A PROJECT REPORT ON:

VEHICO - (VEHICLE RENTAL SYSTEM)

Submitted in partial fulfillment of the requirements of the degree

Of

BACHELOR OF COMPUTER APPLICATION

Submitted By:

ABHISHEK KUMAR PRASAD

ROLL NO: UT-221-032-0009

REGISTRATION NO: 22069007

DEPARTMENT OF BACHELORS IN COMPUTER APPLICATION,

PRAGJYOTISH COLLEGE

SANTIPUR, ASSAM, INDIA

AUGUST-DECEMBER (2024-24).

CERTIFICATE

This is to certify that Abhishek Kumar Prasad , Roll No: (UT-221-032-0009) has successfully completed the project work on "VEHICO"
Paper Code (BCA-HE-5016). This project work is a bonafide work for partial fulfillment of the requirement of the 5th semester Bachelor of Computer Application program in the Department of Computer Application, Pragjyotish College, Guwahati, affiliated to Gauhati University. The work done by him is an academic work and cannot be presented for any other purpose.

PRAGJYOTISH COLLEGE

DEPARTMENT OF COMPUTER APPLICATION

External Signature

Internal Signature
Nibedita Das

ACKNOWLEDGEMENT

Completing a project successfully would remain incomplete without the support of people whose encouragement and guidance has been a great source of inspiration during the course of this project.

We take this opportunity to express our sincere gratitude and heartiest thanks to, Sir Nayan Mahanta H.O.D(BCA) and our teachers Ma'am Nibedita Das and Syeda Shamin Shabnam for giving us an opportunity to work on the project. We would also like to extend our sincere thanks to our guide Ms.Nibedita Das, without whom this project would not have been in its present form. It has been very knowledgeable experience for us to work on this project. Through this report we take the opportunity to express our sincere gratitude and thankfulness to all those who have helped us in making our project a success.

Abhishek Kumar Prasad

Roll No: UT-221-032-0009

BCA 5th Semester

Abstract

Vehico is a vehicle rental management system designed to streamline the process of connecting renters with available vehicles through a secure and intuitive platform. This system emphasizes **administrative control**, allowing only administrators to manage vehicle listings and oversee the rental process effectively.

Built with a robust backend using **Spring Boot** and an interactive frontend powered by **React**, Vehico ensures a seamless user experience. The system's core functionalities include **vehicle management by administrators**, booking requests by renters, and role-based access to maintain operational clarity and security. Renters can browse vehicles, submit booking requests, and track the status of their rentals.

The application utilizes **MySQL** for efficient and reliable database management and incorporates **JWT-based authentication** to secure user data and access. Vehico supports diverse vehicle categories, including cars, bikes, and trucks, catering to a wide range of user preferences.

By combining modern technologies and a focused approach to administrative control, Vehico provides a scalable and efficient solution for the vehicle rental industry, enhancing both user convenience and system reliability.

CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Background

The vehicle rental industry has undergone significant transformation with the advent of digital platforms. These platforms provide users with the convenience of browsing, booking, and managing vehicle rentals online, eliminating the need for traditional manual processes. However, existing systems often lack features like robust security, seamless user experience, and efficient management of vehicle availability and bookings. This project, *Vehico*, addresses these gaps by offering a modern, user-friendly vehicle rental platform.

1.2 Problem Statement

Renting vehicles manually involves cumbersome processes that can lead to inefficiencies, inaccuracies, and poor customer experience. Current digital solutions, while better, often fail to provide role-based access control, a responsive interface, and secure management of sensitive data. There is a need for a solution that integrates advanced technologies to create a secure, scalable, and user-centric vehicle rental platform.

1.3 Project Overview

Vehico is a web-based vehicle rental management system designed to streamline the rental process for both administrators and renters. It provides a centralized platform for admins to manage vehicles, bookings, and user accounts, while renters can browse available vehicles and make rental bookings. Built using Java, Spring Boot, MySQL, and React, the system ensures performance, security, and ease of use.

1.4 Scope of the Project

The scope of *Vehico* includes:

- Development of a role-based access system where admins can manage vehicles and bookings, and renters can request bookings.
- Implementation of a secure authentication mechanism using JWT.
- Designing a responsive and intuitive frontend using React.
- Integration of a robust backend built with Spring Boot and MySQL for data storage.
- Providing detailed insights into vehicle availability, booking history, and user activity.

1.5 Motivation

The motivation behind this project stems from the increasing demand for efficient and secure digital solutions in the vehicle rental industry. With urbanization and rising vehicle-sharing trends, a platform like *Vehico* can bridge the gap between renters and providers, enhancing the overall experience. Additionally, this project provides an opportunity to apply modern development practices and tools, contributing to both personal growth and industry needs.

CHAPTER 2: OBJECTIVES

2.1 Purpose of Vehico

The primary purpose of *Vehico* is to provide a secure, efficient, and user-friendly vehicle rental platform. It aims to bridge the gap between vehicle renters and providers by leveraging technology to streamline rental operations and improve user satisfaction.

2.2 Goals and Objectives

1. **Enhancing Efficiency:**
    - Automating the vehicle rental process to reduce manual effort and operational overhead.
    - Providing real-time updates on vehicle availability and bookings to ensure prompt and reliable services for users.
    - Introducing streamlined workflows for vehicle management, reducing administrative burden.
2. **Ensuring Security:**
    - Implementing JWT-based authentication to secure user data and safeguard sensitive transactions.
    - Utilizing secure password storage and encryption protocols to prevent unauthorized access.
    - Regularly auditing the application for vulnerabilities and ensuring adherence to modern security standards.
3. **Improving User Experience:**
    - Designing a user-friendly interface with intuitive navigation and a responsive layout compatible with various devices.
    - Incorporating features like vehicle search filters, booking summaries, and detailed descriptions to enhance user convenience.
    - Enabling users to track booking statuses and receive notifications for important updates.

4. **Supporting Scalability:**
    - Building a flexible backend architecture that supports increasing user demands and system growth without compromising performance.
    - Implementing database indexing and optimization techniques for efficient data retrieval and management.
    - Preparing the system for future expansion by integrating modular components.
5. **Promoting Transparency:**
    - Providing admins with comprehensive dashboards to monitor bookings, vehicle status, and user activity effectively.
    - Offering renters detailed information, including pricing breakdowns, terms and conditions, and vehicle availability, to ensure informed decision-making.
    - Maintaining transparent policies for booking cancellations, refunds, and dispute resolution.
6. **Encouraging Sustainability:**
    - Supporting eco-friendly vehicle options, such as electric scooters and hybrid cars, to promote sustainable practices.
    - Highlighting vehicles with lower carbon footprints to encourage environmentally conscious decisions.

CHAPTER 3: FEATURES

3.1 Overview of Features

The Vehico project is designed to provide a vehicle rental system with distinct features for both admins and renters. The system ensures secure, role-based access to functionalities, allowing admins to manage the vehicles, bookings, and users, while renters can browse available vehicles and make booking requests. The following sections break down the features based on user roles.

3.2 Role-Based Features

Admin Features

Admins in the Vehico system have full control over the vehicle rental system. They can manage all the vehicle-related operations, approve or reject booking requests, and handle user roles.

Key Admin Features:

- **Add/Edit/Delete Vehicles**: Admins can add new vehicles to the system, edit details of existing vehicles, and remove vehicles that are no longer available for rent.
- **View and Manage Bookings**: Admins can view booking requests and approve or reject them based on availability.
- **Manage Users**: Admins can view user profiles, grant or revoke roles, and handle any issues related to users.
- **Monitor Vehicle Availability**: Admins can check the availability of each vehicle and update availability status (e.g., Available, Unavailable).
- **Access Control**: Only users with an admin role have permission to perform the vehicle management and user management tasks.

Renter Features

Renters in the Vehico system can browse available vehicles, make booking requests, and view their booking history. Renters are restricted from adding, editing, or deleting vehicles and are limited to interacting with vehicles through the booking process.

Key Renter Features:

- **Browse Available Vehicles**: Renters can view a list of available vehicles, filter by type, price, or availability, and view detailed vehicle information.
- **Request a Booking**: Renters can make booking requests for vehicles, specifying the rental period and other necessary details.
- **View Booking History**: Renters can view the status of their previous bookings and check past rental details.
- **Manage Profile**: Renters can manage their personal details, including contact information and preferences.

3.3 Detailed Descriptions of Core Functionalities

Admin Vehicle Management

Admins can add, update, and delete vehicle records. When adding a vehicle, the admin provides key details such as vehicle name, type, make, model, fuel type, capacity, and price per day. Admins are also responsible for uploading images and setting vehicle availability.

Booking Management

Admins can view all booking requests and manage their status. Admins have the ability to approve or reject booking requests based on vehicle availability and the renter's details. Renters are notified once their booking status is updated.

Frontend UI for Admin and Renter

The system's frontend, built with React, presents role-specific views for admins and renters. Admins can access the vehicle management dashboard, while renters can browse vehicles and make bookings. Access control is enforced in the frontend to ensure users see only the features available to their roles.

CHAPTER 4: SYSTEM DESIGN

4.1 High-Level Architecture

The high-level architecture of the Vehico system is designed to provide a seamless user experience with a clear separation of concerns between the front-end and back-end. It consists of a client-side interface (React) interacting with a RESTful API built with Spring Boot. The backend handles user authentication, vehicle management, booking management, and role-based access control.

Key Components:

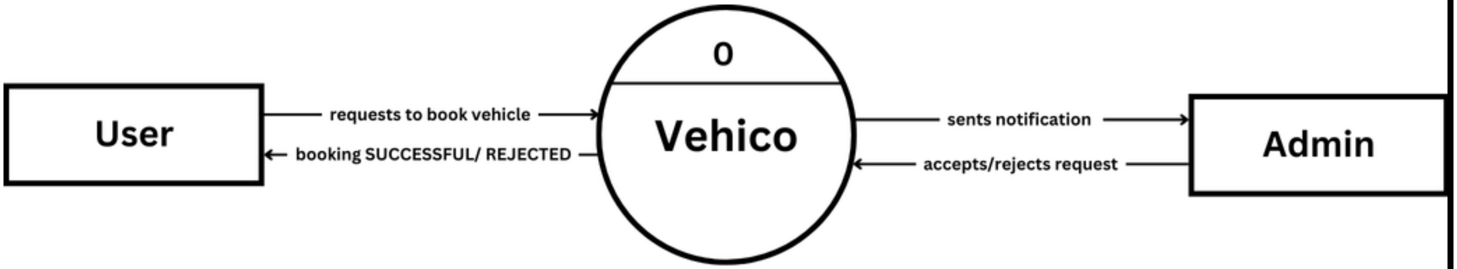
- **Frontend (React):** The user interface that interacts with the backend through HTTP requests. Renters and admins have different views and functionalities based on their roles.
- **Backend (Spring Boot):** The server-side application that processes business logic, handles database operations, and enforces security rules.
- **Database (MySQL):** Stores user information, vehicle details, booking records, and role information.
- **JWT Authentication:** Ensures secure login and role-based access by issuing JSON Web Tokens (JWT) for authenticated users.

4.2 Data Flow Diagram (DFD)

The Data Flow Diagram (DFD) for Vehico illustrates the flow of data through the system, detailing how users, the system, and the database interact with one another.

