



# Advanced Web Development

## Project Report

Student Name		Assessor Name	
Eng Wen Ping 041240370		Ms Farah Wahidah Kamarudin	
Date Issued	Completion Date	Submitted on	
26 May 2025	31 May 2025	31 May 2025	

Project Title	Design, Develop, Implement & Document Used Car Sales Portal
---------------	---

<b>Learner declaration</b>	
I certify that the work submitted for this assignment is my own and research sources are fully acknowledged.	
Students signatures:	Date: 3 April 2025

## Table of Content

---

Task 1	Project Introduction	
1.1	Project Vision	15
1.2	Project Mision	15
1.3	Project Goal	15
Task 2	Project Objective	
2.2	Project Technical Environment	17
Task 3	System Architecture	
Task 4	Spring MVC Structure	
4.1	Model Structure	20
4.1.1	Appointment Model	20
4.1.2	AuditLog Model	21
4.1.3	CarImage Model	22
4.1.4	Car Model	22
4.1.5	Notification Model	24
4.1.6	PasswordResetToken Model	25
4.1.7	Report Model	26
4.1.8	User Model	27
4.2	Controller Structure	28
4.2.1	Auth Controller	28
4.2.2	Home Controller	30
4.2.3	Car Controller	32
4.2.4	User Controller	35

## Table of Content

---

4.2.5 Admin Controller	37
4.3 View Structure	46
4.3.1 Home Page	46
4.3.2 Login Page	46
4.3.3 Register Page	47
4.3.4 Forget Password Page	47
4.3.5 Reset Password Page	48
4.3.6 Car Catalog Page	49
4.3.7 Car Detail Page	50
4.3.8 About Us Page	50
4.3.9 Contact Us Page	51
4.3.10 Profile Page	51
4.3.10.1 Car Posted	52
4.3.10.2 Appointment Booked	52
4.3.10.3 User Notificaiton	53
4.3.11 Car Posting Page	53
4.3.12 Admin Dashboard	54
4.3.12.1 User Management	54
4.3.12.2 Car Management	55
4.3.12.3 Appointment Management	55
4.3.12.4 Reports Management	56

## Table of Content

---

### Task 5 CRUD functionalities Showcase

5.1 Create functionalities	57
5.1.1 Register in the Portal	57
5.1.2 Post a Car for Sale with Picture upload	58
5.1.3 Book an appointment for Test Drive	59
5.1.4 Report a Car (Additional Feature)	61
5.2 Read functionalities	62
5.2.1 login with registered account	62
5.2.2 Visit Car Listing	63
5.2.3 Search for a Car by Make, Model, Registration Year & Price Range	64
5.2.4 View List of Car	65
5.2.5 View List of Appointments	65
5.2.6 View List of Report	65
5.3 Update functionalites	66
5.3.1 Update User Profile	66
5.3.2 Cancel an Appointment	67
5.3.3 Promote User as Administrator / Demote Administrator as User	67
5.3.4 Mark as Sold / Deactivate a Car Post	69
5.3.5 Approve the Car based on bidding price	70
5.4 Delete functionalities	72
5.4.1 Clear Notifications (Extra Feature)	72
5.4.2 Admin Remove a Car Post	73

## Table of Content

---

5.4.3 Admin Removing the User	74
Task 6 Role-based Access	75
6.1 Public Role	77
6.1.1 Public Role Header	78
6.1.2 Public Role Car Detail Page	78
6.2 User Role	79
6.2.1 User Role Header	80
6.2.2 User Role Car Detail Page	80
6.2 Admin Role	81
6.2.1 Admin Role Header	82
6.1.2 Admin Role Car Detail Page	82
Task 7 JUnit Test Plan	83
7.1 Functional Testing	83
7.1.1 UserController Functional Test Table	83
7.1.2 Car Posting Testing	84
7.1.3 Change Car Status into Active Testing	84
7.1.4 Cancel Appointment Testing	85
7.1.5 Clear Notificaiton Testing	85
7.1.6 Cancel Appointment that cannot be canceled Testing	86
7.2 UI Testing	86
7.2.1 UserDashboard UI Test Table	87
7.2.2 UserDashboard Render Testing	87

## Table of Content

---

7.2.3	UserDashboard Pagination Link Testing	88
7.2.4	UserDashboard Clear Notifiication Testing	88
7.2.5	UserDashboard Cancel Appointment Testing	88
7.3	Validation Testing	89
7.3.1	Validation Test Table	89
7.3.2	>5 Images Uploaded in CarPost Testing	90
7.3.3	CarPost with Valid Data Testing	91
7.4	Security Testing	91
7.4.1	Unauthenticated User Access Test	92
7.5	Database Integration Testing	92
7.5.1	Register User Data With Database Test	93
7.6	Role-Based Testing	94
7.6.1	Unauthenticated User Access Test	94
Task 8	Project Limitations	95
8.1	Lack of Application-Level Validation in Models	95
8.2	No File Upload Validation in Car Posting	95
8.3	No Client-Side Validation Enforcement	95
8.4	No Password Complexity Enforcement	96
8.5	No Transactional Rollback for Database Tests	96
8.6	No Rate Limiting or Throttling	96
8.7	No Input Sanitization	96
8.8	Returns to the active after Forms Submission in Sections	97

## Table of Content

---

8.9 User cannot update their username	97
Task 9 Future Improvements	98
9.1 Implement CustomValidators in All of the Models	98
9.2 Add File Upload Validation in CarService	98
9.3 Add Client-Side Validation with JavaScript	98
9.4 Implement Password Complexity Validation in UserService	98
9.5 Add Transactional and Rollback / Use another Database when Testing	99
9.6 Rate Limiting on Sensitive Endpoints	99
9.7 Implement Input Sanitization with a Filter	99
9.8 Return back to the Section when Forms submissions	99
9.9 Set username as unique and Apply username Check	99
Conclusion	100
Appendix	101

## Table of Figures

---

Figure 1	System Architecture Block Diagram	19
Figure 2	AppointmentModel.java	20
Figure 3	AuditLogModel.java	21
Figure 4	CarImageModel.java	22
Figure 5	CarModel.java	23
Figure 6	NotificationModel.java	24
Figure 7	PasswordResetTokenModel.java	25
Figure 8	ReportModel.java	26
Figure 9	UserModel.java	27
Figure 10	AuthController.java	29
Figure 11	HomeController.java	31
Figure 12	CarController.java Part 1	33
Figure 13	CarController.java Part 2	34
Figure 14	UserController.java Part 1	36
Figure 15	UserController.java Part 2	37
Figure 16	AdminController.java Part 1	38
Figure 17	AdminController.java Part 2	39
Figure 18	AdminController.java Part 3	40
Figure 19	AdminController.java Part 4	41
Figure 20	AdminController.java Part 5	42
Figure 21	AdminController.java Part 6	43
Figure 22	AdminController.java Part 7	44

## Table of Figures

---

Figure 23 AdminController.java Part 8	45
Figure 24 home.html	46
Figure 25 login.html	46
Figure 26 register.html	47
Figure 27 reset-password-request.html	47
Figure 28 reset-password.html	48
Figure 29 car-listing.html	49
Figure 30 car-details.html	50
Figure 31 about.html	50
Figure 32 contact.html	51
Figure 33 user-dashboard.html	51
Figure 34 user-dashboard.html Posted Car	52
Figure 35 user-dashboard.html Appointments	52
Figure 36 user-dashboard.html Notification	53
Figure 37 post-car.html	53
Figure 38 admin-dashboard.html	54
Figure 39 admin-dashboard.html User Management	54
Figure 40 admin-dashboard.html Car Management	55
Figure 41 admin-dashboard.html Appointment Management	55
Figure 42 admin-dashboard.html Reports Management	56
Figure 43 Database before registration	57
Figure 44 register.html with data	57

## Table of Figures

---

Figure 45 Database after registration	57
Figure 46 Html Page before Posting Car	58
Figure 47 Html Page after Car Posting	59
Figure 48 Html Page before making appointment	59
Figure 49 Html Page after making appointment	60
Figure 50 User Dashboard of Booked Appoinments	60
Figure 51 Html Page before making report	61
Figure 52 Html Page after making report	61
Figure 53 Admin Dashboard after Reports	62
Figure 54 Login with data	62
Figure 55 Page redirect after Login	62
Figure 56 Car Listing After Cat Car Added	63
Figure 57 Catalog Page After Filtering	64
Figure 58 List of Car in Admin Dashboard	65
Figure 59 List of Appointments in Admin Dashboard	65
Figure 60 List of Report in Admin Dashboard	65
Figure 61 User Profile before Update	66
Figure 62 User Profile after Update	66
Figure 63 Html Page before cancellation	67
Figure 64 Html Page after cancellation	67
Figure 65 Html Page before Promoting	67
Figure 66 Html Page after Promoting	68

## Table of Figures

---

Figure 67	Html Page after Demoting	68
Figure 68	Html Page before Marked and Deactivate	69
Figure 69	Html Page after Marked	69
Figure 70	Html Page after Deactivate	69
Figure 71	Catalog Page after and Marked Deactivate	70
Figure 72	Admin Dashboard Before Approval	70
Figure 73	Admin Dashboard After Approval	71
Figure 74	Catalog Page after Admin Approval	71
Figure 75	Html Page before Clearing Notifications	72
Figure 76	Confirmation Popup	72
Figure 77	Html Page after Clearing Notifications	72
Figure 78	Reason for Deletion Popup	73
Figure 79	Html Page after Car Removed	73
Figure 80	Html Page before User Deletion	74
Figure 81	User Deletion Confirmation Popup	74
Figure 82	Html Page after User Deletion	74
Figure 83	SecurityConfig Part 1	75
Figure 84	SecurityConfig Part 2	76
Figure 85	Public Role Header	78
Figure 86	Public Role Car Detail Page	78
Figure 87	User Role Header	80
Figure 88	User Role Car Detail Page	80

## Table of Figures

---

Figure 89 Admin Role Header	82
Figure 90 Admin Role Car Detail Page	82
Figure 91 UserControllerFunctionalTest.java Output	83
Figure 92 testPostCar_Success()	84
Figure 93 testActivateCar_Success()	84
Figure 94 testCancelAppointment_Success()	85
Figure 95 testClearNotifications_Success()	85
Figure 96 testCancelAppointment_CannotCancel()	86
Figure 97 UserDashboardUITest.java Result	86
Figure 98 testUserDashboard_RendersCorrectly()	87
Figure 99 testUserDashboard_PaginationLinksPresent()	88
Figure 100 testUserDashboard_ClearNotificationsButtonAndModal()	88
Figure 101 testUserDashboardCancelAppointmentForm()	88
Figure 102 ValidationTest.java Output	89
Figure 103 testCarPost_InvalidImages_TooMany()	90
Figure 104 testCarPost_ValidData()	91
Figure 105 SecurityTest.java Output	91
Figure 106 testUnauthenticatedUserAccess_Denied()	92
Figure 107 IntegrationTest.java Output	93
Figure 108 testUserRegistrationWithDatabase()	93
Figure 109 RoleBasedTest.java Output	94
Figure 110 testUserRoleAccessToAdminDashboard_Redirect()	94

## **Table of Tables**

---

7.1.1 UserController Functional Test Table	83
7.2.1 UserDashboard UI Test Table	87
7.3.1 Validation Test Table	89

## **Task 1 Project Introduction**

You are a freelance Web Developer, and you have recently been approached by AutoXpress Pvt Ltd to develop a Used Car Sales portal. The customer wishes the following functionalities on the website:

- Users will be able to register on the portal using the Registration Page.
- Users can search for cars using filters such as Make, Model, Registration, and Price Range.
- After searching, users will be able to view detailed information about the cars.
- The portal will allow users to log in and post a car for sale.

Systematic overview of Project:

There are 2 types of users in this Used Cars Sales portal:

1. Users
2. Administrator

Users should be able to perform following functions in the portal:

1. Register in the Portal
2. Login to the Portal
3. Post a Car for Sale along with Picture upload and bidding price
4. Deactivate an Existing car sale
5. Update their Profile after logging in.
6. Book an appointment for test drive

Administrators should be able to perform the following functions in the portal

1. Register in the Portal
2. Login to the Portal
3. View List of Registered Users
4. Mark a User as Administrator
5. Activate (Mark as Sold) / Deactivate a Car post
6. Update their Profile
7. Approve or Deny the users appointment
8. Approve the Car Post if bidding price is right

Both Users & Administrator

1. Visit Home Page
2. View Car Listing
3. Search for a Car by Make, Model, Registration Year & Price Range
4. About Us Page
5. Contact Us Page

## **1.1 Project Vision**

The vision of this project is to demonstrate practical knowledge and application of Spring MVC web development concepts through the creation of a Used Car Sales Portal for AutoXpress Pvt Ltd.

## **1.2 Project Mision**

The customer wishes the following functionalities on the website:

- Users will be able to register on the portal using the Registration Page.
- Users can search for cars using filters such as Make, Model, Registration, and Price Range.
- After searching, users will be able to view detailed information about the cars.
- The portal will allow users to log in and post a car for sale.

## **1.3 Project Goal**

This project bridges theoretical learning with real-world web application development, reinforcing skills required for enterprise-level Java web solutions , while promoting responsible and sustainable development principle.

## **Task 2 Project Objective**

The scope of this project includes the full-stack development of a web-based Used Car Sales Portal, using technologies and patterns introduced in the course. The portal will be designed to support two types of users - Registered Users and Administrators, each with role-specific access and functionalities.

Covered Concepts and Technologies:

- Spring MVC framework for layered web architecture
- Model-View-Controller (MVC) separation of concerns
- Request Mapping using annotations (@Controller, @GetMapping, @PostMapping)
- Form handling and validation using Spring's form binding
- View rendering using Thymeleaf template engine
- Custom application configuration using Java-based @Configuration classes
- Session-based authentication logic
- CRUD Operations and Filtering/ Search features
- Static and Dynamic Resource Management
- Sustainable development practices including maintainable code, reduced redundancy, energy-aware design, and responsible use of frameworks/libraries

## **2.2 Project Technical Environment**

Below are the following tools used to developed for this project:

- Eclipse Spring Tool Workspace v-4.30 RELEASE
- JavaSE - 17
- Dependencies of (JSP, Spring Web, JDBC, JSTL)
- HTML, CSS, Java
- Bootstrap
- XAMPP MySQL Server
- Spring Boot

### Task 3 System Architecture

The AutoXpress application is a Spring Boot-based web application designed to manage car listings, user registrations and appointments based on project requirement shown in Figure 1 below.

The **MVC, View Layer** (HTML Web Pages) **handles Post and Get Http Request to return or views or redirects** and handle **forms submission**.

The **Security Layer** (SecurityConfig) configures Spring Security for **role-based access control** and **authentication**, ensuring restricted access to endpoints such as : (/admin/\*\* for ADMIN role only).

The **Controller Layer** delegate business logic to services and **manage view navigation**.

The **Service Layer** encapsulates **business logic**, such as UserService for user registration, CarService for car posting, EmailService for Gmail SMTP and Appointment Service appointment management. Services then **interact with repositories** for data persistence and other services for cross-cutting concerns (e.g., notifications) along with model layer. **Model Layer** represent as the application's **data entities** mapping to database table via JPA annotations.

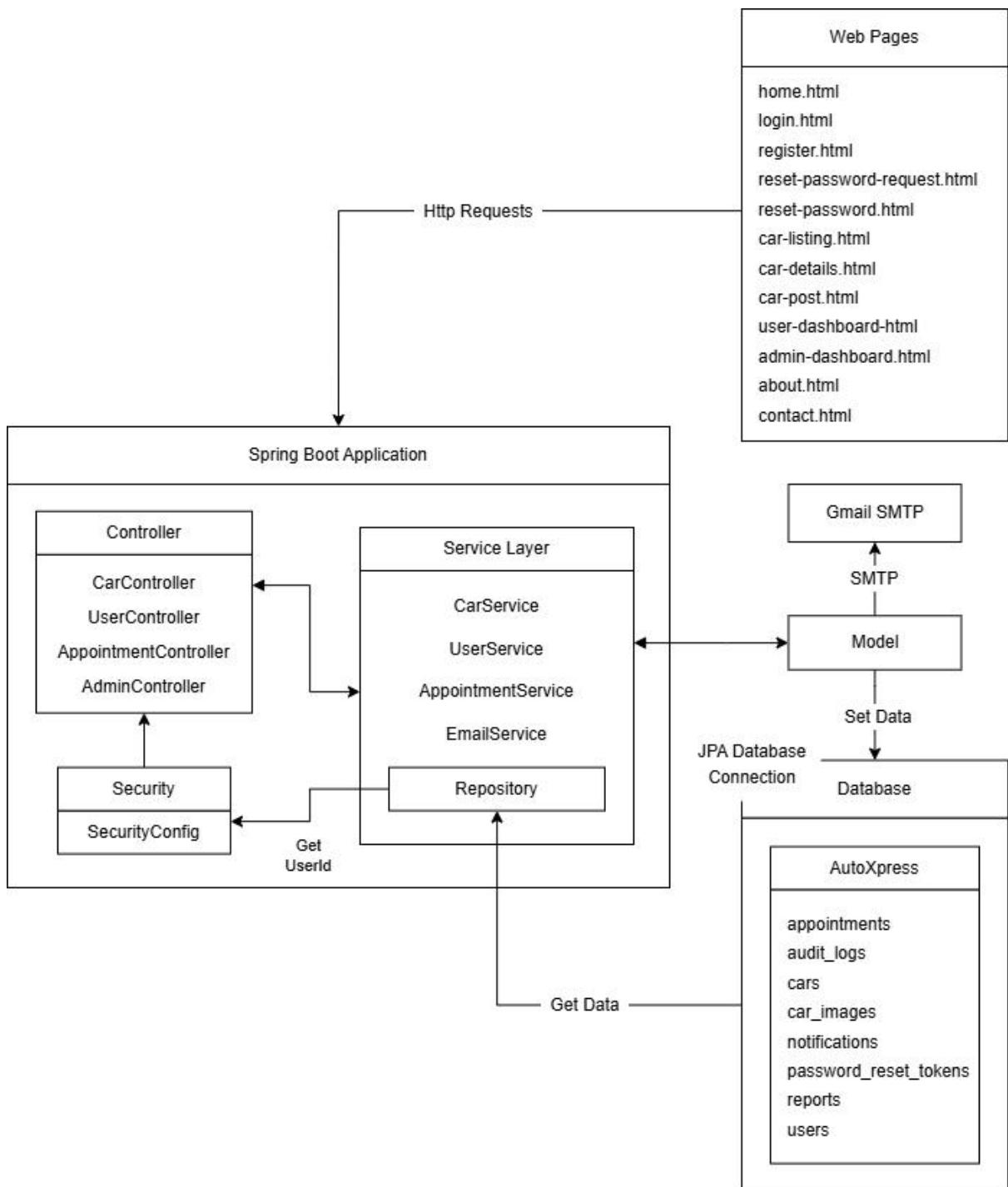


Figure 1 System Architecture Block Diagram

## Task 4 Spring MVC Structure

### 4.1 Model Structure

AutoXpress had a total of 8 Models. All of the class uses getters and setters for accessing and modifying its fields.

