CAR RENTAL SYSTEM :-Using SpringBoot & React and MySql

A Project Synopsis Report

Submitted By:

Aniket Chugh(23BCS12407)

in partial fulfilment for award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE ENGINEERING

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

BONAFIDE CERTIFICATE

Certified that this project report **"CAR RENTAL SYSTEM Using SpringBoot , MySql & React "** is the bonafide work of **" Aniket Chugh"** who carried out the project work under my/our supervision.

SIGNATURE **SIGNATURE**

Dr. Sandeep Singh Kang Er. Deep Prakash Gupta

HEAD OF THE DEPARTMENT **SUPERVISOR**

Submitted for the project viva-voce examination held on 06 November 2025.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

This project implements a scalable, secure, and robust backend service for a Car Rental System. The primary objective is to develop a RESTful API that manages vehicle inventory, user registration, and a multi-stage booking process. The system is built entirely in Java using the Spring Boot framework, designed for seamless integration with a modern single-page application (SPA) frontend like React.

The application utilizes a Layered Architecture (Controller-Service-Repository) for a strict separation of concerns, ensuring maintainability and testability. It leverages Spring Security for role-based user authentication and authorization (JWT-based) and Spring Data JPA for robust database interaction with MySQL.

Key Components

- AdminController.java **(REST API):** Exposes admin-only endpoints for fleet management (e.g., POST /api/admin/car, DELETE /api/admin/car/{id}) and booking management (e.g., GET /api/admin/car/bookings, GET /api/admin/car/booking/{bookingId}/{status}).
- CustomerController.java **(REST API):** Exposes customer-facing endpoints for browsing cars (e.g., GET /api/customer/cars), booking a car (e.g., POST /api/customer/car/book), and viewing personal booking history (e.g., GET /api/customer/my-bookings/{userId}).
- AuthController.java **(Security):** Handles public endpoints for user registration (POST /api/auth/signup) and login (POST /api/auth/login), issuing JSON Web Tokens (JWTs).
- AdminService.java / CustomerService.java **(Business Logic):** Contains the core intelligence, separating logic for administrative tasks (posting cars, approving/rejecting bookings) from customer tasks (browsing, creating pending bookings).
- SecurityConfiguration.java / JwtAuthenticationFilter.java**:** Manages security, rolebased access (ROLE_ADMIN, ROLE_CUSTOMER), and JWT validation.
- **Repositories:** Uses Spring Data JPA repositories (CarRepository.java, UserRepository.java, BookACarRepository.java) for all data persistence.

Data Model

The system relies on three core entities managed by JPA:

- User.java**:** Manages user data, including email, password (hashed), and UserRole (e.g., ROLE_USER, ROLE_ADMIN).
- Car.java**:** Stores vehicle details, including make, model, year, daily price, transmission, color, brand, and an image stored as a byte[].
- BookACar.java**:** Records all rental transactions, linking a User to a Car with fromDate, toDate, total price, calculated days, and a BookCarStatus (e.g., PENDING, APPROVED, REJECTED).

This project demonstrates the effective use of Java, Spring Boot, and Spring Security to create a maintainable, high-performance, and contract-driven backend solution suitable for a production-level car rental platform.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1. Identification of Client

The primary clients for this Car Rental System are twofold:

1. **Rental Custom**The need stems from the modern "on-demand" economy, where consumers expect immediate, digital-first solutions. This project addresses the contemporary issue of digitizing traditional, often manual, rental agency processes.**ers (Users):** Individuals seeking a convenient, self-service platform to browse available vehicles, compare prices, and request to book a car for a specific duration. They require a simple, secure, and responsive interface, which is provided by the frontend SPA that consumes our API.

2. **System Administrators (Agency Staff):** Employees of the car rental agency who need a secure backend interface to manage the vehicle fleet, update car details, view all booking requests, and manage (approve or reject) those bookings.

1.2. Identification of Problem

The traditional car rental industry often relies on manual, paper-based, or outdated booking systems. This leads to several significant problems:

- **Lack of Real-Time Availability:** Customers cannot see if a vehicle is *actually* available without calling or visiting, leading to frustration.
- **Inefficient Fleet Management:** Admins struggle to track which cars are rented, in maintenance, or available, leading to overbooking or underutilization.
- **Manual Data Entry:** Prone to human error, insecure, and slow.
- **Scalability Issues:** Manual systems cannot handle a sudden influx of customers or a growing fleet of vehicles efficiently.

This project addresses the problem of creating a centralized, digital, and automated platform for managing car rentals that is secure, scalable, and provides real-time data to both customers and administrators.

1.3. Identification of Tasks

1. **Security Implementation (Spring Security):**
    - Set up user registration and login functionality using AuthController.
    - Implement JWT (JSON Web Token) for stateless authentication.

- Create a JwtAuthenticationFilter to validate tokens on every request.
- Configure SecuTo develop the Car Rental System backend, the following key tasks were identified and executed:

1. **Data Modeling and Database Design (JPA):**
    - Define the core entities: User.java (for security), Car.java (for inventory), and BookACar.java (for transactions).
    - Establish the relationships between entities (e.g., User to BookACar, Car to BookACar).
    - Define Enums for UserRole and BookCarStatus.

2. rityConfiguration to secure endpoints based on roles (ROLE_ADMIN, ROLE_CUSTOMER).

2. Business Logic Implementation (Quiz Service):

- Develop AdminService to handle posting cars, updating cars, deleting cars, getting all bookings, and changing booking status.
- Develop CustomerService to handle browsing/filtering cars, fetching a single car, creating a PENDING booking, and fetching a user's personal booking history.
- Implement cost calculation logic based on daily rates and rental duration.

3. REST API Endpoint Definition (Controller Layer):

- Define the external API contract using @RestController for AuthController, AdminController, and CustomerController.
- Create clear, resource-based endpoints (e.g., /api/admin/car, /api/customer/car/book) that handle HTTP requests and DTOs.

4. Project Setup and Integration:

- Implement robust input validation for DTOs (e.g., SignupRequest, BookACarDto).
- Create a global exception handler (e.g., for ResourceNotFoundException) to return meaningful error messages to the API consumer.

1.4. Timeline

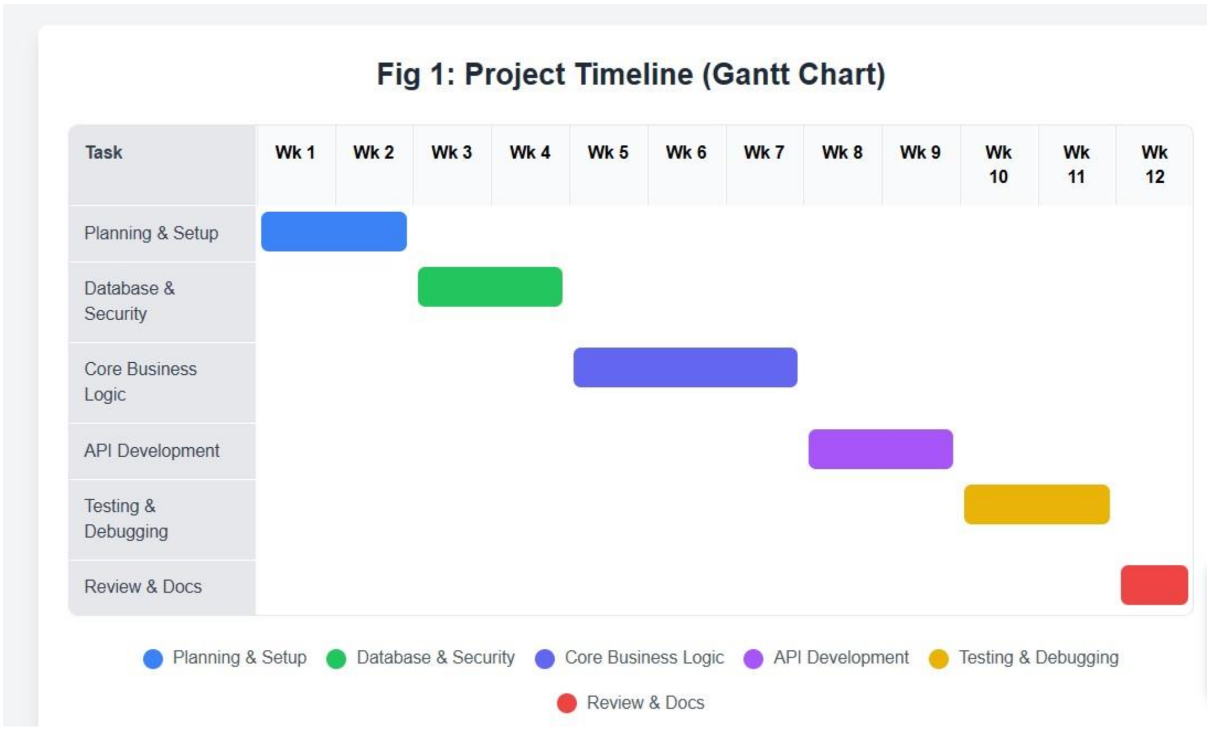


Fig 1 Timeline of the project making.

