CAR STOCK MANAGEMENT SYSTEM

A DBMS PROJECT REPORT

Submitted by

HARSHVARDHAN MHASKE [RA2311003010039]
RITHVIK JASWAL [RA2311003010040]
PRAKRUT DAVE [RA2311003010056]

Under the Guidance of

DR. R. MANGALAGOWRI
Associate Professor
Department Of Computing Technologies

in partial fulfillment of the requirementsfor the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES, COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY

KATTANKULATHUR- 603 203

Department of Computational Intelligence
SRM Institute of Science & Technology
Own Work* Declaration Form

This sheet must be filled in (each box ticked to show that the condition has been met). It must be signed and dated along with your student registration number and included with all assignments you submit – work will not be marked unless this is done.

To be completed by the student for all assessments

Degree/ Course : **Btech Computer Science Engineering**

Student Name : **Harshvardhan Mhaske , Rithvik Jaswal , Prakrut Dave**

Registration Number : **RA2311003010039 , RA2311003010040 , RA2311003010056**

Title of Work : **CAR STOCK MANAGEMENT SYSTEM**

I / We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this assessment is my / our own except where indicated, and that I / We have met the following conditions:

- Clearly referenced / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that 21CSC205P – Database Management System Project report titled "**CAR STOCK MANAGEMENT SYSTEM** " is the bonafide work of "**HARSHVARDHAN MHASKE [RA2311003010039], RITHVIK JASWAL [RA23113010040] , PRAKRUT DAVE [RA23113010056]"** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. R. MANGALAGOWRI

ASSISTANT PROFESSOR
DEPARTMENT OF COMUTION TECHNOLOGIES

SIGNATURE

Dr. G. NIRANJANA

PROFESSOR &HEAD
DEPARTMENT OF COMPUTING TECHNOLOGIES

ACKNOWLEDGEMENTS

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to **Dr. Leenus Jesu Martin M,** Dean-CET, SRM Institute of Science and Technology, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor and Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We encompass our sincere thanks to, **Dr. M. Pushpalatha**, Professor and Associate Chairperson, School of Computing and **Dr. C. Lakshmi,** Professor and Associate Chairperson, School of Computing, SRM Institute of Science and Technology, for their invaluable support.

We are incredibly grateful to our Head of the Department, **Dr. G. Niranjana**, Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our DBMS Coordinators, **Dr. Muthu Kumaran A M J** and **Dr. Jayapradha J** Department of Computing Technologies, and our Audit Professor **Dr. Malathy C,** Department of Networking And Communications, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisors, **Dr. G. Ramya**, Department of Computing Technologies, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our faculty, **Dr. R. Mangalagowri,** Department of Computing Technologies, S.R.M Institute of Science and Technology, for providing us with an opportunity to pursue our project under her mentorship. She provided us with the freedom and support to explore the research topics of our interest. Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank all the staff and students of Computing Technologies, School of Computing, S.R.M Institute of Science and Technology, for their help during our project. Finally, we would like to thank our parents, family members, and friends for their unconditional love, constant support and encouragement.

Harshvardhan Mhaske RA2311003010039

Rithvik jaswal RA2311003010040

Prakrut Dave RA2311003010056

ABSTRACT

The Car Rental Management System is a fully integrated, database-driven application designed to manage and streamline the key operations of a car rental agency. This system automates various tasks such as maintaining vehicle inventory, handling customer bookings, managing user registrations, and overseeing administrative functions, thereby reducing manual workload and increasing operational efficiency. Built using MySQL, the system employs a normalized relational database structure consisting of multiple interconnected tables including admin, tblvehicles, tblusers, tblbooking, and others that ensure minimal redundancy and consistent data relationships through the use of primary and foreign keys. The system supports essential database operations like data insertion, updates, deletions, and complex querying through SQL, enabling real-time tracking of car availability, customer details, and booking history. Security is enforced through encrypted admin credentials and restricted access levels. Each module in the database serves a distinct function—for example, tblvehicles stores car details such as model, brand, fuel type, and rental price; tblbooking records customer rental activity with booking numbers and date ranges; while tblusers keeps track of registered users and their interactions with the system. The design reflects a clear understanding of core DBMS concepts such as schema design, referential integrity, data normalization, and CRUD operations. Furthermore, the system can be scaled to include advanced features like payment gateways, automated notifications, real-time availability dashboards, and a web-based front end. Overall, this project serves as a strong foundation for implementing a reliable, efficient, and secure car rental management solution suitable for real-world deployment in small to mid-sized rental enterprises.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

Abbreviation	Full Form
API	Application Programming Interface
DBMS	Database Management System
DNS	Domain Name System
DoS	Denial of Service
IDS	Intrusion Detection System
IP	Internet Protocol
ICMP	Internet Control Message Protocol
LAN	Local Area Network
MySQL	My Structured Query Language (Database)
NIDS	Network Intrusion Detection System
OS	Operating System
SQL	Structured Query Language
SYN	Synchronize (TCP Flag)
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
GUI	Graphical User Interface

Abbreviation	Full Form
MAC	**Media Access Control**
SIEM	**Security Information and Event Management**
REST	**Representational State Transfer**
JSON	**JavaScript Object Notation**
RAM	**Random Access Memory**
CPU	**Central Processing Unit**
NIC	**Network Interface Card**
HIDS	**Host Intrusion Detection System**
XMAS	**Christmas Tree Scan (TCP scan type)**
NULL	**Null Scan (TCP scan type)**
FIN	**Finish Scan (TCP scan type)**
RFC	**Request for Comments**

CHAPTER 1: INTRODUCTION

1.1 Introduction

In today's fast-paced and technology-driven environment, traditional methods of managing car rental services are no longer sufficient to meet the growing expectations of efficiency, accuracy, and customer satisfaction. The Car Rental Management System (CRMS) has been developed as a robust and fully integrated database application to address these challenges by automating and streamlining the day-to-day operations of a car rental agency.

This system is built on a relational database using MySQL, with a schema that adheres to principles of data normalization and referential integrity. It is designed to support core functionalities such as vehicle inventory management, customer booking processes, user registration and authentication, and administrative monitoring. The database comprises a collection of interrelated tables including admin, tblvehicles, tblusers, and tblbooking. Each table is structured to perform specific roles. The tblvehicles table stores detailed information about the available vehicles including their brand, model, fuel type, price per day, and status. The tblusers table handles user account data and their interactions with the system. The tblbooking table tracks reservation details such as customer ID, vehicle ID, booking dates, and status updates. The admin table manages access control and encrypted login credentials for administrative users.

The system supports all essential CRUD operations and allows for complex SQL queries that enable real-time monitoring of vehicle availability, booking history, and customer activity. These capabilities significantly reduce the reliance on manual recordkeeping, improve accuracy, and allow for scalable business operations. Security is a core aspect of the system, with features such as password encryption, restricted access levels based on user roles, and controlled access to sensitive data. Additionally, the modular nature of the database design makes it extensible for future enhancements. Potential upgrades may include integration with online payment gateways, email or SMS notifications for booking confirmations, real-time availability dashboards, and a web-based or mobile front-end for customer access.

This project not only demonstrates technical proficiency in database management system concepts such as schema design, foreign key relationships, indexing, and query optimization but also presents a practical solution for addressing the real-world needs of car rental agencies. It serves as a solid foundation for further development into a fully operational enterprise-level application.

1.2 Objectives

The primary objective of the Car Rental Management System is to design and develop a fully functional, secure, and scalable platform that automates the key operations of a car rental service. Traditional paper-based or spreadsheet-driven methods are often error-prone, time-consuming, and inefficient in handling increasing customer demands and large volumes of data. This project aims to eliminate such limitations by leveraging a structured relational database model using MySQL, enabling centralized data management, real-time updates, and enhanced operational control. By introducing system automation and enforcing role-based access, the solution ensures smooth interactions between customers, staff, and administrators. The system is not only tailored to fulfill current business requirements but also lays the groundwork for advanced future features such as online payments, mobile support, and data analytics. The following objectives outline the specific goals that guided the development of the system:

- To develop a comprehensive system that automates the end-to-end workflow of a car rental business, including car listings, customer bookings, and user account handling.
- To design a relational database using MySQL with a normalized schema that ensures efficient data organization, avoids redundancy, and maintains referential integrity through the use of primary and foreign keys.
- To enable real-time tracking of car availability, booking schedules, and user activity by implementing optimized SQL queries and efficient database transactions.
- To establish secure login mechanisms for administrators and users using encrypted credentials and session-based authentication to prevent unauthorized access.
- To provide full CRUD (Create, Read, Update, Delete) functionality across all major system modules such as vehicles, users, bookings, and admin settings for dynamic data management.
- To reduce manual workload, improve accuracy in data handling, and decrease the processing time for bookings and vehicle status updates.
- To develop a user-friendly and intuitive interface that caters to both technical and non-technical users, ensuring smooth navigation and interaction with the system.
- To implement role-based access controls that clearly differentiate between administrative privileges and standard user capabilities, ensuring a secure and organized flow of operations.
- To design the system architecture in a modular and extensible manner so it can accommodate future features such as payment gateway integration, email or SMS notifications, business analytics, and web/mobile application integration.
- To provide a reliable and practical solution for small to mid-sized car rental enterprises, with the potential for customization and deployment in real-world commercial environments.

1.3 System Requirements

Hardware Requirements

- Processor: Intel Core i3 or higher
- RAM: Minimum 4 GB (8 GB recommended)
- Storage: At least 500 MB free disk space
- Display: 1024×768 resolution or higher
- Network: Required for server access or multi-user operation

Software Requirements

- Operating System: Windows 10/11, Linux, or macOS
- Local Server Stack: XAMPP (Apache, MySQL, PHP)
- Database: MySQL Server with phpMyAdmin
- Code Editor: Visual Studio Code or equivalent
- Web Browser: Chrome, Firefox, or Edge (latest version)
- Optional Tools: MySQL Workbench for database modeling and ER diagrams

CHAPTER 2: MODULES

2.1 Module 1: Vehicle Management

The Vehicle Management Module is responsible for managing the core asset of the car rental business: its fleet of vehicles. This module provides the functionalities to add, update, delete, and view vehicle information, ensuring that the database accurately reflects the current inventory.

Functions:

- **Add New Vehicles:**
    - Allows administrators to add new vehicles to the system.
    - Captures essential vehicle details, including:
        - Vehicle Title/Name
        - Brand (references tbl_brands)
        - Description/Overview
        - Price Per Day
        - Fuel Type
        - Model Year
        - Seating Capacity
        - Vehicle Images (multiple images - Vimage1 to Vimage5)
        - Features (e.g., Air Conditioner, Power Door Locks - boolean flags)
    - Inserts data into the tbl_vehicles table.
    - Validates data inputs to ensure accuracy and completeness (e.g., ensuring the brand exists, price is a positive number, year is a valid year).
- **Update Vehicle Information:**
    - Enables administrators to modify existing vehicle information.
    - Provides a user interface to edit vehicle details.
    - Updates the corresponding record in the tbl_vehicles table.
    - Handles updates to any of the vehicle attributes mentioned above.
    - Ensures that updates are applied correctly and that data integrity is maintained.
- **Delete Vehicles:**
    - Allows administrators to remove vehicles from the system.

- Provides a mechanism to select the vehicle to be deleted.
- Deletes the corresponding record from the tbl_vehicles table.
- May include a confirmation step to prevent accidental deletions.
- Considers the implications of deleting a vehicle that may be associated with existing or past bookings in the tblbooking table (e.g., may need to handle these bookings or prevent deletion if there are active rentals).

- **View Available Vehicles:**
  - Provides functionality to display a list of vehicles that are currently available for rent.
  - This may involve filtering vehicles based on their availability status (which may be derived from the tblbooking table).
  - Presents vehicle information in a user-friendly format, including relevant details like brand, model, price, and availability.
  - May support sorting and filtering of vehicles based on various criteria (e.g., price, brand, vehicle type).
- **Manage Vehicle Features:**
  - Allows administrators to manage the features offered by each vehicle.
  - This involves setting boolean flags for features like Air Conditioner, Power Door Locks, etc., in the tbl_vehicles table.
  - Provides a user interface to easily enable or disable these features for each vehicle.

 Interactions with Database:

- The module primarily interacts with the tbl_vehicles table.
- It also reads from the tbl_brands table to get the list of available brands when adding or updating vehicles.
- Potentially interacts with the tblbooking table to determine vehicle availability and to handle deletion of vehicles with existing bookings.

 Business Rules:

- Requires that a vehicle must be associated with a valid brand from the tbl_brands table.
- Ensures that prices and other numerical values are within acceptable ranges.
- May implement rules to prevent the deletion of vehicles that are currently rented.
- Maintains data consistency between tbl_vehicles and other related tables.