#### 4.1.1 Appointment Model

The **AppointmentModel** class is a Java entity in a Spring application, mapped to "appointments" table, representing a **scheduled test drive for a car listing**. It includes fields like appointmentId, appointmentUser, appointmentCar, appointmentDate, and appointmentStatus, along with timestamps for creation and updates, all annotated with JPA mappings.

```
7 @Data
8 @Entity
9 @Table(name = "appointments")
10 public class AppointmentModel {
11
12     @Id
13     @GeneratedValue(strategy = GenerationType.IDENTITY)
14     @Column(name = "appointment_id")
15     private Long appointmentId;
16
17     @ManyToOne
18     @JoinColumn(name = "appointment_user_id", nullable = false)
19     private UserModel appointmentUser;
20
21     @ManyToOne
22     @JoinColumn(name = "appointment_car_id", nullable = false)
23     private CarModel appointmentCar;
24
25     @Column(name = "appointment_date", nullable = false)
26     private LocalDateTime appointmentDate;
27
28     @Enumerated(EnumType.STRING)
29     @Column(name = "appointment_status", nullable = false)
30     private AppointmentStatus appointmentStatus;
31
32     @Column(name = "appointment_created_at", nullable = false)
33     private LocalDateTime appointmentCreatedAt;
34
35     @Column(name = "appointment_updated_at", nullable = false)
36     private LocalDateTime appointmentUpdatedAt;
37
38     public enum AppointmentStatus {
39         PENDING, APPROVED, DENIED, CANCELED
40     }
```

Figure 2 AppointmentModel.java

#### 4.1.2 AuditLog Model

The **AuditLogModel** class is a Java entity in a Spring application, mapped to an "audit\_logs" table, representing **logs of administrative actions** within the system. It includes fields such as auditId, auditAdmin, auditAction, auditEntityType, auditEntityId, and auditCreatedAt, all annotated with JPA mappings to ensure proper persistence.

```
7  @Data
8  @Entity
9  @Table(name = "audit_logs")
10 public class AuditLogModel {
11
12     @Id
13     @GeneratedValue(strategy = GenerationType.IDENTITY)
14     @Column(name = "audit_id")
15     private Long auditId;
16
17     @ManyToOne
18     @JoinColumn(name = "audit_admin_id", nullable = false)
19     private UserModel auditAdmin;
20
21     @Column(name = "audit_action", nullable = false)
22     private String auditAction;
23
24     @Column(name = "audit_entity_type")
25     private String auditEntityType;
26
27     @Column(name = "audit_entity_id")
28     private Long auditEntityId;
29
30     @Column(name = "audit_created_at", nullable = false)
31     private LocalDateTime auditCreatedAt;
```

Figure 3 AuditLogModel.java

#### 4.1.3 CarImage Model

The **CarImageModel** class is a Java entity in a Spring application, mapped to a "car\_images" table, representing **images associated with car listings** which is CarModel. It includes fields like imageId, imageCar, imagePath, and imageCreatedAt, annotated with JPA mappings to ensure proper database persistence.

```
@Data  
@Entity  
@Table(name = "car_images")  
public class CarImageModel {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(name = "image_id")  
    private Long imageId;  
  
    @ManyToOne  
    @JoinColumn(name = "image_car_id", nullable = false)  
    @JsonBackReference  
    private CarModel imageCar;  
  
    @Column(name = "image_path", nullable = false)  
    private String imagePath;  
  
    @Column(name = "image_created_at", nullable = false)  
    private LocalDateTime imageCreatedAt;
```

Figure 4 CarImageModel.java

#### 4.1.4 Car Model

The **CarModel** class is a Java entity in a Spring application, mapped to a "cars" table, representing **car listings** within the system. It includes fields such as carId, carUser, carMake, carModel, registrationYear, carPrice, carMileage, fuelType, transmissionType, carStatus, carCreatedAt, carUpdatedAt, and a list of carImages, all annotated with JPA mappings for database persistence.

```

@Data
@Entity
@Table(name = "cars")
public class CarModel {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "car_id")
    private Long carId;

    @ManyToOne
    @JoinColumn(name = "car_user_id", nullable = false)
    private UserModel carUser;

    @Column(name = "car_make", nullable = false)
    private String carMake;

    @Column(name = "car_model", nullable = false)
    private String carModel;

    @Column(name = "registration_year", nullable = false)
    private Integer registrationYear;

    @Column(name = "car_price", nullable = false)
    private Double carPrice;

    @Column(name = "car_mileage")
    private Integer carMileage;

    @Enumerated(EnumType.STRING)
    @Column(name = "fuel_type")
    private FuelType fuelType;

    @Enumerated(EnumType.STRING)
    @Column(name = "transmission_type")
    private TransmissionType transmissionType;

    @Enumerated(EnumType.STRING)
    @Column(name = "car_status", nullable = false)
    private CarStatus carStatus;

    @Column(name = "car_created_at", nullable = false)
    private LocalDateTime carCreatedAt;

    @Column(name = "car_updated_at", nullable = false)
    private LocalDateTime carUpdatedAt;

    @OneToMany(mappedBy = "imageCar", cascade = CascadeType.ALL, orphanRemoval = true)
    @JsonManagedReference
    private List<CarImageModel> carImages;

    public enum FuelType {
        PETROL, DIESEL, ELECTRIC, HYBRID
    }

    public enum TransmissionType {
        MANUAL, AUTOMATIC
    }

    public enum CarStatus {
        PENDING, ACTIVE, INACTIVE, SOLD
    }
}

```

Figure 5 CarModel.java

#### 4.1.5 Notification Model

The **NotificationModel** class is a Java entity in a Spring application, mapped to a "notifications" table, representing **system notifications for users**. It includes fields such as notificationId, notificationUser, notificationMessage, notificationType, notificationCreatedAt, and notificationRead.

```
@Data  
@Entity  
@Table(name = "notifications")  
public class NotificationModel {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(name = "notification_id")  
    private Long notificationId;  
  
    @ManyToOne  
    @JoinColumn(name = "notification_user_id", nullable = false)  
    private UserModel notificationUser;  
  
    @Column(name = "notification_message", nullable = false)  
    private String notificationMessage;  
  
    @Enumerated(EnumType.STRING)  
    @Column(name = "notification_type", nullable = false)  
    private NotificationType notificationType;  
  
    @Column(name = "notification_created_at", nullable = false)  
    private LocalDateTime notificationCreatedAt;  
  
    @Column(name = "notification_read")  
    private Boolean notificationRead;  
  
    public enum NotificationType {  
        BID_UPDATE, APPOINTMENT_STATUS, PASSWORD_RESET, GENERAL  
    }  
}
```

Figure 6 NotificationModel.java

#### 4.1.6 PasswordResetToken Model

The **PasswordResetTokenModel** class is a Java entity in a Spring application, mapped to a "password\_reset\_tokens" table, representing **tokens for user password reset requests**. It includes fields like tokenId, tokenUser, tokenValue, tokenExpiryDate, and tokenCreatedAt.

```
@Data  
@Entity  
@Table(name = "password_reset_tokens")  
public class PasswordResetTokenModel {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(name = "token_id")  
    private Long tokenId;  
  
    @ManyToOne  
    @JoinColumn(name = "token_user_id", nullable = false)  
    private UserModel tokenUser;  
  
    @Column(name = "token_value", nullable = false, unique = true)  
    private String tokenValue;  
  
    @Column(name = "token_expiry_date", nullable = false)  
    private LocalDateTime tokenExpiryDate;  
  
    @Column(name = "token_created_at", nullable = false)  
    private LocalDateTime tokenCreatedAt;
```

Figure 7 PasswordResetTokenModel.java

#### 4.1.7 Report Model

The **ReportModel** class is a Java entity in a Spring application, mapped to a "reports" table, representing **user-submitted reports** on **car listings** for issues like misleading information or irrelevant pictures. It includes fields such as reportId, reportUser, reportCar, reportReason, reportDescription, reportStatus, reportCreatedAt, and reportUpdatedAts.

```
@Data
@Entity
@Table(name = "reports")
public class ReportModel {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "report_id")
    private Long reportId;

    @ManyToOne
    @JoinColumn(name = "report_user_id", nullable = false)
    private UserModel reportUser;

    @ManyToOne
    @JoinColumn(name = "report_car_id", nullable = false)
    private CarModel reportCar;

    @Enumerated(EnumType.STRING)
    @Column(name = "report_reason", nullable = false)
    private ReportReason reportReason;

    @Column(name = "report_description")
    private String reportDescription;

    @Enumerated(EnumType.STRING)
    @Column(name = "report_status", nullable = false)
    private ReportStatus reportStatus;

    @Column(name = "report_created_at", nullable = false)
    private LocalDateTime reportCreatedAt;

    @Column(name = "report_updated_at", nullable = false)
    private LocalDateTime reportUpdatedAt;

    public enum ReportReason {
        IRRELEVANT_PICTURE, MISLEADING_INFO, OTHER
    }

    public enum ReportStatus {
        PENDING, REVIEWED, RESOLVED
    }
}
```

Figure 8 ReportModel.java

#### 4.1.8 User Model

The **UserModel** class is a Java entity in a Spring application, mapped to a "users" table, representing **user accounts** within the system. It includes fields like userId, username, userPassword, email, userRole, userCreatedAt, and userUpdatedAt.

```
@Data  
@Entity  
@Table(name = "users")  
public class UserModel {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(name = "user_id")  
    private Long userId;  
  
    @Column(name = "username", nullable = false, unique = true)  
    private String username;  
  
    @Column(name = "user_password", nullable = false)  
    private String userPassword;  
  
    @Column(name = "email", nullable = false, unique = true)  
    private String email;  
  
    @Enumerated(EnumType.STRING)  
    @Column(name = "user_role", nullable = false)  
    private Role userRole;  
  
    @Column(name = "user_created_at", nullable = false)  
    private LocalDateTime userCreatedAt;  
  
    @Column(name = "user_updated_at", nullable = false)  
    private LocalDateTime userUpdatedAt;  
  
    public enum Role {  
        USER, ADMIN  
    }  
}
```

Figure 9 UserModel.java

## 4.2 Controller Structure

### 4.2.1 Auth Controller

The **AuthController** class is a Spring MVC controller in a Java application, handling user **authentication** and **password reset** workflows. It manages endpoints for **login** (/login), **registration** (/register), **password reset requests** (/password-reset-request), and **password resets** (/password-reset), interacting with **UserService** below to perform operations like **user registration** and **password updates**. The controller uses **@Valid** and **BindingResult** for **form validation**, ensuring robust error handling, and employs **model attributes** to pass data to views like login.html, register.html, and password-reset.html. It leverages dependency injection via **@Autowired** to integrate with UserService for **business logic execution**.

```

// Displays the login page
@GetMapping("/login")
public String login(Model model) {
    return "login";
}

// Displays the registration form with a new UserModel instance
@GetMapping("/register")
public String showRegistrationForm(Model model) {
    model.addAttribute("user", new UserModel());
    return "register";
}

// Handles user registration, validates the form, and redirects to login on success
@PostMapping("/register")
public String registerUser(
    @Valid @ModelAttribute("user") UserModel user,
    BindingResult result,
    Model model) {
    if (result.hasErrors()) {
        return "register";
    }
    try {
        userService.registerUser(user);
        return "redirect:/login?success";
    } catch (Exception e) {
        model.addAttribute("error", e.getMessage());
        return "register";
    }
}
// Displays the password reset request form
@GetMapping("/password-reset-request")
public String showPasswordResetRequestForm(Model model) {
    return "password-reset-request";
}

// Processes password reset requests by email, displays success or error
@PostMapping("/password-reset-request")
public String requestPasswordReset(
    @RequestParam("email") String email,
    Model model) {
    try {
        userService.requestPasswordReset(email);
        model.addAttribute("success", true);
    } catch (Exception e) {
        model.addAttribute("error", "An error occurred while processing your request.");
    }
    return "password-reset-request";
}

// Displays the password reset form for a given token
@GetMapping("/password-reset")
public String showPasswordResetForm(
    @RequestParam("token") String token,
    Model model) {
    model.addAttribute("user", new UserModel());
    return "password-reset";
}

// Handles password reset, validates the form, checks password match, and updates the password
@PostMapping("/password-reset")
public String resetPassword(
    @RequestParam("token") String token,
    @Valid @ModelAttribute("user") UserModel user,
    @RequestParam("confirmPassword") String confirmPassword,
    BindingResult result,
    Model model) {
    if (result.hasErrors()) {
        return "password-reset";
    }
    if (!user.getUserPassword().equals(confirmPassword)) {
        model.addAttribute("error", "Passwords do not match.");
        return "password-reset";
    }
    try {
        userService.resetPassword(token, user.getUserPassword());
        model.addAttribute("success", true);
    } catch (Exception e) {
        model.addAttribute("error", e.getMessage());
    }
    return "password-reset";
}

```

Figure 10 AuthController.java

#### **4.2.2 Home Controller**

The **HomeController** class is a Spring MVC controller in a Java application, managing the core **web navigation** and **car listing functionalities**. It handles endpoints for the **homepage (/)**, **car listings (/cars)**, **AJAX-based car search (/cars/search)**, **contact page (/contact)** and an **about page (/about)**, interacting with **CarService** to **fetch and display paginated car data** with **filtering options** like **make**, **model**, and **price**. The controller **supports dynamic pagination**, **processes search parameters**, and **returns data** for both **full page renders** and **AJAX responses**, ensuring **seamless user interaction** with the `home.html`, `car-listing.html`, and `about-us.html` views.

```

// Displays the homepage
@GetMapping("/")
public String home(Model model) {
    return "home";
}

// Displays the car listing page with pagination and filtered results
@GetMapping("/cars")
public String carListing(@RequestParam(defaultValue = "0") int page, @RequestParam(defaultValue = "5") int size,
    @RequestParam(required = false) String make, @RequestParam(required = false) String carModel,
    @RequestParam(required = false) Integer year, @RequestParam(required = false) Double minPrice,
    @RequestParam(required = false) Double maxPrice, @RequestParam(required = false) Integer mileage,
    @RequestParam(required = false) String fuelType, @RequestParam(required = false) String transmission,
    Model model) {

    Pageable pageable = PageRequest.of(page, size);
    // Convert fuelType and transmission strings to enums
    CarModel.FuelType fuelTypeEnum = (fuelType != null && !fuelType.isEmpty()) ? CarModel.FuelType.valueOf(fuelType)
        : null;
    CarModel.TransmissionType transmissionEnum = (transmission != null && !transmission.isEmpty())
        ? CarModel.TransmissionType.valueOf(transmission)
        : null;

    Page<CarModel> carPage = carService.searchCars(make, carModel, year, minPrice, maxPrice, mileage, fuelTypeEnum,
        transmissionEnum, pageable);

    model.addAttribute("cars", carPage.getContent());
    model.addAttribute("currentPage", carPage.getNumber());
    model.addAttribute("totalPages", carPage.getTotalPages());
    model.addAttribute("totalItems", carPage.getTotalElements());
    model.addAttribute("pageSize", size);

    // Retain search parameters for pagination links
    model.addAttribute("make", make);
    model.addAttribute("carModel", carModel);
    model.addAttribute("year", year);
    model.addAttribute("minPrice", minPrice);
    model.addAttribute("maxPrice", maxPrice);
    model.addAttribute("mileage", mileage);
    model.addAttribute("fuelType", fuelType);
    model.addAttribute("transmission", transmission);

    // Enum values for filter dropdowns
    model.addAttribute("fuelTypes", Arrays.asList(CarModel.FuelType.values()));
    model.addAttribute("transmissionTypes", Arrays.asList(CarModel.TransmissionType.values()));

    return "car-listing";
}

// Handles AJAX requests for car search, returning paginated results as JSON
@GetMapping("/cars/search")
@ResponseBody
public Map<String, Object> searchCarsAjax(@RequestParam(defaultValue = "0") int page,
    @RequestParam(defaultValue = "5") int size, @RequestParam(required = false) String make,
    @RequestParam(required = false) String carModel, @RequestParam(required = false) Integer year,
    @RequestParam(required = false) Double minPrice, @RequestParam(required = false) Double maxPrice,
    @RequestParam(required = false) Integer mileage, @RequestParam(required = false) String fuelType,
    @RequestParam(required = false) String transmission) {

    Map<String, Object> response = new HashMap<>();
    try {
        Pageable pageable = PageRequest.of(page, size);
        // Convert fuelType and transmission strings to enums
        CarModel.FuelType fuelTypeEnum = (fuelType != null && !fuelType.isEmpty())
            ? CarModel.FuelType.valueOf(fuelType)
            : null;
        CarModel.TransmissionType transmissionEnum = (transmission != null && !transmission.isEmpty())
            ? CarModel.TransmissionType.valueOf(transmission)
            : null;

        Page<CarModel> carPage = carService.searchCars(make, carModel, year, minPrice, maxPrice, mileage,
            fuelTypeEnum, transmissionEnum, pageable);

        response.put("cars", carPage.getContent());
        response.put("currentPage", carPage.getNumber());
        response.put("totalPages", carPage.getTotalPages());
        response.put("totalItems", carPage.getTotalElements());
        response.put("pageSize", size);
    } catch (Exception e) {
        System.err.println("Error in /cars/search: " + e.getMessage());
        e.printStackTrace();
        response.put("cars", List.of());
        response.put("currentPage", 0);
        response.put("totalPages", 0);
        response.put("totalItems", 0L);
        response.put("pageSize", size);
    }
    return response;
}

// Display About Us Page
@GetMapping("/about")
public String aboutPage() {
    return "about-us";
}

// Display Contact Us Page
@GetMapping("/contact")
public String contactPage(Model model) {
    return "contact-us";
}

```

Figure 11 HomeController.java

#### **4.2.3 Car Controller**

The **CarController** class is a Spring MVC controller in a Java application, managing car-related interactions such as **viewing car details, posting new cars, booking appointments, and submitting reports**. It handles endpoints like /cars/{id} for car details, /cars/post for listing cars, /cars/{id}/appointment for scheduling test drives, and /cars/{id}/report for reporting issues, **integrating with** services like **CarService, UserService, AppointmentService, ReportService, and NotificationService** to perform these operations. The controller ensures proper form validation using @Valid and BindingResult, **manages file uploads for car images, and sends notifications for user actions**, rendering views such as car-details.html and post-car.html.

```

// Displays the details of a specific car by ID
@GetMapping("/cars/{id}")
public String carDetails(@PathVariable Long id, Model model) {

    CarModel car = carService.getCarById(id);
    if (car == null) {
        return "redirect:/cars?error=CarNotFound";
    }

    model.addAttribute("car", car);
    model.addAttribute("appointment", new AppointmentModel());
    model.addAttribute("report", new ReportModel());

    return "car-details";
}

// Shows the form for posting a new car
@GetMapping("/cars/post")
public String showPostCarForm(Model model, Authentication authentication) {

    String userId = authentication.getName();
    Long userIdLong = Long.parseLong(userId);
    UserModel user = userService.getUserById(userIdLong);

    model.addAttribute("car", new CarModel());
    model.addAttribute("fuelTypes", CarModel.FuelType.values());
    model.addAttribute("transmissionTypes", CarModel.TransmissionType.values());
    Long notificationCount = notificationService.getUnreadNotificationCount(user.getUsername());
    model.addAttribute("notificationCount", notificationCount);
    return "post-car";
}

// Handles the submission of a new car listing with images
@PostMapping("/cars/post")
public String postCar(
    @Valid @ModelAttribute("car") CarModel car,
    BindingResult result,
    @RequestParam("images") List<MultipartFile> images,
    Model model,
    Authentication authentication) throws Exception {
    String userId = authentication.getName();
    Long userIdLong = Long.parseLong(userId);
    UserModel user = userService.getUserById(userIdLong);
    if (user == null) {
        return "redirect:/login";
    }
    if (result.hasErrors() || images.size() < 1 || images.size() > 5) {
        model.addAttribute("fuelTypes", CarModel.FuelType.values());
        model.addAttribute("transmissionTypes", CarModel.TransmissionType.values());
        model.addAttribute("error", images.size() < 1 ? "At least one image is required." :
            images.size() > 5 ? "Maximum 5 images allowed." : null);
        Long notificationCount = notificationService.getUnreadNotificationCount(user.getUsername());
        model.addAttribute("notificationCount", notificationCount);
        return "post-car";
    }
    try {
        carService.saveCar(car, images, user.getUsername());
        notificationService.createNotification(
            user.getUsername(),
            "Car submitted for approval: " + car.getCarMake() + " " + car.getCarModel(),
            NotificationModel.NotificationType.GENERAL
        );
        return "redirect:/user/dashboard?submitted";
    } catch (Exception e) {
        model.addAttribute("fuelTypes", CarModel.FuelType.values());
        model.addAttribute("transmissionTypes", CarModel.TransmissionType.values());
        model.addAttribute("error", "Failed to post car: " + e.getMessage());
        Long notificationCount = notificationService.getUnreadNotificationCount(user.getUsername());
        model.addAttribute("notificationCount", notificationCount);
        return "post-car";
    }
}

```

Figure 12 CarController.java Part 1

```

// Books a test drive appointment for a specific car
@PostMapping("/cars/{id}/appointment")
public String bookAppointment(
    @PathVariable Long id,
    @Valid @ModelAttribute("appointment") AppointmentModel appointment,
    BindingResult result,
    Model model,
    Authentication authentication) throws Exception {

    String userId = authentication.getName();
    Long userIdLong = Long.parseLong(userId);
    UserModel user = userService.getUserById(userIdLong);

    CarModel car = carService.getCarById(id);

    if (car == null || result.hasErrors()) {
        model.addAttribute("car", car);
        model.addAttribute("appointment", appointment);
        model.addAttribute("report", new ReportModel());
        model.addAttribute("error", car == null ? "Car not found." : "Invalid appointment details.");
        Long notificationCount = notificationService.getUnreadNotificationCount(user.getUsername());
        model.addAttribute("notificationCount", notificationCount);
        return "car-details";
    }
    appointmentService.saveAppointment(appointment, car, user.getUsername());
    notificationService.createNotification(
        user.getUsername(),
        "Test drive appointment booked for " + car.getCarMake() + " " + car.getCarModel(),
        NotificationModel.NotificationType.APPOINTMENT_STATUS
    );
    return "redirect:/cars/" + id + "?appointmentBooked";
}

// Submits a report for a specific car listing
@PostMapping("/cars/{id}/report")
public String submitReport(
    @PathVariable Long id,
    @Valid @ModelAttribute("report") ReportModel report,
    BindingResult result,
    Model model,
    Authentication authentication) throws Exception {

    String userId = authentication.getName();
    Long userIdLong = Long.parseLong(userId);
    UserModel user = userService.getUserById(userIdLong);
    CarModel car = carService.getCarById(id);

    if (car == null || result.hasErrors()) {
        model.addAttribute("car", car);
        model.addAttribute("appointment", new AppointmentModel());
        model.addAttribute("report", report);
        model.addAttribute("error", car == null ? "Car not found." : "Invalid report.");
        Long notificationCount = notificationService.getUnreadNotificationCount(user.getUsername());
        model.addAttribute("notificationCount", notificationCount);
        return "car-details";
    }
    reportService.saveReport(report, car, user.getUsername());
    notificationService.createNotification(
        user.getUsername(),
        "Report submitted for " + car.getCarMake() + " " + car.getCarModel(),
        NotificationModel.NotificationType.GENERAL
    );
    return "redirect:/cars/" + id + "?reportSubmitted";
}

```

Figure 13 CarController.java Part 2

#### **4.2.4 User Controller**

The **UserController** class is a Spring MVC controller in a Java application, overseeing user-related functionalities such as **dashboard access, profile management, appointment cancellation, and notification clearing**. It handles endpoints like /user/dashboard for displaying paginated user data, /user/profile for profile updates, /user/appointments/cancel for cancelling test drives, and /user/notifications/clear for removing notifications, interacting with services like **UserService, CarService, AppointmentService, and NotificationService**. The controller employs @Valid and BindingResult for **form validation, supports pagination with Pageable, and manages user authentication** via Authentication, rendering the user-dashboard.html view.

```

// Displays the user dashboard with paginated data for cars, appointments, and notifications
@GetMapping("/user/dashboard")
public String userDashboard(
    Model model,
    Authentication authentication,
    @RequestParam(required = false) String submitted,
    @RequestParam(defaultValue = "0") int pagePostedCars,
    @RequestParam(defaultValue = "0") int pageAppointments,
    @RequestParam(defaultValue = "0") int pageNotifications) {
    String userId = authentication.getName();
    Long userIdLong = Long.parseLong(userId);
    UserModel user = userService.getUserById(userIdLong);
    if (user == null) {
        return "redirect:/login";
    }

    int pageSize = 10; // Limit of 10 items per page

    // Fetch paginated data
    Pageable postedCarsPageable = PageRequest.of(pagePostedCars, pageSize);
    Page<CarModel> postedCarsPage = carService.getCarsByUser(userIdLong, postedCarsPageable);
    model.addAttribute("postedCarsPage", postedCarsPage);
    model.addAttribute("postedCars", postedCarsPage.getContent());

    Pageable appointmentsPageable = PageRequest.of(pageAppointments, pageSize);
    Page<AppointmentModel> appointmentsPage = appointmentService.getAppointmentsByUser(userIdLong, appointmentsPageable);
    model.addAttribute("appointmentsPage", appointmentsPage);
    model.addAttribute("appointments", appointmentsPage.getContent());

    Pageable notificationsPageable = PageRequest.of(pageNotifications, pageSize);
    Page<NotificationModel> notificationsPage = notificationService.getNotificationsByUser(userIdLong, notificationsPageable);
    model.addAttribute("notificationsPage", notificationsPage);
    model.addAttribute("notifications", notificationsPage.getContent());

    model.addAttribute("user", user);
    model.addAttribute("notificationCount", notificationService.getUnreadNotificationCount(user.getUsername()));
    if (submitted != null) {
        model.addAttribute("message", "Car submitted for approval!");
    }
    return "user-dashboard";
}

// Shows the profile form for editing user details
@GetMapping("/user/profile")
public String showProfileForm(Model model, Authentication authentication) {
    String userId = authentication.getName();
    Long userIdLong = Long.parseLong(userId);
    UserModel user = userService.getUserById(userIdLong);
    if (user == null) {
        return "redirect:/login";
    }
    model.addAttribute("user", user);
    model.addAttribute("notificationCount", notificationService.getUnreadNotificationCount(user.getUsername()));
    return "user-dashboard";
}

// Handles profile updates with form validation
@PostMapping("/user/profile")
public String updateProfile(
    @Valid @ModelAttribute("user") UserModel user,
    BindingResult result,
    Model model,
    Authentication authentication) throws Exception {
    String userId = authentication.getName();
    Long userIdLong = Long.parseLong(userId);
    UserModel loggedInUser = userService.getUserById(userIdLong);
    if (loggedInUser == null) {
        return "redirect:/login";
    }
    String username = loggedInUser.getUsername();
    if (result.hasErrors()) {
        model.addAttribute("postedCars", carService.getCarsByUser(userIdLong));
        model.addAttribute("appointments", appointmentService.getAppointmentsByUser(userIdLong));
        model.addAttribute("notifications", notificationService.getNotificationsByUser(userIdLong));
        model.addAttribute("notificationCount", notificationService.getUnreadNotificationCount(username));
        model.addAttribute("user", loggedInUser);
        return "user-dashboard";
    }
    userService.updateUserProfile(user, username);
    notificationService.createNotification(
        username,
        "Profile updated successfully",
        com.autoxpress.model.NotificationModel.NotificationType.GENERAL
    );
    return "redirect:/user/dashboard?updated";
}