1.5 Organization of Report

This report is structured to provide a comprehensive overview of the Car Rental System project, from its conception to its implementation and validation.

- **Chapter 1 (Introduction):** Establishes the project's context, identifies the core problem, and outlines the tasks and timeline.
- **Chapter 2 (Literature Review):** Discusses existing solutions, justifies the chosen technology stack (Spring Boot, MySQL), defines the problem scope, and lists the project's objectives.
- **Chapter 3 (Methodology):** Details the key algorithms (booking, pricing) and the stepby-step implementation process following the layered architecture.
- **Chapter 4 (Result Analysis):** Presents the final implemented solution, analyzes the API's functionality, and provides validation through test results.
- **Chapter 5 (Conclusion):** Summarizes the project's achievements and suggests potential enhancements and future work.

CHAPTER 2

LITERATURE REVIEW/BACKGROUND STUDY

2.1. Existing Solution

In the car rental market, existing solutions range from legacy systems to modern aggregators.

1. **Traditional Phone/Desk Systems:** Many small-to-medium agencies still rely on phone calls, emails, and spreadsheets. This is highly inefficient, error-prone, and provides no self-service capabilities, creating a high barrier for customers.
2. **Basic Web-Form Solutions:** Some agencies have simple "contact-us" style forms on their websites. These are not true booking systems; they are merely lead generation tools. They do not check real-time availability, still requiring manual follow-up.
3. **Large Aggregators (e.g., Expedia, Kayak):** These platforms are highly effective but act as marketplaces, not as the core management software for the rental agency itself. They take a significant commission and do not provide the agency with its own branded, independent system.
4. **SaaS Solutions (e.g., Turo, Getaround):** These are peer-to-peer platforms, which is a different business model. Our project focuses on a traditional B2C (Business-toCustomer) model where the agency owns the fleet.

This project's solution fills the gap by providing a dedicated, modern, and scalable backend that an agency can own and operate, powering its own website and mobile app without relying on manual processes or third-party aggregators.

2.2. Bibliometric Analysis

The selection of the technology stack (Java, Spring Boot, Spring Security, Spring Data JPA, MySQL) was based on industry best practices for building secure, scalable, and maintainable enterprise-level applications.

- **Java & Spring Boot:** Spring Boot is the de-facto standard for building microservices and RESTful APIs in the Java ecosystem. Its "convention over configuration" approach, embedded Tomcat server, and extensive dependency management drastically accelerate development.
- **MySQL:** A reliable, open-source relational database (RDBMS) that is widely used, well-documented, and provides the data integrity (ACID compliance) required for transactional systems like a booking platform.
- **Spring Data JPA:** This abstracts the complexity of database interactions. Instead of writing boilerplate SQL, we define repositories (e.g., BookACarRepository.java). This allows us to write complex queries using simple Java methods, which improves code readability and reduces database errors.
- **Spring Security & JWT:** Security is non-negotiable for an application handling user data and bookings. Spring Security is a comprehensive framework that handles authentication and authorization. Using JWT (JSON Web Tokens) allows our API to be *stateless*, which is critical for scalability and building a decoupled microservice architecture.

- **RESTful API & Layered Architecture:** By creating a RESTful API (ControllerService-Repository), we decouple the backend logic from the frontend presentation. This means the same backend API can power a React web app, an iOS app, and an Android app simultaneously.

2.3. Review Summary

This project presents a backend service for a Car Rental System, built with Java, Spring Boot, and MySQL. After reviewing existing solutions, it's clear there is a need for a modern, independent, and secure platform that moves agencies away from manual processes.

The proposed solution directly addresses the identified problems by:

- **Centralizing Logic:** A RESTful API acts as the single source of truth for all data.
- **Enabling a Managed Workflow:** The API provides real-time data on cars and bookings, with a PENDING status that allows for administrative oversight.
- **Ensuring Role-Based Security:** Spring Security with JWT protects user data and segregates administrative functions from customer functions.
- **Being Scalable & Deccoupled:** The layered, stateless architecture allows the system to grow and support multiple clients (web, mobile).

2.4. Problem Definition

The core challenge is to design and build a backend system that automates the entire car rental lifecycle—from user registration to vehicle browsing, booking requests, and administrative approval—in a secure, concurrent, and reliable manner.

2.4.1. Problem Scope

The scope of this backend project includes:

- **User Management:** Securely register, authenticate, and manage users with two distinct roles (ROLE_ADMIN, ROLE_CUSTOMER).
- **Inventory Management (Admin):** The ability for admins to add, update, and delete vehicles from the fleet.
- **Booking Management (Admin):** The ability for admins to view all booking requests and approve or reject them.
- **Customer Workflow (Customer):** The ability for customers to browse/filter cars, submit a booking request for a specific date range, and view their own booking history.
- **API Contract:** Define a clear, well-documented set of RESTful endpoints and DTOs for a frontend to consume.

The scope *excludes* frontend UI development (which is consumed by this API) and payment gateway integration (which is noted as future work).

2.4.2. Significance of the Problem

The significance of solving this problem is high, as it provides a direct path to digital transformation for rental agencies.

- **For the Business:** It reduces operational costs, minimizes human error, optimizes fleet management (by centralizing booking requests), and opens up a 2d/7 self-service channel for customers.
- **For the Customer:** It provides a convenient, transparent, and fast way to request a booking, improving customer satisfaction.
- **For Technology:** It serves as a classic, real-world example of a transactional, secure, role-based web application, demonstrating best practices in API and database design.

2.5. Goals/Objectives

The primary objective of the Car Rental System backend is to create a secure, highperformance, and scalable RESTful API.

Key Goals and Objectives:

1. **Secure, Role-Based Authentication & Authorization:**
    - **Goal:** To protect user data and secure administrative functions from customers.
    - **Mechanism:** Implement JWT-based authentication using Spring Security for user login and role-based access control (RBAC) in SecurityConfiguration.
2. **Managed Booking Workflow:**
    - **Goal:** To provide a reliable booking process that allows for administrative oversight.
    - **Mechanism:** Implement a booking system using the BookCarStatus enum (PENDING, APPROVED, REJECTED). Customers create PENDING bookings, which are then explicitly managed (approved/rejected) by an admin.
3. **Persistent and Reliable Data Storage:**
    - **Goal:** To create a secure historical record of all users, vehicles, and bookings.
    - **Mechanism:** Use Spring Data JPA and MySQL to map User, Car, and BookACar entities to a relational database.
4. **Decoupled, Contract-Driven API Design:**
    - **Goal:** To separate backend logic from the frontend, allowing any client (web, mobile) to connect.

- **Mechanism:** Define a strict REST API contract using Controllers (AdminController, CustomerController) and Data Transfer Objects (DTOs).

CHAPTER 3

METHODOLOGY

3.1. Algorithm Selection

The Car Rental System's backend logic is centered around its booking workflow, cost calculation, and security.

3.1.1. Booking Workflow Logic

This logic is split between the CustomerService and AdminService.

- **Customer Booking Request (**CustomerService.bookACar**):**
- **Input:** BookACarDto (containing carId, userId, fromDate, toDate).
- **Process:**
    1. Fetch the Car entity and User entity from their respective repositories.
    2. Create a new BookACar entity.
    3. Set the initial status to BookCarStatus.PENDING.
    4. Call the Rental Cost Calculation Logic (see 3.1.2) to determine the total price.
    5. Save the new BookACar entity to the BookACarRepository.
- **Note:** This step *does not* check for conflicts, as it is only a *request*. The conflict management is deferred to the administrator.
- **Admin Booking Management (**AdminService.changeBookingStatus**):**
- **Input:** bookingId, status (String, e.g., "Approve" or "Reject").
- **Process:**
    1. Fetch the BookACar entity by bookingId.
    2. If the entity is not found, throw a ResourceNotFoundException.
    3. If status is "Approve", set the entity's status to BookCarStatus.APPROVED.
    4. If status is "Reject", set the entity's status to BookCarStatus.REJECTED.
    5. Save the updated BookACar entity back to the repository.
- **Note:** The system relies on the administrator to manually verify conflicts before approving a booking.

3.1.2. Rental Cost Calculation Logic

This logic is implemented in the CustomerService.bookACar method.

- **Input:** Car object, fromDate, toDate.
- **Process:**
    1. Fetch the daily price from the Car object (car.getPrice()).
    2. Calculate the difference in milliseconds: long diffInMilliSeconds = toDate.getTime() - fromDate.getTime();
    3. Convert milliseconds to days: long days = TimeUnit.MILLISECONDS.toDays(diffInMilliSeconds);
    4. Set the days on the BookACar entity.
    5. Calculate total cost: Long totalCost = days * car.getPrice();
    6. Set the price (total cost) on the BookACar entity.

