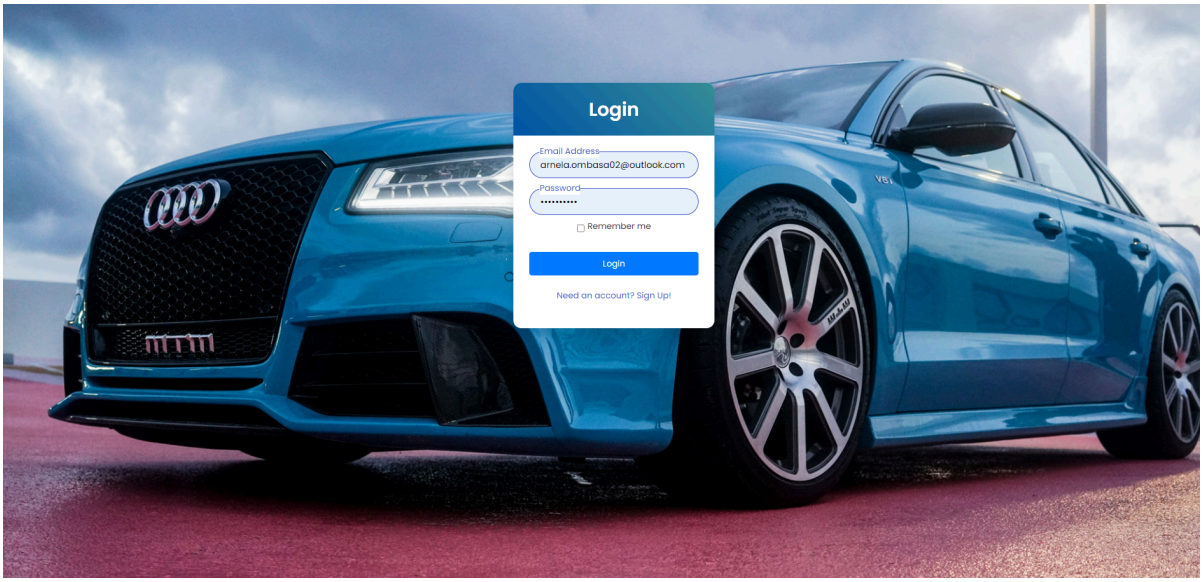


*Figure 10: Login Screen*

## Homepage

After successfully login, the user is redirected to the homepage of the application. On the homepage, the user can enter their desired location and date information to search for available cars. Once the user clicks the "Rent a Car" button, the application will display a list of cars that match the user's search criteria, along with their details and availability.

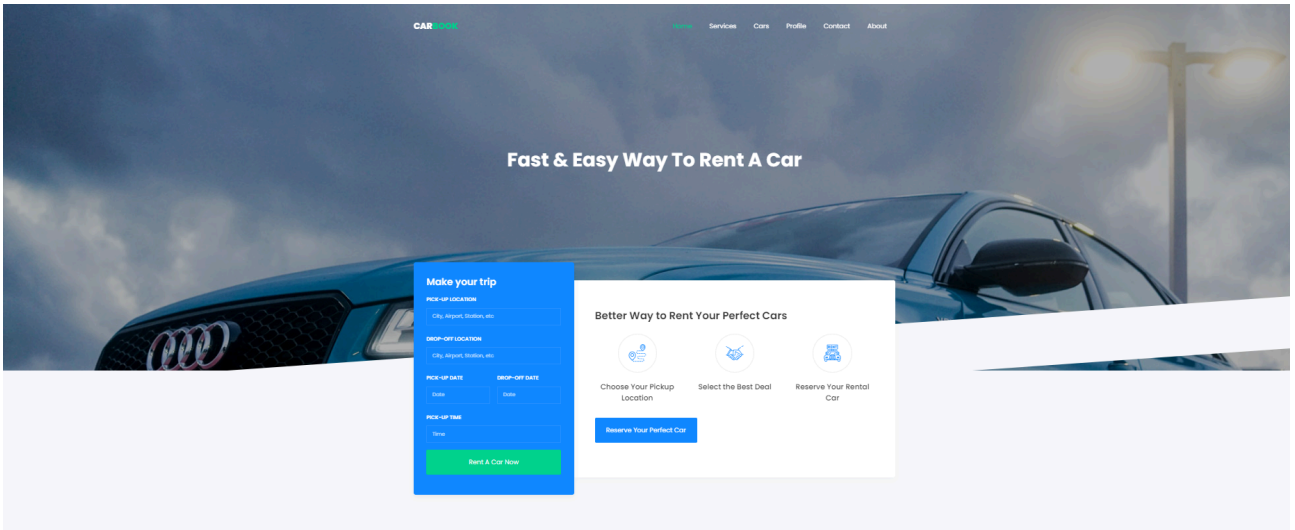


*Figure 12: Homepage Screen*

## Cars Page

The Cars page shows the user a list of all the cars available for booking.

