INTERNSHIP REPORT

A report submitted in partial fulfillment of the requirements for the Award of Degree of

BACHELOR OF TECHNOLOGY
in
INFORMATION TECHNOLOGY
by

Pushpit Sahay
University Roll No: 10300220071

Under Supervision of
Ms. Sanjana Birari
CodeClause
(Duration: 1st May, 2024 to 1st Jun, 2024)

DEPARTMENT OF INFORMATION TECHNOLOGY
HALDIA INSTITUTE OF TECHNOLOGY
(An Autonomous Institution)
Approved by AICTE, Affiliated to Maulana Abul Kalam Azad University of Technology
Haldia, West Bengal - 72165, India
2020-2024

Code Clause

To Whom So IT May Concern

Date - 01 / 06 / 2024

This is to certify that **PUSHPIT SAHAY**, pursuing Information Technology at **HALDIA INSTITUTE OF TECHNOLOGY** has successfully completed an internship with CodeClause from **May-2024 To Jun-2024**.

During this tenure he handled **JAVA DEVELOPMENT INTERN** position.

During the tenure of the Internship, **PUSHPIT SAHAY** has shown a great amount of responsibility, sincerity, and a genuine willingness to learn and zeal to take on new assignments and challenges. In particular, his coordination skills and communication skills are par excellence and his attention to details is impressive

We wish all the very best for your future.

with regards,
CodeClause

SUDAKU CODECLAUSE PVT LTD • CIN U58201PN2023PTC227228 •

MINISTRY OF CORPORATE AFFAIRS
GOVERNMENT OF INDIA

ISO 9001:2015

MSME

Certificate No - CC-CL84039

ACKNOWLEDGEMENT

We express deep gratitude for enthusiasm and valuable suggestions that I got from my guide **Ms. Sanjana Birari**, our trainer and mentor, for successful completion of the dissertation report of the project. This was not possible without his invaluable guidance. Finally, I am thankful to all the people who are related to the project directly or indirectly.

This satisfaction accompanied by the successful completion of any work would be deficient without acknowledging the people who made it achievable and whose stable guidance & encouragement served as an oasis in the desert.

Pushpit Sahay
(10300220071)

ABSTRACT

The Car Rental System is a comprehensive web application designed to streamline car rental management by providing distinct functionalities for two user roles: Admin and Customer. This system addresses the limitations of traditional car rental methods by digitizing processes, enhancing efficiency, accuracy, and user experience. Administrators gain full control over the rental inventory with Create, Read, Update, and Delete (CRUD) operations for car listings, alongside an advanced search functionality for quick retrieval of specific car information and a robust booking request management system to review, approve, or reject booking requests. Customers benefit from an intuitive interface to browse available car listings, utilize an advanced search tool to find their ideal rental vehicle based on various criteria, submit booking requests for specific dates and times, and view their past booking history, ensuring a seamless rental experience. The Car Rental System not only boosts operational efficiency for car rental businesses but also improves customer satisfaction by simplifying the rental process. This project report details the system's design, implementation, and testing, demonstrating its potential as a practical solution for modernizing car rental services.

INDEX

Learning Objectives/Internship Objectives

- Internships are generally thought of to be reserved for college students looking to gain experience in a particular field. However, a wide array of people can benefit from Training Internships in order to receive real world experience and develop their skills.

- An objective for this position should emphasize the skills you already possess in the area and your interest in learning more.

- Internships are utilized in a number of different career fields, including architecture, engineering, healthcare, economics, advertising and many more.

- Some internship is used to allow individuals to perform scientific research while others are specifically designed to allow people to gain first-hand experience working.

- Utilizing internships is a great way to build your resume and develop skills that can be emphasized in your resume for future jobs. When you are applying for a Training Internship, make sure to highlight any special skills or talents that can make you stand apart from the rest of the applicants so that you have an improved chance of landing the position.

INTRODUCTION

The Car Rental System is a dynamic and user-friendly web application designed to revolutionize the car rental industry by providing a seamless and efficient platform for both administrators and customers. This system is tailored to address the inefficiencies and limitations of traditional car rental methods, ensuring a more streamlined and accurate management process.

Purpose and Significance:
• Addresses inefficiencies and limitations of traditional car rental methods.
• Aims to enhance operational efficiency, accuracy, and user experience.

User Roles:
• **Admin:**
- Full control over the rental inventory with CRUD operations for car listings.
- Advanced search functionality for quick retrieval of specific car information.
- Robust booking request management system to review, approve, or reject requests.
• **Customer:**
- Intuitive interface to browse available car listings.
- Advanced search tool to find the ideal rental vehicle based on various criteria.
- Ability to submit booking requests for specific dates and times.
- Option to view past booking history for a comprehensive overview of rental activities.

System Benefits:
• Enhances operational efficiency for car rental businesses.
• Improves customer satisfaction by simplifying the rental process.
• Makes car rentals more accessible and convenient for all users.

Report Outline:
• Details the design, implementation, and testing of the Car Rental System.
• Demonstrates the system's potential as a practical solution for modernizing car rental services.

5.1 Module Description

User Authentication Module:
Objective:
• Enable users to create accounts, log in securely, and book their preferred rental cars.
Features:
• User registration with email verification.
• Secure login and logout functionality.

Car Listing Management Module:
Objective:
• Allow administrators to manage car listings effectively.
Features:
• Create, Read, Update, and Delete (CRUD) operations for car listings.
• Upload and manage car images and details.
• Categorization and tagging of cars for easier management

Booking Request Management Module:
Objective:
• Manage and process customer booking requests efficiently.
Features:
• View and review booking requests.
• Approve or reject booking requests.
• Update booking status and notify customers.

Advanced Car Search Module:
Objective:
• Implement a powerful search engine to help users find the perfect rental car.
Features:
• Integration with a comprehensive car database.
• Full-text search capabilities.
• Advanced search filters for criteria such as car model, price range, and availability.

Customer Browsing and Booking Module:
Objective:
• Provide customers with a seamless interface to browse and book rental cars.
Features:
• Display available cars with detailed descriptions and images.
• Filter and sort car listings based on various criteria.
• Booking form with date and time selection.
• Confirmation and notification of booking status.

Past Booking History Module:
Objective:
• Allow customers to view their past rental bookings.
Features:
• Display list of past bookings with details such as car model, rental dates, and booking status.
• Option to download or print booking history.

Responsive Design Module:
Objective:
• Ensure a seamless user experience across various devices and screen sizes.
Features:
• Responsive design for desktops, tablets, and mobiles.
• Adaptive layouts and touch-friendly interfaces.

ANALSYIS

This project involves a comprehensive analysis and enhancement of existing car rental platforms, aiming to streamline the process of discovering, booking, and managing rental vehicles. The primary objective is to provide users with easy access to detailed information about available cars, including their features, rental rates, and availability, addressing common challenges such as complex search processes, inefficient booking management, and the time-consuming nature of handling multiple rental options. By implementing a user-friendly interface, the system facilitates efficient data retrieval with minimal effort, incorporating features like high-quality images, detailed car specifications, availability status, and pricing information to enable customers to make well-informed decisions. The system is designed to improve the overall user experience with advanced search capabilities, allowing users to filter cars based on various criteria such as model, price range, and rental dates. The Car Rental System is divided into two primary user roles: Admin and Customer. For administrators, the system offers robust tools to manage the car inventory, handle booking requests, and maintain customer records, supporting CRUD operations (Create, Read, Update, Delete) for car listings to ensure the inventory is always current and accurately reflects the available options. Additionally, the booking request management feature allows administrators to efficiently review, approve, or reject booking requests, streamlining the rental process and enhancing customer satisfaction. For customers, the system offers an intuitive interface to browse available car listings effortlessly, submit booking requests for specific dates and times, and view past booking history. This project also focuses on ensuring a seamless user experience across various devices and screen sizes through responsive design, adapting the user interface to different devices, including desktops, tablets, and mobiles, making it convenient for users to access the system from anywhere. Overall, this initiative significantly enhances the car rental experience for both customers and administrators, providing a modernized alternative to traditional car rental systems by saving time, ensuring precision, and maintaining consistency in car listings, booking management, and customer records. The system is designed to boost operational efficiency by facilitating swift data retrieval, providing easy access to customer and car information, and streamlining the management of bookings. In summary, this project contributes to creating a more effective and user-centric car rental management system, benefiting the car rental industry by improving service quality and customer satisfaction.

SOFTWARE/HARDWARE REQUIREMENTS SPECIFICATIONS

System Configurations

The software requirement specification can produce at the culmination of the analysis task. The function and performance allocated to software as part of system engineering are refined by established a complete information description, a detailed functional description, a representation of system behaviour, and indication of performance and design constrain, appropriate validate criteria, and other information pertinent to requirements.

Software Requirements:

- Operating System: Windows 7,8,10, Linux, Mac OS-X
- Coding Language: HTML, CSS, JavaScript, Java
- Front-End: Visual Studio Code
- Back-End: Spring Tool Suite
- Data Base: MySQLWorkbench

Hardware Requirement:

- System: 1.8 GHz Dual-Core Intel Core i5
- HardDisk: 1TB
- Ram: 8GB

TECHNOLOGY

8.1 HTML

HyperText Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser. It defines the meaning and structure of web content. It is often assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript. Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for its appearance.

8.2 CSS

Cascading Style Sheets (CSS) is a stylesheet language used to describe the presentation of a document written in HTML or XML (including XML dialects such as SVG, MathML or XHTML). CSS describes how elements should be rendered on screen, on paper, in speech, or on other media.

8.3 JavaScript

JavaScript (JS) is a lightweight, object-oriented programming language widely used for scripting webpages. It is an interpreted language that enables dynamic interactivity on websites when applied to HTML documents. Introduced in 1995 for adding programs to webpages in the Netscape Navigator browser, JavaScript has since been adopted by all graphical web browsers. It allows users to build modern web applications that interact directly without requiring page reloads, enhancing interactivity and simplicity on traditional websites. Despite its name, JavaScript is not related to the Java programming language; the name was chosen during a time when Java was gaining popularity. Beyond web browsers, databases like CouchDB and MongoDB also use JavaScript as their scripting and query language.

8.4 Java

Java is a widely used object-oriented programming language and software platform that runs on billions of devices, including notebook computers, mobile devices, gaming consoles, medical devices and many others. The rules and syntax of Java are based on the C and C++ languages. One major advantage of developing software with Java is its portability. Once you have written code for a Java program on a notebook computer, it is very easy to move the code to a mobile device. When the language was invented in 1991 by James Gosling of Sun Microsystems (later acquired by Oracle), the primary goal was to be able to "write once, run anywhere.”

Features of Java:-

A list of the most important features of the Java language is given below.

- Simple
- Object-Oriented
- Portable
- Platform independent
- Secured
- Robust
- Architecture neutral
- Interpreted
- High Performance
- Multithreaded
- Dynamic

8.5 Spring Boot

Spring Boot is an open source Java-based framework used to create a micro Service. It is developed by Pivotal Team and is used to build stand-alone and production ready spring applications. This chapter will give you an introduction to Spring Boot and familiarizes you with its basic concepts.

