Declaration of Authorship

We, the undersigned, hereby declare that the project titled *"Digital Car-Yard"* is our original work and has been completed in accordance with the academic standards of SZABIST University under the supervision of *Khawaja Mohiuddin*.

We confirm that all sources and references used during the research and development of this project have been properly cited and acknowledged. The content of this report is original and has not been copied or plagiarized from any external source.

We further declare that this project has not been previously submitted for the award of any degree, diploma, or other qualification at any other university or institution.

Name: ___Hussam Wasti_____

Signature: _____

Date: _____June 24th, 2025___

Name: _____Roha Ali_____

Signature: _____

Date: ____June 24th, 2025_____

Project Description

Digital Car-Yard is a comprehensive web-based application developed to streamline the car buying, selling, and maintenance experience for users in Karachi, Pakistan. The platform addresses common challenges faced by car owners, such as tracking maintenance schedules, accessing reliable service providers, and finding a trusted marketplace for cars and car parts. Through a subscription-based model, users receive timely reminders for essential maintenance tasks like oil changes and inspections, along with access to free services. The integrated marketplace allows for buying and selling of car parts and accessories. Additionally, the app features a chatbot for real-time assistance and a reward system that encourages active vehicle management by offering redeemable points. With a clean and responsive user interface built using modern web technologies, Digital Car-Yard consolidates all aspects of vehicle ownership into a single, convenient platform. The project aims to deliver a smarter, more reliable solution tailored to the needs of Karachi's growing automotive community.

Acknowledgement

In the name of Allah, the most beneficent and most merciful, who gave us the knowledge and courage to work on this project

We are grateful for the outcome and success of this project over the year. Our gratitude towards the people who have provided us with the guidance and assistance to be able to complete this project in such a difficult time.

We would like to thank our supervisor “Khawaja Mohiuddin” of the computer science faculty at Shaheed Zulfiqar Ali Bhutto Institute of Science and Technology. He was an integral part of the project as he was always there when we would get stuck at a point in a project or whenever we required any sort of suggestion. He constantly guided us, motivated us and cooperated with us throughout the duration of this project.

We would like to thank the teachers at Shaheed Zulfiqar Ali Bhutto Institute of Science and Technology, who guided us and taught us throughout our time in the university. We would also like to express our gratitude to our parents and family members who helped and encouraged us during this time. Furthermore, we would to thank the staff at Szabist for allowing us to use their labs and services to be able to complete the project.

Lastly, we would like to extend our gratitude to everyone at Shaheed Zulfiqar Ali Bhutto Institute of Science and Technology for creating an environment for students to thrive in. The quality of education, the cooperative faculty members and the motivation provided by them.

Plagiarism-Free Certificate

This is to certify that the project titled **"Digital Car-Yard"** has been successfully completed and submitted by the undersigned students. The project was carried out under the supervision of **Khawaja Mohiuddin** at **SZABIST University**

We hereby affirm that the content of this report is entirely original and has not been copied or reproduced from any external source. All references and materials used in the research and development of this project have been properly cited and acknowledged.

This project fully complies with the academic and ethical standards of **SZABIST University.** Furthermore, we declare that this work is our own and has not been submitted elsewhere for the award of any degree or qualification.

Name: Hussam Wasti

Registration #: 2112113

Signature: _____

Name: Roha Ali

Registration #: 2112124

Signature: _____

Supervisor: Khawaja Mohiuddin

Signature: _____

18% detected as AI

Thepercentageindicatesthecombinedamountoflikely AI-generated text as wellas likelyAI-generated text that was also likely AI-paraphrased.

Caution: Review required.
It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Detection Groups

206AI-generated only 18% Likely AI-generated text from a large-language model.

0AI-generated text that was AI-paraphrased0%
Likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

Disclaimer
OurAI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (it may misidentify writing that is likely AI generated as AI generated and AI paraphrased or likely AI generated and AI paraphrased writing as only AI generated) so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

Frequently Asked Questions

Howshould I interpretTurnitin'sAIwritingpercentageandfalsepositives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

Whatdoes'qualifyingtext'mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.

Turnitin Originality Report

Processed on: 30-Jun-2025 10:18 PKT
ID: 2696181842
Word Count: 16907
Submitted: 5

FINAL DOCUMENT (AutoRecovered).docx By Hafeez Abbasi

Similarity Index	Similarity by Source	
13%	Internet Sources:	8%
	Publications:	2%
	Student Papers:	11%

2% match (student papers from 14-Jun-2024)
Submitted to Harrisburg University of Science and Technology on 2024-06-14

1% match (Internet from 19-Feb-2025)
https://WWW.coursehero.com/file/232524951/fyp-SDS-COMPLETE-1pdf/

1% match (Internet from 06-Feb-2025)
https://www.coursehero.com/file/222104829/SRS-Earn-in-Dollarspdf/

1% match (student papers from 08-Jun-2023)
Submitted to Higher Education Commission Pakistan on 2023-06-08

< 1% match (Internet from 10-Dec-2024)
https://www.coursehero.com/file/p7ecoi0v/Stimulus-The-user-enters-the-credentials-Full-names-payment-and-shipping-details/

< 1% match (Internet from 25-Nov-2021)
https://www.coursehero.com/file/100209130/Akshar-061-SE-Final-1-converted-1docx/

< 1% match (Internet from 02-Apr-2025)
https://www.coursehero.com/file/72526617/NGO-PROM-Final-Reportdocx/

< 1% match (Internet from 30-Mar-2025)
https://www.coursehero.com/file/71636101/hacktivistpdf/

< 1% match (Internet from 21-Feb-2025)
https://WWW.coursehero.com/file/238757413/FYP-Final-Reportpdf/

Contents

TABLES

FIGURES

Proposal

1. Introduction

The **Digital Car-Yard** is a web-based application designed to simplify the car buying, selling, and maintenance processes for users in Karachi, Pakistan. It offers a subscription-based model, providing users with key features like car maintenance scheduling and reminders, and a marketplace for car parts. Users can also access a network of trusted showrooms and maintenance services. A chatbot and a reward system further enhance user experience by offering personalized assistance and points redeemable for in-app purchases. The platform aims to create a comprehensive solution for car owners, enhancing convenience and efficiency in managing car-related tasks.

2. Objective

The objective of the **Digital Car-Yard** is to create a comprehensive web-based platform that streamlines the car ownership experience by providing a convenient solution for buying, selling, and maintaining vehicles. The platform aims to offer features such as subscription-based car maintenance, and a marketplace for car parts. By integrating trusted services, personalized recommendations, and a reward system, the project seeks to enhance user convenience, efficiency, and satisfaction in managing all aspects of car ownership.

3. Problem Description

The Digital Car-Yard addresses several challenges faced by car owners in Karachi, Pakistan.

What: The project aims to provide a centralized platform for car buying, selling, and maintenance. Many cars' owners struggle with keeping track of maintenance schedules, finding trusted service providers, and navigating the complexities of car transactions. Existing solutions often lack integration, forcing users to rely on multiple platforms for different needs.

Why: The rapid growth of car ownership in urban areas has created a demand for more efficient management tools. Car owners frequently encounter issues such as missed maintenance, unreliable service providers, and difficulty in buying or selling vehicles. These challenges not only lead to increased costs but also affect the overall ownership experience. By consolidating these services into one platform, Digital Car-Yard seeks to enhance user convenience, reliability, and satisfaction.

How: The platform will utilize a subscription-based model to offer features such as maintenance scheduling and reminders, and a marketplace for car parts. Additionally, a network of trusted showrooms and maintenance services will be integrated to ensure quality and reliability. By leveraging modern web development technologies, the application will provide a user-friendly interface and seamless interactions, making it easier for car owners to manage all aspects of their vehicles efficiently.

4. Target Industry

The target industry for your **Digital Car-Yard** project primarily falls within the **automotive and technology sectors**. Here's a breakdown of the relevant industries:

- **Automotive Industry**:
  Car Dealerships: The platform will facilitate transactions between buyers and sellers, connecting users with trusted showrooms.
  Maintenance and Repair Services: By integrating a network of service providers, the app supports users in managing vehicle maintenance efficiently.
- **Technology Industry**:
  Web Development: Utilizing modern web technologies, the project represents a digital solution to traditional car ownership challenges.
  E-commerce: The marketplace feature enables online transactions for car parts and rentals, tapping into the growing trend of e-commerce.
- **Subscription Services**:
  The subscription-based model aligns with the increasing demand for services that offer convenience and flexibility, catering to users' maintenance needs.

