ReCarNation: The Ultimate Platform for Buying and Selling Pre-Owned Cars

A Project Report
Presented to
CMPE-282
Spring, 2025

By
Team - Nano Clouds
Ananya Shetty
Apoorva Shastry
Junie Mariam Varghese
Rinku Tekchandani

May 11, 2025

ABSTRACT

The rise of the used car market demands an efficient, secure, and scalable platform for buying and selling pre-owned vehicles. ReCarNation Cloud enhances the existing Django-based web application by migrating it to a cloud-native architecture on AWS, ensuring improved scalability, security, and automation.

The platform features a user-friendly interface where buyers can browse and filter car listings, while sellers can list and manage their vehicles with image uploads. The admin dashboard provides listing approvals, user management, and fraud detection, ensuring a trusted marketplace.

Authentication is Okta-based SSO, integrated with GitHub SSO for development workflow. The backend is deployed on AWS EC2 (Django + PostgreSQL on RDS), while static and media files are managed using Amazon S3. API endpoints are secured and exposed. Continuous deployment is automated with Cloud based Jenkins and Elastic Beanstalk. Additional features include social media integration, document storage, and optional Salesforce CRM connectivity.

This Proof-of-Concept (PoC) demonstrates enterprise cloud migration with serverless elements, security best practices, and automated DevOps workflows.

ACKNOWLEDGEMENTS

We would like to express our sincere gratitude to the individuals and organizations who supported us throughout the development of the ReCarNation-NanoClouds project.

Firstly, we are immensely grateful to Professor Andrew Bond for their Exceptional support, constructive feedback, and continuous encouragement during every stage of this project. Their mentorship played a crucial role in shaping our ideas into a functional and impactful application. We also thank San Jose State University (SJSU) for providing the necessary resource information and a supportive platform to transform this initiative into reality.

We would like to acknowledge the dedication and teamwork of our team members, whose hard work and creative ideas were the driving force behind this project. Every effort played a crucial role in bringing our vision to life.

Special thanks go to the open-source communities and platforms, including Django, PostgreSQL, and GitHub, for their robust tools and detailed documentation. These resources significantly streamlined our development process. Online learning platforms such as Udemy, Coursera, RealPython, FreeCodeCamp, and Stack Overflow were invaluable in providing solutions and enhancing our technical knowledge.

To everyone who contributed directly or indirectly to the successful completion of this project, we thank you for your support and encouragement. Your contributions have been pivotal in achieving our goals.

Table of Contents

List of Figures

List of Tables

CHAPTER 1. INTRODUCTION

1.1 Project goals and objectives

The project aims to streamline the process of buying and selling pre-owned vehicles through automation and cloud-native technologies. The primary goal is to develop a practical, scalable, and user-friendly platform that facilitates seamless communication between buyers and sellers for a secure and efficient experience.

A key focus is on delivering a safe and secure platform, leveraging Okta-based SSO to protect user data and ensure privacy.

To support buyers, the platform offers a robust search and filtering system, along with detailed car listings that aid in transparent and informed decision-making. For sellers, a structured listing process with image uploads enhances ease of use and listing visibility.

The platform is deployed on AWS infrastructure, including Django on Elastic Beanstalk, PostgreSQL via Amazon RDS, and S3 for media and static content. CI/CD is automated through cloud based Jenkins, GitHub Actions, and AWS, ensuring reliable and efficient deployment.

Scalability and adaptability remain central, with support for additional features like fraud detection, social media integration, document storage. The project addresses real challenges in the used vehicle market by offering a secure, automated, and cloud-optimized solution.

1.2 Problem and motivation

The current landscape for buying and selling pre-owned vehicles remains inefficient, with traditional methods and many digital platforms lacking scalability, security, and user experience. Buyers often face unclear listings and limited filtering tools, while sellers struggle with complex listing processes and poor visibility. Inadequate administrative oversight leads to unverified content and diminished trust.

Moreover, existing platforms frequently fail to implement secure and seamless authentication, resulting in privacy risks and fragmented workflows. As user bases grow, performance bottlenecks become common due to lack of cloud scalability.

This project is driven by the need to modernize the pre-owned car market with a secure, automated, and scalable platform. By integrating Okta-based SSO, cloud-native

architecture (AWS EC2, RDS, S3, CloudFront), and automated CI/CD workflows, the platform ensures robust authentication, high availability, and a seamless user experience. Features like advanced search, image-based listings, and administrative controls enhance transparency, trust, and operational efficiency—laying a strong foundation for future growth and innovation.

1.3 Project application and impact

This project holds strong applicability and impact across academic, industrial, and societal domains. Academically, it serves as a practical example of cloud-native deployment using Django, Amazon RDS, S3, and Okta-based SSO. It contributes to understanding scalable architectures, secure authentication, and user-centric design, supporting both learning and future research.

In industry, the platform presents a reliable and adaptable model for automating used car marketplaces. By integrating AWS infrastructure and CI/CD pipelines with tools like Jenkins and GitHub Actions, it ensures high availability, security, and operational efficiency—offering a reference model for similar digital commerce platforms.

Socially, the platform enhances trust and convenience in vehicle transactions, providing a transparent and secure experience for buyers and sellers. It also lays a strong foundation for future innovations such as AI-powered analytics and CRM integration, addressing evolving market demands.

1.4 Project results and expected deliverables

The project results in a functional, cloud-native web application for buying and selling pre-owned vehicles. Key features include Okta-based SSO authentication, a user-friendly listing interface, and admin tools for listing approval and user management. Buyers can browse detailed car listings with filtering options, while sellers can efficiently upload and manage vehicle data with images.

Deliverables include the full Django-based codebase, integrated with AWS services such as EC2, RDS, S3, and CloudFront, along with CI/CD automation via Jenkins, GitHub Actions, and Cloud based Jenkins pipeline. Supporting materials include system documentation, deployment scripts, and a technical report outlining the architecture, features, and implementation of cloud services and security practices. These deliverables showcase a scalable and secure platform designed to modernize the used car marketplace

1.5 Market research

The online used car market is managed by several key players, each offering distinct solutions for buying and selling pre-owned vehicles. Carvana offers end-to-end online car purchasing with home delivery has 10% of the market share which has high operational costs as the concern and can expect delay in the delivery with 7 days return policy.

Vroom, which Specializes in reconditioning and delivering vehicles, has 8% market share and faced quality assurance challenges. In addition, Auto trader aggregates listings from dealers and private sellers which have 15% of the market share and has limited buyer - seller communication as the concern.

Finally, Shift Technologies, which holds about 5% of the market, has carved out its niche by offering unique services like home test drives. Shifts allow customers to schedule test drives at their convenience, enhancing the car-buying experience. The platform also provides financing options, warranties, and detailed inspection reports for every car listed.

Company	**Market Share**	**Key Features**
Carvana	30%	360-degree viewing, home delivery
Vroom	10%	Warranty, nationwide delivery
AutoTrader	15%	Advanced filters, financing tools
CarGurus	8%	Deal ratings, pricing insights, dealer reviews
Shift	5%	Home test drives, financing, warranties

Table 1: Market Share Breakdown

CHAPTER 2. BACKGROUND AND RELATED WORK

2.1 Background and used technologies

This project was developed in response to the growing demand for a secure, scalable, and user-centric platform to support the buying and selling of pre-owned vehicles. As the used car market expands with the need for affordable and convenient transportation, existing platforms reveal issues in security, usability, and performance. This project addresses those gaps with a modern, cloud-native solution.

The platform is built using Django for backend development and is hosted on AWS Elastic Beanstalk to ensure scalability and reliability. It incorporates a user-friendly frontend supported by Amazon S3 for fast content delivery. Secure user authentication is implemented using Okta-based Single Sign-On (SSO), replacing traditional OAuth methods and supporting GitHub SSO for development workflows.

The system uses Amazon RDS (PostgreSQL) for structured data storage, ensuring consistency and integrity across vehicle listings and user information. Automated deployment is achieved through a CI/CD pipeline using Jenkins, GitHub Actions, and Cloud based Jenkins Pipeline, enabling seamless updates and efficient operations. Together, these technologies deliver a reliable, secure, and extensible marketplace tailored to the evolving demands of the automotive sector.

Technologies Used

Frontend: HTML, CSS, JavaScript, Bootstrap, served via Amazon S3 and CloudFront

Backend : Python Django, deployed on AWS Elastic Beanstalk

Database :Amazon RDS (PostgreSQL)

Machine Learning Model: XGBoost, Pandas, Numpy, Matplotlib

Authentication: Okta-based Single Sign-On (SSO), with GitHub OAuth integration

Messaging: SMTP

Version Control: GitHub

CI/CD : Jenkins,GitHub Actions, Cloud Based Jenkins Pipeline

2.2 **State-of-the-art**

The key points in the used car market are to be in favour of the new technologies to enhance user experience and the growth of transparency, and streamline the buying process. Artificial intelligence also plays a crucial role in price optimization, offering more accurate pricing models based on real-time data.

The integration of financing tools and warranties by companies like AutoTrader, and Shift provides added value for users, simplifying the decision-making process for buyers. Additionally, platforms are increasingly using machine learning and data analytics to offer insights into car valuations, help sellers optimize their pricing, and predict the future value of vehicles.

As online buying and selling continue to dominate the market, the convenience of home delivery test drives is driving demand, with companies like Carvana leading the way. These innovations are not only improving customer experience but are also reshaping the overall used car market.

In the end, the state-of-the-art in the used car marketplace is noted by the integration of the latest technologies, enhanced user interfaces, and a customer-first approach. The market is increasingly shifting towards a fully online experience, are key to gaining a competitive edge. The companies mentioned above, with their diverse features and strong market presence, are setting new standards for what consumers can expect from their used car buying and selling experiences.

RecarNation can differentiate itself in the competitive used car marketplace by combining advanced features like 360-degree car views (as seen on Carvana) with the direct buyer-seller interaction seen on platforms like Facebook Marketplace. This blend of technology and flexibility can offer users a unique experience. To build trust and transparency, RecarNation can introduce detailed, structured listings and a verification process for both sellers and vehicles, addressing concerns seen on platforms like Craigslist. Focusing on a user-friendly interface will resolve usability issues present in sites like Autotrader. RecarNation can also ensure scalability by optimizing its back-end infrastructure to handle an expanding inventory, similar to industry leaders. Finally, targeting eco-friendly and pre-owned vehicles can carve out a unique niche, attracting sustainability-conscious buyers in a growing market.

2.3 Literature survey

The literature survey explores key research and projects related to online automotive marketplaces, focusing on areas such as user behavior, trust mechanisms, scalability, and technological advancements. Existing research highlights the factors that influence user satisfaction in these platforms. For instance, a study by Doe and Smith (2020) on consumer preferences in e-commerce automotive platforms identifies transparency, ease of use, and reliability as critical elements for user satisfaction. The study emphasizes the importance of advanced search options, high-quality images, and clear pricing structures in enhancing the user experience. Similarly, Lee and Tan (2019) explored trust mechanisms in peer-to-peer car dealerships, finding that verification processes, user reviews, and secure payment gateways help build trust. Platforms like Carvana and TrueCar leverage return policies and certification programs to reduce buyer hesitation. In terms of scalability, Chen and Kumar (2021) discuss the importance of scalable backend architectures in e-commerce platforms, with cloud services such as AWS and Google Cloud being integral to managing high traffic and dynamic content delivery. Their research underscores the need for caching and database optimization to maintain performance during peak usage periods. Furthermore, Patel and Jones (2022) demonstrate how artificial intelligence tools, such as recommendation engines and predictive analytics, can personalize user experiences, tailoring suggestions based on user behavior.

In addition to academic studies, several research projects have contributed significantly to the field. One such project, developed by the MIT Media Lab, is the Next-Generation Car Marketplace (NGCM), which explored the use of blockchain technology to ensure data integrity and trust without intermediaries. The project showcased how decentralized systems could revolutionize online automotive marketplaces (Brown & Li, 2020). Another important contribution comes from the Stanford HCI Group's project on Enhancing Usability in Automotive E-Commerce. This project focused on UX strategies to simplify complex tasks like advanced search and filtering, showing that clean interfaces and guided workflows improve user retention (Wong & Zhao, 2018). Additionally, the GreenCar Initiative by the University of Cambridge aimed to promote sustainability in automotive marketplaces by encouraging the reuse and recycling of pre-owned vehicles. The project highlighted the environmental benefits and user incentives for choosing pre-owned vehicles, thus contributing to sustainable practices in online automotive sales (Taylor & Singh, 2021). These studies and projects play a significant role in advancing the understanding and development of online automotive marketplaces.

CHAPTER 3. SYSTEM REQUIREMENTS AND ANALYSIS

3.1. Domain and business Requirements

3.1.1. Domain Requirements

The domain requirements define the specific functionalities and features the system must implement to address the unique needs of the pre-owned automobile marketplace.

1. **User Management:**
    - Implement Single Sign-On (SSO) for secure authentication.
    - Allow users to create and manage profiles for buyers, sellers, and admins.
    - Provide account recovery mechanisms for forgotten passwords or locked accounts.
2. **Vehicle Listings:**
    - Enable sellers to create comprehensive listings with vehicle specifications, images, and price predictions.
    - Offer intuitive tools for editing and managing listings post-creation.
    - Archives expired or inactive listings automatically.
3. **Search and Filtering:**
    - Provide advanced search options to filter vehicles by price, make, model, mileage, year.
    - Allow users to save and revisit their favorite searches for convenience.
4. **Communication Tools:**
    - Integrate secure messaging to facilitate buyer-seller negotiations.
    - Enable notifications for new messages, offers, or status changes on listings.
5. **Administrative Oversight:**
    - Implement workflows for approving or rejecting listings to maintain marketplace integrity.
    - Include reporting mechanisms for resolving disputes and addressing flagged content.
6. **Scalability and Performance:**
    - Design the platform to support rapid growth in user base and concurrent usage.
    - Ensure a responsive and seamless user experience across devices.