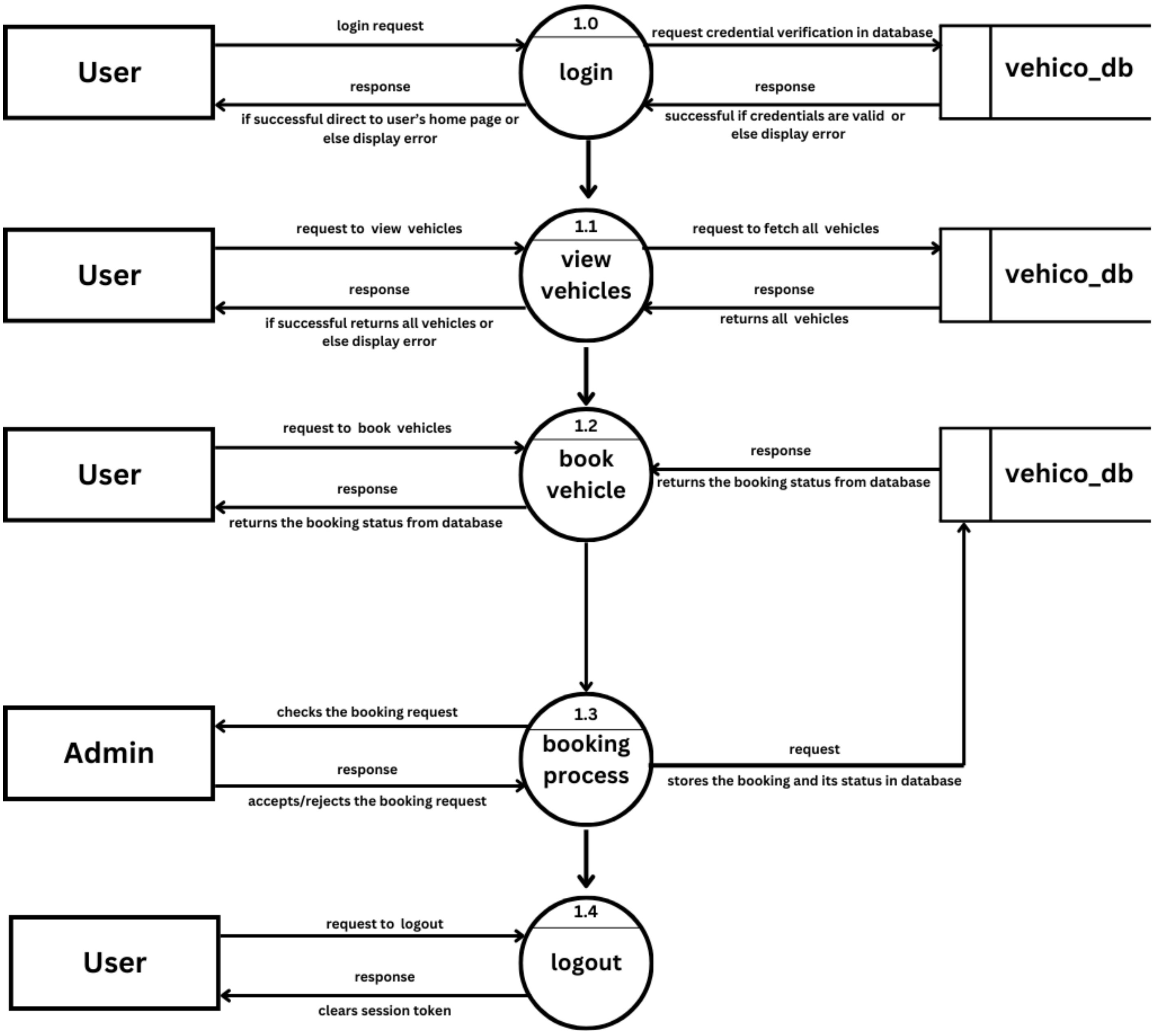
Level 0: Context Diagram This is the top-level view of the system, showing external entities and their interaction with the system.

- **Users (Renters, Admins):** Send requests (e.g., log in, view vehicles, request bookings) to the system.
- **Vehico System:** Processes requests, communicates with the database, and returns responses to users.
- **Database:** Stores all relevant data such as vehicle details, user profiles, and booking records.

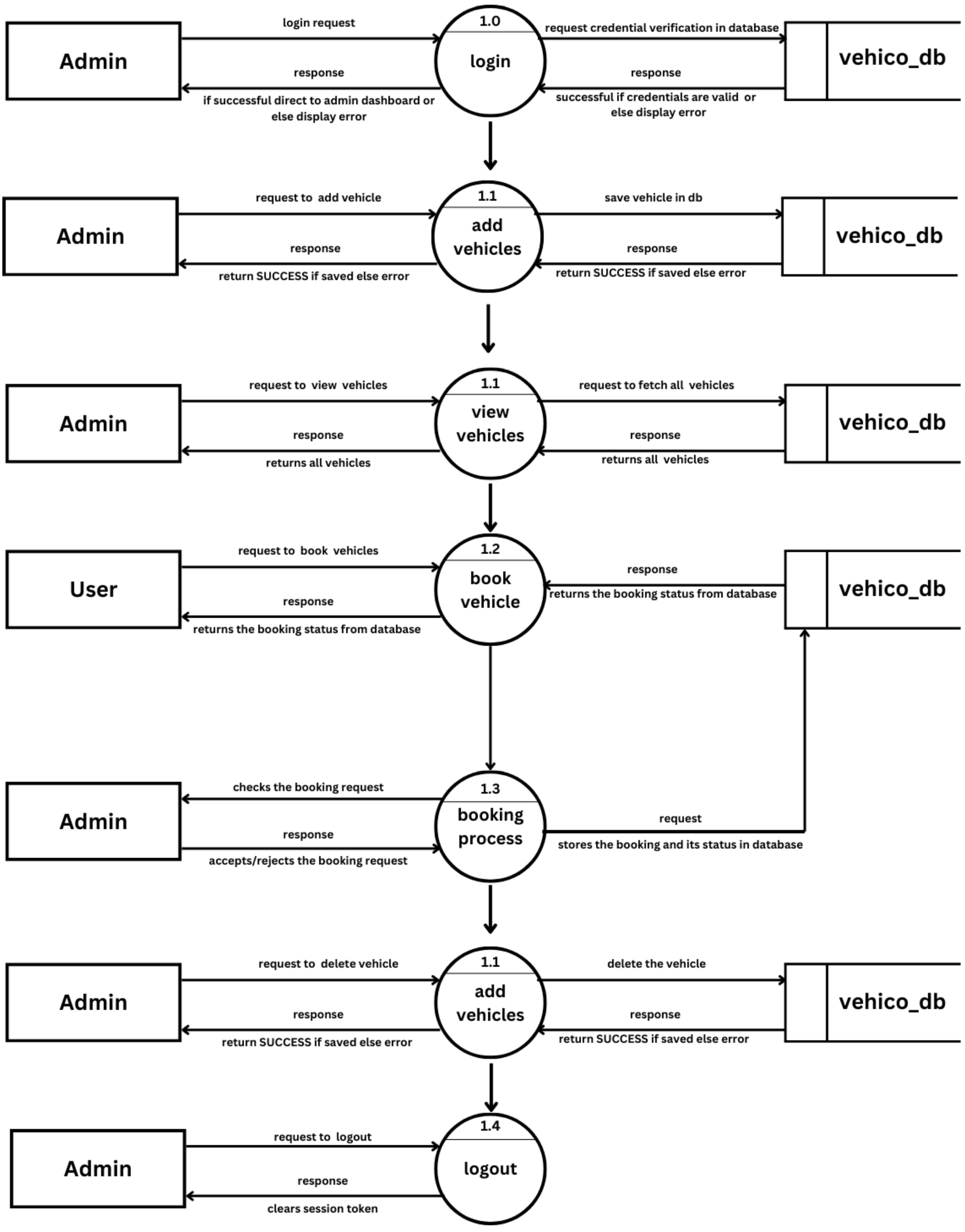


Level 1: Detailed DFD

USER DFD:


User
login request
1.0
login
request credential verification in database
vehico_db
response
if successful direct to user's home page or else display error
response
successful if credentials are valid or else display error
User
request to view vehicles
1.1
view vehicles
request to fetch all vehicles
vehico_db
response
if successful returns all vehicles or else display error
response
returns all vehicles
User
request to book vehicles
1.2
book vehicle
response
returns the booking status from database
vehico_db
response
returns the booking status from database
Admin
checks the booking request
1.3
booking process
request
stores the booking and its status in database
response
accepts/rejects the booking request
User
request to logout
1.4
logout
response
clears session token

ADMIN DFD:



Admin
login request
response
if successful direct to admin dashboard or else display error
1.0
login
request credential verification in database
response
successful if credentials are valid or else display error
vehico_db
Admin
request to add vehicle
response
return SUCCESS if saved else error
1.1
add vehicles
save vehicle in db
response
return SUCCESS if saved else error
vehico_db
Admin
request to view vehicles
response
returns all vehicles
1.1
view vehicles
request to fetch all vehicles
response
returns all vehicles
vehico_db
User
request to book vehicles
response
returns the booking status from database
1.2
book vehicle
response
returns the booking status from database
vehico_db
Admin
checks the booking request
response
accepts/rejects the booking request
1.3
booking process
request
stores the booking and its status in database
Admin
request to delete vehicle
response
return SUCCESS if saved else error
1.1
add vehicles
delete the vehicle
response
return SUCCESS if saved else error
vehico_db
Admin
request to logout
response
clears session token
1.4
logout

4.3 Entity Relationship Diagram (ERD)

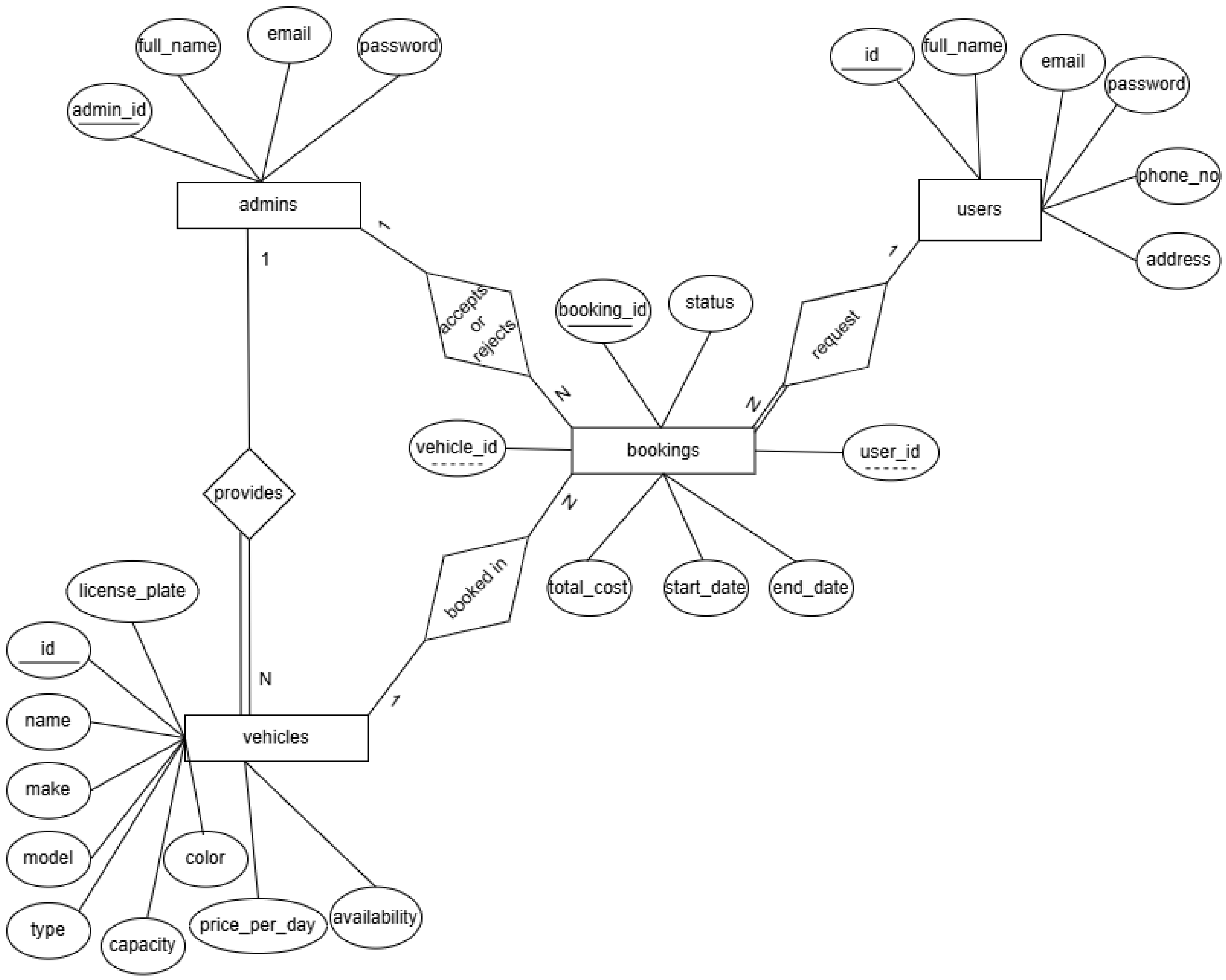
The Entity Relationship Diagram (ERD) visualizes the entities within the Vehico system and their relationships. It defines the structure of the database and shows how data is connected.

Entities:

- **User:** Contains fields like userId, email, password, role (admin/renter), etc.
- **Vehicle:** Includes fields like vehicleId, name, make, model, type, pricePerDay, etc.
- **Booking:** Represents a booking request, containing fields such as bookingId, userId, vehicleId, startDate, endDate, status, etc.
- **Admin:** A specialized user who can manage vehicles and bookings.

Relationships:

- **User to Booking:** A user (renter) can make multiple bookings. This is a one-to-many relationship.
- **Vehicle to Booking:** A vehicle can have many bookings, but each booking is associated with one vehicle. This is a one-to-many relationship.
- **User to Admin:** Admin is a specialized user, which means a user can either be an admin or a renter, but not both at the same time.



4.4 Database Schema Design

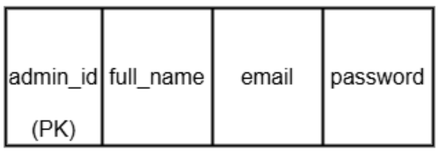
The database schema for Vehico defines the structure of tables, their columns, and the relationships between them. It follows a relational model using MySQL as the database engine.

