# Car Rental System Project Report

**Authors:** GAURANG DALVADI & DARSHAN KALATHIYA  
**Date:** [Insert Date]

## Abstract

This report presents the design, implementation, and evaluation of a full-stack Car Rental System. The system provides a modern, user-friendly platform for customers to browse, book, and manage car rentals, and for administrators to manage inventory and bookings. The solution leverages a React frontend, a Spring Boot backend, and a MySQL database, with secure authentication and robust error handling. This document details the system’s architecture, features, database design, implementation, testing, and future enhancement opportunities.

## Table of Contents

1. [Introduction](#introduction)
2. [System Overview](#system-overview)
3. [System Architecture](#system-architecture)
4. [Frontend Functionality](#frontend-functionality)
5. [Backend Functionality](#backend-functionality)
6. [Database Design](#database-design)
7. [Implementation Details](#implementation-details)
8. [Testing and Validation](#testing-and-validation)
9. [Security Considerations](#security-considerations)
10. [Performance Optimizations](#performance-optimizations)
11. [Deployment and Configuration](#deployment-and-configuration)
12. [Future Enhancements](#future-enhancements)
13. [Conclusion](#conclusion)
14. [References](#references)
15. [Appendices](#appendices)

## Introduction

### Background and Motivation

Car rental services are essential for urban mobility, travel, and tourism. Traditional rental processes are often manual and inefficient. This project aims to digitize and streamline the car rental process, providing a seamless experience for both customers and administrators.

### Problem Statement

Manual car rental management leads to inefficiencies, errors, and poor customer experience. There is a need for an integrated, automated system that supports real-time car availability, secure bookings, and easy management.

### Objectives

* Provide a user-friendly platform for car rental.
* Enable secure authentication and role-based access.
* Support full CRUD operations for cars and bookings.
* Ensure robust error handling and security.

## System Overview

The Car Rental System is a full-stack web application with the following major components: - **Frontend:** React.js SPA for customer and admin interfaces. - **Backend:** Spring Boot REST API for business logic and data management. - **Database:** MySQL for persistent storage of users, cars, and bookings. - **Image Storage:** (Current: Database as BLOB; Recommended: Filesystem with path in DB)

### Key Features

* User registration and login (JWT-based)
* Car browsing, search, and details
* Booking system with date selection and price calculation
* Booking history and status tracking
* Admin dashboard for car and booking management
* Responsive, modern UI

## System Architecture

### High-Level Architecture

graph TD  
 A[Frontend (React)] --REST API--> B[Backend (Spring Boot)]  
 B --JPA/Hibernate--> C[MySQL Database]  
 B --Image Upload--> D[Image Storage (DB or Filesystem)]

* **Frontend:** Handles all user interactions, authentication, and API requests.
* **Backend:** Exposes REST endpoints, handles business logic, authentication, and data persistence.
* **Database:** Stores users, cars, bookings, and (currently) images.
* **Image Storage:** Images are currently stored as BLOBs in the database; recommended to store only the path and save files on the server.

## Frontend Functionality

### Customer Features

* **User Authentication:** Register and login with JWT token support.
* **Car Browsing:** View all available cars with search and filter functionality.
* **Car Details:** Detailed view of individual cars with specifications.
* **Booking System:** Book cars with date selection and price calculation.
* **Booking History:** View all past and current bookings with status tracking.

### Admin Features

* **Admin Dashboard:** Comprehensive dashboard with statistics.
* **Car Management:** Add, edit, and delete cars from the inventory.
* **Booking Management:** Approve or reject customer bookings.
* **Booking Overview:** View all bookings in a table format.

### Technical Features

* **Responsive Design:** Mobile-friendly interface.
* **Modern UI:** Clean and intuitive user interface.
* **Real-time Updates:** Dynamic content updates.
* **Error Handling:** Comprehensive error handling and user feedback.
* **Loading States:** Smooth loading animations.

## Backend Functionality

### API Endpoints

#### Authentication

* POST /api/auth/login — User login
* POST /api/auth/signup — User registration

#### Customer Endpoints

* GET /api/customer/cars — Get all cars
* GET /api/customer/car/{id} — Get car by ID
* POST /api/customer/car/book — Book a car
* GET /api/customer/car/bookings/{userId} — Get user bookings
* POST /api/customer/car/search — Search cars

#### Admin Endpoints

* GET /api/admin/cars — Get all cars (admin)
* POST /api/admin/car — Add new car
* PUT /api/admin/car/{id} — Update car
* DELETE /api/admin/car/{id} — Delete car
* GET /api/admin/car/bookings — Get all bookings
* GET /api/admin/car/booking/{bookingId}/{status} — Change booking status

### Roles

* **Admin:** Full access to car and booking management.
* **Customer:** Can browse, book, and view their own bookings.