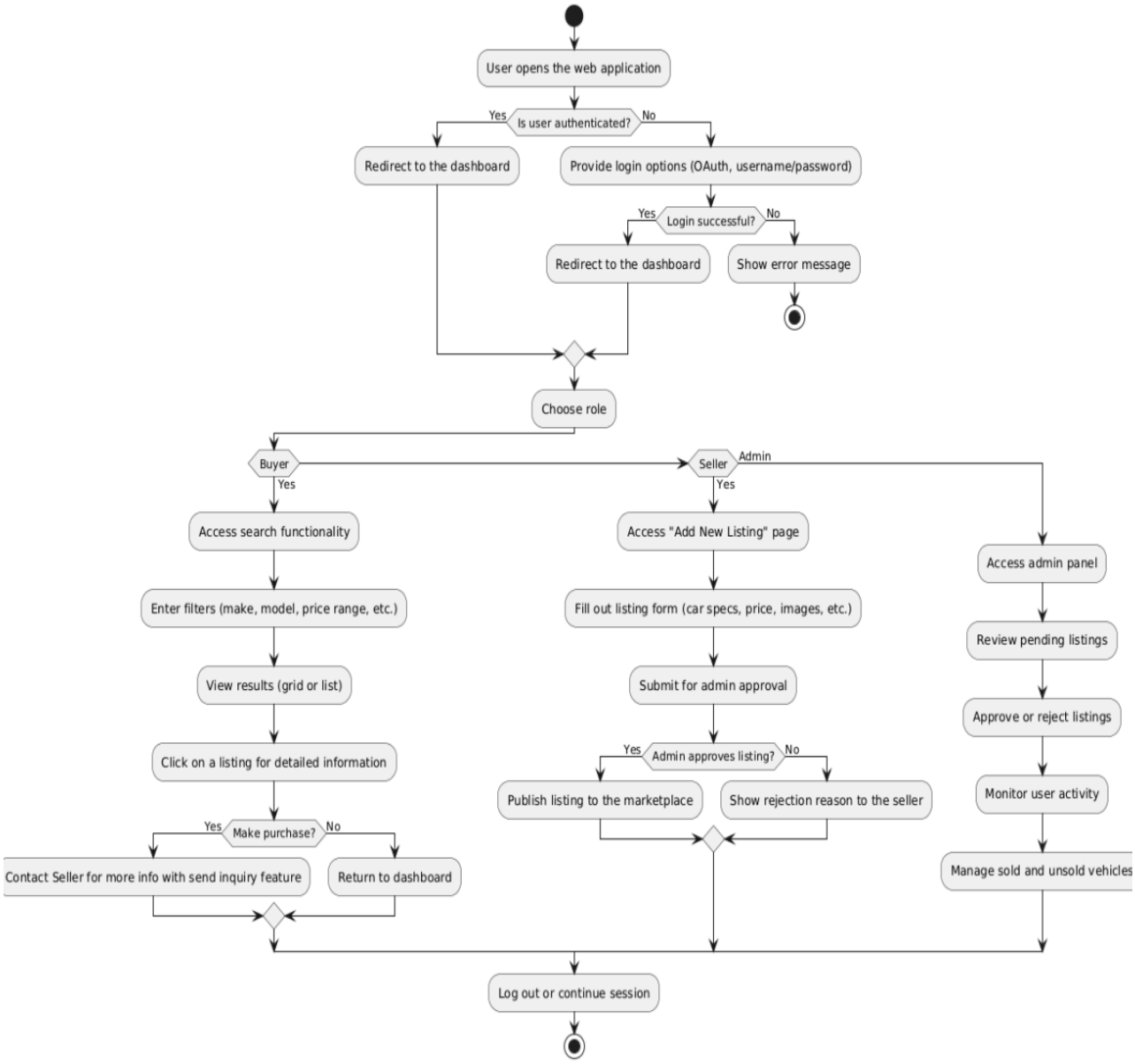

Fig1: Process Summary Activity Diagram

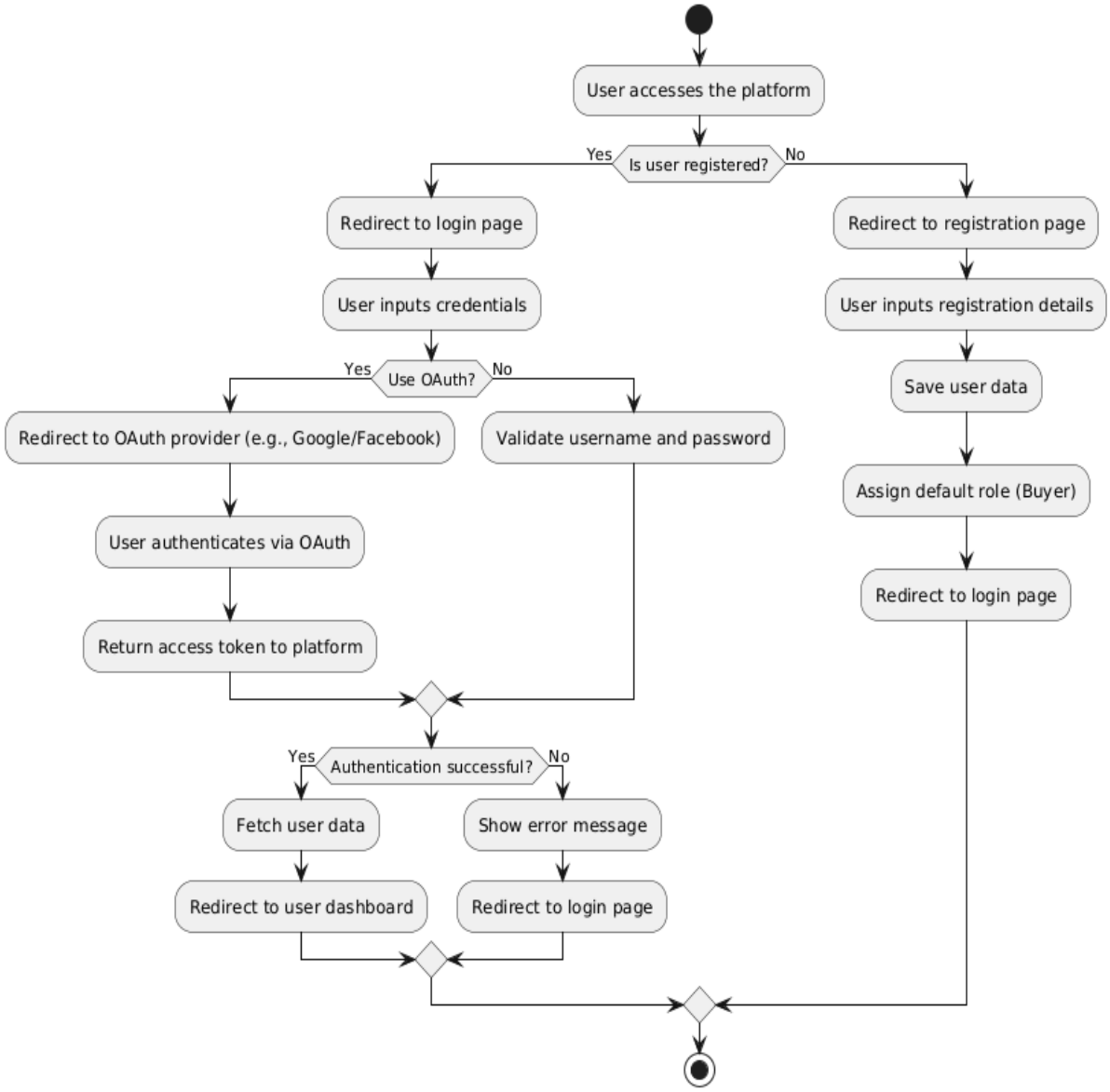

Fig 2: Activity diagram for User Registration and Login

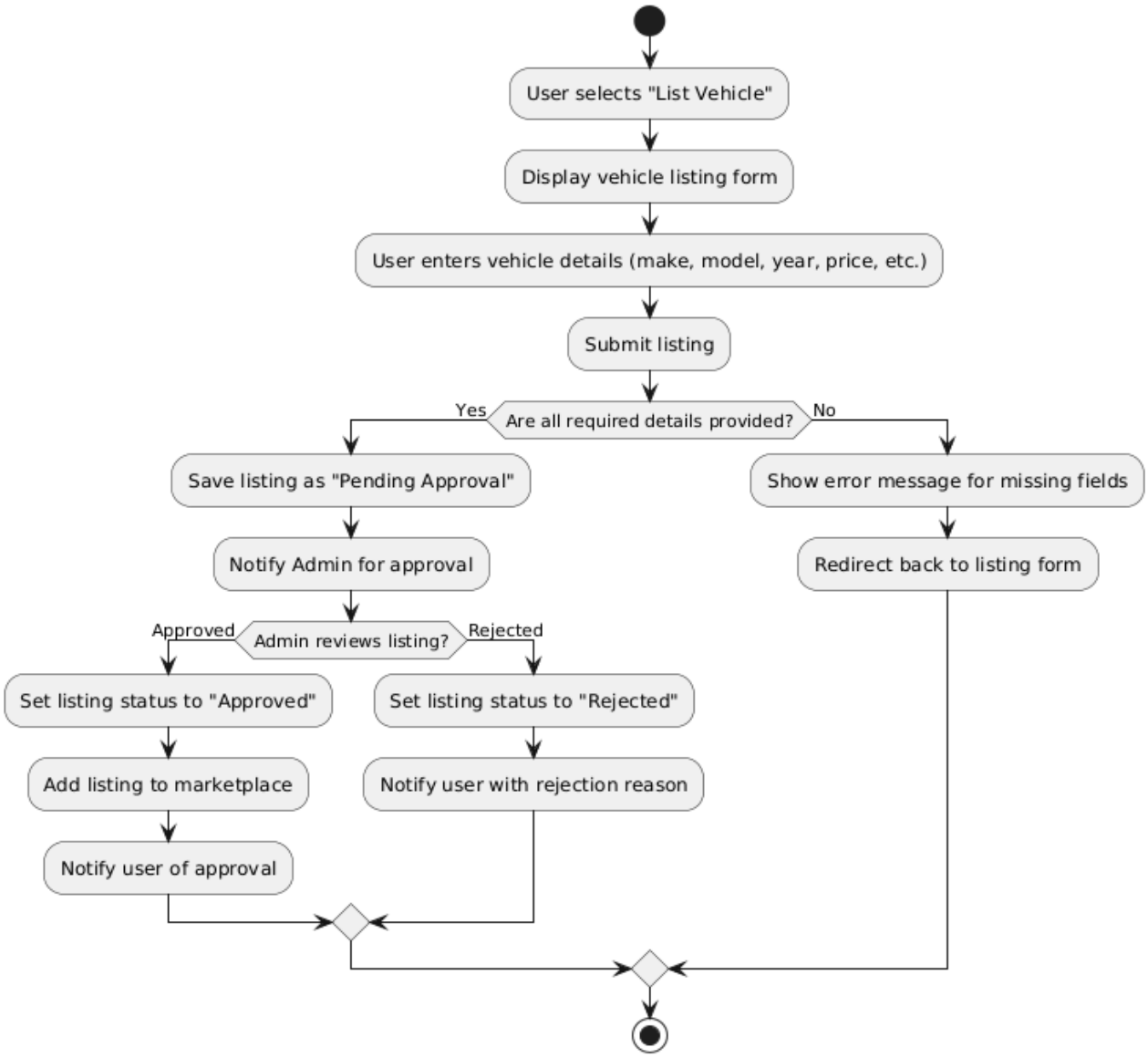

Fig 3: Activity diagram of Vehicle Listing Process

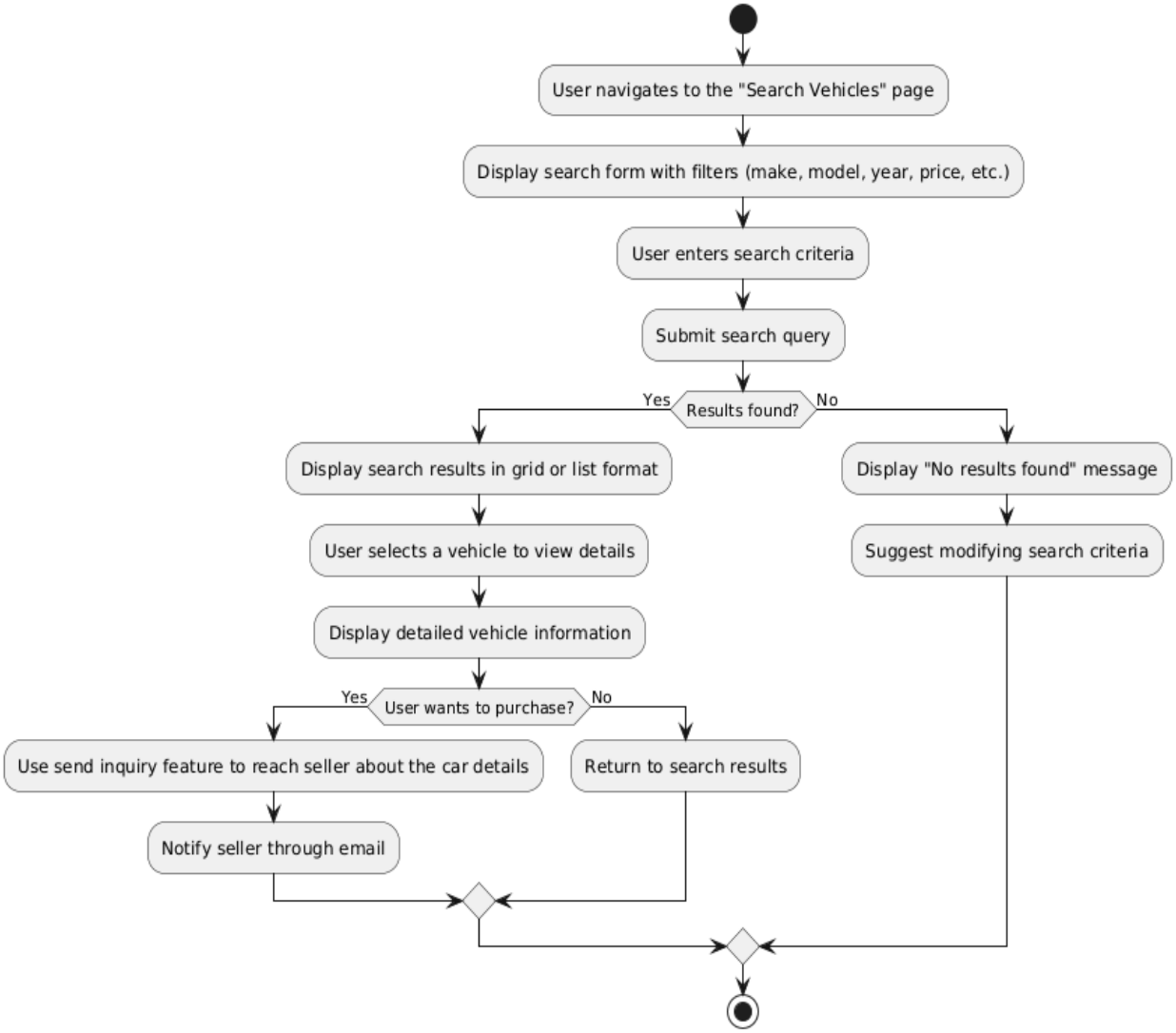

Fig 4: Activity diagram of Vehicle Search and Purchase process

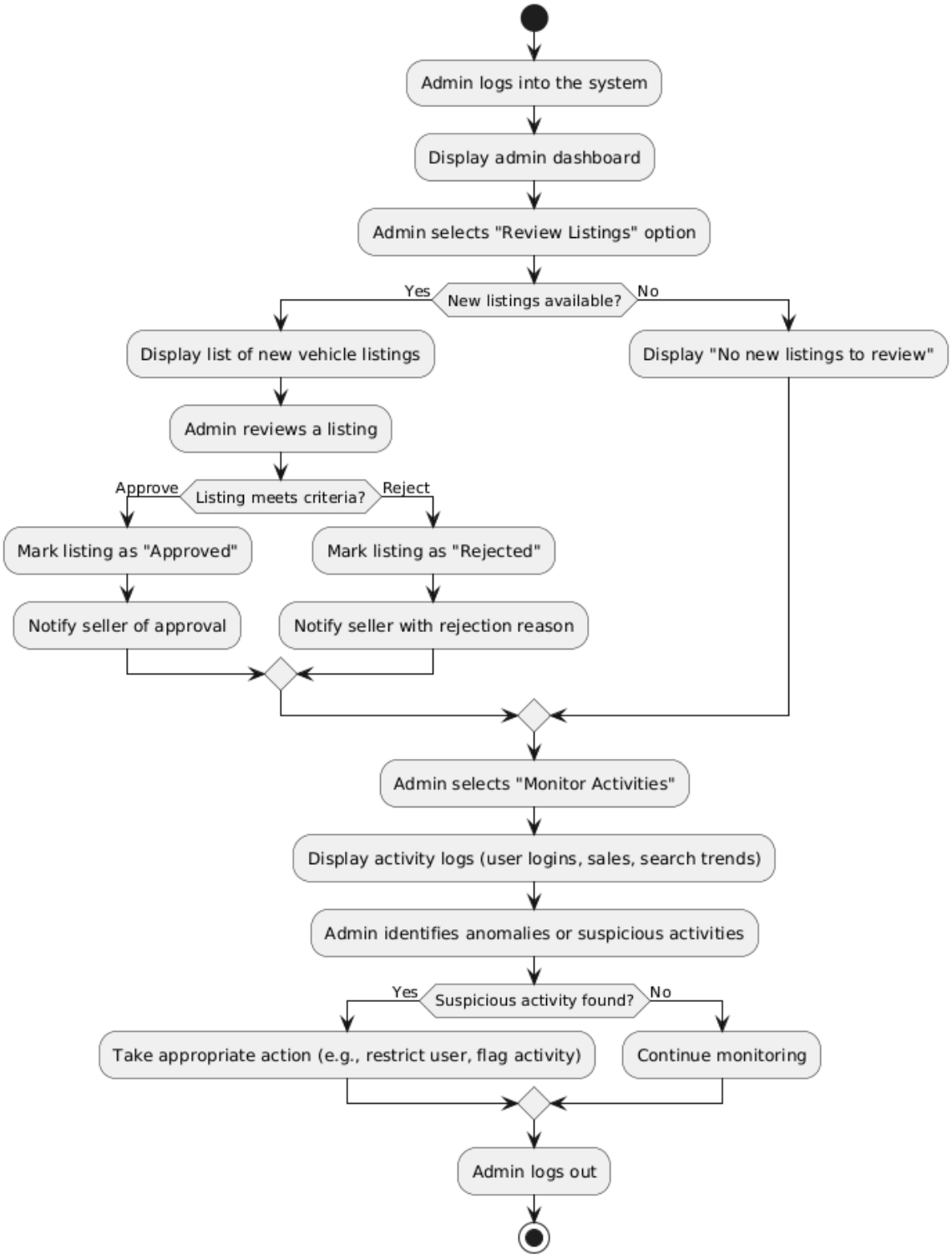

Fig 5: Activity diagram of Administrative Management

3.1.2. Business Requirements

The business requirements focus on aligning the project with the organizational objectives to achieve market success and user satisfaction.

1. **Customer Acquisition and Retention:**
    - Simplify the registration to attract a wide user base.
    - Enhance user experience with an intuitive design and absolute navigation.
2. **Revenue Generation:**
    - Introduce a freemium model where basic features are free.
    - Allow dealerships to advertise and promote inventory for a fee.
3. **Trust and Transparency:**
    - Foster user trust by providing accurate car information, verified listings, and secure transactions.
    - Display price insights using historical and market trends to assist both buyers and sellers.
4. **Compliance and Security:**
    - Ensure compliance with data protection laws like GDPR and CCPA.
    - Implement robust encryption for all user data and transactions.
5. **Market Differentiation:**
    - Introduce innovative features like price prediction, AI-driven recommendations, and detailed car history reports.
    - Build partnerships with insurance and financing companies for added user convenience.
6. **Scalability and Adaptability:**
    - Design a flexible architecture to accommodate future features and integrations, such as AI analytics or expanded payment options.