5. Methodology

To tackle the challenges faced by car owners in Karachi through the Digital Car-Yard project, the following approach will be implemented:

- **Technology Stack**:
  Frontend: Utilize **Flutter** for a responsive user interface. The **Flutter** library will be used to enhance user experience with dynamic content rendering.
  Backend: Implement **NodeJS** for server-side development in JavaScript, allowing for scalable and modular architecture. **Java** will be employed for specific business logic handling.
- **Database Management**:
  Use **MongoDB-Atlas** to store user data, car listings, maintenance records, and transaction histories. This relational database will facilitate efficient data querying and management.
- **API Development**:
  Develop **RESTful APIs** to enable seamless communication between the frontend and backend. These APIs will manage user authentication, car listings, maintenance scheduling, and rental transactions.
- **Data Validation**:
  Implement **Joi** for data validation on the backend, ensuring that all inputs meet specified criteria to maintain data integrity.
- **Algorithm for Notifications**:
  Utilize a **cron job** algorithm to schedule and send notifications for maintenance reminders and refueling updates. This will ensure users receive timely alerts.
- **Third-Party Libraries**:
  Use libraries like **Axios** for making HTTP requests to APIs and **Socket.io** for real-time notifications, particularly for user interactions and chat functionalities with the AI chatbot.

- **Testing Techniques**:
  Employ **Jest** for unit testing of individual components and **Postman** for API testing to ensure all functionalities work as intended before deployment.
- **Deployment and Monitoring**:
  Host the application on platforms like **Heroku** or **AWS**, utilizing tools like **Google Analytics** for monitoring user engagement and performance.

This structured approach ensures that all aspects of the project are addressed efficiently, ultimately leading to a robust and user-friendly application.

6. Project Scope

The Digital Car-Yard project aims to provide a comprehensive web-based platform designed to streamline the car buying, selling, and maintenance experience for users in Karachi, Pakistan. The scope of the project includes the following key aspects:

- **Core Features**:
  Car Buying and Selling: Users can browse, list, and purchase vehicles through a user-friendly marketplace.
  Subscription-Based Car Maintenance: Users receive scheduled reminders for maintenance tasks, along with access to free services (e.g., oil changes).
  Car Parts Marketplace: Users can buy and sell car parts and modifications within the platform.
  Trusted Showrooms and Services: Access to a network of verified showrooms and maintenance services to ensure reliability.
  Chatbot: A chatbot to assist users with inquiries and provide support.
  Reward System: Users earn points for managing their vehicles actively, which can be redeemed for app features or discounts.
- **Target Audience:**
  Car owners, buyers, and sellers in Karachi who seek an efficient and reliable platform for managing their vehicles.
  Technological Implementation:
  Development will utilize modern web technologies, including Flutter, NodeJS, MongoDB-Atlas, MongoDB Database to create a scalable and responsive application.
- **Limitations:**
  The project is focused on the Karachi market initially, with plans for potential expansion based on user feedback and demand.
  Firebase will not be used for backend services, relying instead on custom-built APIs with NodeJS and MongoDB.
- **Future Enhancements:**
  Potential integration of advanced features such as user reviews, rating systems for services, and enhanced analytics for user engagement.

This project scope outlines the key objectives and limitations of the **Digital Car-Yard** project, ensuring clarity on the intended outcomes and functionalities.

7. Feasibility Study

With the defined scope of the **Digital Car-Yard** project, meeting the project schedule is feasible, given effective risk management and resource allocation.

- **Risks Involved:**

  Technical Integration: Difficulties may arise when integrating frontend and backend components.
  Mitigation: Use well-defined API contracts and conduct regular integration tests.

  Scope Creep: New feature requests can lead to delays.
  Mitigation: Implement a strict change management process to evaluate impacts.

  Resource Availability: Key team members may become unavailable.
  Mitigation: Maintain a flexible team structure and ensure thorough documentation.

- **Resource Requirements:**

  Human Resources: Frontend and backend developers, UI/UX designers, and QA testers are essential.

  Computing Resources: Development laptops, hosting services (like AWS), and database management tools (e.g., MySQL Workbench).

  Software Tools: Git for version control, Trello or Jira for project management, and Figma for design prototypes.

8. Solution Application Areas

The Digital Car-Yard project holds significant value for the automotive sector, particularly for car owners in Karachi, Pakistan. Here's how the target domain can benefit from the proposed solution:

- **Streamlined Processes**: The platform simplifies the car buying, selling, and maintenance processes, reducing the time and effort required by users. This enhances overall user experience and satisfaction.
- **Convenience and Accessibility**: Users gain access to a centralized hub for all their car-related needs, maintenance scheduling, and a marketplace for car parts. This eliminates the need to navigate multiple platforms.
- **Cost Efficiency**: By offering subscription-based maintenance services, users can save on regular upkeep costs and access free services, such as oil changes.
- **Trust and Reliability**: The integration of a network of trusted showrooms and maintenance services builds user confidence in transactions and service quality.
- **Data-Driven Insights**: The platform can collect valuable data on user preferences and behavior, enabling further enhancements and personalized experiences in the future.

Overall, the **Digital Car-Yard** project provides real value by addressing the main points of car ownership, ultimately leading to a more efficient, reliable, and enjoyable experience for users.

9. Tools/Technology

- Frontend Development:
  Flutter
- Backend Development:
  NodeJS
- Java
  MongoDB-Atlas
- Database:
  MongoDB
- API Development:
  RESTful APIs
- Testing:
  Jest
  Postman
- Project Management:
  Trello or Jira
- Design Tools:
  Figma or Adobe XD
- Hosting and Deployment:
  Heroku or AWS

10. Expertise of the Team Members

- **Frontend Developer**: Proficient in MongoDB-Atlas and Flutter for dynamic UI development.
- **Backend Developer**: Skilled in NodeJS and Java for server-side development and API integration.
- **Database Administrator**: Experienced in MongoDb for database design and management.
- **UI/UX Designer**: Proficient in Figma or Adobe XD, focusing on user-friendly interface design.
- **QA Tester**: Knowledgeable in testing methodologies and experienced with Jest and Postman for testing.
- **Project Manager**: Strong organizational skills with experience in project management tools like Trello or Jira.

11. Milestones

- **Project Initiation**
  Define project scope and objectives
  Conduct market research and user surveys
  Identify target audience and key stakeholders
- **Planning and Design (Months 1-3)**
  Develop the system architecture
  Create wireframes and UI/UX prototypes

Design the database schema
Develop a project management plan and timeline

- **Frontend Development (Months 4-6)**
  Implement UI using Flutter and MongoDb-Atlas
  Integrate NodeJS for dynamic content rendering
  Develop core features for user registration and car listings
- **Backend Development (Months 7-9)**
  Implement MongoDb-Atlas/NodeJS for API development
  Create RESTful APIs for user management and car transactions
  Integrate MongoDb for data storage and management
- **Testing and Integration (Months 10-11)**
  Conduct unit testing and integration testing
  Perform user acceptance testing with target users
  Implement feedback from testing to refine features
- **Deployment and Launch (Month 12)**
  Deploy the application on a cloud platform (e.g., Heroku, AWS)
  Conduct final testing and quality assurance
  Launch the application and gather user feedback

- **FYP-I Features:**
  User profile management
  Admin Panel
  Car listings and search functionality
  Chatbot (human to human) for user assistance
  Basic maintenance reminder system (oil changes, tire rotations, vehicle inspections)
- **FYP-II Features:**
  Marketplace for car parts: Product Listing, Shopping Cart, Payment Gateway, Chat between user and shopkeeper