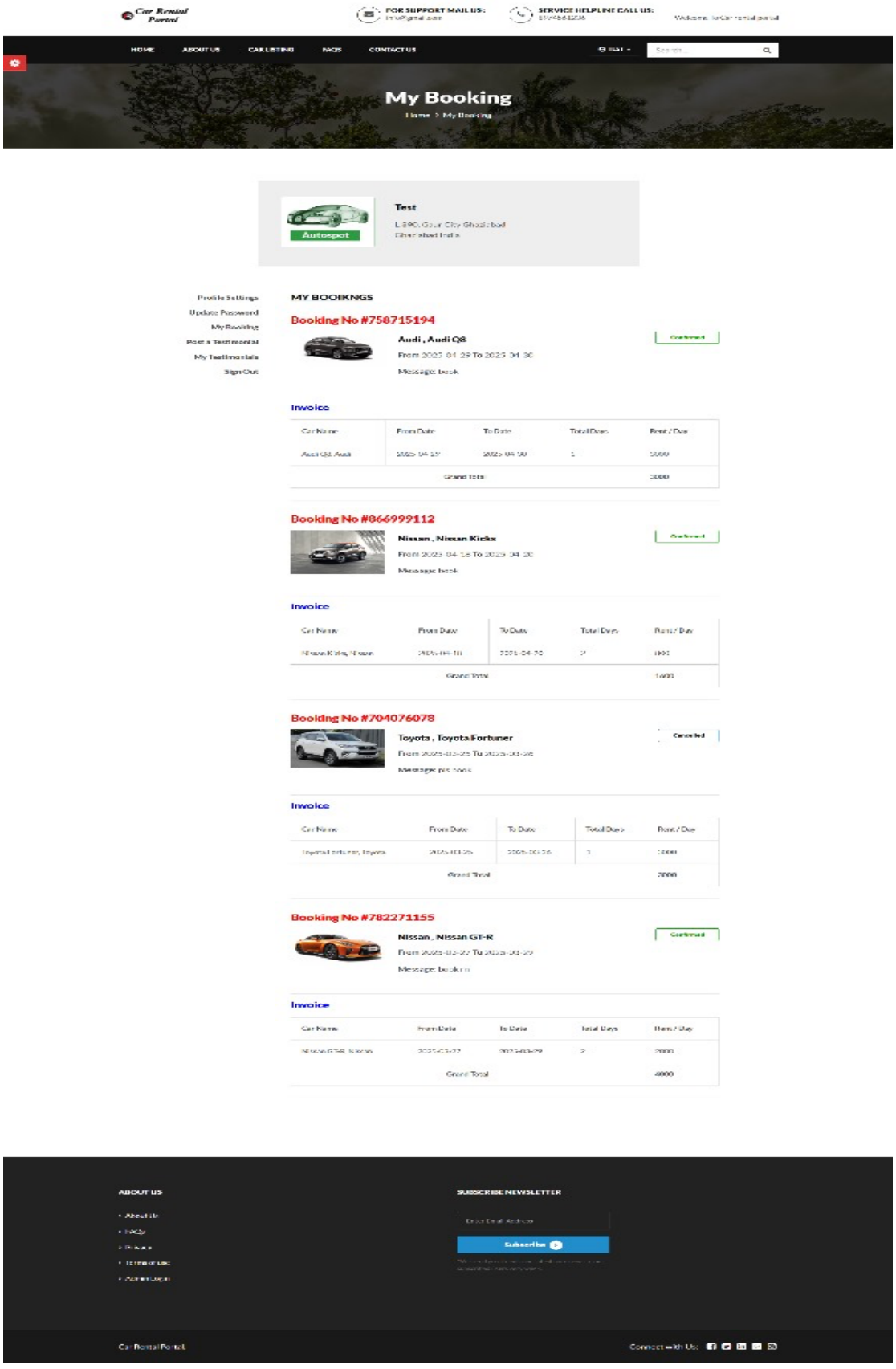

Fig 2.1 vehicle Management

2.2 Module 2: Brand Management

The Brand Management Module is responsible for managing the list of car brands available in the car rental system. This module provides the functionality to add, update, delete, and list car brands, ensuring that the system's database accurately reflects the valid car brands for vehicle inventory management.

 Functions:

- **Add New Car Brands**:
    - Allows administrators to add new car brands to the system.
    - Captures brand names.
    - Inserts new brands into the tbl_brands table.
    - Validates data input to ensure brand names are unique and meet any defined formatting requirements.
- **Update Existing Brand Information**:
    - Enables administrators to modify the names of existing car brands.
    - Retrieves the current brand information.
    - Updates the corresponding record in the tbl_brands table.
    - Ensures that updated brand names are unique.
- **Delete Brands from the System**:
    - Allows administrators to remove car brands from the system.
    - Checks if any vehicles are associated with the brand in the tbl_vehicles table.
    - If no vehicles are associated, deletes the brand from the tbl_brands table.
    - If vehicles are associated, prevents deletion and displays an error message to maintain data integrity.
- **List All Available Brands**:
    - Provides functionality to retrieve a list of all car brands stored in the tbl_brands table.
    - Presents the list of brands in a user-friendly format.

 Interactions with Database:

- The module primarily interacts with the tbl_brands table.
- It may also interact with the tbl_vehicles table to check for brand associations before deleting a brand.

 Business Rules:

- Brand names must be unique to avoid duplication in the system.
- Deletion of a brand should be restricted if there are vehicles associated with that brand in the tbl_vehicles table. This rule prevents data inconsistency and ensures that vehicle records always reference valid car brands.

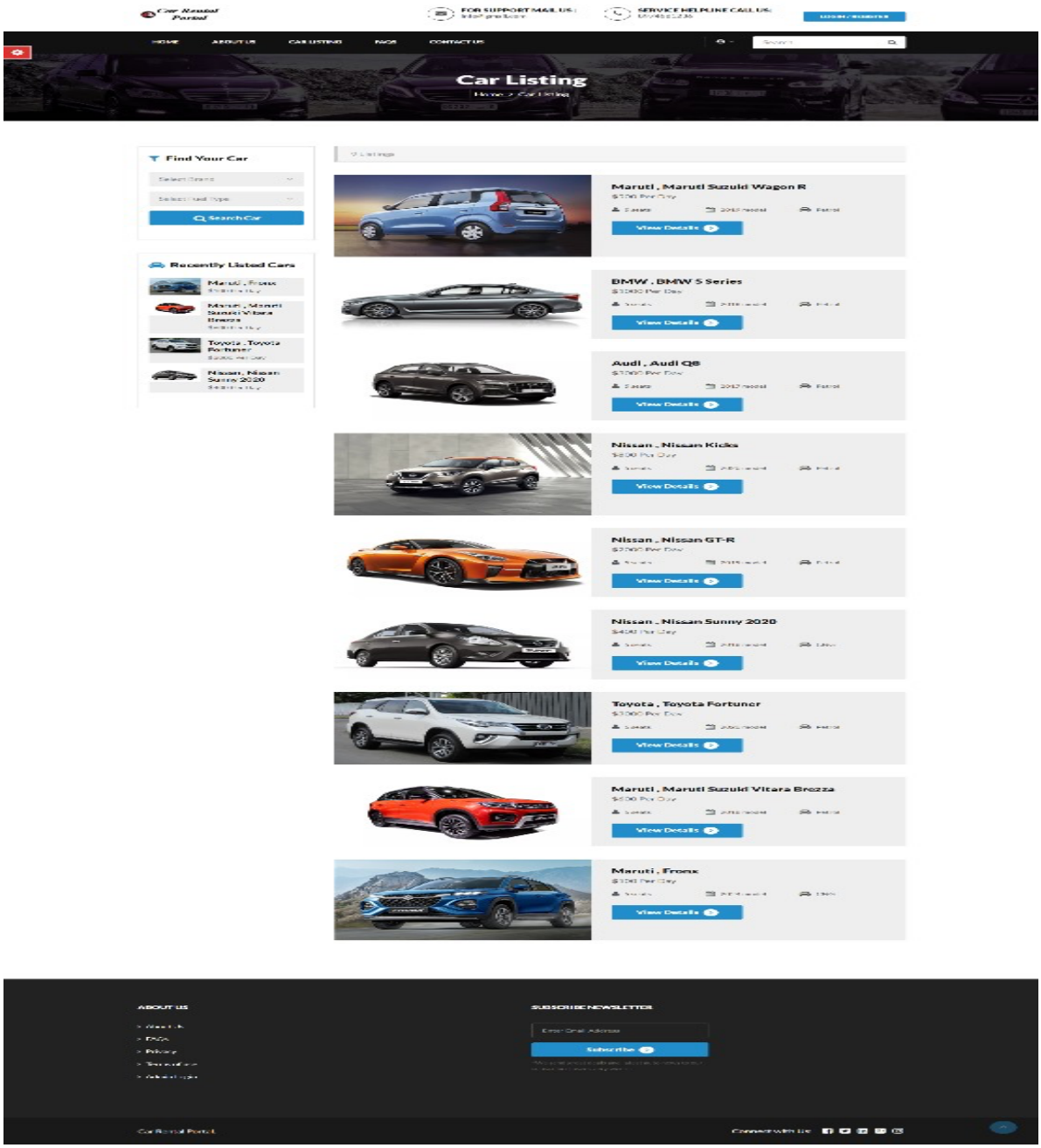

Fig 2.2 Brand management

2.3 Module 3: User Management

The User Management Module is responsible for managing customer information within the car rental system. This includes handling user registration, authentication, profile management, and password management. The module ensures secure and efficient management of user data, which is crucial for the overall functionality of the system.

 Functions:

- **User Registration and Account Creation:**
    - Allows new users to register and create accounts in the system.
    - Captures user details such as full name, email address, contact number, date of birth, address, city, and country.
    - Stores user credentials (email and password) securely in the tbl_users table.
    - Validates user input to ensure data accuracy and completeness.
    - May send a confirmation email to the user upon successful registration.
- **User Login and Authentication**:
    - Enables registered users to log in to the system using their credentials (email and password).
    - Authenticates users against the data stored in the tbl_users table.
    - Manages user sessions to maintain login status.
    - Implements security measures to protect against unauthorized access.
- **User Profile Management (View, Edit**):
    - Allows users to view and edit their profile information.
    - Retrieves user details from the tbl_users table for display.
    - Enables users to update their personal information, such as address, contact number, etc.
    - Ensures that updated information is saved correctly in the tbl_users table.
- **Password Management (Reset, Change):**
    - Provides functionality for users to manage their passwords.
    - Allows users to change their existing passwords.
    - Implements password reset functionality for users who have forgotten their passwords.
    - Ensures that new passwords meet the defined complexity requirements.

- Handles password encryption and storage securely.

Interactions with Database:

- The module primarily interacts with the tbl_users table.

Business Rules:

- Ensures that email addresses are unique to prevent duplicate accounts.
- Enforces password complexity requirements (e.g., minimum length, special characters) to enhance security.
- Handles user authentication and authorization to control access to different parts of the system.

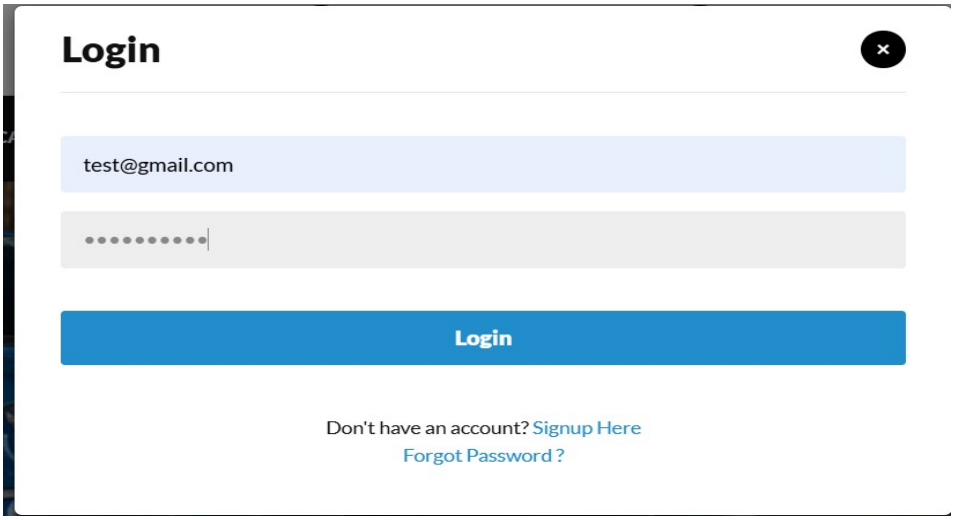

Fig 2.3.1 User Login

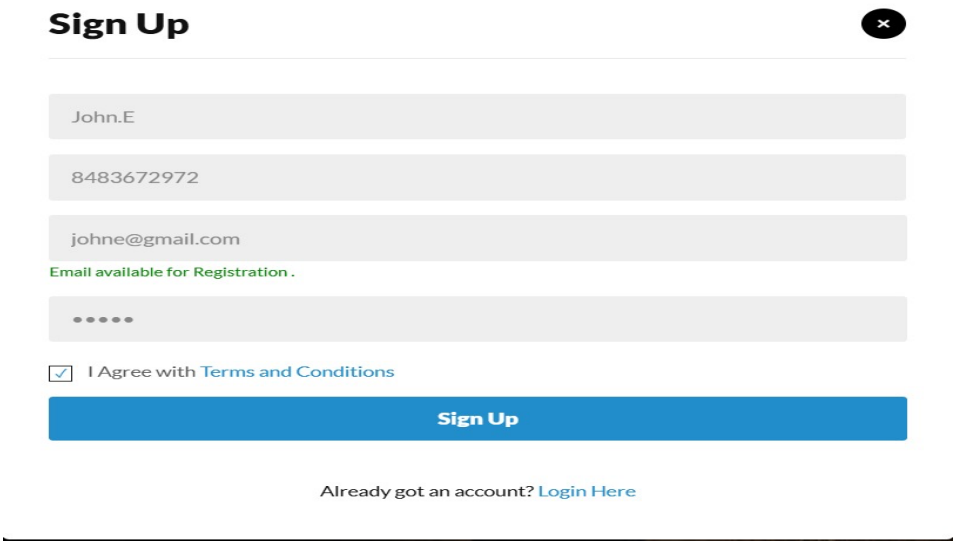

Fig 2.3.2 User Sign up

2.4 Module 4: Booking Management

The Booking Management Module handles vehicle reservations. Key functions include:

Managing bookings:

- This encompasses the entire lifecycle of a booking, from the initial customer request to the final confirmation.
- It involves the process of creating new booking records in the database, ensuring all necessary details such as customer information, vehicle selection, and rental dates are captured accurately and efficiently.
- A crucial part of this function is validating the booking information to prevent errors and inconsistencies, ensuring that the requested vehicle is available for the specified period and that all required data fields are complete and correctly formatted.