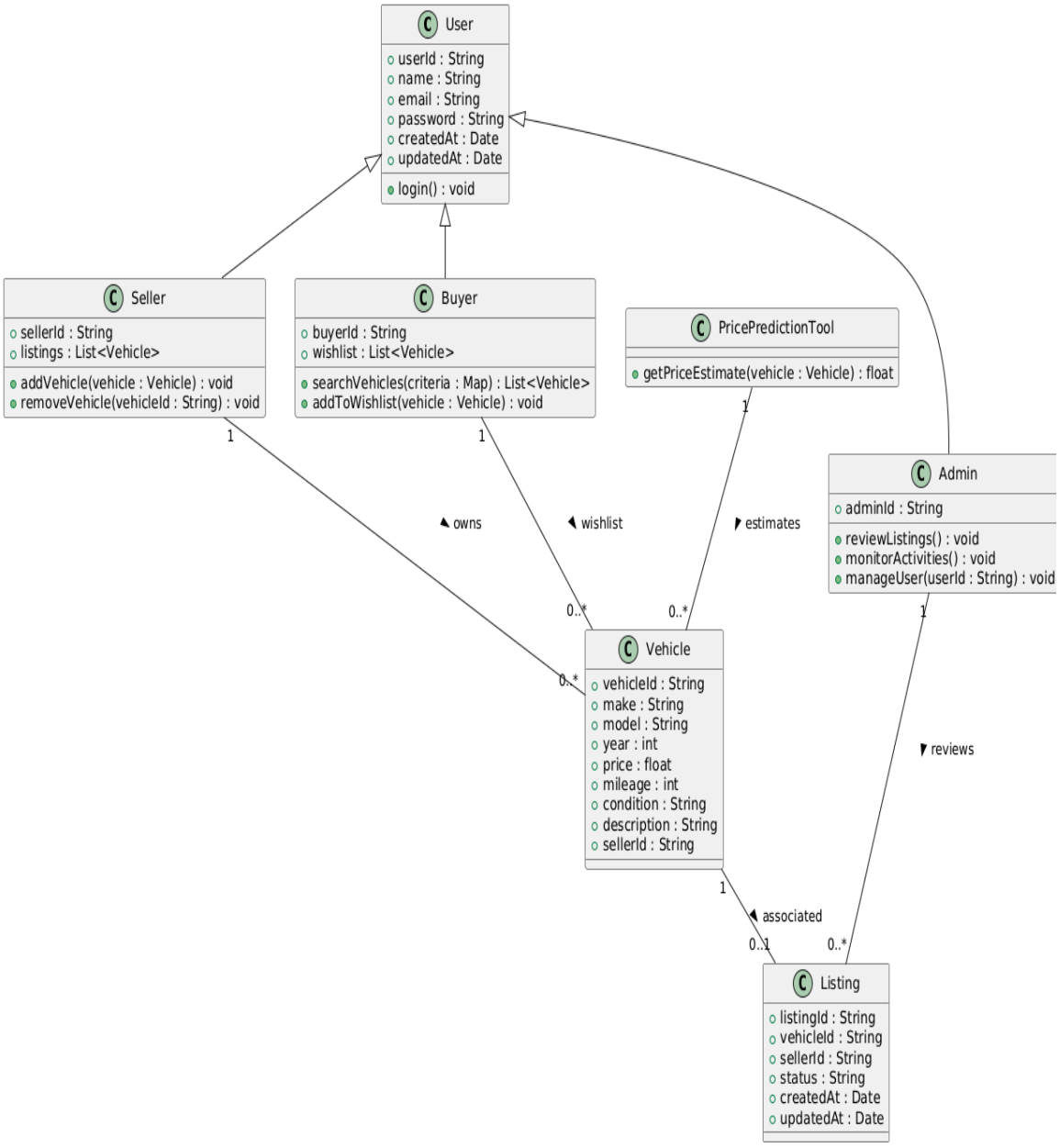

Fig 6: Class diagram

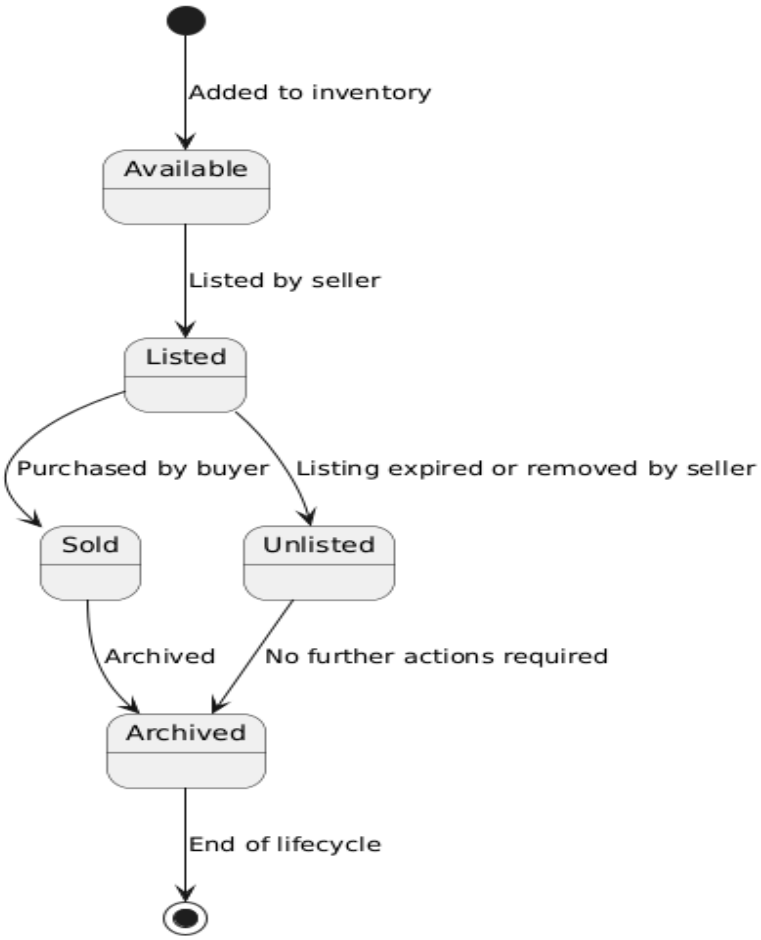

Fig 7: State Machine Diagram for vehicle

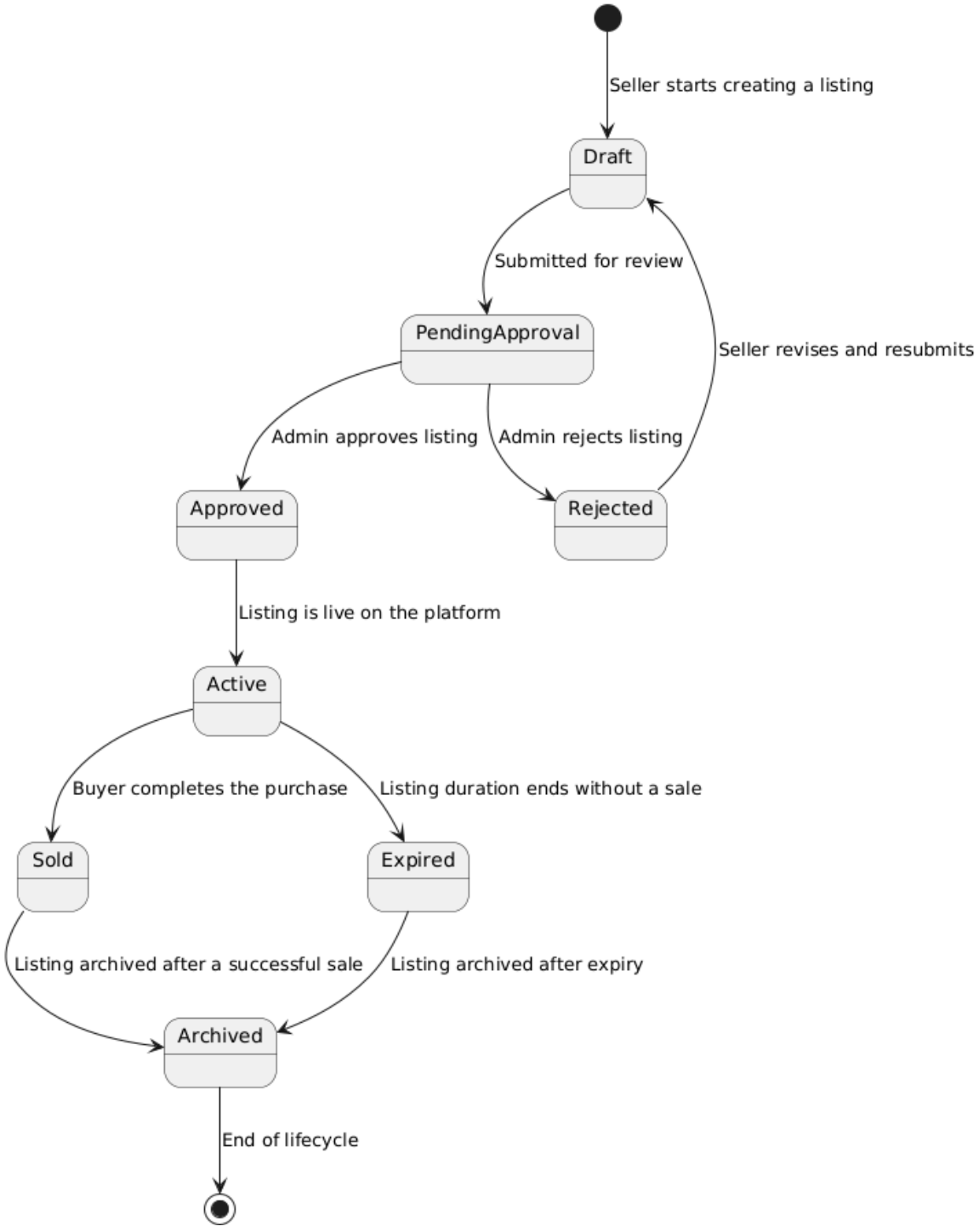

Fig 8: State Machine Diagram for Listing

3.2. Customer-oriented requirements

The customer-oriented requirements for the RecarNation project focus on identifying the primary user groups and defining their respective use cases. This section includes a textual description and a table summarizing the details.

Expected User Groups

1. **Buyers:**
   Individuals looking to purchase used cars on the platform. They require advanced search, filtering, and communication features to evaluate and contact sellers effectively.
   Buyers can filter cars by specific attributes such as make, model, year, mileage, price range, color, fuel type, transmission type, and location. This feature ensures they find vehicles tailored to their preferences.
   Each car listing should include comprehensive information, such as photos, specifications, seller contact details, and inspection history.

2. **Sellers:**
   Individuals or dealerships wishing to sell cars. They need a straightforward interface for listing their vehicles, managing existing listings, and monitoring buyer inquiries.
   A step-by-step guide to list a vehicle, including fields for photos, descriptions, specifications (make, model, mileage, condition), and pricing.
   A feature that provides sellers with an estimated market value of their vehicle based on data such as make, model, and condition.

3. **Administrators:**
   Platform managers are responsible for moderating content, verifying listings, handling disputes, and ensuring smooth platform operations**.**
   Administrators must have the ability to review and approve or reject vehicle listings based on platform guidelines.
   Systems to verify the identities of sellers and validate vehicle details, ensuring authenticity and reducing fraudulent activities

USER GROUP	USE CASE	DESCRIPTION
BUYERS	Search for vehicles	Use advanced filters (e.g., price, brand, mileage) to find vehicles matching preferences.
	Contact sellers	Secure messaging system to negotiate pricing and finalize transactions.
	Bookmark listings	Save favorite listings for future reference.
SELLERS	List vehicles for sale	Input vehicle details, upload photos, and specify pricing.
	Manage listings	Edit, update, or delete active vehicle listings.
	Track performance	View listing views and engagement metrics.
ADMINISTRATOR	Approve/reject listings	Review vehicle details and approve or reject listings.
	User management	Manage user accounts and resolve disputes.
	System monitoring	Oversee platform performance and flag potential issues.

Table 2: User Groups and Related Use Cases

3.3. System (or component) function requirements

The functional requirements define the core system features of the RecarNation platform. These requirements outline the system's expected functionality by specifying inputs, behavior, and outputs.

Functional Requirements

1. User Authentication and Profile Management
    - Input: User-provided credentials or Okta-Based Cloud SSO.
    - Behavior: Validate user credentials and create/update user profiles.
    - Output: Grant or deny access and provide personalized user dashboards.
2. Advanced Search and Filtering
    - Input: Search queries and filter criteria such as make, model, year, and price range.
    - Behavior: Process the query and apply filters to the database.
    - Output: Display relevant car listings matching the search criteria.
3. Car Listing Creation and Management
    - Input: Seller-provided details, including car specifications, condition, and images.
    - Behavior: Validate and store the submitted data in the system.
    - Output: Publish the listing or return errors for corrections.
4. Buyer-Seller Communication
    - Input: Messages sent by buyers or sellers via the platform.
    - Behavior: Facilitate direct and secure communication between users.
    - Output: Deliver messages and maintain a chat history for reference.
5. Content Moderation
    - Input: Car listings and reports submitted by users.
    - Behavior: Review and approve listings or take actions on reported issues.
    - Output: Approved, flagged, or removed listings.
6. Transaction Support
    - Input: Buyer and seller agreement to proceed with a transaction.
    - Behavior: Generate transaction details and maintain a record of the transaction.
    - Output: Confirm transaction completion and update car listing status.

FEATURE	INPUTS	BEHAVIOR	OUTPUTS
User Authentication	Login credentials, SSO token	Verify user identity and provide access.	User is logged in and redirected to dashboard.
Vehicle Search and Filter	Search query, filter criteria	Search database for matching vehicles.	List of vehicles meeting criteria.
Create Listings	Vehicle details, images	Validate and store listing information in the database.	Confirmation of successful listing.
Messaging System	Message content, recipient details	Send and display messages between users.	Notification of sent/received messages.

Table 3: Functional Requirements table

3.4. System performance and non-functional requirements

This section defines the key performance and nonfunctional requirements for the RecarNation project. Non-functional requirements refer to the system overall characteristics, ensuring the system operates optimally, securely, and in compliance with standards.

Performance Requirements

1. System Response Time
    - Requirement: The system should respond to user actions, such as searching or submitting forms, within 2 seconds under normal load conditions.
    - Description: Ensuring fast and responsive interactions will improve user experience.
2. System Throughput
    - Requirement: The system should be able to handle at least hundred concurrent users performing operations like browsing, searching, messaging, etc.
    - Description: The system should support a minimum of 100 users concurrently without significant performance disturbance.

3. Database Query Performance
    - Requirement: Database queries should return results within 1 second for searches or car listings, and 3 seconds for more complex queries.
    - Description: Optimized database queries will improve the speed of fetching listings, performing advanced searches, and displaying detailed pages.

Capacity Requirements

1. Scalability
    - Requirement: The system should scale horizontally to accommodate an increasing number of users.
    - Description: The system must be capable of scaling out to support higher demand during peak times.
2. Data Storage Capacity
    - Requirement: The system should support up to 10TB of user and transaction data.
    - Description: The database should be able to accommodate a growing volume of user profiles, car listings, and transaction records.

Availability Requirements

1. Uptime
    - Requirement: The system should have an availability of 99.9% uptime per month, allowing for maintenance windows.
    - Description: The platform should be reliably available for users, with minimal downtime for updates or maintenance.
2. Disaster Recovery
    - Requirement: The system should have automatic backups with a recovery time objective (RTO) of 1 hour and a recovery point objective (RPO) of 15 minutes.
    - Description: In case of failure, the system should be able to recover quickly with minimal data loss.

Compliance to Standards

1. Legal and Regulatory Compliance
    - Requirement: The system should comply with applicable data protection laws when handling personal user information.
    - Description: Personal data should be processed in accordance with industry regulations to ensure user privacy and security.
2. Accessibility Compliance
    - Requirement: The system should be comfortable with WCAG 2.1 accessibility standards for users with disabilities.

- Description: The platform should be usable by people with various disabilities, providing features like keyboard navigation, screen reader support, and color contrast adjustments.

Security Requirements

1. Data Encryption
    - Requirement: All sensitive data, including user passwords, and personal details, should be encrypted.
    - Description: Encryption of sensitive information ensures that even if data is intercepted, it remains unreadable where as secure..
2. Secure Authentication
    - Requirement: The system should implement multi-factor authentication for users logging in to their profiles.
    - Description: Adding an extra layer of security makes unauthorized users cannot access accounts even if they know the password.
3. Security Auditing and Logging
    - Requirement: The system must maintain audit trail of key actions such as logins, and updates.
    - Description: Logging and auditing are important for identifying and investigating any unauthorized access

REQUIREMENT TYPE	REQUIREMENT DESCRIPTION	PRIORITY	MEASUREMENT
System Response Time	User actions, including searching and form submissions, should complete within 2 seconds.	High	< 2 seconds for 95% of operations
System Throughput	Support up to 100 concurrent users performing operations like browsing and messaging.	High	< 100 users simultaneously without significant lag
Database Query Performance	Query results for listings and search actions should return within 1 second.	High	< 1 second for most queries
Scalability	Must support scaling to handle up to 10,000 active users concurrently.	High	Horizontal scaling for large numbers of users
Data Storage Capacity	Database should handle up to 10TB of data.	Medium	Scalable storage for user data and transactions
Uptime	The system should be available 99.9% of the time.	High	99.9% availability per month
Disaster Recovery	RTO of 1 hour, RPO of 15 minutes.	High	System restores with minimal data loss
Legal Compliance	Must comply with GDPR or CCPA regulations for user privacy.	High	Regular compliance checks

Accessibility Compliance	Compliant with WCAG 2.1 standards to ensure usability for users with disabilities.	Medium	Accessibility testing and audit
Data Encryption	Encrypt sensitive data with AES-256 to protect user and transaction information.	High	AES-256 encryption for sensitive data
Secure Authentication	Implement multi-factor authentication for user login.	High	MFA implementation

Table 4: Non Functional Requirements table

3.5. **System behavior requirements**

System behavior requirements describe how the system should respond to various inputs and states over time. These behaviors are important for understanding the sequence of actions that the system must follow in different scenarios.
In this section, we will use UML state diagrams to visually represent the system's behavior. These diagrams will focus on key processes such as user authentication, car listing creation, and the transaction process.