12.Project Schedule

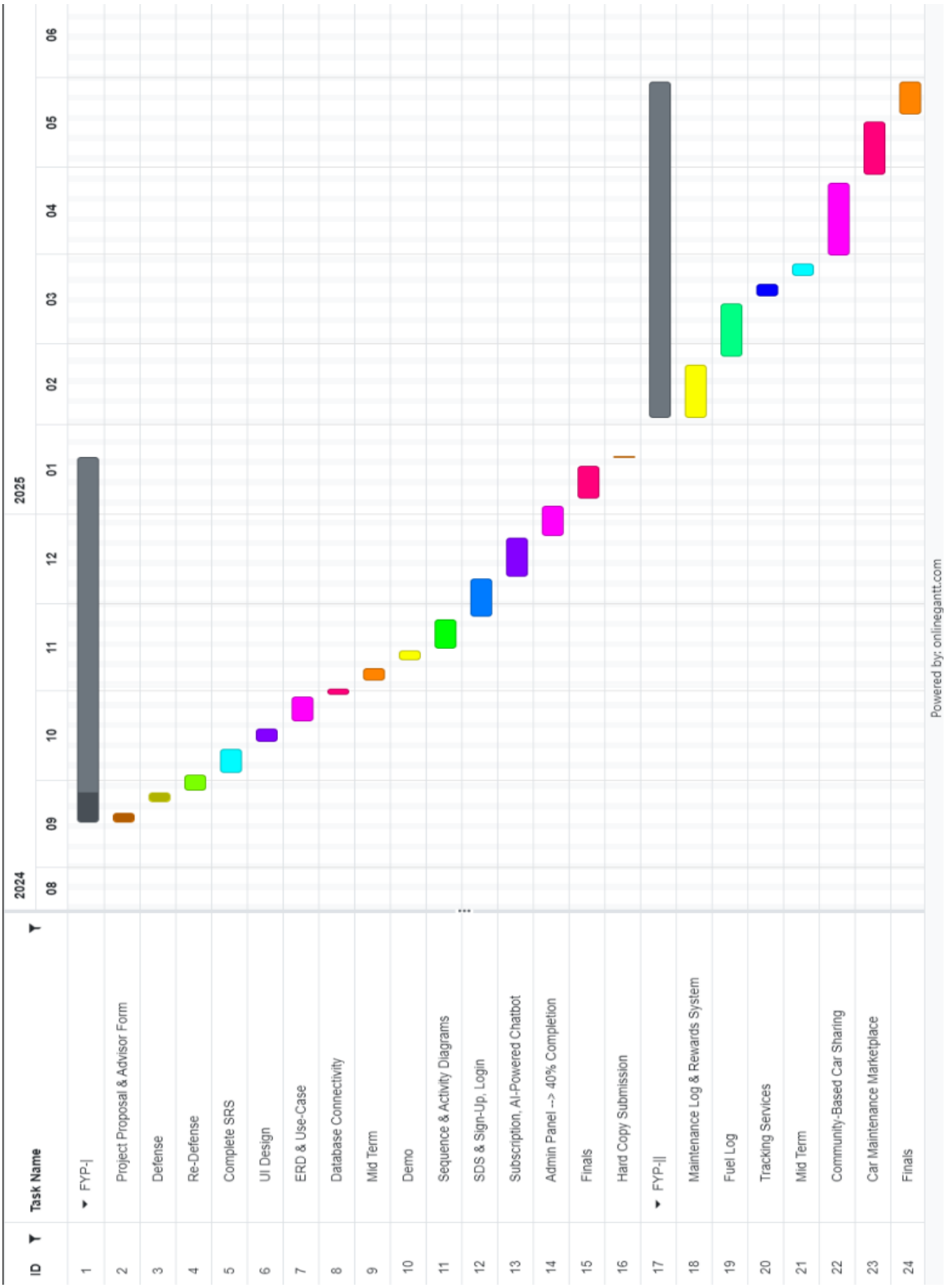

Figure 1 Project Schedule

13.Work Breakdown Structure:

FYP-I

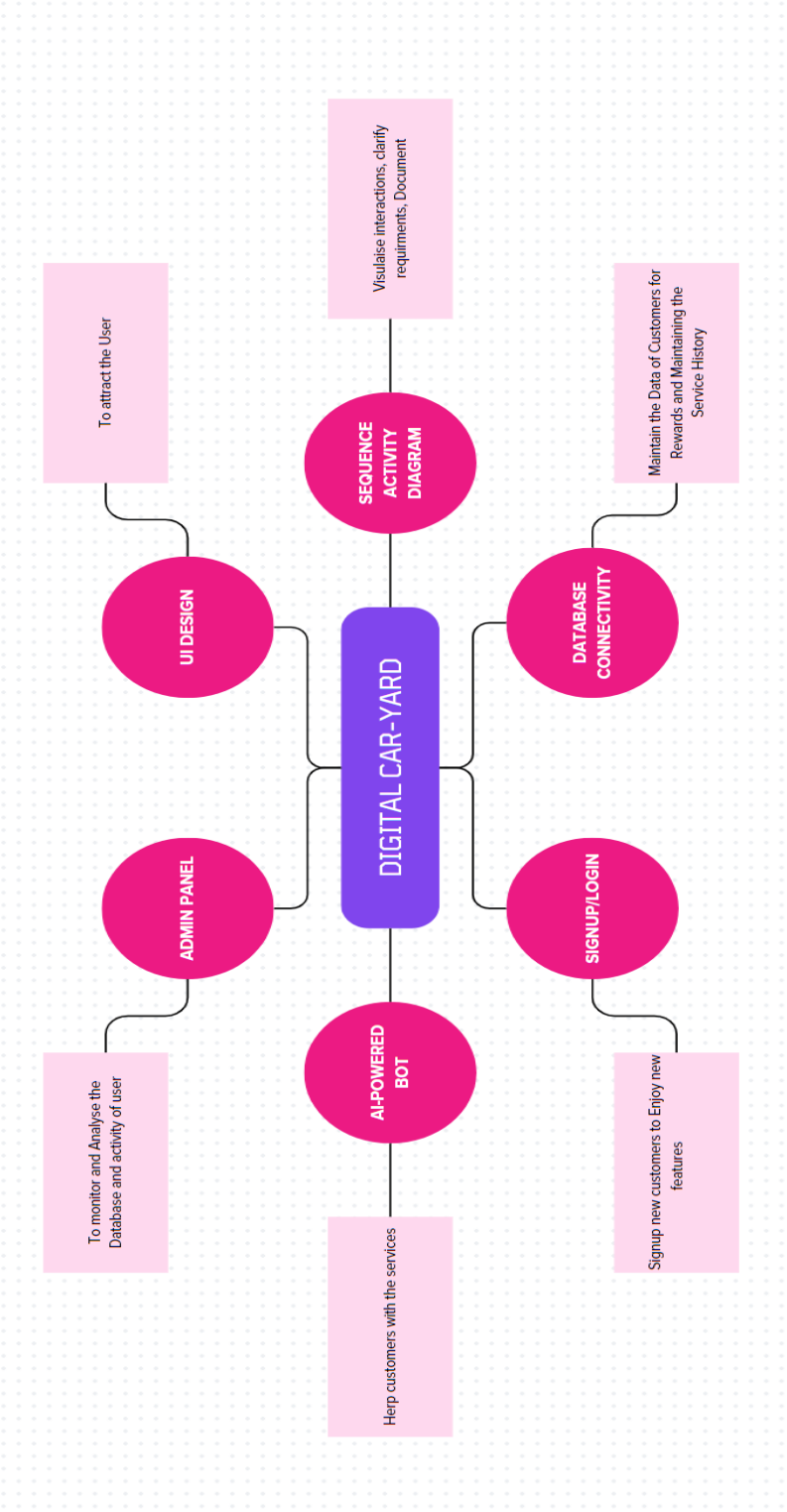

Figure 2 Work Breakdown (FYP-1)