Viewing details:

- This function provides access to comprehensive booking information for both users and administrators.
- It allows them to review the specifics of a booking, including vehicle details (make, model, features), customer information (name, contact details), rental dates (start and end dates), and booking status (pending, confirmed, completed, cancelled).
- This detailed view provides a complete overview of the reservation, facilitating communication and ensuring that all parties have access to the necessary information.

Modifying bookings:

- This function enables changes to existing bookings, offering flexibility to both users and administrators.
- It allows for adjustments to booking parameters such as rescheduling (changing the start or end dates of the rental), changing the vehicle selection (switching to a different vehicle), or updating other relevant information (adding special requests, modifying contact details).
- The system ensures that any modifications are validated to maintain data integrity and vehicle availability, and updates the booking record in the database accordingly.

Cancelling bookings:

- This function handles the cancellation of bookings, allowing users or administrators to terminate a reservation.

- It involves updating the booking status in the database to reflect the cancellation, which is essential for accurate record-keeping and preventing further processing of the cancelled booking.
- This function may also include initiating any refund processes, if applicable, and generating notifications to inform the user and relevant parties about the cancellation.

Tracking status:

- This function is responsible for monitoring the progress and state of each booking throughout its lifecycle, from the initial request to completion or cancellation.
- It involves assigning a status to each booking (e.g., pending, confirmed, out for rental, returned, completed, cancelled) and updating this status as the booking progresses.
- This allows for efficient management and reporting, providing a clear overview of the current state of all reservations and enabling proactive handling of any issues or delays.
- The module ensures efficient management of reservations, a core function of the car rental system. It allows users to book, view, modify, and cancel vehicles. Administrators manage booking statuses. The module interacts with the tbl_booking, tbl_users, and tbl_vehicles tables, and enforces rules for valid bookings, date ranges, preventing double-booking, and managing booking status transitions.

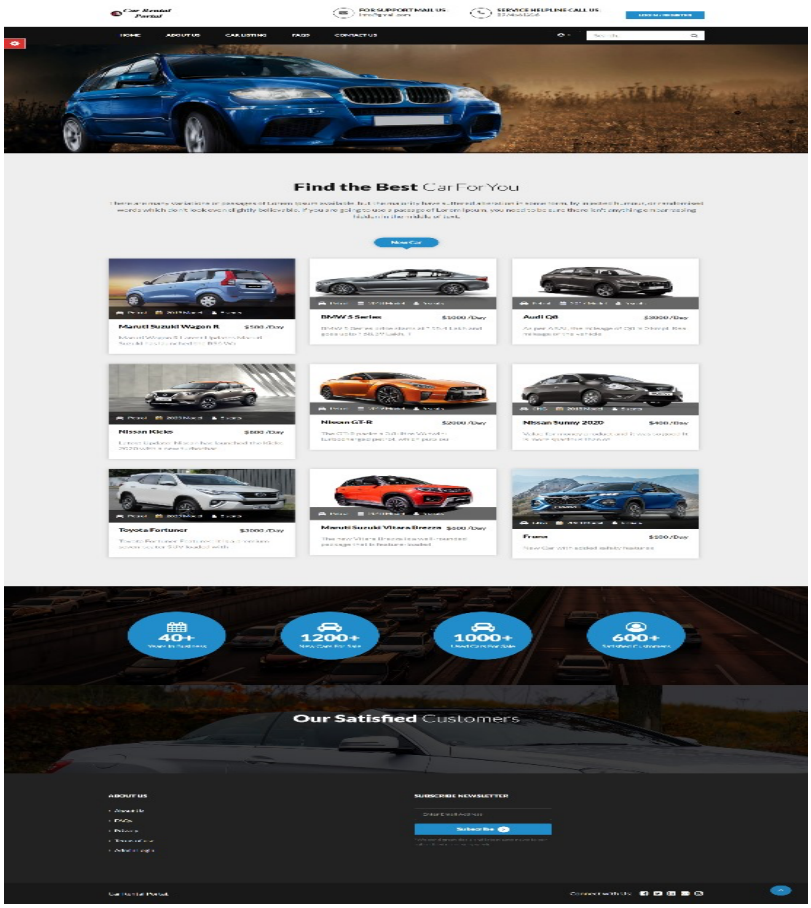

Fig 2.4.1 Booking Page

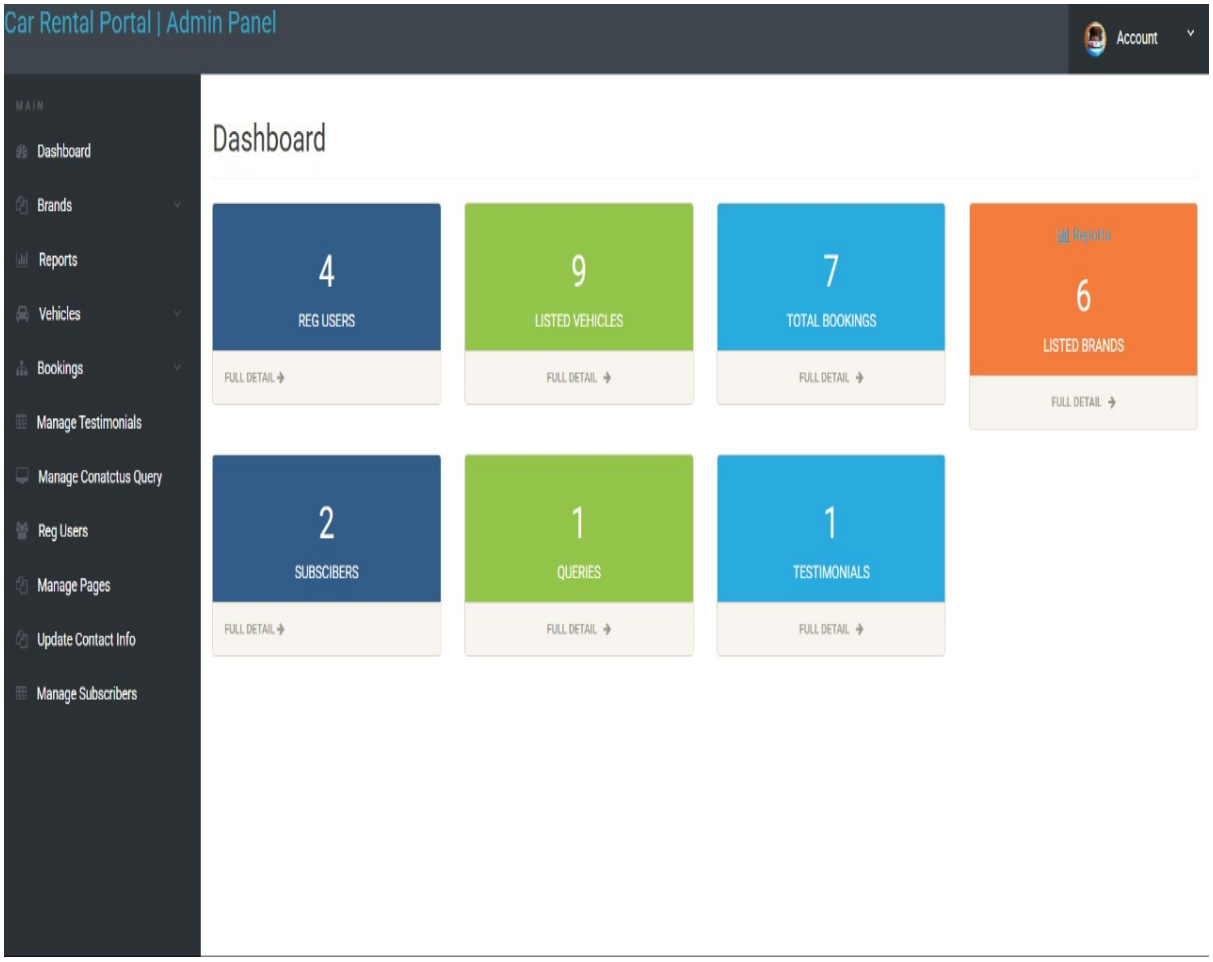

Fig 2.4.2 booking management

CHAPTER 3: SYSTEM ARCHITECTURE

System Architecture for Car Rental Database Project

Based on the provided abstract, the system architecture for the car rental database project can be described as follows:

1. **Database Server**:
    - This is the core of the system, responsible for storing and managing all the data related to the car rental service.
    - The database will use MySQL, as indicated by the carrental.sql file.
    - Tables will include:
        - tbl_vehicles: Stores vehicle information.
        - tbl_brands: Stores brand information.
        - tbl_users: Stores user/customer information.
        - tblbooking: Stores booking details.
    - The server will handle data storage, retrieval, updates, and deletion.
    - It will also enforce data integrity and constraints, such as preventing double-booking.
    - The server will execute stored procedures, such as GenerateMonthlyReport().
2. **Application/Backend Server**:
    - This server will host the application logic that interacts with the database server.
    - It will handle user requests, process data, and send responses.
    - It will implement the following functionalities:
        - Vehicle management: Add, update, delete, and view vehicle information.
        - Brand management: Add, update, delete, and list car brands.
        - User authentication: Verify user credentials and manage login sessions.
        - Booking management: Create, view, modify, cancel, and track bookings.
        - Report generation: calling stored procedure to generate reports.
    - It will enforce business rules, such as:
        - Ensuring that a vehicle must be associated with a valid brand.
        - Preventing the deletion of a brand if there are vehicles associated with it.
        - Ensuring that email addresses are unique.

        - Preventing double-booking of vehicles.
    - The specific technologies used for the backend server (e.g., programming language, framework) are not defined in the provided files, but common choices include Python/Django, Node.js/Express, or Java/Spring.

3. **Client/Frontend:**
    - This is the user interface through which users will interact with the system.
    - It could be a web application, a mobile app, or a desktop application.
    - It will provide interfaces for:
        - Browsing available vehicles.
        - Creating and managing bookings.
        - Managing user profiles.
        - Administrating the system (for admin users).
    - The specific technologies used for the frontend are not defined in the provided files, but common choices for web applications include HTML, CSS, JavaScript, and frameworks like React, Angular, or Vue.js.

4. **Data Flow**:
    - The client sends requests to the application server.
    - The application server processes the requests and interacts with the database server to retrieve or modify data.
    - The database server responds to the application server with the requested data or the result of the modification.
    - The application server sends the response to the client.