Advantages of Spring Boot:-

Spring Boot offers the following advantages to its developers-

- Easy to understand and develop spring applications
- Increases productivity
- Reduces the development time

Features of Spring Boot:-

- Web Development
- SpringApplication
- Application events and listeners
- Admin features
- Externalized Configuration
- Properties Files
- YAML Support
- Type-safe Configuration
- Logging
- Security

Spring MVC

A Spring MVC is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection. Spring MVC provides an elegant solution to use MVC in spring framework by the help of DispatcherServlet. Here, DispatcherServlet is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views.

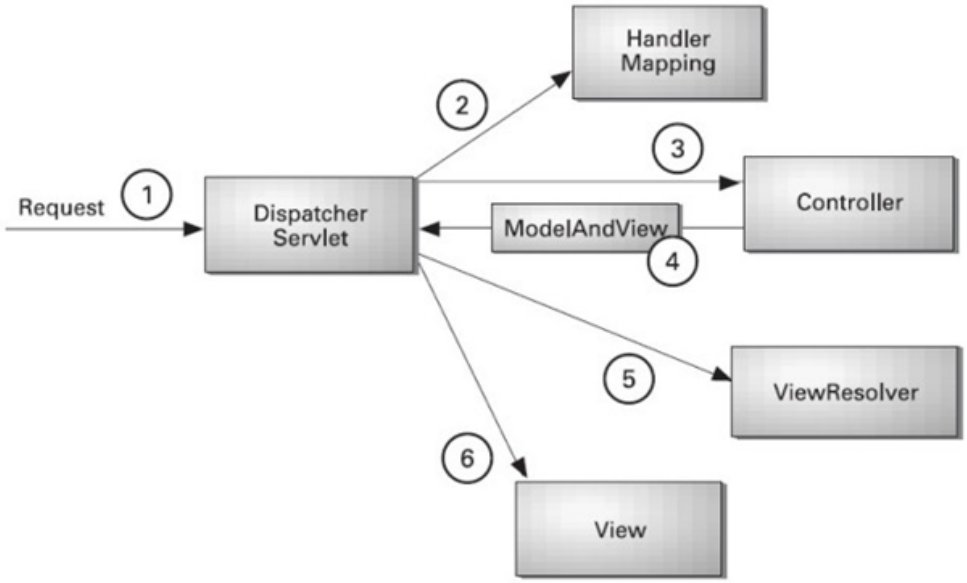
Model - A model contains the data of the application. A data can be a single object or a collection of objects.

Controller - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.

View - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page. Although spring also supports other view technologies such as Apache Velocity, Thymeleaf and FreeMarker.

Flow of Spring Web MVC:-

- As displayed in the figure, all the incoming request is intercepted by the DispatcherServlet that works as the front controller.
- The DispatcherServlet gets an entry of handler mapping from the XML file and forwards the request to the controller.
- The controller returns an object of ModelAndView.
- The DispatcherServlet checks the entry of view resolver in the XML file and invokes the specified view component.



Advantages of Spring MVC Framework:-

- Separate roles - The Spring MVC separates each role, where the model object, controller, command object, view resolver, DispatcherServlet, validator, etc. can be fulfilled by a specialized object.
- Light-weight - It uses light-weight servlet container to develop and deploy your application.
- Powerful Configuration - It provides a robust configuration for both framework and application classes that includes easy referencing across contexts, such as from web controllers to business objects and validators.
- Rapid development - The Spring MVC facilitates fast and parallel development.
- Reusable business code - Instead of creating new objects, it allows us to use the existing business objects.
- Easy to test - In Spring, generally we create JavaBeans classes that enable you to inject test data using the setter methods.
- Flexible Mapping - It provides the specific annotations that easily redirect the page.

8.6 SQL

SQL (Structured Query Language) is a standardized programming language designed for managing and manipulating relational databases. It is used for various operations such as querying, updating, inserting, and deleting data. SQL is essential for interacting with databases and is the foundation for database management systems (DBMS).

Features of SQL (Structured Query Language):-

- DDL (Data Definition Language): CREATE, ALTER, DROP commands for defining database schema.
- DML (Data Manipulation Language): SELECT, INSERT, UPDATE, DELETE commands for data manipulation.
- DCL (Data Control Language): GRANT, REVOKE commands for permissions.
- TCL (Transaction Control Language): COMMIT, ROLLBACK, SAVEPOINT for transaction management.
- Query Optimization: Efficient querying through indexes and optimization techniques.
- Join Operations: INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN to combine tables.

- Set Operations: UNION, INTERSECT, EXCEPT for combining query results.
- Functions and Aggregates: COUNT, SUM, AVG, MIN, MAX for calculations.
- Views: Virtual tables to simplify complex queries and enhance security.
- Stored Procedures and Triggers: Automated responses and precompiled SQL statements.

MySQL

MySQL is an open-source relational database management system (RDBMS) based on SQL. It is one of the most popular databases used for web applications and is a central component of the LAMP (Linux, Apache, MySQL, PHP/Python/Perl) stack. MySQL is known for its performance, reliability, and ease of use.

Features of MySQL:-

- Open Source: Free and community-supported.
- High Performance: Optimized for fast data processing.
- Scalability: Handles large databases and supports horizontal scaling.
- Security: User authentication, access privileges, and SSL encryption.
- Multiple Storage Engines: InnoDB, MyISAM, Memory for different use cases.
- ACID Compliance: Reliable transactions with Atomicity, Consistency, Isolation, Durability.
- Replication: Master-slave replication for backup and load balancing.
- Cross-Platform Support: Runs on Linux, Windows, macOS, Unix.
- Comprehensive Support: Extensive documentation and support options.
- Integration with Development Tools: Compatible with various development environments and tools.

CODING

controllers:-

AdminController.java

```
package com.project.controllers;
import com.project.dtos.CarDto;
import com.project.dtos.SearchCarDto;
import com.project.services.AdminService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.io.IOException;
import java.util.List;
@RestController
@CrossOrigin
@RequiredArgsConstructor
@RequestMapping("/api/admin")
public class AdminController {
   private final AdminService adminService;
   @PostMapping("/car")
   public ResponseEntity<?> postCar(@ModelAttribute CarDto carDto) {
      boolean success = adminService.postCar(carDto);
      if (success)
         return ResponseEntity.status(HttpStatus.CREATED).build();
      return ResponseEntity.status(HttpStatus.BAD_REQUEST).build();
   }
   @GetMapping("/cars")
   public ResponseEntity<List<CarDto>> getAllCars() {
      List<CarDto> carDtoList = adminService.getAllCars();
      return ResponseEntity.ok(carDtoList);
   }
   @DeleteMapping("/car/{carId}")
   public ResponseEntity<Void> deleteCar(@PathVariable Long carId) {
      adminService.deleteCar(carId);
      return ResponseEntity.noContent().build();
   }
   @GetMapping("/car/{carId}")
   public ResponseEntity<CarDto> getCarById(@PathVariable Long carId) {
      CarDto carDto = adminService.getCarById(carId);
      if (carDto != null) return ResponseEntity.ok(carDto);
      return ResponseEntity.notFound().build();
   }
   @PutMapping("/car/{carId}")
   public ResponseEntity<?> updateCar(@PathVariable Long carId, @ModelAttribute CarDto
carDto) throws IOException {
      try {
         boolean success = adminService.updateCar(carId, carDto);
```

```
        if (success) return ResponseEntity.ok().build();
        return ResponseEntity.notFound().build();
      } catch (Exception e) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(e);
      } }
  @GetMapping("/car/bookings")
  public ResponseEntity<?> getBookings() {
    return ResponseEntity.ok(adminService.getBookings());
  }
  @GetMapping("/car/booking/{bookingId}/{status}")
  public ResponseEntity<?> changeBookingStatus(@PathVariable Long bookingId,
@PathVariable String status) {
    boolean success = adminService.changeBookingStatus(bookingId, status);
    if (success) return ResponseEntity.ok().build();
    return ResponseEntity.notFound().build();
  }
  @PostMapping("/car/search")
  public ResponseEntity<?> searchCar(@RequestBody SearchCarDto searchCarDto) {
    return ResponseEntity.ok(adminService.searchCar(searchCarDto));
  }
}
```

AuthController.java

```
package com.project.controllers;
import com.project.dtos.AuthenticationRequest;
import com.project.dtos.AuthenticationResponse;
import com.project.dtos.SignupRequest;
import com.project.dtos.UserDto;
import com.project.entities.Users;
import com.project.repositories.UserRepository;
import com.project.services.AuthService;
import com.project.services.UserService;
import com.project.utils.JwtUtil;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.DisabledException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.Optional;
```

```
@RestController
@CrossOrigin
@RequiredArgsConstructor
@RequestMapping("/api/auth")
public class AuthController {
   private final AuthenticationManager authenticationManager;
   private final UserService userService;
   private final JwtUtil jwtUtil;
   private final UserRepository userRepository;
   private final AuthService authService;
   @PostMapping("/signup")
   public ResponseEntity<?> createCustomer(@RequestBody SignupRequest signupRequest) {
      if (authService.hasCustomerWithEmail(signupRequest.getEmail()))
         return ResponseEntity.status(HttpStatus.NOT_ACCEPTABLE).body("Email already exist.
Try again with another email");
      UserDto createdUserDto = authService.createCustomer(signupRequest);
      if (createdUserDto == null) return
ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Bad Request!");
      return ResponseEntity.status(HttpStatus.CREATED).body(createdUserDto);
   }
   @PostMapping("/login")
   public AuthenticationResponse createAuthenticationToken(@RequestBody
AuthenticationRequest authenticationRequest) throws
         BadCredentialsException,
         DisabledException,
         UsernameNotFoundException {
      try {
         authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(authenticationRequest.getEmail(),
authenticationRequest.getPassword()));
      } catch (BadCredentialsException e) {
         throw new BadCredentialsException("Incorrect username or password.");
      }
      final UserDetails userDetails =
userService.userDetailsService().loadUserByUsername(authenticationRequest.getEmail());
      Optional<Users> optionalUser =
userRepository.findFirstByEmail(userDetails.getUsername());
      final String jwt = jwtUtil.generateToken(userDetails);
      AuthenticationResponse authenticationResponse = new AuthenticationResponse();
      if (optionalUser.isPresent()) {
         authenticationResponse.setJwt(jwt);
         authenticationResponse.setUserId(optionalUser.get().getId());
         authenticationResponse.setUserRole(optionalUser.get().getUserRole());
      }
      return authenticationResponse;
   }
}
```