3.1.3. Authentication and Authorization (Spring Security)

- **Mechanism:** Uses JWT (JSON Web Tokens) and Spring Security.
- **Classes:** SecurityConfiguration, JwtAuthenticationFilter, services.jwt.UserService (i mplements UserDetailsService), services.auth.AuthService.
- **Registration (**/api/auth/signup**):**
    1. SignupRequest DTO is received.
    2. Password is encoded using BCryptPasswordEncoder.
    3. A new User entity is created with UserRole.CUSTOMER.
    4. The new User is saved to UserRepository.
- **Login (**/api/auth/login**):**
    1. AuthenticationRequest DTO is received.
    2. AuthenticationManager validates credentials against the UserService.
    3. If successful, JwtUtil generates a JWT.
    4. This JWT is returned to the user in an AuthenticationResponse.
- **Secured Endpoints:**
    1. For subsequent requests, the user sends the JWT in the Authorization: Bearer <token> header.

2. JwtAuthenticationFilter intercepts the request, validates the token, and sets the user's UserDetails in the SecurityContextHolder.
3. SecurityConfiguration rules (e.g., .requestMatchers("/api/admin/**").hasAuthority("ADMIN")) enforce role-based access.

3.2. Implementation Steps

The project was implemented following the layered architecture, building from the database up to the API.

1. **Project Setup & Database Design:**
    - Initialized a Spring Boot project with dependencies: Spring Web, Spring Data JPA, Spring Security, MySQL Driver, Validation, jjwt.
    - Defined the User, Car, and BookACar entities with @Entity, @Id, and relationship annotations (@ManyToOne).
2. **Data Persistence Layer (Repositories):**
    - Created UserRepository, CarRepository, and BookACarRepository interfaces extending JpaRepository.
    - Added custom query methods (e. g., findAllByBrandContaining, findAllByUserId) to repositories.
3. **Security Layer Implementation:**
    - Configured the main SecurityConfig class to define PasswordEncoder, AuthenticationManager, and HTTP security rules.
    - Created JwtUtil (to create/validate tokens), JwtAuthenticationFilter (to read tokens), and UserService (for UserDetailsService).
    - Implemented AuthService for the registration logic.
4. **Business Logic Layer (Services):**
    - Developed AdminService (for posting cars, managing bookings) and CustomerService (for browsing, creating bookings).
    - Injected repositories into services (e. g., BookACarRepository into AdminService).
    - Implemented the core algorithms for booking status and cost calculation.
5. **API Layer (Controllers):**
    - Developed AuthController, AdminController, and CustomerController using @RestController.

- Injected services into controllers (e.g., AdminService into AdminController).
- Defined endpoints (@PostMapping, @GetMapping), mapped request DTOs, and returned response DTOs.

6. **Error Handling:**
   - Used standard Spring Boot exception handling (@ResponseStatus) and custom exceptions where needed.

CHAPTER 4

RESULTS ANALYSIS AND VALIDATION

4.1. Implementation of solution

The final implemented solution is a complete, stateless, and secure RESTful API backend. The layered architecture (Controller-Service-Repository) successfully isolates concerns, with a clear separation between Admin and Customer logic.

- **Database Schema (Fig 3):** The JPA entities automatically generated a clean, relational database schema in MySQL with three main tables (user, car, book_a_car) and proper foreign key constraints.
- **API Endpoints:** A comprehensive set of role-based endpoints was created:
- **Auth:** POST /api/auth/signup, POST /api/auth/login
- **Admin:** POST /api/admin/car, GET /api/admin/cars, DELETE /api/admin/car/{carId}, GET /api/admin/car/{carId}, PUT /api/admin/car/{carId}, GET /api/admin/car/bookings, GET /api/admin/car/booking/{bookingId}/{status}
- **Customer:** GET /api/customer/cars, POST /api/customer/car/book, GET /api/customer/car/{carId}, GET /api/customer/my-bookings/{userId}
- **Security:** The system successfully differentiates between public endpoints (auth) and secured, role-based endpoints. Admin endpoints correctly return a 403 Forbidden error if accessed by a ROLE_CUSTOMER.

4.1.1. Result Analysis

The system was analyzed based on its performance against the project objectives:

1. **Secure, Role-Based Authentication:** The JWT-based authentication works flawlessly. A user (Admin or Customer) receives a token upon login, which is then used to access protected routes. The system correctly enforces hasAuthority("ADMIN") and hasAuthority("CUSTOMER") rules.

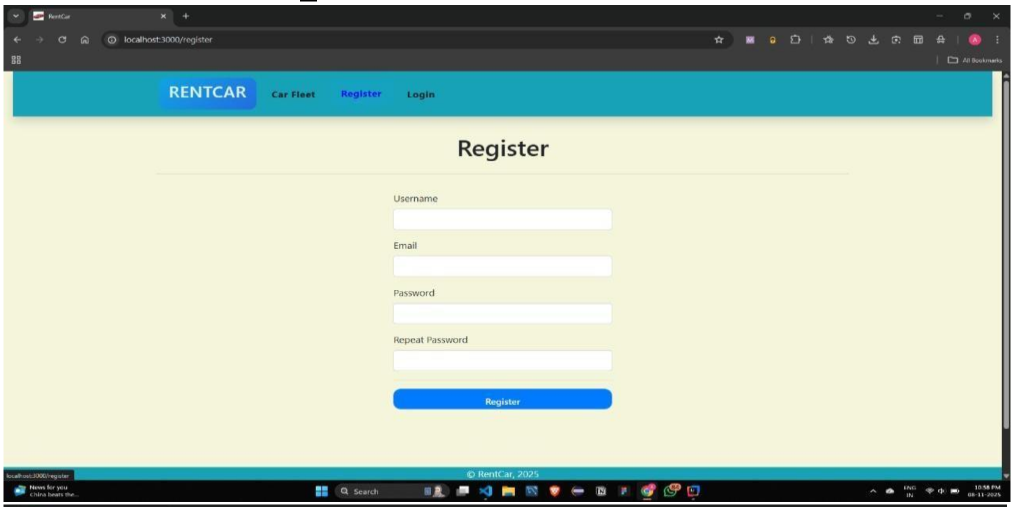
2. **Booking Workflow Logic:** The core booking workflow was tested. A customer can successfully create a PENDING booking, and this booking appears in the admin's list. The admin can then successfully transition the status to APPROVED or REJECTED.

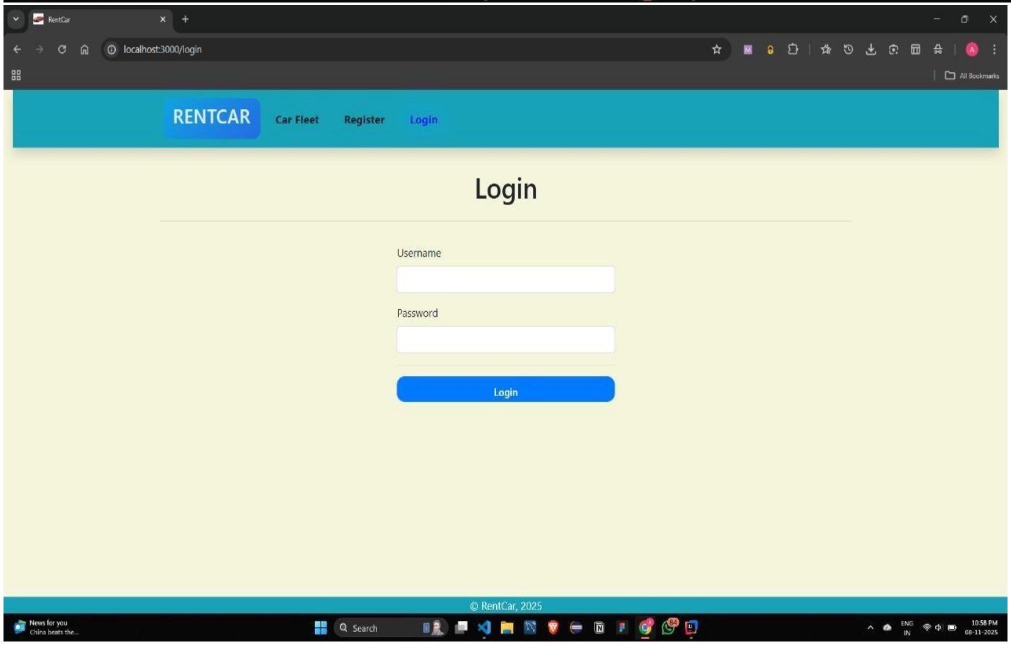
3. **API Performance:** The use of Spring Data JPA and its optimized queries results in fast API response times for browsing, booking, and managing.