FYP-II

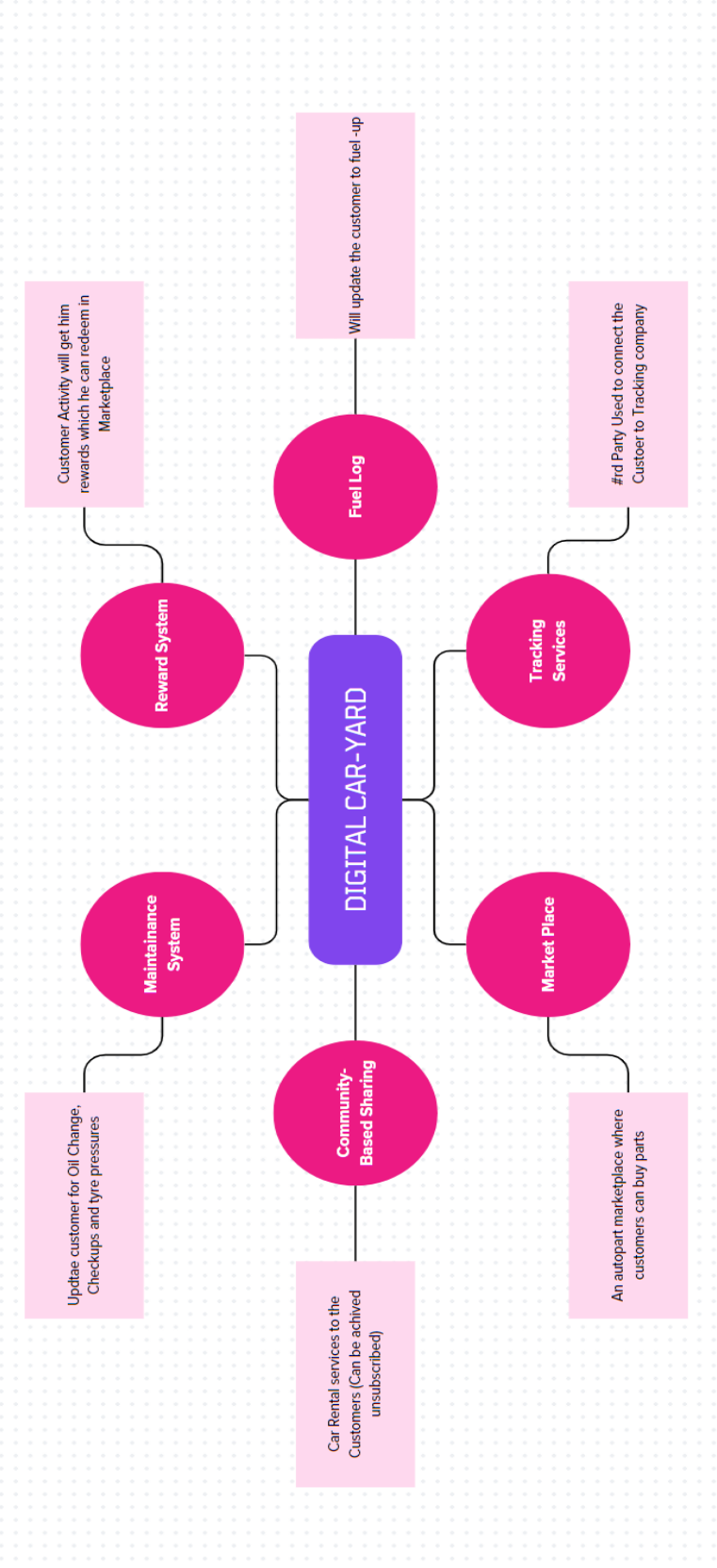

Figure 3 Work Breakdown (FYP-2)

Software Requirement Specifications

1. Introduction

1.1 Purpose

The purpose of this document is to provide details of our Web App “Digital Car-Yard”. This document includes all the functional and non-functional requirements of our system. It specifies the scope of the project and give the user an outline of the whole system. It also provides a list of all functionalities and features.

1.2 Document Conventions

The format of this SRS is simple. Indentation is used on general topics and or specific points of interest with “Times font” and are in a larger font size compared to the regular text. The headings have been Bold and the text size used is 14 and for regular text the text size used is 12.

1.3 Intended Audience and Reading Suggestions

This document is intended to be read by the project manager and the teachers.

1.4 Product Scope

Digital Car-Yard is a web-based platform designed to streamline the buying, selling, and maintenance of vehicles. It allows users to manage car listings, track maintenance tasks, and access a marketplace for car parts and services. The app offers personalized reminders for vehicle upkeep and supports communication between users and service providers. Its goal is to simplify vehicle ownership and provide a comprehensive solution for car-related transactions and maintenance.

1.5 References

See page no. 139.

2. Overall Description

2.1 Product Perspective

Digital Car-Yard integrates various subsystems, such as user profile management, car listings, and marketplace services, with seamless interaction between these components. External interfaces, like payment gateways and third-party APIs, will be incorporated to handle transactions and services in future iterations.

2.2 Product Functions

The main functions of the product are listed as follows:

- Users will be able to create, update, and manage their personal profiles.
- Users will be able to view and track maintenance tasks.
- Users will be able to search for cars using filters such as price, model, and location.
- Users will be able to receive reminders for scheduled vehicle maintenance tasks like oil changes, tire rotations, and inspections.
- Users will be able to interact with a human-to-human chatbot for real-time assistance and guidance on the platform.
- Users will be able to browse and buy car parts and services.
- Users will be able to sell car parts and communicate with buyers through a built-in chat feature.
- Users will be able to use a shopping cart and payment gateway to complete transactions.
-

2.3 User Classes and Characteristics

- **Users**: Can buy cars and car parts, manage their profiles, and track maintenance tasks through an intuitive interface.
- **Admins**: Can manage car listings, user accounts, and oversee system content for smooth platform operation.
- **Shopkeepers**: Can list car parts, manage inventory, and interact with users via messaging within the marketplace.

2.4 Operating Environment

Hardware Preference:

- No specific hardware required; the web application is accessible on any device (desktop, laptop, tablet, smartphone) that supports modern web browsers.
- Devices should have internet connectivity and a basic display capability.

Software Preference:

- **Operating System**: Compatible with all major operating systems, including Windows, macOS, Linux, Android, and iOS.
- **Web Browser**: Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari (latest versions recommended).
- **Backend**: MongoDb database for data storage, Node.js for backend hosted on cloud servers.
- **Frontend**: Flutter or similar web technologies for building user interfaces.
- **Other Components**: Integration with third-party payment gateway APIs for processing transactions in the marketplace.

2.5 Design and Implementation Constraints

The app must have a user-friendly interface, which will allow user to use the app with ease. Furthermore, the design should be intuitive, ensuring that users can manage car listings, track maintenance, and access the marketplace without any technical difficulties. The system should be optimized for both desktop and mobile devices, offering a seamless experience across all platforms. Security measures, such as secure login and data encryption, must be integrated to protect user information, especially during transactions.

2.6 User Documentation

User manual will be provided. User can also interact with the human-to-human chatbot for any queries.

2.7 Assumptions and Dependencies

The following assumptions are made:

- User will create an authentic account.
- User will enter valid credentials when signing up.
- Application will be accessed via modern web browsers, and consistent internet connectivity on both desktop and mobile devices.
- Shopkeepers will independently manage their listings and inventory within the marketplace, using the platform's tools.
- External factors, such as changes in API availability or legal regulations for online transactions, could impact the functionality or compliance of the system.

3. External Interface Requirements

3.1 User Interfaces

See page 129

3.2 Hardware Interfaces

No direct hardware interfaces, as this is a web-based application.

3.3 Software Interfaces

Database:

- **MongoDB** for storing user profiles, car listings, maintenance logs, marketplace items, and transaction data.
- **Incoming data**: User details, car listings, maintenance logs.
- **Outgoing data**: Car search results, user profiles, maintenance reminders.

Backend Development:

- **MongoDB** and **Node.js** for processing requests, managing data, and interacting with the database.
- **Incoming data**: User inputs (e.g., car listings, maintenance logs).
- **Outgoing data**: Search results, maintenance alerts, transaction updates.

Frontend Framework:

- **Flutter** for building and updating the user interface dynamically.
- **Incoming data**: Data from the backend (car listings, user profiles, maintenance data).
- **Outgoing data**: User interactions (search, marketplace activity, chat messages).

API Development:

- **RESTful APIs** for communication between frontend, backend, and external systems (e.g., payment gateways).
- **Incoming data**: API requests for database access, payment confirmations.
- **Outgoing data**: JSON responses containing search results, user information, and transaction statuses.

Payment Gateway:

- **Stripe API** for secure transactions in the marketplace.
- **Incoming data**: Transaction details from the app.
- **Outgoing data**: Payment confirmations and billing records.

Communication Protocols:

- **HTTP/HTTPS** for secure communication between frontend, backend, and third-party APIs.
- **Incoming data**: User requests.
- **Outgoing data**: Search results, purchase confirmations.

3.4 Communications Interfaces

- **HTTP/HTTPS** protocols for secure communication.
- **Email notifications** for reminders and confirmations.

4. System Features

4.1 Login

4.1.1 Description and Priority

This feature allows users to securely access their accounts using their registered email and password. It includes validations to ensure correct credentials are provided and allows for multiple user types, including admins, shopkeepers, and regular users.