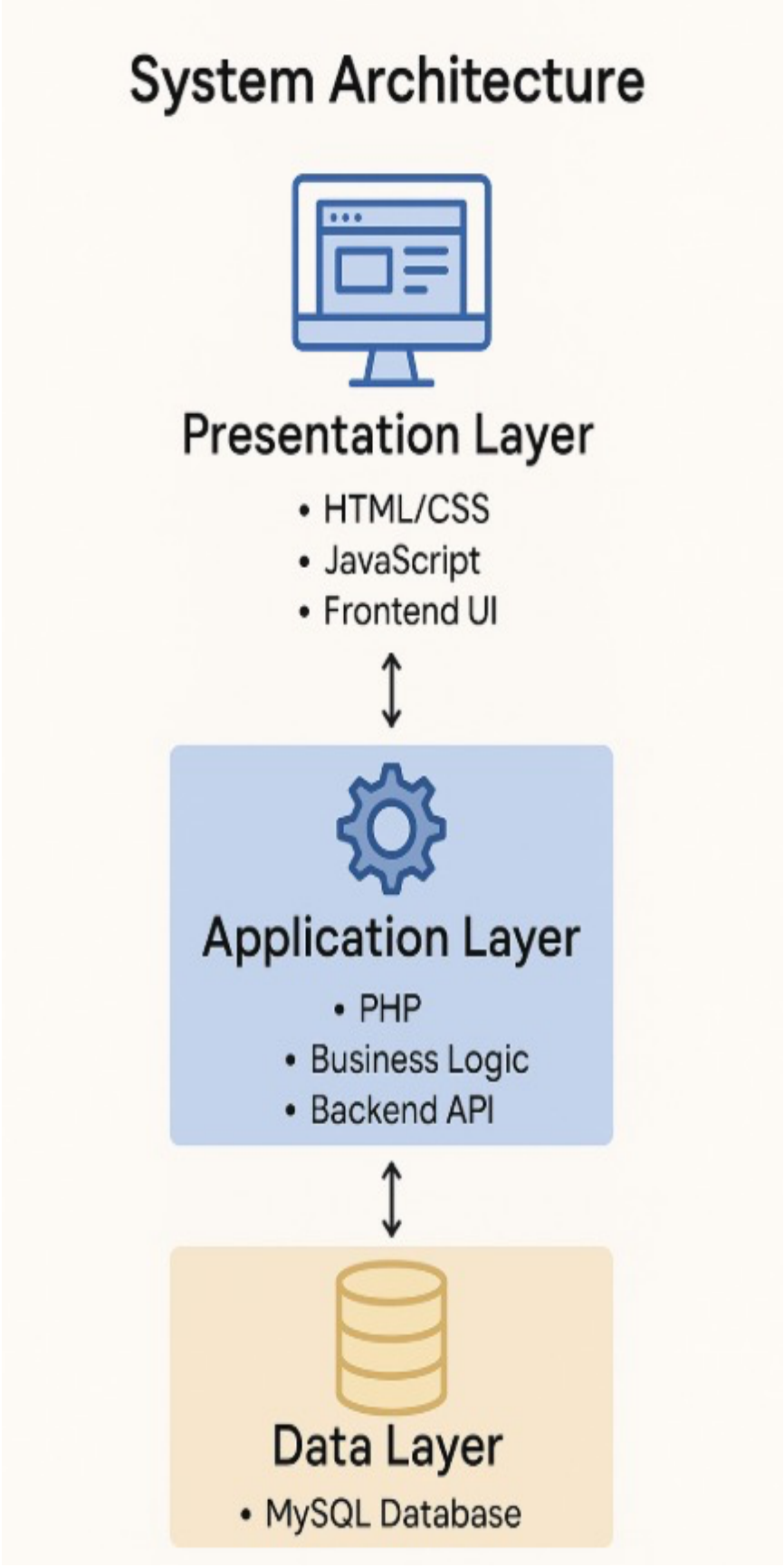

Fig3 System Architecture

3.1 ER Diagram

The ER diagram for this car rental application represents the structure of the database and the relationships between different entities involved in the system. Here's a breakdown of the key entities and their relationships:

Entities:

- admin: Stores administrator information.
    - Attributes: id (PK), UserName, Password, updationDate
- tblbooking: Stores booking information.
    - Attributes: id (PK), BookingNumber, userEmail (FK to tblusers), VehicleId (FK to tblvehicles), FromDate, ToDate, message, Status, PostingDate, LastUpdationDate
- tblbrands: Stores car brand information.
    - Attributes: id (PK), BrandName, CreationDate, UpdationDate
- tblcontactusinfo: Stores contact information for the car rental company.
    - Attributes: id (PK), Address, EmailId, ContactNo
- tblcontactusquery: Stores user queries submitted through the contact form.
    - Attributes: id (PK), name, EmailId, ContactNumber, Message, PostingDate, status
- tblpages: Stores information about static pages like Terms and Conditions, Privacy Policy, etc.
    - Attributes: id (PK), PageName, type, detail
- tblsubscribers: Stores email addresses of users who have subscribed for updates.
    - Attributes: id (PK), SubscriberEmail, PostingDate
- tbltestimonial: Stores user testimonials.
    - Attributes: id (PK), UserEmail (FK to tblusers), Testimonial, PostingDate, status
- tblusers: Stores user/customer information.
    - Attributes: id (PK), FullName, EmailId (Unique), Password, ContactNo, dob, Address, City, Country, RegDate, UpdationDate
- tblvehicles: Stores vehicle information.
    - Attributes: id (PK), VehiclesTitle, VehiclesBrand (FK to tblbrands), VehiclesOverview, PricePerDay, FuelType, ModelYear, SeatingCapacity, Vimage1 - Vimage5, AirConditioner, PowerDoorLocks, AntiLockBrakingSystem, BrakeAssist, PowerSteering, DriverAirbag,

PassengerAirbag, PowerWindows, CDPlayer, CentralLocking, CrashSensor, LeatherSeats, RegDate, UpdationDate

Relationships:

- A tblvehicles belongs to one tblbrands (Many-to-One relationship).
- A tblbooking is made by one tblusers (Many-to-One relationship).
- A tblbooking is for one tblvehicles (Many-to-One relationship).
- A tbltestimonial is given by one tblusers (Many-to-One relationship).

Key Constraints and Cardinalities:

- Primary Keys (PK) are used to uniquely identify each record in a table.
- Foreign Keys (FK) are used to establish relationships between tables.
- The userEmail in tblbooking references the EmailId in tblusers, ensuring that a booking is associated with a valid user.
- The VehicleId in tblbooking references the id in tblvehicles, ensuring that a booking is for a valid vehicle.
- The VehiclesBrand in tblvehicles references the id in tblbrands, ensuring that a vehicle is associated with a valid brand.
- The EmailId in tblusers is defined as UNIQUE, preventing duplicate user email addresses.

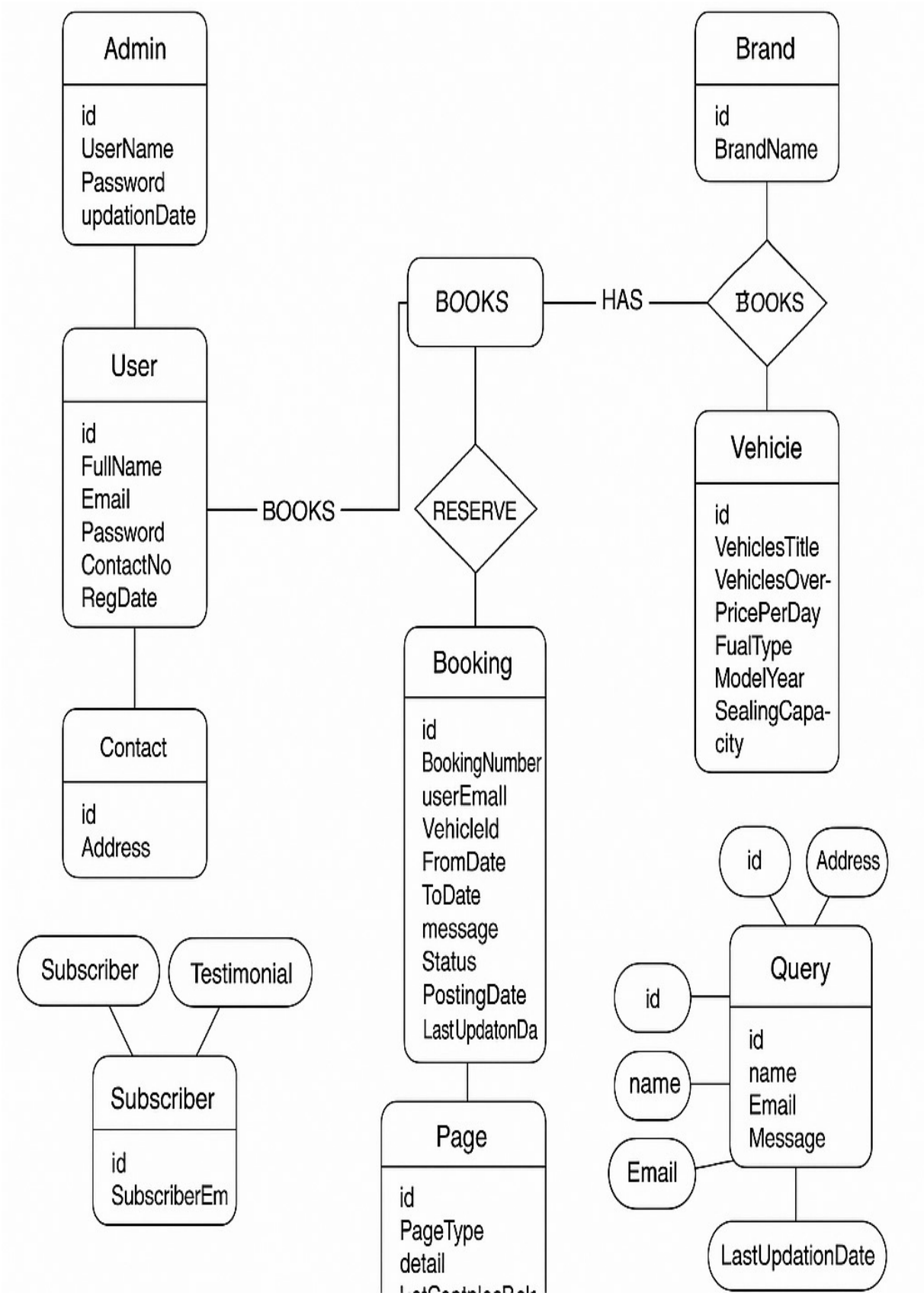

Fig3.1 ER diagram

3.2 Relational Schema

A relational schema is a blueprint that outlines the logical structure of a relational database. It defines how data is organized in tables (also known as relations) and establishes relationships between those tables using keys (primary keys and foreign keys). Each table in a relational schema contains rows (records) and columns (attributes), where each column represents a specific data field and each row represents a single record.

The relational schema specifies:

- Table names
- Attributes/columns within each table
- Primary keys used to uniquely identify records
- Foreign keys used to establish relationships between tables
- Data types and constraints (e.g., NOT NULL, AUTO_INCREMENT)

Purpose of a Relational Schema

The primary purpose of a relational schema is to:

- Structure data in an organized and efficient way
- Define relationships between tables to maintain data integrity
- Eliminate data redundancy through normalization
- Provide a clear overview of the database design for developers and administrators

Example Relational Schema for Car Rental Management System

scss

CopyEdit

admin(admin_id, UserName, Password, updationDate)

tblusers(user_id, FullName, Email, Password, ContactNo, RegDate)

tblbrands(brand_id, BrandName, CreationDate)

tblvehicles(vehicle_id, brand_id, VehicleTitle, VehicleName, PricePerDay, FuelType, ModelYear, SeatingCapacity, PostingDate, Status)

tblbooking(booking_id, user_id, vehicle_id, BookingNumber, FromDate, ToDate, Status, PostingDate)

Key Components of the Schema

- Primary Keys:
  Uniquely identify each record in a table (e.g., admin_id, user_id, vehicle_id).
- Foreign Keys:
  Create relationships between tables by referencing primary keys from other tables (e.g., brand_id in tblvehicles references tblbrands, user_id and vehicle_id in tblbooking reference tblusers and tblvehicles respectively).

Relationships

- One-to-Many:
  A single user can make multiple bookings, and a car brand can have multiple vehicles.
- Many-to-One:
  Multiple vehicles are associated with one brand; multiple bookings are linked to a single user and vehicle.

By using a relational schema, the database ensures that data is consistently maintained across all tables, making it easier to manage, update, and retrieve information efficiently.