Key System Behaviors and State Transitions

1. User Authentication Behavior
    - State Description:
        - Initial State: The user is on the login page.
        - Logged Out: The user is not authenticated and must log in to proceed.
        - Logged In: After a successful login, the user is authenticated and redirected to their profile or homepage.
        - Failed Login: If the login credentials are incorrect, the system transitions to this state to prompt the user to try again.
    - State Transitions:

        - From Logged Out to Logged In: When the user enters valid credentials or uses Okta-Based Cloud SSO.
        - From Logged Out to Failed Login: If the login credentials are invalid.
        - From Failed Login to Logged Out: After the user corrects credentials and tries again.

2. Car Listing Creation Behavior
    - State Description:
        - Initial State: The seller is on the "Create Listing" page.
        - Draft State: The user has filled out the car details but has not yet submitted the listing.
        - Submitted State: The car listing has been submitted successfully.
        - Approval Pending: The listing is awaiting admin approval (if applicable).
        - Active Listing: The listing is live and visible to buyers.
        - Deactivated Listing: If the seller decides to remove the listing or if it is flagged by an admin.
    - State Transitions:
        - From Initial State to Draft State: When the seller begins entering car details.
        - From Draft State to Submitted State: Once the seller submits the listing.
        - From Submitted State to Approval Pending: If the listing requires admin review.
        - From Approval Pending to Active Listing: If the listing is approved.
        - From Active Listing to Deactivated Listing: If the seller removes or deactivates the listing.

3. Transaction Behavior
    - State Description:
        - Initial State: Buyer views the car details.
        - Transaction Started: The buyer expresses interest and agrees to the transaction terms.
        - Transaction Pending: The system processes the transaction, verifies details, and finalizes the agreement.
        - Transaction Complete: The transaction is confirmed, and both the buyer and seller are notified.

        - Transaction Cancelled: Either party can cancel the transaction before completion.
    - State Transitions:
        - From Initial State to Transaction Started: When the buyer expresses interest in the car.
        - From Transaction Started to Transaction Pending: When the buyer and seller agree to the terms.
        - From Transaction Pending to Transaction Complete: Once the transaction is successfully processed.
        - From Transaction Started to Transaction Cancelled: If either the buyer or seller cancels the transaction.

State	**Trigger**	**Next State**	**Action/Description**
Logged Out	User enters credentials	Logged In	Validate login and grant access to dashboard or homepage.
Logged In	User logs out	Logged Out	End session and prompt user to log in again.
Failed Login	User enters incorrect credentials	Logged Out	Display error message and prompt user to try again.
Logged In	User navigates away or logs out	Logged Out	End session and log out user.

Table 5: State Diagram Table for User Authentication

State	Trigger	Next State	Action/Description
Initial	User fills out car listing details	Draft State	Car details are entered but not yet submitted.
Draft State	User submits car listing	Submitted State	Validate and store car listing.
Submitted State	Admin reviews listing	Approval Pending	The listing is waiting for admin approval.
Approval Pending	Admin approves listing	Active Listing	Listing is visible to buyers.
Active Listing	Seller or admin deactivates listing	Deactivated Listing	Remove or deactivate the listing.

Table 6: State Diagram Table for Car Listing Creation

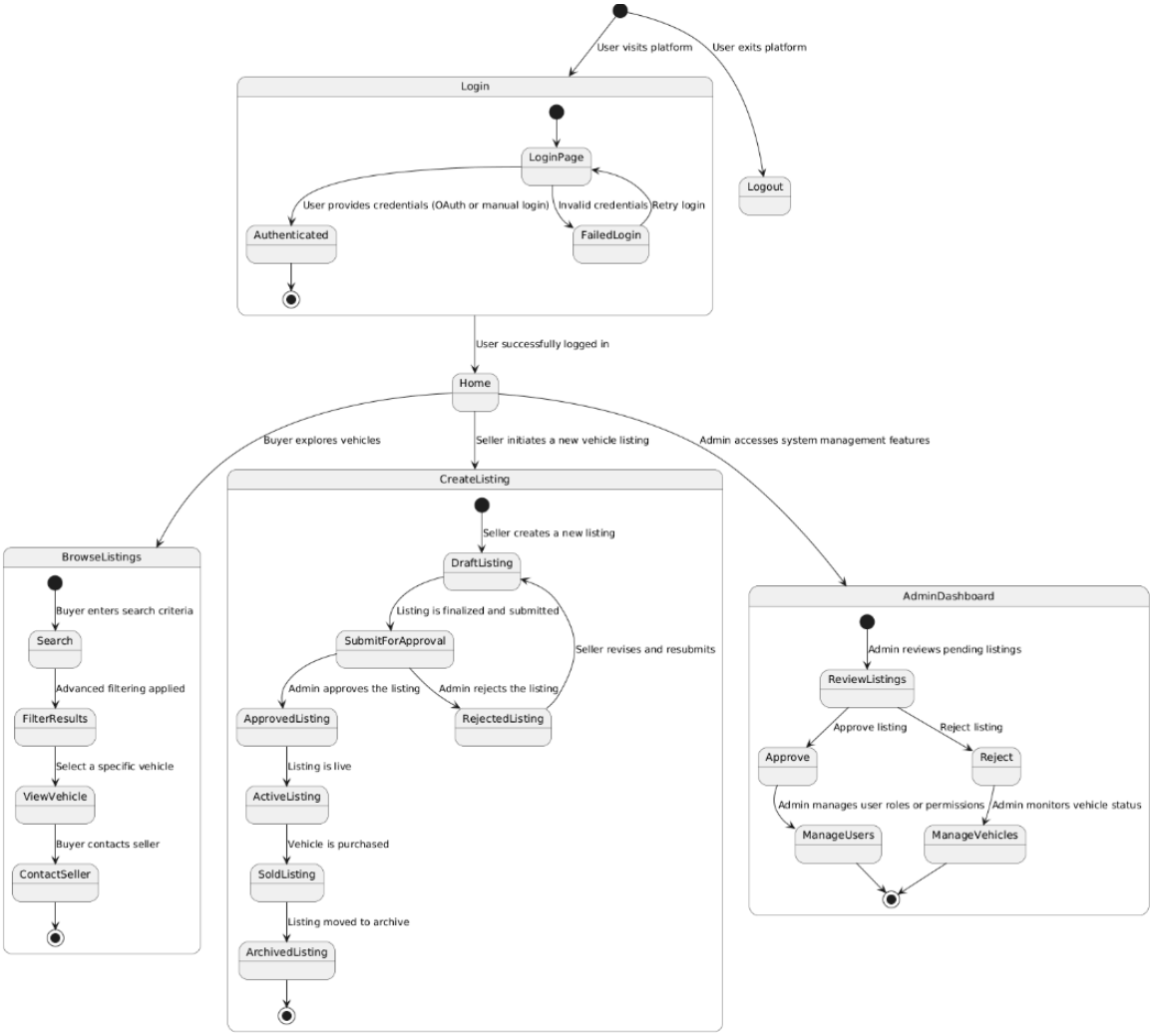

Fig 9 : State diagram for system-level behaviors

3.6. Context and interface requirements

This section specifies the environments and systems necessary to support the development, testing, and deployment of the RecarNation project. Additionally, it covers the system interface requirements, including the user interface structure and design.

Context Environment Requirements

The RecarNation project requires specific environments and tools to function effectively during development, testing, and deployment. These environments include both software and hardware configurations required to run the project successfully.

1. Development Environment
   - Software:
     Backend Framework: Django (Python) deployed on Elastic Beanstalk
     Frontend Framework: HTML5 with CSS served via Amazon S3 and CloudFront.
     Database: PostgreSQL on Amazon RDS (used for storing user data, car listings, transactions, etc.)
     Version Control: Git (GitHub repository for collaboration and source control)
     Integrated Development Environment (IDE): Visual Studio Code, PyCharm, or IntelliJ IDEA
     Python Version: 3.8 or above
   - Hardware:
     Local Development Machine: A MacBook, Windows, or Linux machine with sufficient RAM (8GB or more) and CPU power (minimum 2 cores).
2. Testing Environment
   - Software:
     Testing Frameworks:
       - Backend Testing: Pytest or Django's built-in testing tools for unit and integration testing.
       - Frontend Testing: Selenium for testing components and interactions.

     Continuous Integration (CI)/Continuous Deployment (CD): GitHub Actions Jenkins for automating testing and deployment.
     Load Testing Tools: Apache JMeter or Locust for load and performance testing.
   - Hardware:
     Test Servers: Staging servers replicating production environments where automated and manual tests are conducted.

3. Deployment Environment
- Web Server: Django web server for deploying.
- Database Service: PostgreSQL running on Django framework.

System Interface Requirements

This section describes the system interfaces required to interact with external systems and the interface requirements for internal components (such as database, authentication service, etc.)

1. **User Interface (UI) Requirements**

The RecarNation platform's user interface is designed to provide an intuitive experience for both buyers and sellers. The UI is optimized for simplicity and responsiveness, ensuring accessibility across various devices, including desktop and mobile platforms.

Key UI Features:

1. Homepage Layout
    - Components:
        - Navigation Bar: Links to "Home," "Buy Cars," "Sell Cars," "Login/Signup," and "Contact Us."
        - Search Bar: For searching cars based on make, model, price range, and other filters.
        - Car Categories Section: Display categories like "Sedans," "SUVs," and "Trucks."
        - Featured Listings: A section showcasing popular or featured cars.
    - User Flow:
        - Users can view the homepage, search for cars, or navigate to specific car listings.
2. Buy Page
    - Components:
        - Filter Sidebar: Filters for price, make, model, year, mileage, etc.
        - Car Listings Grid: A grid or list view to display cars based on search criteria.
        - Pagination: For browsing large numbers of listings.
    - User Flow:
        - Users can refine their search and view car details by clicking on individual listings.
3. Sell Page
    - Components:

Car Details Form: Sellers can enter car details such as make, model, year, price, and mileage.

Submit Button: To submit the listing for review and approval.

- User Flow:

  Sellers fill in car details and submit the listing, which will be displayed after admin approval.

4. Login/Signup Pages
   - Components:

     Cloud SSO (Single Sign-On): Okta based cloud sso

     User Credentials Form: For users to enter username and password for traditional login.

     Error Message: If login fails, display an appropriate error message.
   - User Flow:

     New users can sign up, and existing users can log in to access their profiles.
5. Car Detail Page
   - Components:

     Car Specifications: Showcasing detailed information like make, model, year, mileage, price, and condition.

     Seller Contact Info: Information about the seller for direct messaging or calls.
   - User Flow:

     Buyers can review the car details and decide whether to make a purchase.
6. Price Prediction page
   - Components:

     Car Details Form: Sellers can enter car details such as make, model, year, price, and mileage.
     - Submit Button: To submit the listing for review and approval.
   - User Flow:

     Buyers/Sellers can review the predicted price and decide whether to make a purchase.

Page	**Description**	**Components**	**User Action**

Homepage	Displays general information, featured cars, and search bar for buyers.	Navigation, Search Bar, Featured Listings	View cars, search cars, navigate to other pages.
Buy Page	Allows users to browse cars based on filters.	Car Grid/List, Filter Sidebar, Pagination	Filter and view cars, click for details.
Sell Page	Sellers can submit their car listings.	Form for Car Details, Submit Button	Fill details and submit listings.
Login	For user authentication through SSO or credentials.	Login Form, SSO Buttons, Error Message	Login or sign up, retry on failure.
Car Detail Page	Provides detailed information on a specific car listing.	Car Specs, Contact Info, Image Gallery	View details, contact the seller, or initiate purchase.
Price Predictor Page	Provides the predicted price of the car details provided.	Form for Car Details, Submit Button	Fill details and submit listings.

Table 7 : Interface design table

3. External System Interfaces.

- Third-Party Authentication:
  System: Okta-Based Cloud SSO for Single Sign-On.
- Email and SMS Notification Service:
  System: SMTP for sending email notifications.

3.7. Technology and resource requirements

This section outlines the technologies and resources necessary to develop, test, deploy, and maintain the RecarNation project. The requirements are classified into various categories to provide a clear understanding of the tools, platforms, and resources utilized.

1. Software Requirements

The development and deployment of the ReCarNation Cloud platform require the following software tools and technologies:

Development Frameworks and Tools

- **Backend Framework**: Django (Python) on Elastic Beanstalk
  Handles business logic, API development, and database interactions.

- **Frontend**: HTML, CSS, JavaScript on Amazon S3
  Used for building a responsive, user-friendly interface; served via Amazon S3 and CloudFront.

- **Database Management System**: Amazon RDS (PostgreSQL)
  Provides secure, scalable storage for user data, listings, and transactions.

- **Authentication Service**: Okta-based SSO with GitHub OAuth
  Ensures secure and seamless login, replacing traditional OAuth 2.0.

- **API Framework**: Django REST Framework (DRF)
  Enables efficient communication between frontend and backend services.

- **Version Control**: GitHub
  Facilitates source code management and collaborative development.

- **Package Manager**: pip
  Manages Python libraries and dependencies.

- **Orchestration** : Docker

Testing Tools

- **Unit Testing**: Pytest, Django's built-in test suite

- **API Testing**: Postman
  Performance Testing: Locust or Apache JMeter

- **UI Testing**: Selenium

Deployment and CI/CD Tools

- **Hosting and Deployment**:
    - Backend hosted on **AWS Elastic Beanstalk**
    - Static/media files served via **Amazon S3**
- **CI/CD Pipeline**:
    - **Jenkins**, **GitHub webhooks** for automated build and deployment workflows

2. Hardware Requirements

Development Machines

- Minimum Specifications:

  CPU: Quad-core processor
  RAM: 8GB
  Storage: 256GB SSD
  OS: macOS or Windows

Local Testing Machines:

- Devices for testing UI compatibility:

  Desktop: Windows and macOS

4. Resource Requirements

Third-party Services

- Notification Service: SMTP (email notifications).

CHAPTER 4. SYSTEM DESIGN

4.1 System architecture design

The architecture of the used car marketplace web application is designed to be scalable, secure, and efficient. This diagram illustrates the cloud-native architecture of the ReCarNation platform, hosted entirely on AWS.Below is a breakdown of the system architecture and the relationships between various components:

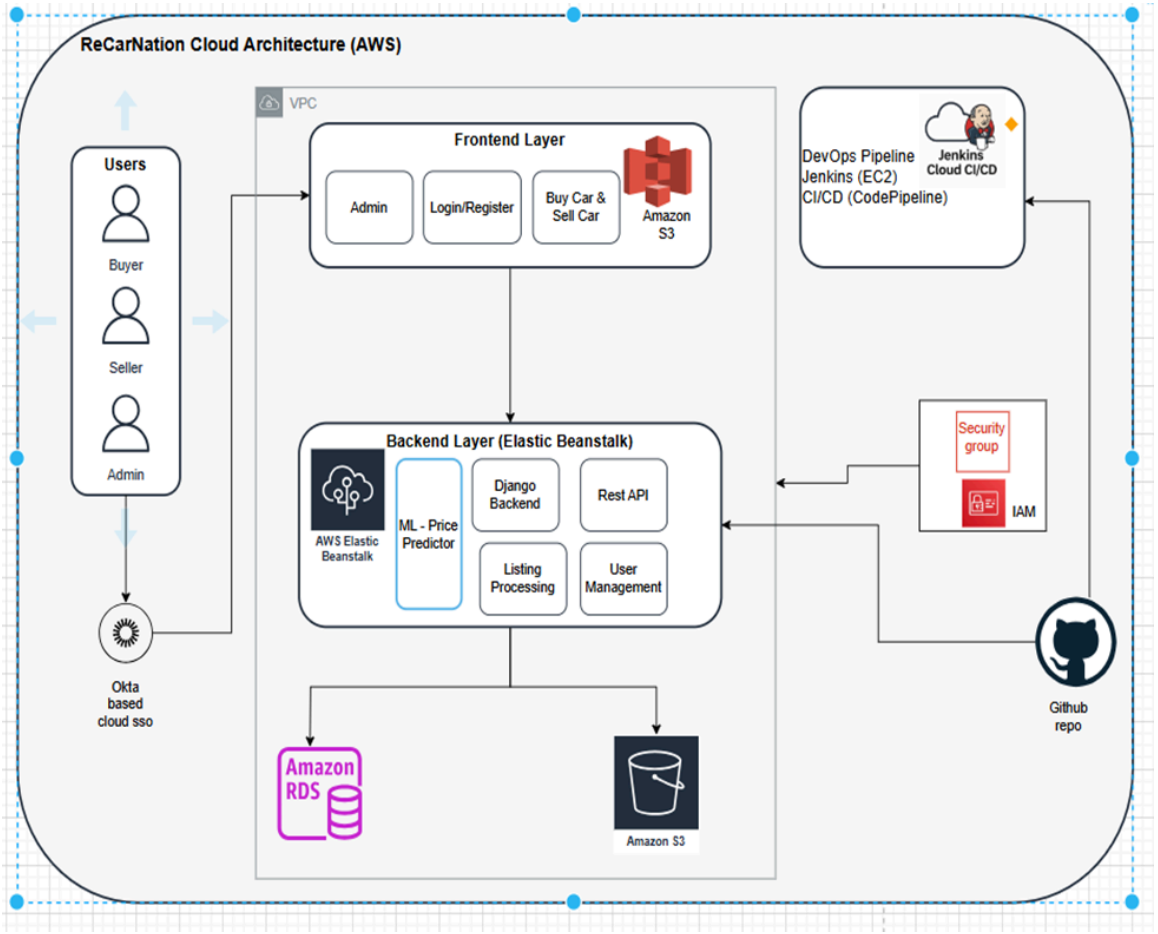

Fig 10 : System Architecture Diagram

Client-Side (Frontend Layer)

The web interface is developed using HTML, CSS, JavaScript, and Bootstrap, offering a responsive and user-friendly experience for buyers, sellers, and admins. It is served via Amazon S3, allowing scalable, low-latency access to static assets. Users authenticate

through Okta-based cloud SSO, providing secure and seamless login experiences. Buyers can search and filter vehicles, view detailed listings, and receive real-time price suggestions. Sellers can list vehicles and manage their content, while admins can log in through a secured interface for moderation and management.