Tables:

1. **User Table**
   - Columns: userId (PK), email, password,, fullName,phoneNo,address
2. **Vehicle Table**
   - Columns: vehicleId (PK), name, make, model, type, fuelType, capacity, pricePerDay, description, availability, licensePlate
3. **Booking Table**
   - Columns: bookingId (PK), userId (FK), vehicleId (FK), startDate, endDate, status,totalCost
4. **Admin Table**
   - Columns: adminId (PK), fullName,email,password

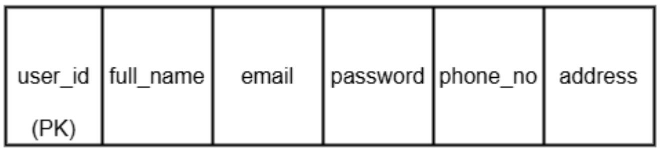
The schema also includes foreign key constraints to ensure referential integrity between entities (e.g., a Booking must have an associated User and Vehicle).

admins Table



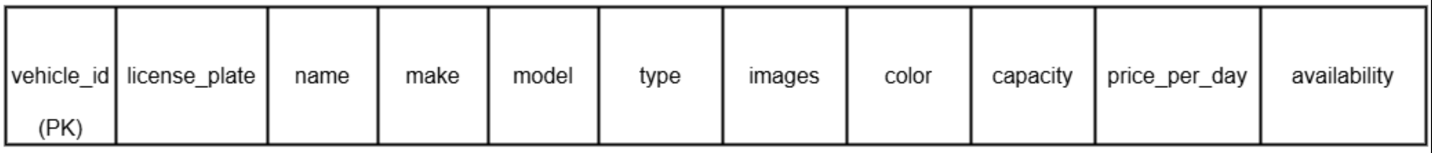
admin_id (PK)	full_name	email	password

users Table



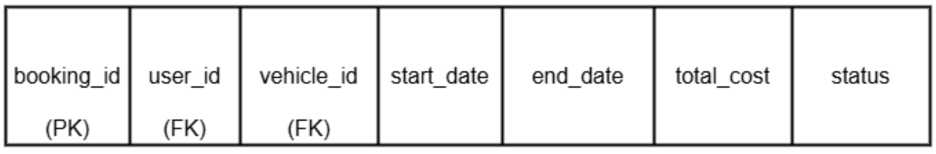
user_id (PK)	full_name	email	password	phone_no	address

vehicles Table



vehicle_id (PK)	license_plate	name	make	model	type	images	color	capacity	price_per_day	availability

bookings Table



booking_id (PK)	user_id (FK)	vehicle_id (FK)	start_date	end_date	total_cost	status

4.5 Data Dictionary

The Data Dictionary provides detailed descriptions of the database tables, fields, and their relationships within the Vehico system. It serves as a reference for understanding the meaning, data type, and constraints for each field in the system's database.

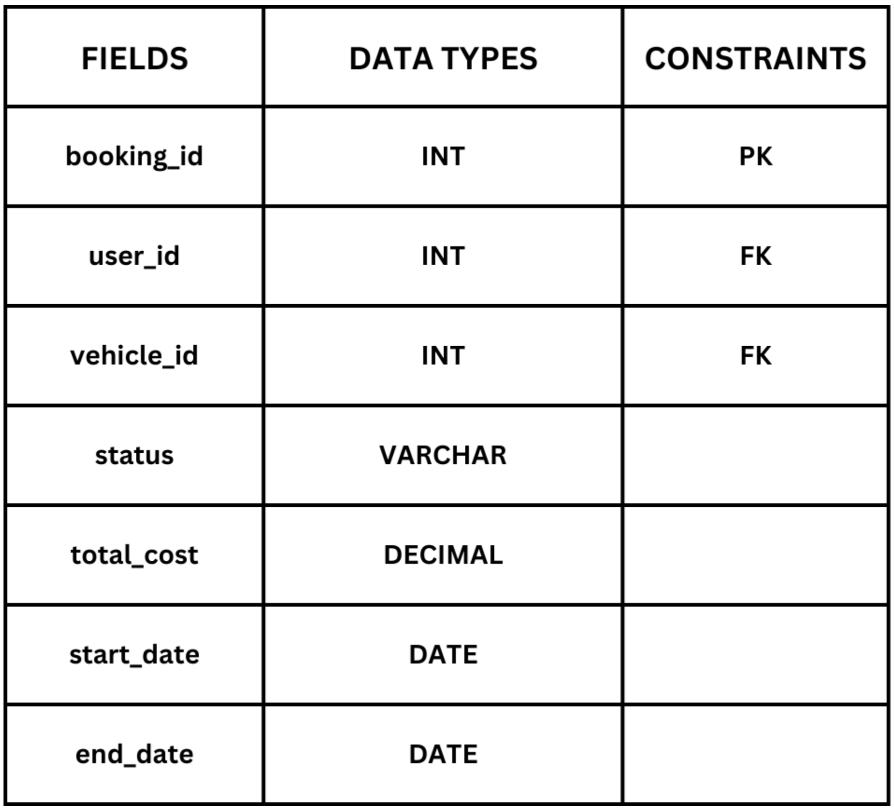
Table 1: User

FIELDS	DATA TYPES	CONSTRAINTS
user_id	INT	PK
full_name	VARCHAR	
email	VARCHAR	
password	VARCHAR	
phone_no	VARCHAR	
address	VARCHAR	

Table 2: Admin

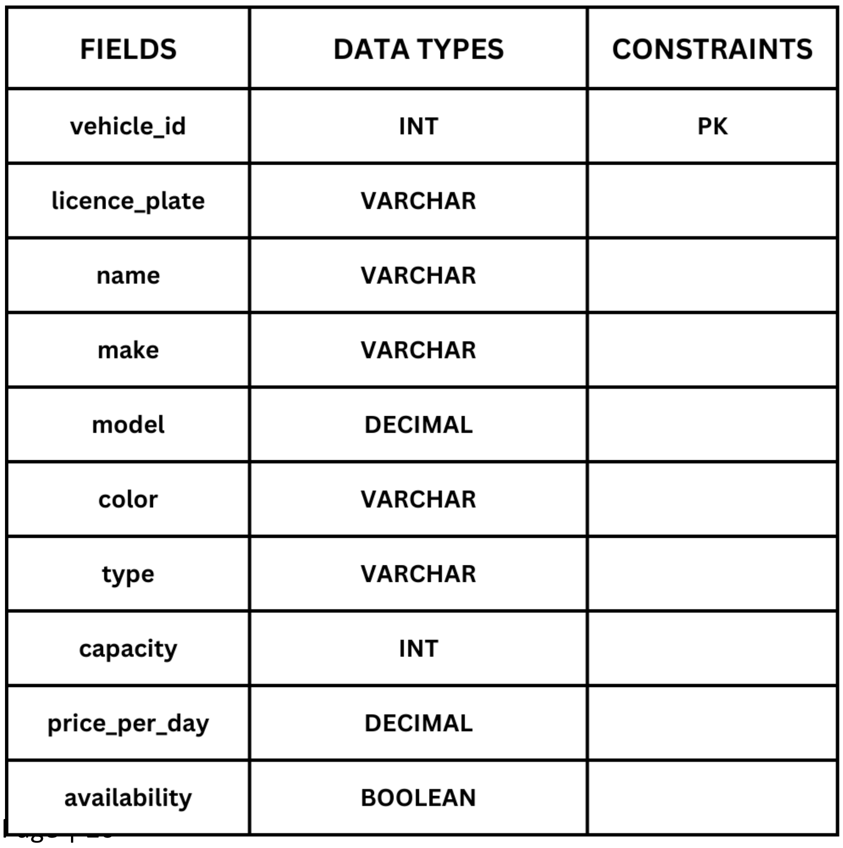
FIELDS	DATA TYPES	CONSTRAINTS
admin_id	INT	PK
full_name	VARCHAR	
email	VARCHAR	
password	VARCHAR	

3: Booking



FIELDS	DATA TYPES	CONSTRAINTS
booking_id	INT	PK
user_id	INT	FK
vehicle_id	INT	FK
status	VARCHAR	
total_cost	DECIMAL	
start_date	DATE	
end_date	DATE	

Table 4: Vehicle



FIELDS	DATA TYPES	CONSTRAINTS
vehicle_id	INT	PK
licence_plate	VARCHAR	
name	VARCHAR	
make	VARCHAR	
model	DECIMAL	
color	VARCHAR	
type	VARCHAR	
capacity	INT	
price_per_day	DECIMAL	
availability	BOOLEAN	

CHAPTER 5: SYSTEM ARCHITECTURE

5.1 Application Architecture

Backend (Spring Boot)

The backend of the vehicle rental system is built using **Spring Boot**, which provides a robust framework for building RESTful services, managing dependencies, and simplifying development with features like annotations and integration with databases. The backend handles business logic, user authentication, and interaction with the database.

Frontend (React)

The frontend is developed using **React**, a popular JavaScript library for building interactive user interfaces. React provides dynamic component rendering, state management, and seamless integration with the REST API for data exchange.

REST API Layer

The system's communication between the frontend and backend is facilitated by a **REST API layer**. This layer is responsible for exposing endpoints that allow the frontend to send and receive data securely and efficiently. It supports CRUD operations for users, vehicles, and bookings, ensuring a seamless flow of data across the system.

5.2 Technologies Used

Programming Languages

- **Java** (for backend development)
- **JavaScript** (for frontend development)

Frameworks and Libraries

- **Spring Boot** (backend framework)
- **Hibernate** (for Object-Relational Mapping)
- **React** (frontend library)
- **Axios** (for making HTTP requests from React)
- **Lombok** (for reducing boilerplate code in Java)

Database

- **MySQL** is used as the relational database for storing user, vehicle, and booking information. It is configured with Spring Data JPA for efficient data access and management.

Tools and Version Control

- **Maven** (for project dependency management)
- **Postman** (for API testing)
- **Git** (for version control)
- **IntelliJ IDEA** (IDE for backend development)
- **Visual Studio Code** (IDE for frontend development)

5.3 Component Overview

The system is divided into several key components:

1. **Authentication and Authorization**
    - Implements JWT-based authentication to secure endpoints.
    - Ensures role-based access control (Admin and Renter).
2. **Vehicle Management**
    - Enables admins to add, update, and delete vehicles.
    - Provides renters with a list of available vehicles for booking.
3. **Booking Management**
    - Allows renters to request bookings.
    - Enables admins to accept or reject booking requests.
4. **Frontend Components**
    - Dynamic forms for booking requests and vehicle listings.
    - Real-time updates for booking statuses and vehicle availability.

CHAPTER 6: SECURITY IMPLEMENTATION

6.1 JWT-Based Authentication

The system implements **JWT (JSON Web Token)**-based authentication to secure endpoints and ensure a stateless communication protocol. Users receive a token upon successful login, which is used to validate their identity and access rights during subsequent requests.

6.2 Password Encryption

To ensure user security, passwords are encrypted using **bcrypt hashing** before storing them in the database. This ensures that even if the database is compromised, the passwords remain protected and unreadable.

6.3 Role-Based Access Control

Role-based access control (RBAC) ensures that users have access only to the resources and actions permitted for their roles. The system defines two primary roles:

- **Admin**: Can manage vehicles, users, and bookings.
- **Renter**: Can view available vehicles and make booking requests.

RBAC is enforced at the endpoint level using Spring Security annotations, ensuring fine-grained control over resource access.

CHAPTER 7: CODE IMPLEMENTATION

7.1 Key Backend Code Snippets

This section highlights key backend code snippets that are crucial for implementing the core features of the Vechico project.