```

Figure 14 UserController.java Part 1

```

// Cancels a user's appointment if it's pending or approved
@PostMapping("/user/appointments/cancel")
public String cancelAppointment(@RequestParam("appointmentId") Long appointmentId, Model model, Authentication authentication) {
    String userId = authentication.getName();
    Long userIdLong = Long.parseLong(userId);
    UserModel user = userService.getUserById(userIdLong);
    if (user == null) {
        return "redirect:/login";
    }
    String username = user.getUsername();
    AppointmentModel appointment = appointmentService.getAppointmentById(appointmentId);
    if (appointment == null || !appointment.getAppointmentUser().getUserId().equals(userIdLong)) {
        model.addAttribute("error", "Failed to cancel appointment: Invalid appointment or permission.");
        model.addAttribute("user", user);
        model.addAttribute("postedCars", carService.getCarsByUser(userIdLong));
        model.addAttribute("appointments", appointmentService.getAppointmentsByUser(userIdLong));
        model.addAttribute("notifications", notificationService.getNotificationsByUser(userIdLong));
        model.addAttribute("notificationCount", notificationService.getUnreadNotificationCount(username));
        return "user-dashboard";
    }
    if (appointment.getAppointmentStatus() != AppointmentModel.AppointmentStatus.PENDING &&
        appointment.getAppointmentStatus() != AppointmentModel.AppointmentStatus.APPROVED) {
        model.addAttribute("error", "Cannot cancel an appointment that is not pending or approved.");
        model.addAttribute("user", user);
        model.addAttribute("postedCars", carService.getCarsByUser(userIdLong));
        model.addAttribute("appointments", appointmentService.getAppointmentsByUser(userIdLong));
        model.addAttribute("notifications", notificationService.getNotificationsByUser(userIdLong));
        model.addAttribute("notificationCount", notificationService.getUnreadNotificationCount(username));
        return "user-dashboard";
    }
    appointmentService.cancelAppointment(appointmentId);
    notificationService.createNotification(
        username,
        "Test drive appointment cancelled for " + appointment.getAppointmentCar().getCarMake() + " " + appointment.getAppointmentCar().getCarModel(),
        NotificationModel.NotificationType.APPOINTMENT_STATUS
    );
    return "redirect:/user/dashboard?appointmentCancelled";
}

// Clears all notifications for the logged-in user
@PostMapping("/user/notifications/clear")
public String clearNotifications(Authentication authentication) {
    String userId = authentication.getName();
    Long userIdLong = Long.parseLong(userId);
    UserModel user = userService.getUserById(userIdLong);
    if (user == null) {
        return "redirect:/login";
    }
    notificationService.clearNotifications(userIdLong);
    return "redirect:/user/dashboard?notificationsCleared";
}

```

Figure 15 UserController.java Part 2

#### 4.2.5 Admin Controller

The **AdminController** class is a Spring MVC controller in a Java application, overseeing administrative tasks such as **user management**, **car listing moderation**, **appointment approvals**, and **report resolutions**. It handles endpoints like /admin/dashboard for displaying admin data, /admin/users/mark-admin for role changes, /admin/cars/activate for approving cars, /admin/appointments/approve for managing test drives, and /admin/reports/resolve for handling user reports, interacting with services like **UserService**, **CarService**, **AppointmentService**, **ReportService**, **NotificationService**, and **AuditLogService**. The controller employs **@Valid** and **BindingResult** for form validation, supports AJAX responses with  **ResponseEntity**, and manages admin authentication via **Authentication**, rendering views like admin-dashboard.html.

```

// Displays the admin dashboard with user, car, appointment, and report data
@GetMapping("/admin/dashboard")
public String adminDashboard(Model model, Authentication authentication) {
    String userId = authentication.getName();
    Long userIdLong = Long.parseLong(userId);
    if (!userService.isAdmin(userIdLong)) {
        return "redirect:/login";
    }
    UserModel user = userService.getUserById(userIdLong);
    if (user == null) {
        return "redirect:/login";
    }
    model.addAttribute("user", user);
    model.addAttribute("users", userService.getAllUsers());
    model.addAttribute("cars", carService.getAllCars());
    model.addAttribute("appointments", appointmentService.getAllAppointments());
    model.addAttribute("reports", reportService.getAllReports());
    model.addAttribute("notificationCount", notificationService.getUnreadNotificationCount(user.getUsername()));
    return "admin-dashboard";
}

// Marks a user as an admin and logs the action
@PostMapping("/admin/users/mark-admin")
public String markUserAsAdmin(@RequestParam("userId") Long userId, Authentication authentication) {
    String adminUserId = authentication.getName();
    Long adminUserIdLong = Long.parseLong(adminUserId);
    if (!userService.isAdmin(adminUserIdLong)) {
        return "redirect:/login";
    }
    try {
        UserModel user = userService.markUserAsAdmin(userId);
        UserModel admin = userService.getUserById(adminUserIdLong);
        auditLogService.logAction(
            admin.getUsername(),
            "Marked user as admin: " + user.getUsername(),
            "User",
            userId
        );
        notificationService.createNotification(
            user.getUsername(),
            "You have been assigned admin privileges.",
            NotificationModel.NotificationType.GENERAL
        );
        return "redirect:/admin/dashboard?adminMarked";
    } catch (Exception e) {
        return "redirect:/admin/dashboard?error=UserNotFound";
    }
}

// Demotes an admin to a regular user and logs the action
@PostMapping("/admin/users/make-user")
public String makeUser(@RequestParam("userId") Long userId, Authentication authentication) {
    String adminUserId = authentication.getName();
    Long adminUserIdLong = Long.parseLong(adminUserId);
    if (!userService.isAdmin(adminUserIdLong)) {
        return "redirect:/login";
    }
    try {
        UserModel user = userService.getUserById(userId);
        if (user == null) {
            return "redirect:/admin/dashboard?error=UserNotFound";
        }
        UserModel admin = userService.getUserById(adminUserIdLong);
        if (admin == null || user.getId().equals(adminUserIdLong)) {
            return "redirect:/admin/dashboard?error=CannotDemoteSelf";
        }
        user.setRole(UserModel.Role.USER);
        user.setUserUpdatedAt(LocalDateTime.now());
        userService.saveUser(user);
        auditLogService.logAction(admin.getUsername(), "Demoted user to USER: " + user.getUsername(), "User",
            userId);
        notificationService.createNotification(user.getUsername(),
            "Your admin privileges have been removed. You are now a regular user.",
            NotificationModel.NotificationType.GENERAL);
        return "redirect:/admin/dashboard?userDemoted";
    } catch (Exception e) {
        return "redirect:/admin/dashboard?error=DemotionFailed";
    }
}

```

Figure 16 AdminController.java Part 1

```

// Deletes a user (except admins) via AJAX and logs the action
@PostMapping("/admin/users/delete")
@ResponseBody
public ResponseEntity<Map<String, String>> deleteUser(@RequestParam("userId") Long userId, Authentication authentication) {
    Map<String, String> response = new HashMap<>();
    String adminUserId = authentication.getName();
    Long adminUserIdLong = Long.parseLong(adminUserId);
    if (!userService.isAdmin(adminUserIdLong)) {
        response.put("status", "error");
        response.put("message", "Unauthorized access.");
        return ResponseEntity.status(403).body(response);
    }
    try {
        UserModel user = userService.getUserById(userId);
        if (user == null) {
            response.put("status", "error");
            response.put("message", "User not found.");
            return ResponseEntity.status(404).body(response);
        }
        if (user.getUserRole() == UserModel.Role.ADMIN) {
            response.put("status", "error");
            response.put("message", "Cannot delete an admin user.");
            return ResponseEntity.status(400).body(response);
        }
        UserModel admin = userService.getUserById(adminUserIdLong);
        userService.deleteUser(userId);
        auditLogService.logAction(admin.getUsername(), "Deleted user: " + user.getUsername(), "User", userId);
        response.put("status", "success");
        response.put("message", "User deleted successfully.");
        return ResponseEntity.ok(response);
    } catch (Exception e) {
        response.put("status", "error");
        response.put("message", "Failed to delete user: " + e.getMessage());
        return ResponseEntity.status(500).body(response);
    }
}

// Edits a user's details via AJAX and logs the action
@PostMapping("/admin/users/edit")
@ResponseBody
public ResponseEntity<Map<String, String>> editUser(@RequestParam("userId") Long userId,
                                                       @RequestParam("username") String username, @RequestParam("email") String email,
                                                       Authentication authentication) {
    Map<String, String> response = new HashMap<>();
    String adminUserId = authentication.getName();
    Long adminUserIdLong = Long.parseLong(adminUserId);
    if (!userService.isAdmin(adminUserIdLong)) {
        response.put("status", "error");
        response.put("message", "Unauthorized access.");
        return ResponseEntity.status(403).body(response);
    }
    try {
        UserModel existingUser = userService.getUserById(userId);
        if (existingUser == null) {
            response.put("status", "error");
            response.put("message", "User not found.");
            return ResponseEntity.status(404).body(response);
        }
        UserModel admin = userService.getUserById(adminUserIdLong);
        existingUser.setUsername(username);
        existingUser.setEmail(email);
        existingUser.setUserUpdatedAt(LocalDateTime.now());
        userService.saveUser(existingUser);
        auditLogService.logAction(admin.getUsername(), "Edited user: " + username, "User", userId);
        notificationService.createNotification(username, "Your profile has been updated by an admin.",
                                              NotificationModel.NotificationType.GENERAL);
        response.put("status", "success");
        response.put("message", "User updated successfully.");
        return ResponseEntity.ok(response);
    } catch (Exception e) {
        response.put("status", "error");
        response.put("message", "Failed to edit user: " + e.getMessage());
        return ResponseEntity.status(500).body(response);
    }
}

```

Figure 17 AdminController.java Part 2

```

// Edits a user's details via AJAX and logs the action
@PostMapping("/admin/users/edit")
@ResponseBody
public ResponseEntity<Map<String, String>> editUser(@RequestParam("userId") Long userId,
    @RequestParam("username") String username, @RequestParam("email") String email,
    Authentication authentication) {
    Map<String, String> response = new HashMap<>();
    String adminUserId = authentication.getName();
    Long adminUserIdLong = Long.parseLong(adminUserId);
    if (!userService.isAdmin(adminUserIdLong)) {
        response.put("status", "error");
        response.put("message", "Unauthorized access.");
        return ResponseEntity.status(403).body(response);
    }
    try {
        UserModel existingUser = userService.getUserById(userId);
        if (existingUser == null) {
            response.put("status", "error");
            response.put("message", "User not found.");
            return ResponseEntity.status(404).body(response);
        }
        UserModel admin = userService.getUserById(adminUserIdLong);
        existingUser.setUsername(username);
        existingUser.setEmail(email);
        existingUser.setUserUpdatedAt(LocalDateTime.now());
        userService.saveUser(existingUser);
        auditLogService.logAction(admin.getUsername(), "Edited user: " + username, "User", userId);
        notificationService.createNotification(username, "Your profile has been updated by an admin.",
            NotificationModel.NotificationType.GENERAL);
        response.put("status", "success");
        response.put("message", "User updated successfully.");
        return ResponseEntity.ok(response);
    } catch (Exception e) {
        response.put("status", "error");
        response.put("message", "Failed to edit user: " + e.getMessage());
        return ResponseEntity.status(500).body(response);
    }
}

```

Figure 18 AdminController.java Part 3

```

// Edits a car's details via AJAX, updates its status, and logs the action
@PostMapping("/admin/cars/edit")
@ResponseBody
public ResponseEntity<Map<String, String>> editCar(@Valid @ModelAttribute("car") CarModel car, BindingResult result,
    @RequestParam("carStatus") String carStatus, Authentication authentication) {
    Map<String, String> response = new HashMap<>();
    String adminUserId = authentication.getName();
    Long adminUserIdLong = Long.parseLong(adminUserId);
    if (!userService.isAdmin(adminUserIdLong)) {
        response.put("status", "error");
        response.put("message", "Unauthorized access.");
        return ResponseEntity.status(403).body(response);
    }
    if (result.hasErrors()) {
        response.put("status", "error");
        response.put("message", "Invalid input data.");
        return ResponseEntity.badRequest().body(response);
    }
    try {
        CarModel existingCar = carService.getCarById(car.getCarId());
        if (existingCar == null) {
            response.put("status", "error");
            response.put("message", "Car not found.");
            return ResponseEntity.status(404).body(response);
        }
        // Validate status transition (e.g., can't go from SOLD to ACTIVE)
        CarModel.CarStatus newStatus = CarModel.CarStatus.valueOf(carStatus);
        if (existingCar.getCarStatus() == CarModel.CarStatus.SOLD && newStatus != CarModel.CarStatus.SOLD) {
            response.put("status", "error");
            response.put("message", "Cannot change status of a sold car.");
            return ResponseEntity.status(400).body(response);
        }
        UserModel admin = userService.getUserById(adminUserIdLong);
        existingCar.setCarMake(car.getCarMake());
        existingCar.setCarModel(car.getCarModel());
        existingCar.setRegistrationYear(car.getRegistrationYear());
        existingCar.setCarPrice(car.getCarPrice());
        existingCar.setCarMileage(car.getCarMileage());
        existingCar.setFuelType(car.getFuelType());
        existingCar.setTransmissionType(car.getTransmissionType());
        existingCar.setCarStatus(newStatus); // Update carStatus
        existingCar.setCarUpdatedAt(LocalDateTime.now());
        carService.updateCar(existingCar);
        auditLogService.logAction(admin.getUsername(), "Edited car: " + car.getCarMake() + " " + car.getCarModel(),
            "Car", car.getCarId());
        notificationService.createNotification(existingCar.getCarUser().getUsername(),
            "Your car listing has been updated by an admin: " + car.getCarMake() + " " + car.getCarModel(),
            NotificationModel.NotificationType.GENERAL);
        response.put("status", "success");
        response.put("message", "Car updated successfully.");
        return ResponseEntity.ok(response);
    } catch (IllegalArgumentException e) {
        response.put("status", "error");
        response.put("message", "Invalid status value: " + carStatus);
        return ResponseEntity.status(400).body(response);
    } catch (Exception e) {
        response.put("status", "error");
        response.put("message", "Failed to edit car: " + e.getMessage());
        return ResponseEntity.status(500).body(response);
    }
}

```

Figure 19 AdminController.java Part 4

```

// Deletes a car via AJAX, notifies the owner, and logs the action
@PostMapping("/admin/cars/delete")
@ResponseBody
public ResponseEntity<Map<String, String>> deleteCar(@RequestParam("carId") Long carId,
    @RequestParam("reason") String reason, Authentication authentication) {
    Map<String, String> response = new HashMap<>();
    String adminUserId = authentication.getName();
    Long adminUserIdLong = Long.parseLong(adminUserId);
    if (!userService.isAdmin(adminUserIdLong)) {
        response.put("status", "error");
        response.put("message", "Unauthorized access.");
        return ResponseEntity.status(403).body(response);
    }
    try {
        CarModel car = carService.getCarById(carId);
        if (car == null) {
            response.put("status", "error");
            response.put("message", "Car not found.");
            return ResponseEntity.status(404).body(response);
        }
        UserModel admin = userService.getUserById(adminUserIdLong);
        String carOwnerUsername = car.getCarUser().getUsername();
        carService.deleteCar(carId);
        auditLogService.logAction(admin.getUsername(),
            "Deleted car: " + car.getCarMake() + " " + car.getCarModel() + " (Reason: " + reason + ")", "Car",
            carId);
        notificationService.createNotification(carOwnerUsername, "Admin has removed this car. Reason: " + reason,
            NotificationModel.NotificationType.GENERAL);
        response.put("status", "success");
        response.put("message", "Car deleted successfully.");
        return ResponseEntity.ok(response);
    } catch (Exception e) {
        response.put("status", "error");
        response.put("message", "Failed to delete car: " + e.getMessage());
        return ResponseEntity.status(500).body(response);
    }
}

// Activates a car listing if the price is reasonable and logs the action
@PostMapping("/admin/cars/activate")
public String activateCar(@RequestParam("carId") Long carId, Authentication authentication) {
    System.out.println("Activating car with ID: " + carId);
    String adminUserId = authentication.getName();
    Long adminUserIdLong = Long.parseLong(adminUserId);
    if (!userService.isAdmin(adminUserIdLong)) {
        System.out.println("User is not an admin: " + adminUserIdLong);
        return "redirect:/login";
    }
    try {
        CarModel car = carService.getCarById(carId);
        if (car == null) {
            System.out.println("Car not found: " + carId);
            return "redirect:/admin/dashboard?error=CarNotFound";
        }
        if (car.getCarPrice() < 2000) {
            System.out.println("Unreasonable price for car ID " + carId + ": " + car.getCarPrice());
            return "redirect:/admin/dashboard?error=UnreasonablePrice&price=" + car.getCarPrice();
        }
        UserModel admin = userService.getUserById(adminUserIdLong);
        carService.activateCar(carId);
        auditLogService.logAction(admin.getUsername(),
            "Approved car listing: " + car.getCarMake() + " " + car.getCarModel(), "Car", carId);
        notificationService.createNotification(car.getCarUser().getUsername(),
            "Your car listing has been approved: " + car.getCarMake() + " " + car.getCarModel(),
            NotificationModel.NotificationType.GENERAL);
        System.out.println("Car activated successfully: " + carId);
        return "redirect:/admin/dashboard?carApproved";
    } catch (Exception e) {
        System.out.println("Error activating car ID " + carId + ": " + e.getMessage());
        return "redirect:/admin/dashboard?error=CarNotFound";
    }
}

```

Figure 20 AdminController.java Part 5

```

// Deactivates a car listing and logs the action
@PostMapping("/admin/cars/deactivate")
public String deactivateCar(@RequestParam("carId") Long carId, Authentication authentication) {
    String adminUserId = authentication.getName();
    Long adminUserIdLong = Long.parseLong(adminUserId);
    if (!userService.isAdmin(adminUserIdLong)) {
        return "redirect:/login";
    }
    try {
        CarModel car = carService.deactivateCar(carId);
        UserModel admin = userService.getUserById(adminUserIdLong);
        auditLogService.logAction(admin.getUsername(),
            "Deactivated car: " + car.getCarMake() + " " + car.getCarModel(), "Car", carId);
        notificationService.createNotification(car.getCarUser().getUsername(),
            "Your car has been deactivated: " + car.getCarMake() + " " + car.getCarModel(),
            NotificationModel.NotificationType.GENERAL);
        return "redirect:/admin/dashboard?carDeactivated";
    } catch (Exception e) {
        return "redirect:/admin/dashboard?error=CarNotFound";
    }
}

// Marks a car as sold and logs the action
@PostMapping("/admin/cars/sell")
public String sellCar(@RequestParam("carId") Long carId, Authentication authentication) {
    String adminUserId = authentication.getName();
    Long adminUserIdLong = Long.parseLong(adminUserId);
    if (!userService.isAdmin(adminUserIdLong)) {
        return "redirect:/login";
    }
    try {
        CarModel car = carService.transactSale(carId);
        UserModel admin = userService.getUserById(adminUserIdLong);
        auditLogService.logAction(admin.getUsername(),
            "Marked car as sold: " + car.getCarMake() + " " + car.getCarModel(), "Car", carId);
        notificationService.createNotification(car.getCarUser().getUsername(),
            "Your car has been sold: " + car.getCarMake() + " " + car.getCarModel(),
            NotificationModel.NotificationType.GENERAL);
        return "redirect:/admin/dashboard?carSold";
    } catch (Exception e) {
        return "redirect:/admin/dashboard?error=CarNotFound";
    }
}

// Approves a test drive appointment and logs the action
@PostMapping("/admin/appointments/approve")
public String approveAppointment(@RequestParam("appointmentId") Long appointmentId, Authentication authentication) {
    String adminUserId = authentication.getName();
    Long adminUserIdLong = Long.parseLong(adminUserId);
    if (!userService.isAdmin(adminUserIdLong)) {
        return "redirect:/login";
    }
    try {
        AppointmentModel appointment = appointmentService.approveAppointment(appointmentId);
        UserModel admin = userService.getUserById(adminUserIdLong);
        auditLogService.logAction(admin.getUsername(),
            "Approved appointment for car ID: " + appointment.getAppointmentCar().getCarId(), "Appointment",
            appointmentId);
        notificationService.createNotification(appointment.getAppointmentUser().getUsername(),
            "Your test drive appointment has been approved.",
            NotificationModel.NotificationType.APPOINTMENT_STATUS);
        return "redirect:/admin/dashboard?appointmentApproved";
    } catch (Exception e) {
        return "redirect:/admin/dashboard?error=AppointmentNotFound";
    }
}