Application Layer (Backend on Elastic Beanstalk)

The backend is built with Django and hosted on AWS Elastic Beanstalk, providing an auto-scaled and managed environment. It acts as the core processing layer, handling:

- Car listing operations – create, update, and delete vehicle listings.
- User management – handles roles, authentication, and permissions.
- REST APIs – facilitate communication between frontend and backend.
- ML Price Prediction – integrates a trained XGboost model for estimating vehicle prices.
- Admin controls – listing approvals, content moderation, and user oversight.

Data Layer (Amazon RDS and S3)

All structured data is stored in Amazon RDS (PostgreSQL), which ensures durability, automated backups, and encryption at rest and in transit. It holds user profiles, car listings, transaction data, and pricing history. Amazon S3 is used to store user-uploaded images and media files, offering secure and scalable storage with high availability.

Price Prediction Engine

A machine learning-based price predictor, integrated within the backend, leverages historical data, vehicle specifications (make, model, year, mileage), and market trends. It uses a XGBoost model to return optimal price estimates, helping sellers make data-informed pricing decisions.

Admin Panel

The admin dashboard is embedded in the Django backend and accessible through secure login. Admins can:

- Approve or reject car listings
- Manage user roles and permissions
- Monitor system usage and platform integrity
- Moderate content to maintain trust and quality

Third-Party Authentication

User login is managed through Okta SSO, with support for social login providers like GitHub and Google, enhancing user convenience and account security.

Continuous Integration & Delivery (CI/CD)

A robust Jenkins CI/CD pipeline, hosted on Amazon EC2, automates code integration, testing, and deployment using Cloud based Jenkins Pipeline. It:

- Pulls code from GitHub
- Verifies builds in Dockerized Jenkins agents
- Deploys updates to Elastic Beanstalk with permission checks and retry logic
  This pipeline ensures that updates are delivered reliably and efficiently, minimizing manual errors and deployment downtime.

4.2 System data and database design

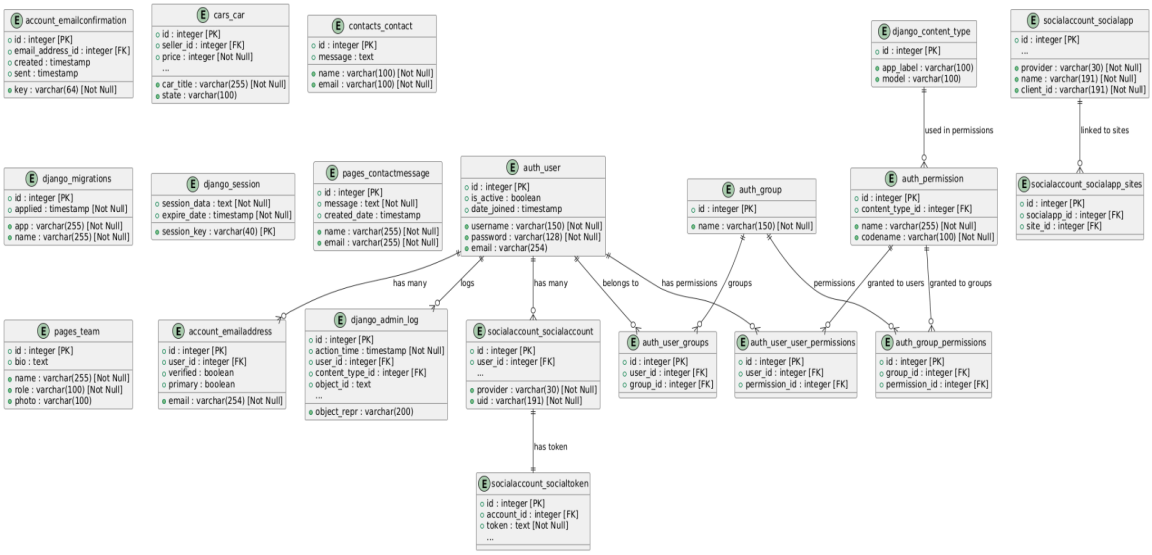

Fig 11: ER Diagram for the system data and database

The cardealer_db database is designed to support a car dealership platform by organizing user accounts, car listings, contact management, and social account integrations. Its modular structure ensures logical separation of responsibilities. Authentication and authorization functionalities are managed through tables like auth_user, auth_group, and auth_permission, which also facilitate role-based access control. Car listings are central to the database, with the cars_car table capturing essential details such as car title, price, model, year, and seller information. Optional multimedia fields for photos allow the inclusion of rich content for each listing. Contact management is handled through contacts_contact and pages_contactmessage tables, which store inquiries and messages sent by users, enabling communication between buyers and sellers. Social account integration tables under the socialaccount prefix provide modern login mechanisms, supporting third-party logins from platforms like Google.

The database design is normalized to eliminate redundancy and ensure data consistency. For instance, user details are centralized in the auth_user table and referenced across related tables like account_emailaddress and socialaccount_socialaccount. Similarly, permissions and roles are structured into dedicated tables (auth_permission, auth_group) and linked through many-to-many relationships for flexible role management. Relationships between entities are clearly defined, such as the one-to-many relationship

where a user can own multiple cars, or the many-to-many relationship between users and groups for role assignments.

This design is highly scalable, allowing the system to handle increasing data volumes efficiently. It supports multiple car listings per seller, manages growing contact inquiries independently, and integrates seamlessly with third-party login providers. By organizing data into distinct modules, the database ensures ease of maintenance, consistency, and adaptability for future requirements, making it well-suited for the demands of a modern online car dealership platform.

4.3 System interface and connectivity design

a) the external interfaces to third-party systems/components, and system user interfaces

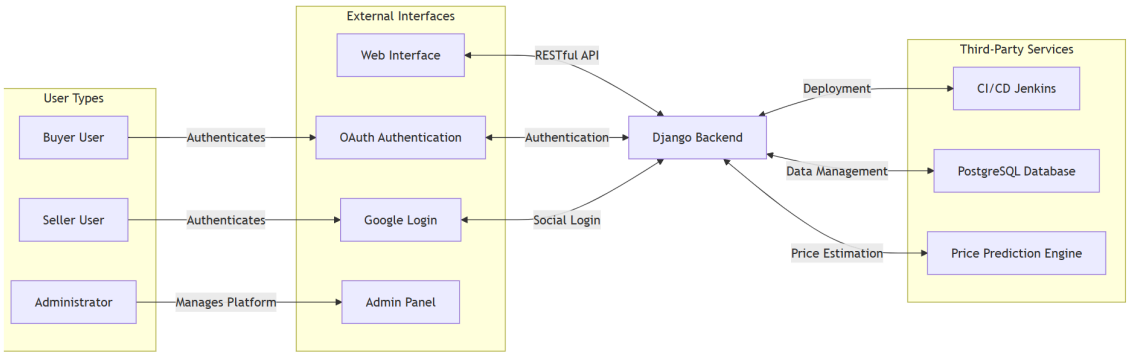

Fig 12: External interfaces and System User Interfaces

b) System connectivity protocols.

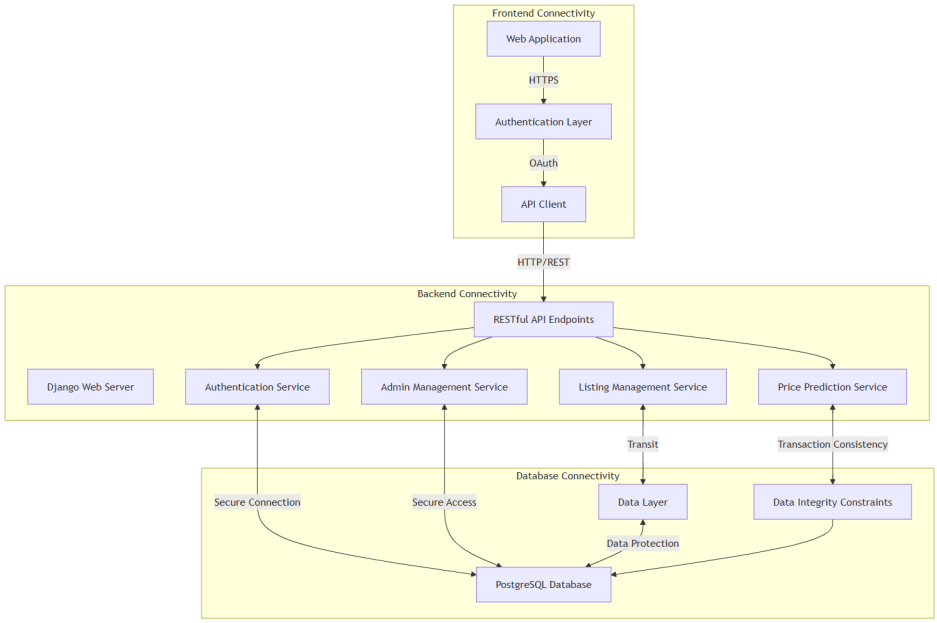

Fig 13: System Connectivity Protocol

Component Connectivity and Flow

1. **User Access**: Users interact with the frontend interface, which communicates with the backend through RESTful APIs. The frontend sends requests for car listings, user data, and transaction details, while the backend processes these requests, interacts with the database, and returns appropriate responses.
2. **Authentication**: The authentication process involves the frontend sending authentication requests to the backend, which verifies credentials using OAuth and issues a secure token for session management.
3. **Price Prediction**: When a seller submits a listing, the backend triggers the price prediction engine to provide an estimated selling price, which is then displayed to the seller.
4. **Admin Controls**: Admins interact with the admin panel to review and manage users and listings. The admin interface communicates directly with the backend to retrieve and update data.

4.4 System user interface design

This section includes the initial draft of the intended user interface (UI) for the system, which has been designed to be simple and efficient, yet use friendly. The objective here is

to present an essential navigational platform for buyers and sellers and administrators alike in their efforts to provide a smooth, clean, and easily accessible site.

Graphical User Interface (GUI) Structure

The platform comprises such important pages designed and tailored to the different specific requirements:

- **Landing Page**
  The landing page put login and signup on the forte screen.
- **Dashboard**
  The dashboard now personalized itself as a buyer, seller, administrator, and notification for important actions, plus quick-link features with a user-by-user perspective.
- **Vehicle Listings Page**
  Buyers can browse cars in grid or list view, and they can also make use of the advanced filters based on the price, model, mileage, and year attributes, as well as sorting mechanisms attached to it, to make searching a lot easier.
- **Create a Listing**
  A simple step-by-step process will be followed for sellers to create a listing for their vehicle. They are expected to put in their car specifications, upload images alongside it, and even offer pricing suggestions to make a competitive listing.
- **Admin Control Panel**
  An admin panel where all management of roles given to users with reviewing and approving or rejecting listings of vehicles can be expressed to oversee all activities on the platform.

Operation Flow

In fact, the interaction flow is designed to be simple and efficient for every category of users:

- **Buyers**
  Start with logging in.
  Browse vehicle listings using search and filtering tools.
  View and access information regarding selected vehicles.
- **Sellers**
  Login to access the dashboard
  Commences with creating a vehicle listing entry

Posted for admin approval once completed
Check the status of the listing and edit if necessary

- **Administrators**
  Log in to visit the control panel.
  Checks all pending listings and either approves or disapproves it.
  Manages user rights and ensures healthy operation of the platform.