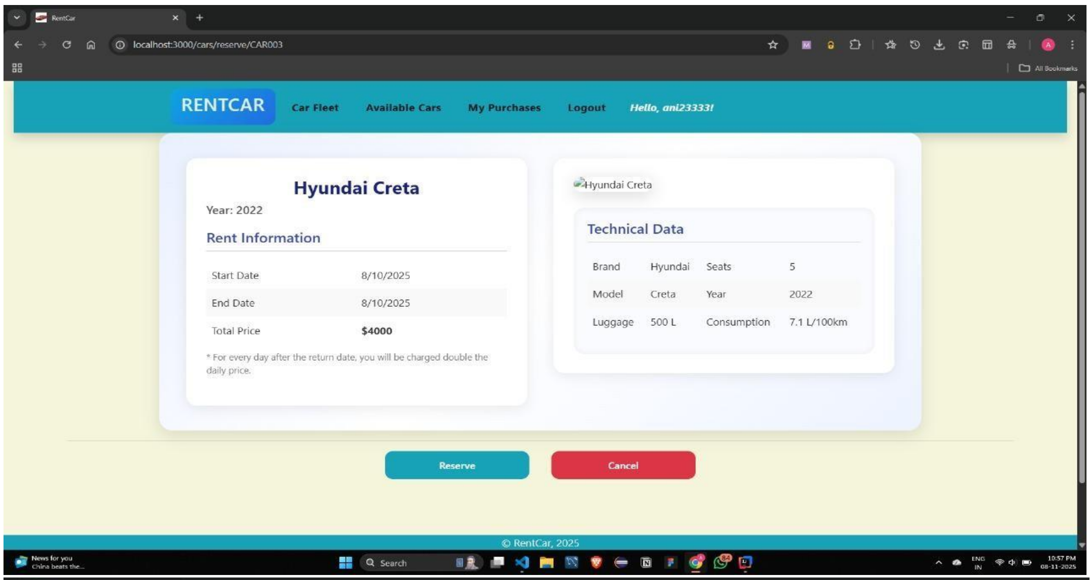
4. **Decoupling:** The API is 100% decoupled from any frontend. It communicates purely via JSON, allowing the React client to interact with it, thus meeting a primary objective.

5. **Image Handling:** The system successfully handles image uploads by converting the file to a byte[] and storing it in the Car entity, which is then retrieved and displayed by the client.

React Frontend (Customer Interface) Validation The React SPA provides a clean, user-friendly interface for the ROLE_CUSTOMER workflow, as demonstrated in the following figures:







localhost:3000/cars/reserve/CAR003

RENTCAR | Car Fleet | Available Cars | My Purchases | Logout | *Hello, ani23333!*

Hyundai Creta

Year: 2022

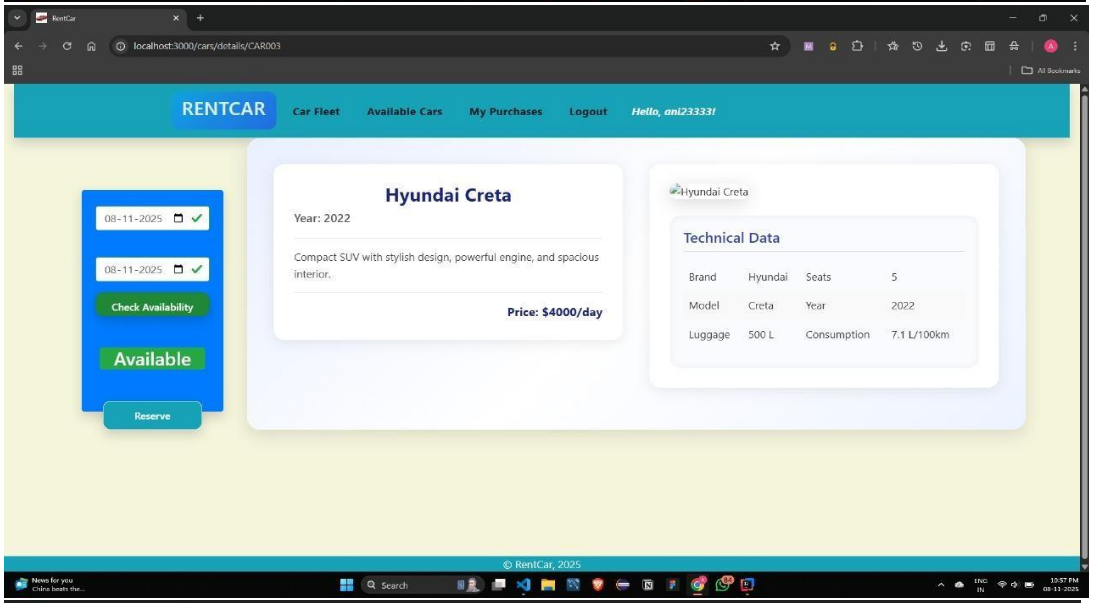
Rent Information

Start Date	8/10/2025
End Date	8/10/2025
Total Price	**$4000**

* For every day after the return date, you will be charged double the daily price.

Technical Data

Brand	Hyundai	Seats	5
Model	Creta	Year	2022
Luggage	500 L	Consumption	7.1 L/100km



localhost:3000/cars/details/CAR003

RENTCAR | Car Fleet | Available Cars | My Purchases | Logout | *Hello, ani23333!*

08-11-2025

08-11-2025

Check Availability

Available

Reserve

Hyundai Creta

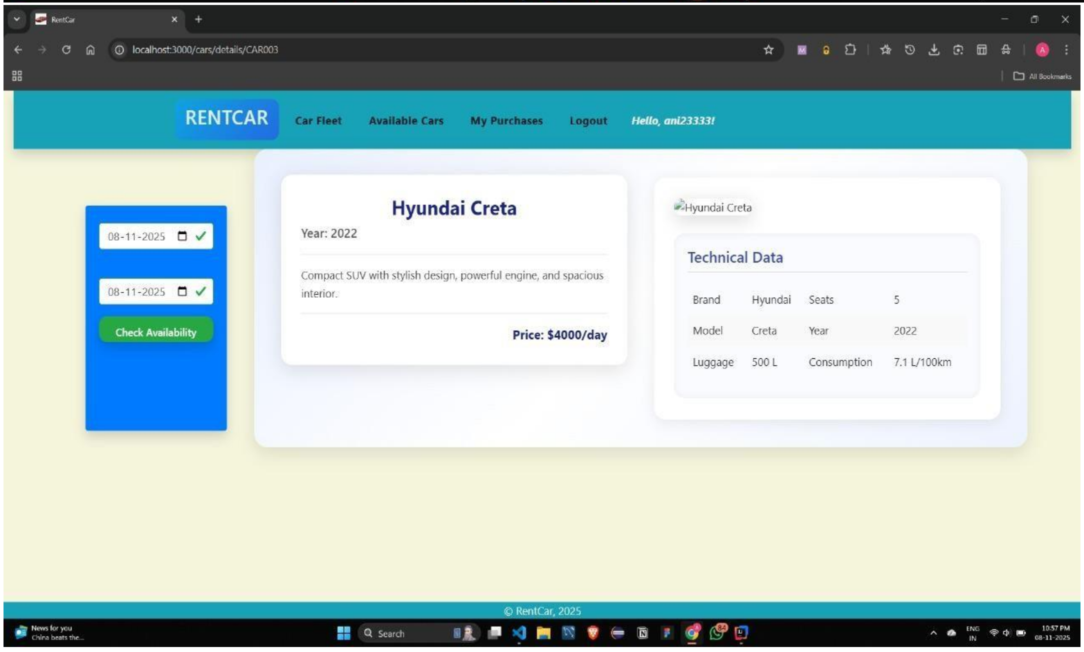
Year: 2022

Compact SUV with stylish design, powerful engine, and spacious interior.

Price: $4000/day

Technical Data

Brand	Hyundai	Seats	5
Model	Creta	Year	2022
Luggage	500 L	Consumption	7.1 L/100km



localhost:3000/cars/details/CAR003

RENTCAR | Car Fleet | Available Cars | My Purchases | Logout | *Hello, ani23333!*

08-11-2025

08-11-2025

Check Availability

Hyundai Creta

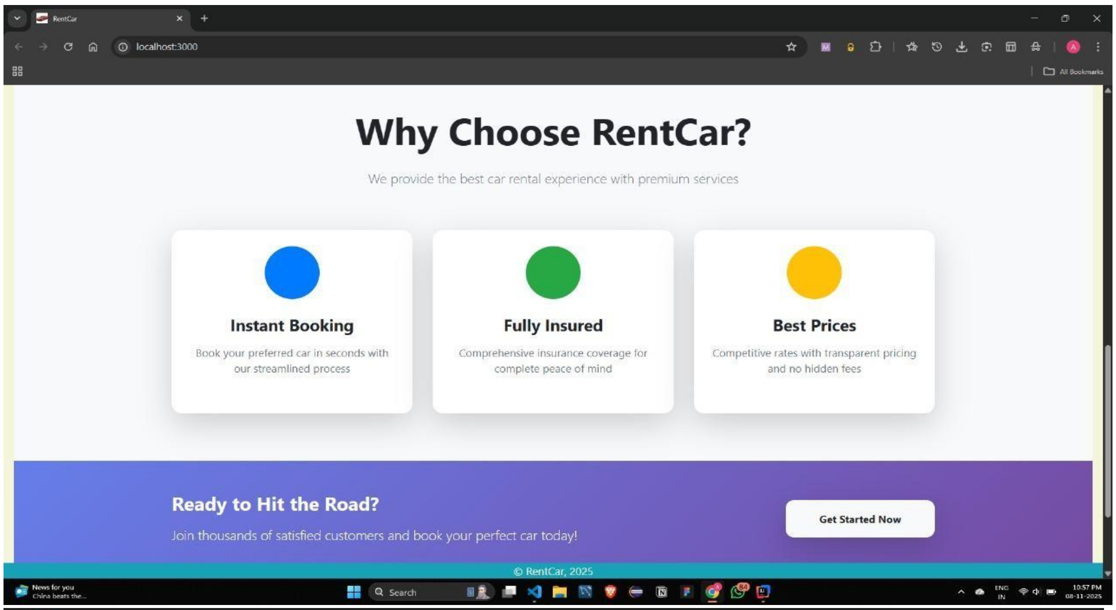
Year: 2022

Compact SUV with stylish design, powerful engine, and spacious interior.