4.1.2 Stimulus/Response Sequences

Use Case Name: Login.
Description: Users can securely log in to access the system's features.
Actors: Users, Admin, Shopkeepers.
Goal: Authenticate users to grant access to their respective dashboards.
Pre-Conditions: The user must have a registered account with valid credentials.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** The user enters their email and password on the login page. **System Response:** The system validates the credentials against stored data. 2. **User Action (Step 2):** If the credentials are correct, the user clicks the login button. **System Response:** The system redirects the user to their respective dashboard.
Alternate Flow (Invalid Credentials): **User Action**: The user enters incorrect credentials. **System Response**: The system displays an error message like "Invalid email or password."
Exception Flow (System Errors): **System Failure:** The system is unable to connect to the database. **System Response:** The system displays an error message, "Unable to process your request. Please try again later."
Post Conditions: The user is successfully logged in or shown appropriate error messages.

Table 1 Login (S/RS)

4.1.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall authenticate users based on a valid email and password.	High	Pending
Req 2	The system shall display an error message for invalid login attempts.	High	Pending
Req 3	The system shall redirect authenticated users to their respective dashboards.	High	Pending

Table 2 Login (FR)

4.2 Forgot Password

4.2.1 Description and Priority

This feature helps users recover access to their accounts in case they forget their password. Users receive a password reset link via email after entering their registered email address, ensuring secure account recovery.

4.2.2 Stimulus/Response Sequences

Use Case Name: Forgot Password.
Description: Users can reset their password if they forget it.
Actors: Users, Shopkeepers, Admins.
Goal: Allow users to reset their password securely.
Pre-Conditions: The user must have a registered email address.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** The user clicks on "Forgot Password" on the login page. **System Response:** The system displays a password reset form. **2. User Action (Step 2):** The user enters their registered email address and submits the form. **System Response:** The system sends a password reset link to the provided email. 3. **User Action (Step 3):** The user clicks the link and creates a new password. **System Response:** The system updates the password and confirms the change.
Alternate Flow (Unregistered Email): **User Action**: The user enters an unregistered email. **System Response**: The system displays an error message, "Email not found."
Exception Flow (Invalid or unregistered email): **System Failure:** The user enters an email address that is not associated with an account. **System Response:** The system displays an error message like "The email entered is not recognized. Please check and try again.
Post Conditions: The password is reset successfully, or an appropriate error message is displayed.

Table 3 Forget password (S/RS)

4.2.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall allow users to request a password reset by providing their email.	High	Pending
Req 2	The system shall validate the email against registered accounts.	High	Pending
Req 3	The system shall send a password reset link to the user's email.	High	Pending
Req 4	The system shall allow users to create a new password through the reset link.	High	Pending

Table 4 forget password (FR)

4.3 Log out

4.3.1 Description and Priority

This feature allows users to securely log out of their accounts, terminating their session and redirecting them to the login page. This ensures data security and prevents unauthorized access.

4.3.2 Stimulus/Response Sequences

Use Case Name: Log out.
Description: This functionality allows users to securely log out of the system.
Actors: Users (Car buyers, Sellers, Shopkeepers, Admins).
Goal: Securely log the user out of their session and redirect them to the login page.
Pre-Conditions: The user must be logged in.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** The user clicks on the logout button. **System Response:** The system terminates the user session and redirects the user to the login page.
Alternate Flow (Session Already Expired): **User Action**: The user attempts to log out after the session has already expired. **System Response**: The system displays a message like "Session expired. Please log in again."
Exception Flow (System Error): **System Failure:** The system encounters an issue while logging the user out. **System Response:** The system displays an error message like "Unable to log out at the moment. Please try again later."
Post Conditions: The user session is successfully terminated.

Table 5 Logout (S/RS)

4.3.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall provide a logout button accessible from all pages when the user is logged in.	High	Pending
Req 2	The system shall terminate the session upon clicking the logout button.	High	Pending
Req 3	The system shall redirect the user to the login page after successful logout.	High	Pending
Req 4	The system shall display an error message if the logout process fails due to system issues.	Medium	Pending
Req 5	The system shall notify users if their session has already expired during the logout attempt.	Medium	Pending

Table 6 Logout (FR)

4.4 Chat

4.4.1 Description and Priority

This use case describes the functionality that allows a user to initiate and engage in a chat with the shopkeeper or the admin for product inquiries, order-related questions or any sort of assistance.

4.4.2 Stimulus/Response Sequences

Use Case Name: Chat
Description: This use case allows a user to initiate and manage a chat with the shopkeeper and the admin for product or service inquiries.
Actors: Users.
Goal: Provide real-time assistance through a chat interface that connects users to the admin or the shopkeeper.
Pre-Conditions: The user must be logged in to access full support features.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** The user clicks on the chat icon for assistance. **System Response:** The system transfers the chat to admin or shopkeeper, who assists the user.
Alternate Flow (No Agent Available): **User Action**: The user requests assistance, but shopkeeper or admin is not available. **System Response**: The system shows a message like "Admin is currently busy. Please try again later."
Exception Flow (Unable to send): **User Action:** The user tries sends a message, but unable to send. **System Response:** The system shows a message like “Unable to send message, Try again Later”
Post Conditions: The user receives the required assistance or is notified to retry later.

Table 7 Chat (S/RS)

4.4.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall display a chat interface where users can communicate with shopkeeper or admin.	High	Pending
Req 2	The system shall allow users to send messages and receive real-time replies.	High	Pending
Req 3	The system shall notify users when a shopkeeper or admin is available for assistance.	Medium	Pending
Req 4	The system shall maintain a chat history for users to refer back to previous conversations.	Medium	Pending

Table 7 Chat (FR)

4.5 Maintenance Reminder System

4.5.1 Description and Priority

This system sends users notifications reminding them of scheduled car maintenance tasks, such as oil changes, tire rotations, and inspections. This helps users keep their vehicles in optimal condition.

4.5.2 Stimulus/Response Sequences

Use Case Name: Maintenance Reminder System
Description: Users can set reminders for routine car maintenance (e.g., oil changes, tire rotations).
Actors: Users.
Goal: Enable users to set up automatic reminders for their vehicle's maintenance.
Pre-Conditions: The user must be logged in.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** The user navigates to the **Maintenance Reminders** section and selects **Register**. **System Response:** The system displays a registration form to enter vehicle details (model, year, etc.) and maintenance preferences. 2. **User Action (Step 2):** The user fills out the form with all required details (e.g., type of maintenance, frequency). **System Response:** The system validates the input and registers the user for maintenance reminders. 3. **User Action (Step 3):** **System Response:** The system schedules and sends reminders to the user based on the provided details (via email).
Alternate Flow (No Vehicle Details Provided): **User Action**: The user attempts to register without providing complete vehicle details **System Response**: The system prompts the user to provide complete details.
Exception Flow (System Failure): **System Failure:** The system encounters an error during registration **System Response:** The system shows an error message and asks the user to try again later.
Post Conditions: Maintenance reminder is set successfully, and the system sends timely notifications.

Table 8 Maintenance reminder system (S/RS)

4.5.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall display a registration form requiring vehicle details and maintenance preferences.	High	Pending
Req 2	The system shall store maintenance preferences and schedule reminders accordingly.	High	Pending
Req 3	The system shall notify users of upcoming maintenance based on their preferences via email.	High	Pending
Req 4	The system shall store maintenance history and allow users to view completed tasks.	Medium	Pending

Table 9 Maintenance reminder system (FR)

4.6 Navigate to Marketplace

4.6.1 Description and Priority

This use case allows users to access the marketplace to browse car parts, accessories, and services. It provides an entry point for users to explore available products, organized by categories or filters.

4.6.2 Stimulus/Response Sequences

Use Case Name: Navigate to Marketplace.
Description: This use case allows users to access the marketplace to browse car parts and services.
Actors: Users.
Goal: Enable users to enter the marketplace and view a list of available products and services.
Pre-Conditions: The user must be logged in.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** User clicks the "Marketplace" button on the navigation bar or homepage. **System Response:** The system retrieves and displays a list of available items from the database. 2. **User Action (Step 2):** User sees categorized sections (e.g., car parts, accessories). **System Response:** The system organizes items based on categories or filters and displays them.
Alternate Flow (No Items Available): **User Action**: The marketplace is empty (e.g., no items listed yet). **System Response:** The system displays a message: "No items available in the marketplace."
Exception Flow (System Errors): **System Failure:** The system fails to retrieve marketplace data from the database. **System Response:** The system displays an error message and suggests retrying later.
Post Conditions: The user is successfully redirected to the marketplace, viewing available products or services.