```

Figure 21 AdminController.java Part 6

```

// Denies a test drive appointment and logs the action
@PostMapping("/admin/appointments/deny")
public String denyAppointment(@RequestParam("appointmentId") Long appointmentId, Authentication authentication) {
    String adminUserId = authentication.getName();
    Long adminUserIdLong = Long.parseLong(adminUserId);
    if (!userService.isAdmin(adminUserIdLong)) {
        return "redirect:/login";
    }
    try {
        AppointmentModel appointment = appointmentService.denyAppointment(appointmentId);
        UserModel admin = userService.getUserById(adminUserIdLong);
        auditLogService.logAction(admin.getUsername(),
            "Denied appointment for car ID: " + appointment.getAppointmentCar().getCarId(), "Appointment",
            appointmentId);
        notificationService.createNotification(appointment.getAppointmentUser().getUsername(),
            "Your test drive appointment has been denied.",
            NotificationModel.NotificationType.APPOINTMENT_STATUS);
        return "redirect:/admin/dashboard?appointmentDenied";
    } catch (Exception e) {
        return "redirect:/admin/dashboard?error=AppointmentNotFound";
    }
}

// Shows the admin profile form within the dashboard
@GetMapping("/admin/profile")
public String showAdminProfileForm(Model model, Authentication authentication) {
    String userId = authentication.getName();
    Long userIdLong = Long.parseLong(userId);
    if (!userService.isAdmin(userIdLong)) {
        return "redirect:/login";
    }
    UserModel user = userService.getUserById(userIdLong);
    if (user == null) {
        return "redirect:/login";
    }
    model.addAttribute("user", user);
    model.addAttribute("users", userService.getAllUsers());
    model.addAttribute("cars", carService.getAllCars());
    model.addAttribute("appointments", appointmentService.getAllAppointments());
    model.addAttribute("reports", reportService.getAllReports());
    model.addAttribute("notificationCount", notificationService.getUnreadNotificationCount(user.getUsername()));
    return "admin-dashboard";
}

```

Figure 22 AdminController.java Part 7

```

// Updates the admin's profile and logs the action
@PostMapping("/admin/profile")
public String updateAdminProfile(@Valid @ModelAttribute("user") UserModel user, BindingResult result, Model model,
        Authentication authentication) {
    String userId = authentication.getName();
    Long userIdLong = Long.parseLong(userId);
    if (!userService.isAdmin(userIdLong)) {
        return "redirect:/login";
    }
    UserModel loggedinUser = userService.getUserById(userIdLong);
    if (loggedinUser == null) {
        return "redirect:/login";
    }
    String username = loggedinUser.getUsername();
    if (result.hasErrors()) {
        model.addAttribute("users", userService.getAllUsers());
        model.addAttribute("cars", carService.getAllCars());
        model.addAttribute("appointments", appointmentService.getAllAppointments());
        model.addAttribute("reports", reportService.getAllReports());
        model.addAttribute("notificationCount", notificationService.getUnreadNotificationCount(username));
        return "admin-dashboard";
    }
    try {
        userService.updateUserProfile(user, username);
        auditLogService.logAction(username, "Updated admin profile", "User", user.getUserId());
        notificationService.createNotification(username, "Profile updated successfully",
                NotificationModel.NotificationType.GENERAL);
        return "redirect:/admin/dashboard?profileUpdated";
    } catch (Exception e) {
        model.addAttribute("error", "Failed to update profile: " + e.getMessage());
        model.addAttribute("users", userService.getAllUsers());
        model.addAttribute("cars", carService.getAllCars());
        model.addAttribute("appointments", appointmentService.getAllAppointments());
        model.addAttribute("reports", reportService.getAllReports());
        model.addAttribute("notificationCount", notificationService.getUnreadNotificationCount(username));
        return "admin-dashboard";
    }
}

// Resolves a user-submitted report and logs the action
@PostMapping("/admin/reports/resolve")
public String resolveReport(@RequestParam("reportId") Long reportId, Authentication authentication) {
    String userId = authentication.getName();
    Long userIdLong = Long.parseLong(userId);
    if (!userService.isAdmin(userIdLong)) {
        return "redirect:/login";
    }
    try {
        ReportModel report = reportService.resolveReport(reportId);
        UserModel admin = userService.getUserById(userIdLong);
        auditLogService.logAction(admin.getUsername(),
                "Resolved report for car ID: " + report.getReportCar().getCarId(), "Report", reportId);
        notificationService.createNotification(report.getReportUser().getUsername(),
                "Your report has been resolved.", NotificationModel.NotificationType.GENERAL);
        return "redirect:/admin/dashboard?reportResolved";
    } catch (Exception e) {
        return "redirect:/admin/dashboard?error=ReportNotFound";
    }
}

```

Figure 23 AdminController.java Part 8

### 4.3 View Structure

The view structure consists total of 12 Html Pages shown in below

#### 4.3.1 Home Page

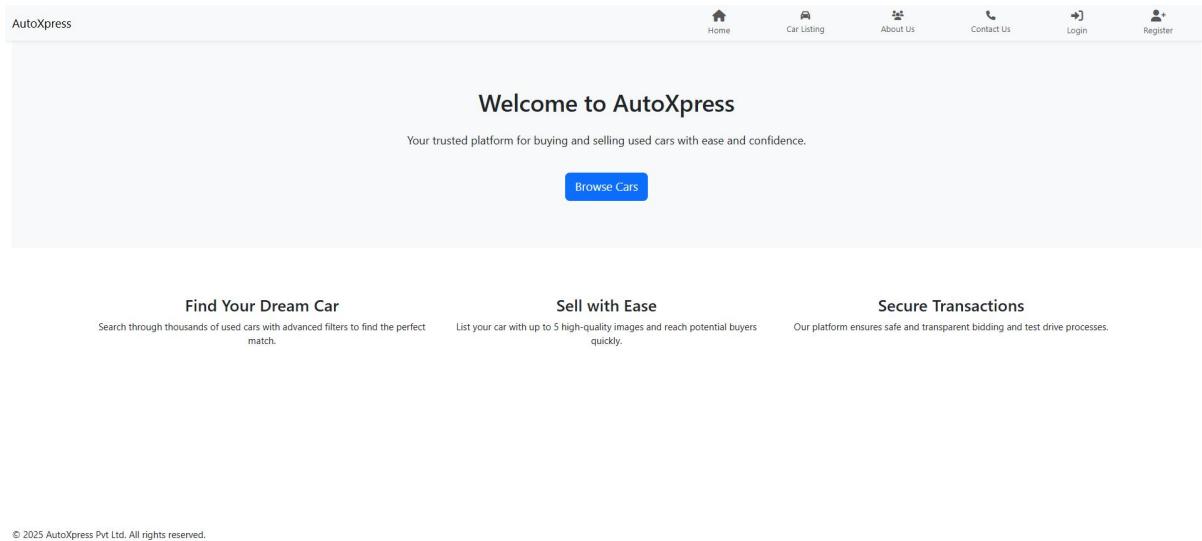


Figure 24 home.html

#### 4.3.2 Login Page

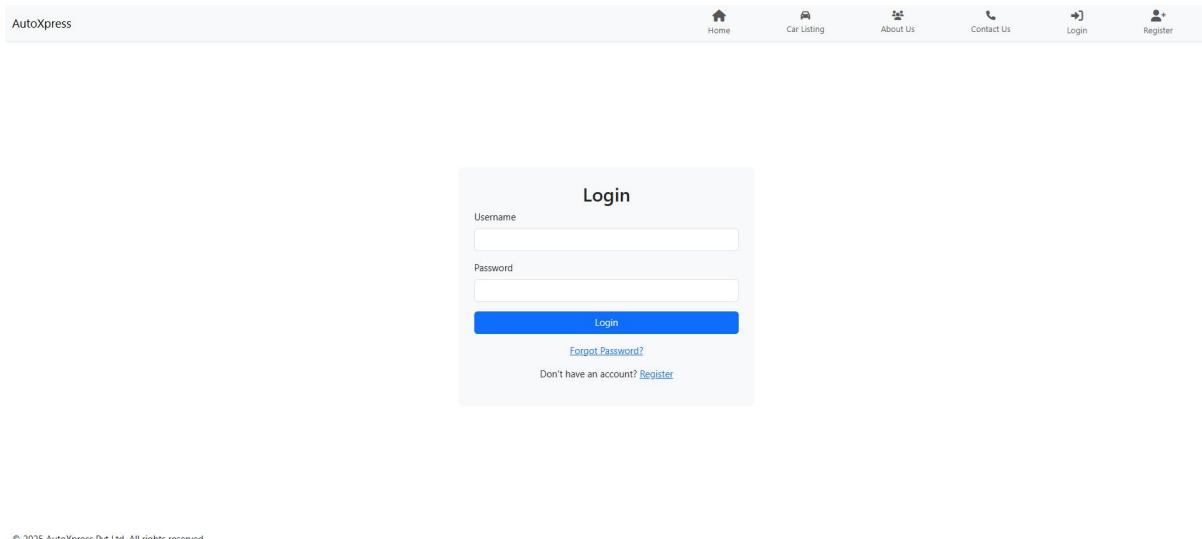


Figure 25 login.html

### 4.3.3 Register Page

The screenshot shows the AutoXpress website's header with links for Home, Car Listing, About Us, Contact Us, Login, and Register. Below the header is a registration form titled "Register". The form has three input fields: "Username", "Email", and "Password", each with a placeholder text above it. A blue "Register" button is centered below the password field. At the bottom of the form, there is a link "Already have an account? [Log in](#)".

© 2025 AutoXpress Pvt Ltd. All rights reserved.

Figure 26 register.html

### 4.3.4 Forget Password Page

The screenshot shows the AutoXpress website's header with links for Home, Car Listing, About Us, Contact Us, Login, and Register. Below the header is a password reset form titled "Password Reset Request". It has one input field for "Email Address" with a placeholder text above it. A blue "Request Password Reset" button is centered below the email field. At the bottom of the form, there are two links: "Remembered your password? [Log in](#)" and "Don't have an account? [Register](#)".

© 2025 AutoXpress Pvt Ltd. All rights reserved.

Figure 27 reset-password-request.html

#### 4.3.5 Reset Password Page

The screenshot shows the 'Reset Password' page of the AutoXpress website. At the top, there is a navigation bar with icons for Home, Car Listing, About Us, Contact Us, Login, and Register. The main content area has a light gray background and features a central 'Reset Password' form. The form has two input fields: 'New Password' and 'Confirm New Password', both with placeholder text. Below these fields is a blue rectangular button labeled 'Reset Password'. Underneath the button, there is small text that says 'Remembered your password? [Log in](#)' and 'Don't have an account? [Register](#)'. The bottom left corner of the page contains a small copyright notice: '© 2025 AutoXpress Pvt Ltd. All rights reserved.'

Figure 28 reset-password.html

#### 4.3.6 Car Catalog Page

The Car Filters are in the position of **sticky**, which means the **filter** will always be shown in the same position when u scroll vertically of the page

AutoXpress

Home Car Listing About Us Contact Us Login Register

### Car Listings

Filter Cars

Make

Model

Year

Min Price

Max Price

Max Mileage

Fuel Type  Any

Transmission  Any

© 2025 AutoXpress Pvt Ltd. All rights reserved.

**Toyota Camry**  
Price: \$25000  
Mileage: 40000 miles  
Fuel Type: HYBRID  
Transmission: AUTOMATIC

**Honda Civic**  
Price: \$100000  
Mileage: 40000 miles  
Fuel Type: DIESEL  
Transmission: MANUAL

**BMW x5**  
Price: \$300000  
Mileage: 35000 miles  
Fuel Type: PETROL  
Transmission: AUTOMATIC

**Toyota Corolla**  
Price: \$50000  
Mileage: 20000 miles  
Fuel Type: HYBRID  
Transmission: MANUAL

**Tesla X**  
Price: \$90000  
Mileage: 200000 miles  
Fuel Type: ELECTRIC  
Transmission: AUTOMATIC

Previous  Next

Figure 29 car-listing.html

### 4.3.7 Car Detail Page

The screenshot shows the car detail page for a Toyota Camry. At the top, there's a navigation bar with links for Home, Car Listing, Car Post, About Us, Contact Us, My Profile, and Logout. Below the navigation is a section titled "Car Details" featuring a black Toyota Camry. To the right of the car, the model is identified as "Toyota Camry" and the year as "2019". Detailed specifications listed include Price: \$25000.0, Mileage: 40000 miles, Fuel Type: HYBRID, Transmission: AUTOMATIC, Status: ACTIVE, and Posted By: Wen Ping. On the left, there's a "Book a Test Drive" form with fields for Appointment Date (mm/dd/yyyy) and a "Book Appointment" button. On the right, there's a "Report This Car" form with fields for Reason (dropdown menu), Description (text area), and a "Submit Report" button. At the bottom left, a copyright notice reads "© 2025 AutoXpress Pvt Ltd. All rights reserved."

Figure 30 car-details.html

### 4.3.8 About Us Page

The screenshot shows the "About Us" page. At the top, there's a navigation bar with links for Home, Car Listing, Car Post, About Us, Contact Us, My Profile, and Logout. The main content area has a blue header with the title "About AutoXpress" and the subtitle "Your Trusted Platform for Buying and Selling Cars". Below this, there's a section titled "Our Mission" with a paragraph about the platform's dedication to simplifying car buying and selling. There's also a "Why Choose AutoXpress?" section with three boxes: "Easy Car Listing" (with a car icon), "Seamless Test Drives" (with a checkmark icon), and "Secure Transactions" (with a shield icon). At the bottom left, a copyright notice reads "© 2025 AutoXpress Pvt Ltd. All rights reserved."

Figure 31 about.html

#### 4.3.9 Contact Us Page

The screenshot shows the 'Contact Us' page of the AutoXpress website. At the top, there's a navigation bar with links for Home, Car Listing, Car Post, About Us, Contact Us, My Profile, and Logout. Below the navigation is a blue header bar with the text 'Contact Us' and a subtext 'We're Here to Help You with Your Car Trading Needs'. On the left, there's a 'Get in Touch' section containing contact information: Email (support@autoxpress.com), Phone (+1 (555) 123-4567), and Address (123 Auto Lane, Car City, CA 90210). On the right, there's a 'Send Us a Message' form with fields for Name, Email, Subject, and Message, followed by a 'Send Message' button.

Figure 32 contact.html

#### 4.3.10 Profile Page

The screenshot shows the 'User Dashboard' page of the AutoXpress website. At the top, there's a navigation bar with links for Home, Car Listing, Car Post, About Us, Contact Us, My Profile, and Logout. Below the navigation is a sidebar with links for Profile Update, Posted Cars, Booked Appointments, and Notifications. The main content area is titled 'Profile Update' and contains fields for Username (Wen Ping) and Email (flazeping@gmail.com), with a 'Update Profile' button at the bottom. At the bottom of the page, there's a copyright notice: '© 2025 AutoXpress Pvt Ltd. All rights reserved.'

Figure 33 user-dashboard.html

#### 4.3.10.1 Car Posted

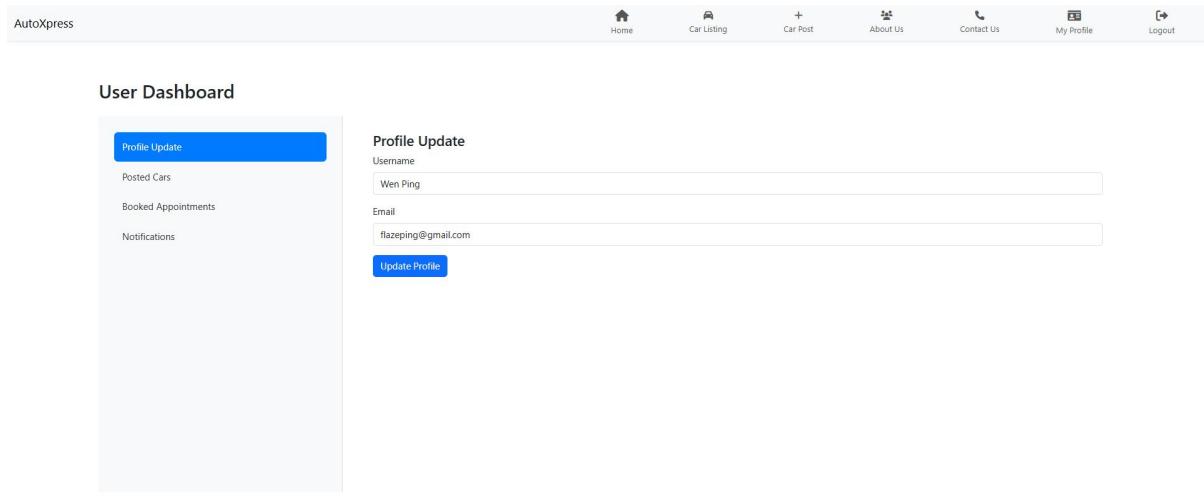


Figure 34 user-dashboard.html Posted Car

#### 4.3.10.2 Appointment Booked

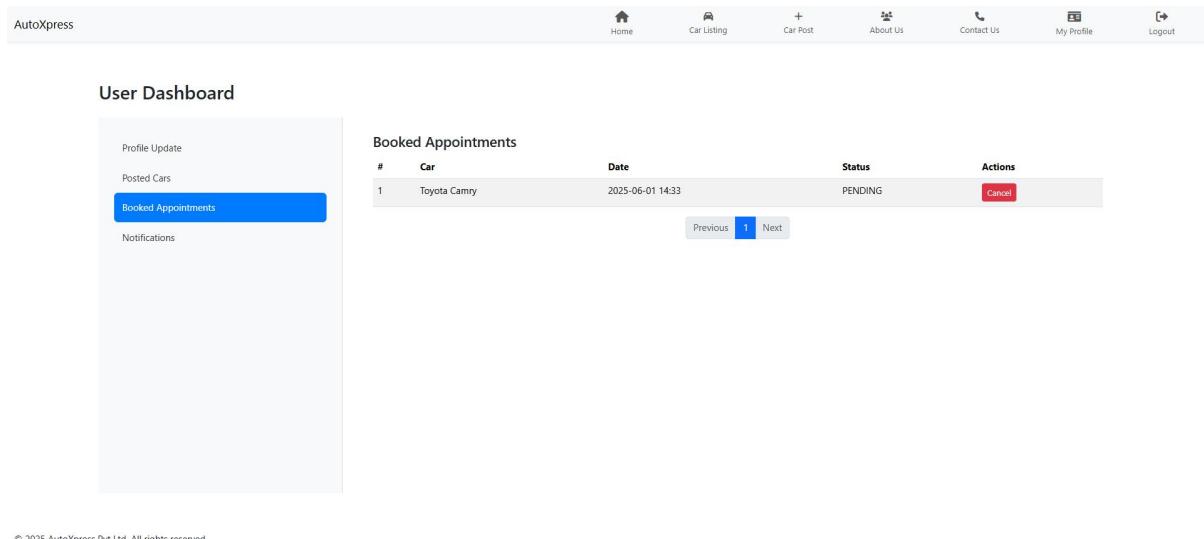


Figure 35 user-dashboard.html Appointments

#### 4.3.10.3 User Notificaiton

The screenshot shows the 'User Dashboard' page of the AutoXpress application. At the top, there is a navigation bar with links for Home, Car Listing, Car Post, About Us, Contact Us, My Profile, and Logout. Below the navigation bar, the main content area is titled 'User Dashboard'. On the left, there is a sidebar with links for Profile Update, Posted Cars, Booked Appointments, and Notifications (which is highlighted with a blue background). The main content area is titled 'Notifications' and contains a table with two rows of data:

#	Message	Type	Date
1	Welcome to AutoXpress! Your account has been created.	GENERAL	2025-05-31 12:33:12
2	Test drive appointment booked for Toyota Camry	APPOINTMENT_STATUS	2025-05-31 12:33:33

At the bottom of the notifications table, there are buttons for 'Previous', '1', and 'Next'.

© 2025 AutoXpress Pvt Ltd. All rights reserved.

Figure 36 user-dashboard.html Notification

#### 4.3.11 Car Posting Page

The screenshot shows the 'Post a Car for Sale' page of the AutoXpress application. At the top, there is a navigation bar with links for Home, Car Listing, Car Post, About Us, Contact Us, My Profile, and Logout. The main content area is titled 'Post a Car for Sale'. It features a form with various input fields:

- Make \* (input field)
- Model \* (input field)
- Registration Year \* (input field)
- Price (\$) \* (input field)
- Mileage (miles, optional) (input field)
- Fuel Type (optional) (dropdown menu)
- Transmission (optional) (dropdown menu)
- Images (1-5, each <10MB) \* (input field)
- Choose Files | No file chosen (button)
- Upload 1 to 5 images in any image format.

At the bottom of the form, there is a blue 'Post Car' button.

Figure 37 post-car.html

#### 4.3.12 Admin Dashboard

The screenshot shows the Admin Dashboard interface. At the top, there is a navigation bar with links: Home, Car Listing, Car Post, About Us, Contact Us, Admin Dashboard, and Logout. Below the navigation bar, the title "Admin Dashboard" is displayed. On the left, a sidebar menu includes "Profile" (which is currently selected and highlighted in blue), "Manage Users", "Manage Cars", "Manage Appointments", and "Manage Reports". The main content area is titled "Admin Profile" and contains fields for "Username" (admin\_user) and "Email" (admin@example.com). A blue "Update Profile" button is located at the bottom of this section. At the bottom of the page, a copyright notice reads: "© 2025 AutoXpress Pvt Ltd. All rights reserved."

Figure 38 admin-dashboard.html

##### 4.3.12.1 User Management

The screenshot shows the Admin Dashboard interface, specifically the "Manage Users" section. At the top, there is a navigation bar with links: Home, Car Listing, Car Post, About Us, Contact Us, Admin Dashboard, and Logout. Below the navigation bar, the title "Admin Dashboard" is displayed. On the left, a sidebar menu includes "Profile" (highlighted in blue), "Manage Users" (which is currently selected and highlighted in blue), "Manage Cars", "Manage Appointments", and "Manage Reports". The main content area is titled "Manage Users" and displays a table of user information. The table has columns: #, Username, Email, Role, and Action. The data in the table is as follows:

#	Username	Email	Role	Action
1	admin_user	admin@example.com	ADMIN	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Make Admin</a>
2	Wen Ping	flazeping@gmail.com	USER	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Make Admin</a>
3	john_doe	john.doe@example.com	USER	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Make Admin</a>

At the bottom of the page, a copyright notice reads: "© 2025 AutoXpress Pvt Ltd. All rights reserved."