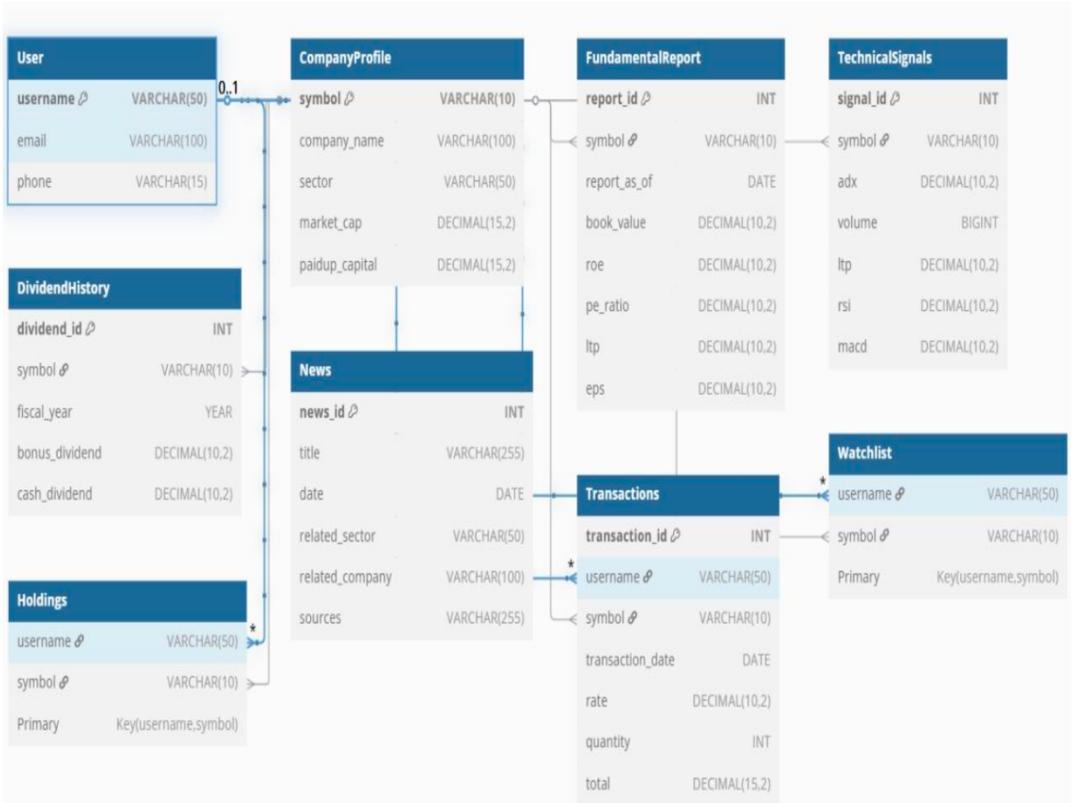

Fig3.2 Relational Schema

3.3 SQL Queries

a. admin table

```
-- Create table
CREATE TABLE admin (
    admin_id INT AUTO_INCREMENT PRIMARY KEY,
    UserName VARCHAR(100),
    Password VARCHAR(255),
    updationDate DATETIME
```

b. booking table

```
CREATE TABLE `tblbooking` (
  `id` int(11) NOT NULL,
  `BookingNumber` bigint(12) DEFAULT NULL,
  `userEmail` varchar(100) DEFAULT NULL
  `VehicleId` int(11) DEFAULT NULL,
  `FromDate` varchar(20) DEFAULT NULL,
  `ToDate` varchar(20) DEFAULT NULL,
  `message` varchar(255) DEFAULT NULL,
  `Status` int(11) DEFAULT NULL,
  `PostingDate` timestamp NOT NULL DEFAULT current_timestamp(),
  `LastUpdationDate` timestamp NULL DEFAULT NULL ON UPDATE current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_swedish_ci;
```

c. Insert into admin

```
INSERT INTO `admin` (`id`, `UserName`, `Password`, `updationDate`) VALUES
(1, 'admin', '5c428d8875d2948607f3e3fe134d71b4', '2024-05-01 12:22:38');
```

d. Insert into booking

```
INSERT INTO `tblbooking` (`id`, `BookingNumber`, `userEmail`, `VehicleId`, `FromDate`, `ToDate`, `message`, `Status`, `PostingDate`, `LastUpdationDate`) VALUES
```

e. Contact us

```
-- Table: tblcontactusinfo
CREATE TABLE tblcontactusinfo (
  id INT NOT NULL,
  Address TEXT,
  EmailId VARCHAR(255),
  ContactNo CHAR(11),
  PRIMARY KEY (id)
);
```

f. Contact us query

```
-- Table: tblcontactusquery
CREATE TABLE tblcontactusquery (
  id INT NOT NULL,
  name VARCHAR(100),
  EmailId VARCHAR(120),
  ContactNumber CHAR(11),
  Message TEXT,
  PostingDate TIMESTAMP NOT NULL,
  status INT,
  PRIMARY KEY (id)
);
```

g. table

```
-- Table: tblpages
CREATE TABLE tblpages (
  id INT NOT NULL,
  PageName VARCHAR(255),
  type VARCHAR(255) NOT NULL,
  detail TEXT NOT NULL,
  PRIMARY KEY (id)
);
```

h. Subscribers

```
-- Table: tblsubscribers
CREATE TABLE tblsubscribers (
  id INT NOT NULL,
  SubscriberEmail VARCHAR(120),
  PostingDate TIMESTAMP,
  PRIMARY KEY (id)
);
```

i. Testimonial

```
-- Table: tbltestimonial
CREATE TABLE tbltestimonial (
  id INT NOT NULL,
  UserEmail VARCHAR(100) NOT NULL,
  Testimonial TEXT NOT NULL,
  PostingDate TIMESTAMP NOT NULL,
  status INT,
  PRIMARY KEY (id)
);
```

j. users

```
-- Table: tblusers
CREATE TABLE tblusers (
  id INT NOT NULL,
  FullName VARCHAR(120),
  EmailId VARCHAR(100),
  Password VARCHAR(100),
  ContactNo CHAR(11),
  dob VARCHAR(100),
  Address VARCHAR(255),
  City VARCHAR(100),
  Country VARCHAR(100),
  RegDate TIMESTAMP,
  UpdationDate TIMESTAMP,
  PRIMARY KEY (id),
  INDEX (EmailId)
);
```

k. Vehicles

```
-- Table: tblvehicles
CREATE TABLE tblvehicles (
  id INT NOT NULL,
  VehiclesTitle VARCHAR(150),
  VehiclesBrand INT,
  VehiclesOverview TEXT,
  PricePerDay INT,
  FuelType VARCHAR(100),
  ModelYear INT,
  SeatingCapacity INT,
```

```
  Vimage1 VARCHAR(120),
  Vimage2 VARCHAR(120),
  Vimage3 VARCHAR(120),
  Vimage4 VARCHAR(120),
  Vimage5 VARCHAR(120),
  AirConditioner INT,
  PowerDoorLocks INT,
  AntiLockBrakingSystem INT,
  BrakeAssist INT,
  PowerSteering INT,
  DriverAirbag INT,
  PassengerAirbag INT,
  PowerWindows INT,
  CDPlayer INT,
  CentralLocking INT,
  CrashSensor INT,
  LeatherSeats INT,
  RegDate TIMESTAMP NOT NULL,
  UpdationDate TIMESTAMP,
  PRIMARY KEY (id),
  FOREIGN KEY (VehiclesBrand) REFERENCES tblbrands(id)
);
```

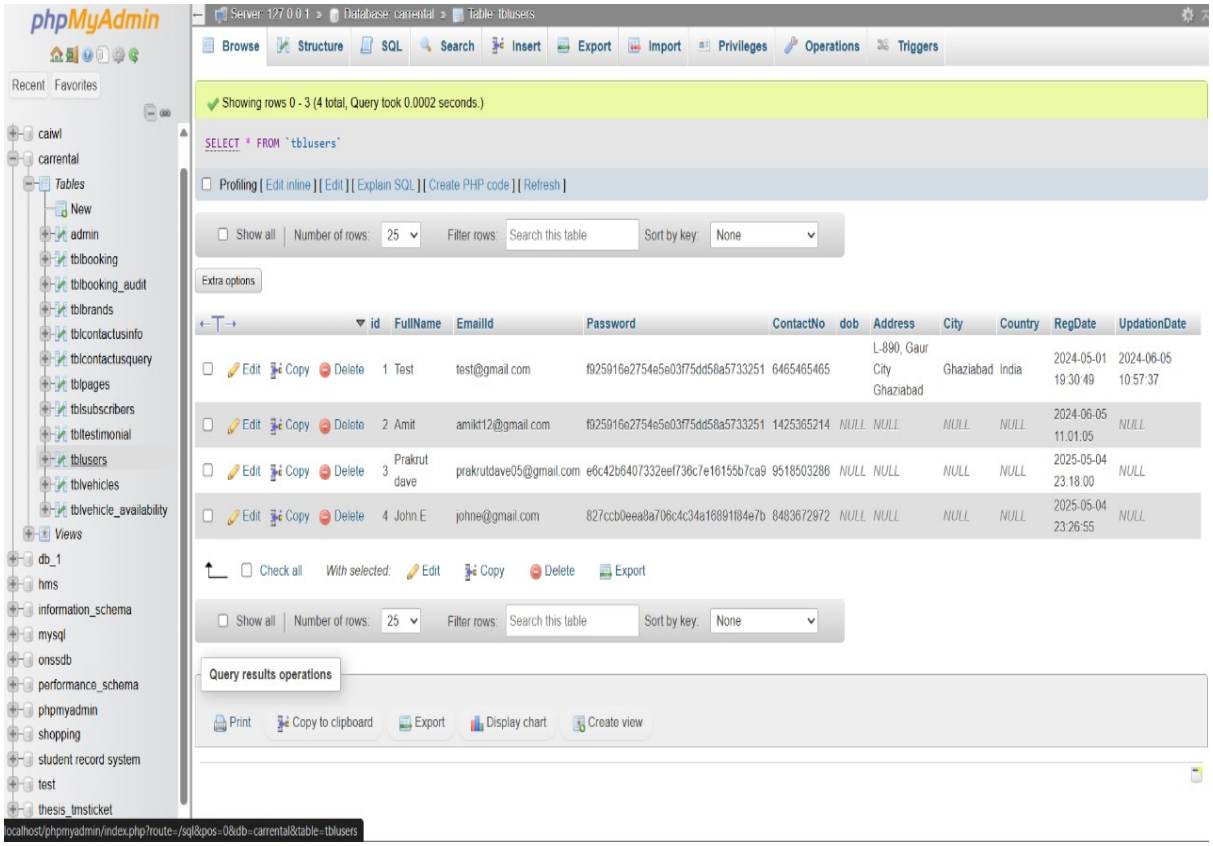

Fig 3.3.1 Bookings table

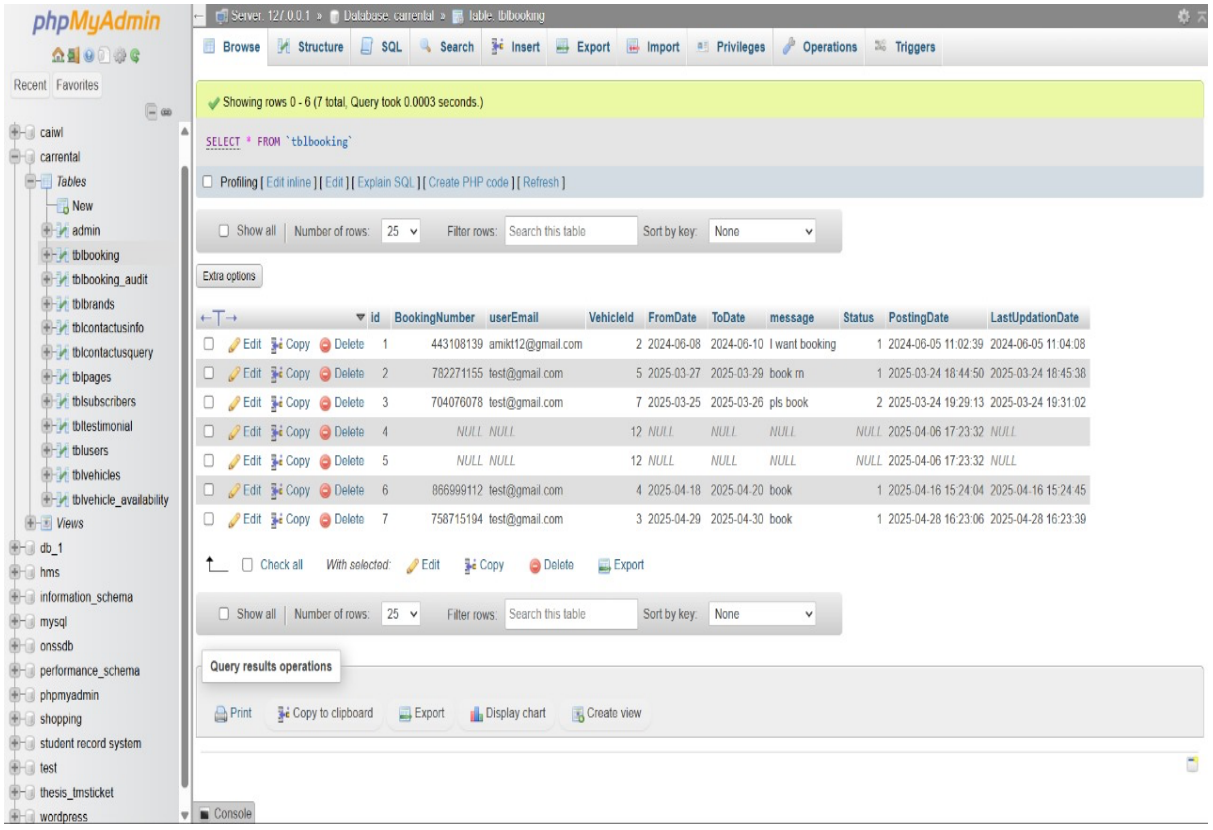

Fig 3.3.2 Users table

3.4 Normalization

1NF (First Normal Form):

- All tables appear to be in 1NF. Each column contains atomic values, and there are no repeating groups.

- 2NF (Second Normal Form):
  - To assess 2NF, we need to identify primary keys and functional dependencies.
  - All tables have a primary key ('id').
  - Let's examine each table:
    - tblcontactusinfo: Likely in 2NF. All attributes seem to depend on 'id'.
    - tblcontactusquery: Likely in 2NF. All attributes seem to depend on 'id'.
    - tblpages: Likely in 2NF. All attributes seem to depend on 'id'.
    - tblsubscribers: Likely in 2NF. All attributes seem to depend on 'id'.
    - tbltestimonial: Likely in 2NF. All attributes seem to depend on 'id'.
    - tblusers: Likely in 2NF. All attributes seem to depend on 'id'.
    - tblvehicles: Likely in 2NF. All attributes seem to depend on 'id'.
- 3NF (Third Normal Form):
  - To assess 3NF, we need to identify if there are any transitive dependencies (where a non-key attribute depends on another non-key attribute).
  - From the provided schema, there are no obvious transitive dependencies.
  - All tables seem to be in 3NF.