Table 10 Navigate to marketplace (S/RS)

4.6.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall provide a navigation option for users to access the marketplace.	High	Pending
Req 2	The system shall retrieve and display a list of available items upon navigation.	High	Pending
Req 3	The system shall notify users if the marketplace is empty.	Medium	Pending

Table 11 Navigate to marketplace (FR)

4.7 Search Cars/Car Parts

4.7.1 Description and Priority

This feature enables users to search for cars or car parts using partial or full names. It supports letter-matching functionality to return relevant results.

4.7.2 Stimulus/Response Sequences

Use Case Name: Search Cars/Car Parts.
Description: This feature enables users to search for cars or car parts using partial or full names.
Actors: Users.
Goal: To enable users to quickly and efficiently find car or car part listings by providing an intuitive and flexible search mechanism.
Pre-Conditions: The user must be logged in
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** User enters a letter in the search bar. **System Response:** System performs a letter-matching search to match the input query with car or car part names in the database. 2. **User Action (Step 2):** User views the search results page with matching car listings/car parts. **System Response:** The system shows all matching listings/car parts, including a summary for each car.
Alternate Flow (No Results Found): **User Action**: The user enters search criteria that return no results. **System Response**: The system displays nothing.
Exception Flow (System Errors): **System Failure:** The system encounters an error retrieving search result. **System Response:** The system displays an error message and asks the user to try again.
Post Conditions: The user views the search results successfully.

Table 12 Search car/car parts (S/RS)

4.7.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall provide a search bar for users to input car/car parts search criteria.	High	Pending
Req 2	The system shall display matching cars/car parts in a list with basic information (e.g., make, model, price).	High	Pending
Req 3	The system shall notify users when no cars/car parts match their search criteria.	Medium	Pending

Table 13 search car/car parts (FR)

4.8 View Car/Car Parts Details

4.8.1 Description and Priority

This feature allows users to view detailed information about a specific car/car part from the search results or listing. It provides a comprehensive overview, including images, specifications, and seller contact details.

4.8.2 Stimulus/Response Sequences

Use Case Name: View Car/Car Parts Details.
Description: Users can view detailed information about a selected car/car part from the search results.
Actors: Users.
Goal: Allow users to access full details of a car/car part for informed decision-making.
Pre-Conditions: The user must be logged in.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** User clicks on a car listing or a car part from the search results. **System Response:** The system displays the car's/car parts' full details, including images, specifications, and price. 2. **User Action (Step 2):** User scrolls through the car's/car parts' information (e.g., description, seller contact details). **System Response:** The system provides a seamless display of all relevant car/car part information.
Alternate Flow (No Data Available): **User Action**: The user clicks on a listing, but some data (e.g., images or description) is unavailable. **System Response**: The system shows a message like "Some details are currently unavailable."
Exception Flow (System Errors) **System Failure:** The system encounters an error loading the car/car part details. **System Response:** The system displays an error message and asks the user to retry.
Post Conditions: The user views the car's/car parts' full details successfully.

Table 14 view car/ car part details (S/RS)

4.8.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall allow users to click on a listing to view full details of the car/car part.	High	Pending
Req 2	The system shall display complete car/car part information, including images, specifications, and price.	High	Pending

Table 15 view cars/ car part details (FR)

4.9 Payment Gateway

4.9.1 Description and Priority

The platform will integrate a secure payment gateway to process transactions for purchases made through the marketplace. This feature will handle various payment methods (e.g., credit/debit cards, digital wallets).

4.9.2 Stimulus/Response Sequences

Use Case Name: Payment Gateway.
Description: Users can make secure payments for products or services on the platform.
Actors: Users (Car parts buyers).
Goal: Allow users to complete transactions using secure payment methods.
Pre-Conditions: The user must have a valid cart and proceed to checkout.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** The user navigates to the payment page after confirming the cart items. **System Response:** The system presents the user with payment options. 2. **User Action (Step 2):** The user enters payment information and clicks on "Pay Now". **System Response:** The system validates the payment details and processes the transaction. 3. **User Action (Step 3):** The payment is confirmed. **System Response:** The system shows a confirmation message and sends a receipt to the user's email.
Alternate Flow (Invalid Inputs): **User Action**: The user enters incorrect payment details (e.g., expired card). **System Response**: The system shows an error message and requests valid payment details.
Exception Flow (System Errors): **System Failure:** The payment processor is unavailable or there is a network issue. **System Response:** The system shows a message like "Payment gateway temporarily unavailable. Please try again later."
Post Conditions: The payment is processed, and the user receives a confirmation.

Table 16 payment gateway (S/RS)

4.9.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall integrate a secure payment gateway to process transactions for users purchasing products.	High	Pending
Req 2	The system shall ensure that sensitive payment information is encrypted and securely transmitted.	High	Pending
Req 3	The system shall notify users when a payment has been successfully completed or if there are errors in payment.	High	Pending
Req 4	The system shall allow users to review their order summary before confirming payment.	Medium	Pending
Req 5	The system shall integrate with the user's account to track payment history and manage refunds/cancellations.	Medium	Pending

Table 17 payment gateway (FR)

4.10 Chat (Admin, Shopkeeper)

4.10.1 Description and Priority

This use case details the functionality for the shopkeeper or the admin to manage, respond to, and organize chat messages from users.

4.10.2 Stimulus/Response Sequences

Use Case Name: Chat (Admin, Shopkeeper).
Description: This use case allows the shopkeeper and the admin to manage and respond to messages from users.
Actors: Admin, Shopkeeper.
Goal: Enable shopkeeper and admin to respond to user inquiries, view conversation history, and manage chats.
Pre-Conditions: The shopkeeper or the admin must be logged in.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** Admin/Shopkeeper navigates to the chat interface to view ongoing conversations. **System Response:** The system displays a list of chats. 2. **User Action (Step 2):** Admin/Shopkeeper selects a chat and sends a response to the user's query. **System Response:** The system records the message and delivers it to the user in real time.
Alternate Flow (No Active Chats): **User Action**: No users have initiated chats with the shopkeeper or the admin. **System Response**: The system displays a message: "No active chats at the moment."
Exception Flow (System Errors) **System Failure:** The system encounters an error while sending the message. **System Response:** The system displays an error message and suggests retrying.
Post Conditions: The shopkeeper or the admin successfully responds to users or is informed of any issues.

Table 18 Chat (Admin. Shopkeeper) (S/RS)

4.10.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall display a list of active chats.	High	Pending
Req 2	The system shall allow shopkeepers to send real-time responses to users.	High	Pending
Req 3	The system shall display the complete conversation history for each chat.	High	Pending

Table 19 Chat (Admin, Shopkeeper) (FR)

4.11 List Cars

4.11.1 Description and Priority

This feature allows users to list their cars on the platform by providing details like make, model, year, price, and condition. The listing will require admin approval before it becomes visible to other users.

4.11.2 Stimulus/Response Sequences

Use Case Name: List Cars
Description: Users can submit car details to create a listing, pending admin approval.
Actors: Users (Car Sellers).
Goal: Allow users to list cars for sale on the platform.
Pre-Conditions: The user must be logged in.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** User clicks on the "List a Car" button and fills out the car's details (e.g., make, model, year, price, description). **System Response:** The system saves the listing as pending for admin approval. 2. **User Action (Step 2):** Admin reviews the car listing. **System Response:** The system either approves or rejects the listing and notifies the user.
Alternate Flow (Missing Information): **User Action**: The user submits the form without completing all required fields. **System Response**: The system prompts the user to fill in the missing details.
Exception Flow (System Errors) **System Failure:** The system encounters an error while submitting the car listing. **System Response:** The system displays an error message and asks the user to try again.
Post Conditions: The car listing is successfully created and awaits admin approval.

Table 20 List Cars (S/RS)

4.11.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall provide a form for users to input car details for listing.	High	Pending
Req 2	The system shall save the listing as pending until admin approval is granted.	High	Pending
Req 3	The system shall validate that all required fields are filled before submission.	Medium	Pending

Table 21 List Cars (FR)

4.12 View Car Listings

4.12.1 Description and Priority

This feature allows users to view all their submitted car listings, along with their approval status.