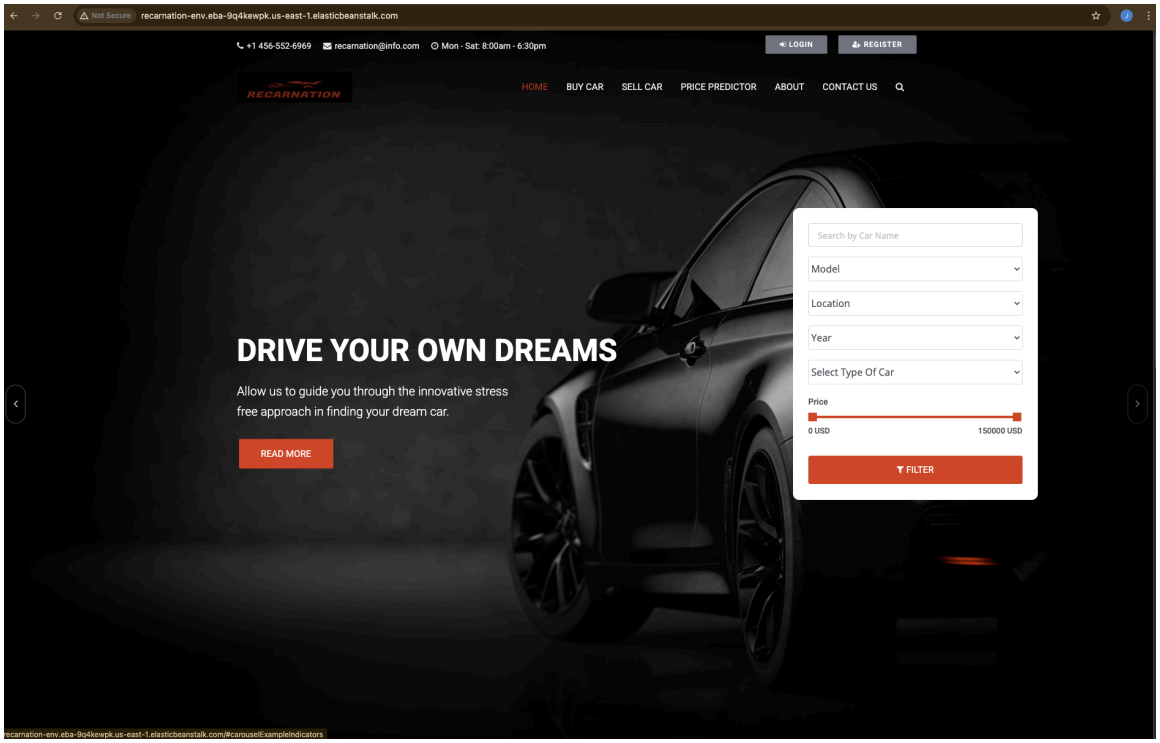

Figure 14: Home Page

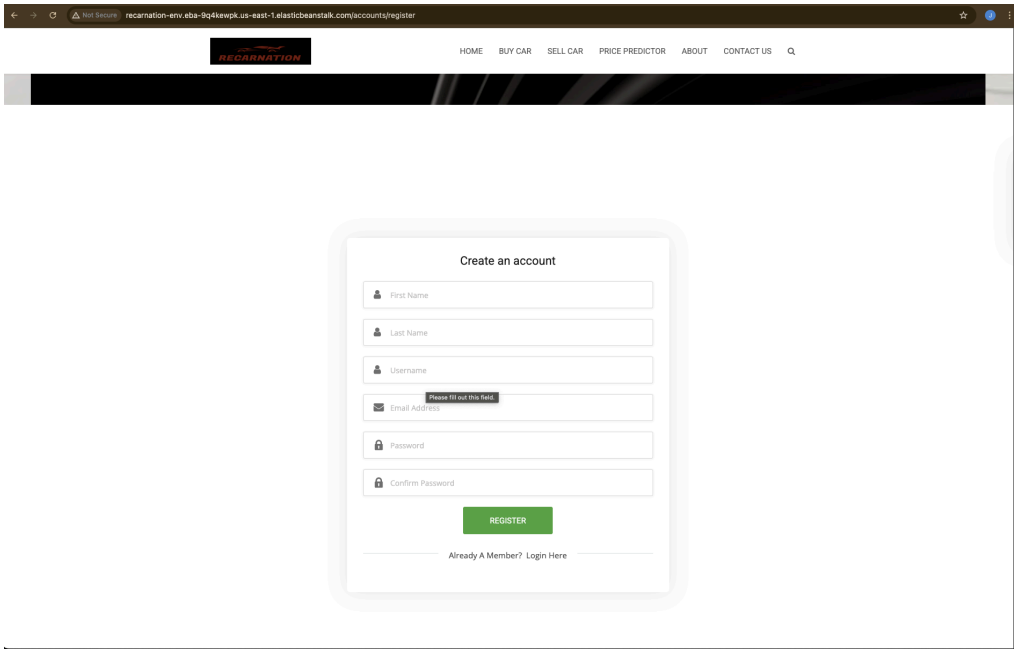

Figure15: Register Page

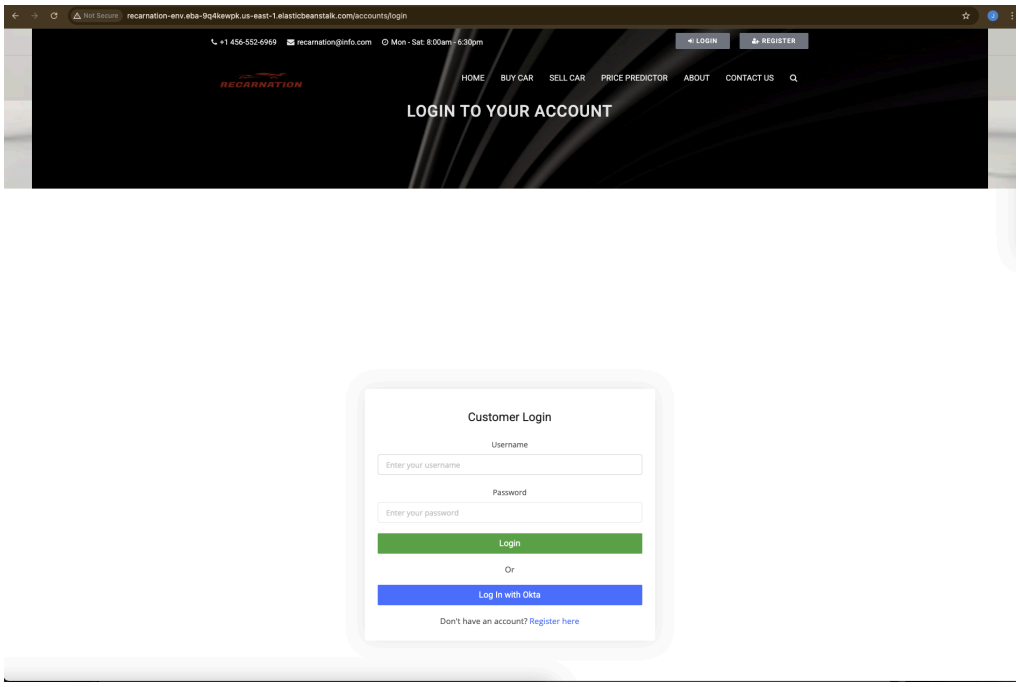

Figure16: Login Page

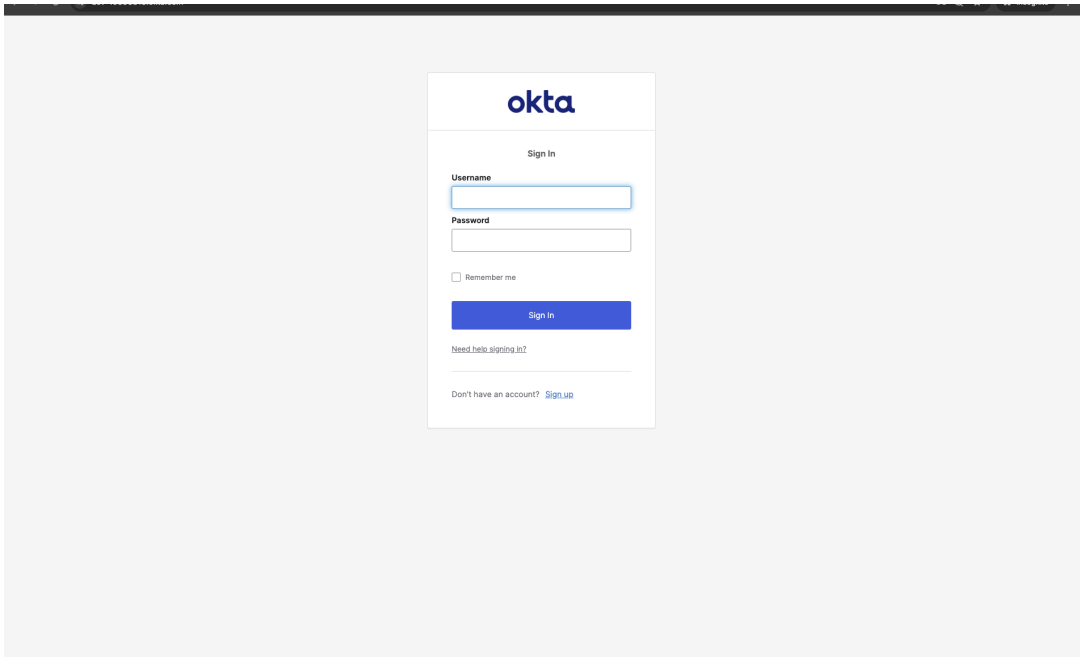

Fig 17: Login with Okta

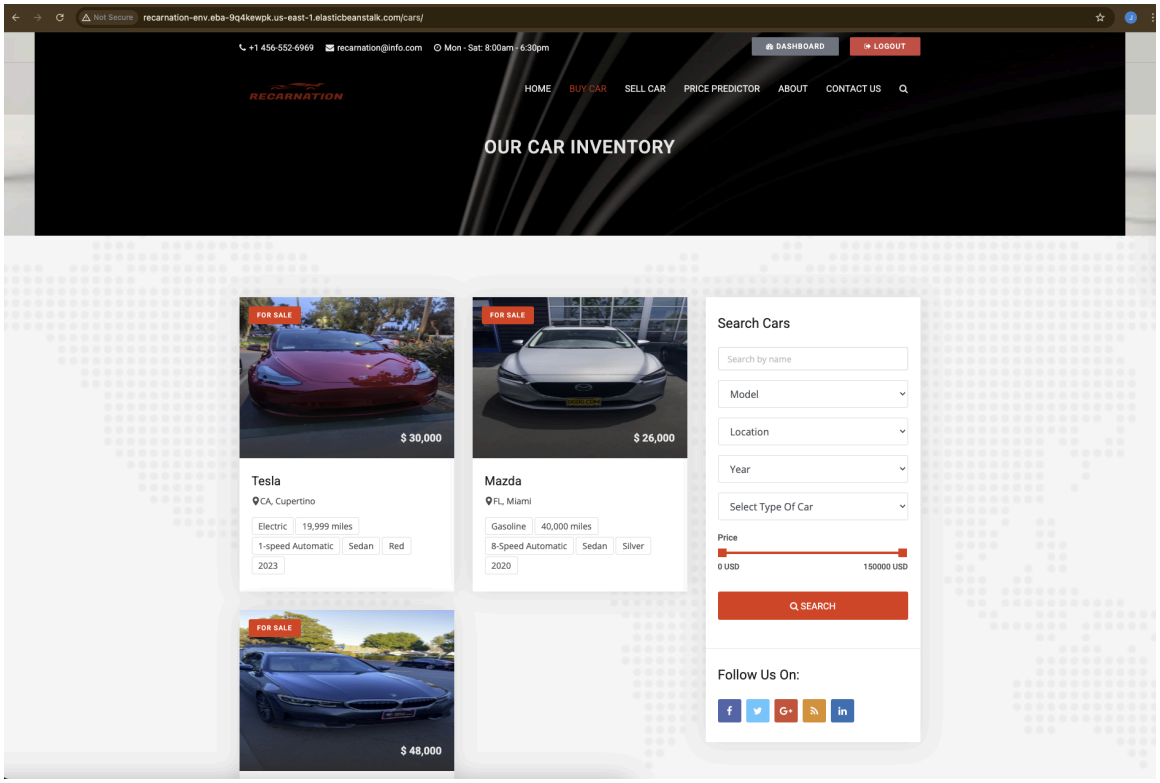

Figure 18: Buy Car Page

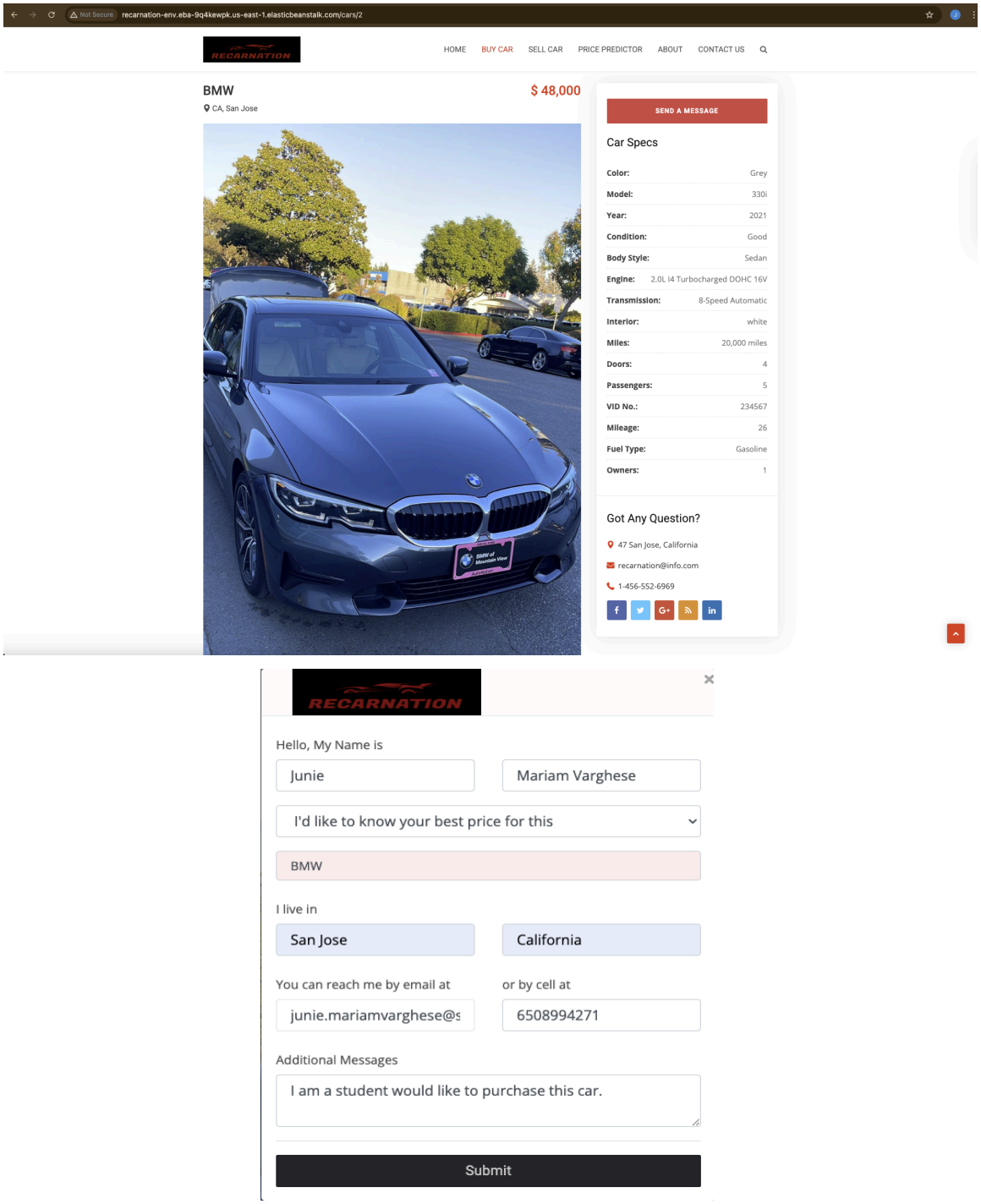

Figure 19: Car Details Page and the Car Inquiry Dialogue Box

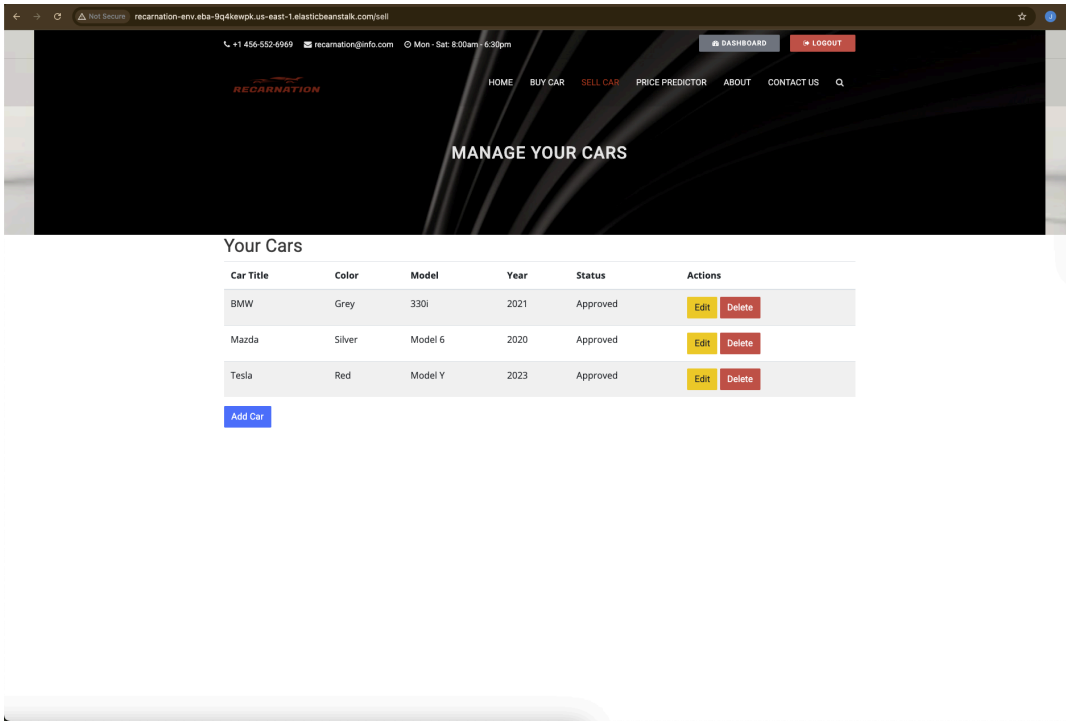

Figure 20: Sell Car Page

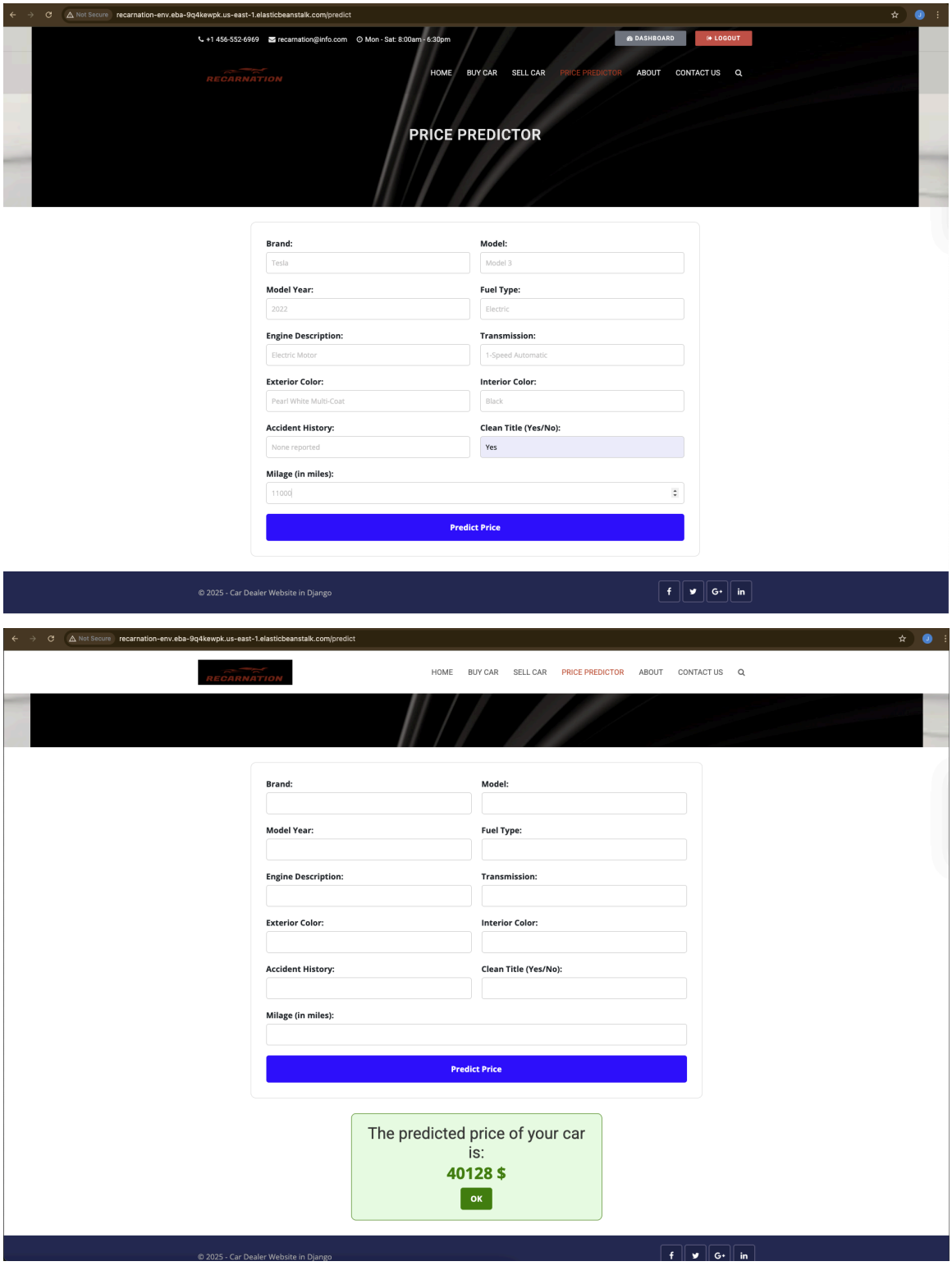

Figure 21: Price Predictor Page

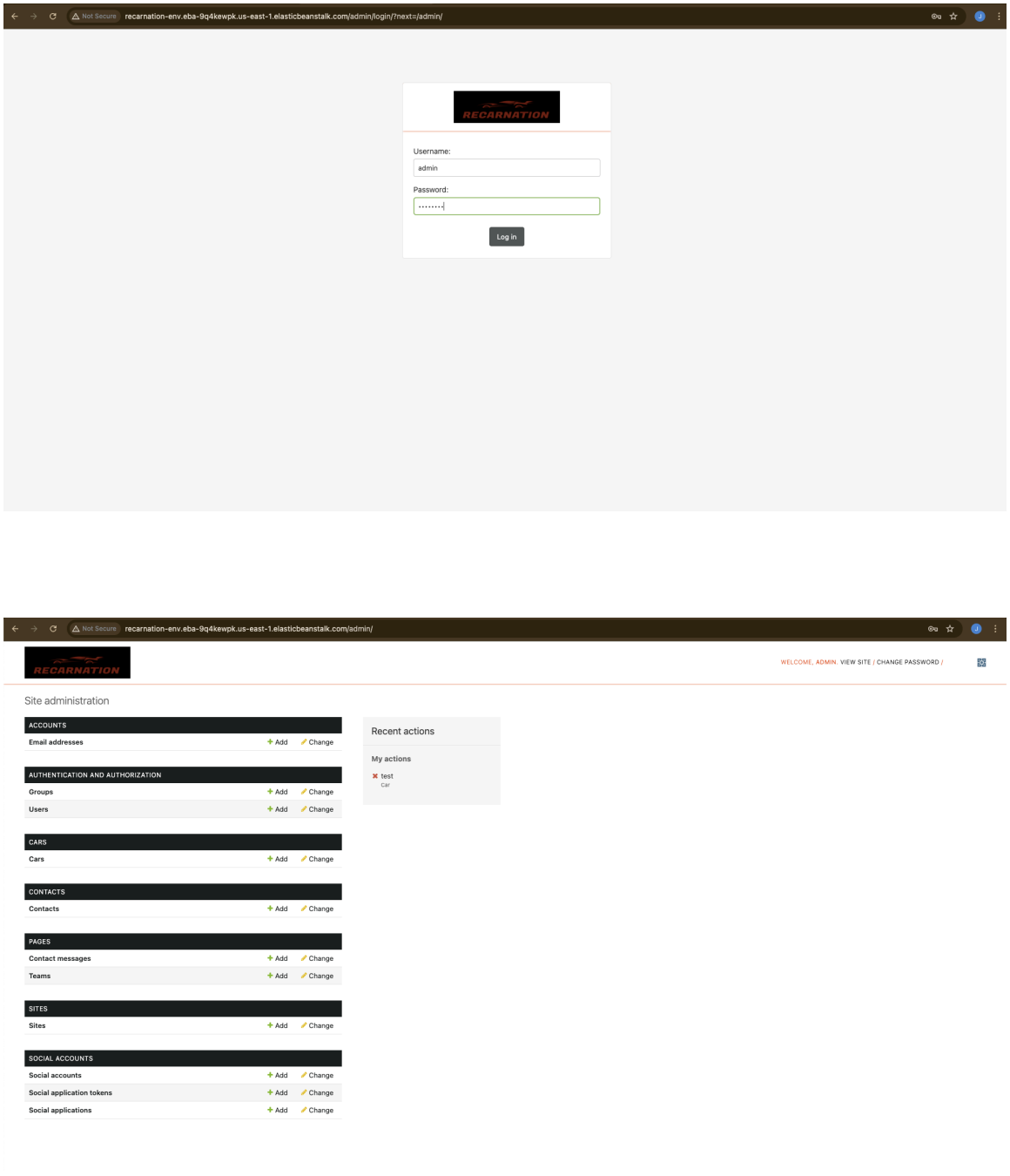

Figure 22: Admin Page

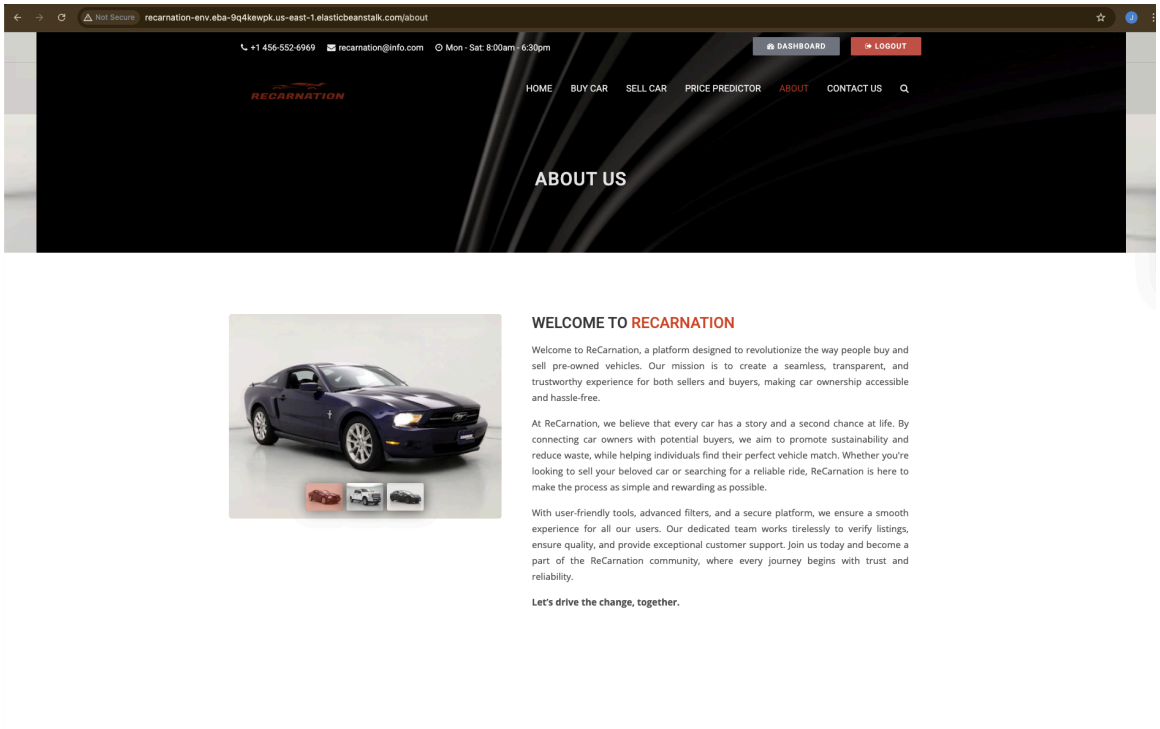

Figure 23: About Page

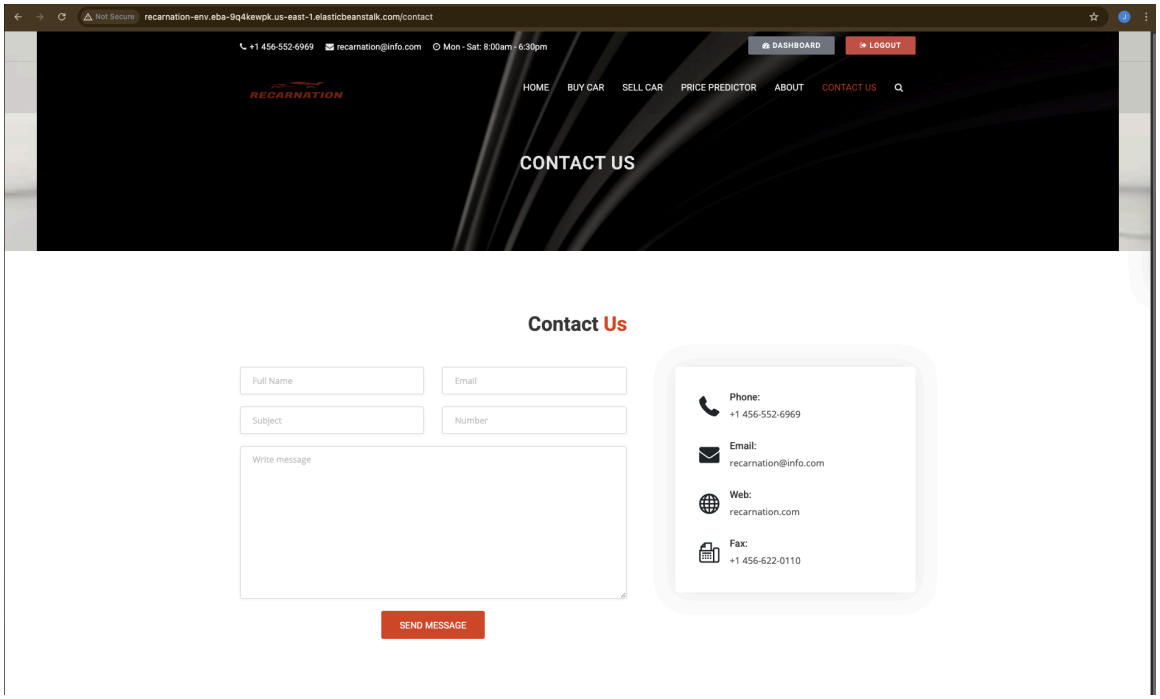

Figure 24: Contact Us Page

4.5 System component API and logic design

This section outlines different software components, their APIs, and the logic design behind them. The design ensures modularity, scalability, and seamless communication between components while maintaining security and efficiency.

Component Overview
The main modules of the platform, with other components interacting through RESTful APIs, take care of the buying, selling, and administrative functions. A summary of the main modules is given below:

- **Authentication Service**
  Handles user authentication using OAuth or classical logins.
  Key Features: User Login, Signup, Token-based Authentication.
- **Vehicle Listing Service**
  Provides creation, search, filtering, and management of listings of vehicles.
  Key Features: Listing creation, advanced searches, and status updates.
- **User Management Service**
  Manages user roles, profiles, and permissions.
  Key Features Role-based access control, profile update.
- **Admin service**
  It allows the administrator to moderate approved and rejected listings, manage users, and monitor the performance of the platform.
  Key Features Listing approvals/rejections and user management.
- **Price Prediction Service**
  It predicts the best price depending on the specification of a car and market trends.
  Key Features Price suggestion to sellers.
- **Notification Service**
  Send alerts/updates to users on listings, approvals, and purchases.
  Key Features Email as well as in-app notifications.

Logical Design
The internal logic is meant for secure and efficient operations:

- **Authentication Logic**
  Validates user credentials and issues tokens.
  OAuth tokens are validated with the provider before allocating internal session tokens

- **Vehicle Listing Logic**
  When sellers create listings, they will be flagged as "Pending Approval" and thus will be kept pending for review by an admin.
  Therefore, the buyers can apply filters and sorting to quickly find their desired vehicles.
- **Admin Moderation Logic**
  The Admins can approve or reject listings based on predefined rules.Thus, these approved listings will be made visible for viewing by buyers, while the rejected ones can be revised by the seller.
- **Price Prediction Logic**