CustomerController.java

```
package com.project.controllers;
import com.project.dtos.BookACarDto;
import com.project.dtos.CarDto;
import com.project.dtos.SearchCarDto;
import com.project.services.CustomerService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
@CrossOrigin
@RequestMapping("/api/customer")
@RequiredArgsConstructor
public class CustomerController {
   private final CustomerService customerService;
   @GetMapping("/cars")
   public ResponseEntity<List<CarDto>> getAllCars() {
      List<CarDto> carDtoList = customerService.getAllCars();
      return ResponseEntity.ok(carDtoList);
   }
   @GetMapping("/car/{carId}")
   public ResponseEntity<CarDto> getCarById(@PathVariable Long carId) {
      CarDto carDto = customerService.getCarById(carId);
      if (carDto != null) return ResponseEntity.ok(carDto);
      return ResponseEntity.notFound().build();
   }
   @PostMapping("/car/book/{carId}")
   public ResponseEntity<?> bookACar(@PathVariable Long carId, @RequestBody BookACarDto
bookACarDto) {
      boolean success = customerService.bookACar(carId, bookACarDto);
      System.out.print(bookACarDto);
      if (success) return ResponseEntity.ok().build();
      return ResponseEntity.status(HttpStatus.BAD_REQUEST).build();
   }
   @GetMapping("/car/bookings/{userId}")
   public ResponseEntity<?> getBookingsByUserId(@PathVariable Long userId) {
      return ResponseEntity.ok(customerService.getBookingsByUserId(userId));
   }
   @PostMapping("/car/search")
   public ResponseEntity<?> searchCar(@RequestBody SearchCarDto searchCarDto) {
      return ResponseEntity.ok(customerService.searchCar(searchCarDto));
   }
}
```

dtos:-

AuthenticationRequest.java

```
package com.project.dtos;
import lombok.Data;
@Data
public class AuthenticationRequest {
    private String email;
    private String password;
}
```

AuthenticationResponse.java

```
package com.project.dtos;
import com.project.enums.UserRole;
import lombok.Data;
@Data
public class AuthenticationResponse {
    private String jwt;
    private UserRole userRole;
    private Long userId;
}
```

BookACarDto.java

```
package com.project.dtos;
import lombok.Data;
import java.util.Date;
import com.project.enums.BookCarStatus;
@Data
public class BookACarDto {
    private Long id;
    private Date fromDate;
    private Date toDate;
    private Long days;
    private Long amount;
    private BookCarStatus bookCarStatus;
    private Long userId;
    private String email;
    private String username;
    private String employee;
    private String dealership;
}
```

CarDealershipDto.java

```
package com.project.dtos;
import lombok.Data;
@Data
public class CarDealershipDto {
    private Long id;
```

```
    private String employee;
    private String dealership;
}
```

CarDto.java

```
package com.project.dtos;
import org.springframework.web.multipart.MultipartFile;
import lombok.Data;
@Data
public class CarDto {
    private Long id;
    private String name;
    private String color;
    private String transmission;
    private String brand;
    private String type;
    private String modelYear;
    private Integer price;
    private String employee;
    private String dealership;
    private String description;
    private MultipartFile image;
    private byte[] returnedImage;
}
```

CarDtoList.java

```
package com.project.dtos;
import lombok.Data;
import java.util.List;
@Data
public class CarDtoList {
    private List<CarDto> carDtoList;
}
```

CarImageDto.java

```
package com.project.dtos;
import org.springframework.web.multipart.MultipartFile;
import lombok.Data;
@Data
public class CarImageDto {
    private Long id;
    private MultipartFile image;
    private byte[] returnedImage;
}
```

CarTypeDto.java

```
package com.project.dtos;
import lombok.Data;
```

```
@Data
public class CarTypeDto {
    private Long id;
    private String description;
    private Integer price;
}
```

SearchCarDto.java

```
package com.project.dtos;
import lombok.Data;
@Data
public class SearchCarDto {
    private String brand;
    private String type;
    private String transmission;
    private String color;
}
```

SignupRequest.java

```
package com.project.dtos;
import lombok.Data;
@Data
public class SignupRequest {
    private String email;
    private String name;
    private String password;
}
```

UserDto.java

```
package com.project.dtos;
import com.project.enums.UserRole;
import lombok.Data;
@Data
public class UserDto {
    private Long id;
    private String name;
    private String email;
    private String password;
    private UserRole userRole;
}
```

entities:-

BookACar.java

```
package com.project.entities;
import com.project.dtos.BookACarDto;
import com.project.enums.BookCarStatus;
import com.fasterxml.jackson.annotation.JsonIgnore;
```

```
import jakarta.persistence.*;
import lombok.Data;
import org.hibernate.annotations.OnDelete;
import org.hibernate.annotations.OnDeleteAction;
import java.util.Date;
@Entity
@Data
public class BookACar {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date fromDate;
    private Date toDate;
    private Long days;
    private Long amount;
    private String employee;
    private String dealership;
    private BookCarStatus bookCarStatus;
    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "user_id", nullable = false)
    @OnDelete(action = OnDeleteAction.CASCADE)
    @JsonIgnore
    private Users user;
    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "car_id", nullable = false)
    @OnDelete(action = OnDeleteAction.CASCADE)
    @JsonIgnore
    private Car car;
    public BookACarDto getBookACarDto() {
        BookACarDto bookACarDto = new BookACarDto();
        bookACarDto.setId(id);
        bookACarDto.setFromDate(fromDate);
        bookACarDto.setToDate(toDate);
        bookACarDto.setDays(days);
        bookACarDto.setAmount(amount);
        bookACarDto.setBookCarStatus(bookCarStatus);
        bookACarDto.setEmail(user.getEmail());
        bookACarDto.setUsername(user.getName());
        bookACarDto.setUserId(user.getId());
        bookACarDto.setEmployee(employee);
        bookACarDto.setDealership(dealership);
        return bookACarDto;
    }
}
```

Car.java

```
package com.project.entities;
import jakarta.persistence.*;
```

```
import lombok.Data;
import com.project.dtos.CarDto;
@Entity
@Data
@Table(name = "cars")
public class Car {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String color;
    private String transmission;
    private String brand;
    private String type;
    private String modelYear;
    private String description;
    private Integer price;
    private String employee;
    private String dealership;
    @Column(columnDefinition = "longblob")
    private byte[] image;
    public CarDto getCarDto() {
        CarDto carDto = new CarDto();
        carDto.setId(id);
        carDto.setName(name);
        carDto.setDescription(description);
        carDto.setColor(color);
        carDto.setType(type);
        carDto.setPrice(price);
        carDto.setTransmission(transmission);
        carDto.setModelYear(modelYear);
        carDto.setBrand(brand);
        carDto.setReturnedImage(image);
        carDto.setEmployee(employee);
        carDto.setDealership(dealership);
        return carDto;
    }
}
```

CarImage.java

```
package com.project.entities;
import com.project.dtos.CarImageDto;
import jakarta.persistence.*;
import lombok.Data;
@Entity
@Data
@Table(name="cars_image")
public class CarImage {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(columnDefinition = "longblob")
    private byte[] image;
    public CarImageDto getCarImage() {
        CarImageDto carImage = new CarImageDto();
        carImage.setId(id);
        carImage.setReturnedImage(image);
        return carImage;
    }
}
```

CarType.java

```
package com.project.entities;
import jakarta.persistence.*;
import lombok.Data;
import com.project.dtos.CarTypeDto;
@Entity
@Data
@Table(name = "car_types")
public class CarType {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String description;
    private Integer price;
    public CarTypeDto getCarType() {
        CarTypeDto carType = new CarTypeDto();
        carType.setId(id);
        carType.setDescription(description);
        carType.setPrice(price);
        return carType;
    }
}
```

Dealership.java

```
package com.project.entities;
import com.project.dtos.CarDealershipDto;
import jakarta.persistence.*;
import lombok.Data;
@Entity
@Data
@Table(name = "dealerships")
public class Dealership {
    @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;
```

```
    private String employee;
    private String dealership;
    public CarDealershipDto getCarDealership() {
        CarDealershipDto carDealership = new CarDealershipDto();
        carDealership.setId(id);
        carDealership.setEmployee(employee);
        carDealership.setDealership(dealership);
        return carDealership;
    }
}
```

Users.java

```
package com.project.entities;
import jakarta.persistence.*;
import lombok.Data;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import com.project.enums.UserRole;
import java.util.Collection;
import java.util.List;
@Entity
@Data
@Table(name = "users")
public class Users implements UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;
    private String password;
    private UserRole userRole;
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return List.of(new SimpleGrantedAuthority(userRole.name()));
    }
    @Override
    public String getUsername() {
        return email;
    }
    @Override
    public boolean isAccountNonExpired() {
        return true;
    }
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }
    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }
    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

repositories:-

BookACarRepository.java

```
package com.project.repositories;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.project.entities.BookACar;
import java.util.List;
@Repository
public interface BookACarRepository extends JpaRepository<BookACar, Long> {
   List<BookACar> findAllByUserId(Long userId);
}
```

CarRepository.java

```
package com.project.repositories;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.project.entities.Car;
@Repository
public interface CarRepository extends JpaRepository<Car, Long> {
}
```

UserRepository.java

```
package com.project.repositories;
import com.project.entities.Users;
import com.project.enums.UserRole;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.Optional;
@Repository
public interface UserRepository extends JpaRepository<Users, Long> {
   Optional<Users> findFirstByEmail(String email);
   Users findByUserRole(UserRole userRole);
}
```

services:-

AdminService.java

```
package com.project.services;
import java.io.IOException;
import java.util.List;
import java.util.Objects;
import java.util.Optional;
import java.util.stream.Collectors;
import org.springframework.data.domain.Example;
import org.springframework.data.domain.ExampleMatcher;
import org.springframework.stereotype.Service;
import com.project.dtos.BookACarDto;
import com.project.dtos.CarDto;
import com.project.dtos.CarDtoList;
import com.project.dtos.SearchCarDto;
```

```
import com.project.entities.BookACar;
import com.project.entities.Car;
import com.project.enums.BookCarStatus;
import com.project.repositories.BookACarRepository;
import com.project.repositories.CarRepository;
import lombok.RequiredArgsConstructor;
@Service
@RequiredArgsConstructor
public class AdminService {
    private final CarRepository carRepository;
    private final BookACarRepository bookACarRepository;
    public boolean postCar(CarDto carDto) {
        try {
            Car car = new Car();
            car.setName(carDto.getName());
            car.setBrand(carDto.getBrand());
            car.setColor(carDto.getColor());
            car.setPrice(carDto.getPrice());
            car.setType(carDto.getType());
            car.setDescription(carDto.getDescription());
            car.setModelYear(carDto.getModelYear());
            car.setTransmission(carDto.getTransmission());
            car.setImage(carDto.getImage().getBytes());
            car.setEmployee(carDto.getEmployee());
            car.setDealership(carDto.getDealership());
            carRepository.save(car);
            return true;
        } catch (Exception e) {
            return false;
        }
    }
    public List<CarDto> getAllCars() {
        return carRepository.findAll().stream().map(Car::getCarDto).collect(Collectors.toList());
    }
    public void deleteCar(Long carId) {
        carRepository.deleteById(carId);
    }
    public CarDto getCarById(Long cardId) {
        Optional<Car> optionalCar = carRepository.findById(cardId);
        return optionalCar.map(Car::getCarDto).orElse(null);
    }
    public boolean updateCar(Long carId, CarDto carDto) throws IOException {
        Optional<Car> optionalCar = carRepository.findById(carId);
        if (optionalCar.isPresent()) {
            Car existingCar = optionalCar.get();
            existingCar.setName(carDto.getName());
            existingCar.setBrand(carDto.getBrand());
            existingCar.setColor(carDto.getColor());
```