Price: $4000/day

Technical Data

Brand	Hyundai	Seats	5
Model	Creta	Year	2022
Luggage	500 L	Consumption	7.1 L/100km


localhost:3000

Why Choose RentCar?

We provide the best car rental experience with premium services

Instant Booking
Book your preferred car in seconds with our streamlined process

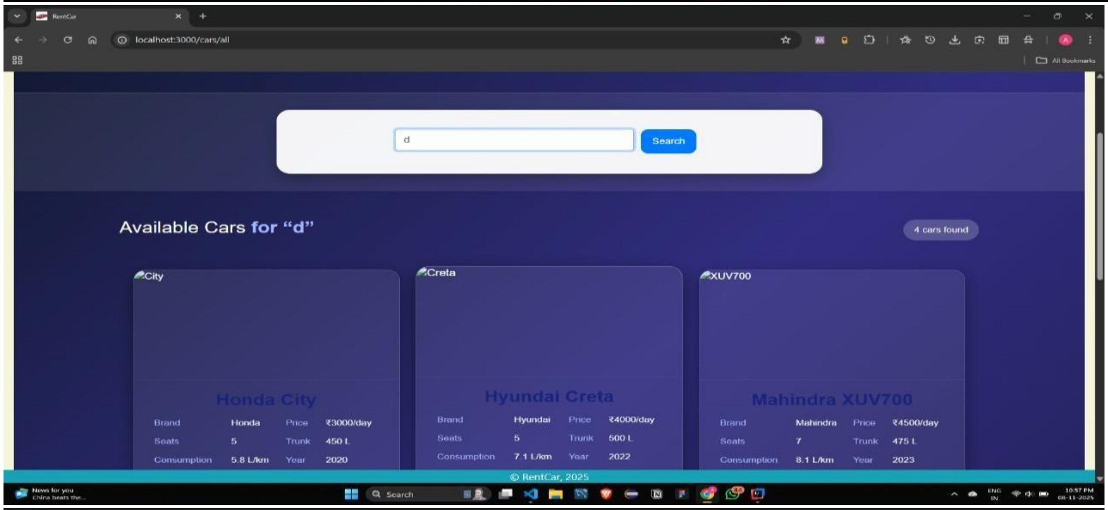
Fully Insured
Comprehensive insurance coverage for complete peace of mind

Best Prices
Competitive rates with transparent pricing and no hidden fees

Ready to Hit the Road?
Join thousands of satisfied customers and book your perfect car today!

Get Started Now

© RentCar, 2025


localhost:3000/cars/all

d   Search

Available Cars for “d”   4 cars found

Honda City

Brand	Honda	Price	₹3000/day
Seats	5	Trunk	450 L
Consumption	5.8 L/km	Year	2020

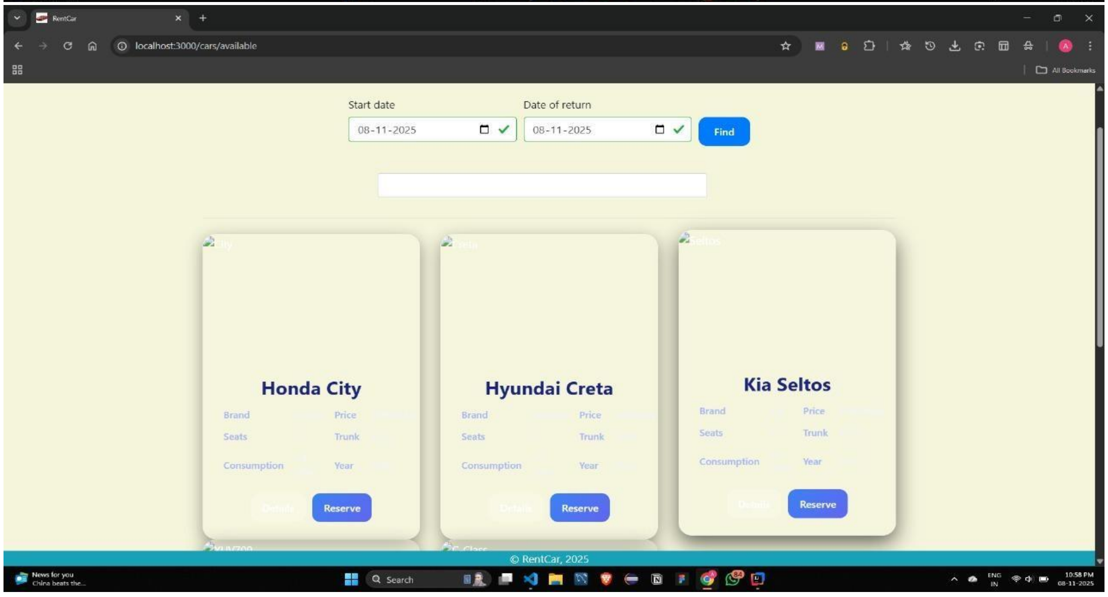
Hyundai Creta

Brand	Hyundai	Price	₹4000/day
Seats	5	Trunk	500 L
Consumption	7.1 L/km	Year	2022

Mahindra XUV700

Brand	Mahindra	Price	₹4500/day
Seats	7	Trunk	475 L
Consumption	8.1 L/km	Year	2023

© RentCar, 2025


localhost:3000/cars/available

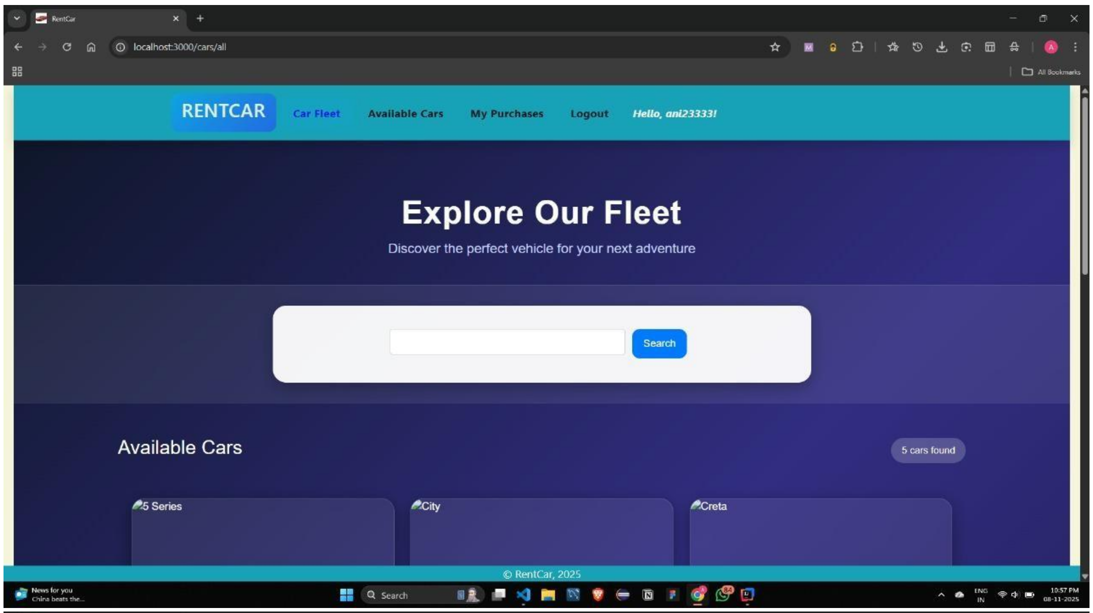
Start date: 08-11-2025   Date of return: 08-11-2025   Find