Backend Structure:

AdminController.java:

```
package com.abhishek.VehicalRentalSystem.controllers;

import com.abhishek.VehicalRentalSystem.dto.requests.CreateVehicleRequest;
import com.abhishek.VehicalRentalSystem.entities.Booking;
import com.abhishek.VehicalRentalSystem.entities.Vehicle;
import com.abhishek.VehicalRentalSystem.enums.BOOKING_STATUS;
import com.abhishek.VehicalRentalSystem.services.admin.AdminService;
import com.abhishek.VehicalRentalSystem.utils.JwtUtil;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/admin")
public class AdminController {

    private final AdminService adminService;
    private final JwtUtil jwtUtil; // Inject JwtUtil

    public AdminController(AdminService adminService, JwtUtil jwtUtil) {
        this.adminService = adminService;
        this.jwtUtil = jwtUtil;
    }

    // Manage Vehicles
    @PostMapping("/vehicles")
    public ResponseEntity<Vehicle> addVehicle(@RequestBody CreateVehicleRequest
createVehicleRequest,
                                              @RequestHeader("Authorization") String token) {
        // Admin authentication and role check would be done inside adminService
        Vehicle vehicle = adminService.addVehicle(createVehicleRequest);
        return new ResponseEntity<>(vehicle, HttpStatus.CREATED);
    }

    @PutMapping("/vehicles/{vehicleId}")
    public ResponseEntity<Vehicle> updateVehicle(@PathVariable Long vehicleId,
                                                 @RequestBody CreateVehicleRequest updatedVehicle,
                                                 @RequestHeader("Authorization") String token) {
        // Admin authentication and role check would be done inside adminService
        Vehicle vehicle = adminService.updateVehicle(vehicleId, updatedVehicle);
        return new ResponseEntity<>(vehicle, HttpStatus.OK);
    }

    @DeleteMapping("/vehicles/{vehicleId}")
    public ResponseEntity<String> deleteVehicle(@PathVariable Long vehicleId,
                                                @RequestHeader("Authorization") String token) {
        // Admin authentication and role check would be done inside adminService
        adminService.deleteVehicle(vehicleId);
        return new ResponseEntity<>("Vehicle deleted successfully",
HttpStatus.NO_CONTENT);
    }

    @GetMapping("/vehicles")
    public ResponseEntity<List<Vehicle>>
getAllVehicles(@RequestHeader("Authorization") String token) {
        // Admin authentication and role check would be done inside adminService
        List<Vehicle> vehicles = adminService.getAllVehicles();
        return new ResponseEntity<>(vehicles, HttpStatus.OK);
    }

    @GetMapping("/vehicles/{vehicleId}")
    public ResponseEntity<Vehicle> getVehicleById(@PathVariable Long vehicleId,
                                                  @RequestHeader("Authorization") String token) {
        // Admin authentication and role check would be done inside adminService
        Vehicle vehicle = adminService.getVehicleById(vehicleId);
        return new ResponseEntity<>(vehicle, HttpStatus.OK);
    }

    @GetMapping("/vehicles/search")
    public ResponseEntity<List<Vehicle>> searchVehicles(@RequestParam String keyword,
                                                        @RequestHeader("Authorization") String token) {
        // Admin authentication and role check would be done inside adminService
        List<Vehicle> vehicles = adminService.searchVehicles(keyword);
        return new ResponseEntity<>(vehicles, HttpStatus.OK);
    }

    // Manage Bookings
    @GetMapping("/bookings")
    public ResponseEntity<List<Booking>>
getAllBookings(@RequestHeader("Authorization") String token) {
        // Admin authentication and role check would be done inside adminService
        List<Booking> bookings = adminService.getAllBookings();
        return new ResponseEntity<>(bookings, HttpStatus.OK);
    }

    @PutMapping("/bookings/{bookingId}")
    public ResponseEntity<Booking> updateBookingStatus(@PathVariable Long
bookingId,
                                                        @RequestParam BOOKING_STATUS status,
                                                        @RequestHeader("Authorization") String token) {
        // Admin authentication and role check would be done inside adminService
        Booking booking = adminService.updateBookingStatus(bookingId, status, token);
        return new ResponseEntity<>(booking, HttpStatus.OK);
    }

    @GetMapping("/bookings/status")
    public ResponseEntity<List<Booking>> getBookingsByStatus(@RequestParam
BOOKING_STATUS status,
                                                              @RequestHeader("Authorization") String token) {
        // Admin authentication and role check would be done inside adminService
        List<Booking> bookings = adminService.getBookingsByStatus(status);
        return new ResponseEntity<>(bookings, HttpStatus.OK);
    }
}
```

CustomerController.java:

```
package com.abhishek.VehicalRentalSystem.controllers;

import com.abhishek.VehicalRentalSystem.dto.requests.BookingRequest;
import com.abhishek.VehicalRentalSystem.entities.Booking;
import com.abhishek.VehicalRentalSystem.entities.Vehicle;
import com.abhishek.VehicalRentalSystem.services.customer.CustomerService;
import com.abhishek.VehicalRentalSystem.utils.JwtUtil;
import jakarta.validation.Valid;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/customer")
public class CustomerController {

   private final CustomerService customerService;
   private final JwtUtil jwtUtil; // Inject JWTUtil

   public CustomerController(CustomerService customerService, JwtUtil jwtUtil) {
      this.customerService = customerService;
      this.jwtUtil = jwtUtil;
   }

   // Get all available vehicles
   @GetMapping("/vehicles")
   public ResponseEntity<List<Vehicle>>
getAllAvailableVehicles(@RequestParam(value = "type", required = false) String type)
{
      List<Vehicle> vehicles = customerService.getAllAvailableVehicles(type);
      return new ResponseEntity<>(vehicles, HttpStatus.OK);
   }

   // Get vehicle details by ID
   @GetMapping("/vehicles/{vehicleId}")
   public ResponseEntity<Vehicle> getVehicleById(@PathVariable Long vehicleId) {
      Vehicle vehicle = customerService.getVehicleById(vehicleId);
      return new ResponseEntity<>(vehicle, HttpStatus.OK);
   }

   // Customer creates a booking
   @PostMapping("/bookings")
   public ResponseEntity<Booking> createBooking(@RequestBody @Valid
BookingRequest bookingRequest,
                                          @RequestHeader("Authorization") String token) {
       // Pass the token to the service
       Booking booking = customerService.createBooking(bookingRequest, token);
       return new ResponseEntity<>(booking, HttpStatus.CREATED);
   }
   // Get all bookings of the customer
   @GetMapping("/bookings")
   public ResponseEntity<List<Booking>>
getCustomerBookings(@RequestHeader("Authorization") String token) {
       Long customerId = Long.valueOf(jwtUtil.extractUserId(token)); // Use JWTUtil to
extract user ID from token
       List<Booking> bookings = customerService.getCustomerBookings(customerId,
token);
       return new ResponseEntity<>(bookings, HttpStatus.OK);
   }
   // Get details of a specific booking
   @GetMapping("/bookings/{bookingId}")
   public ResponseEntity<Booking> getBookingById(@PathVariable Long bookingId) {
       Booking booking = customerService.getBookingById(bookingId);
       return new ResponseEntity<>(booking, HttpStatus.OK);
   }

   // Cancel a booking
   @DeleteMapping("/bookings/{bookingId}")
   public ResponseEntity<String> cancelBooking(@PathVariable Long bookingId,
@RequestHeader("Authorization") String token) {
       // Pass the token to the service for cancellation
       boolean isCancelled = customerService.cancelBooking(bookingId, token);
       if (isCancelled) {
           return new ResponseEntity<>("Booking cancelled successfully",
HttpStatus.OK);
       }
       return new ResponseEntity<>("Booking not found or cannot be cancelled",
HttpStatus.BAD_REQUEST);
   }

   @GetMapping("/vehicles/search")
   public ResponseEntity<List<Vehicle>> searchVehicles(@RequestParam String
keyword) {
       List<Vehicle> vehicles = customerService.searchVehicles(keyword);
       return new ResponseEntity<>(vehicles, HttpStatus.OK);
   }}
```

7.2 Key Restful API’s

Admin APIs

1. **POST /admin/vehicles**
   Adds a new vehicle to the system (Admin only).
2. **PUT /admin/vehicles/{id}**
   Updates the details of an existing vehicle (Admin only).
3. **DELETE /admin/vehicles/{id}**
   Deletes a vehicle from the system (Admin only).
4. **GET /admin/vehicles**
   Retrieves a list of all vehicles (Admin only).
5. **GET /admin/bookings**
   Retrieves all booking requests (Admin only).
6. **PUT /admin/bookings/{id}/accept**
   Accepts a booking request (Admin only).
7. **PUT /admin/bookings/{id}/reject**
   Rejects a booking request (Admin only).

Customer APIs

1. **GET /customer/vehicles**
   Retrieves a list of all available vehicles (Customer only).
2. **POST /customer/bookings**
   Allows the customer to request a booking for a vehicle (Customer only).
3. **GET /customer/bookings/{id}**
   Retrieves details of a specific booking request (Customer only).
4. **GET /customer/bookings**
   Retrieves a list of all booking requests made by the customer (Customer only).
5. **PUT /customer/bookings/{id}/cancel**
   Allows the customer to cancel a booking (Customer only).

7.3 Key Frontend Code Snippets

This section demonstrates key frontend code snippets for interacting with the backend APIs.

FrontEnd ProjectStructure :

Home.jsx:

```
import React from 'react'
import './Home.css'
import MultiItemCarousel from './MultiItemCarousel'
import {luxuryVehicles} from './luxuryVehicles'
import VehicleCard from '../Vehicle/VehicleCard'

export const Home = () => {
  return (
    <div className='pb-10'>
      <section className='banner -z-50 relative flex flex-col justify-center items-center'>
        <div className='w-[60vw] z-10 text-center'>
          <p className='text-2xl lg:text-6xl font-bold z-10 py-5'>Vehico</p>
          <p className='z-10 text-gray-300 text-sm lg:text-xl'>
            Welcome to Vehico, your one-stop solution for hassle-free vehicle rentals! Whether
you're looking for a car, bike, or even a truck, Vehico connects renters with a wide variety
of vehicles to suit every need.
            Our platform is designed to provide a seamless rental experience with features like:
            Easy Booking: Explore and book vehicles effortlessly with just a few clicks.
            Flexible Options: Choose from various vehicle types such as cars, bikes, trucks,
buses, SUVs, and more.
            Secure Transactions: Enjoy peace of mind with secure payment methods and
verified users.
            Personalized Dashboard: Admins can manage vehicles and bookings efficiently,
while customers can view and track their rentals.
            Real-Time Notifications: Stay updated with booking approvals, rejections, or status
changes instantly.
```

```
          Vehico aims to bridge the gap between vehicle providers and renters, offering a
convenient, reliable, and trustworthy platform. Whether you're planning a short trip or
need a vehicle for business, we're here to make it happen!

          </p>

        </div>

        <div className='cover absolute top-0 right-0'>


        </div>

        <div className='fadout '>


        </div>

      </section>

      <section className='p-10 lg:py-10 lg:px-20'>

        <p className='text-2xl font-semibold text-gray-400 py-3 pb-5'>Top Luxury Cars</p>

        <MultiItemCarousel />

      </section>


      <section className='px-5 lg:px-20 pt-5'>

        <h1 className='text-2xl font-semibold text-gray-400 pb-5'>Luxury Cars We Have
</h1>

        <div className='flex flex-wrap items-center justify-around gap-3'>

          {

            luxuryVehicles.map((item,index)=>

            <VehicleCard key={index} name={item.name} image={item.imageUrl}
description={item.description} price={item.pricePerDay}/>

            )

          }

        </div>

      </section>

    </div>

  )

}
```

CHAPTER 8: CHALLENGES AND SOLUTIONS

8.1 Technical Challenges Faced

Throughout the development of the **Vechico** vehicle rental system, several technical challenges arose, which required innovative solutions to ensure the project was successful.

1. **Vehicle Availability Management**
    - **Challenge**: Managing the real-time availability of vehicles proved difficult, especially when dealing with multiple bookings and ensuring no double bookings occur.
    - **Solution**: Implemented a simple flag (isAvailable) within the vehicle entity, and synchronized the vehicle status during booking requests. Additional checks were added to verify availability before confirming any booking.
2. **User Authentication and Authorization**
    - **Challenge**: Implementing secure and efficient user authentication and authorization was crucial, especially when using JWT for secure login and access control.
    - **Solution**: Used **JWT** tokens for stateless authentication. Each API call required the token to verify the user's role (Admin or Customer). This ensured proper access control throughout the system.
3. **Handling Booking Conflicts**
    - **Challenge**: Managing booking requests for the same vehicle on overlapping dates was challenging.
    - **Solution**: Designed the booking system to check the dates for conflicts before processing a new booking. A status of "pending" was set initially, allowing the admin to approve or reject the booking after verification.

8.2 Security Concerns Addressed

Security was a top priority in the **Vechico** project, and several concerns were addressed to ensure the integrity of the system:

1. **Data Encryption**
   - **Concern**: Protecting sensitive user and vehicle data from unauthorized access.
   - **Solution**: Encrypted sensitive data such as user passwords using **BCrypt** hashing. Communication between the frontend and backend was secured using **HTTPS** to prevent eavesdropping.
2. **Role-Based Access Control**
   - **Concern**: Unauthorized access by users to restricted resources.
   - **Solution**: Used role-based access control (RBAC) to differentiate between Admin and Customer roles, restricting access to certain endpoints based on the user's role.
3. **SQL Injection Prevention**
   - **Concern**: Protecting the database from SQL injection attacks.
   - **Solution**: Used **JPA** and **Prepared Statements** to prevent SQL injection attacks by ensuring queries are safely executed.
4. **Cross-Site Request Forgery (CSRF) Prevention**
   - **Concern**: Preventing unauthorized actions from being performed on behalf of the user.
   - **Solution**: Enabled **CSRF** protection in Spring Security to prevent attackers from submitting forms on behalf of authenticated users.