```
            existingCar.setPrice(carDto.getPrice());
            existingCar.setType(carDto.getType());
            existingCar.setDescription(carDto.getDescription());
            existingCar.setModelYear(carDto.getModelYear());
            existingCar.setTransmission(carDto.getTransmission());
            existingCar.setEmployee(carDto.getEmployee());
            existingCar.setDealership(carDto.getDealership());
            if (carDto.getImage() != null)
                existingCar.setImage(carDto.getImage().getBytes());
            carRepository.save(existingCar);
            return true;
        }
        return false;
    }
    public List<BookACarDto> getBookings() {
        return
bookACarRepository.findAll().stream().map(BookACar::getBookACarDto).collect(Collectors.toLis
t());
    }
    public boolean changeBookingStatus(Long bookingId, String status) {
        Optional<BookACar> optionalBookACar = bookACarRepository.findById(bookingId);
        if (optionalBookACar.isPresent()) {
            BookACar existingBookACar = optionalBookACar.get();
            if (Objects.equals(status, "Approve"))
                existingBookACar.setBookCarStatus(BookCarStatus.APPROVED);
            else
                existingBookACar.setBookCarStatus(BookCarStatus.REJECTED);
            bookACarRepository.save(existingBookACar);
            return true;
        }
        return false;
    }
    public CarDtoList searchCar(SearchCarDto searchCarDto) {
        Car car = new Car();
        car.setBrand(searchCarDto.getBrand());
        car.setType(searchCarDto.getType());
        car.setTransmission(searchCarDto.getTransmission());
        car.setColor(searchCarDto.getColor());
        ExampleMatcher projectMatcher = ExampleMatcher.matchingAll()
                .withMatcher("brand",
ExampleMatcher.GenericPropertyMatchers.contains().ignoreCase())
                .withMatcher("type", ExampleMatcher.GenericPropertyMatchers.contains().ignoreCase())
                .withMatcher("transmission",
ExampleMatcher.GenericPropertyMatchers.contains().ignoreCase())
                .withMatcher("color",
ExampleMatcher.GenericPropertyMatchers.contains().ignoreCase());
        Example<Car> carExample = Example.of(car,projectMatcher);
        List<Car> cars = carRepository.findAll(carExample);
```

```
        CarDtoList carDtoList = new CarDtoList();
        carDtoList.setCarDtoList(cars.stream().map(Car::getCarDto).collect(Collectors.toList()));
        return carDtoList;
    }
}
```

AuthService.java

```
package com.project.services;
import com.project.dtos.SignupRequest;
import com.project.dtos.UserDto;
import com.project.entities.Users;
import com.project.enums.UserRole;
import com.project.repositories.UserRepository;
import jakarta.annotation.PostConstruct;
import lombok.RequiredArgsConstructor;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;
@Service
@RequiredArgsConstructor
public class AuthService {
    private final UserRepository userRepository;
    @PostConstruct
    public void createAdminAccount(){
        Users adminAccount = userRepository.findByUserRole(UserRole.ADMIN);
        if (adminAccount == null){
            Users newAdminAccount = new Users();
            newAdminAccount.setName("Admin");
            newAdminAccount.setUserRole(UserRole.ADMIN);
            newAdminAccount.setEmail("admin@test.com");
            newAdminAccount.setPassword(new BCryptPasswordEncoder().encode("admin"));
            userRepository.save(newAdminAccount);
        }
    }
    public UserDto createCustomer(SignupRequest signupRequest) {
        Users user = new Users();
        user.setEmail(signupRequest.getEmail());
        user.setName(signupRequest.getName());
        user.setPassword(new BCryptPasswordEncoder().encode(signupRequest.getPassword()));
        user.setUserRole(UserRole.CUSTOMER);
        Users createdCustomer = userRepository.save(user);
        UserDto cretaedUserDto = new UserDto();
        cretaedUserDto.setId(createdCustomer.getId());
        cretaedUserDto.setName(createdCustomer.getName());
        cretaedUserDto.setEmail(createdCustomer.getEmail());
        cretaedUserDto.setUserRole(createdCustomer.getUserRole());
        return cretaedUserDto;
    }
```

```
    public boolean hasCustomerWithEmail(String email) {
        return userRepository.findFirstByEmail(email).isPresent();
    }
}
```

CustomerService.java

```
package com.project.services;
import com.project.dtos.BookACarDto;
import com.project.dtos.CarDto;
import com.project.dtos.CarDtoList;
import com.project.dtos.SearchCarDto;
import com.project.entities.BookACar;
import com.project.entities.Car;
import com.project.entities.Users;
import com.project.enums.BookCarStatus;
import com.project.repositories.BookACarRepository;
import com.project.repositories.CarRepository;
import com.project.repositories.UserRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Example;
import org.springframework.data.domain.ExampleMatcher;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;
import java.util.concurrent.TimeUnit;
import java.util.stream.Collectors;
@Service
@RequiredArgsConstructor
public class CustomerService {
    private final CarRepository carRepository;
    private final UserRepository userRepository;
    private final BookACarRepository bookACarRepository;
    public List<CarDto> getAllCars() {
        return carRepository.findAll().stream().map(Car::getCarDto).collect(Collectors.toList());
    }
    public CarDto getCarById(Long carId) {
        Optional<Car> optionalCar = carRepository.findById(carId);
        return optionalCar.map(Car::getCarDto).orElse(null);
    }
    public boolean bookACar(Long carId, BookACarDto bookACarDto) {
        Optional<Users> optionalUser = userRepository.findById(bookACarDto.getUserId());
        Optional<Car> optionalCar = carRepository.findById(carId);
        System.out.print(optionalCar);
        if (optionalCar.isPresent() && optionalUser.isPresent()) {
            BookACar bookACar = new BookACar();
            long diffInMilliseconds = bookACarDto.getToDate().getTime() -
bookACarDto.getFromDate().getTime();
            long days = TimeUnit.MILLISECONDS.toDays(diffInMilliseconds);
```

```
            bookACar.setDays(days);
            bookACar.setUser(optionalUser.get());
            bookACar.setCar(optionalCar.get());
            bookACar.setAmount(optionalCar.get().getPrice() * days);
            bookACar.setFromDate(bookACarDto.getFromDate());
            bookACar.setToDate(bookACarDto.getToDate());
            bookACar.setBookCarStatus(BookCarStatus.PENDING);
            bookACar.setEmployee(optionalCar.get().getEmployee());
            bookACar.setDealership(optionalCar.get().getDealership());
            bookACarRepository.save(bookACar);
            return true;
        }
        return false;
    }
    public List<BookACarDto> getBookingsByUserId(Long userId) {
        return
bookACarRepository.findAllByUserId(userId).stream().map(BookACar::getBookACarDto).collect(
Collectors.toList());
    }
    public CarDtoList searchCar(SearchCarDto searchCarDto) {
        Car car = new Car();
        car.setBrand(searchCarDto.getBrand());
        car.setType(searchCarDto.getType());
        car.setTransmission(searchCarDto.getTransmission());
        car.setColor(searchCarDto.getColor());
        ExampleMatcher projectMatcher = ExampleMatcher.matchingAll()
                .withMatcher("brand",
ExampleMatcher.GenericPropertyMatchers.contains().ignoreCase())
                .withMatcher("type", ExampleMatcher.GenericPropertyMatchers.contains().ignoreCase())
                .withMatcher("transmission",
ExampleMatcher.GenericPropertyMatchers.contains().ignoreCase())
                .withMatcher("color",
ExampleMatcher.GenericPropertyMatchers.contains().ignoreCase());
        Example<Car> carExample = Example.of(car,projectMatcher);
        List<Car> cars = carRepository.findAll(carExample);
        CarDtoList carDtoList = new CarDtoList();
        carDtoList.setCarDtoList(cars.stream().map(Car::getCarDto).collect(Collectors.toList()));
        return carDtoList;
    }
}
```

UserService.java

```
package com.project.services;
import lombok.RequiredArgsConstructor;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
```

```
import com.project.repositories.UserRepository;
@Service
@RequiredArgsConstructor
public class UserService {
   private final UserRepository userRepository;
   public UserDetailsService userDetailsService() {
      return new UserDetailsService() {
         public UserDetails loadUserByUsername(String username) {
            return userRepository.findFirstByEmail(username)
                  .orElseThrow(() -> new UsernameNotFoundException("User not found"));
         }
      };
   }
}
```

CarRentalApplication.java