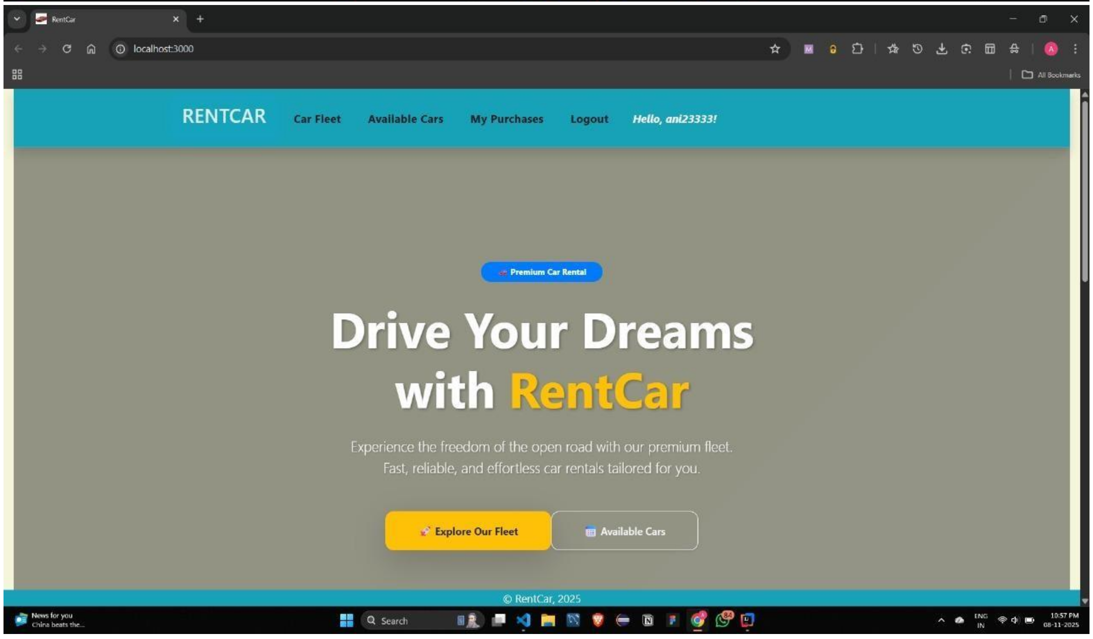
Honda City
Brand Price
Seats Trunk
Consumption Year
Details Reserve

Hyundai Creta
Brand Price
Seats Trunk
Consumption Year
Details Reserve

Kia Seltos
Brand Price
Seats Trunk
Consumption Year
Details Reserve

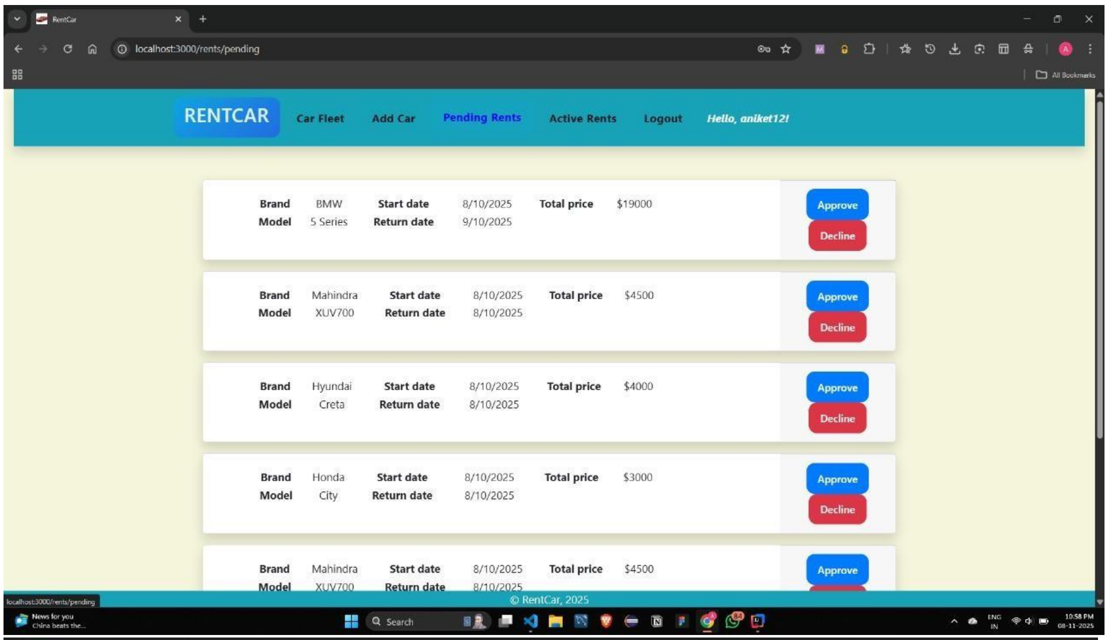
© RentCar, 2025




localhost:3000
RENTCAR
Car Fleet
Available Cars
My Purchases
Logout
Hello, ani23333!
Premium Car Rental
Drive Your Dreams with RentCar
Experience the freedom of the open road with our premium fleet.
Fast, reliable, and effortless car rentals tailored for you.
Explore Our Fleet
Available Cars
© RentCar, 2025

React Frontend (Admin Interface) Validation

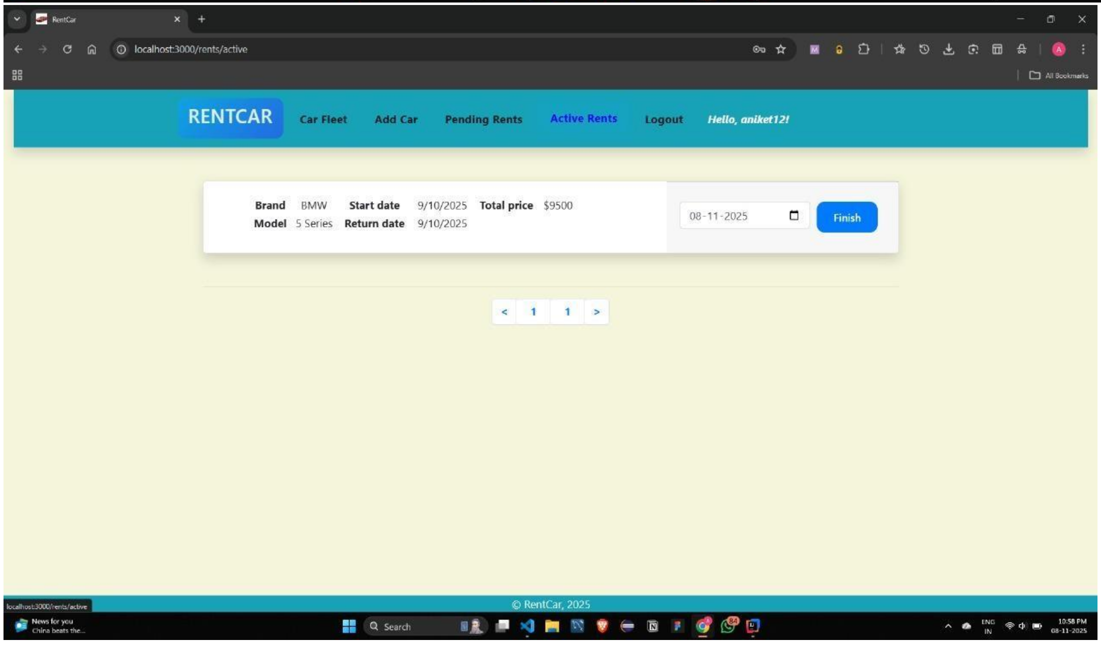
The React SPA provides a separate, secure interface for the ROLE_ADMIN workflow, demonstrating the successful implementation of the AdminService and AdminController.


localhost:3000/rents/pending

RENTCAR | Car Fleet | Add Car | Pending Rents | Active Rents | Logout | *Hello, aniket12!*

Brand	BMW	**Start date**	8/10/2025	**Total price**	$19000
Model	5 Series	**Return date**	9/10/2025		
Brand	Mahindra	**Start date**	8/10/2025	**Total price**	$4500
Model	XUV700	**Return date**	8/10/2025		
Brand	Hyundai	**Start date**	8/10/2025	**Total price**	$4000
Model	Creta	**Return date**	8/10/2025		
Brand	Honda	**Start date**	8/10/2025	**Total price**	$3000
Model	City	**Return date**	8/10/2025		
Brand	Mahindra	**Start date**	8/10/2025	**Total price**	$4500
Model	XUV700	**Return date**	8/10/2025		