Figure 39 admin-dashboard.html User Management

#### 4.3.12.2 Car Management

The screenshot shows the Admin Dashboard for AutoXpress. At the top, there is a navigation bar with links for Home, Car Listing, Car Post, About Us, Contact Us, Admin Dashboard, and Logout. Below the navigation bar, the title "Admin Dashboard" is displayed. On the left, a sidebar menu includes Profile, Manage Users, **Manage Cars** (which is highlighted with a blue background), Manage Appointments, and Manage Reports. The main content area is titled "Manage Car Posts" and contains a table with the following data:

#	Make	Model	Year	Price	Status	Actions
1	Toyota	Camry	2019	\$25000.0	ACTIVE	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Deactivate</a> <a href="#">Mark as Sold</a>
2	Honda	Civic	2018	\$10000.0	ACTIVE	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Deactivate</a> <a href="#">Mark as Sold</a>
3	BMW	x5	2019	\$30000.0	ACTIVE	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Deactivate</a> <a href="#">Mark as Sold</a>
4	Toyota	Corolla	2022	\$5000.0	INACTIVE	<a href="#">Edit</a> <a href="#">Delete</a>
5	Tesla	X	2024	\$90000.0	PENDING	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Approve</a>

At the bottom left, a copyright notice reads "© 2025 AutoXpress Pvt Ltd. All rights reserved."

Figure 40 admin-dashboard.html Car Management

#### 4.3.12.3 Appointment Management

The screenshot shows the Admin Dashboard for AutoXpress. At the top, there is a navigation bar with links for Home, Car Listing, Car Post, About Us, Contact Us, Admin Dashboard, and Logout. Below the navigation bar, the title "Admin Dashboard" is displayed. On the left, a sidebar menu includes Profile, Manage Users, Manage Cars, **Manage Appointments** (which is highlighted with a blue background), and Manage Reports. The main content area is titled "Manage Test Drive Appointments" and contains a table with the following data:

#	User	Car	Date	Status	Actions
1	john_doe	Toyota Camry	2025-06-01 14:33	PENDING	<a href="#">Approve</a> <a href="#">Deny</a>

At the bottom left, a copyright notice reads "© 2025 AutoXpress Pvt Ltd. All rights reserved."

Figure 41 admin-dashboard.html Appointment Management

#### 4.3.12.4 Reports Management

The screenshot shows the Admin Dashboard of the AutoXpress application. At the top, there is a navigation bar with links for Home, Car Listing, Car Post, About Us, Contact Us, Admin Dashboard, and Logout. The main content area is titled "Admin Dashboard". On the left, a sidebar titled "Admin Dashboard" contains links for Profile, Manage Users, Manage Cars, Manage Appointments, and a prominent blue button labeled "Manage Reports". The main content area is titled "Manage Reports" and displays a table with one row of data. The table columns are: #, User, Car, Reason, Description, Status, and Action. The data row is: 1, john\_doe, BMW x5, MISLEADING\_INFO, Testing, PENDING, and a blue "Resolve" button.

#	User	Car	Reason	Description	Status	Action
1	john_doe	BMW x5	MISLEADING_INFO	Testing	PENDING	<button>Resolve</button>

© 2025 AutoXpress Pvt Ltd. All rights reserved.

Figure 42 admin-dashboard.html Reports Management

## Task 5 CRUD functionalities Showcase

### 5.1 Create functionalities

#### 5.1.1 Register in the Portal

Register with user data below,

Username : Big Cat

Email : bigcat@example.com

Password : abc123

user_id	username	user_password	email	user_role	user_created_at	user_updated_at
11	admin_user	\$2a\$10\$7dxYKBK2NWIdQ/PtJOfFJO7ch/kUKRjXbk438PNI6Yv...	admin@example.com	ADMIN	2025-05-31 04:20:06	2025-05-31 12:20:29
12	Wen Ping	\$2a\$10\$7xnKuJ0fTnSRkuxcUCBRBOc39Zq95topaKg/.FMMUUQ...	flazeping@gmail.com	USER	2025-05-31 04:20:22	2025-05-31 04:20:22
13	john_doe	\$2a\$10\$9m3kQvH0ZYFSIHU67oSgePXWZ20l1/bBqJZhACi9MJ...	john.doe@example.com	USER	2025-05-31 04:33:12	2025-05-31 04:33:12

cted: Edit Copy Delete Export

Figure 43 Database before registration

### Register

Username

Email

Password

Already have an account? [Log in](#)

Figure 44 register.html with data

user_id	username	user_password	email	user_role	user_created_at	user_updated_at
11	admin_user	\$2a\$10\$7dxYKBK2NWIdQ/PtJOfFJO7ch/kUKRjXbk438PNI6Yv...	admin@example.com	ADMIN	2025-05-31 04:20:06	2025-05-31 12:20:29
12	Wen Ping	\$2a\$10\$7xnKuJ0fTnSRkuxcUCBRBOc39Zq95topaKg/.FMMUUQ...	flazeping@gmail.com	USER	2025-05-31 04:20:22	2025-05-31 04:20:22
13	john_doe	\$2a\$10\$9m3kQvH0ZYFSIHU67oSgePXWZ20l1/bBqJZhACi9MJ...	john.doe@example.com	USER	2025-05-31 04:33:12	2025-05-31 04:33:12
14	Big Cat	\$2a\$10\$kA.55TSOW/wu3FEdmNwH3uT1mbPZyF74Zl8lmEkFt0...	bigcat@example.com	USER	2025-05-31 04:51:39	2025-05-31 04:51:39

cted: Edit Copy Delete Export

Figure 45 Database after registration

### 5.1.2 Post a Car for Sale with Picture upload

Post a Car with Picture upload and bidding price,

Make : Cat

Model : Oren

Registration Year : 2025

Price : 25000

Mileage : 10000

Fuel Type : Electric

Transmission : Automatic

Images : carcat.jpg

After Posting, the car require to be approve to shown in Catalog Page

**Post a Car for Sale**

Make \*  Model \*

Registration Year \*

Price (\$) \*

Mileage (miles, optional)  Fuel Type (optional)

Transmission (optional)

Images (1-5, each <10MB) \*   
Upload 1 to 5 images in any image format.

Figure 46 Html Page before Posting Car

## User Dashboard

Profile Update

Posted Cars

Booked Appointments

Notifications

#	Make	Model	Year	Price	Status
1	Cat	Oren	2025	\$25000.0	PENDING

Figure 47 Html Page after Car Posting

### 5.1.3 Book an appointment for Test Drive

Book Appointment for Test Drive on 'Cat Oren',

Date Time : 17 June 2025 05:00 PM

AutoXpress

Home Car Listing + Car Post About Us Contact Us My Profile Logout

### Car Details



**Cat Oren**  
Year: 2025  
Price: \$25000.0  
Mileage: 10000 miles  
Fuel Type: ELECTRIC  
Transmission: AUTOMATIC  
Status: ACTIVE  
Posted By: Big Cat

#### Actions

Book a Test Drive

Appointment Date: 06/17/2025 05:00 PM

**Book Appointment**

#### Report This Car

Reason: Select a reason

Description:

**Submit Report**

© 2025 AutoXpress Pvt Ltd. All rights reserved.

Figure 48 Html Page before making appointment

AutoXpress

Home Car Listing + Car Post About Us Contact Us My Profile Logout

### Car Details



**Cat Oren**

**Year:** 2025  
**Price:** \$25000.0  
**Mileage:** 10000 miles  
**Fuel Type:** ELECTRIC  
**Transmission:** AUTOMATIC  
**Status:** ACTIVE  
**Posted By:** Big Cat

**Actions**

**Book a Test Drive**

Appointment Date: mm/dd/yyyy --:-- --

**Book Appointment**

Appointment booked successfully!

**Report This Car**

Reason: Select a reason

Description:

Submit Report

© 2025 AutoXpress Pvt Ltd. All rights reserved.

Figure 49 Html Page after making appointment

### User Dashboard

Profile Update  
Posted Cars  
**Booked Appointments**  
Notifications

Booked Appointments				
#	Car	Date	Status	Actions
1	Cat Oren	2025-06-17 17:00	PENDING	<a href="#">Cancel</a>

Previous 1 Next

Figure 50 User Dashboard of Booked Appointments

### 5.1.4 Report a Car (Additional Feature)

Report the car 'Cat Oren' for,

Reason : Misleading Info

Description : WHAT THE HELL IS CAT CAR ????????????

AutoXpress

Home Car Listing + Car Post About Us Contact Us My Profile Logout

**Car Details**

**Cat Oren**

**Year:** 2025  
**Price:** \$25000.0  
**Mileage:** 10000 miles  
**Fuel Type:** ELECTRIC  
**Transmission:** AUTOMATIC  
**Status:** ACTIVE  
**Posted By:** Big Cat

**Actions**

Book a Test Drive  
Appointment Date  
mm/dd/yyyy --:-- --  
Book Appointment

**Report This Car**

Reason  
MISLEADING\_INFO  
Description  
WHAT THE HELL IS CAT CAR ????????????

Submit Report

© 2025 AutoXpress Pvt Ltd. All rights reserved.

Figure 51 Html Page before making report

AutoXpress

Home Car Listing + Car Post About Us Contact Us My Profile Logout

**Car Details**

**Cat Oren**

**Year:** 2025  
**Price:** \$25000.0  
**Mileage:** 10000 miles  
**Fuel Type:** ELECTRIC  
**Transmission:** AUTOMATIC  
**Status:** ACTIVE  
**Posted By:** Big Cat

**Actions**

Book a Test Drive  
Appointment Date  
mm/dd/yyyy --:-- --  
Book Appointment

**Report This Car**

Select a reason  
Description  
Report submitted successfully!

© 2025 AutoXpress Pvt Ltd. All rights reserved.

Figure 52 Html Page after making report

## Admin Dashboard

The screenshot shows the Admin Dashboard interface. On the left, there is a sidebar with links: Profile, Manage Users, Manage Cars, Manage Appointments, and a prominent blue button labeled "Manage Reports". The main area is titled "Manage Reports" and contains a table with two rows of data. The columns are labeled #, User, Car, Reason, Description, Status, and Action. Row 1: #1, User john\_doe, Car BMW x5, Reason MISLEADING\_INFO, Description Testing, Status PENDING, Action Resolve (button). Row 2: #2, User john\_doe, Car Cat Oren, Reason MISLEADING\_INFO, Description WHAT THE HELL IS CAT CAR ????????????, Status PENDING, Action Resolve (button).

#	User	Car	Reason	Description	Status	Action
1	john_doe	BMW x5	MISLEADING_INFO	Testing	PENDING	<button>Resolve</button>
2	john_doe	Cat Oren	MISLEADING_INFO	WHAT THE HELL IS CAT CAR ????????????	PENDING	<button>Resolve</button>

Figure 53 Admin Dashboard after Reports

## 5.2 Read functionalities

### 5.2.1 login with registered account

Login with previously registered account,

Username : Big Cat

Password : abc123

The screenshot shows a login form titled "Login". It has fields for "Username" (Big Cat) and "Password" (abc123). Below the password field is a "Login" button. At the bottom of the form, there are links for "Forgot Password?" and "Don't have an account? [Register](#)".

Figure 54 Login with data

The screenshot shows the User Dashboard. At the top, there is a navigation bar with links: Home, Car Listing, Car Post, About Us, Contact Us, My Profile, and Logout. Below the navigation bar, the title "User Dashboard" is displayed. On the left, there is a sidebar with a blue button labeled "Profile Update" and links for Posted Cars, Booked Appointments, and Notifications. The main area is titled "Profile Update" and contains fields for "Username" (Big Cat) and "Email" (bigcat@example.com), with a "Update Profile" button at the bottom.

Figure 55 Page redirect after Login

## 5.2.2 Visit Car Listing

Register with user data below and check the database,

The screenshot shows the 'Car Listings' section of the AutoXpress website. On the left, there is a 'Filter Cars' sidebar with input fields for Make, Model, Year, Min Price, Max Price, Max Mileage, Fuel Type (with 'Any' selected), and Transmission (with 'Any' selected). Below the sidebar is a 'Clear' button. The main area displays five car listings in a grid:

- Toyota Camry**  
Price: \$25000  
Mileage: 40000 miles  
Fuel Type: HYBRID  
Transmission: AUTOMATIC  
[View Details](#)
- Honda Civic**  
Price: \$100000  
Mileage: 40000 miles  
Fuel Type: DIESEL  
Transmission: MANUAL  
[View Details](#)
- BMW x5**  
Price: \$300000  
Mileage: 35000 miles  
Fuel Type: PETROL  
Transmission: AUTOMATIC  
[View Details](#)
- Toyota Corolla**  
Price: \$50000  
Mileage: 20000 miles  
Fuel Type: HYBRID  
Transmission: MANUAL  
[View Details](#)
- Tesla X**  
Price: \$90000  
Mileage: 200000 miles  
Fuel Type: ELECTRIC  
Transmission: AUTOMATIC  
[View Details](#)

At the bottom right of the grid, there are navigation buttons: 'Previous', '1', '2', and 'Next'.

Figure 56 Car Listing After Cat Car Added

### 5.2.3 Search for a Car by Make, Model, Registration Year & Price Range

Use the Filter Below in Catalog Page,

Make : Toyota

Model : Corolla

Registration Year : 2022

Min Price : 2000

Max Price : 50000

Catalog Page before Filtering are shown in Figure 54.

The screenshot shows the AutoXpress website's catalog page. At the top, there is a navigation bar with links for Home, Car Listing, Car Post, About Us, Contact Us, Admin Dashboard, and Logout. On the left, a sidebar titled "Car Listings" contains a "Filter Cars" section with input fields for Make (Toyota), Model (Corolla), Year (2022), Min Price (1000), Max Price (110000), Max Mileage (empty), Fuel Type (Any), and Transmission (Any). A "Clear" button is also present. To the right, a single car listing is displayed for a "Toyota Corolla". The listing includes a thumbnail image of the car, the model name, and details: Price: \$50000, Mileage: 20000 miles, Fuel Type: HYBRID, and Transmission: MANUAL. A "View Details" button is located below the listing. At the bottom of the page, there is a copyright notice: "© 2025 AutoXpress Pvt Ltd. All rights reserved."

Figure 57 Catalog Page After Filtering

### 5.2.4 View List of Car

Admin Dashboard

Manage Car Posts						
#	Make	Model	Year	Price	Status	Actions
1	Toyota	Camry	2019	\$25000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>
2	Honda	Civic	2018	\$100000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>
3	BMW	x5	2019	\$300000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>
4	Toyota	Corolla	2022	\$50000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>
5	Tesla	X	2024	\$90000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>
6	Cat	Oren	2025	\$25000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>

Figure 58 List of Car in Admin Dashboard

### 5.2.5 View List of Appointments

Register with user data below and check the database,

Admin Dashboard

Manage Test Drive Appointments					
#	User	Car	Date	Status	Actions
1	john_doe	Toyota Camry	2025-06-01 14:33	PENDING	<button>Approve</button> <button>Deny</button>
2	Wen Ping	Cat Oren	2025-06-17 17:00	PENDING	<button>Approve</button> <button>Deny</button>

Figure 59 List of Appointments in Admin Dashboard

### 5.2.6 View List of Report

Register with user data below and check the database,

Admin Dashboard

Manage Reports						
#	User	Car	Reason	Description	Status	Action
1	john_doe	BMW x5	MISLEADING_INFO	Testing	PENDING	<button>Resolve</button>
2	john_doe	Cat Oren	MISLEADING_INFO	WHAT THE HELL IS CAT CAR ????????????	PENDING	<button>Resolve</button>

Figure 60 List of Report in Admin Dashboard

## 5.3 Update functionalites

### 5.3.1 Update User Profile

Update User Information,

Email : bigcat@example.com > bigcat@example.cat

#### Profile Update

Username

Big Cat

Email

bigcat@example.com

Update Profile

Figure 61 User Profile before Update

Profile updated successfully!

#### Profile Update

Username

Big Cat

Email

bigcat@example.cat

Update Profile

Figure 62 User Profile after Update

### 5.3.2 Cancel an Appointment

Cancel Appointment on the car 'Cat Oren'

User Dashboard

Booked Appointments				
#	Car	Date	Status	Actions
1	Cat Oren	2025-06-17 17:00	PENDING	<button>Cancel</button>
Previous 1 Next				

Figure 63 Html Page before cancellation

User Dashboard

Booked Appointments				
#	Car	Date	Status	Actions
1	Cat Oren	2025-06-17 17:00	CANCELED	
Previous 1 Next				

Figure 64 Html Page after cancellation

### 5.3.3 Promote User as Administrator / Demote Administrator as User

Promote Big Cat as Administrator and Demote it back to User,

Admin Dashboard						
Profile		Manage Users				
Manage Users		#	Username	Email	Role	Action
1	admin_user	admin@example.com	ADMIN	<button>Edit</button> <button>Delete</button> <button>Make Admin</button>		
2	Wen Ping	wenziping@gmail.com	USER	<button>Edit</button> <button>Delete</button> <button>Make Admin</button>		
3	john_doe	john.doe@example.com	USER	<button>Edit</button> <button>Delete</button> <button>Make Admin</button>		
4	Big Cat	bigcat@example.cat	USER	<button>Edit</button> <button>Delete</button> <button>Make Admin</button>		

© 2025 AutoXpress Pvt Ltd. All rights reserved.

Figure 65 Html Page before Promoting

The screenshot shows the Admin Dashboard of the AutoXpress application. At the top, there is a navigation bar with links: Home, Car Listing, Car Post, About Us, Contact Us, Admin Dashboard, and Logout. A success message "User marked as admin successfully!" is displayed in a green bar. The main content area is titled "Admin Dashboard" and contains a sidebar with "Profile" and "Manage Users" (which is highlighted in blue). Below the sidebar is a table titled "Manage Users" with the following data:

#	Username	Email	Role	Action
1	admin_user	admin@example.com	ADMIN	
2	Wen Ping	flazeping@gmail.com	USER	Edit Delete Make Admin
3	john_doe	john.doe@example.com	USER	Edit Delete Make Admin
4	Big Cat	bigcat@example.cat	ADMIN	Edit Make User

At the bottom left, a copyright notice reads "© 2025 AutoXpress Pvt Ltd. All rights reserved."

Figure 66 Html Page after Promoting

The screenshot shows the Admin Dashboard of the AutoXpress application. The layout is identical to Figure 66, with the same navigation bar, success message, and sidebar. The "Manage Users" button in the sidebar is highlighted in blue. The "Manage Users" table shows the following data:

#	Username	Email	Role	Action
1	admin_user	admin@example.com	ADMIN	
2	Wen Ping	flazeping@gmail.com	USER	Edit Delete Make Admin
3	john_doe	john.doe@example.com	USER	Edit Delete Make Admin
4	Big Cat	bigcat@example.cat	USER	Edit Delete Make Admin

At the bottom left, a copyright notice reads "© 2025 AutoXpress Pvt Ltd. All rights reserved."

Figure 67 Html Page after Demoting

### 5.3.4 Mark as Sold / Deactivate a Car Post

Mark Tesla X as Sold and Deactivate ‘Cat Oren’

Admin Dashboard

#	Make	Model	Year	Price	Status	Actions
1	Toyota	Camry	2019	\$25000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>
2	Honda	Civic	2018	\$100000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>
3	BMW	x5	2019	\$300000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>
4	Toyota	Corolla	2022	\$50000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>
5	Tesla	X	2024	\$90000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>
6	Cat	Oren	2025	\$25000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>

Figure 68 Html Page before Marked and Deactivate

#	Make	Model	Year	Price	Status	Actions
1	Toyota	Camry	2019	\$25000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>
2	Honda	Civic	2018	\$100000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>
3	BMW	x5	2019	\$300000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>
4	Toyota	Corolla	2022	\$50000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>
5	Tesla	X	2024	\$90000.0	SOLD	<button>Edit</button> <button>Delete</button>
6	Cat	Oren	2025	\$25000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>

Figure 69 Html Page after Marked

#	Make	Model	Year	Price	Status	Actions
1	Toyota	Camry	2019	\$25000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>
2	Honda	Civic	2018	\$100000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>
3	BMW	x5	2019	\$300000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>
4	Toyota	Corolla	2022	\$50000.0	ACTIVE	<button>Edit</button> <button>Delete</button> <button>Deactivate</button> <button>Mark as Sold</button>
5	Tesla	X	2024	\$90000.0	SOLD	<button>Edit</button> <button>Delete</button>
6	Cat	Oren	2025	\$25000.0	INACTIVE	<button>Edit</button> <button>Delete</button>

Figure 70 Html Page after Deactivate

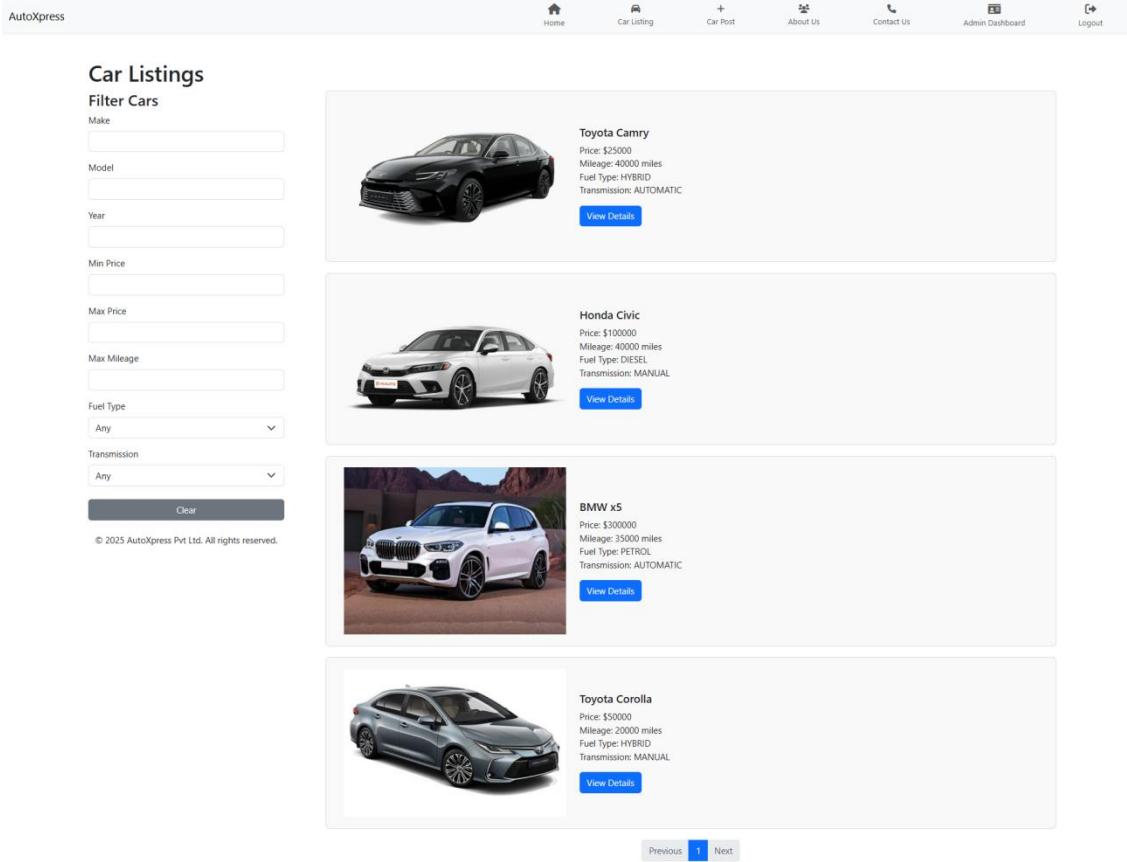


Figure 71 Catalog Page after and Marked Deactivate

### 5.3.5 Approve the Car based on bidding price

Approve the Car created previously (Cat Oren)

Admin Dashboard

Profile

Manage Users

**Manage Cars**

Manage Appointments

Manage Reports

**Manage Car Posts**

#	Make	Model	Year	Price	Status	Actions
1	Toyota	Camry	2019	\$25000.0	ACTIVE	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Deactivate</a> <a href="#">Mark as Sold</a>
2	Honda	Civic	2018	\$100000.0	ACTIVE	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Deactivate</a> <a href="#">Mark as Sold</a>
3	BMW	x5	2019	\$300000.0	ACTIVE	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Deactivate</a> <a href="#">Mark as Sold</a>
4	Toyota	Corolla	2022	\$50000.0	ACTIVE	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Deactivate</a> <a href="#">Mark as Sold</a>
5	Tesla	X	2024	\$90000.0	ACTIVE	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Deactivate</a> <a href="#">Mark as Sold</a>
6	Cat	Oren	2025	\$25000.0	PENDING	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Approve</a>

Figure 72 Admin Dashboard Before Approval

AutoXpress

Home Car Listing + Car Post About Us Contact Us Admin Dashboard Logout

Car approved successfully!