```
package com.project;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class CarRentalApplication {
   public static void main(String[] args) {
      SpringApplication.run(CarRentalApplication.class, args);
   }
}
```

SCREENSHOTS

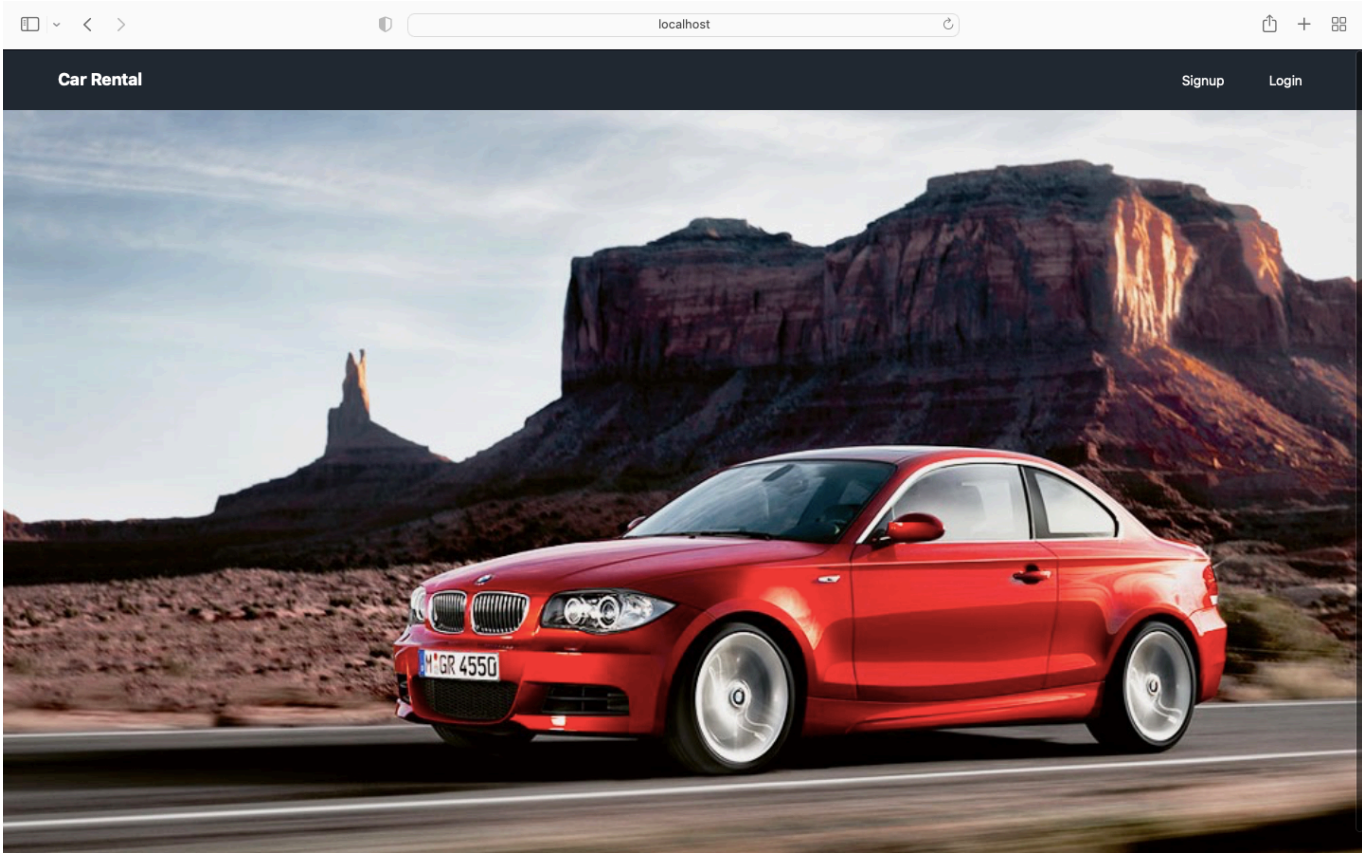


FIG 1: HOME PAGE

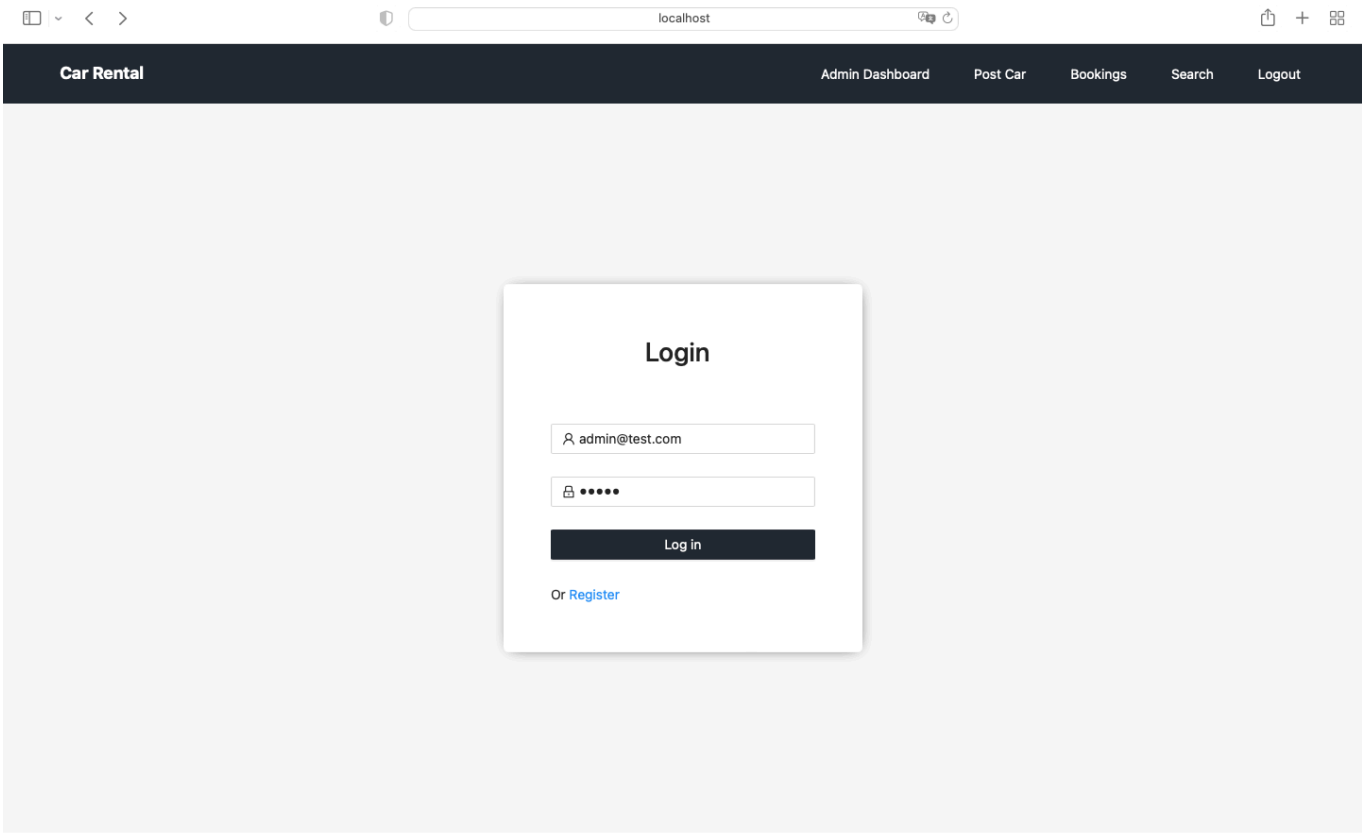


FIG 2: ADMIN LOGIN PAGE

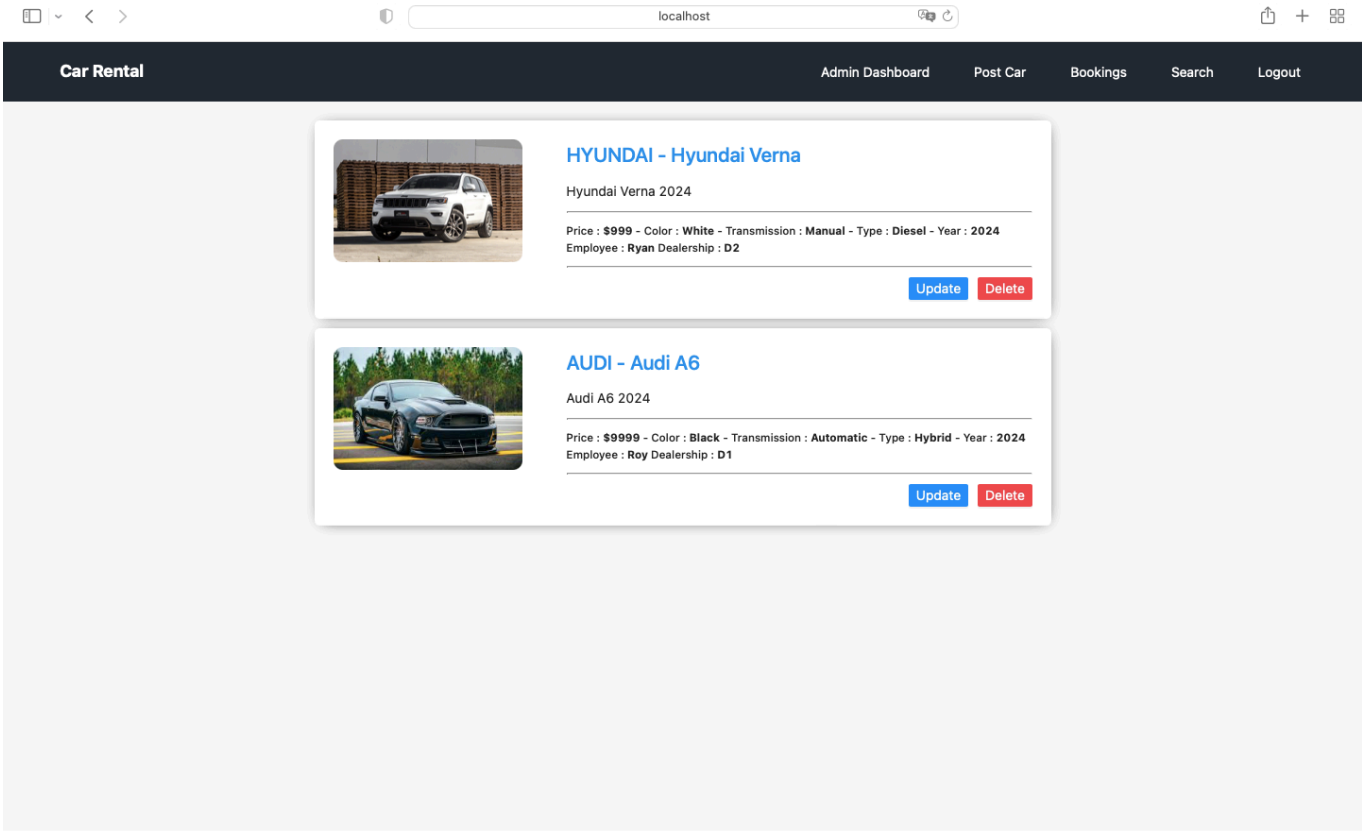


FIG3: ADMIN DASHBOARD

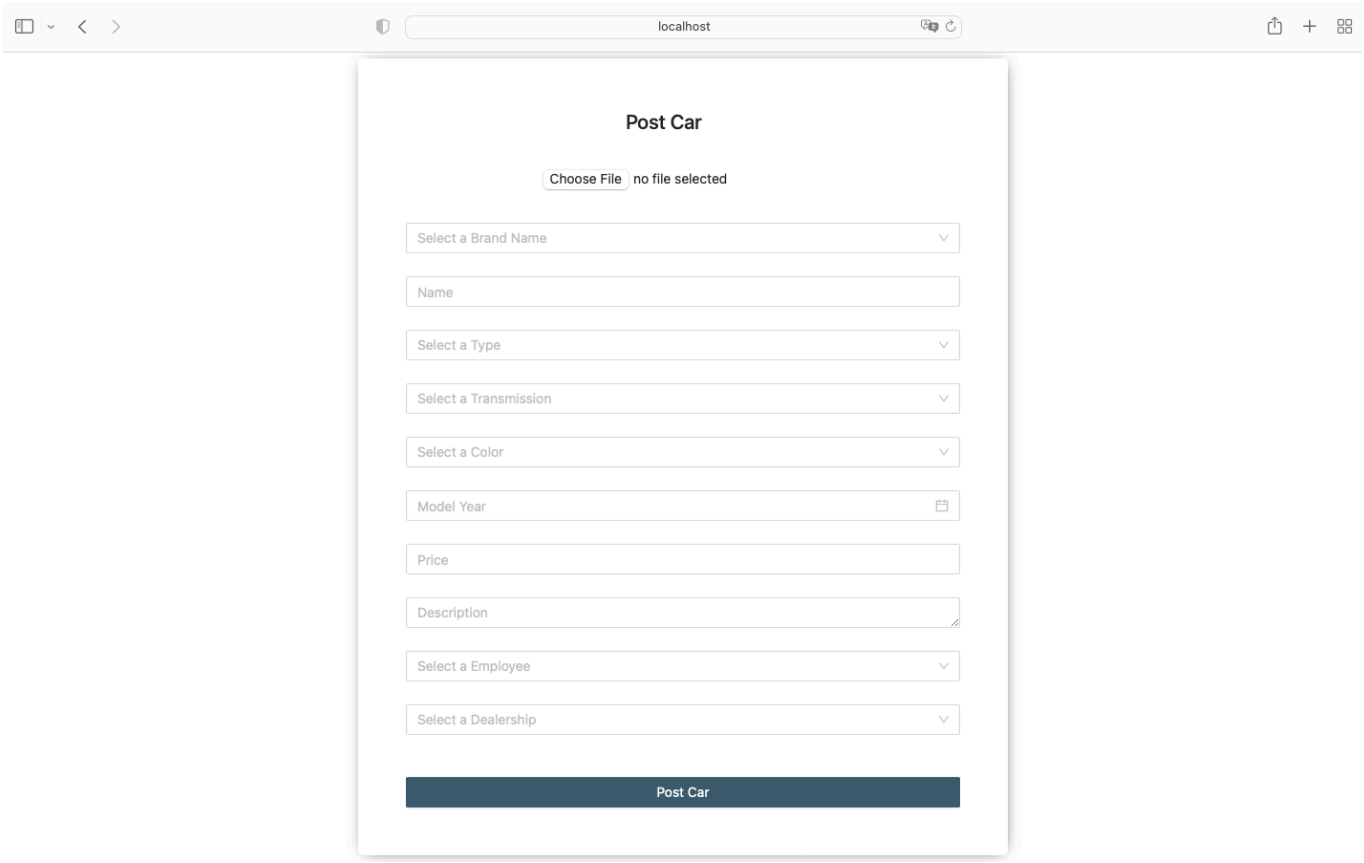


FIG 4: POST CAR PAGE

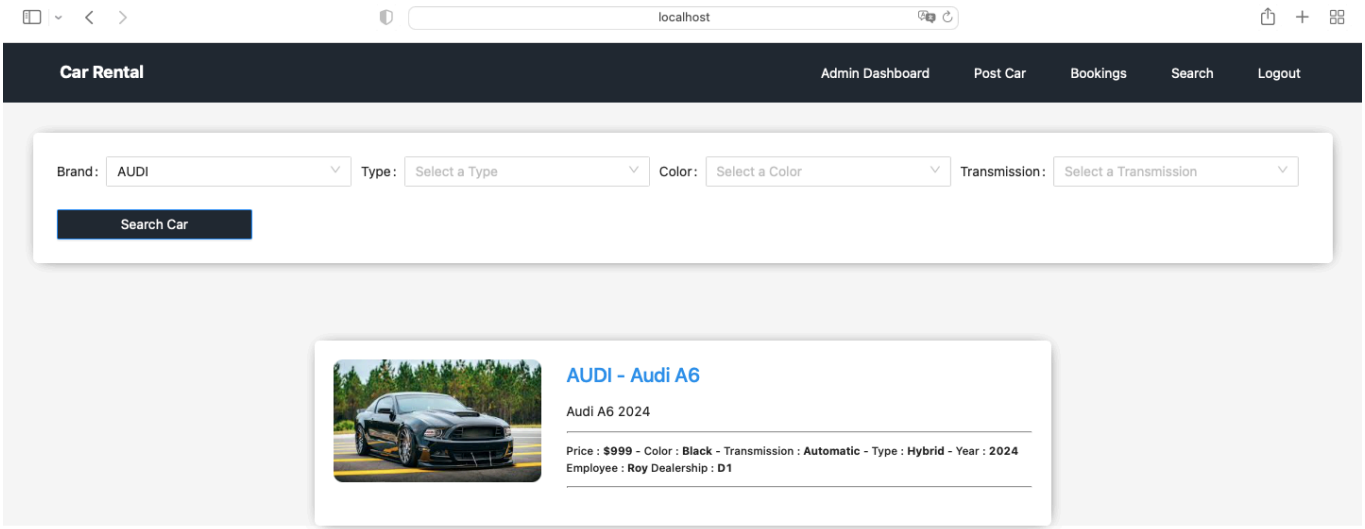


FIG 5: ADMIN SEARCH PAGE

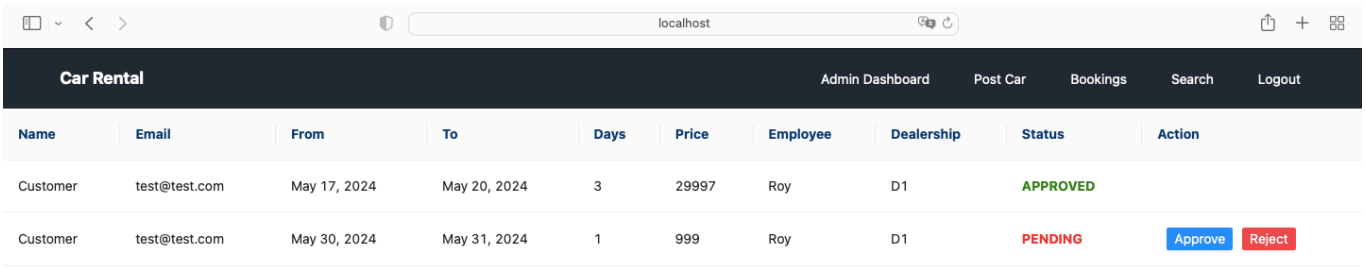


FIG 6.1 BOOKING REQUEST MANAGEMENT

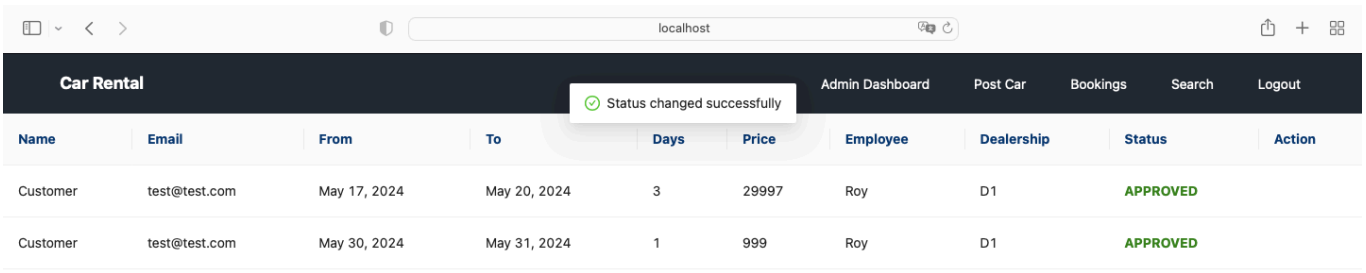


FIG 6.2 BOOKING REQUEST MANAGEMENT

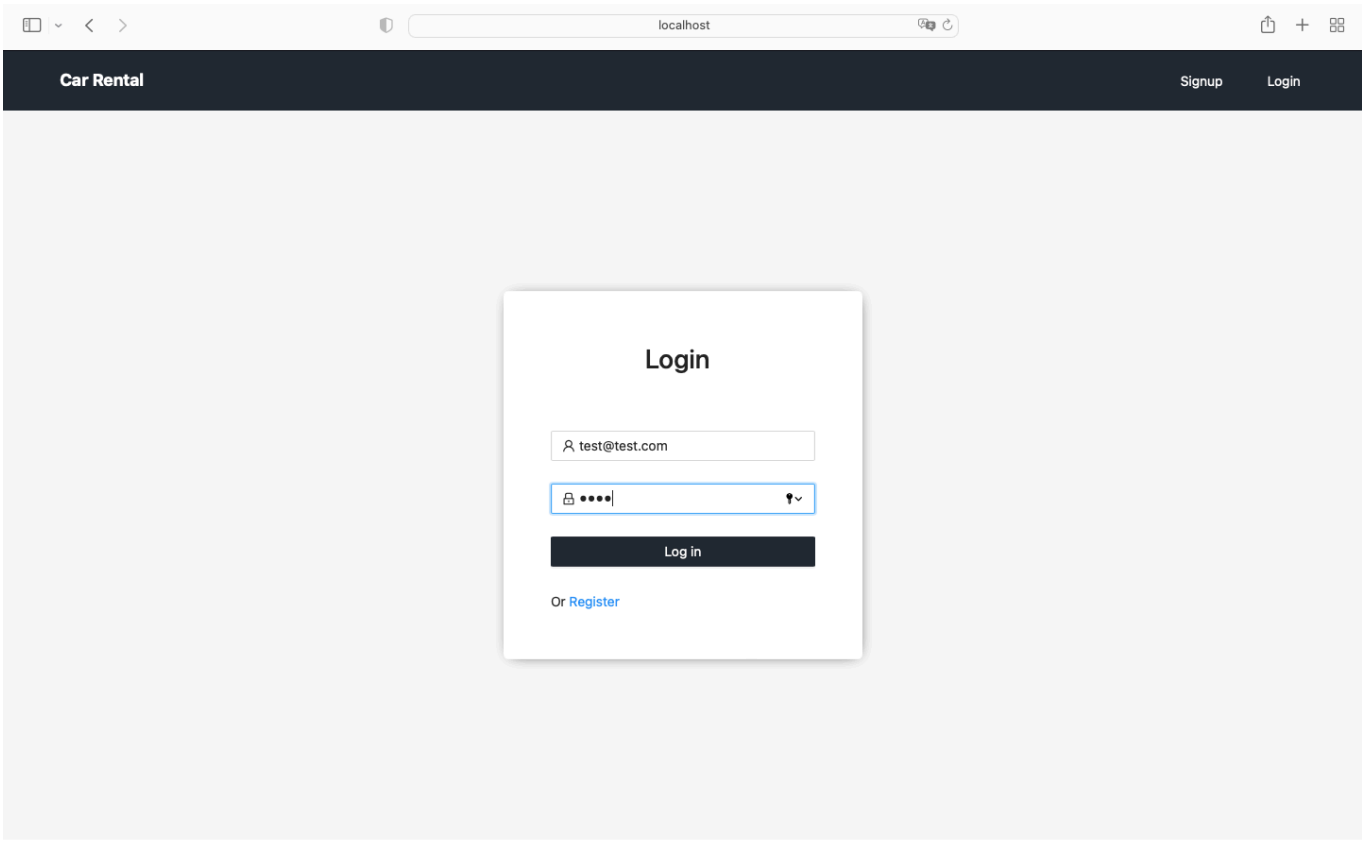


FIG 7: USER LOGIN PAGE

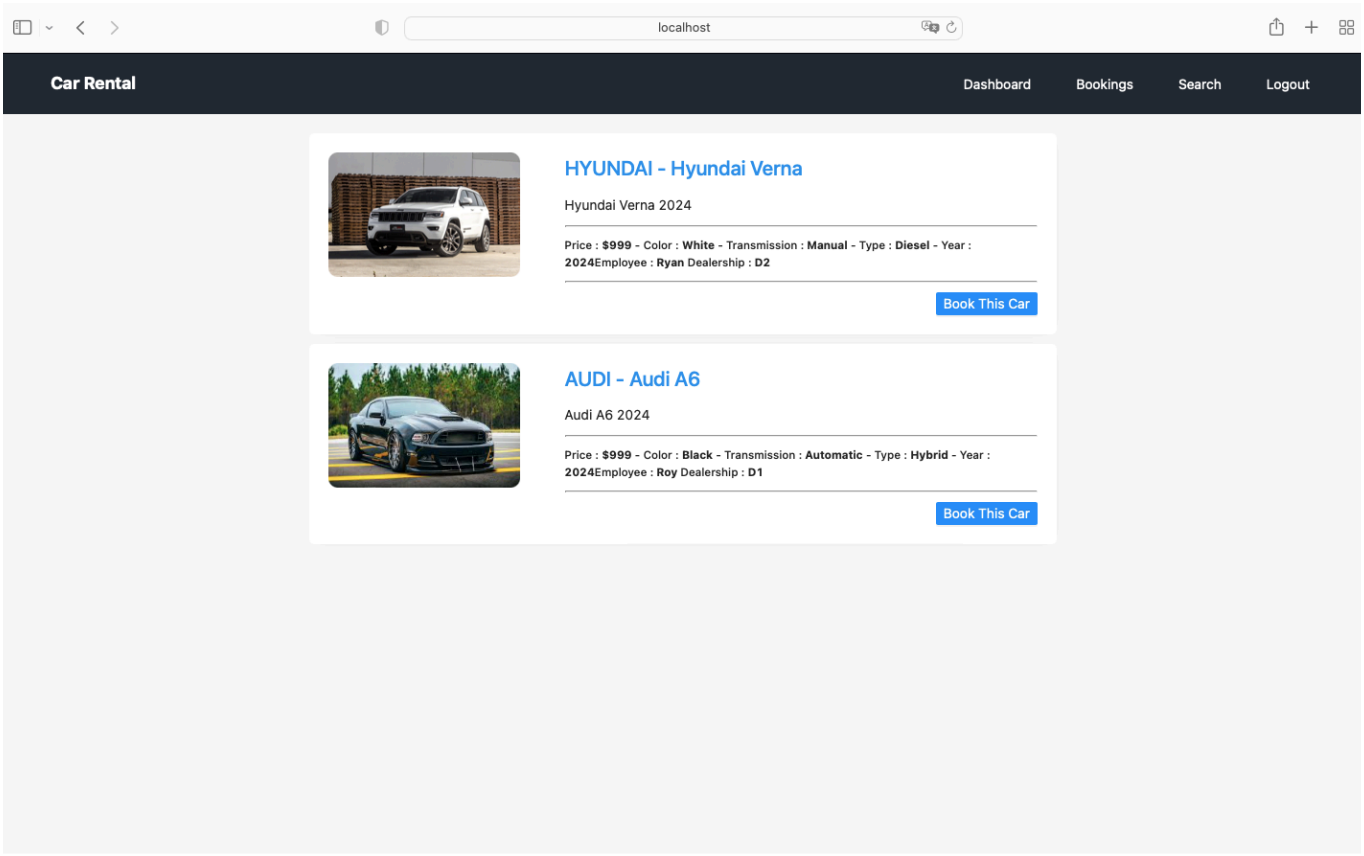


FIG 8: USER DASHBOARD