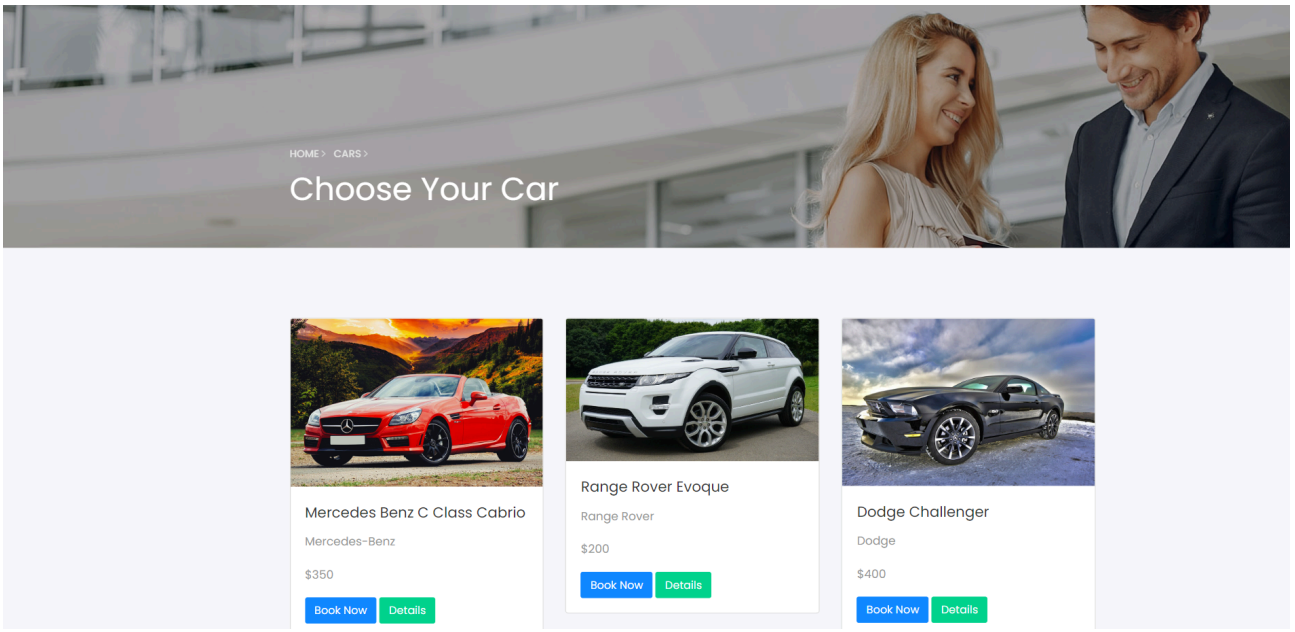


*Figure 13: Cars Page*

For each car, the user can can  perform the following actions:

**View Car Details:** The user can click on a car to view more detailed information about the vehicle, including its specifications, features, and images.

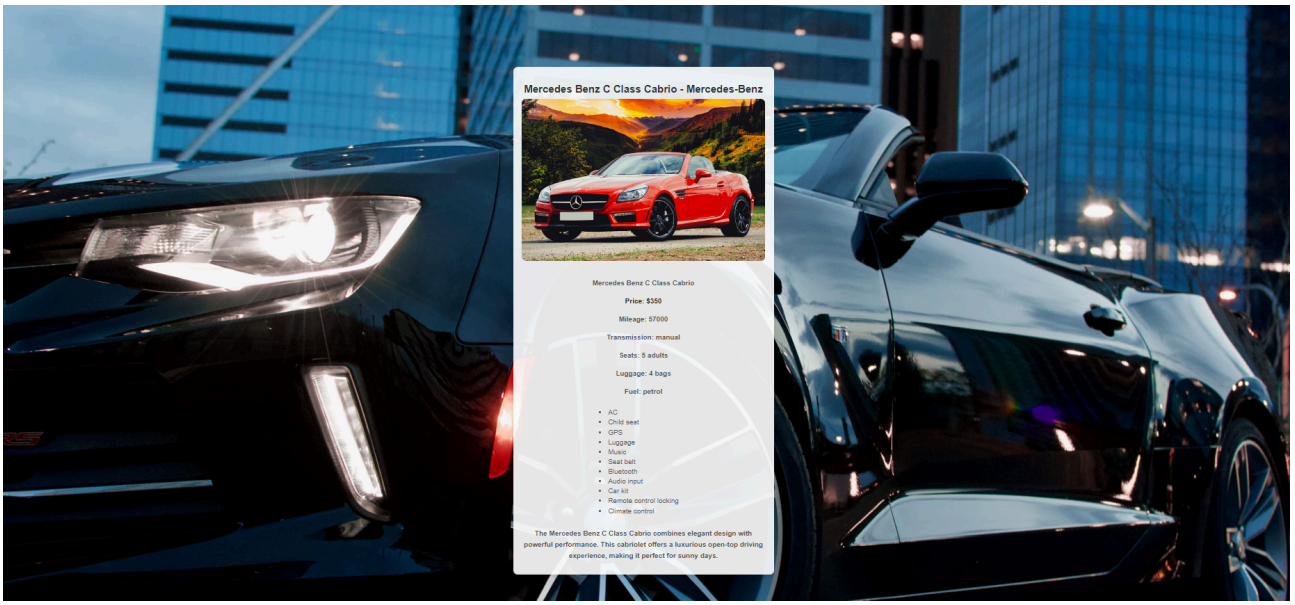


*Figure 14: Car Details Screen*

**Book Car:** The user can select a car and proceed to the booking process to reserve the vehicle for their desired dates.