### Image Handling

* **Current:** Images are stored as BLOBs in the database.
* **Recommended:** Store only the image path in the database and save files in a server directory.

## Database Design

### Main Entities

#### User

* id (Long, PK)
* name (String)
* email (String, unique)
* password (String, hashed)
* userRole (Enum: ADMIN, CUSTOMER)

#### Car

* id (Long, PK)
* brand (String)
* color (String)
* name (String)
* type (String)
* transmission (String)
* description (String)
* price (Long)
* year (Date)
* image (byte[] as BLOB, currently)

#### BookACar

* id (Long, PK)
* fromDate (Date)
* toDate (Date)
* days (Long)
* price (Long)
* bookCarStatus (Enum: APPROVED, REJECTED, etc.)
* user (FK to User)
* car (FK to Car)

### Entity-Relationship Diagram (Textual)

* **User** 1—\* **BookACar** \*—1 **Car**
* Each booking links a user to a car for a date range.

## Implementation Details

### Backend

* **Controllers:** AdminController, AuthController, CustomerController
* **Services:** Business logic for car, booking, and authentication management
* **DTOs:** Data transfer objects for API communication
* **Security:** JWT-based authentication, role checks, CORS configuration
* **Image Upload:** Multipart file handling (see recommended improvements)

### Frontend

* **Pages:** Home, Login, Register, Car List, Car Detail, Booking History, Admin Dashboard
* **Components:** AddCarModal, EditCarModal, Navbar
* **API Service:** Centralized API calls with Axios
* **State Management:** React context for authentication
* **Validation:** Form validation and error feedback

### Data Flow

* User interacts with React forms
* Data sent via Axios to Spring Boot REST API
* Backend processes request, updates database, returns response
* Frontend updates UI accordingly

## Testing and Validation

### Manual Testing

* Car addition, editing, and deletion
* Booking creation and status updates
* Authentication and role-based access
* Error handling and edge cases

### Automated Testing

* (To be implemented) Unit and integration tests for backend services and controllers
* Frontend component and integration tests

### Test Documentation

* See CAR\_ADD\_FUNCTIONALITY\_TEST.md for detailed test cases

## Security Considerations

* **Authentication:** JWT token validation, secure password storage
* **Authorization:** Role-based access control for admin/customer
* **CORS:** Configured for secure cross-origin requests
* **File Uploads:** Validate file type and size, secure storage
* **Input Validation:** Prevent SQL injection, XSS, and other attacks

## Performance Optimizations

### Frontend

* Lazy loading of modals and components
* Efficient state management
* Optimized re-renders

### Backend

* Efficient database queries
* Proper error handling
* Optimized file processing

## Deployment and Configuration

### Prerequisites

* Node.js (v14+), npm or yarn
* Java (Spring Boot), MySQL

### Installation

1. Clone the repository
2. Install frontend dependencies (npm install)
3. Configure backend database and properties
4. Start backend (mvn spring-boot:run or use provided scripts)
5. Start frontend (npm start)

### Configuration

* API base URL in src/services/api.js
* Database and server settings in application.properties

## Future Enhancements

* Store images on filesystem, not as BLOBs
* Image compression and optimization
* Bulk car import
* Advanced filtering and search
* Real-time updates (WebSocket)
* Car categories, pricing tiers, availability calendar
* Maintenance tracking

## Conclusion

The Car Rental System provides a robust, scalable, and user-friendly platform for car rental management. With a modern tech stack, secure authentication, and comprehensive features, it streamlines the rental process for both customers and administrators. Future enhancements will further improve performance, usability, and scalability.

## References

* React documentation: https://reactjs.org/
* Spring Boot documentation: https://spring.io/projects/spring-boot
* MySQL documentation: https://dev.mysql.com/doc/
* Axios: https://axios-http.com/
* Material UI: https://mui.com/

## Appendices

### A. API Endpoint List

(See Backend Functionality section)

### B. Key Code Snippets

* See implementation files for detailed code

### C. Project Structure

* See README and Implementation Summary for file layout

*End of Report*