8.3 Lessons Learned

The development of the **Vechico** project provided several valuable lessons that can be applied to future projects:

1. **Importance of Proper Database Design**
    - Ensuring proper relationships and indexes in the database from the outset can significantly improve performance and avoid future complexities. In particular, understanding how **foreign keys** and **cascade operations** work is essential for maintaining data integrity.
2. **Effective Error Handling**
    - Implementing a global error handling mechanism in the backend with clear error messages allows for faster debugging and better user experience. Using exception handling with **@ControllerAdvice** in Spring Boot improved overall stability.
3. **Testing and Validation Are Key**
    - Proper testing, including unit tests for business logic and integration tests for API endpoints, is crucial to ensure the system behaves as expected. Validating user input before it reaches the database is essential for preventing issues.
4. **User Experience Matters**
    - Ensuring a smooth and intuitive user interface is just as important as backend functionality. Using **React** and implementing responsive design ensured that users had an easy-to-use platform, regardless of the device they used.
5. **Scalability Considerations**
    - Even though this project was relatively small, learning how to design the system with scalability in mind is crucial for handling growth in the future. Separating concerns and modularizing the code is key to making the system more scalable.

CHAPTER 9: FUTURE ENHANCEMENTS

9.1 Features Planned for Future Versions

While the current version of **Vechico** provides core functionalities for vehicle rental, several new features are planned for future versions to enhance the user experience and add more flexibility:

1. **Payment Gateways**
   - **Description**: Integrating payment options such as credit/debit cards, UPI, and wallets to allow users more payment flexibility.
   - **Benefits**: Increases convenience for users and expands the system's usability across different regions..
2. **Vehicle Ratings and Reviews**
   - **Description**: Allowing customers to rate and review vehicles after use, helping future customers make informed decisions.
   - **Benefits**: Adds a layer of transparency and builds trust among users.
3. **Push Notifications**
   - **Description**: Sending push notifications to users for important events, such as booking confirmations, availability of vehicles, and upcoming bookings.
   - **Benefits**: Keeps users informed in real-time, improving engagement.
4. **Vehicle Insurance Integration**
   - **Description**: Adding the option for customers to purchase vehicle insurance during the booking process.
   - **Benefits**: Provides added security and convenience for customers, offering them peace of mind.

9.2 Scalability Considerations

As the application grows, it is important to plan for scalability to handle increased traffic, data, and user demand. Here are some considerations for scaling the **Vechico** project in the future:

1. **Microservices Architecture**
   - **Consideration**: Transitioning from a monolithic architecture to a microservices-based design will allow the system to scale more easily by separating concerns into distinct services (e.g., user management, booking management, vehicle inventory).
   - **Benefits**: Each service can be scaled independently based on demand, improving performance and maintainability.
2. **Database Sharding and Replication**
   - **Consideration**: As the user and booking data grows, implementing database sharding (splitting data across multiple databases) and replication (creating read replicas) will help distribute the load and improve query performance.
   - **Benefits**: Increases the system's ability to handle large amounts of data and concurrent requests.
3. **Caching Strategies**
   - **Consideration**: Implementing caching mechanisms for frequently accessed data, such as vehicle lists and booking details, will reduce database load and speed up response times.
   - **Benefits**: Reduces latency and improves overall user experience.
4. **Cloud-Based Infrastructure**
   - **Consideration**: Migrating the application to a cloud-based infrastructure (e.g., AWS, Google Cloud, Azure) will provide flexibility to scale resources up or down based on demand.
   - **Benefits**: Allows the system to scale quickly and efficiently without the need for significant upfront investment in hardware.

9.3 User Feedback and Suggestions

User feedback plays a crucial role in improving the system. As **Vechico** continues to grow, collecting and analyzing user suggestions will help prioritize enhancements and ensure the platform meets user needs.

1. **User Interface Improvements**
   - **Feedback**: Some users have requested a more modern and intuitive user interface for both the admin and customer portals.
   - **Action**: Future versions will focus on redesigning the UI, making it more responsive, and enhancing the visual experience with better navigation.
2. **More Vehicle Categories**
   - **Feedback**: Customers have asked for additional vehicle categories, such as luxury cars and bikes.
   - **Action**: In the future, the platform will offer a wider range of vehicle types, catering to different customer needs.
3. **Faster Booking Process**
   - **Feedback**: Some users have suggested streamlining the booking process to make it faster and more efficient.
   - **Action**: The next version will optimize the booking flow by reducing the number of steps and providing a simpler, more intuitive experience.
4. **Improved Search Filters**
   - **Feedback**: Users have requested more advanced search filters for finding vehicles based on specific criteria like price range, fuel type, and vehicle features.
   - **Action**: Enhanced search and filtering options will be introduced to allow users to find the most suitable vehicles quickly.
5. **Better Customer Support Integration**
   - **Feedback**: Users have expressed the need for better customer support, including live chat and quicker response times.
   - **Action**: Future versions will include integrated customer support features, such as live chat and quicker issue resolution systems.

CHAPTER 10: CONCLUSION

10.1 Summary of the Project

The **Vechico** vehicle rental system was developed to provide a seamless and efficient platform for managing vehicle rentals. Built with Java, Spring Boot, MySQL, and React, the system was designed with two primary user roles: **Admin** and **Customer**.

The **Admin** is responsible for adding, updating, and deleting vehicles, while the **Customer** can browse available vehicles, make booking requests, and view booking details. The system also includes crucial features such as JWT-based authentication for secure login, role-based access control, and email notifications for real-time updates.

Key aspects of the system include:

- **Vehicle Management**: Admins can add, update, and delete vehicles, while customers can search for and book available vehicles.
- **Booking System**: The platform allows customers to make booking requests, which can then be accepted or rejected by the admin.
- **Security**: The project employs robust security measures, including JWT for authentication, CSRF protection, and secure password hashing.
- **Scalability**: Future plans for scalability include adopting a microservices architecture and implementing caching and load balancing strategies.

Overall, the project successfully demonstrates the implementation of a vehicle rental system with a focus on security, usability, and performance.

10.2 Impact and Contributions

The **Vechico** project provides a valuable solution for the vehicle rental industry by streamlining the process of managing and renting vehicles. It contributes to both the academic and practical understanding of web development, particularly in the areas of:

1. **Web Application Development**: Through the integration of technologies like Spring Boot for the backend, React for the frontend, and MySQL for data storage, the project showcases a comprehensive understanding of full-stack web development.

2. **Security Best Practices**: The project emphasizes the importance of implementing security measures such as JWT, password encryption, and role-based access control. These practices ensure that user data remains safe and that only authorized individuals can perform sensitive operations.
3. **User-Centric Design**: The system's intuitive interface and role-specific features ensure that both admins and customers can efficiently use the platform. The feedback collected from users has been instrumental in planning for future enhancements, ensuring the system evolves to meet user needs.
4. **Scalable Architecture**: The project incorporates considerations for future scalability, which are essential for any growing system. By planning for microservices, database sharding, and cloud infrastructure, the system is designed to handle increasing demand without compromising performance.

In conclusion, **Vechico** has laid a solid foundation for a practical, user-friendly, and scalable vehicle rental platform, with further enhancements planned to make it even more efficient and feature-rich. It has made significant contributions to both personal learning and the broader field of vehicle rental solutions.

CHAPTER 11: APPENDICES

11.1 Full Database Schema

The database schema for the **Vechico** vehicle rental system consists of several tables that were automatically generated using Spring Boot's JPA/Hibernate framework. These tables include:

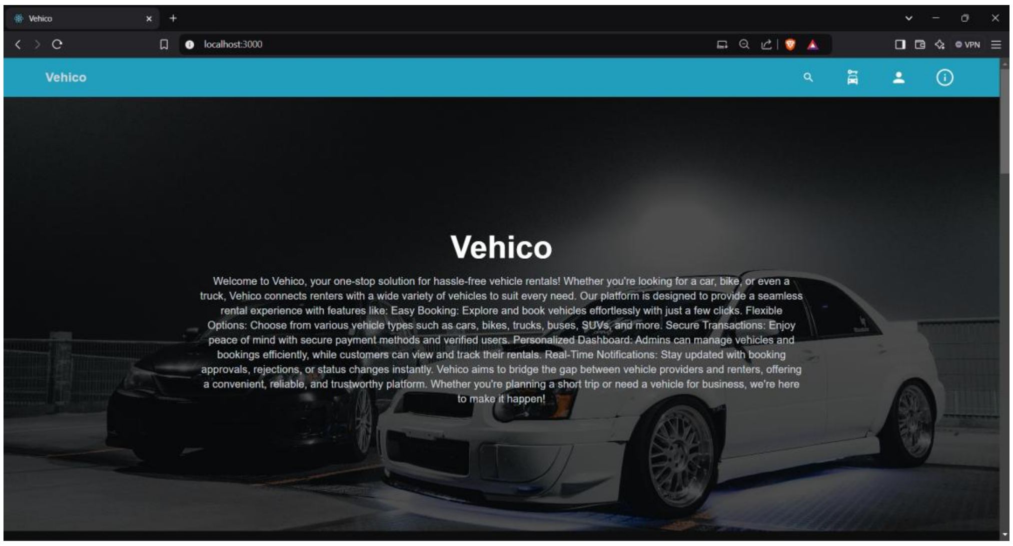
- **Admin Table**: Stores admin details such as email, password, and role.
- **User Table**: Stores customer information including email, name, and contact details.
- **Vehicle Table**: Contains details of the vehicles available for rent, including name, type, fuel type, and price.
- **Booking Table**: Tracks booking requests made by users, linked to both the customer and the vehicle.

The relationships between these entities were also automatically managed by JPA, using annotations such as @ManyToOne and @OneToMany to ensure the integrity and structure of the data.

11.2 Screenshots of UI

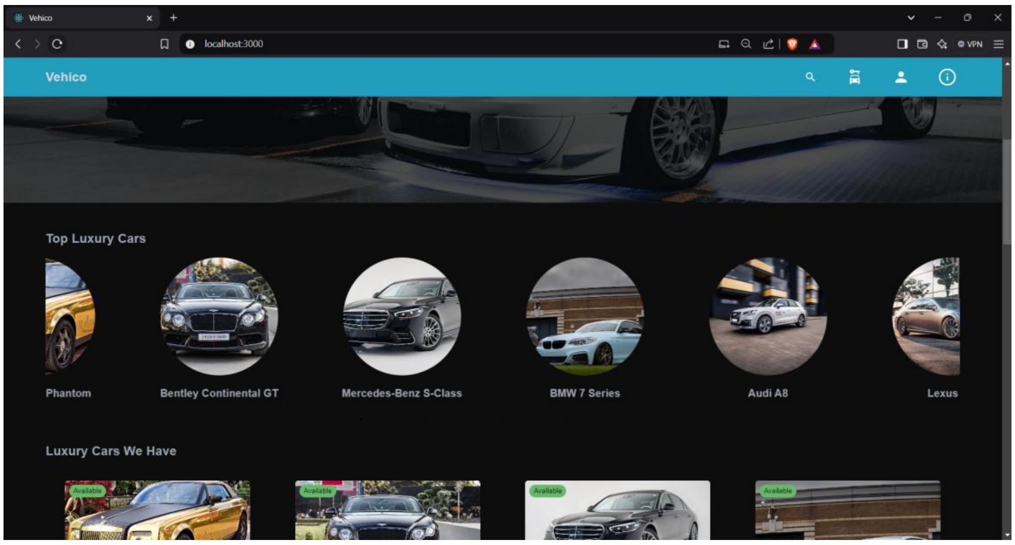
Below are screenshots of the key sections of the **Vechico** system's user interface

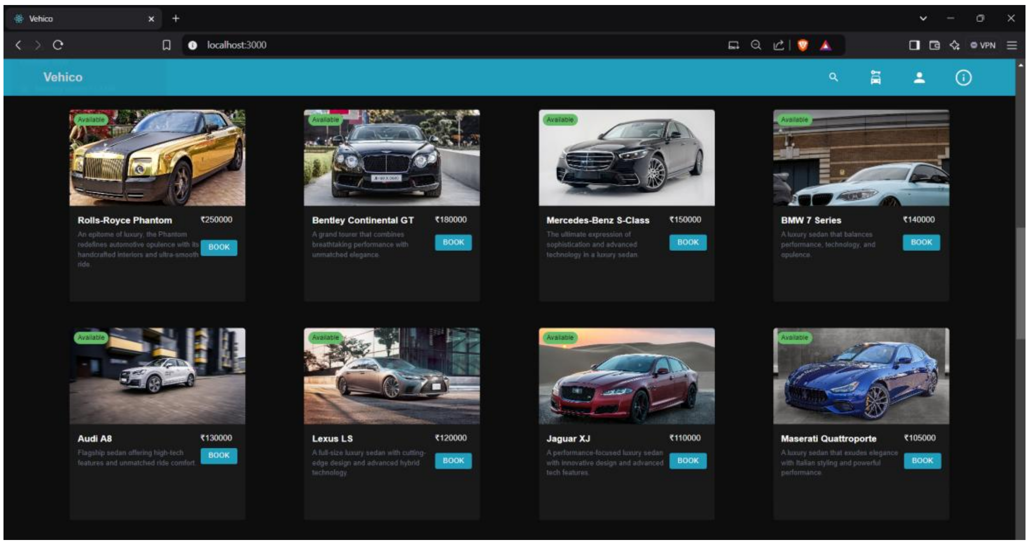
1. Home Page



Vehico

Welcome to Vehico, your one-stop solution for hassle-free vehicle rentals! Whether you're looking for a car, bike, or even a truck, Vehico connects renters with a wide variety of vehicles to suit every need. Our platform is designed to provide a seamless rental experience with features like: Easy Booking: Explore and book vehicles effortlessly with just a few clicks. Flexible Options: Choose from various vehicle types such as cars, bikes, trucks, buses, SUVs, and more. Secure Transactions: Enjoy peace of mind with secure payment methods and verified users. Personalized Dashboard: Admins can manage vehicles and bookings efficiently, while customers can view and track their rentals. Real-Time Notifications: Stay updated with booking approvals, rejections, or status changes instantly. Vehico aims to bridge the gap between vehicle providers and renters, offering a convenient, reliable, and trustworthy platform. Whether you're planning a short trip or need a vehicle for business, we're here to make it happen!