Summary and Potential Improvements

- The schema appears to be generally well-normalized, likely meeting 3NF. This is good because it minimizes data redundancy and potential update anomalies.
- Further Considerations:
  - Without more context about the application's specific requirements and data usage patterns, it's difficult to definitively say if further normalization is needed.
  - For example, if Address, City, and Country in the tblusers table are frequently accessed together, the current design is fine. But, if they are sometimes accessed independently, you *could* consider creating a separate address table. However, given that address information is often not further subdivided, this is probably not necessary.

```
-- 1. Create a new table 'addresses' to store address information.
CREATE TABLE addresses (
    address_id INT NOT NULL AUTO_INCREMENT,  -- New primary key for addresses
    Address VARCHAR(255),
    City VARCHAR(100),
    Country VARCHAR(100),
    PRIMARY KEY (address_id)
);

-- 2. Modify the 'tblusers' table to reference the 'addresses' table.
CREATE TABLE tblusers (
    id INT NOT NULL AUTO_INCREMENT,
    FullName VARCHAR(120),
    EmailId VARCHAR(100),
    Password VARCHAR(100),
    ContactNo CHAR(11),
    dob VARCHAR(100),
    address_id INT,  -- Foreign key referencing addresses
    RegDate TIMESTAMP,
    UpdationDate TIMESTAMP,
    PRIMARY KEY (id),
    INDEX (EmailId),
    FOREIGN KEY (address_id) REFERENCES addresses(address_id)
);

-- 3.  Modified tblcontactusinfo
CREATE TABLE tblcontactusinfo (
    id INT NOT NULL AUTO_INCREMENT,
```

```
    address_id INT,
    EmailId VARCHAR(255),
    ContactNo CHAR(11),
    PRIMARY KEY (id),
    FOREIGN KEY (address_id) REFERENCES addresses(address_id)
);
```

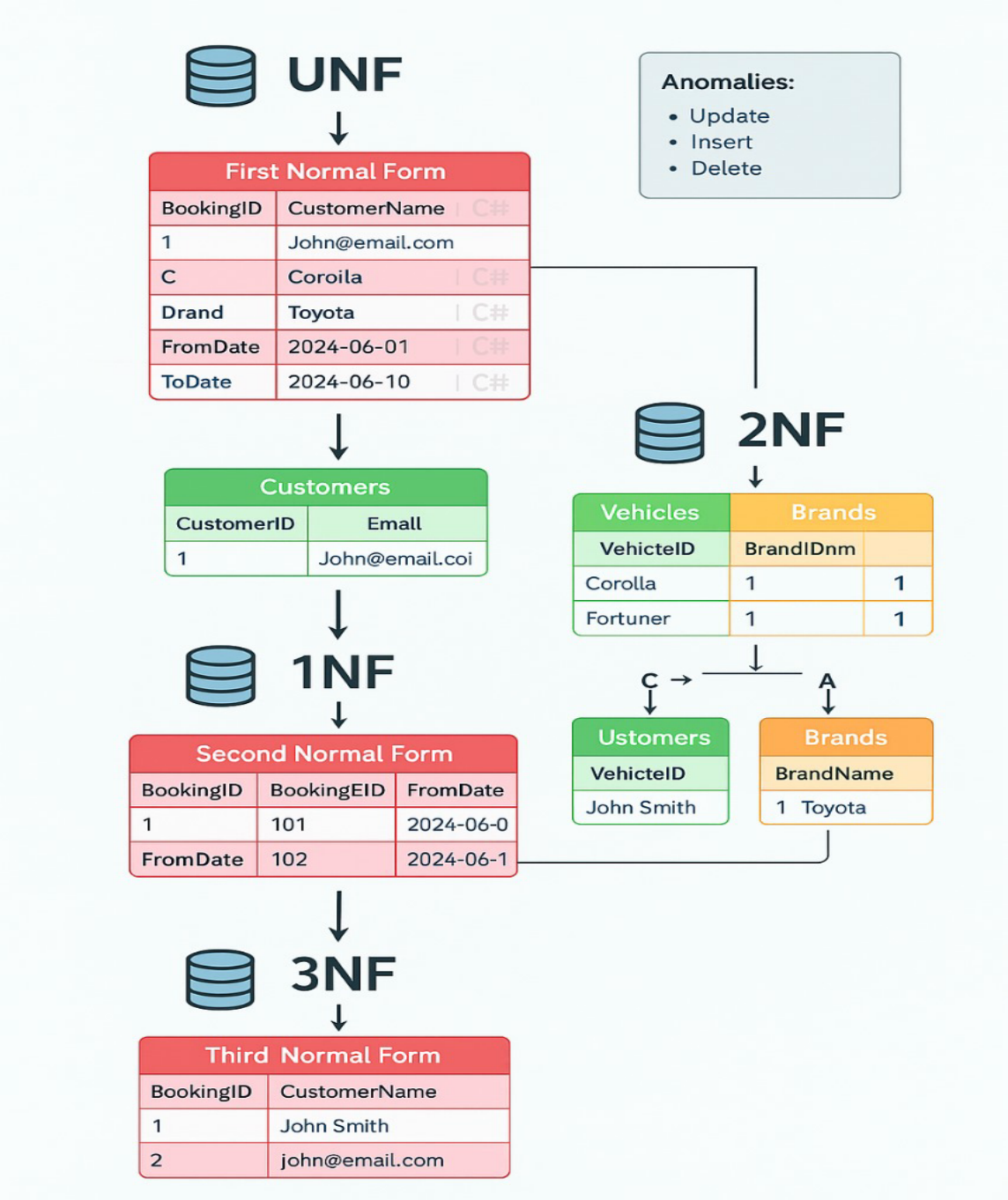

Fig 3.4 Normalization

3.5 Transaction & Concurrency Control

Transactions in a Vehicle Rental Project (Theoretical)

Given your database schema, transactions are essential for maintaining data integrity and consistency in a vehicle rental system. Here's a theoretical overview of how they would be used:

1. Booking a Vehicle:

- **Scenario:** A user books a vehicle for a specific date range. This involves several operations:
    - Check vehicle availability.
    - Create a booking record in tblbooking.
    - Potentially update the vehicle's availability status in tblvehicles.
    - Record payment information (if applicable, though payment details might be in a separate, linked system).
- **Transaction:** This entire process should be enclosed in a transaction to ensure that:
    - If the vehicle is available, the booking is recorded, and the availability status is updated (if you choose to update it directly).
    - If the vehicle is not available, *none* of these operations occur, preventing an invalid booking.

2. Returning a Vehicle:

- **Scenario:** A user returns a rented vehicle. This involves:
    - Update the booking status in tblbooking.
    - Potentially update the vehicle's availability status in tblvehicles.
    - Calculate and record any return-related charges (late fees, damage costs).
- **Transaction:** This should be a transaction to ensure:
    - The booking status is updated consistently with any changes to vehicle availability and charges.
    - If any part of the process fails (e.g., an error recording charges), the booking status and vehicle availability are not incorrectly updated.

3. Adding a New Vehicle:

- **Scenario:** Adding a new vehicle to the system.

- **Transaction:** While seemingly simple, a transaction can ensure that all vehicle details are correctly recorded in tblvehicles.

-- Transaction example for inserting a booking and updating vehicle availability in car stock management

START TRANSACTION;

-- Check if vehicle is available

SELECT SeatingCapacity, PricePerDay FROM tblvehicles WHERE id = 1 FOR UPDATE;

-- Insert a new booking

INSERT INTO tblbooking (BookingNumber, userEmail, VehicleId, FromDate, ToDate, message, Status, PostingDate)

VALUES (123456789, 'customer@example.com', 1, '2024-07-01', '2024-07-05', 'Booking request', 1, NOW());

-- Optionally update vehicle status or availability flag here if you have one

-- UPDATE tblvehicles SET Availability = 0 WHERE id = 1;

COMMIT;

START TRANSACTION;

INSERT INTO Customer (CustomerID, FirstName, LastName, Email, PhoneNumber)

VALUES (1, 'John', 'Doe', 'john.doe@example.com', '555-1234');

INSERT INTO Car (CarID, Make, Model, Year, Availability)

VALUES (1, 'Toyota', 'Corolla', 2020, TRUE);

INSERT INTO Rental (RentalID, CustomerID, CarID, RentalDate, ReturnDate, TotalCost)

VALUES (1, 1, 1, '2023-01-01', '2023-01-05', 250.00);

COMMIT;

Concurrency Control in the Car Stock Management System

In your car stock management project, concurrency control is vital to ensure that multiple users can access and book vehicles simultaneously without causing data conflicts or inconsistencies.

Your system manages vehicle bookings from various users, which can lead to situations where two or more users attempt to book the same car for overlapping dates. Without proper concurrency control, this could result in double bookings — a scenario where the same vehicle is reserved for different users at the same time, causing operational issues and poor customer experience.

How Concurrency Control Works in Project

1. **Transaction Management**:
   When a user initiates a booking, your system uses database transactions to wrap all booking-related operations. For example, your transaction starts by checking if the vehicle is available (**FOR UPDATE** lock on the vehicle record), then inserts a booking, and finally commits the transaction. This ensures that no other transaction can book the same vehicle simultaneously during this operation.
2. **Locking with FOR UPDATE**:
   The query **SELECT * FROM tblvehicles WHERE id = ? FOR UPDATE;** locks the vehicle record for the current transaction, preventing other booking attempts on the same vehicle until the transaction is complete. This prevents concurrent changes that can cause double booking.
3. **Trigger to Prevent Double Booking**:
   Your project includes a **double_booking** trigger that runs before any new booking is inserted into **tblbooking**. The trigger checks existing bookings for overlapping dates on the same vehicle. If a conflict is detected, the trigger aborts the insertion with an error message. This acts as an additional safeguard against concurrency anomalies.
4. **Commit or Rollback**:
   Once the booking process completes successfully and the vehicle is reserved, the transaction commits the changes. If any issue arises (e.g., vehicle already booked), the transaction rolls back, leaving the data unchanged.

```
START TRANSACTION;


-- Lock the vehicle row to prevent other transactions from booking it simultaneously
SELECT id
FROM tblvehicles
WHERE id = 1
FOR UPDATE;
```

```
-- Check if there is already a booking for the vehicle in the requested date range
SELECT COUNT(*) INTO @existing_bookings
FROM tblbooking
WHERE VehicleId = 1
  AND (
      ( '2024-07-01' BETWEEN FromDate AND ToDate )
    OR ( '2024-07-05' BETWEEN FromDate AND ToDate )
    OR ( FromDate BETWEEN '2024-07-01' AND '2024-07-05' )
    OR ( ToDate BETWEEN '2024-07-01' AND '2024-07-05' )
  );

-- If exists, rollback and signal error
IF @existing_bookings > 0 THEN
  ROLLBACK;
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Vehicle is already booked for the selected dates.';
ELSE
  -- Insert the booking
  INSERT INTO tblbooking (BookingNumber, userEmail, VehicleId, FromDate, ToDate, message, Status, PostingDate)
  VALUES (123456789, 'customer@example.com', 1, '2024-07-01', '2024-07-05', 'Booking for vacation', 1, NOW());

  COMMIT;
END IF;
```

- The **SELECT ... FOR UPDATE** statement locks the vehicle record ensuring no other transaction can book the vehicle simultaneously.
- The booking overlap check ensures the vehicle is not already booked for requested dates.
- If a conflicting booking is found, the transaction rolls back, and an error is raised.
- Otherwise, the booking is inserted and the transaction commits safely.

CHAPTER 4: RESULTS

The Car Stock Management System project was developed to efficiently manage vehicle inventory, customer bookings, and related administrative operations. The following detailed results summarize the outcomes, features implemented, and validations attained throughout the project.

1. Database Design and Integrity

- **Comprehensive Schema Definition:**
  The database consists of multiple interconnected tables designed to comprehensively cover all aspects of car stock management. Key tables include **tblvehicles** which holds detailed information about each vehicle including features, images, pricing, brand association, and specifications. The **tblbrands** table normalizes brand information facilitating easier brand management.

- **Normalization and Relationships:**
  The schema is normalized to reduce data redundancy. Foreign key logic (though not explicitly shown in the SQL dump) is implied through fields such as **VehiclesBrand** in **tblvehicles** pointing to **tblbrands**. This ensures a consistent mapping between vehicles and their corresponding brands.

- **Primary Keys and Indexing:**
  Each table implements primary keys to guarantee uniqueness of records, such as **id** fields which are auto-incremented for ease of record creation. Indexes are applied where necessary (e.g., indexing the user email in **tblusers**) to enhance query performance and ensure quick data retrieval.