### Admin Dashboard

Profile Manage Users Manage Appointments Manage Reports

**Manage Car Posts**

#	Make	Model	Year	Price	Status	Actions
1	Toyota	Camry	2019	\$25000.0	ACTIVE	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Deactivate</a> <a href="#">Mark as Sold</a>
2	Honda	Civic	2018	\$100000.0	ACTIVE	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Deactivate</a> <a href="#">Mark as Sold</a>
3	BMW	x5	2019	\$300000.0	ACTIVE	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Deactivate</a> <a href="#">Mark as Sold</a>
4	Toyota	Corolla	2022	\$50000.0	ACTIVE	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Deactivate</a> <a href="#">Mark as Sold</a>
5	Tesla	X	2024	\$90000.0	ACTIVE	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Deactivate</a> <a href="#">Mark as Sold</a>
6	Cat	Oren	2025	\$25000.0	ACTIVE	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Deactivate</a> <a href="#">Mark as Sold</a>

© 2025 AutoXpress Pvt Ltd. All rights reserved.

Figure 73 Admin Dashboard After Approval

AutoXpress

Home Car Listing + Car Post About Us Contact Us My Profile Logout

### Car Listings

**Filter Cars**

Make

Model

Year

Min Price

Max Price

Max Mileage

Fuel Type  Any

Transmission  Any

[Clear](#)

© 2025 AutoXpress Pvt Ltd. All rights reserved.



**Cat Oren**  
 Price: \$25000  
 Mileage: 10000 miles  
 Fuel Type: ELECTRIC  
 Transmission: AUTOMATIC

[View Details](#)

Previous 2 Next

Figure 74 Catalog Page after Admin Approval

## 5.4 Delete functionalities

### 5.4.1 Clear Notifications (Extra Feature)

User Dashboard

Notifications			
#	Message	Type	Date
1	Welcome to AutoXpress! Your account has been created.	GENERAL	2025-05-31 12:51:39
2	Car submitted for approval: Cat Oren	GENERAL	2025-05-31 13:26:14
3	Your car listing has been approved: Cat Oren	GENERAL	2025-05-31 13:38:42
4	Your car listing has been updated by an admin: Cat Oren	GENERAL	2025-05-31 13:40:29
5	Your car listing has been approved: Cat Oren	GENERAL	2025-05-31 13:41:08
6	Profile updated successfully	GENERAL	2025-05-31 14:03:25
7	Profile updated successfully	GENERAL	2025-05-31 14:04:50
8	Profile updated successfully	GENERAL	2025-05-31 14:05:37
9	Profile updated successfully	GENERAL	2025-05-31 14:05:40
10	Profile updated successfully	GENERAL	2025-05-31 14:05:44

Figure 75 Html Page before Clearing Notifications

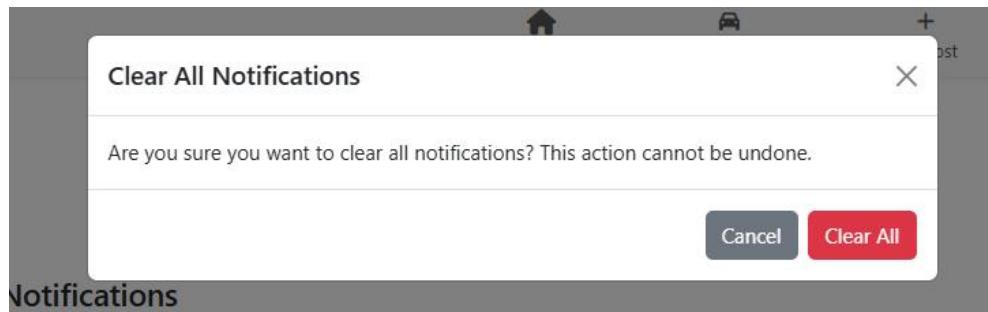


Figure 76 Confirmation Popup

User Dashboard	
Profile Update	
Posted Cars	
Booked Appointments	
Notifications	
Notifications	
No notifications.	

Figure 77 Html Page after Clearing Notifications

### 5.4.2 Admin Remove a Car Post

Remove the Car 'Cat Oren' shown in Figure 68 with descriptions of:

Reason of Deletion : Misleading Info: Invalid Car Make and Model

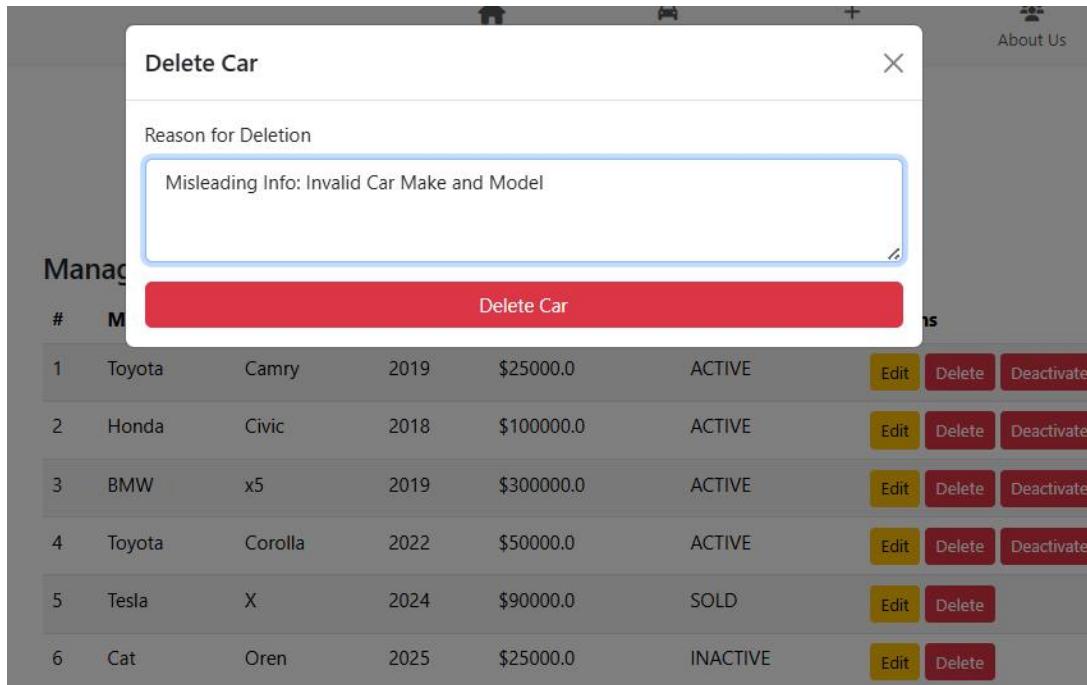


Figure 78 Reason for Deletion Popup

### Admin Dashboard

The screenshot shows the Admin Dashboard with a sidebar on the left containing 'Profile', 'Manage Users', 'Manage Cars' (which is highlighted in blue), 'Manage Appointments', and 'Manage Reports'. The main area is titled 'Manage Car Posts' and displays a table with columns: #, Make, Model, Year, Price, Status, and Actions. The table now only contains five rows of data, indicating that the car 'Cat Oren' has been successfully deleted.

#	Make	Model	Year	Price	Status	Actions
1	Toyota	Camry	2019	\$25000.0	ACTIVE	Edit Delete Deactivate Mark as Sold
2	Honda	Civic	2018	\$100000.0	ACTIVE	Edit Delete Deactivate Mark as Sold
3	BMW	x5	2019	\$300000.0	ACTIVE	Edit Delete Deactivate Mark as Sold
4	Toyota	Corolla	2022	\$50000.0	ACTIVE	Edit Delete Deactivate Mark as Sold
5	Tesla	X	2024	\$90000.0	SOLD	Edit Delete

Figure 79 Html Page after Car Removed

### 5.4.3 Admin Removing the User

Remove the User 'Big Cat'

#### Admin Dashboard

The screenshot shows the Admin Dashboard with a sidebar on the left containing 'Profile', 'Manage Users' (which is highlighted in blue), 'Manage Cars', 'Manage Appointments', and 'Manage Reports'. The main area is titled 'Manage Users' and displays a table with the following data:

#	Username	Email	Role	Action
1	admin_user	admin@example.com	ADMIN	<button>Edit</button> <button>Delete</button> <button>Make Admin</button>
2	Wen Ping	flazeping@gmail.com	USER	<button>Edit</button> <button>Delete</button> <button>Make Admin</button>
3	john_doe	john.doe@example.com	USER	<button>Edit</button> <button>Delete</button> <button>Make Admin</button>
4	Big Cat	bigcat@example.cat	USER	<button>Edit</button> <button>Delete</button> <button>Make Admin</button>

Figure 80 Html Page before User Deletion

A modal dialog box titled 'Delete User' is displayed in the center. It contains the question 'Are you sure you want to delete this user?' and a large red button labeled 'Delete User' at the bottom.

Below the dialog, the 'Manage Users' table is shown again, but the row for 'Big Cat' is missing, indicating it has been deleted.

#	Username	Email	Role	Action
1	admin_user	admin@example.com	ADMIN	<button>Edit</button> <button>Delete</button> <button>Make Admin</button>
2	Wen Ping	flazeping@gmail.com	USER	<button>Edit</button> <button>Delete</button> <button>Make Admin</button>
3	john_doe	john.doe@example.com	USER	<button>Edit</button> <button>Delete</button> <button>Make Admin</button>
4	Big Cat	bigcat@example.cat	USER	<button>Edit</button> <button>Delete</button> <button>Make Admin</button>

Figure 81 User Deletion Confirmation Popup

#### Admin Dashboard

The screenshot shows the Admin Dashboard with the same sidebar and 'Manage Users' table. The row for 'Big Cat' is now missing from the table, confirming its deletion.

#	Username	Email	Role	Action
1	admin_user	admin@example.com	ADMIN	<button>Edit</button> <button>Delete</button> <button>Make Admin</button>
2	Wen Ping	flazeping@gmail.com	USER	<button>Edit</button> <button>Delete</button> <button>Make Admin</button>
3	john_doe	john.doe@example.com	USER	<button>Edit</button> <button>Delete</button> <button>Make Admin</button>

Figure 82 Html Page after User Deletion

## Task 6 Role-based Access

The SecurityConfig class is a Spring Security configuration that enforces role-based access control and **secures web endpoints**. It defines a **SecurityFilterChain** to **permit public access** and **restrict access** based on role “USER” or “ADMIN”. The configuration includes a custom UserDetailsService that loads users by username from UserRepository, mapping their roles for authentication. It also implements form-based login with a custom success handler redirecting **users to role-specific dashboards, logout functionality, session management** with a maximum of one session, and **exception handling** for access denial and **authentication failures**. Additionally, it uses **BCryptPasswordEncoder** for **password encryption**, ensuring secure user credential management.

```
@Bean
public AccessDeniedHandler accessDeniedHandler() {
    return (request, response, accessDeniedException) -> {
        // Get the current authentication object
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

        // Check if the user is authenticated and has the USER role
        if (authentication != null && authentication.getAuthorities().stream()
            .anyMatch(grantedAuthority -> grantedAuthority.getAuthority().equals("ROLE_USER"))) {
            // Redirect USER role users to /user/dashboard with an error message
            response.sendRedirect("/user/dashboard?error=Access%20denied.%20You%20do%20not%20have%20permission%20to%20access%20this%20page.");
        } else {
            // For unauthenticated users or other roles, redirect to login page
            response.sendRedirect("/login?error=Please%20log%20in%20to%20access%20this%20page.");
        }
    };
}

@Bean
public AuthenticationSuccessHandler authenticationSuccessHandler() {
    return (HttpServletRequest request, HttpServletResponse response, Authentication authentication) -> {
        String userId = authentication.getName(); // Now userId instead of username
        UserModel user = userRepository.findById(Long.parseLong(userId))
            .orElseThrow(() -> new UsernameNotFoundException("User not found with ID: " + userId));
        String redirectUrl = user.getUserRole() == UserModel.Role.USER ? "/user/dashboard" : "/admin/dashboard";
        response.sendRedirect(redirectUrl);
    };
}

@Bean
public UserDetailsService userDetailsService() {
    return username -> {
        // Load user by username (as entered in login form)
        UserModel user = userRepository.findByUsername(username)
            .orElseThrow(() -> new UsernameNotFoundException("User not found: " + username));
        // Use userId as the principal in UserDetails
        return User.withUsername(String.valueOf(user.getUserId()))
            .password(user.getUserPassword())
            .roles(user.getUserRole().name())
            .build();
    };
}

@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

Figure 83 SecurityConfig Part 1

```

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Autowired
    private UserRepository userRepository;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        // Define public endpoints
        RequestMatcher publicEndpoints = new OrRequestMatcher(
            new AntPathRequestMatcher("/"),
            new AntPathRequestMatcher("/cars"),
            new AntPathRequestMatcher("/cars/**"),
            new AntPathRequestMatcher("/cars/search"),
            new AntPathRequestMatcher("/about"),
            new AntPathRequestMatcher("/contact"),
            new AntPathRequestMatcher("/register"),
            new AntPathRequestMatcher("/login"),
            new AntPathRequestMatcher("/password-reset-request"),
            new AntPathRequestMatcher("/password-reset"),
            new AntPathRequestMatcher("/css/**"),
            new AntPathRequestMatcher("/js/**"),
            new AntPathRequestMatcher("/images/**")
        );
        // Define matcher for non-public endpoints
        RequestMatcher nonPublicEndpoints = new NegatedRequestMatcher(publicEndpoints);

        http
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/", "/cars", "/cars/**", "/cars/search", "/about", "/contact", "/register", "/login",
                    "/password-reset-request", "/password-reset", "/css/**", "/js/**", "/images/**", "/error").permitAll()
                .requestMatchers("/cars/**appointment", "/cars/**report", "/user/**", "/cars/post").hasRole("USER")
                .requestMatchers("/admin/**").hasRole("ADMIN")
            )
            .formLogin(form -> form
                .LoginPage("/login")
                .successHandler(authenticationSuccessHandler())
                .permitAll()
            )
            .logout(logout -> logout
                .logoutRequestMatcher(new AntPathRequestMatcher("/logout", "GET"))
                .logoutSuccessUrl("/login?logout")
                .clearAuthentication(true)
                .permitAll()
            )
            .sessionManagement(session -> session
                .sessionFixation().migrateSession()
                .maximumSessions(1)
                .expiredUrl("/login?expired")
            )
            .exceptionHandling(exception -> exception
                .defaultAuthenticationEntryPointFor(
                    (request, response, authException) -> {
                        if ("XMLHttpRequest".equals(request.getHeader("X-Requested-With"))) {
                            response.setStatus(HttpStatus.SC_UNAUTHORIZED);
                            response.getWriter().write("{\"error\": \"Unauthorized\"}");
                        } else {
                            response.sendRedirect("/login");
                        }
                    },
                    nonPublicEndpoints
                )
                .accessDeniedHandler(accessDeniedHandler())
            );
        return http.build();
    }
}

```

Figure 84 SecurityConfig Part 2

## 6.1 Public Role

The Public Role applies to all unauthenticated users and defines access to endpoints that do not require login credentials. In the application, public endpoints are explicitly permitted for all users, ensuring that essential features like browsing cars and accessing informational pages are available without authentication. The following endpoints are accessible under the Public Role are:

- **/** : The **homepage** of the application.
- **/cars** : The main **car listing page** with pagination and filtering capabilities.
- **/cars/\*\*** : Includes specific **car detail pages** (e.g., **/cars/{id}**) and search functionality.
- **/cars/search** : An **AJAX endpoint for searching cars** dynamically.
- **/about** : The **About Us page** providing information about the platform.
- **/contact** : The **Contact Us page** for submitting inquiries.
- **/register** : The user registration page.
- **/login** : The login page for user authentication.
- **/password-reset-request** : The page to request a password reset.
- **/password-reset** : The page to reset a password using a token.
- **/css/\*\*, /js/\*\*, /images/\*\*** : Static resources (CSS, JavaScript, and images) necessary for rendering pages.

These endpoints are configured to be accessible to all users via the `permitAll()` method in the `SecurityFilterChain` bean, ensuring that unauthenticated users can interact with core features of the application without restrictions.

### 6.1.1 Public Role Header

Public Role Header has Home, Car Listing, About Us, Contact Us, Login & Register Navigator Bar.

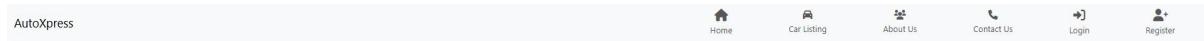


Figure 85 Public Role Header

### 6.1.2 Public Role Car Detail Page

Public Role Car Details only display car details where user cannot make an appointment or report the car post with a hint asking user to log in to use the feature.

A screenshot of the AutoXpress website showing a car detail page. The top navigation bar is identical to Figure 85. Below it, the main content area shows a black Toyota Camry. The title 'Car Details' is above the car image. To the right of the car, the model 'Toyota Camry' is listed along with its year (2019), price (\$25000.00), mileage (40000 miles), fuel type (HYBRID), transmission (AUTOMATIC), status (ACTIVE), and the poster (Wen Ping). At the bottom of the page, there is a note: 'Please [log in](#) to book a test drive or report this car.' and a copyright notice: '© 2025 AutoXpress Pvt Ltd. All rights reserved.'

Figure 86 Public Role Car Detail Page

## 6.2 User Role

The User Role, represented as ROLE\_USER in UserModel.Role, is assigned to **authenticated users** who have **registered and logged** in as regular users. This role grants access to features that require **user authentication**, such as **managing personal car listings, booking appointments, and submitting reports**. The SecurityConfig.java defines the following access rules for the User Role:

- **/cars/\*/appointment** : Allows users to **book test drive appointments** for specific cars. This endpoint ensures that **only authenticated users can schedule appointments**, protecting the system from unauthorized access.
- **/cars/\*/report** : Permits users to **submit reports** on car listings, such as flagging misleading information or irrelevant pictures.
- **/user/\*\*** : Grants access to all user-related endpoints, including the **user dashboard** (/user/dashboard), **profile management** (/user/profile), **appointment cancellation** (/user/appointments/cancel), and **notification clearing** (/user/notifications/clear).
- **/cars/post** : Enables users to post new car listings, including uploading images and specifying car details. This feature is restricted to authenticated users to prevent anonymous submissions.

The User Role is enforced using the hasRole("USER") method in the SecurityFilterChain bean, ensuring that only users with the ROLE\_USER authority can access these endpoints. Upon successful login, the **authenticationSuccessHandler** **redirects users** with the User Role to **/user/dashboard**, providing a personalized experience.

### 6.2.1 User Role Header

User Role Header has Home, Car Listing, About Us, Contact Us, My Profile & Log Out Navigator Bar.



Figure 87 User Role Header

### 6.2.2 User Role Car Detail Page

User Role Car Details display all the information of the car and let user to book appointment or make a report on the car.

A screenshot of the AutoXpress website's car detail page for a Toyota Camry. At the top, there is a navigation bar identical to Figure 87. Below it, the main content area starts with a heading "Car Details". A large image of a black Toyota Camry is centered. To the right of the car, the model name "Toyota Camry" is displayed, along with its year "Year: 2019", price "\$25000.00", mileage "40000 miles", fuel type "HYBRID", transmission "AUTOMATIC", status "ACTIVE", and the poster "Posted By: Wen Ping". On the left side, there is a section titled "Actions" with a "Book a Test Drive" button and a date input field. On the right side, there is a "Report This Car" section with a "Reason" dropdown menu, a "Description" text area, and a "Submit Report" button. At the bottom left, there is a copyright notice: "© 2025 AutoXpress Pvt Ltd. All rights reserved.".

Figure 88 User Role Car Detail Page

## 6.2 Admin Role

The Admin Role, represented as `ROLE_ADMIN` in `UserModel.Role`, is assigned to **authenticated users** with elevated **privileges**, typically administrators responsible for managing the platform. This role **grants access to administrative functions** such as user management, car moderation, appointment approvals, and report resolutions. The `SecurityConfig.java` defines the following access rules for the Admin Role:

- **/admin/\*\***: Grants access to all admin-related endpoints, including the :
  1. **Admin dashboard** (`/admin/dashboard`)
  2. **User management** (`/admin/users/mark-admin`, `/admin/users/make-user`,  
`/admin/users/delete`, `/admin/users/edit`)
  3. **Car management** (`/admin/cars/edit`, `/admin/cars/delete`, `/admin/cars/activate`,  
`/admin/cars/deactivate`, `/admin/cars/sell`)
  4. **Appointment management** (`/admin/appointments/approve`,  
`/admin/appointments/deny`)
  5. **Report resolution** (`/admin/reports/resolve`)

The Admin Role is enforced using the `hasRole("ADMIN")` method in the `SecurityFilterChain` bean, ensuring that only users with the `ROLE_ADMIN` authority can access these endpoints. The **authenticationSuccessHandler** redirects users with the **Admin Role** to **/admin/dashboard** upon login, providing a centralized interface for administrative tasks.

Additionally, the **accessDeniedHandler** ensures that if a user with `ROLE_USER` attempts to access an admin endpoint, they are **redirected to /user/dashboard with an error message**, while unauthenticated users are redirected to `/login`.

### 6.2.1 Admin Role Header

Admin Role Header has Home, Car Listing, About Us, Contact Us, Admin Dashboard & Log Out Navigator Bar.



Figure 89 Admin Role Header

### 6.1.2 Admin Role Car Detail Page

Admin Role Car Details only display car details where its purpose to let admin view the car images to prevent overloading the list.

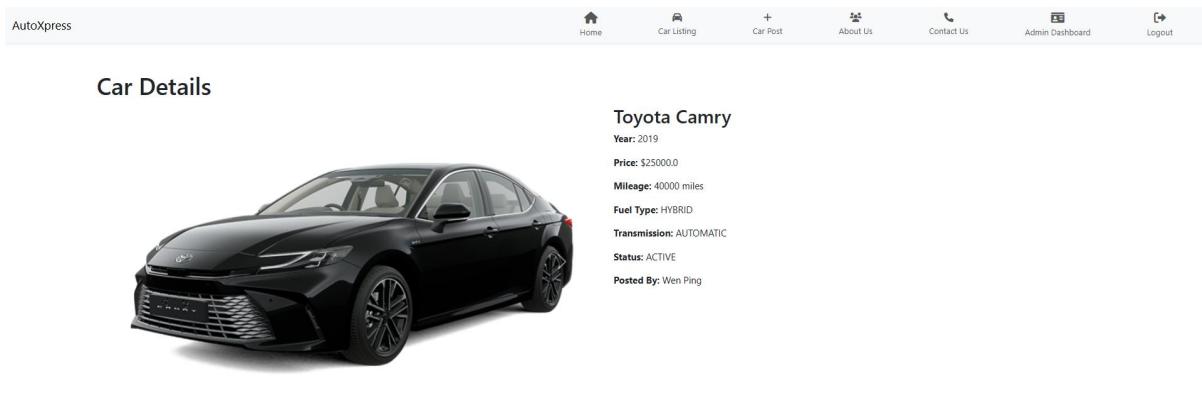


Figure 90 Admin Role Car Detail Page

## Task 7 JUnit Test Plan

### 7.1 Functional Testing

The functional tests focus on the UserController and related services (CarService, AppointmentService, NotificationService). The tests simulate user interactions by invoking service methods directly, as some endpoints (e.g., /cars/post) are handled by other controllers (e.g., CarController). Below are the console output for UserController Functional Test.

The screenshot shows the Eclipse IDE interface with the JUnit View and Console View open. The JUnit View displays a successful run with 5/5 tests passed, 0 errors, and 0 failures. The Console View shows the log output for the UserControllerFunctionalTest, which includes setting up the test environment, starting various test cases, and confirming their successful completion.