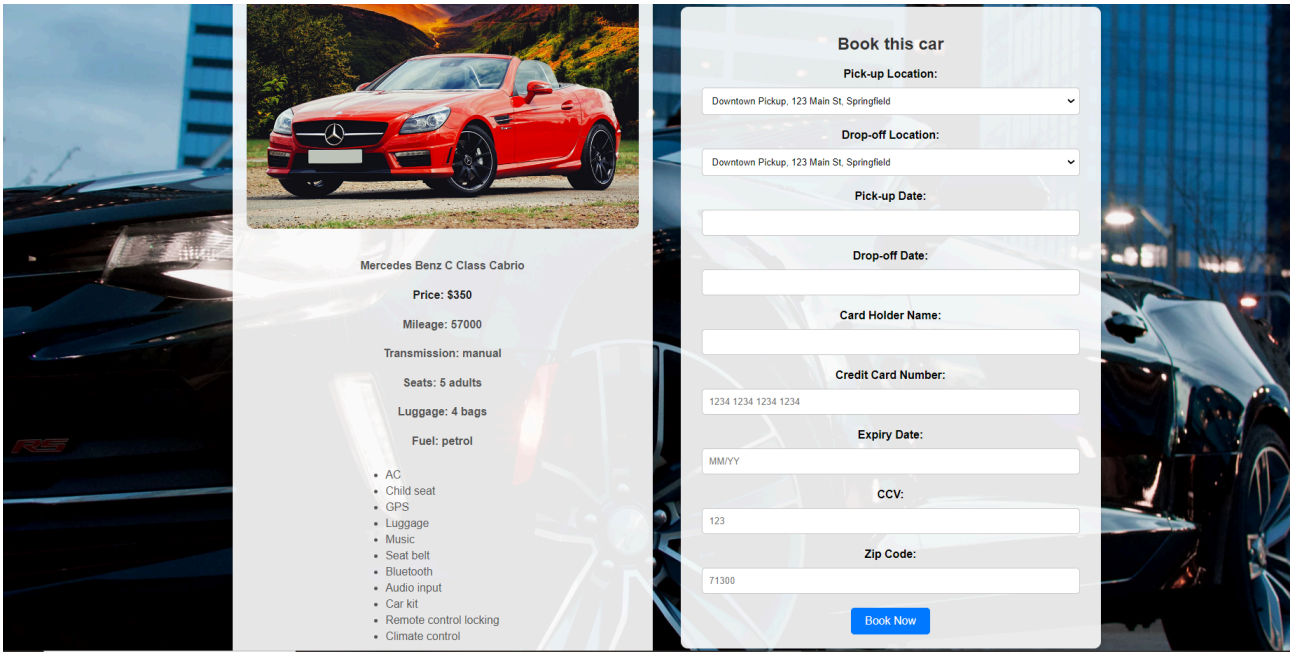


*Figure 15: Book Car Screen*

## User Profile Page

The User Profile page allows the user to view and edit their account informationThe user can also log out of the application from this page.