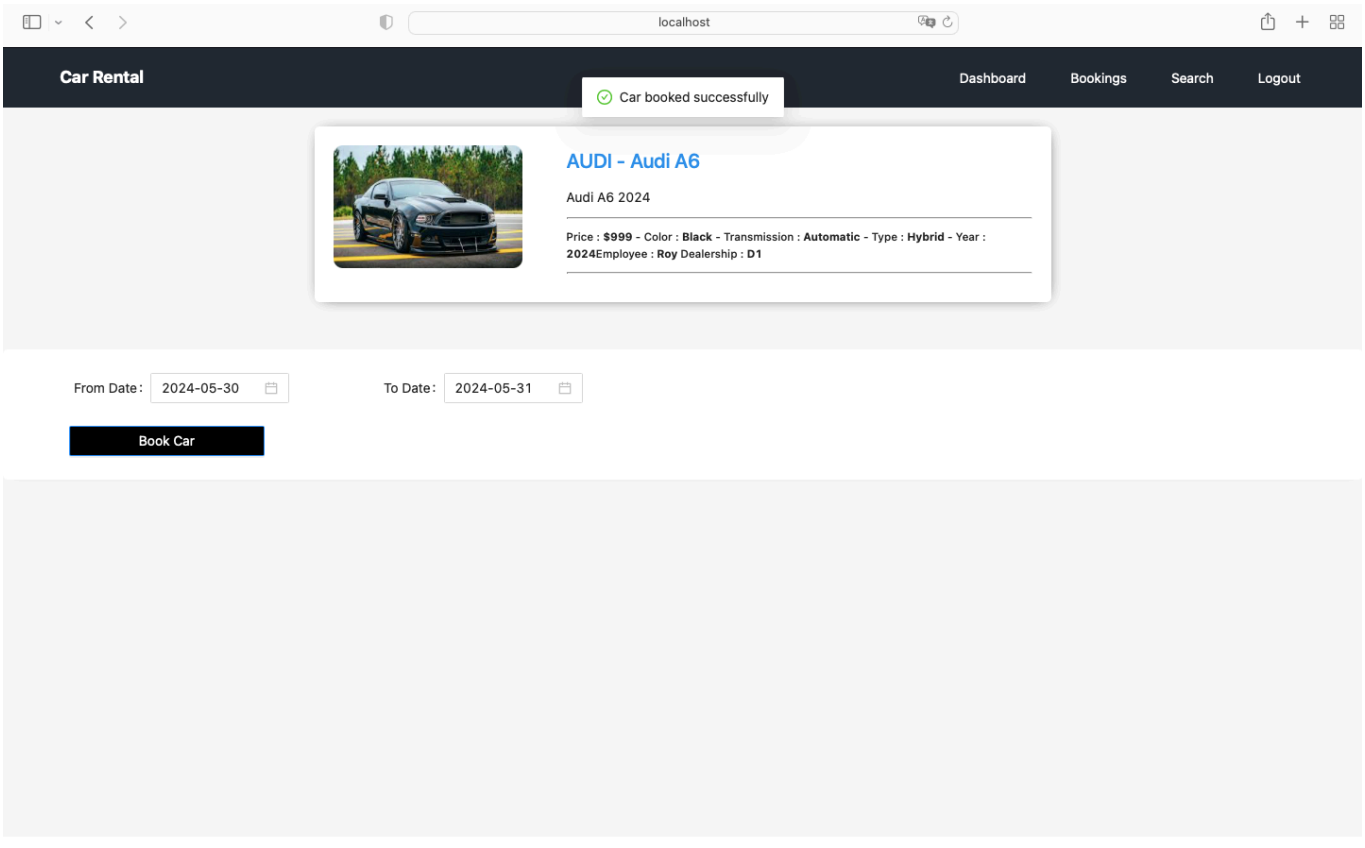


FIG 9: BOOKING PAGE

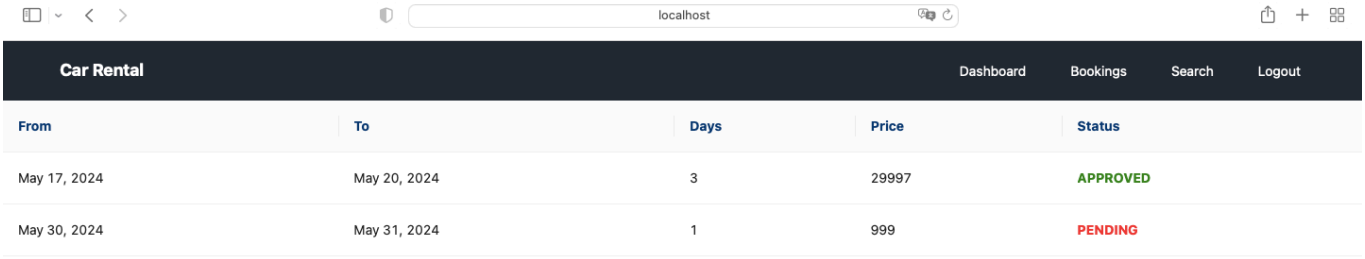


FIG 10: BOOKING HISTORY

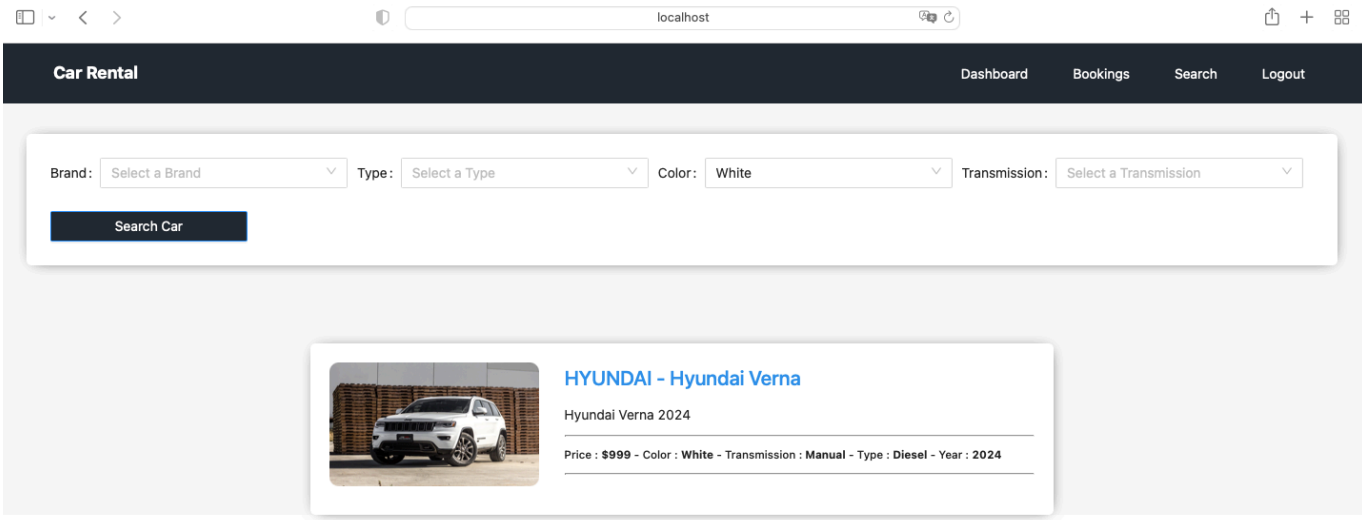


FIG 11: USER SEARCH PAGE

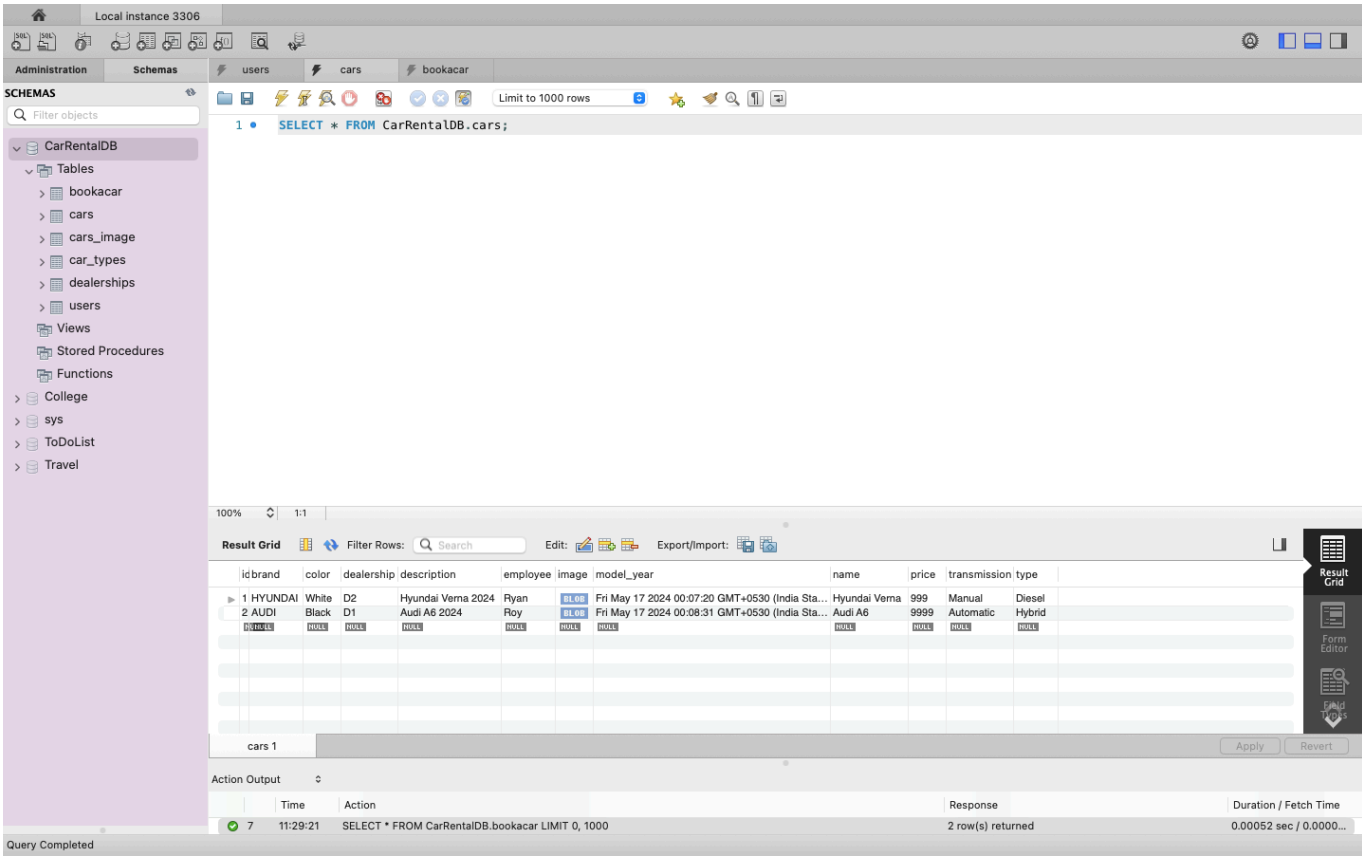


FIG 12: MYSQL WORKBENCH - CAR DATABASE

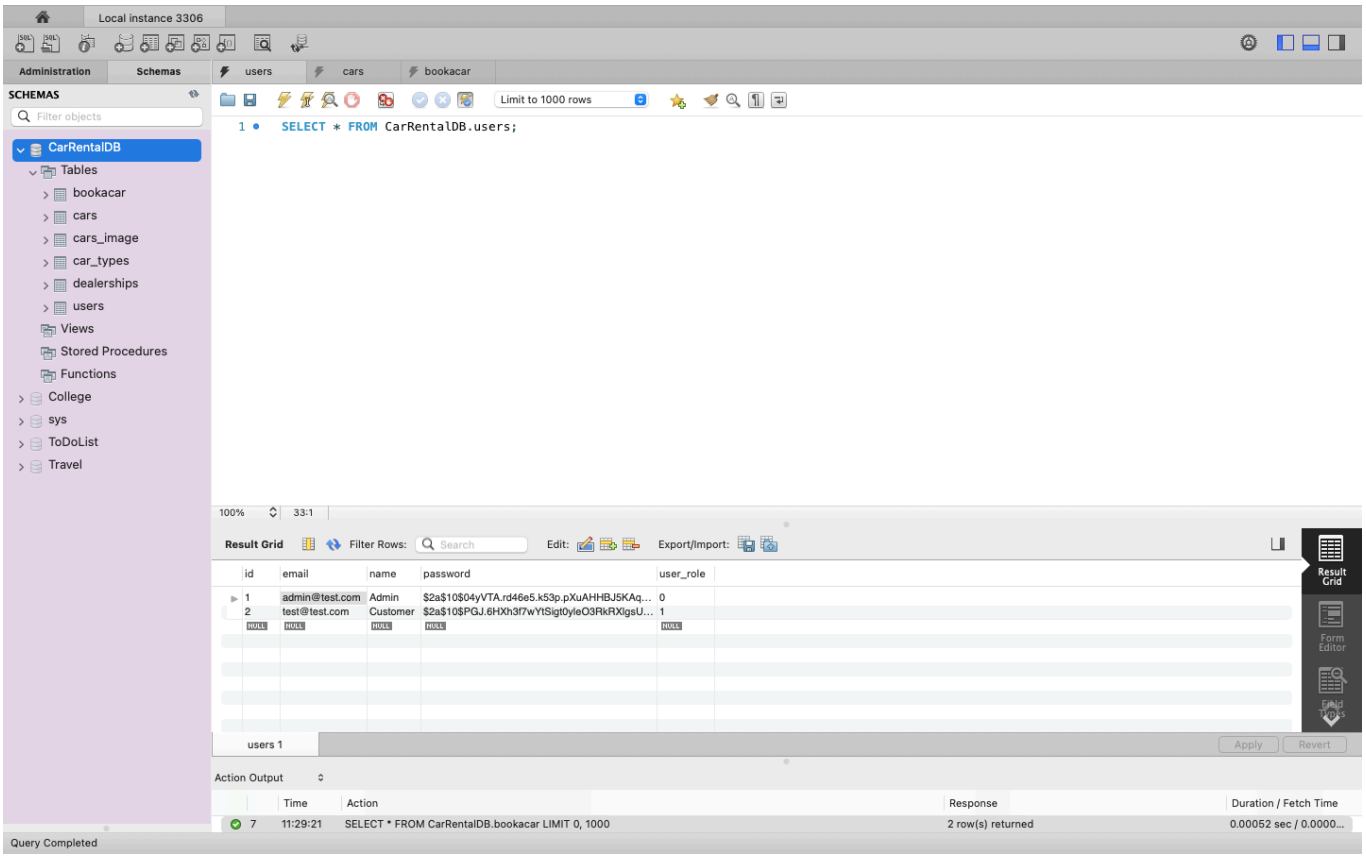


FIG 13: MYSQL WORKBENCH - USER DATABASE

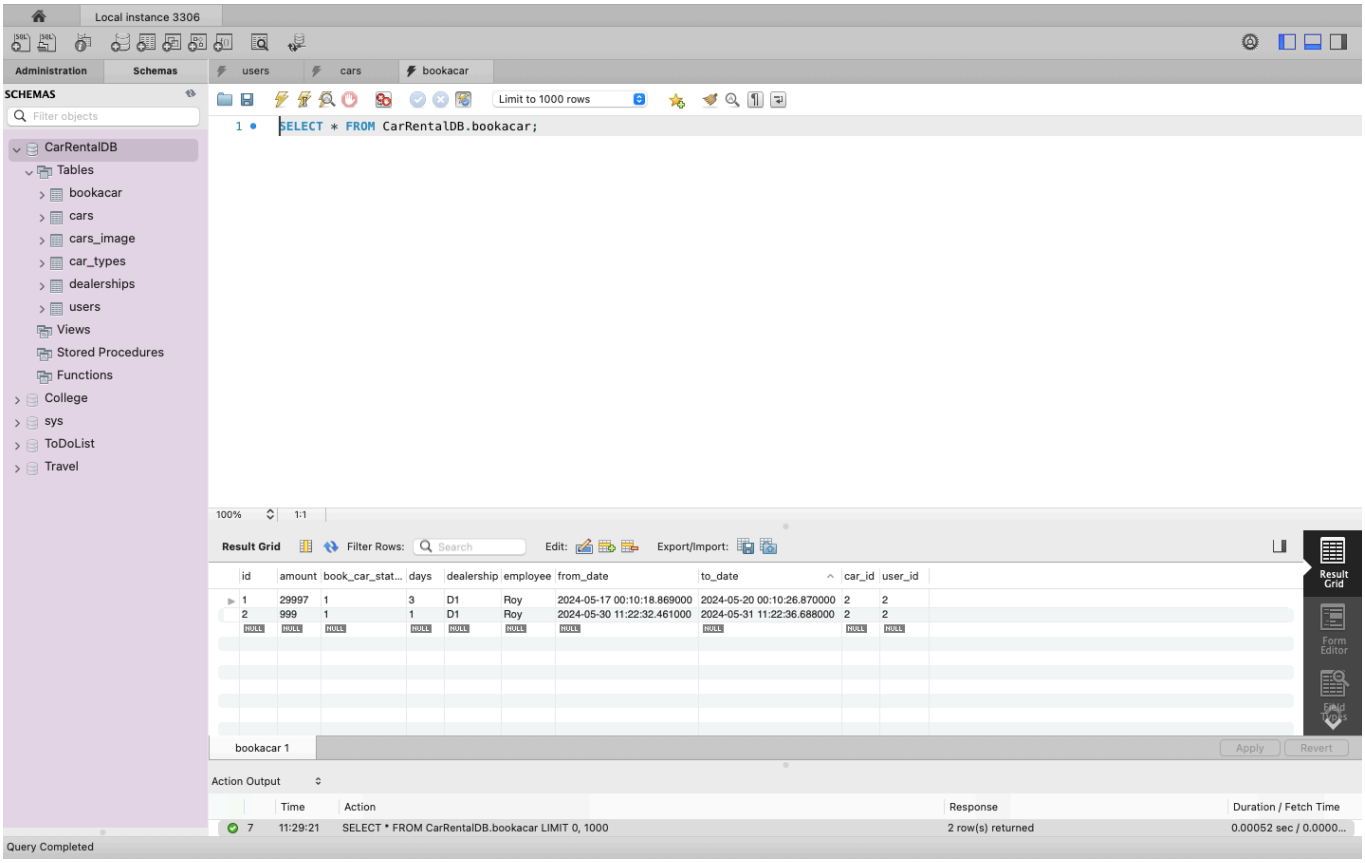


FIG 14: MYSQL WORKBENCH - BOOKING DATABASE

CONCLUSION

The Car Rental System project successfully addresses the inefficiencies and limitations of traditional car rental methods by providing a modern, user-friendly platform that enhances both customer and administrative experiences. By implementing comprehensive modules for user authentication, car listing management, booking request handling, advanced car search, and responsive design, the system ensures