  Inputs include car specifications, condition, and market data. A competitive price using machine learning algorithms is calculated by using the algorithm.

API Design

Each component exposes RESTful APIs, ensuring a standardized approach to communication between the frontend and backend. Below is a description of key API endpoints:

1. Authentication API

Endpoint	Method	Description	Parameters
/accounts/login	POST	Authenticates the user and issues a token	Email, Password
/accounts/register	POST	Registers a new user	User details
/accounts/login	GET	Handles OAuth-based login	Provider, Token

Table 8: Authentication API Table

2. **Vehicle Listing API**

Endpoint	Method	Description	Parameters
/cars	GET	Fetches all active listings	Filters (optional)
/cars/:id	GET	Fetches details of a specific car	Listing ID
/sell	POST	Creates a new vehicle listing	Listing data
/sell/:id	PUT	Updates an existing listing	Listing ID, Updates

Table 9: Vehicle Listing API

3. Admin API

Endpoint	Method	Description	Parameters
/admin/cars/car	GET	Fetches all pending listings	None
/admin/cars/:id	POST	Approves or rejects a listing	Listing ID, Action
/admin/auth/users	GET	Fetches all registered users	None

Table 10: Admin API table

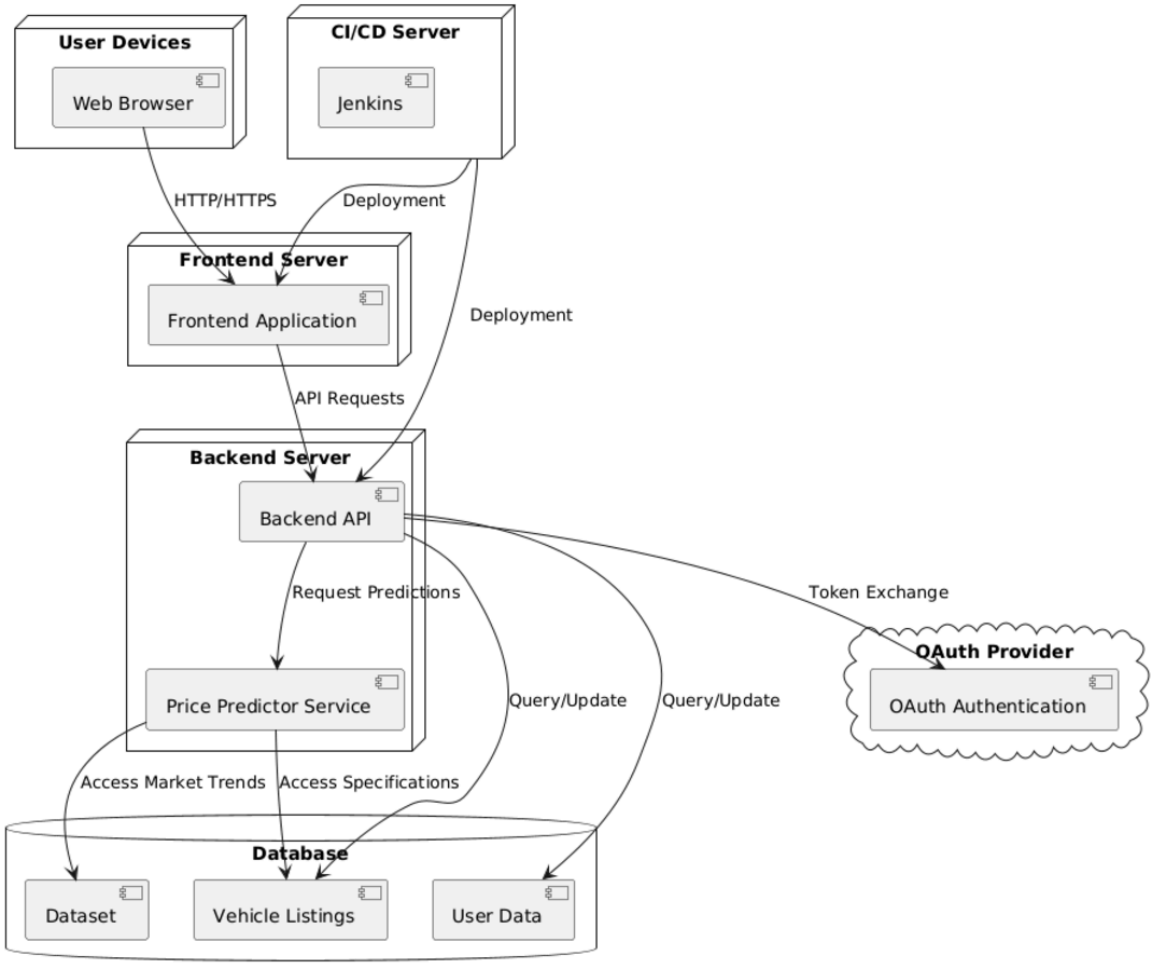

Figure 25: Deployment Architecture of the system

CHAPTER 5. SYSTEM IMPLEMENTATION

5.1. System implementation summary

The implementation of the ReCarNation cloud platform was driven by priorities of security, scalability, and user-centric design. The frontend, built with HTML, CSS, and JavaScript, is hosted on Amazon S3 to ensure responsive access across all device types. The backend, developed using Django and deployed on AWS Elastic Beanstalk, handles business logic, REST APIs, and role-based user management.

Authentication is powered by Okta-based cloud Single Sign-On (SSO) with GitHub OAuth integration, securing both user and admin logins. Buyers benefit from advanced search and filtering, while sellers can easily list vehicles through guided forms. Admins access a robust control panel to manage listings, approve or reject posts, and monitor platform activity.

A standout feature is the machine learning-based price prediction module, integrated into the backend. Using a linear regression model, it analyzes car specs and market trends to recommend optimal listing prices for sellers.

Data storage is managed via Amazon RDS (PostgreSQL), ensuring secure and efficient handling of user data, listings, transactions, and administrative records. Continuous integration and delivery are automated through a Jenkins CI/CD pipeline on EC2, to streamline deployments. Though automated testing is planned for future iterations, all modules underwent thorough manual testing using Postman and browser tools, ensuring platform stability.

5.2. System implementation issues and resolutions

Several challenges were encountered and resolved during development:

- **ML Model Integration**: Integrating the price predictor required a dedicated Django API endpoint. After tuning and validation, the model successfully provided accurate and consistent pricing suggestions aligned with market data.