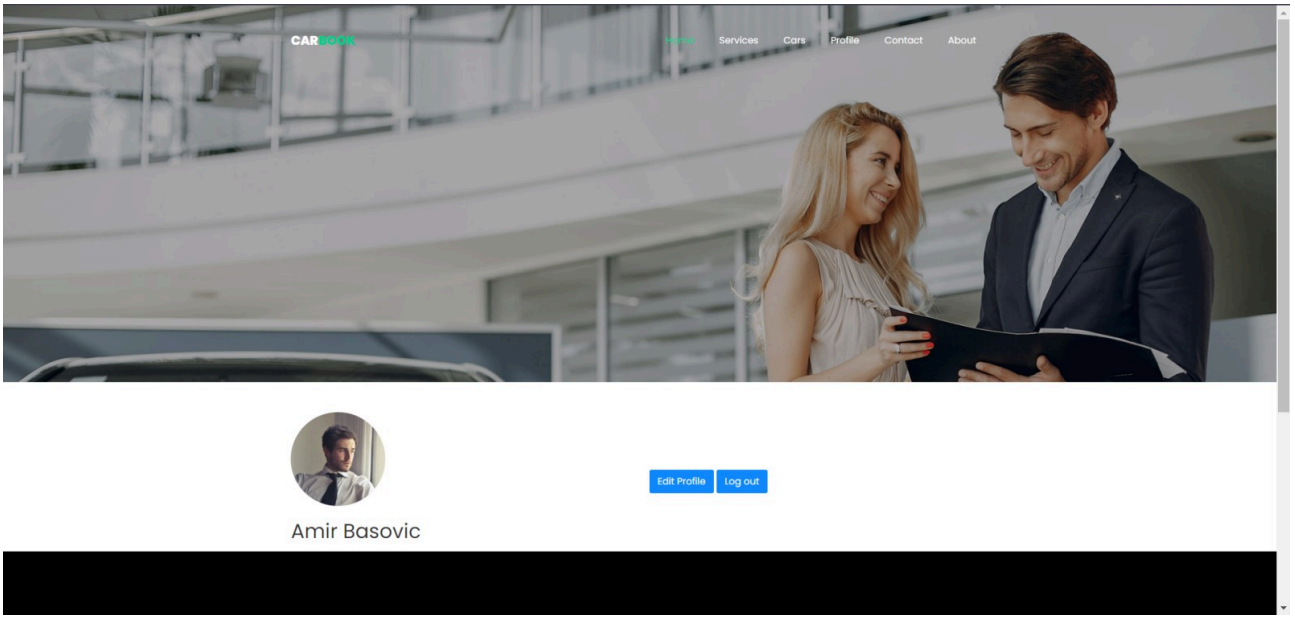


*Figure 16: User Profile Screen*

When the user clicks on the "Edit Profile" button, the page displays two forms:

### Edit User Data

This form allows the user to update their personal information. The user can enter the new data and click "Save Changes" to update their profile.

### Change Password

This form enables the user to change their account password. The user needs to enter their current password, new password, and confirm the new password. After filling out the form, the user can click "Change Password" to update their login credentials.

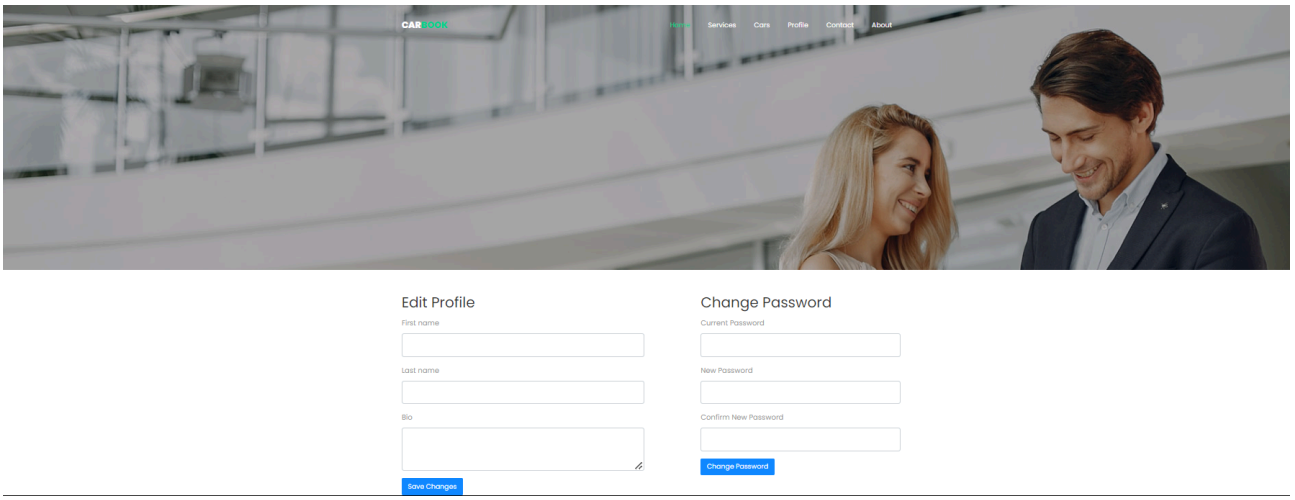


*Figure 17: Edit Profile Screen*

### Contact Page

The Contact page allows users to leave reviews or comments about the application. This page includes a form with the following fields:

*Name* - The user can enter their name in this field.

*Email* - The user needs to provide their email address in this field.

*Subject* - The user can enter a subject or title for their message.

*Message* - This is a text area where the user can type their review or comment.

After filling out the form, the user can click the "Send message" button to send their feedback. The application will then process the user's input and store the review or comment in the database for further analysis and management.