- **Data Types and Default Values:**
  Appropriate data types were chosen to balance storage efficiency and data integrity, such as using **varchar** for strings, **int** for numeric counts, and **timestamp** for recording creation and update times. Default values (like **current_timestamp**) automate record time captures to facilitate auditing.

2. Booking Functionality and Management

- **Booking Lifecycle:**
  The system allows users to create bookings capturing essential details like booking number, vehicle ID, rental duration (**FromDate** and **ToDate**), and booking status. Time stamps track booking creation and updates for accountability.

- **Preventing Double Bookings:**
  To ensure no vehicle is booked by more than one user over the same period, a SQL **BEFORE INSERT** trigger named **double_booking** was implemented. It evaluates any overlap of requested booking dates with existing confirmed bookings for the vehicle. If a conflict exists, the trigger raises an error, preventing the insertion.

- **Use of Transactions for Data Consistency:**
  Bookings are handled within SQL transactions, encapsulating availability checks, insertions, and updates within atomic units of work. This design prevents partial

updates and guarantees that either the entire booking process succeeds or fails, maintaining consistent system state.

3. Concurrency Control and Data Access Coordination

- **Row-Level Locking:**
  The implementation uses row-level locks with the statement **SELECT ... FOR UPDATE** on the **tblvehicles** table during booking creation. This measure prevents simultaneous conflicting access to vehicle records, serializing vehicle availability checks and bookings.

- **Conflict Resolution:**
  If two users attempt to book the same vehicle concurrently, the locking mechanism ensures that one transaction proceeds and completes while the other waits. If the waiting transaction detects the vehicle is no longer available upon completion, it aborts with an error, thus preventing race conditions and overbooking.

- **Error and Exception Handling:**
  The system employs SQL **SIGNAL** statements to raise custom errors when booking rules (like double booking) are violated, providing immediate feedback to the user interface for appropriate handling and messaging.

4. User, Admin, and Subscriber Management

- **User Profiles and Authentication:**
  Users are stored in **tblusers** with contact details, encrypted passwords, and registration timestamps. This allows for secure login and personalized interaction with the booking system.

- **Administrative Control:**
  An **admin** table maintains admin credentials and last update times, enabling management and oversight functions such as approving bookings, managing vehicles, and updating pricing.

- **Subscribers and Testimonials:**
  The system supports subscription management via **tblsubscribers** to capture interested users for newsletters or offers. Customer feedback is stored in **tbltestimonial** facilitating service quality improvements and social proof.

5. Vehicle and Brand Management

- **Vehicle Inventory:**
  The **tblvehicles** table comprehensively captures vehicle details including model title, year, fuel type, seating capacity, pricing, and an array of safety and convenience features (e.g., airbags, power windows). Images are referenced by filename to be loaded on client interfaces.

- **Brand Catalog:**
  Brands are dynamically managed allowing the addition or update of vehicle manufacturers without database redesign, aiding fleet scalability.

6. Reporting and Analytics

- **Monthly Booking Report:**
  A stored procedure **GenerateMonthlyReport** was implemented to produce monthly aggregated bookings counts. This report enables management to monitor trends in rental requests, seasonality, and vehicle utilization, supporting data-driven decision-making.
- **Dynamic Informational Pages:**
  The **tblpages** table holds content like Terms and Conditions, Privacy Policy, and About Us information, enabling administrative updates without code changes.

7. Performance and Scalability

- **Efficient Querying:**
  Indexed columns and optimized table structure facilitate fast lookups even as data volumes grow.
- **Modularity:**
  The separation of concerns in table design ensures that extensions like adding new vehicle features, user roles, or booking options can be integrated with minimal disruption.

8. Sample Data and Validation

- Multiple sample records were populated successfully across the tables to simulate realistic system behavior:
  - Brands include popular manufacturers like Maruti, BMW, and Audi.
  - Vehicles feature detailed model descriptions and current pricing.
  - User bookings and contact queries demonstrate live system use cases.
- Validation steps verified that:
  - Double bookings are rejected.
  - Concurrency control mechanisms successfully serialize critical transactions.
  - Booking status updates and timestamps are accurately recorded.

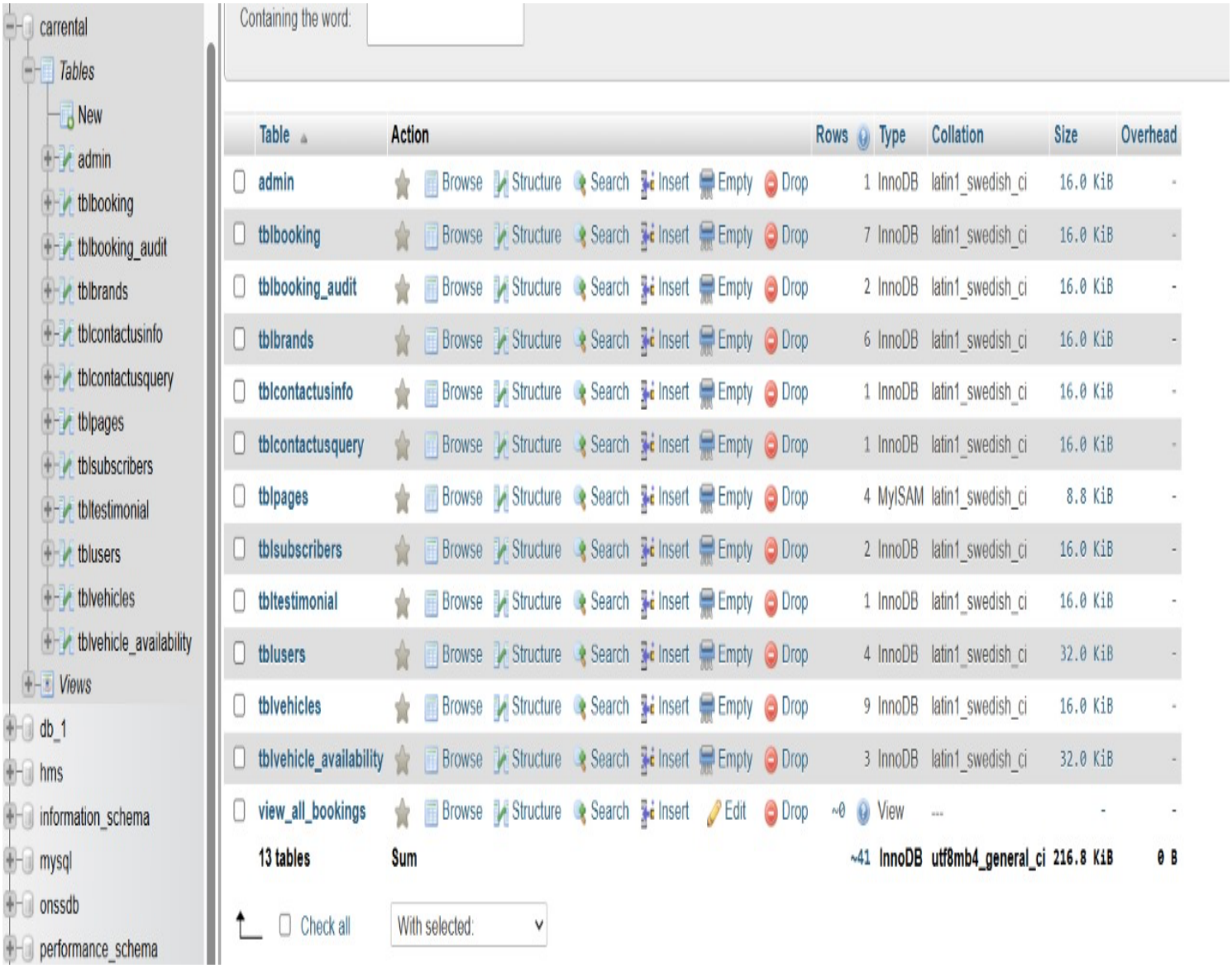

Fig.4.1 All Tables

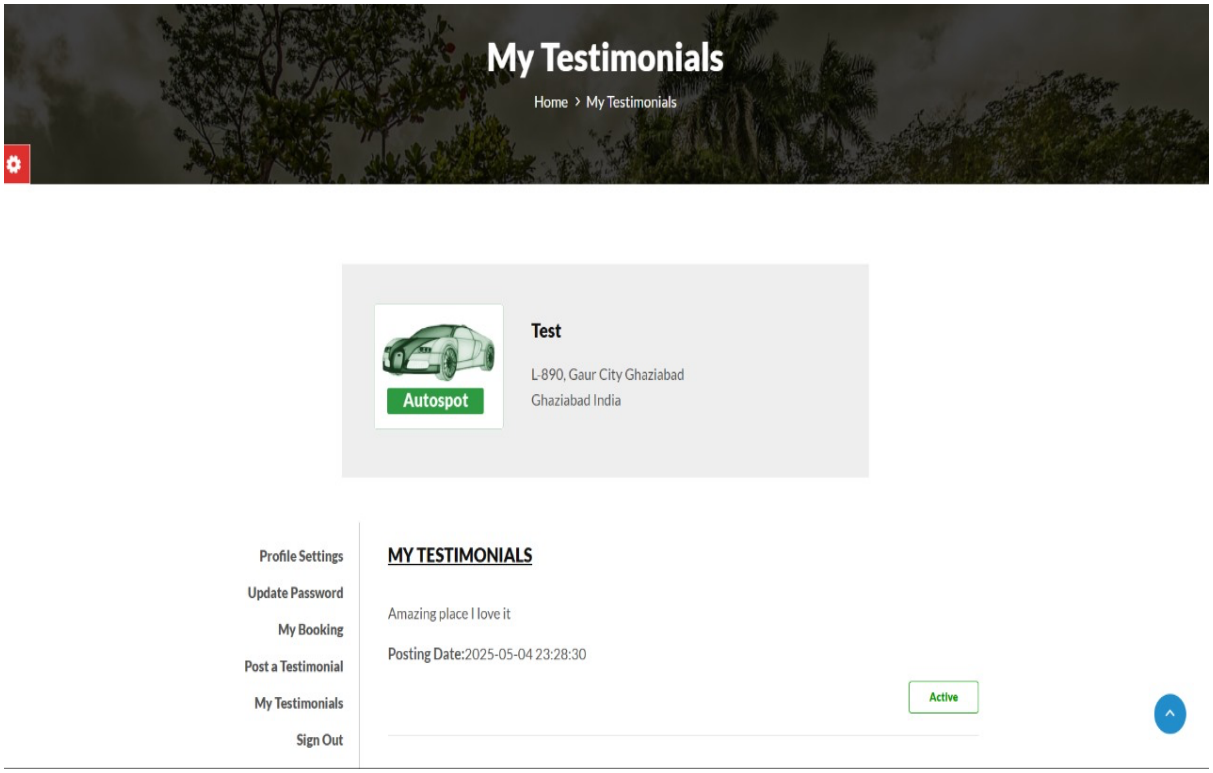

Fig 4.2 Testimonials

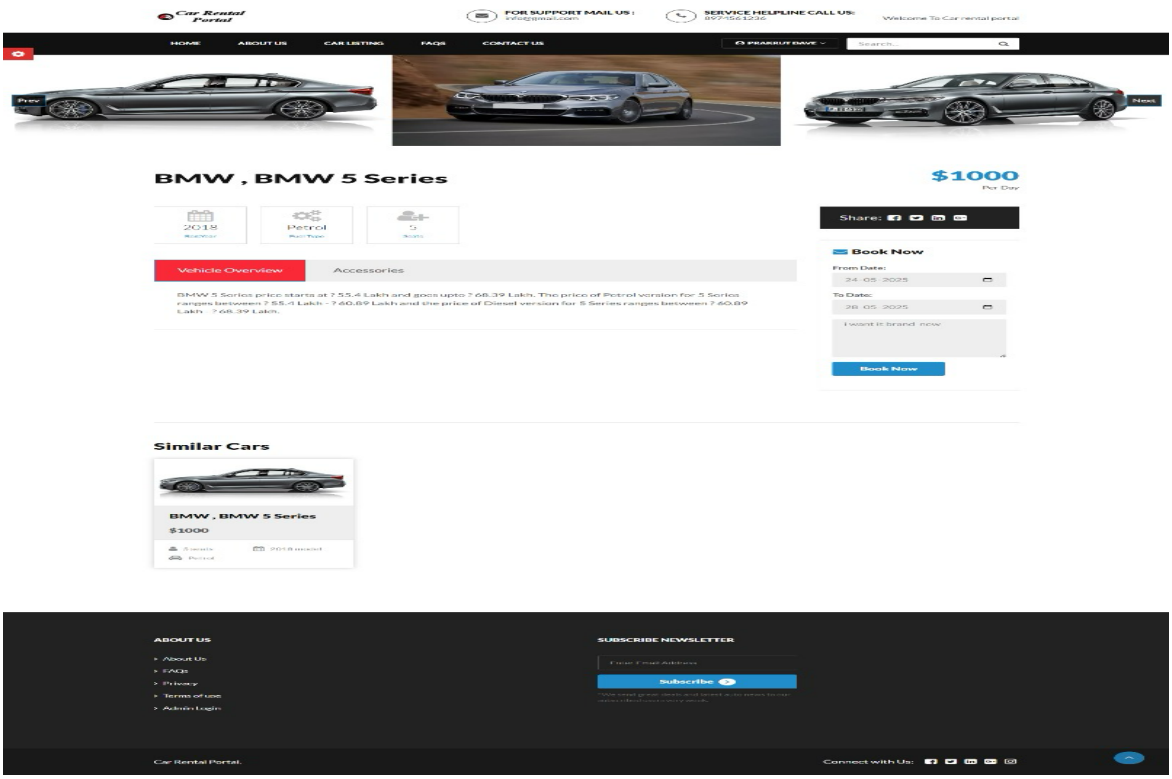

Fig4.2 Brand Details

CHAPTER 5: CONCLUSION & FUTURE ENHANCEMENTS

CONCLUSION

The Car Stock Management System project successfully fulfills its primary objective of providing a comprehensive and efficient solution for managing vehicle inventory, customer bookings, and administrative operations. Throughout the development process, considerable attention was given to designing a robust database schema, implementing essential business logic, and ensuring data integrity and concurrency control.