4.12.2 Stimulus/Response Sequences

Use Case Name: View Car Listings.
Description: Users can view all the car listings they have created, including their status.
Actors: Users (Car Sellers).
Goal: Allow users to manage and track the status of their car listings.
Pre-Conditions: The user must be logged in.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** User navigates to the "My Cars" section. **System Response:** The system displays all the car listings created by the user, along with their status (pending, approved, or rejected).
Alternate Flow (No Listings Found): **User Action**: The user has not created any listings yet. **System Response**: The system displays a message like "No listings found. Start creating one!"
Exception Flow (System Errors) **System Failure:** The system encounters an error retrieving the user's listings. **System Response:** The system displays an error message and asks the user to try again.
Post Conditions: The user successfully views their car listings.

Table 22 View Car Listings (S/RS)

4.12.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall display all car listings created by the user, along with their approval status.	High	Pending
Req 2	The system shall show a message when no listings are available for the user.	Medium	Pending

Table 23 View Car Listings (FR)

4.13 Delete Car Listing

4.13.1 Description and Priority

This feature allows users to delete their car listings from the platform, removing them permanently.

4.13.2 Stimulus/Response Sequences

Use Case Name: Delete Car Listing.
Description: Users can delete their car listings permanently from the platform.
Actors: Users (Sellers).
Goal: Allow users to remove their car listings.
Pre-Conditions: The user must be logged in.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** User navigates to "My Listings" and selects a car listing to delete. **System Response:** The system permanently removes the listing and notifies the user.
Alternate Flow (Cancelled Deletion): **User Action**: The user cancels the deletion when prompted for confirmation. **System Response**: The system retains the listing and returns to the "My Listings" page.
Exception Flow (System Errors) **System Failure:** The system encounters an error while deleting the listing. **System Response:** The system displays an error message and asks the user to try again.
Post Conditions: The car listing is successfully deleted, or the action is canceled.

Table 24 Delete Car Listing (S/RS)

4.13.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall allow users to delete their car listings permanently.	High	Pending
Req 2	The system shall prompt the user for confirmation before deleting a car listing.	Medium	Pending

Table 25 Delete Car Listings (FR)

4.14 Add to Cart

4.14.1 Description and Priority

This feature allows users to add car parts to their shopping cart, allowing them to gather items they intend to purchase.

4.14.2 Stimulus/Response Sequences

Use Case Name: Add to Cart.
Description: Users can add car parts to their shopping cart for potential purchase.
Actors: Users.
Goal: Allow users to save car parts in their cart for easy access when ready to checkout.
Pre-Conditions: The user must be logged in.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** User navigates to the car part they want to purchase. **System Response:** The system displays the part's details and an option to add it to the cart. 2. **User Action (Step 2):** User clicks "Add to Cart." **System Response:** The system adds the item to the user's cart.
Alternate Flow (Item Already in Cart): **User Action**: The user tries to add an item that's already in their cart. **System Response**: The system informs the user that the item is already in the cart and suggests updating the quantity.
Exception Flow (System Errors) **System Failure:** The system encounters an error while adding the item to the cart. **System Response:** The system displays an error message and asks the user to try again.
Post Conditions: The item is successfully added to the user's cart.

Table 26 Add to Cart (S/RS)

4.14.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall allow users to add car parts to their shopping cart.	High	Pending
Req 2	The system shall prevent duplicate items in the cart unless the user updates the quantity.	Medium	Pending

Table 27 Add to Cart (FR)

4.15 Delete from Cart

4.15.1 Description and Priority

This feature allows users to remove items from their shopping cart, either due to a change of mind or an error in selection.

4.15.2 Stimulus/Response Sequences

Use Case Name: Delete from Cart.
Description: Users can remove car parts from their shopping cart.
Actors: Users.
Goal: Allow users to delete unwanted items from their cart.
Pre-Conditions: The user must be logged in and have items in the cart.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** User navigates to the cart and selects the item they want to delete. **System Response:** The system displays a delete option for the selected item. 2. **User Action (Step 2):** User clicks "Delete" for the item. **System Response:** The system removes the item from the cart and displays a confirmation message. 3. **User Action (Step 3):** **System Response:**
Alternate Flow (Cancelled Deletion): **User Action**: The user decides not to delete the item when prompted. **System Response**: The system retains the item in the cart.
Exception Flow (System Errors) **System Failure:** The system encounters an error while removing the item from the cart. **System Response:** The system displays an error message and asks the user to try again.
Post Conditions: The item is successfully removed from the user's cart.

Table 28 Delete from Cart (S/RS)

4.15.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall allow users to delete items from them shopping cart.	High	Pending
Req 2	The system shall prompt the user to confirm deletion.	Medium	Pending

Table 29 Delete from Cart (FR)

4.16 View Cart

4.16.1 Description and Priority

This feature allows users to view all the items in their cart, along with details like price, quantity, and total cost, before proceeding to checkout.

4.16.2 Stimulus/Response Sequences

Use Case Name: View Cart.
Description: Users can view all items currently in their cart.
Actors: Users.
Goal: Allow users to review their cart items, modify quantities, and view the total cost.
Pre-Conditions: The user must be logged in.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** User clicks on the "Cart" icon or navigates to the cart page. **System Response:** The system displays all items in the cart with details like price, quantity, and total cost.
Alternate Flow (Empty Cart): **User Action**: The user has no items in the cart. **System Response**: The system displays a message like "Your cart is empty" and suggests browsing products.
Exception Flow (System Errors) **System Failure:** The system encounters an error while retrieving the cart items. **System Response:** The system displays an error message and asks the user to try again.
Post Conditions: The user successfully views their cart contents.

Table 30 View Cart (S/RS)

4.16.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall display all items in the user's cart, along with details like price and quantity.	High	Pending
Req 2	The system shall calculate and display the total cost of all items in the cart.	High	Pending
Req 3	The system shall display a message when the cart is empty.	Medium	Pending

Table 31 View Cart (FR)

4.17 Checkout

4.17.1 Description and Priority

This feature allows users to proceed with the purchase of the items in their cart, providing payment and shipping details to complete the transaction.

4.17.2 Stimulus/Response Sequences

Use Case Name: Checkout.
Description: Users can complete the purchase of items in their cart by providing payment information.
Actors: Users.
Goal: Allow users to complete their purchase and place an order.
Pre-Conditions: The user must be logged in and have items in their cart.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** User clicks on the "Checkout" button from the cart page. **System Response:** The system prompts the user for payment and shipping details. 2. **User Action (Step 2):** User enters payment and shipping details and confirms the purchase. **System Response:** The system processes the payment, confirms the order, and provides an order summary.
Alternate Flow (Insufficient Funds): **User Action**: The payment fails due to insufficient funds or payment rejection. **System Response**: The system notifies the user of the payment failure and suggests retrying with a different method.
Exception Flow (System Errors) **System Failure:** The system encounters an error while processing the payment. **System Response:** The system displays an error message and asks the user to try again.
Post Conditions: The order is successfully placed and payment is processed.

Table 32 Checkout (S/RS)

4.17.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall prompt users for payment and shipping details during checkout.	High	Pending
Req 2	The system shall process the payment and confirm the order.	High	Pending
Req 3	The system shall display a summary of the order upon successful payment.	Medium	Pending
Req 4	The system shall notify users of any payment errors and provide options to retry.	Medium	Pending

Table 33 Checkout (FR)

4.18 Sign up

4.18.1 Description and Priority

This feature allows new users, such as car buyers and sellers, to create an account by providing basic information. This functionality ensures a personalized experience and access to platform features.

4.18.2 Stimulus/Response Sequences

Use Case Name: Sign up.
Description: This functionality allows users to create a new account.
Actors: Users (Car buyers, Sellers).
Goal: Enable users to register an account on the platform.
Pre-Conditions: The user must not have an existing account.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** The user clicks on the "Sign Up" button. **System Response:** The system displays a registration form. 2. **User Action (Step 2):** The user enters required details (e.g., name, email, password, phone number). **System Response:** The system validates the input fields. 3. **User Action (Step 3):** The user submits the registration form. **System Response:** The system creates a new user account and displays a success message.
Alternate Flow (Incomplete Form Submission): **User Action**: The user submits the form without filling in all the required fields. **System Response**: The system highlights the missing fields and displays a message like "Please fill in all required fields."
Exception Flow (Duplicate Email): **System Failure:** The user enters an email address already associated with an existing account. **System Response:** The system displays a message like "Email already in use. Please use a different email or log in."
Post Conditions: A new user account is successfully created.