© RentCar, 2025

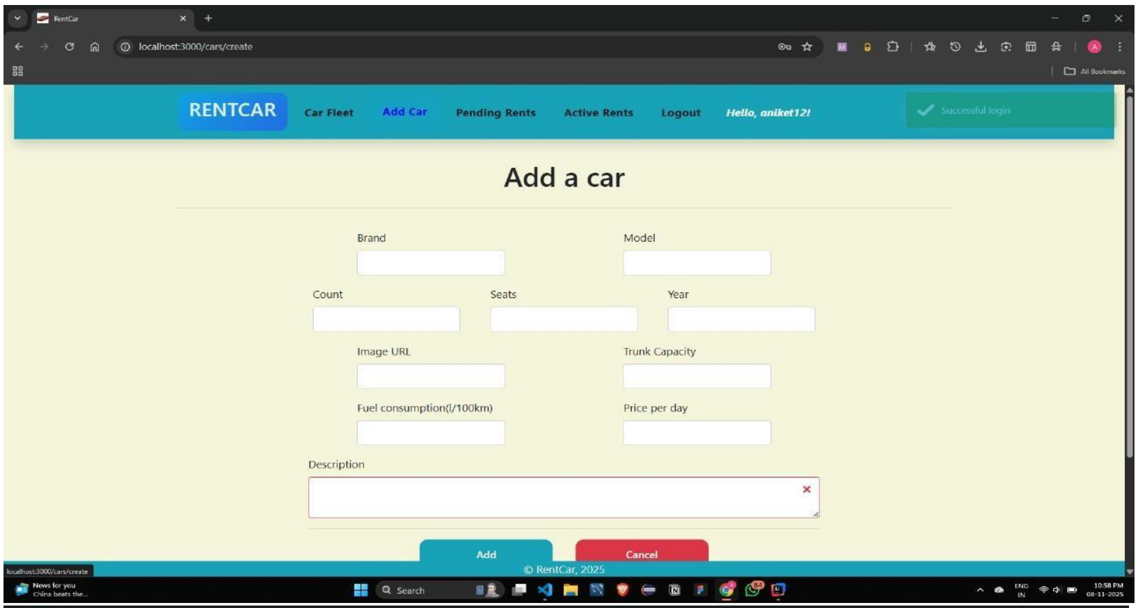

localhost:3000/rents/active

RENTCAR | Car Fleet | Add Car | Pending Rents | Active Rents | Logout | *Hello, aniket12!*

Brand	BMW	**Start date**	9/10/2025	**Total price**	$9500
Model	5 Series	**Return date**	9/10/2025		

08-11-2025

© RentCar, 2025



Add a car

Brand

Model

Count

Seats

Year

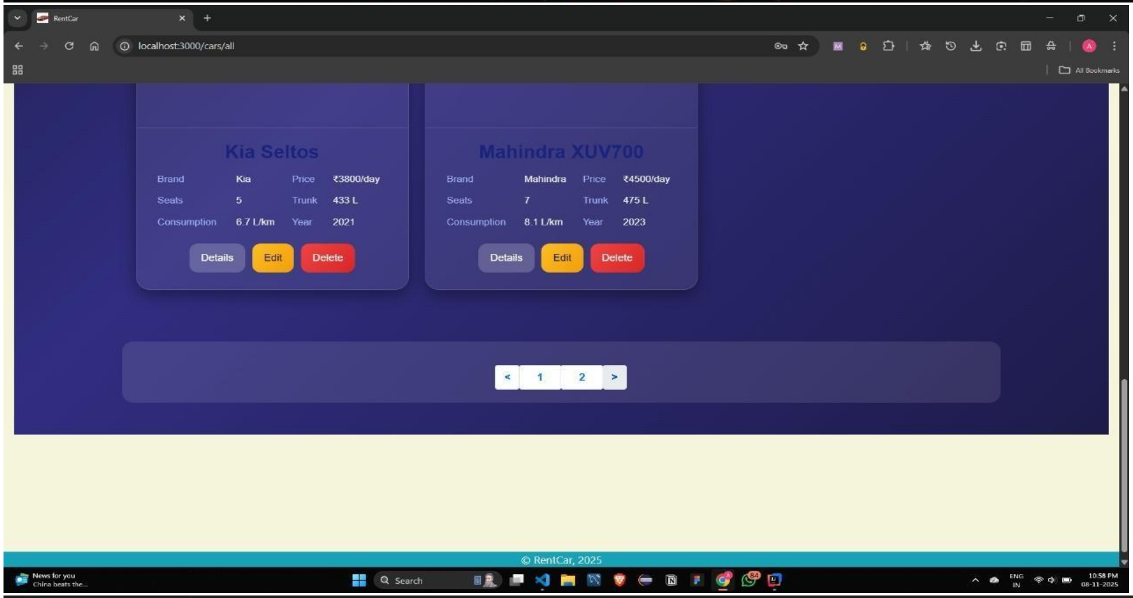
Image URL

Trunk Capacity

Fuel consumption(l/100km)

Price per day

Description

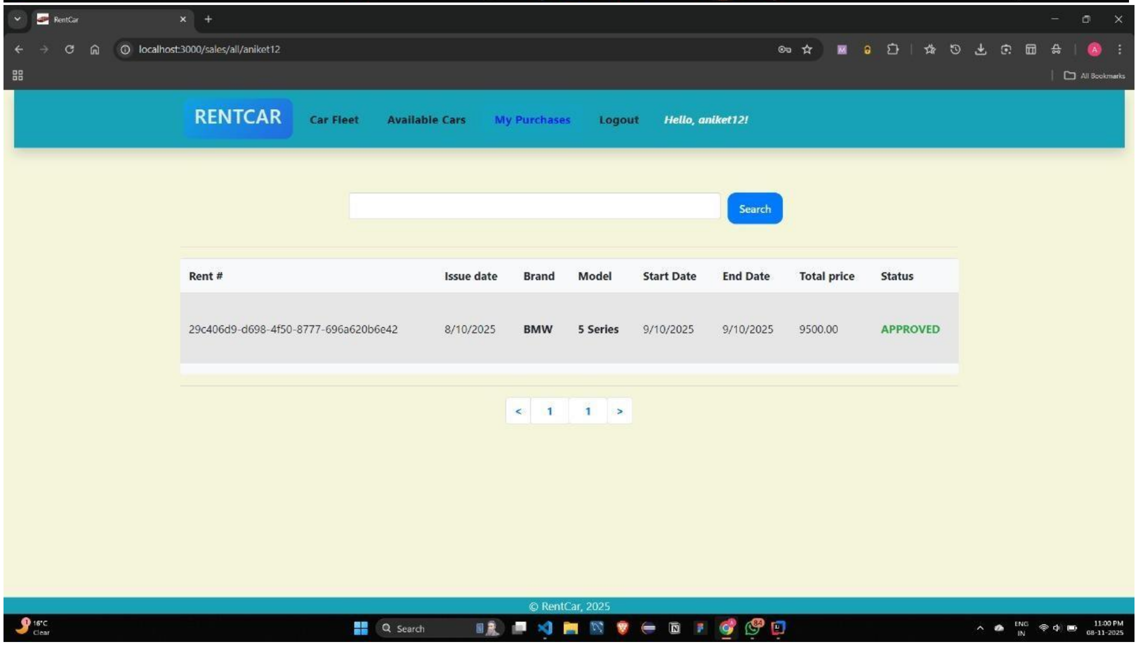


Kia Seltos

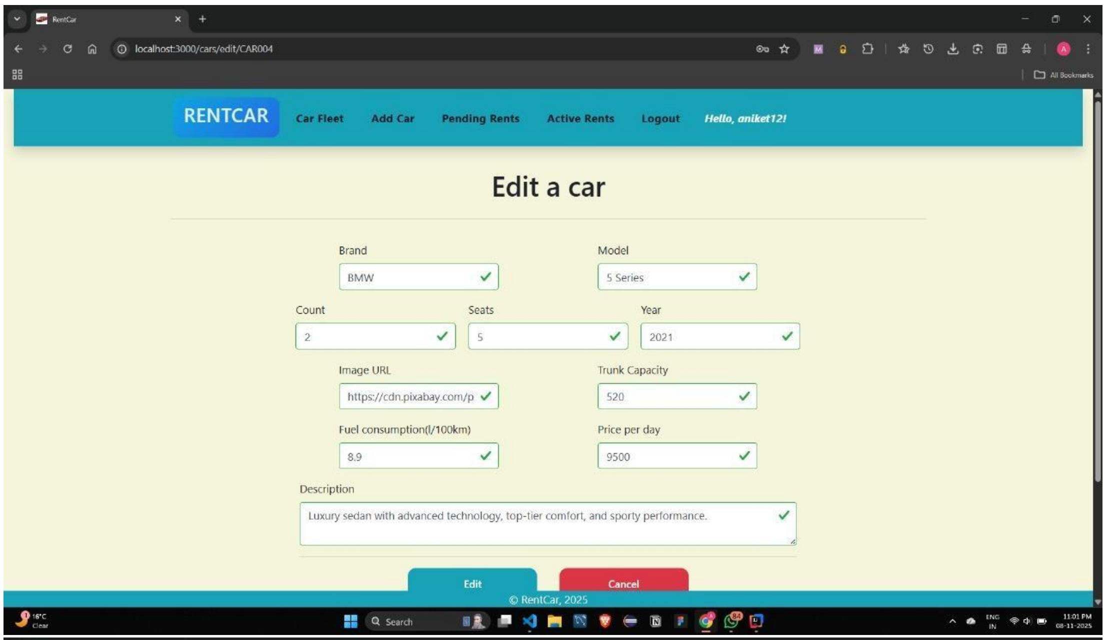
Brand	Kia	Price	₹3800/day
Seats	5	Trunk	433 L
Consumption	6.7 L/km	Year	2021