Rolls-Royce Phantom ₹250000
An epitome of luxury, the Phantom redefines automotive opulence with its handcrafted interiors and ultra-smooth ride.

Bentley Continental GT ₹180000
A grand tourer that combines breathtaking performance with unmatched elegance.

Mercedes-Benz S-Class ₹150000
The ultimate expression of sophistication and advanced technology in a luxury sedan.

BMW 7 Series ₹140000
A luxury sedan that balances performance, technology, and opulence.

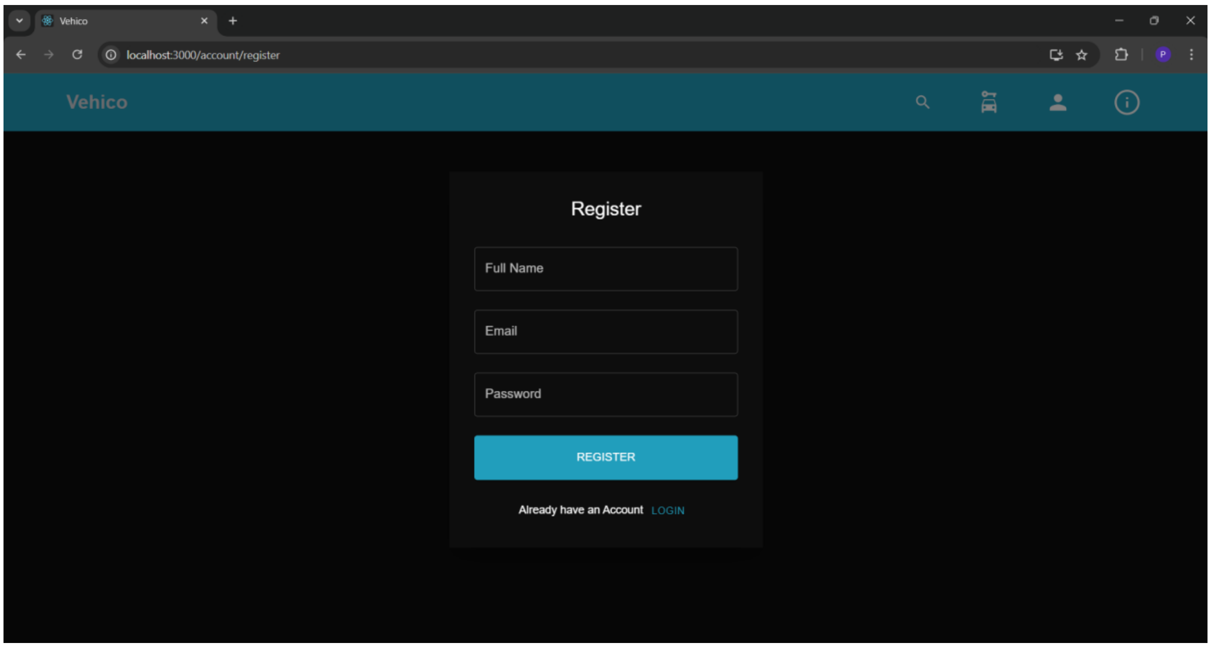
Audi A8 ₹130000
Flagship sedan offering high-tech features and unmatched ride comfort.

Lexus LS ₹120000
A full-size luxury sedan with cutting-edge design and advanced hybrid technology.

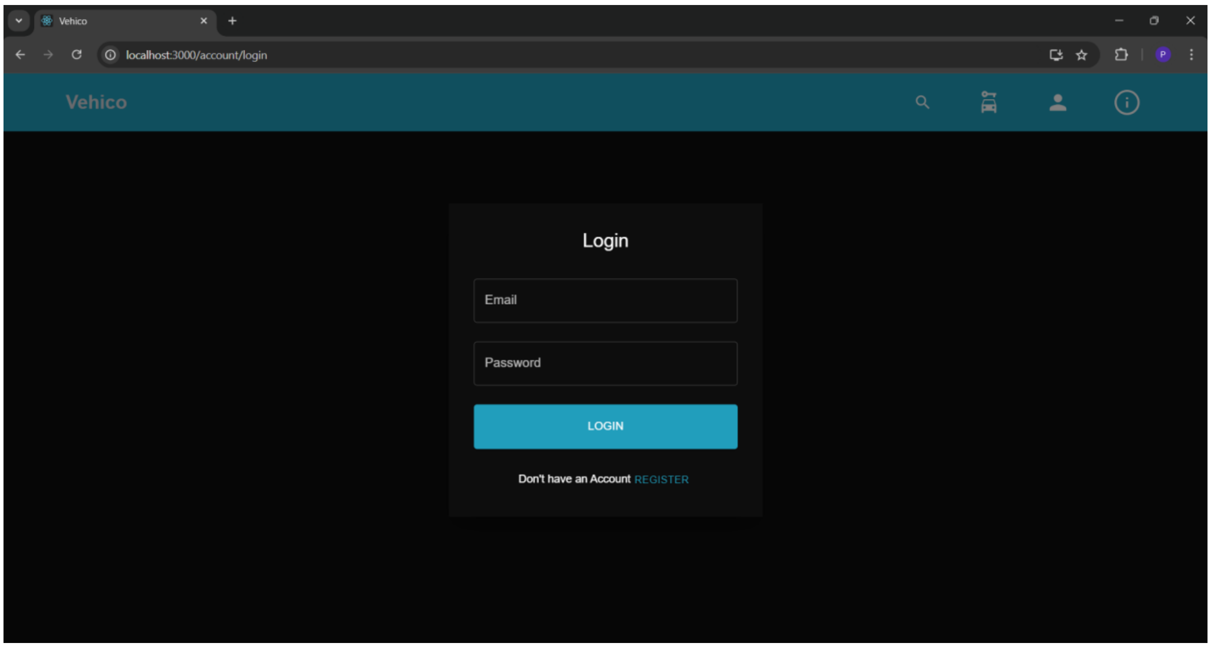
Jaguar XJ ₹110000
A performance-focused luxury sedan with innovative design and advanced tech features.

Maserati Quattroporte ₹105000
A luxury sedan that exudes elegance with Italian styling and powerful performance.

2. Register:

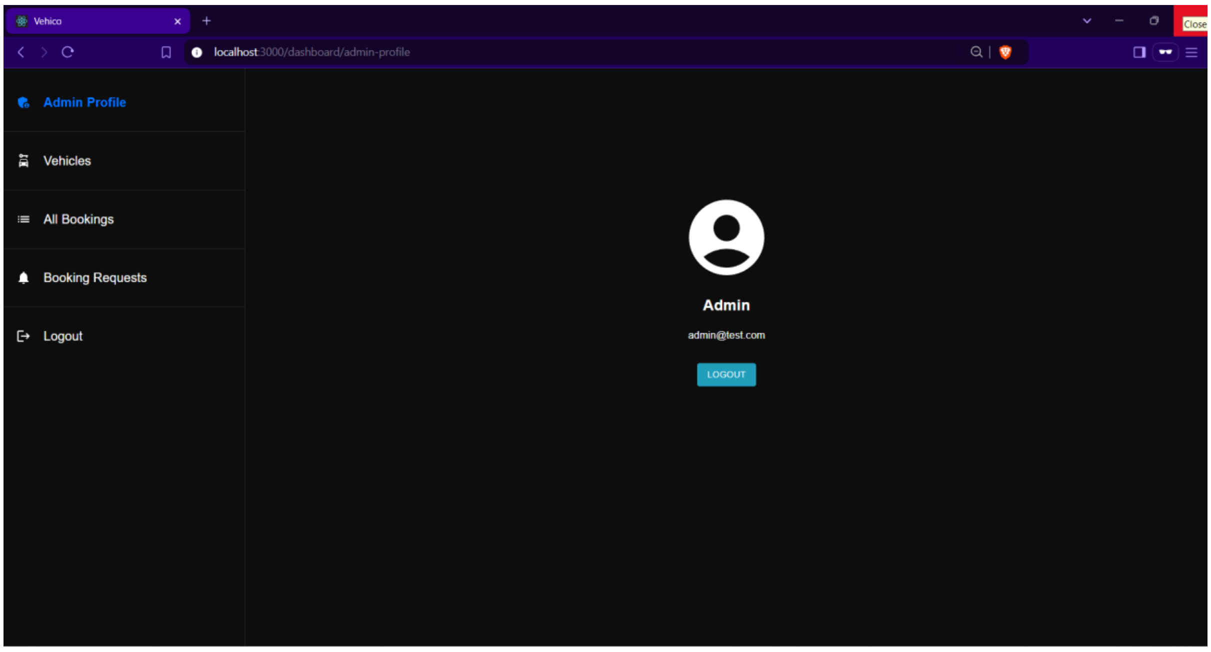

Vehico
localhost:3000/account/register
Vehico
Register
Full Name
Email
Password
REGISTER
Already have an Account LOGIN

3. Login:

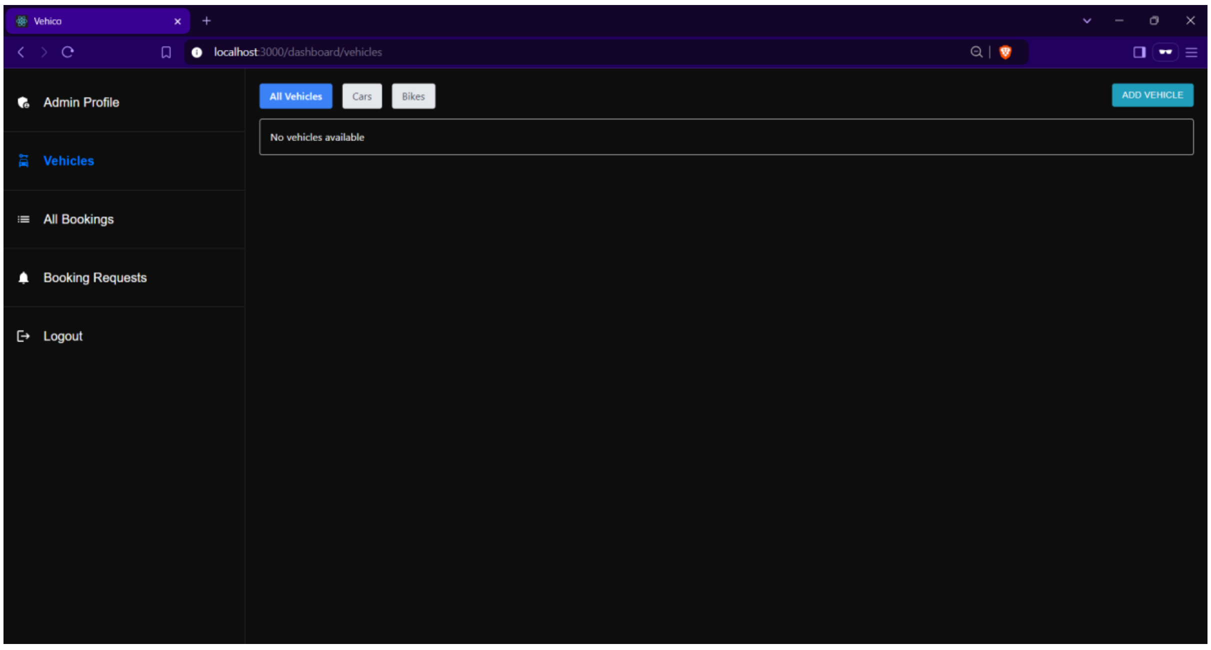


4.Admin Dashboard

Admin Profile:

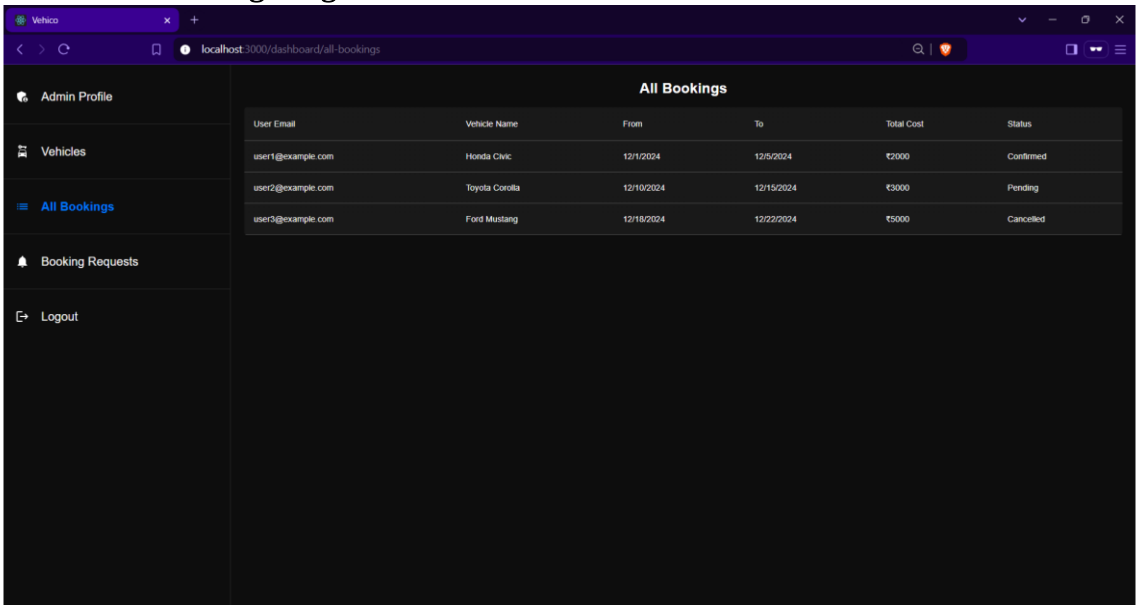


Admin Vehicle Page:



No vehicles available

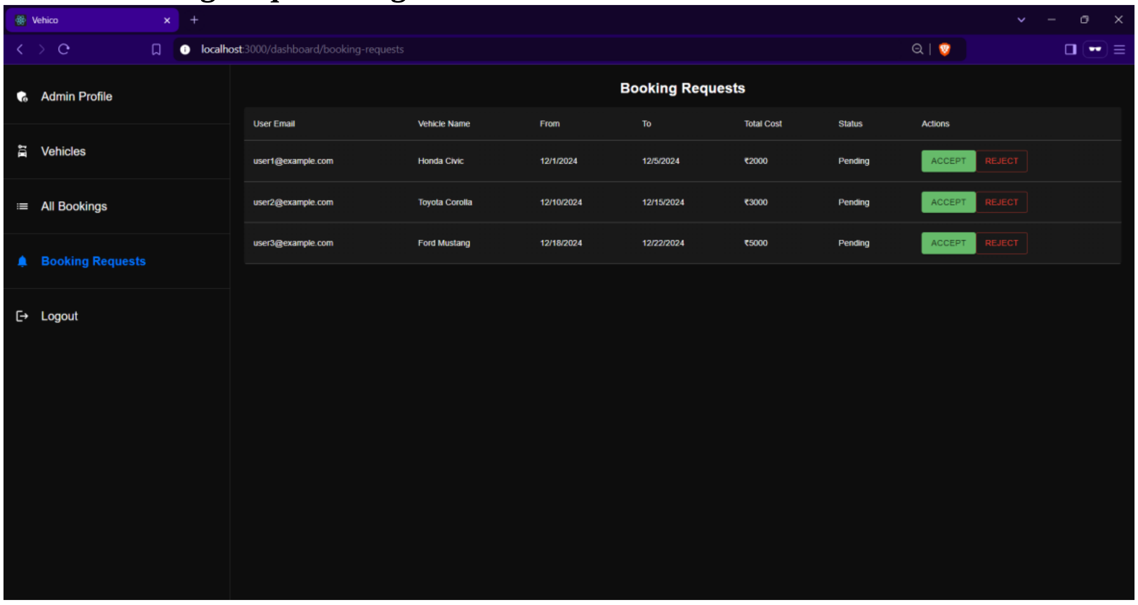
Admin All Bookings Page :



All Bookings

User Email	Vehicle Name	From	To	Total Cost	Status
user1@example.com	Honda Civic	12/1/2024	12/5/2024	₹2000	Confirmed
user2@example.com	Toyota Corolla	12/10/2024	12/15/2024	₹3000	Pending
user3@example.com	Ford Mustang	12/18/2024	12/22/2024	₹5000	Cancelled

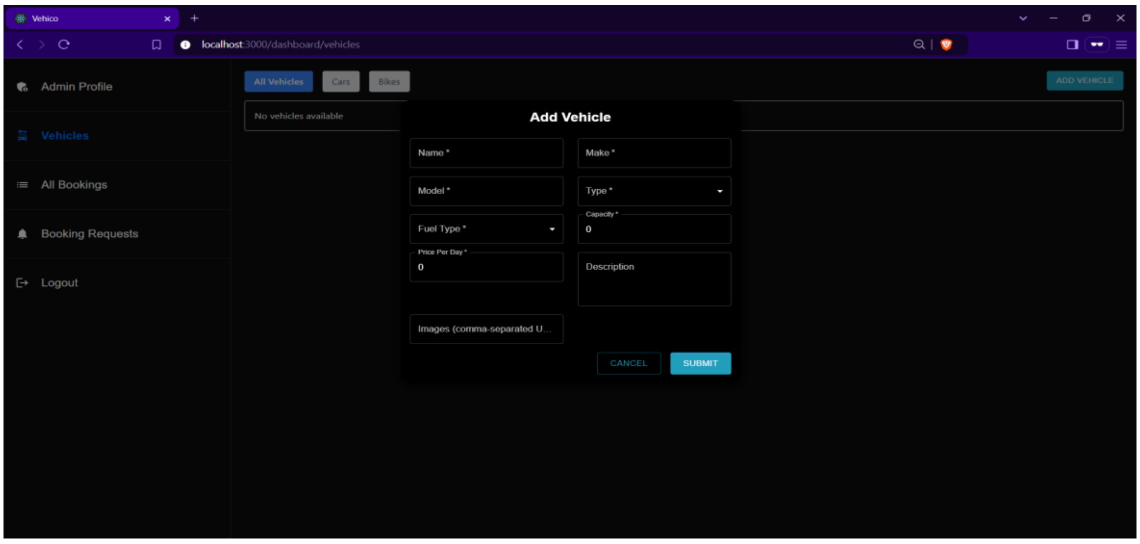
Admin Booking Requests Page:



Booking Requests

User Email	Vehicle Name	From	To	Total Cost	Status	Actions
user1@example.com	Honda Civic	12/1/2024	12/5/2024	₹2000	Pending	ACCEPT REJECT
user2@example.com	Toyota Corolla	12/10/2024	12/15/2024	₹3000	Pending	ACCEPT REJECT
user3@example.com	Ford Mustang	12/18/2024	12/22/2024	₹5000	Pending	ACCEPT REJECT

Admin Add Vehicle Modal:

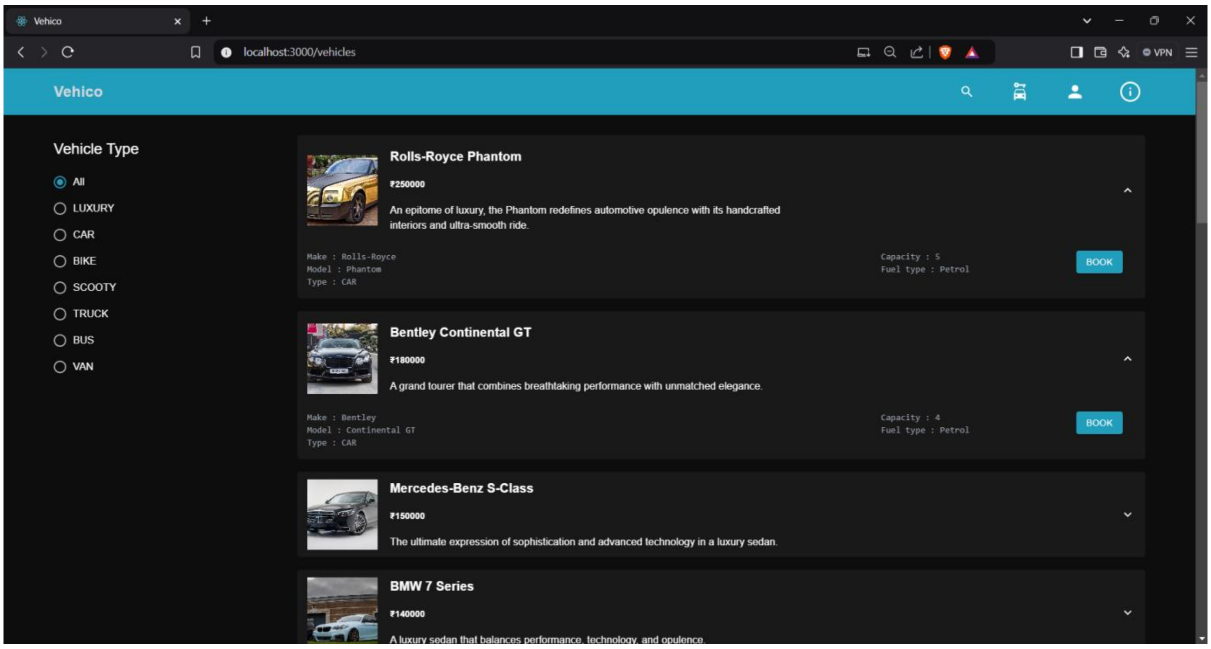


No vehicles available

Add Vehicle

Name *	Make *
Model *	Type *
Fuel Type *	Capacity * 0
Price Per Day * 0	Description
Images (comma-separated U...	

5. Customer Vehicle Listings


Vehico

localhost:3000/vehicles

Vehico

Vehicle Type

- [✓ All]
- LUXURY
- CAR
- BIKE
- SCOOTY
- TRUCK
- BUS
- VAN

Rolls-Royce Phantom

₹250000

An epitome of luxury, the Phantom redefines automotive opulence with its handcrafted interiors and ultra-smooth ride.

```
Make : Rolls-Royce                Capacity : 5
Model : Phantom                   Fuel type : Petrol
Type : CAR
```

BOOK

Bentley Continental GT

₹180000

A grand tourer that combines breathtaking performance with unmatched elegance.

```
Make : Bentley                    Capacity : 4
Model : Continental GT            Fuel type : Petrol
Type : CAR
```

BOOK

Mercedes-Benz S-Class

₹150000

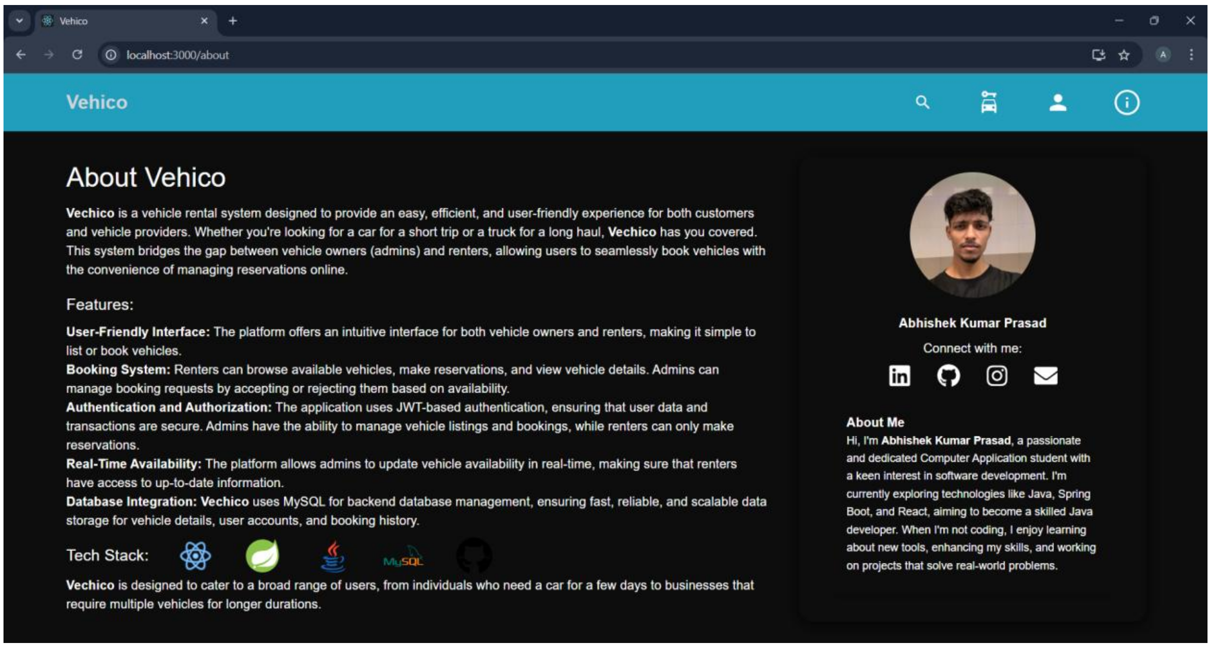
The ultimate expression of sophistication and advanced technology in a luxury sedan.