```
<terminated> UserControllerFunctionalTest [JUnit] C:\spring tool\sts-4.30.0.RELEASE\plugins\org.eclipse.jdt.openjdk.hotspot.jre
Setting up test environment for UserControllerFunctionalTest at 2025-05-30T10:20:23.386625
Starting test: testCancelAppointment_CannotCancel
Test testCancelAppointment_CannotCancel completed successfully
Setting up test environment for UserControllerFunctionalTest at 2025-05-30T10:20:23.585453200
Starting test: testActivateCar_Success
Test testActivateCar_Success completed successfully
Setting up test environment for UserControllerFunctionalTest at 2025-05-30T10:20:23.630237300
Starting test: testPostCar_Success
Test testPostCar_Success completed successfully
Setting up test environment for UserControllerFunctionalTest at 2025-05-30T10:20:23.680028200
Starting test: testCancelAppointment_Success
Test testCancelAppointment_Success completed successfully
Setting up test environment for UserControllerFunctionalTest at 2025-05-30T10:20:23.725857100
Starting test: testClearNotifications_Success
Test testClearNotifications_Success completed successfully
```

Figure 91 UserControllerFunctionalTest.java Output

#### 7.1.1 UserController Functional Test Table

Test Case	Description	Expected Output
TC 7.1.2	Simulates posting a car by directly invoking <code>carService.saveCar</code>	<code>carService.saveCar</code> and <code>notificationService.createNotification</code> are called once.
TC 7.1.3	Simulates car activation by invoking <code>carService.activateCar</code>	<code>carService.activateCar</code> and <code>notificationService.createNotification</code> are called once.
TC 7.1.4	cancelling a pending appointment via <code>userController.cancelAppointment</code>	Redirects to <code>/user/dashboard?appointmentCancelled</code> ; services are called as expected.
TC 7.1.5	Tests clearing notifications via <code>userController.clearNotifications</code>	Redirects to <code>/user/dashboard?NotificationsCleared</code> ; <code>notificationService.ClearNotifications</code> is called once.
TC 7.1.6	Tests cancelling a non-pending/approved appointment (status: DENIED).	Returns <code>user-dashboard</code> with an error message; <code>appointmentService.cancelAppointment</code> is not called.

### 7.1.2 Car Posting Testing

```
// Test 1: Functional test for posting a car
@Test
void testPostCar_Success() throws Exception {
    System.out.println("Starting test: testPostCar_Success");

    // Arrange: Setting up test data and necessary stubbings
    List<MultipartFile> images = new ArrayList<>();
    images.add(multipartFile);
    // Removed unused stubbings to avoid UnnecessaryStubbingException
    // multipartFile.isEmpty(), multipartFile.getSize(), userService.getUserId(), and bindingResult.hasErrors()
    // are not used since we're directly calling the service method
    doNothing().when(carService).saveCar(any(CarModel.class), anyList(), anyString());
    doNothing().when(notificationService).createNotification(anyString(), anyString(), any(NotificationModel.NotificationType.class));

    // Act: Simulate the controller's behavior by directly invoking the service method
    // Since /cars/post is not in UserController, we directly call the service method to mimic the controller's action
    carService.saveCar(car, images, "john_doe");
    notificationService.createNotification(
        "john_doe",
        "Car submitted for approval: " + car.getCarMake() + " " + car.getCarModel(),
        NotificationModel.NotificationType.GENERAL
    );

    // Assert: Verify the service calls
    verify(carService, times(1)).saveCar(car, images, "john_doe");
    verify(notificationService, times(1)).createNotification(
        eq("john_doe"),
        eq("Car submitted for approval: Toyota Camry"),
        eq(NotificationModel.NotificationType.GENERAL)
    );
}

System.out.println("Test testPostCar_Success completed successfully");
}
```

Figure 92 testPostCar\_Success()

### 7.1.3 Change Car Status into Active Testing

```
// Test 2: Functional test for activating a car (admin flow, but we'll test the service layer called by UserController)
@Test
void testActivateCar_Success() throws Exception {
    System.out.println("Starting test: testActivateCar_Success");

    // Arrange: Mocking dependencies
    when(carService.activateCar(1L)).thenReturn(car);

    // Act: Simulate the service call (this would typically be in AdminController, but we're testing the flow)
    carService.activateCar(1L);
    notificationService.createNotification(
        user.getUsername(),
        "Your car listing has been approved: " + car.getCarMake() + " " + car.getCarModel(),
        NotificationModel.NotificationType.GENERAL
    );

    // Assert: Verify the interactions
    verify(carService, times(1)).activateCar(1L);
    verify(notificationService, times(1)).createNotification(
        eq("john_doe"),
        eq("Your car listing has been approved: Toyota Camry"),
        eq(NotificationModel.NotificationType.GENERAL)
    );
}

System.out.println("Test testActivateCar_Success completed successfully");
}
```

Figure 93 testActivateCar\_Success()

#### 7.1.4 Cancel Appointment Testing

```
// Test 3: Functional test for cancelling an appointment
@Test
void testCancelAppointment_Success() throws Exception {
    System.out.println("Starting test: testCancelAppointment_Success");

    // Arrange: Mocking dependencies
    when(authentication.getName()).thenReturn("1");
    when(userService.getUserId(1L)).thenReturn(user);
    when(appointmentService.getAppointmentById(1L)).thenReturn(appointment);
    doNothing().when(appointmentService).cancelAppointment(1L);
    doNothing().when(notificationService).createNotification(anyString(), anyString(), any(NotificationModel.NotificationType.class));

    // Act: Call the cancelAppointment method
    String result = userController.cancelAppointment(1L, model, authentication);

    // Assert: Verify the redirect and interactions
    assertEquals("redirect:/user/dashboard?appointmentCancelled", result);
    verify(appointmentService, times(1)).cancelAppointment(1L);
    verify(notificationService, times(1)).createNotification(
        eq("john_doe"),
        eq("Test drive appointment cancelled for Toyota Camry"),
        eq(NotificationModel.NotificationType.APPOINTMENT_STATUS)
    );
    System.out.println("Test testCancelAppointment_Success completed successfully");
}
```

Figure 94 testCancelAppointment\_Success()

#### 7.1.5 Clear Notificaiton Testing

```
// Test 4: Functional test for clearing notifications
@Test
void testClearNotifications_Success() {
    System.out.println("Starting test: testClearNotifications_Success");

    // Arrange: Mocking dependencies
    when(authentication.getName()).thenReturn("1");
    when(userService.getUserId(1L)).thenReturn(user);
    doNothing().when(notificationService).clearNotifications(1L);

    // Act: Call the clearNotifications method
    String result = userController.clearNotifications(authentication);

    // Assert: Verify the redirect and interactions
    assertEquals("redirect:/user/dashboard?notificationsCleared", result);
    verify(notificationService, times(1)).clearNotifications(1L);

    System.out.println("Test testClearNotifications_Success completed successfully");
}
```

Figure 95 testClearNotifications\_Success()

### 7.1.6 Cancel Appointment that cannot be canceled Testing

```
// Test 5: Functional test for cancelling an appointment that cannot be cancelled (not PENDING or APPROVED)
@Test
void testCancelAppointment_CannotCancel() throws Exception {
    System.out.println("Starting test: testCancelAppointment_CannotCancel");

    // Arrange: Mocking dependencies
    when(authentication.getName()).thenReturn("1");
    appointment.setAppointmentStatus(AppointmentModel.AppointmentStatus.DENIED);
    when(userService.getUserById(1L)).thenReturn(user);
    when(appointmentService.getAppointmentById(1L)).thenReturn(appointment);

    // Act: Call the cancelAppointment method
    String result = userController.cancelAppointment(1L, model, authentication);

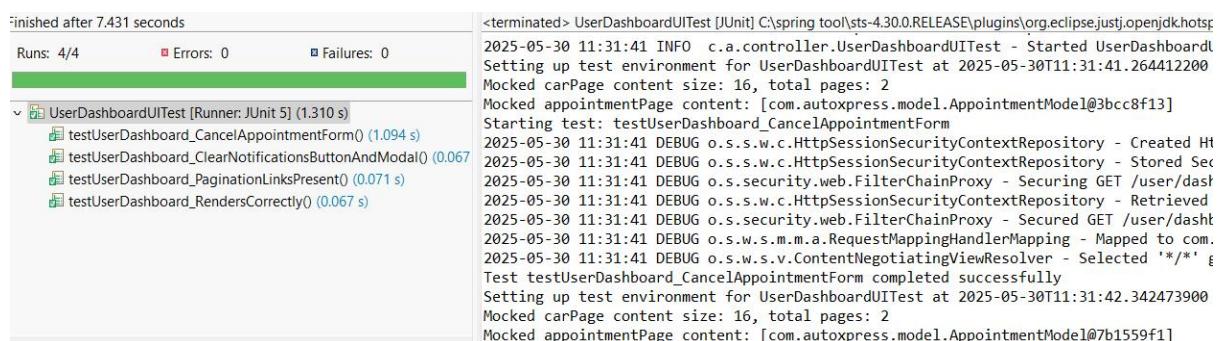
    // Assert: Verify the error handling
    assertEquals("user-dashboard", result);
    verify(model, times(1)).addAttribute(eq("error"), eq("Cannot cancel an appointment that is not pending or approved."));
    verify(appointmentService, never()).cancelAppointment(anyLong());

    System.out.println("Test testCancelAppointment_CannotCancel completed successfully");
}
```

Figure 96 testCancelAppointment\_CannotCancel()

## 7.2 UI Testing

UI testing for the AutoXpress application verifies that the web interface renders correctly and contains the expected elements as seen by the user. It ensures that key pages, such as the user dashboard, display the correct content, forms, and navigation elements, providing a seamless user experience. Below Figure are the console output for UserDashboard UI Test.



```
<terminated> UserDashboardUITest [JUnit] C:\spring tool\sts-4.30.0.RELEASE\plugins\org.eclipse.jst.j.openjdk.hotsp
2025-05-30 11:31:41 INFO c.a.controller.UserDashboardUITest - Started UserDashboardl
Setting up test environment for UserDashboardUITest at 2025-05-30T11:31:41.264412200
Mocked carPage content size: 16, total pages: 2
Mocked appointmentPage content: [com.autoxpress.model.AppointmentModel@3bcc8f1]
Starting test: testUserDashboard_CancelAppointmentForm
2025-05-30 11:31:41 DEBUG o.s.s.w.c.HttpSessionSecurityContextRepository - Created Ht
2025-05-30 11:31:41 DEBUG o.s.s.w.c.HttpSessionSecurityContextRepository - Stored Sec
2025-05-30 11:31:41 DEBUG o.s.security.web.FilterChainProxy - Securing GET /user/dash
2025-05-30 11:31:41 DEBUG o.s.s.w.c.HttpSessionSecurityContextRepository - Retrieved
2025-05-30 11:31:41 DEBUG o.s.s.w.c.HttpSessionSecurityContextRepository - Secured GET /user/dash
2025-05-30 11:31:41 DEBUG o.s.w.s.m.m.a.RequestMappingHandlerMapping - Mapped to com.
2025-05-30 11:31:41 DEBUG o.s.w.s.v.ContentNegotiatingViewResolver - Selected '*/*' &
Test testUserDashboard_CancelAppointmentForm completed successfully
Setting up test environment for UserDashboardUITest at 2025-05-30T11:31:42.342473900
Mocked carPage content size: 16, total pages: 2
Mocked appointmentPage content: [com.autoxpress.model.AppointmentModel@7b1559f1]
```

Figure 97 UserDashboardUITest.java Result

### 7.2.1 UserDashboard UI Test Table

Test Case	Description	Expected Output
TC 7.2.2	Verifies that the user dashboard renders with all panels (Profile, Posted Cars, Appointments, Notifications).	Dashboard renders with all expected panels (confirmed by HTML element IDs).
TC 7.2.3	Ensures pagination links are present in the Posted Cars panel when there are multiple pages of cars.	Pagination section (Posted Cars Pagination) and pagePostedCars parameter are present.
TC 7.2.4	Confirms the Clear Notifications button and modal are rendered, including the form to clear notifications.	Button, modal, and form (action="/user/notifications/clear") are present.
TC 7.2.5	Validates that the Cancel Appointment form is rendered for a pending appointment.	Form (action="/user/appointments/cancel") and appointmentId field are present.

### 7.2.2 UserDashboard Render Testing

```
// Test 1: Verify that the user dashboard renders correctly
@Test
void testUserDashboard_RendersCorrectly() throws Exception {
    System.out.println("Starting test: testUserDashboard_RendersCorrectly");

    // Act: Make a GET request to /user/dashboard
    mockMvc.perform(MockMvcRequestBuilders.get("/user/dashboard"))
        // Assert: Verify the response and rendered content
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.content()
            .string(org.hamcrest.Matchers.containsString("<h2>User Dashboard</h2>")))
        .andExpect(MockMvcResultMatchers.content()
            .string(org.hamcrest.Matchers.containsString("id=\"profile-panel\"")))
        .andExpect(MockMvcResultMatchers.content()
            .string(org.hamcrest.Matchers.containsString("id=\"posted-cars-panel\"")))
        .andExpect(MockMvcResultMatchers.content()
            .string(org.hamcrest.Matchers.containsString("id=\"appointments-panel\"")))
        .andExpect(MockMvcResultMatchers.content()
            .string(org.hamcrest.Matchers.containsString("id=\"notifications-panel\"")));

    System.out.println("Test testUserDashboard_RendersCorrectly completed successfully");
}
```

Figure 98 testUserDashboard\_RendersCorrectly()

### 7.2.3 UserDashboard Pagination Link Testing

```
// Test 2: Verify pagination links are present in the Posted Cars panel
@Test
void testUserDashboard_PaginationLinksPresent() throws Exception {
    System.out.println("Starting test: testUserDashboard_PaginationLinksPresent");

    // Act: Make a GET request to /user/dashboard
    mockMvc.perform(MockMvcRequestBuilders.get("/user/dashboard"))
        // Assert: Verify pagination links in the Posted Cars panel
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.content()
            .string(org.hamcrest.Matchers.containsString("Posted Cars Pagination")))
        .andExpect(
            MockMvcResultMatchers.content().string(org.hamcrest.Matchers.containsString("pagePostedCars")));

    System.out.println("Test testUserDashboard_PaginationLinksPresent completed successfully");
}
```

Figure 99 testUserDashboard\_PaginationLinksPresent()

### 7.2.4 UserDashboard Clear Notification Testing

```
// Test 3: Verify the Clear Notifications button and modal are present
@Test
void testUserDashboard_ClearNotificationsButtonAndModal() throws Exception {
    System.out.println("Starting test: testUserDashboard_ClearNotificationsButtonAndModal");

    // Act: Make a GET request to /user/dashboard
    mockMvc.perform(MockMvcRequestBuilders.get("/user/dashboard"))
        // Assert: Verify the Clear Notifications button and modal
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.content()
            .string(org.hamcrest.Matchers.containsString("data-bs-target=\">#clearNotificationsModal\"")))
        .andExpect(MockMvcResultMatchers.content()
            .string(org.hamcrest.Matchers.containsString("id=\"clearNotificationsModal\"")));
        // Check for the rendered action attribute instead of Thymeleaf syntax
        .andExpect(MockMvcResultMatchers.content()
            .string(org.hamcrest.Matchers.containsString("action=\"/user/notifications/clear\"")));

    System.out.println("Test testUserDashboard_ClearNotificationsButtonAndModal completed successfully");
}
```

Figure 100 testUserDashboard\_ClearNotificationsButtonAndModal()

### 7.2.5 UserDashboard Cancel Appointment Testing

```
// Test 4: Verify the form for cancelling an appointment is present
@Test
void testUserDashboard_CancelAppointmentForm() throws Exception {
    System.out.println("Starting test: testUserDashboard_CancelAppointmentForm");

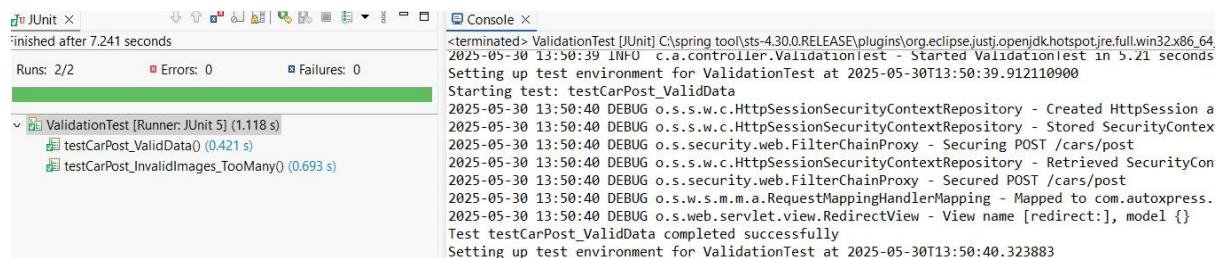
    // Act: Make a GET request to /user/dashboard
    mockMvc.perform(MockMvcRequestBuilders.get("/user/dashboard"))
        // Assert: Verify the Cancel Appointment form
        .andExpect(MockMvcResultMatchers.status().isOk())
        // Check for the rendered action attribute instead of Thymeleaf syntax
        .andExpect(MockMvcResultMatchers.content().string(org.hamcrest.Matchers.containsString("action=\"/user/appointments/cancel\"")))
        .andExpect(MockMvcResultMatchers.content().string(org.hamcrest.Matchers.containsString("name=\"appointmentId\"")));

    System.out.println("Test testUserDashboard_CancelAppointmentForm completed successfully");
}
```

Figure 101 testUserDashboardCancelAppointmentForm()

## 7.3 Validation Testing

Validation testing for the AutoXpress application ensures that the application enforces proper input validation rules, rejecting invalid data and accepting valid data according to defined constraints. This phase focuses on validating form submissions and endpoint inputs to maintain data integrity and provide appropriate user feedback.



The screenshot shows the Eclipse IDE interface with the JUnit view open. The status bar indicates "finished after 7.241 seconds" with 2/2 runs, 0 errors, and 0 failures. The test tree shows two tests under "ValidationTest [Runner: JUnit 5]": "testCarPost\_ValidData() (0.421 s)" and "testCarPost\_InvalidImages\_TooMany() (0.693 s)". The Java console tab displays the execution log:

```
<terminated> ValidationTest [JUnit] C:\spring tool\sts-4.30.0.RELEASE\plugins\org.eclipse.jdt.core\openjdk.hotspot.jre.full.win32.x86_64
2025-05-30 13:50:39 INFO c.a.controller.ValidationTest - Started ValidationTest in 5.21 seconds
Setting up test environment for ValidationTest at 2025-05-30T13:50:39.912110900
Starting test: testCarPost_ValidData
2025-05-30 13:50:40 DEBUG o.s.s.w.c.HttpSessionSecurityContextRepository - Created HttpSession a
2025-05-30 13:50:40 DEBUG o.s.s.w.c.HttpSessionSecurityContextRepository - Stored SecurityContext
2025-05-30 13:50:40 DEBUG o.s.security.web.FilterChainProxy - Securing POST /cars/post
2025-05-30 13:50:40 DEBUG o.s.s.w.c.HttpSessionSecurityContextRepository - Retrieved SecurityCon
2025-05-30 13:50:40 DEBUG o.s.security.web.FilterChainProxy - Secured POST /cars/post
2025-05-30 13:50:40 DEBUG o.s.w.s.m.m.a.RequestMappingHandlerMapping - Mapped to com.autoxpress.
2025-05-30 13:50:40 DEBUG o.s.web.servlet.view.RedirectView - View name [redirect:], model {}
Test testCarPost_ValidData completed successfully
Setting up test environment for ValidationTest at 2025-05-30T13:50:40.323883
```

Figure 102 ValidationTest.java Output

### 7.3.1 Validation Test Table

Test Case	Description	Expected Output
<b>TC 7.3.2</b>	Validates that submitting a car posting with more than 5 images is rejected.	Returns the post-car view (200 OK) with an error message "Maximum 5 images allowed."
<b>TC 7.3.3</b>	Confirms that a valid car posting with one image is accepted.	Redirects to /user/dashboard?submitted (302 Found).

### 7.3.2 >5 Images Uploaded in CarPost Testing

```
// Test 1: Validate car posting with invalid images (too many images)
@Test
void testCarPost_InvalidImages_TooMany() throws Exception {
    System.out.println("Starting test: testCarPost_InvalidImages_TooMany");

    // Arrange: Create a valid car with too many images (more than 5)
    MockMultipartFile[] images = new MockMultipartFile[6];
    for (int i = 0; i < 6; i++) {
        images[i] = new MockMultipartFile(
            "images",
            "test-image-" + i + ".jpg",
            "image/jpeg",
            ("test image content " + i).getBytes()
        );
    }

    // Act: Submit the car posting form with too many images, including CSRF token
    mockMvc.perform(MockMvcRequestBuilders.multipart("/cars/post")
        .file(images[0])
        .file(images[1])
        .file(images[2])
        .file(images[3])
        .file(images[4])
        .file(images[5])
        .param("car.carMake", "Toyota")
        .param("car.carModel", "Camry")
        .param("car.carPrice", "15000.0")
        .with(SecurityMockMvcRequestPostProcessors.user("1").authorities(new SimpleGrantedAuthority("ROLE_USER")))
        .with(SecurityMockMvcRequestPostProcessors.csrf()) // Add CSRF token
        .with(request -> {
            request.setMethod("POST");
            return request;
        }))
        // Assert: Verify the response (should return to the form with errors)
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("post-car"))
        .andExpect(MockMvcResultMatchers.model().attributeExists("error"))
        .andExpect(MockMvcResultMatchers.model().attribute("error", "Maximum 5 images allowed."));

    System.out.println("Test testCarPost_InvalidImages_TooMany completed successfully");
}
```

Figure 103 testCarPost\_InvalidImages\_TooMany()

### 7.3.3 CarPost with Valid Data Testing

```

// Test 2: Validate car posting with valid data
@Test
void testCarPost_ValidData() throws Exception {
    System.out.println("Starting test: testCarPost_ValidData");

    // Arrange: Create a valid car with one image
    MockMultipartFile image = new MockMultipartFile(
        "images",
        "test-image.jpg",
        "image/jpeg",
        "test image content".getBytes()
    );

    // Mock the service calls to simulate successful posting
    doNothing().when(carService).saveCar(any(CarModel.class), anyList(), anyString());
    doNothing().when(notificationService).createNotification(anyString(), anyString(), any(NotificationModel.NotificationType.class));
    when(notificationService.getUnreadNotificationCount("john_doe")).thenReturn(0L);

    // Act: Submit the car posting form with valid data, including CSRF token
    mockMvc.perform(MockMvcRequestBuilders.multipart("/cars/post")
        .file(image)
        .param("car.carMake", "Toyota")
        .param("car.carModel", "Camry")
        .param("car.carPrice", "15000.0")
        .with(SecurityMockMvcRequestPostProcessors.user("1").authorities(new SimpleGrantedAuthority("ROLE_USER")))) // Explicitly set user authority
        .with(SecurityMockMvcRequestPostProcessors.csrf()) // Add CSRF token
        .with(request -> {
            request.setMethod("POST");
            return request;
        }))
        // Assert: Verify the response (should redirect to dashboard with success)
        .andExpect(MockMvcResultMatchers.status().is3xxRedirection())
        .andExpect(MockMvcResultMatchers.redirectedUrl("/user/dashboard?submitted"));

    System.out.println("Test testCarPost_ValidData completed successfully");
}

```

Figure 104 testCarPost\_ValidData()

## 7.4 Security Testing

The security test focuses on the CarController and the /cars/post endpoint in the AutoXpress application. The test validates that unauthenticated users are denied access to this endpoint, ensuring proper access control enforcement.

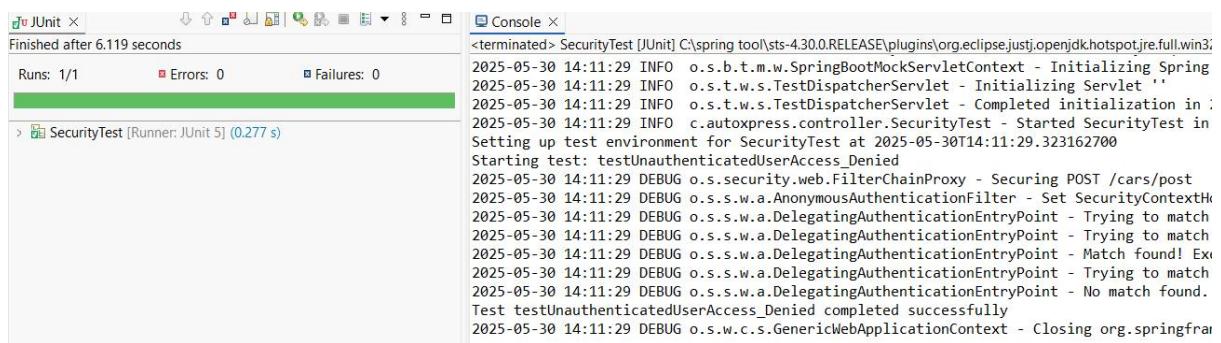


Figure 105 SecurityTest.java Output

#### 7.4.1 Unauthenticated User Access Test

This Test are use to verify that an unauthenticated user attempting to access /cars/post is denied access. It is expected to returns 401 Unauthorized for unauthenticated requests, ensuring access control.

```
// Test 1: Verify that unauthenticated users are denied access to /cars/post
@Test
void testUnauthenticatedUserAccess_Denied() throws Exception {
    System.out.println("Starting test: testUnauthenticatedUserAccess_Denied");

    // Arrange: Create a valid car posting request with one image
    MockMultipartFile image = new MockMultipartFile(
        "images",
        "test-image.jpg",
        "image/jpeg",
        "test image content".getBytes()
    );

    // Act: Submit the car posting form as an unauthenticated user (no user set)
    mockMvc.perform(MockMvcRequestBuilders.multipart("/cars/post")
        .file(image)
        .param("car.carMake", "Toyota")
        .param("car.carModel", "Camry")
        .param("car.carPrice", "15000.0")
        .with(SecurityMockMvcRequestPostProcessors.csrf()) // Add CSRF token
        .with(request -> {
            request.setMethod("POST");
            return request;
        })
    )
    // Assert: Verify the response (should return 401 Unauthorized for unauthenticated user)
    .andExpect(MockMvcResultMatchers.status().isUnauthorized());

    System.out.println("Test testUnauthenticatedUserAccess_Denied completed successfully");
}
```

Figure 106 testUnauthenticatedUserAccess\_Denied()

## 7.5 Database Integration Testing

Integration testing for the AutoXpress application ensures that its components—controllers, services, and repositories—work together seamlessly to perform core functionalities. This phase verifies the integration between the AuthController, UserService, and UserRepository for user registration, ensuring that a new user is correctly saved to the database and the application responds appropriately.

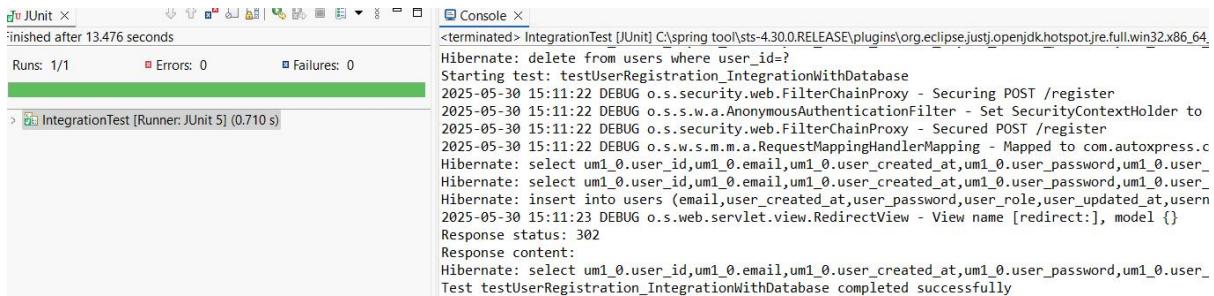


Figure 107 IntegrationTest.java Output

### 7.5.1 Register User Data With Database Test

This Test are use to verify that a user registration request is processed, saved to the database, and redirects to the login page.

```

// Test 1: Verify integration between AuthController, UserService, and UserRepository for user registration
@Test
void testUserRegistration_IntegrationWithDatabase() throws Exception {
    System.out.println("Starting test: testUserRegistration_IntegrationWithDatabase");

    // Arrange: Format LocalDateTime fields for the request
    DateTimeFormatter formatter = DateTimeFormatter.ISO_LOCAL_DATE_TIME;
    String createdAt = LocalDateTime.now().format(formatter);
    String updatedAt = LocalDateTime.now().format(formatter);

    // Act: Submit a user registration request with all required fields
    MvcResult result = mockMvc.perform(MockMvcRequestBuilders.post("/register")
        .param("username", "john_doe")
        .param("email", "john.doe@example.com")
        .param("userPassword", "password123")
        .param("userRole", "USER")
        .param("userCreatedAt", createdAt) // Set required field
        .param("userUpdatedAt", updatedAt) // Set required field
        .with(SecurityMockMvcRequestPostProcessors.csrf()) // Add CSRF token
        .with(request -> {
            request.setMethod("POST");
            return request;
        }))
        .andExpect(MockMvcResultMatchers.status().is3xxRedirection())
        .andExpect(MockMvcResultMatchers.redirectedUrl("/login?success"))
        .andReturn();

    // Debug: Inspect the response
    System.out.println("Response status: " + result.getResponse().getStatus());
    System.out.println("Response content: " + result.getResponse().getContentAsString());

    // Verify that the user was saved to the database
    UserModel savedUser = userRepository.findByUsername("john_doe").orElse(null);
    assertNotNull(savedUser, "User should be saved in the database");
    assertEquals("john_doe", savedUser.getUsername(), "Username should match");
    assertEquals("john.doe@example.com", savedUser.getEmail(), "Email should match");

    System.out.println("Test testUserRegistration_IntegrationWithDatabase completed successfully");
}

```

Figure 108 testUserRegistrationWithDatabase()

## 7.6 Role-Based Testing

The Role-based testing for the AutoXpress application ensures that the application enforces role-based access control, restricting access to endpoints based on user roles. This phase verifies that users with the USER role are denied access to admin-only endpoints and redirected appropriately with an error message.



```
JUnit X
Finished after 13.319 seconds
Runs: 1/1 Errors: 0 Failures: 0
> RoleBasedTest [Runner: JUnit 5] (0.848 s)

<terminated> RoleBasedTest [JUnit] C:\spring tool\sts-4.30.0.RELEASE\plugins\org.eclipse.jdt\openjdkhotspot\jre\full\win32\x86_64_21.0.6.v20250130-0529\jre\bin\javaw.exe !
2025-05-30 15:31:31 DEBUG o.s.w.s.m.m.a.RequestMappingHandlerMapping - Mapped to com.autoxpress.controller.AuthController.register
Hibernate: select uml_0.user_id,uml_0.email,uml_0.user_created_at,uml_0.user_password,uml_0.user_role,uml_0.user_updated_at,uml_0.user_updated_by from user uml_0 where uml_0.user_id = ?
Hibernate: select uml_0.user_id,uml_0.email,uml_0.user_created_at,uml_0.user_password,uml_0.user_role,uml_0.user_updated_at,uml_0.user_updated_by from user uml_0 where uml_0.user_id = ?
Hibernate: insert into user (email,user_created_at,user_password,user_role,user_updated_at,username) values (?,?, ?, ?, ?, ?)
2025-05-30 15:31:32 DEBUG o.s.web.servlet.view.RedirectView - View name [redirect:], model {}
2025-05-30 15:31:32 DEBUG o.s.s.w.c.HttpSessionSecurityContextRepository - Retrieved SecurityContextImpl [Authentication=UsernamePasswordAuthenticationToken[Principal=john_doe, Credentials=[PROTECTED], Authenticated=true, Details=null, GrantedAuthorities=[ROLE_USER]]]
Starting test: testUserRoleAccessToAdminDashboard Redirects
2025-05-30 15:31:32 DEBUG o.s.s.w.c.HttpSessionSecurityContextRepository - Created HttpSession as SecurityContext is non-default
2025-05-30 15:31:32 DEBUG o.s.s.w.c.HttpSessionSecurityContextRepository - Stored SecurityContextImpl [Authentication=UsernamePasswordAuthenticationToken[Principal=john_doe, Credentials=[PROTECTED], Authenticated=true, Details=null, GrantedAuthorities=[ROLE_USER]]]
2025-05-30 15:31:32 DEBUG o.s.security.web.FilterChainProxy - Securing GET /admin/dashboard
2025-05-30 15:31:32 DEBUG o.s.s.w.c.HttpSessionSecurityContextRepository - Retrieved SecurityContextImpl [Authentication=UsernamePasswordAuthenticationToken[Principal=john_doe, Credentials=[PROTECTED], Authenticated=true, Details=null, GrantedAuthorities=[ROLE_USER]]]
Response status: 302
Response content:
Hibernate: select uml_0.user_id,uml_0.email,uml_0.user_created_at,uml_0.user_password,uml_0.user_role,uml_0.user_updated_at,uml_0.user_updated_by from user uml_0 where uml_0.user_id = ?
Test testUserRoleAccessToAdminDashboard Redirects completed successfully
```

Figure 109 RoleBasedTest.java Output

### 7.6.1 Unauthenticated User Access Test

This Test are use to verify that a user with the USER role is redirected when attempting to access /admin/dashboard.

```
// Test 1: Verify that a USER role user is redirected when accessing admin
// dashboard
@Test
@WithMockUser(username = "1", roles = { "USER" }) // Use userId as username, as per userDetailsService
void testUserRoleAccessToAdminDashboard_Redirects() throws Exception {
    System.out.println("Starting test: testUserRoleAccessToAdminDashboard_Redirects");

    // Act: Attempt to access the admin dashboard as a USER role user
    MvcResult result = mockMvc.perform(MockMvcRequestBuilders.get("/admin/dashboard"))
        .andExpect(MockMvcResultMatchers.status().is3xxRedirection())
        .andExpect(MockMvcResultMatchers.redirectedUrl(
            "/user/dashboard?error=Access%20denied.%20You%20do%20not%20have%20permission%20to%20access%20this%20page."))
        .andReturn();

    // Debug: Inspect the response
    System.out.println("Response status: " + result.getResponse().getStatus());
    System.out.println("Response content: " + result.getResponse().getContentAsString());

    // Verify that the user exists in the database (from setup)
    UserModel savedUser = userRepository.findByUsername("john_doe").orElse(null);
    assertNotNull(savedUser, "User should exist in the database");
    assertEquals("john_doe", savedUser.getUsername(), "Username should match");
    assertEquals("john.doe@example.com", savedUser.getEmail(), "Email should match");

    System.out.println("Test testUserRoleAccessToAdminDashboard_Redirects completed successfully");
}
```

Figure 110 testUserRoleAccessToAdminDashboard\_Redirect()

## Task 8 Project Limitations

### 8.1 Lack of Application-Level Validation in Models

Models like UserModel and CarModel lack Jakarta Bean Validation annotations (e.g., @NotBlank, @Email), relying on database-level constraints (nullable = false, unique = true). For example, UserModel's username and email fields are not validated for format or content before database operations.

- **Impact:** Validation errors are caught late at the database layer, **leading to generic error messages** (e.g., database exceptions) and a **poor user experience**. Business rules (e.g., email format, password complexity) are not enforced at the application level, **increasing the risk of invalid data**.

### 8.2 No File Upload Validation in Car Posting

Car posting functionality (e.g., in UserController) does not validate file uploads for type or content.

- **Impact:** Users can upload invalid or malicious files, leading to potential storage issues or security vulnerabilities.

### 8.3 No Client-Side Validation Enforcement

The application relies on server-side validation without enforcing client-side validation for inputs like email or password complexity, despite HTML5 attributes in forms.

- **Impact:** Invalid data submissions increase server load and degrade user experience, as errors are only caught after a server round trip.

## **8.4 No Password Complexity Enforcement**

UserModel and UserService do not enforce password complexity rules, allowing weak passwords despite using BCryptPasswordEncoder.

- **Impact:** Increases security risks, as users can set weak passwords, not meeting industry standards.

## **8.5 No Transactional Rollback for Database Tests**

Tests like IntegrationTest.java and RoleBasedTest.java manually clean up test data without using @Transactional rollback, which could automatically undo database changes.

- **Impact:** Manual cleanup is error-prone, and test data may persist if a test fails before cleanup, affecting subsequent tests.

## **8.6 No Rate Limiting or Throttling**

Endpoints like /register and /login has no rate limiting or throttling.

- **Impact:** Increases risk of brute-force attacks and server overload under high traffic.

## **8.7 No Input Sanitization**

User inputs like username, & email are not sanitized, increasing the risk of XSS attacks.

- **Impact:** Vulnerable to security exploits if inputs are displayed unsanitized in the UI.

## **8.8 Returns to the active after Forms Submission in Sections**

Forms submission in user-dashboard.html and admin-dashboard will keep redirect to the default active section instead of staying in its relevant section

- **Impact:** reduce user experience when forms submission in user-dashboard and admin-dashboard

## **8.9 User cannot update their username**

User cannot update their username while only relying on admin. This is because of username are used for login, even if authentication are using userId. Errors would occur if the user update its username exactly same as other user and this would cause problem where the database will fail to fetch the correct information of the user

- **Impact:** reduce user experience and troubles admin to check if username is unique for update

## Task 9 Future Improvements

### 9.1 Implement CustomValidators in All of the Models

Enhance Model with **appropriate validation annotations** to enforce business rules at the application level before database operations which will provide immediate, user-friendly feedback for **better user experience** and **reduce the database load**.

### 9.2 Add File Upload Validation in CarService

Enhance CarService.saveCar to **validate file types** and **sizes** before processing to **prevents invalid or malicious file uploads**, **reducing security risks** and **ensuring** only supported file **types** are processed such as :

```
if (!List.of("image/jpeg", "image/png").contains(contentType)) {  
    throw new IllegalArgumentException(  
        "Only JPEG and PNG images are allowed."  
    );  
}
```

### 9.3 Add Client-Side Validation with JavaScript

Add **JavaScript validation** to **forms** or **inputs** in all of the **HTML pages** to **enforce** business rules before submission. This **reduces server load** by catching invalid inputs client-side, **improving user experience** with immediate feedback.

### 9.4 Implement Password Complexity Validation in UserService

Add a method in UserService to **validate password complexity** before saving to enforces strong passwords, **improving security** by preventing weak passwords.

## **9.5 Add Transactional and Rollback / Use another Database when Testing**

The first Improvement to be done is to modify all of the Junit Test by **adding @Transactional with rollback**. Transactional will **automatically rolls back** database changes **after each test**, eliminating the risk of test data persistence and reducing test execution time by avoiding manual deletes. Or Else, **configure a separate H2 database** for test that **isolates test data** from production data, **improving test reliability** and safety to avoid affecting the main database.

## **9.6 Rate Limiting on Sensitive Endpoints**

Use a library like Bucket4j to implement **rate limiting on sensitive endpoints** like /register and /login. This is used to **prevents** brute-force **attacks** like DDOS and **reduce server load** by limiting the number of request per minutes.

## **9.7 Implement Input Sanitization with a Filter**

Implement Input Sanitization with by **creating a filter to sanitize** use inputs before reaching the controller. This will encode HTML characters where <script> will become “&lt;script&gt;” to prevent XSS attacks.

## **9.8 Return back to the Section when Forms submissions**

Remove ‘active’ class in html and add ‘active’ to its relevant class for every sections of the forms submission. This will let admin-dashboard and user-dashboard remain at its relevant sections when making forms submissions such as edit user profile or approve car posting.

## **9.9 Set username as unique and Apply username Check**

Set the username as unique in database while ajax username checking in user profile for username update in user profile.

## **Conclusion**

As a conclusion of the report, I will summarize all the task I had done to develop a Used Car Sales Portal which are : encompassing the project introduction, design and implementation of six database-integrated pages, development of eight models and five controllers, and configuration of application properties. I applied Spring Boot to create a robust Model-View-Controller (MVC) architecture, integrated form validation and data handling, and implemented role-based access control for users and administrators. Additionally, I conducted extensive JUnit testing, including functional, UI, validation, security, and database integration tests, while identifying project limitations and proposing future improvements to enhance validation, security, and user experience. This project significantly deepened my understanding of web application development and prepared me for future career opportunities in software engineering.

## **Appendix**

### **Github Link**

Project, <https://github.com/Trae-ralv/Used-Car-Sales-Portal>