- **Responsive Frontend Design**: Achieving cross-browser compatibility demanded use of modern CSS frameworks and extensive testing. The result is a smooth, responsive interface across all major platforms.

- **Secure Admin Panel**: Role-based access control was crucial. This was implemented using Django's built-in authentication and permissions system to restrict sensitive operations to verified admins.

- **Database Optimization**: As data volume increased, performance bottlenecks appeared. These were mitigated through SQL query optimization, indexing of key columns, and schema adjustments to improve efficiency.

- **Authentication Integration**: Combining standard credentials with Okta and GitHub SSO involved careful middleware setup and validation, ensuring a seamless login experience for both users and developers.

- **Cloud Readiness**: While the final deployment is upcoming, the application was fully architected for AWS, with modular components and automated deployment scripts ready for production use.

Manual testing ensured stability during development. Tools like Postman and browser debuggers were used extensively, laying the foundation for future enhancements including automated test integration, AI-powered fraud detection, and global feature expansion.

5.3. Used Technologies and Tools

The ReCarNation Cloud platform is built using a modern, cloud-native tech stack that ensures scalability, security, and a smooth user experience. Each technology and tool was selected to fulfill specific requirements in the system's architecture, development, deployment, and performance.

Frontend Development

The frontend is designed using HTML, CSS, JavaScript, and Bootstrap, offering a responsive, cross-platform UI. Static assets are hosted on Amazon S3, ensuring low-latency access and fast rendering on any device. The interface supports interactive car browsing, filtering, and listing, tailored for buyers, sellers, and admins.

Backend Development

The backend is built with the Django framework (Python) and deployed using AWS Elastic Beanstalk, which provides managed infrastructure, automatic scaling, and

simplified deployment. Django handles core server-side logic, RESTful API exposure, user management, listing processing, and ML model integration.

Database Management

Structured data such as user profiles, vehicle listings, and transaction history is managed via Amazon RDS (PostgreSQL). RDS offers managed backups, automatic failover, and secure, scalable relational data storage with high performance.

Machine Learning Integration

The price prediction module is implemented using Python-based ML libraries such as Pandas, NumPy, and Scikit-learn. Hosted alongside the backend, it applies a linear regression model to analyze car attributes and market data, helping sellers estimate optimal pricing.

Authentication and Security

User authentication is managed via Okta-based Single Sign-On (SSO) integrated with GitHub OAuth for developer workflows. IAM roles and Security Groups enforce secure access to AWS services and backend resources. Django's built-in auth system provides fine-grained role-based access control for users and admins.

CI/CD and DevOps

The project uses Jenkins (on EC2) for continuous integration for deployment automation. Jenkins orchestrates environment setup, dependency management, and deployment tasks, while also running safety checks and validation against Elastic Beanstalk environments.

Testing and Debugging

API endpoints are tested using Postman, and the frontend is validated through browser dev tools. These tools assist in functional verification, debugging, and performance tuning across modules.

Version Control

Git is used for source control, and GitHub hosts the repository, enabling seamless team collaboration and code review workflows. Integration with Jenkins ensures automated testing and deployment with each update.

This comprehensive stack ensures a reliable, scalable, and secure platform for the used car marketplace, while also enabling rapid development and continuous delivery.

CHAPTER 6. SYSTEM TESTING AND EXPERIMENT

6.1Testing and experiment scope

In our application, we have implemented a combination of automated functional tests and manual testing to ensure the reliability and correctness of the core features. The tests focus on key features such as user authentication, car management (including approval and rejection processes), and the integrity of the database. Below is a detailed breakdown of the automated test cases implemented, alongside the manual testing process for other parts of the application.

1. User Authentication & Registration Tests

- Login with Valid Credentials: We tested whether a user can log in successfully using valid credentials. A valid login should redirect the user to the dashboard page.
- Login with Invalid Credentials: This test checks if the system correctly handles invalid login attempts, providing appropriate error messages.
- Registration Success: We tested if a user can successfully register with valid data and a matching password confirmation. The registration should redirect the user to the dashboard.
- Password Mismatch: This test verifies that the registration form returns an error when the password and confirmation do not match.
- Duplicate Username: We tested the registration process when a user attempts to register with an already existing username. The form should return an appropriate error message.
- Duplicate Email: This test checks if the system properly handles duplicate email registrations by showing an error message when the email is already associated with another account.
- Authenticated Dashboard View: This test ensures that only authenticated users can access the dashboard. Unauthenticated users should be redirected to the login page.
- Unauthenticated Dashboard Access: We tested that unauthenticated users trying to access the dashboard are redirected to the login page.
- Logout Functionality: We tested the logout view to ensure that authenticated users can log out successfully and are redirected to the homepage.

2. Car Model Tests

- Car Creation: We tested whether a car instance can be created successfully with the expected attributes (e.g., title, city, status, etc.). This ensures that the system handles car data correctly when a new car is added.
- Car String Representation: We verified that the string representation of the Car model returns the car title as expected, making it easier to identify cars in the admin panel or when displayed elsewhere.

3. Approve Cars View Tests

- Approve Cars View (Staff User): We tested whether a staff user can access the car approval view, where they can approve or reject cars. This view is restricted to staff users, and the test verifies that a staff user can view cars pending approval.
- Approve Car Action: We simulated the action of approving a car. This test verifies that when a staff user approves a car, its status is updated from "Pending" to "Approved" in the database.
- Reject Car Action: Similar to the approved action, we tested the process of rejecting a car. The car's status should change from "Pending" to "Rejected" when a staff user selects the 'reject' option.

While the tests above cover the core functionalities of the application, other features were tested manually. These include:

- Car Listing and Search: The functionality of listing all cars and searching for cars by different attributes (e.g., city, make, model, etc.) was manually tested to ensure proper functionality.
- Admin Views and Permissions: Additional admin-specific workflows, such as managing car listings, editing car information, and reviewing submitted inquiries, were manually tested to verify that the correct permissions are enforced.
- User Interactions: Various user interactions, such as editing profiles, updating contact information, and submitting inquiries, were manually tested to ensure smooth and error-free user experiences.

6.2 Testing and experiment approaches

The primary goal of this test plan is to ensure the robustness, security, usability, and performance of the application. This includes testing critical features such as user authentication, car management, and approval/rejection workflows. To achieve this, we have devised a comprehensive strategy that involves a combination of unit testing,

functional testing, integration testing, and manual testing. Each of these testing methods addresses different aspects of the system, ensuring that all key features and components are thoroughly validated before deployment.

1. Test Methods

a. Unit Testing

Unit testing is an essential part of the test plan, aimed at validating individual components of the application. It ensures that each part of the system—such as models, views, and form validations—functions correctly in isolation. Unit tests are automated and run frequently during the development process to detect errors early. These tests are primarily written for smaller units of functionality, for instance:

- Model Tests: Ensuring that data objects, such as car and user models, are created and saved correctly in the database.
- View Tests: Verifying that views return the expected status codes and templates based on different inputs.

The unit tests are already implemented and tested, as seen in the CarModelTest and ApproveCarsViewTest classes. These tests cover key scenarios, such as car creation and approval/rejection workflows.

b. Functional Testing

Functional testing focuses on verifying the core user workflows of the application, ensuring that each flow behaves as expected from the user's perspective. This includes testing user authentication (login and registration), car creation, and managing the approval or rejection of cars.

- Login and Registration Testing: Ensuring that users can register successfully and log in with correct credentials while being appropriately redirected.
- Car Management Testing: Testing the process of creating and editing cars, as well as verifying that users with the appropriate permissions can approve or reject cars.

Functional testing helps validate the user interface and overall behavior of the system when interacting with it through a browser or API.

c. Integration Testing

Integration testing ensures that different components of the application interact as expected. This involves testing the interactions between the front end and the back end, ensuring that the system behaves correctly when components are combined. For example:

- Car Approval Workflow: Testing the approval action triggers both the correct status change in the database and appropriate notifications to the user.
- User Authentication and Permissions: Ensuring that actions like car approval or rejection are only accessible to users with the correct permissions (e.g., staff members).

Integration tests help identify any issues that arise when multiple components are combined, ensuring smooth operation when the system is in use.

d. Manual Testing

Manual testing plays a critical role in identifying issues that may be difficult to capture through automated tests. This testing is primarily focused on:

- UI/UX Testing: Manually verifying the layout, responsiveness, and user-friendliness of the user interface.
- Exploratory Testing: Actively exploring the application to uncover edge cases, usability issues, or other unforeseen problems that might not be covered by automated test cases.

While unit, functional, and integration tests cover most of the application's functionality, manual testing is essential for assessing the user experience, especially with respect to UI design and permission handling.

2. Test Criteria

The success of the testing process is based on several key criteria, which are used to evaluate the system's performance and behavior. These criteria include:

a. Correctness

Correctness refers to the application's ability to meet functional requirements and deliver the expected results. For example:

- Verifying that a car is correctly created and saved in the database with the right details.
- Ensuring that when a car is approved, its status is correctly updated to "Approved".

Automated tests already cover many of these correctness checks, but additional manual tests will be written to check for edge cases and unexpected inputs.

b. Usability

Usability ensures that the application is user-friendly and intuitive. The registration, login, car creation, and approval processes should all be straightforward and easy to navigate. Testing will focus on the following:

- Navigation: Ensuring that users can easily navigate through pages and understand the next steps.
- Error Messages: Ensuring that informative and helpful error messages are displayed when users make mistakes.

c. Security

Security testing ensures that sensitive parts of the system are properly secured and that unauthorized users cannot perform sensitive actions like approving or rejecting cars. Key areas for security testing include:

- Authentication: Ensuring that only registered and authorized users can access their accounts.
- Permissions: Verifying that only staff members can approve or reject cars.
- Data Protection: Ensuring that sensitive information, such as passwords, is stored securely and is not accessible to unauthorized users.

d. Performance

Performance testing evaluates how well the application performs under various loads. The system should be able to handle multiple concurrent users and operate efficiently without delays. Testing will focus on:

- Response Times: Ensuring that page load times remain reasonable under different loads.
- Concurrent Users: Simulating multiple users performing tasks like login or car approval and evaluating the system's response.

e. Edge Case Handling

Edge cases involve scenarios that are rare or extreme, such as inputting invalid data or performing actions in an unusual order. The system should gracefully handle these situations and provide appropriate feedback to the user. Tests will focus on:

- Boundary Conditions: Verifying that the system correctly handles large or small inputs.
- Invalid Input: Ensuring that users cannot submit incomplete or incorrect forms.

3. Test Coverage

Test coverage includes both automated and manual tests, ensuring that all critical functionality is tested and validated.

A test coverage chart will be provided to highlight the percentage of coverage for each category (unit tests, functional tests, integration tests, and manual tests). The chart will show:

- Automated Tests: Which portions of the system are covered by unit, functional, and integration tests.
- Manual Tests: Areas like UI/UX and exploratory tests that are not covered by automation.

4. Future Testing Plans

While unit tests are already written and executed, future plans include the development of the following test cases:

- Functional Tests: Writing more detailed functional tests to cover different user flows, such as car approval/rejection, and testing various edge cases related to data input.
- Integration Tests: Testing complex workflows, like the interaction between user authentication, car management, and approval actions, to ensure that all system components interact as expected.
- Performance Testing: Conducting load testing to assess the application's responsiveness and performance under heavy traffic.

Manual testing will continue to focus on usability, user interface, and exploratory testing to ensure a seamless user experience.

The test plan is designed to ensure that the application is functional, secure, usable, and performs well under various conditions. It uses a combination of automated tests and manual testing to comprehensively cover all critical features, including user authentication, car management, and approval workflows. As the project progresses, additional test cases for performance, security, and edge cases will be written to provide further coverage and ensure the system is ready for deployment.

6.3 Testing and experiment

The test scripts developed for this project are designed to cover all critical system functionality, including user authentication, car management, approval/rejection workflows, and security validation. The following test scripts were executed:

Test Category	Test Script	Purpose	Scope	Outcome
Unit Testing	CarModelTest	To test that the car model correctly handles car creation and updates	Creating cars with valid inputs, validating required fields, and updating car status.	All tests passed, with correct car creation and status updates.
Unit Testing	ApproveCarsViewTest	To verify the car approval functionality.	Ensuring only authorized users (admin) can approve cars and update car status.	All tests passed, with successful status updates and restricted access for unauthorized users.
Functional Testing	UserLoginTest	To validate the login functionality for registered users.	Tests valid and invalid login attempts, ensuring proper error messages are displayed.	All scenarios passed, error messages displayed correctly, and successful login leads to the correct dashboard.
Functional Testing	CarCreationTest	To validate that the car creation process works smoothly.	Tests form submission for car details and ensures the car is added to the database	Passed, with the car appearing in the admin's dashboard upon

				successful creation.
Integration Testing	UserAuthenticationAndCarApprovalTest	To test that a user can login and approve/reject cars.	Testing the integration of login functionality with car approval actions.	Integration passed with no errors, users with appropriate roles can approve/reject cars.
Integration Testing	CarApprovalAndNotificationTest	To validate that a car's approval triggers the correct status update and notification.	Verifying that the approval workflow updates car status and sends notifications.	All steps passed successfully, with accurate status updates and notifications sent.
Manual Testing		To cover edge cases, UI/UX, and unusual user flows.	Testing different user roles, input data variations, and UI responsiveness.	Majority of manual test scenarios passed, though UI responsiveness issues identified for further refinement.

Table 11:System Test Scripts Summary and Outcomes

Figure 26 :Test Run Output for Recarnation Web Application

Overall, the system's testing phase revealed several critical issues, most of which were promptly addressed. The test coverage was extensive, including unit, functional, integration, and manual testing, ensuring that all critical features function as expected. The bug report analysis highlights a few remaining moderate issues, which are being actively worked on. The experimental case study showed the core workflows functioning as intended, with some minor UI and performance adjustments still required. Future testing will focus on edge cases, further performance optimization, and additional validation on user permissions.

CHAPTER 7. CONCLUSION AND FUTURE WORK

7.1 Project summary

The project successfully created and deployed a secure web application on AWS designed to modernize the buying and selling of pre-owned vehicles. The platform features a clean, intuitive interface that caters to both buyers and sellers. Buyers benefit from an advanced search and filtering system that allows them to browse vehicles based on personal preferences, while sellers can easily list their cars with detailed specifications.

Key features include a price prediction tool, administrative controls, and secure user authentication, all of which enhance the platform's reliability and user experience. The backend is supported by a robust database hosted on AWS services to securely manage car listings, user data, and transaction history. The project also integrates continuous integration and automated testing via Jenkins, ensuring consistent updates and system reliability.

Overall, the platform addresses the rising demand for affordable transportation by offering a transparent and secure solution for the pre-owned vehicle market.

7.2 Future work

Future development will focus on personalization, trust, globalization, and user engagement. With the platform now successfully deployed on AWS Elastic Beanstalk, key infrastructure concerns like scalability and load balancing are efficiently managed, allowing the team to focus on new features and user experience.

AI-Based Recommendations: The platform will integrate AI algorithms to provide personalized car suggestions based on user behavior, search history, and preferences—making it easier for buyers to find vehicles that meet their specific needs.

Fraud Detection: To ensure user safety, AI-powered fraud detection systems will be introduced. These will analyze patterns in listings and user activity to detect and flag suspicious behavior, helping prevent scams and maintaining a trustworthy marketplace.

Enhanced Trust Mechanisms: Building on the current verification processes, future updates will include features such as vehicle history reports, third-party inspections, and seller ratings, offering users greater confidence in their transactions.

Mobile Accessibility: A mobile-friendly version of the platform will be developed to provide on-the-go access for browsing, managing listings, and communicating with buyers or sellers.

Globalization: To support international users, the platform will add multi-language and multi-currency support, making it easier for people from different regions to use the application in their native language and conduct transactions in their local currency.

References

1. Doe, J., & Smith, R. (2020). Consumer Preferences in E-Commerce Automotive Platforms. *Journal of E-Commerce Research*, 15(3), 145-160.
2. Lee, M., & Tan, P. (2019). Building Trust in Online Car Dealerships. *International Journal of Consumer Studies*, 43(2), 200-215.
3. Chen, Y., & Kumar, A. (2021). Scalable Architectures for E-Commerce Platforms. *ACM Transactions on Web*, 15(4), 1-20.
4. Patel, S., & Jones, H. (2022). The Role of Artificial Intelligence in Personalizing User Experiences. *Artificial Intelligence Review*, 56(5), 450-470.
5. Brown, K., & Li, J. (2020). Next-Generation Car Marketplace: Blockchain and Transparency. MIT Media Lab Reports.
6. Wong, A., & Zhao, F. (2018). Enhancing Usability in Automotive E-Commerce. *Stanford HCI Group Research Papers*.
7. Taylor, D., & Singh, R. (2021). GreenCar Initiative: Promoting Sustainability in Online Automotive Sales. *University of Cambridge Research Journal*.
8. Shutterstock. (2024). *Black matte sports car with grunge overlay*. Shutterstock. https://www.shutterstock.com/image-illustration/black-matte-sports-car-grunge-overlay-529860217
9. hdcarwallpapers.com. *Jaguar F-Type R 4K*. HD Car Wallpapers. https://www.hdcarwallpapers.com/walls/jaguar_f_type_r_4k_2-HD.jpg.
10. All the listed car images used in this application were taken by the authors for the purpose of this project.
11. All the figures and diagrams were created using Mermaid Diagramming Tool, PlantUML and draw.io Diagramming Software.