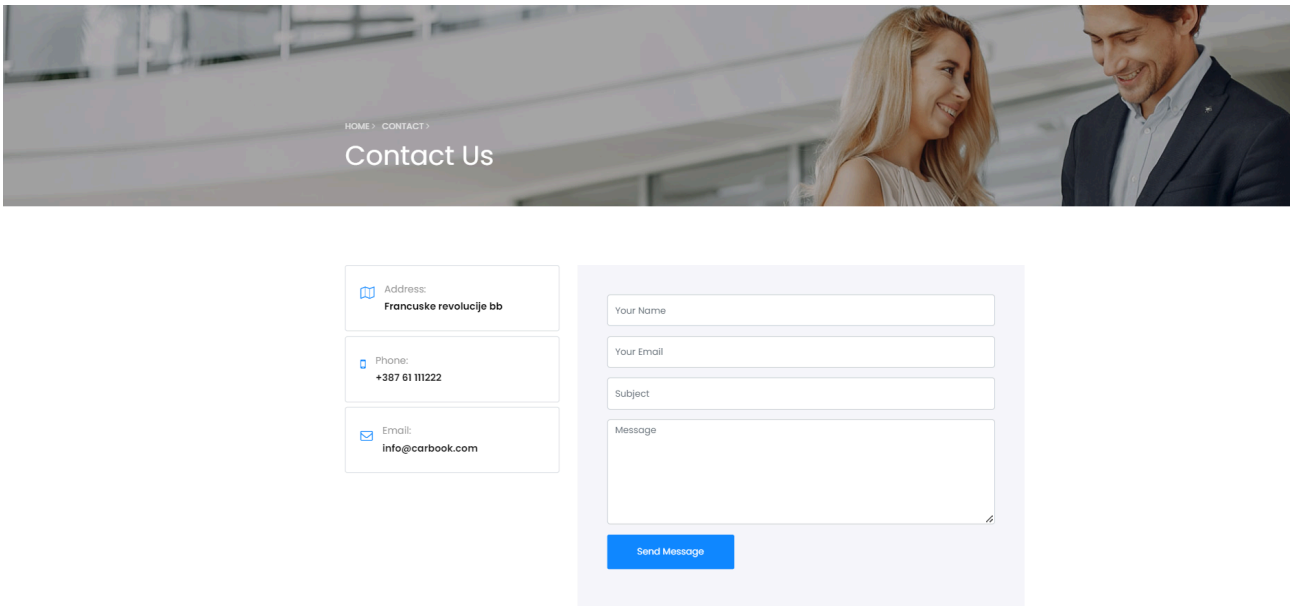


*Figure 18: Contact Us Screen*

## About Us Page

On our About Us page, not only can you learn more about the core values and vision that drive our operations, but you can also easily navigate to Cars page. Simply click the "Search Vehicles" button below to start your journey and choose the perfect car that meets your needs.

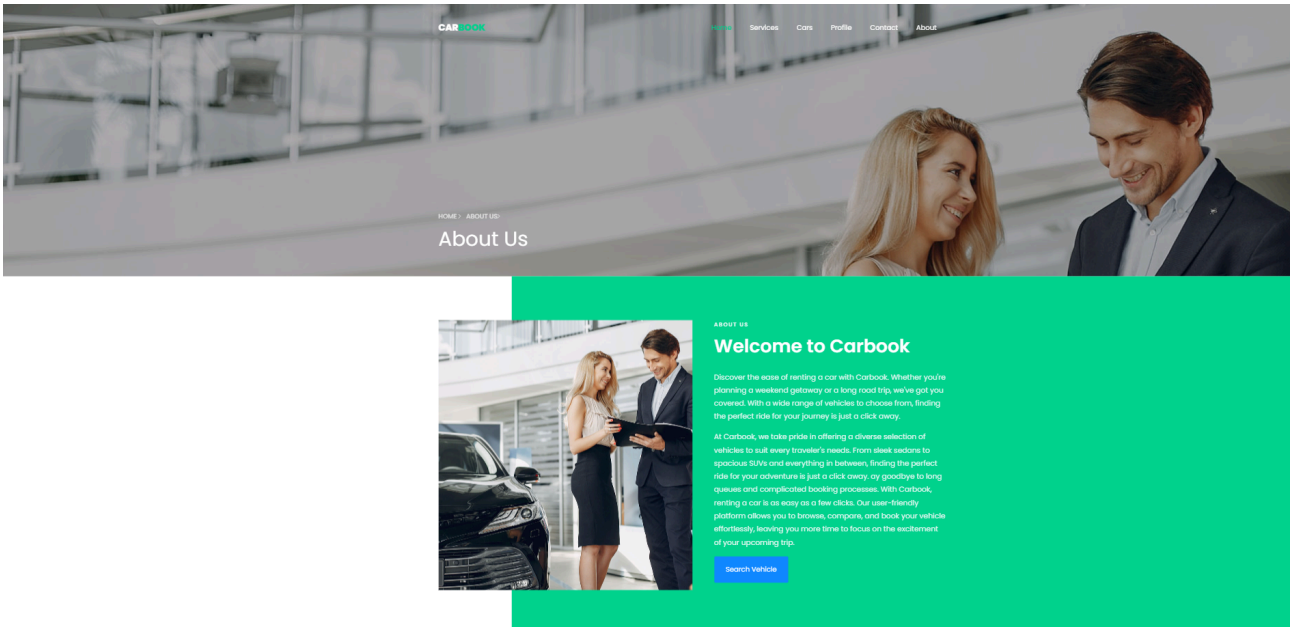


*Figure 19: About Us Screen*

Admin Page

The Admin Page is a crucial component of the application, designed to provide administrators with a comprehensive control panel for managing various aspects of the car rental system. This page is accessible only to users with administrative privileges and is crafted to ensure efficient management of the application's data and user interactions.