Mahindra XUV700

Brand	Mahindra	Price	₹4500/day
Seats	7	Trunk	475 L
Consumption	8.1 L/km	Year	2023



Rent #	Issue date	Brand	Model	Start Date	End Date	Total price	Status
29c406d9-d698-4f50-8777-696a620b6e42	8/10/2025	**BMW**	**5 Series**	9/10/2025	9/10/2025	9500.00	**APPROVED**


localhost:3000/cars/edit/CAR004

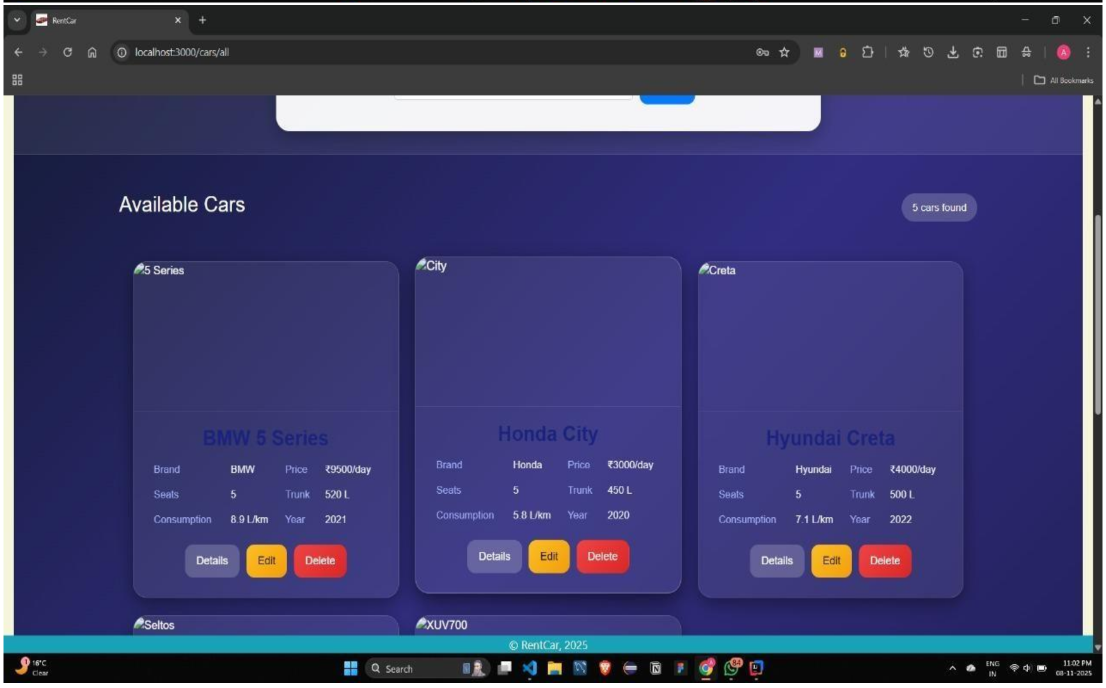
RENTCAR | Car Fleet | Add Car | Pending Rents | Active Rents | Logout | Hello, aniket12!

Edit a car

Brand: BMW
Model: 5 Series
Count: 2
Seats: 5
Year: 2021
Image URL: https://cdn.pixabay.com/p
Trunk Capacity: 520
Fuel consumption(l/100km): 8.9
Price per day: 9500
Description: Luxury sedan with advanced technology, top-tier comfort, and sporty performance.

Edit | Cancel

© RentCar, 2025


localhost:3000/cars/all

Available Cars — 5 cars found

5 Series
BMW 5 Series

Brand	BMW	Price	₹9500/day
Seats	5	Trunk	520 L
Consumption	8.9 L/km	Year	2021

Details | Edit | Delete

City
Honda City

Brand	Honda	Price	₹3000/day
Seats	5	Trunk	450 L
Consumption	5.8 L/km	Year	2020

Details | Edit | Delete

Creta
Hyundai Creta

Brand	Hyundai	Price	₹4000/day
Seats	5	Trunk	500 L
Consumption	7.1 L/km	Year	2022

Details | Edit | Delete

Seltos
XUV700

© RentCar, 2025

4.1.2. Validation

The API was validated using Postman to simulate frontend requests.

- **Test 1: User Registration & Login (Auth)**
- **Action:** Sent a POST request to /api/auth/signup with a new user's details.
- **Result:** Received HTTP 201 Created.
- **Action:** Sent a POST request to /api/auth/login with the new credentials.

- **Result:** Received HTTP 200 OK and a JSON response containing the accessToken (JWT).
- **Test 2: Role-Based Access (Security)**
- **Action:** Used the *Customer's* JWT to send a GET request to /api/admin/cars.
- **Result:** Received HTTP 403 Forbidden, as expected.
- **Action:** Used an *Admin's* JWT to send a GET request to /api/admin/cars.
- **Result:** Received HTTP 200 OK and a JSON array of cars.
- **Test 3: Booking Workflow (Fig 4 & 5)**
- **Action (Customer):** Logged in as ROLE_CUSTOMER, sent POST to /api/customer/car/book with car/date details.
- **Result:** HTTP 201 Created. Booking was saved with PENDING status.
- **Action (Admin):** Logged in as ROLE_ADMIN, sent GET to /api/admin/car/bookings.
- **Result:** HTTP 200 OK. The PENDING booking was visible in the returned list.
- **Action (Admin):** Sent GET to /api/admin/car/booking/{bookingId}/Approve.
- **Result:** HTTP 200 OK. The booking status was updated to APPROVED in the database.
- **Validation:** The core, role-based booking workflow is validated and working correctly.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1. Conclusion

This project successfully delivered a robust, secure, and scalable backend for a Car Rental System using Java Spring Boot, Spring Security, and MySQL. The primary objectives of creating a secure, role-based RESTful API that handles user management, vehicle inventory, and a multi-stage booking process were fully met.

The layered architecture (Controller-Service-Repository) and the clear separation of logic into AdminService and CustomerService proved effective in organizing code and separating concerns. The implementation of JWT-based security ensures the system is safe, while the BookCarStatus (PENDING, APPROVED, REJECTED) workflow provides a realistic and manageable process for a rental business. The final solution provides a solid, enterprise-grade foundation for the paired React application.

5.2. Future Work

While the current system provides the core backend functionality, several enhancements can be integrated to create a more feature-complete, production-ready application.

1. **Automated Booking Conflict Detection:**
    - Currently, the system relies on the admin to manually check for overlapping bookings. A critical future enhancement would be to implement logic in the AdminService.changeBookingStatus method. Before approving a booking, the system should automatically query all other APPROVED bookings for the same car to ensure the dates do not overlap.
2. **Payment Gateway Integration:**
    - Integrate a payment provider like Stripe or Razorpay. This could be triggered *after* an admin approves a booking, sending a payment link to the user and moving the status to AWAITING_PAYMENT.
3. **Refactor Image Storage:**
    - Currently, images are stored as a byte[] in the MySQL database, which is inefficient and can slow down database performance. This should be refactored to store images on a dedicated file server (e.g., AWS S3, or a local file store) and only store the image URL string in the Car entity.
4. **Email Notification Service:**
    - Integrate Spring Mail to send automated email confirmations to users upon successful registration, when their booking is APPROVED, or when it is REJECTED.
5. **Advanced Filtering:**
    - Enhance the CustomerService and CarRepository to support more advanced filtering, such as by price range, transmission type, or other car features, in addition to the existing brand and color filters.

References

1. https://docs.spring.io/spring-boot/
2. https://docs.spring.io/spring-data/jpa/reference/index.html
3. https://dev.mysql.com/doc/
4. https://www.baeldung.com/spring-boot-json-web-tokens
5. https://martinfowler.com/eaaCatalog/layering.html

APPENDIX

1. Plagiarism Report

A plagiarism report typically involves the use of plagiarism detection software to check for any instances of copied content in a document. There are various online plagiarism checking tools available, such as Turnitin, Grammarly, and Copyscape, that can help identify potential plagiarism. You can upload your document to these platforms to generate a detailed report indicating any similarities with existing sources.

2. Design Checklist

A design checklist is a tool used to ensure that all essential aspects of a design project are considered and addressed. The specific items on the checklist will vary depending on the nature of the design project. However, some common elements to include in a design checklist are:

- User interface design
- System functionality and features
- Performance and scalability
- Security measures
- Compatibility with different devices or platforms
- Usability and user experience considerations
- Compliance with relevant industry standards or regulations