efficient management and streamlined processes. Administrators benefit from robust tools that allow meticulous control over the rental inventory and booking requests, while customers enjoy an intuitive interface for browsing, booking, and viewing past rentals. The system's advanced search capabilities and responsive design ensure that users can access and interact with the platform seamlessly across various devices. Overall, this project demonstrates significant improvements in operational efficiency, accuracy, and user satisfaction, contributing to a more effective and user-centric car rental management system. By modernizing the car rental process, this system stands to benefit the car rental industry by providing a reliable and efficient solution for managing rentals and improving customer service.

REFERENCES

For successfully completing this project. We have taken help from the following web resources:-

1. **HTML:** MDN Web Docs. (n.d.). HTML: HyperText Markup Language. Retrieved from https://developer.mozilla.org/en-US/docs/Web/HTML
2. **CSS:** MDN Web Docs. (n.d.). CSS: Cascading Style Sheets. Retrieved from https://developer.mozilla.org/en-US/docs/Web/CSS
3. **JavaScript:** MDN Web Docs. (n.d.). JavaScript Guide. Retrieved from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
4. **Java:** Oracle. (n.d.). Java Documentation. Retrieved from https://docs.oracle.com/en/java/
5. **MySQL:** MySQL Official Documentation. (n.d.). Retrieved from https://dev.mysql.com/doc/
6. **Spring Boot:** Spring Boot Official Documentation. (n.d.). Retrieved from https://spring.io/projects/spring-boot
7. **Maven:** Maven Official Documentation. (n.d.). Retrieved from https://maven.apache.org/guides/index.html