Functionalities of the Admin Page:

1. **Add New Car**
    - Administrators can add new cars to the system by entering car details. This function ensures that the car inventory is up-to-date and diverse to meet user demands.
2. **Delete Car**
    - This function allows administrators to remove cars from the listing that are no longer available or fit for rental, helping maintain a fresh and appealing car inventory.
3. **Update Car**
    - Administrators can update existing car information to reflect any changes in specifications, availability, or conditions, ensuring that the information provided to users is accurate and reliable.
4. **Add New Location**
    - As Carbook expands, new rental locations can be added to the system through this functionality, widening the scope of service and convenience for users.
5. **Delete Location**
    - This function is used to remove outdated or non-operational rental locations from the system, which helps in maintaining accurate and relevant service areas.
6. **Update Location**
    - Administrators can update details about the rental locations to reflect changes like address modifications or service offerings, ensuring users have access to correct and useful information.
7. **Get All Users**
    - This functionality provides administrators with a list of all registered users, allowing for a broad overview and the ability to monitor user activity within the application.
8. **Add New User**
    - Administrators can manually add new users to the system, useful for setting up accounts for staff members or for special customer service situations.
9. **Delete User**
    - This function allows for the removal of users who violate the application's terms of service or request account deletion, maintaining a healthy user environment.

10. **Get All Bookings**
    - Administrators can view all car bookings, which aids in tracking vehicle availability, user activity, and rental patterns, facilitating better management and planning.

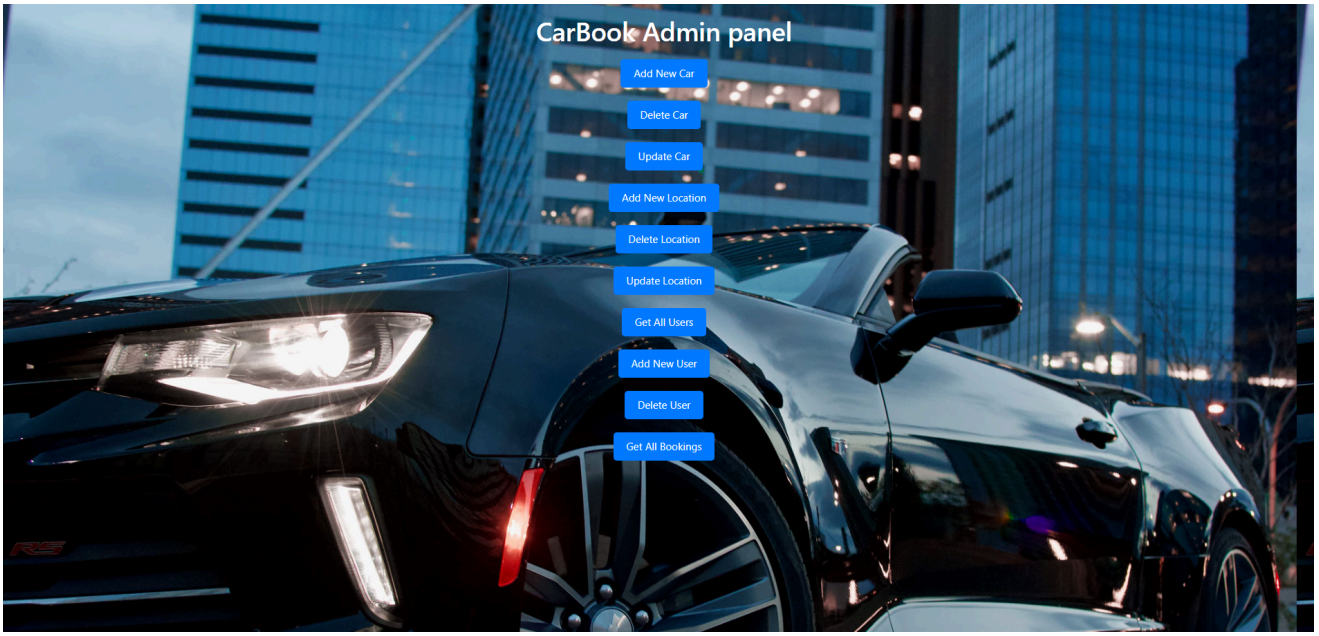


*Figure 20: Admin Panle Screen*

Examples of the add, update and delete forms are shown on the figures bellow.