BMW 7 Series

₹140000

A luxury sedan that balances performance, technology, and opulence.

6. About Page:


Vehico

localhost:3000/about

Vehico

About Vehico

Vechico is a vehicle rental system designed to provide an easy, efficient, and user-friendly experience for both customers and vehicle providers. Whether you're looking for a car for a short trip or a truck for a long haul, **Vechico** has you covered. This system bridges the gap between vehicle owners (admins) and renters, allowing users to seamlessly book vehicles with the convenience of managing reservations online.

Features:

User-Friendly Interface: The platform offers an intuitive interface for both vehicle owners and renters, making it simple to list or book vehicles.
Booking System: Renters can browse available vehicles, make reservations, and view vehicle details. Admins can manage booking requests by accepting or rejecting them based on availability.
Authentication and Authorization: The application uses JWT-based authentication, ensuring that user data and transactions are secure. Admins have the ability to manage vehicle listings and bookings, while renters can only make reservations.
Real-Time Availability: The platform allows admins to update vehicle availability in real-time, making sure that renters have access to up-to-date information.
Database Integration: Vechico uses MySQL for backend database management, ensuring fast, reliable, and scalable data storage for vehicle details, user accounts, and booking history.

Tech Stack:

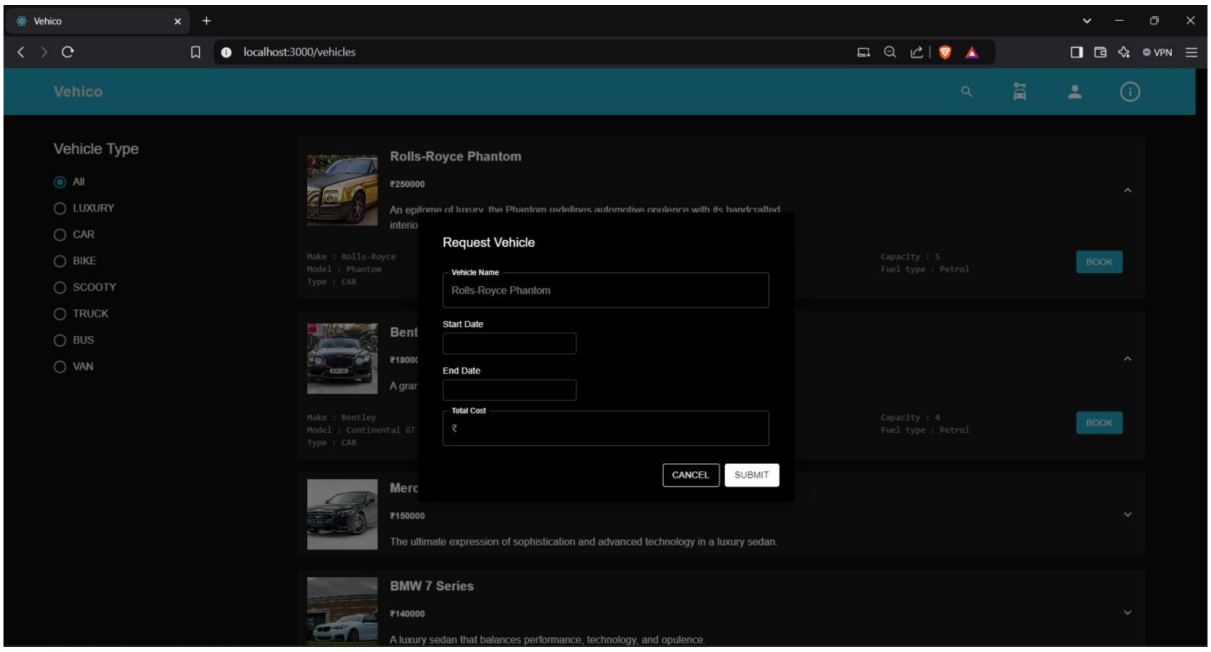
Vechico is designed to cater to a broad range of users, from individuals who need a car for a few days to businesses that require multiple vehicles for longer durations.

Abhishek Kumar Prasad

Connect with me:

About Me
Hi, I'm **Abhishek Kumar Prasad**, a passionate and dedicated Computer Application student with a keen interest in software development. I'm currently exploring technologies like Java, Spring Boot, and React, aiming to become a skilled Java developer. When I'm not coding, I enjoy learning about new tools, enhancing my skills, and working on projects that solve real-world problems.

7. Booking Request Modal (Customer):



Rolls-Royce Phantom

₹250000

An epitome of luxury, the Phantom redefines automotive opulence with its handcrafted interio

Make : Rolls-Royce
Model : Phantom
Type : CAR

Capacity : 5
Fuel type : Petrol

Request Vehicle

Vehicle Name: Rolls-Royce Phantom

Start Date:

End Date:

Total Cost: ₹

Bent

₹18000

A grar

Make : Bentley
Model : Continental GT
Type : CAR

Capacity : 4
Fuel type : Petrol

Merc

₹150000

The ultimate expression of sophistication and advanced technology in a luxury sedan.

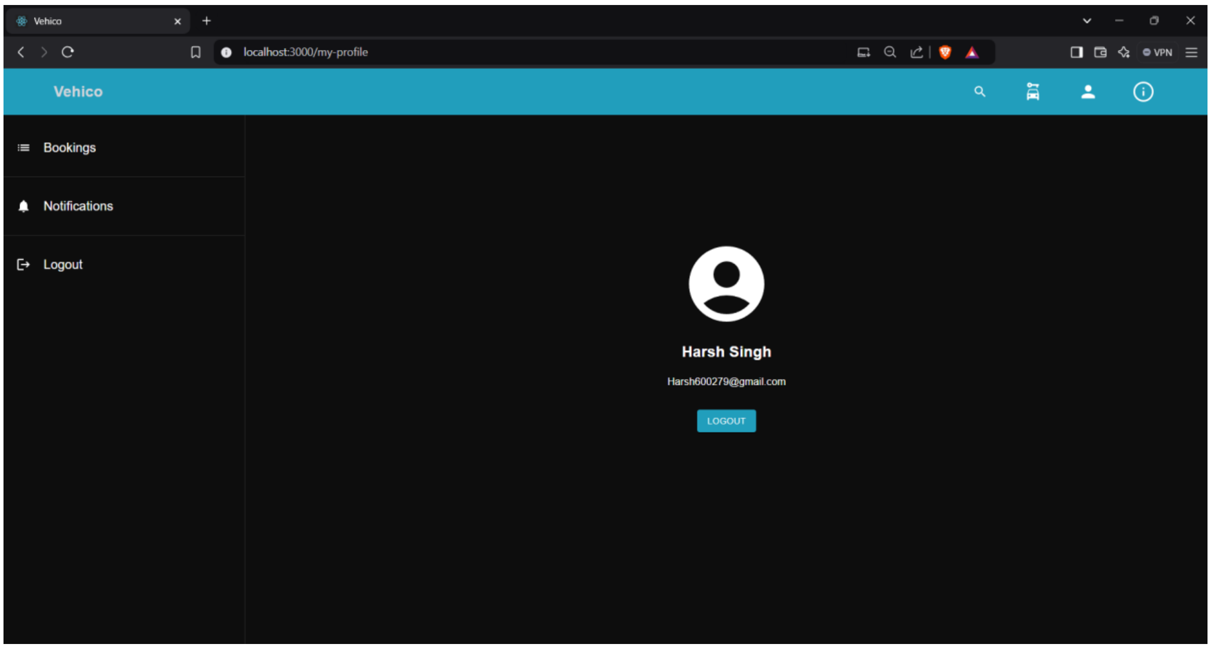
BMW 7 Series

₹140000

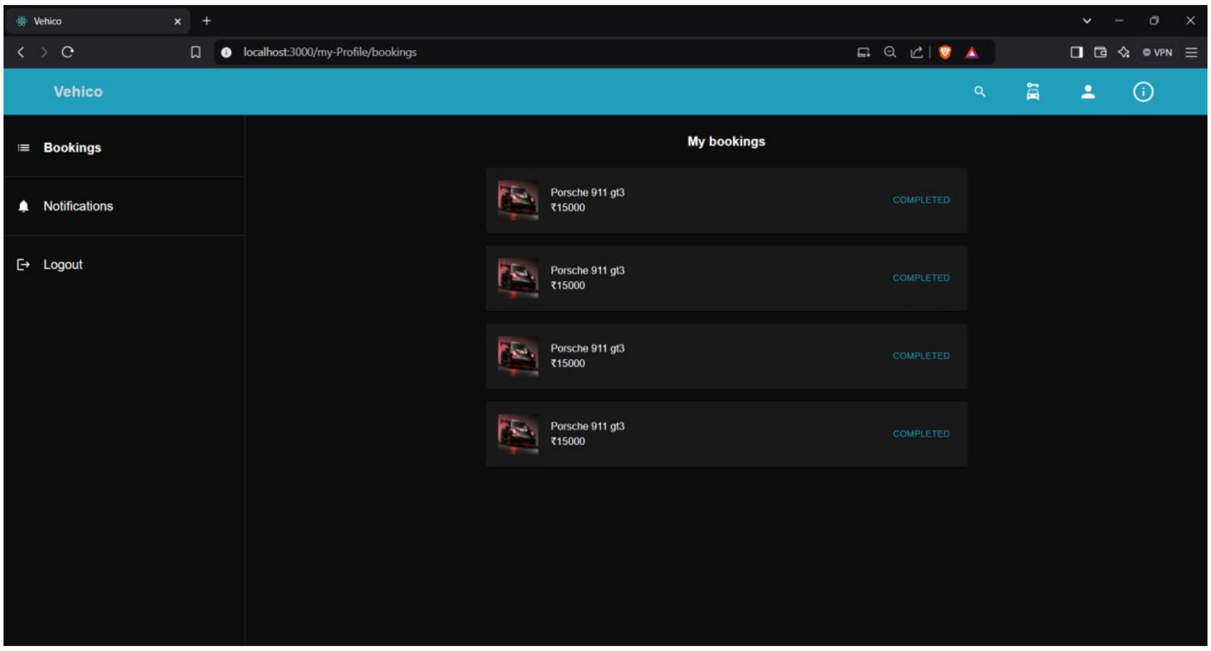
A luxury sedan that balances performance, technology, and opulence.

8. User Profile:

User Profile Page:



Customer Bookings Page:



My bookings

Porsche 911 gt3
₹15000
COMPLETED

Porsche 911 gt3
₹15000
COMPLETED

Porsche 911 gt3
₹15000
COMPLETED

Porsche 911 gt3
₹15000
COMPLETED

CHAPTER 12: REFERENCES

12.1 Books, Journals, and Articles

1. **"Spring in Action"** by Craig Walls
   A comprehensive guide to learning Spring Framework, covering core concepts, features, and the Spring ecosystem.
2. **"Java Persistence with Hibernate"** by Christian Bauer and Gavin King
   A complete resource for understanding Hibernate and JPA, which was instrumental in setting up the persistence layer for **Vechico**.
3. **"Clean Code: A Handbook of Agile Software Craftsmanship"** by Robert C. Martin
   A must-read for writing maintainable and clean code, with valuable practices applied throughout this project.
4. **"Design Patterns: Elements of Reusable Object-Oriented Software"** by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides
   Provides essential knowledge of design patterns, which helped in structuring the application architecture efficiently.

12.2 Online Tutorials and Resources

1. **Spring Framework Documentation**
   https://spring.io/docs
   Official documentation for Spring Boot, Spring Data, and other Spring modules, which helped throughout the project development.
2. **Baeldung - Spring Tutorials**
   https://www.baeldung.com/
   A rich source of tutorials covering various Spring-related topics, particularly useful for learning Spring Boot and security implementations.
3. **Java Brains YouTube Channel**
   https://www.youtube.com/user/koushks
   A valuable resource for video tutorials on Java and Spring, offering in-depth explanations of various concepts.
4. **React Official Documentation**
   https://reactjs.org/docs/
   The official documentation for React, providing detailed instructions on creating interactive UIs.

13.3 Frameworks, Libraries, and Tools

1. **Spring Boot**
   A framework used for creating stand-alone, production-grade Spring-based applications with minimal configuration.
2. **Spring Security**
   Used for implementing authentication and authorization in the application, with JWT for secure user sessions.
3. **Hibernate & JPA**
   Hibernate is used as the ORM tool to interact with MySQL, providing a simple way to manage the persistence layer.
4. **React.js**
   A JavaScript library for building user interfaces, specifically for managing the frontend of the **Vechico** system.
5. **MySQL**
   The relational database management system used for storing the application's data, including user details, vehicles, and bookings.
6. **JWT (JSON Web Tokens)**
   Used for securing the REST APIs by providing token-based authentication.
7. **Maven**
   A build automation tool used to manage project dependencies and build processes for the **Vechico** project.
8. **Postman**
   A tool used for testing and interacting with REST APIs, which was essential for debugging and verifying the backend functionality.
9. **VS Code & Eclipse**
   IDEs used for writing and debugging the code for the backend and frontend components of the system.