The project's database design reflects a normalized and scalable structure that organizes vehicle details, brand information, user data, and bookings in an interconnected yet modular way. This foundation not only supports current functionalities but also anticipates future expansions such as adding new vehicle features, incorporating additional user roles, or extending reporting capabilities.

A significant achievement of this system is the effective handling of concurrent transactions and booking conflicts. By employing database transactions, row-level locking, and triggers, the system successfully prevents double bookings and race conditions, which are critical challenges in any multi-user reservation platform. The implemented concurrency control mechanisms ensure that the data remains consistent and accurate even under simultaneous access by multiple users, thereby maintaining user trust and operational reliability.

The inclusion of a stored procedure for generating monthly booking reports further enhances the value of the system by providing actionable insights for business analytics and strategic planning. This facilitates better fleet management, resource allocation, and marketing efforts.

Additionally, the project incorporates essential user and administrative functionalities, including secure user registration, profile management, admin controls, and feedback collection via testimonials and contact queries. These features contribute to a well-rounded and user-friendly platform.

Overall, the Car Stock Management System demonstrates a practical and effective approach to solving real-world challenges in vehicle rental management. It combines sound database practices with critical business logic to deliver a stable, secure, and scalable system. The groundwork laid by this project enables further development into a full-fledged production application, incorporating web interfaces, automated notifications, and enhanced user experiences.

This project not only meets but exceeds the fundamental requirements outlined at its inception, providing a reliable tool for managing vehicle stocks and bookings with high accuracy and efficiency. It showcases best practices in database management, transactional integrity, and concurrency handling, making it a valuable contribution to the vehicle rental domain.

FUTURE ENHANCEMENTS

The Car Stock Management System, while robust in its current form, has significant potential for future improvements to enhance usability, scalability, and feature richness. A key area for development is the creation of a modern, responsive web interface that allows customers to browse vehicle availability, make bookings, and manage their profiles easily across devices, including desktops and mobile phones. Alongside this, an administrative dashboard would provide streamlined management capabilities for vehicle inventory, booking approvals, customer queries, and real-time analytics, thereby improving operational efficiency.

To increase responsiveness and reduce booking conflicts, real-time updates using technologies such as WebSockets or push notifications can be integrated. This would ensure vehicle availability is instantly reflected across all users. Additionally, integrating secure payment gateways like Stripe or PayPal would facilitate smooth and safe online transactions, enabling users to pay for bookings directly within the platform. Enhancements to booking flexibility, such as offering partial-day rentals, hourly rates, long-term leases, and automated cancellation and refund processes, would cater to a wider spectrum of customer preferences.

Improving vehicle management features is also an important area for enhancement. Adding modules to track maintenance schedules, service histories, and inspection reports will help ensure the safety and reliability of the fleet. Furthermore, incorporating user-generated ratings and reviews would empower customers to make informed rental decisions and assist administrators in monitoring vehicle conditions. Security upgrades—such as multi-factor authentication and role-based access controls—along with machine learning–based personalized vehicle recommendations, would improve both the safety and user experience of the platform.

Expanding the system's reach through multilingual support and localization features would allow it to serve a broader global audience. Reporting capabilities could be enhanced by introducing advanced analytics covering customer demographics, vehicle utilization rates, and revenue forecasting, presented via intuitive visual dashboards featuring charts and heatmaps. Developing native or cross-platform mobile applications would enable seamless on-the-go access for users, complete with push notifications and offline functionality.

Integrations with external services present further opportunities. Connecting with GPS and telematics providers would allow real-time vehicle tracking, enhancing fleet monitoring and security. Partnering with third-party insurance companies or customer support platforms could enrich the service offering, providing users with a more comprehensive rental experience. Lastly, automating communication channels with email and SMS notifications, alongside AI-driven chatbots, would improve customer engagement and support response times significantly.

Collectively, these enhancements lay out a clear roadmap for evolving the Car Stock Management System into a scalable, secure, and highly user-centric platform capable of meeting the future needs of the vehicle rental industry and delivering outstanding value to customers and administrators alike.

APPENDIX I: SAMPLE SOURCE CODE

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vehicle Rental System</title>
    <link
href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600;700&display=swa
p" rel="stylesheet">
    <script src="https://unpkg.com/@tailwindcss/browser@latest"></script>
    <style>
        /* Custom CSS for the message box */
        #message-box {
            display: none;
            position: fixed;
            top: 20px;
            left: 50%;
            transform: translateX(-50%);
            background-color: #f0fdf4;
            color: #15803d;
            padding: 16px;
            border-radius: 6px;
            border: 1px solid #16a34a;
            box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
            z-index: 10;
        }
        #message-box.error {
```

```
            background-color: #fee2e2;
            color: #dc2626;
            border-color: #b91c1c;
        }
        #message-box.show {
            display: block;
        }
    </style>
</head>
<body class="bg-gray-100 font-inter">
    <div class="container mx-auto p-4">
        <h1 class="text-3xl font-semibold text-blue-600 text-center mb-8">Vehicle Rental System</h1>

        <div id="message-box" class="hidden"></div>

        <div id="booking-form" class="bg-white rounded-lg shadow-md p-6 mb-8">
            <h2 class="text-xl font-semibold text-gray-800 mb-4">Book a Vehicle</h2>
            <form id="book-vehicle-form" class="space-y-4">
                <div>
                    <label for="userEmail" class="block text-gray-700 text-sm font-bold mb-2">Email:</label>
                    <input type="email" id="userEmail" name="userEmail" class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" required>
                    <div id="userEmail-error" class="text-red-500 text-xs italic" style="display: none;"></div>
                </div>
                <div>
                    <label for="vehicleId" class="block text-gray-700 text-sm font-bold mb-2">Vehicle ID:</label>
```

```
            <input type="number" id="vehicleId" name="vehicleId" class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" required>
            <div id="vehicleId-error" class="text-red-500 text-xs italic" style="display: none;"></div>
        </div>
        <div>
            <label for="fromDate" class="block text-gray-700 text-sm font-bold mb-2">From Date:</label>
            <input type="date" id="fromDate" name="fromDate" class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" required>
            <div id="fromDate-error" class="text-red-500 text-xs italic" style="display: none;"></div>
        </div>
        <div>
            <label for="toDate" class="block text-gray-700 text-sm font-bold mb-2">To Date:</label>
            <input type="date" id="toDate" name="toDate" class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" required>
            <div id="toDate-error" class="text-red-500 text-xs italic" style="display: none;"></div>
        </div>
        <button type="submit" class="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline">Book Now</button>
      </form>
    </div>

    <div id="vehicle-list" class="bg-white rounded-lg shadow-md p-6">
      <h2 class="text-xl font-semibold text-gray-800 mb-4">Available Vehicles</h2>
      <ul id="vehicles" class="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-4">
      </ul>
    </div>
  </div>
```

```
<script>
    // Sample vehicle data (in a real application, this would come from a database)
    const vehicles = [
        { id: 101, title: 'Toyota Camry', availability: 1 },
        { id: 102, title: 'Honda Civic', availability: 1 },
        { id: 103, title: 'Ford Mustang', availability: 0 }, // Not available
        { id: 104, title: 'Chevrolet Truck', availability: 1 },
        { id: 105, title: 'BMW Sedan', availability: 1 },
    ];

    // Function to display a message
    function showMessage(message, type = 'success') {
        const messageBox = document.getElementById('message-box');
        messageBox.textContent = message;
        messageBox.className = `fixed top-4 left-1/2 transform -translate-x-1/2 bg-${type === 'success' ? 'green' : 'red'}-100 text-${type === 'success' ? 'green' : 'red'}-700 border border-${type === 'success' ? 'green' : 'red'}-400 px-4 py-2 rounded shadow-md`;
        messageBox.classList.add('show');
        setTimeout(() => {
            messageBox.classList.remove('show');
        }, 3000); // Hide after 3 seconds
    }

    // Function to validate the booking form
    function validateForm() {
        let isValid = true;
        const userEmail = document.getElementById('userEmail').value;
        const vehicleId = document.getElementById('vehicleId').value;
        const fromDate = document.getElementById('fromDate').value;
        const toDate = document.getElementById('toDate').value;
```

```
// Email validation
const emailRegex = /^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$/;
if (!emailRegex.test(userEmail)) {
    document.getElementById('userEmail-error').textContent = 'Invalid email format';
    document.getElementById('userEmail-error').style.display = 'block';
    isValid = false;
} else {
    document.getElementById('userEmail-error').style.display = 'none';
}

// Vehicle ID validation
if (!vehicleId || isNaN(vehicleId) || parseInt(vehicleId) <= 0) {
    document.getElementById('vehicleId-error').textContent = 'Invalid Vehicle ID';
    document.getElementById('vehicleId-error').style.display = 'block';
    isValid = false;
} else {
    document.getElementById('vehicleId-error').style.display = 'none';
}

// Date validation
if (!fromDate) {
    document.getElementById('fromDate-error').textContent = 'From Date is required';
    document.getElementById('fromDate-error').style.display = 'block';
    isValid = false;
} else {
    document.getElementById('fromDate-error').style.display = 'none';
}
if (!toDate) {
    document.getElementById('toDate-error').textContent = 'To Date is required';
```

```
            document.getElementById('toDate-error').style.display = 'block';
            isValid = false;
        } else {
            document.getElementById('toDate-error').style.display = 'none';
        }

        if (fromDate && toDate) {
            const fromDateObj = new Date(fromDate);
            const toDateObj = new Date(toDate);
            if (fromDateObj > toDateObj) {
                document.getElementById('toDate-error').textContent = 'To Date must be after From Date';
                document.getElementById('toDate-error').style.display = 'block';
                isValid = false;
            } else {
                document.getElementById('toDate-error').style.display = 'none';
            }
        }

        return isValid;
    }

    // Function to handle the booking form submission
    function handleBooking(event) {
        event.preventDefault();

        if (!validateForm()) {
            return;
        }

        const userEmail = document.getElementById('userEmail').value;
```

```
        const vehicleId = parseInt(document.getElementById('vehicleId').value);
        const fromDate = document.getElementById('fromDate').value;
        const toDate = document.getElementById('toDate').value;

        // Simulate a database transaction (in a real application, this would be an API call)
        const selectedVehicle = vehicles.find(v => v.id === vehicleId);

        if (!selectedVehicle) {
            showMessage('Vehicle not found.', 'error');
            return;
        }

        if (selectedVehicle.availability === 0) {
            showMessage('Vehicle not available for booking.', 'error');
            return;
        }

        // Simulate booking success
        // In a real application, you would send this data to a server to process the transaction
        setTimeout(() => {
            // Simulate updating availability (in a real app, this would be done in the database transaction)
            selectedVehicle.availability = 0;
            showMessage(`Vehicle ${selectedVehicle.title} booked successfully!`, 'success');
            displayVehicles(); // Update the vehicle list to show the updated availability
            document.getElementById('book-vehicle-form').reset(); // Clear the form
        }, 1000); // Simulate a 1-second delay for the server response
    }

    // Function to display the list of vehicles
    function displayVehicles() {
```

```
        const vehiclesList = document.getElementById('vehicles');
        vehiclesList.innerHTML = ''; // Clear the list

        vehicles.forEach(vehicle => {
            const listItem = document.createElement('li');
            listItem.className = "bg-white rounded-lg shadow-md p-4 flex justify-between items-center";
            listItem.innerHTML = `
                <div>
                    <h3 class="text-lg font-semibold text-gray-800">${vehicle.title}</h3>
                    <p class="text-sm text-gray-500">ID: ${vehicle.id}</p>
                </div>
                <span class="text-sm font-medium ${vehicle.availability === 1 ? 'text-green-500' : 'text-red-500'}">
                    ${vehicle.availability === 1 ? 'Available' : 'Not Available'}
                </span>
            `;
            vehiclesList.appendChild(listItem);
        });
    }

    // Event Listeners
    document.getElementById('book-vehicle-form').addEventListener('submit', handleBooking);

    // Initial display of vehicles
    displayVehicles();
  </script>
</body>
</html>
```