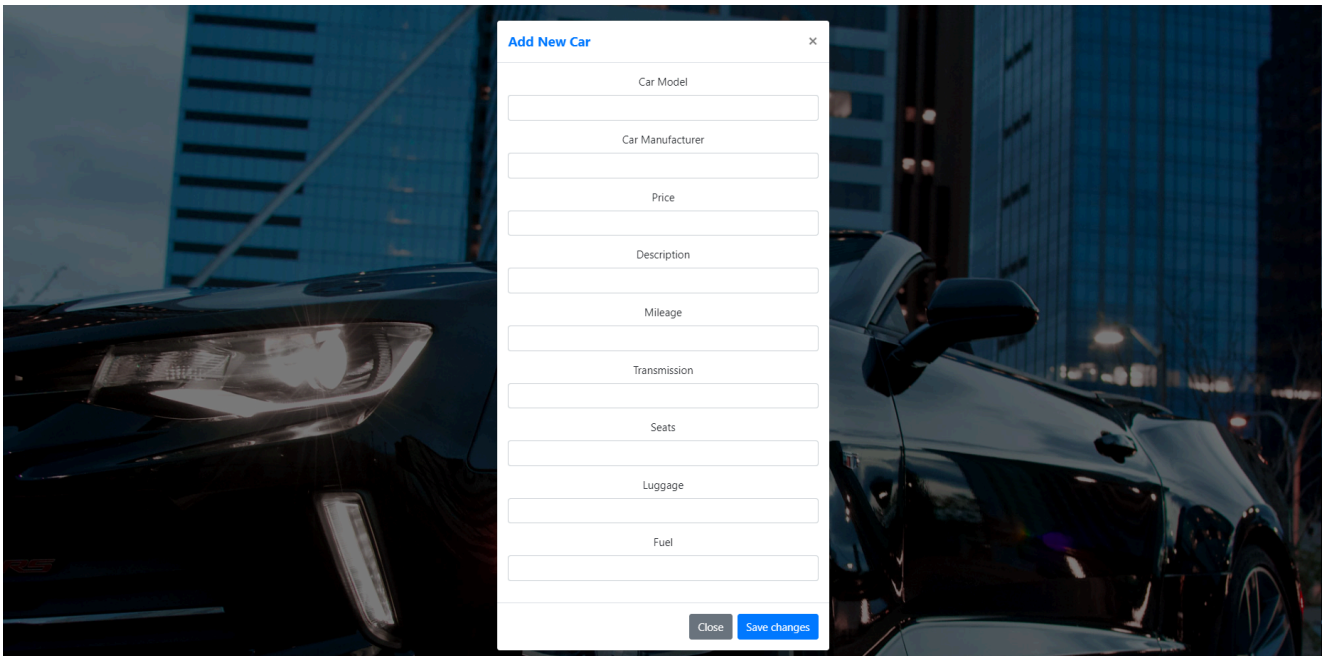


*Figure 21: Add form*

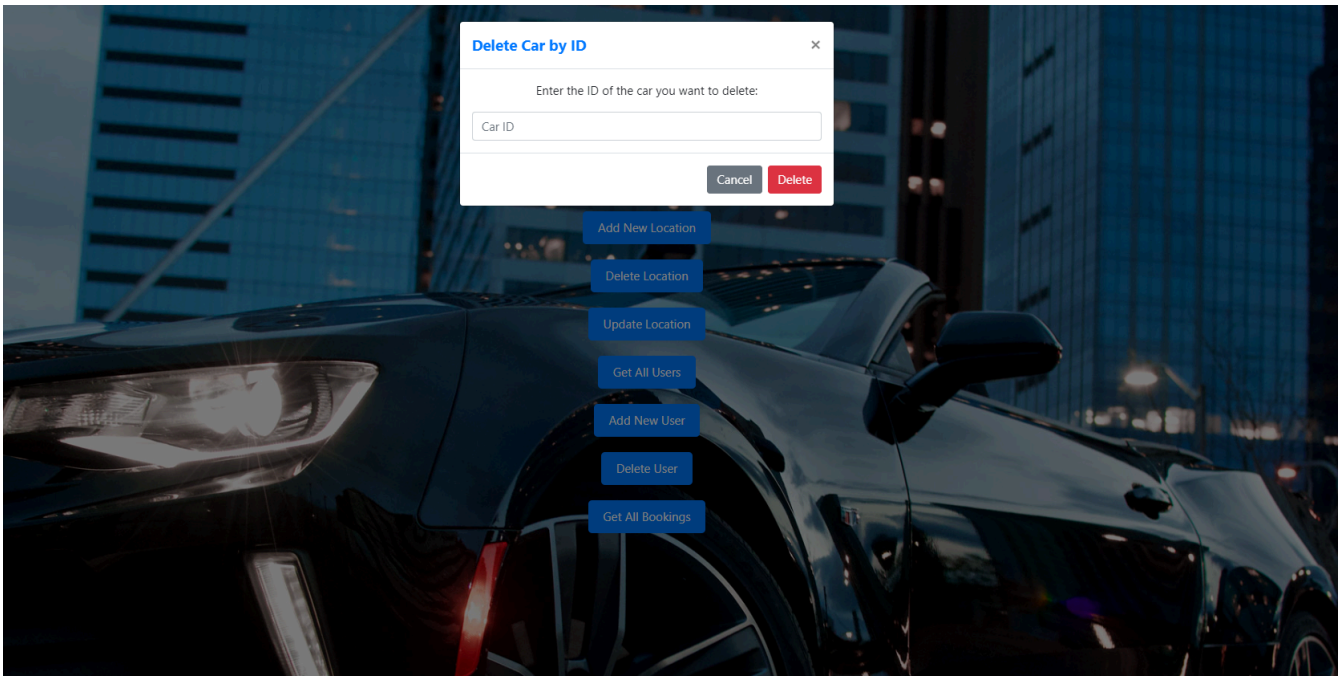


*Figure 22: Delete form*

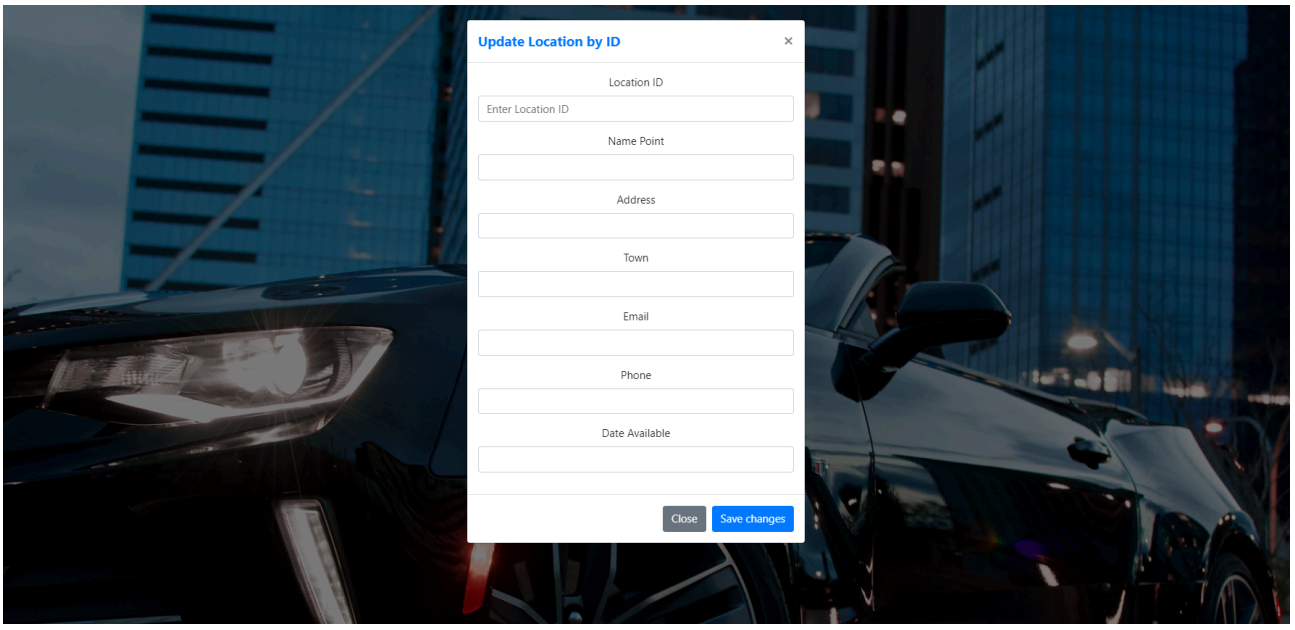


*Figure 23: Update form*

## 2.6. Tests

**Types of Tests:** Unit Tests

Unit tests are designed to test individual components or units of the application in isolation from the rest of the codebase. They are critical for verifying that each part of the application performs as intended.

**Example Unit Test: CarServiceTest**

The CarServiceTest class is a unit test for the CarService class. It uses PHPUnit, a popular testing framework for PHP, to test the functionality of the car service logic.

**Tests Descriptions:**

- **testGetCarById()**
    - **Purpose:** To verify that the get_car_by_id method retrieves the correct car details for a given car ID.
    - **Setup:** Mocks the CarDao and simulates a return value when the get_car_by_id method is called.
    - **Verification:** Asserts that the car details returned by the CarService match the expected car data.

- **testAddCar()**

    **Purpose:** To ensure that the add method correctly adds a new car and returns the car data with an assigned ID.
    **Setup:** Mocks the CarDao and simulates the return value when the add method is called.
    **Verification:** Asserts that the result contains an ID and the other car data fields match the expected values.

- **testUpdateCar()**
    - **Purpose:** To check that the update_car method updates the car information correctly for a given car ID.
    - **Setup:** Mocks the CarDao and expects the update_car method to be called with the correct parameters.
    - **Verification:** Ensures that the update_car method is called once with the correct car ID and data.

These unit tests are crucial for maintaining the integrity and reliability of the CarService class, ensuring that the core functionalities such as retrieving, adding, and updating cars work as expected.

They are located in **RentACar-SoftwareEngineering/tests/CarServiceTest.php.**

## 3. Conclusion

Implementing the Car Rental System was an enriching and insightful experience. This project not only allowed us to apply theoretical knowledge in a practical setting but also challenged us to solve real-world problems using modern web technologies.

Overall, we are satisfied with the implementation of the project. The system is functional, user-friendly, and secure, meeting the primary objectives we set out to achieve. The backend, built with Flight PHP, efficiently handles all server-side operations, while the frontend, designed with HTML, JavaScript, CSS, and SCSS, provides a smooth and responsive user experience. MySQL serves as a reliable database for managing all essential data.

However, as with any project, there are areas for improvement. In the future, we could enhance the user interface further by incorporating more advanced JavaScript frameworks like React to make the application even more interactive and dynamic. Additionally, implementing automated testing for both the frontend and backend could help in maintaining the code quality and catching bugs early in the development process. One of the most challenging aspects of the project was ensuring robust security measures.

In conclusion, this project provided a valuable opportunity to develop a full-stack web application, improve our coding skills, and understand the intricacies of software engineering.