Table 34 Signup (S/RS)

4.18.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall provide a "Sign Up" button accessible on the login page.	High	Pending
Req 2	The system shall display a form for users to enter required information (name, email, password, etc.).	High	Pending
Req 3	The system shall validate user inputs to ensure data accuracy and completeness.	High	Pending

Table 35 Sign up (FR)

4.19 Add Car Part

4.19.1 Description and Priority

This feature allows shopkeepers to list car parts for sale by providing details like part name, description, price, and stock quantity. The parts will be immediately visible to users after submission since the shopkeeper has already been approved by the admin.

4.19.2 Stimulus/Response Sequences

Use Case Name: Add Car Part.
Description: Shopkeepers can add car parts to the marketplace by providing necessary details.
Actors: Shopkeepers.
Goal: Allow shopkeepers to list car parts for users to browse and purchase.
Pre-Conditions: The shopkeeper must be logged in.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** Shopkeeper navigates to the "Add Car Part" section. **System Response:** The system displays a form for entering part details (e.g., name, price, description, stock). 2. **User Action (Step 2):** Shopkeeper fills in the required details and submits the form. **System Response:** The system validates the data, saves the listing, and makes it visible to users.
Alternate Flow (Missing or Invalid Data): **User Action**: The shopkeeper submits the form with incomplete or invalid data. **System Response**: The system highlights the errors and prompts the shopkeeper to correct them.
Exception Flow (System Errors) **System Failure:** The system encounters an error while saving the car part details. **System Response:** The system displays an error message and asks the shopkeeper to try again.
Post Conditions: The car part is successfully added and displayed in the marketplace.

Table 36 Add to cart (S/RS)

4.19.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall provide a form for shopkeepers to add car part details.	High	Pending
Req 2	The system shall make the car part immediately visible in the marketplace after submission.	High	Pending

Table 37 Add to cart (FR)

4.20 Delete Car Part

4.20.1 Description and Priority

This feature allows shopkeepers to permanently remove car parts from the marketplace.

4.20.2 Stimulus/Response Sequences

Use Case Name: Delete Car Part.
Description: Shopkeepers can delete car part listings that are no longer available or relevant.
Actors: Shopkeepers.
Goal: Allow shopkeepers to manage their inventory by removing outdated car parts.
Pre-Conditions: The shopkeeper must be logged in and have existing car part listings.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** Shopkeeper navigates to their car part listing and selects the "Delete" option. **System Response:** The system prompts the shopkeeper for confirmation. 2. **User Action (Step 2):** Shopkeeper confirms the deletion. **System Response:** The system deletes the car part and removes it from the marketplace.
Alternate Flow (Cancelled Deletion): **User Action**: The shopkeeper decides not to delete the car part after seeing the confirmation prompt. **System Response**: The system retains the car part in the marketplace.
Exception Flow (System Errors) **System Failure:** The system encounters an error while deleting the car part. **System Response:** The system displays an error message and asks the shopkeeper to try again.
Post Conditions: The car part is successfully deleted, or the action is cancelled.

Table 38 Delete Car Part (S/RS)

4.20.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall allow shopkeepers to delete their car part listings.	High	Pending
Req 2	The system shall prompt the shopkeeper for confirmation before deleting the car part.	Medium	Pending

Table 39 Delete Car Part (FR)

4.21 View Car Parts

4.21.1 Description and Priority

This feature allows shopkeepers to view all car parts they have listed, along with details like stock quantity, price, and status.

4.21.2 Stimulus/Response Sequences

Use Case Name: View Car Parts.
Description: Shopkeepers can view a list of all their car part listings.
Actors: Shopkeepers.
Goal: Allow shopkeepers to monitor their inventory and sales information.
Pre-Conditions: The shopkeeper must be logged in.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** Shopkeeper navigates to the "My Listings" section. **System Response:** The system displays all car parts listed by the shopkeeper, with details like name, stock, and price.
Alternate Flow (No Listings Available): **User Action**: The shopkeeper has not listed any car parts yet. **System Response**: The system displays a message like "No parts listed yet. Start creating one!"
Exception Flow (System Errors) **System Failure:** The system encounters an error retrieving the car part listings. **System Response:** The system displays an error message and asks the shopkeeper to try again.
Post Conditions: The shopkeeper successfully views their car part listings.

Table 40 View Car Part (S/RS)

4.21.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall display a list of all car parts listed by the shopkeeper, with relevant details.	High	Pending
Req 2	The system shall notify the shopkeeper if no listings are available.	Medium	Pending

Table 41 View Car Part (FR)

4.22 View System Analytics

4.22.1 Description and Priority

This feature provides admins with analytics and insights about platform usage, such as the number of active users, number of listings, and transaction volumes. It helps in monitoring the platform's performance and planning improvements.

4.22.2 Stimulus/Response Sequences

Use Case Name: View System Analytics
Description: Admin can view analytics like total users, listings, transactions, and system usage trends.
Actors: Admin.
Goal: Provide insights into platform performance and usage.
Pre-Conditions: Admin must be logged in.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** The admin selects "View Platform Metrics" from the dashboard. **System Response:** The system displays analytics dashboards with charts and statistics.
Alternate Flow (Metrics unavailable): **Admin Action**: The admin tries to view platform metrics during a period of inactivity or insufficient data collection. **System Response**: The system displays a message such as "No metrics available for this time period. Please try again later."
Exception Flow (System Errors) **System Failure:** The system encounters an error while retrieving metrics (e.g., database connection issue). **System Response:** The system shows an error message like "Unable to fetch metrics at the moment. Please contact technical support."
Post Conditions: The admin successfully views the metrics and insights for the platform.

Table 42 View System Analytics (S/RS)

4.22.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall display statistics on total users, car listings, and transactions.	High	Pending

Table 43 View System Analytics (FR)

4.23 Approve Car Listing

4.23.1 Description and Priority

This feature allows admins to review and approve car listings submitted by users, making them visible on the platform.

4.23.2 Stimulus/Response Sequences

Use Case Name: Approve Car Listing.
Description: Admins can approve car listings to make them publicly visible.
Actors: Admin.
Goal: Ensure only approved and verified listings are visible to users.
Pre-Conditions: The admin must be logged in, and pending car listing requests must exist.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** Admin navigates to the "Pending Car Listings" section. **System Response:** The system displays all pending car listing requests. 2. **User Action (Step 2):** Admin selects a car listing to review and clicks "Approve." **System Response:** The system approves the listing and notifies the user.
Alternate Flow (No Pending Listings): **User Action**: There are no pending car listings to review. **System Response**: The system displays a message like "No pending car listings at the moment."
Exception Flow (System Errors) **System Failure:** The system encounters an error while approving the car listing. **System Response:** The system displays an error message and asks the admin to try again.
Post Conditions: The car listing is approved and visible to all users.

Table 44 Approve Car Listing (S/RS)

4.23.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall allow admins to approve pending car listing requests.	High	Pending
Req 2	The system shall display a success message after a listing is approved.	Medium	Pending

Table 45 Approve Car Listing (FR)

4.24 Reject Car Listing

4.24.1 Description and Priority

This feature allows admins to reject car listings submitted by users, removing them from the review queue.

4.24.2 Stimulus/Response Sequences

Use Case Name: Reject Car Listing.
Description: Admins can reject car listings they deem unfit or invalid for the platform.
Actors: Admin.
Goal: Ensure that only valid and high-quality car listings are allowed on the platform.
Pre-Conditions: The admin must be logged in, and pending car listing requests must exist.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** Admin navigates to the "Pending Car Listings" section. **System Response:** The system displays all pending car listing requests. 2. **User Action (Step 2):** Admin selects a car listing to review and clicks "Reject." **System Response:** The system rejects the listing and notifies the user.
Alternate Flow (No Pending Listings): **User Action**: There are no pending car listings to review. **System Response**: The system displays a message like "No pending car listings at the moment."
Exception Flow (System Errors): **System Failure:** The system encounters an error while rejecting the car listing. **System Response:** The system displays an error message and asks the admin to try again.
Post Conditions: The car listing is rejected, and the user is notified.

Table 46 Reject Car Listing (S/RS)

4.24.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall allow admins to reject pending car listing requests.	High	Pending

Table 47 Reject Car Listing (FR)

4.25 View Car Listing Requests

4.25.1 Description and Priority

This feature allows admins to view all pending car listing requests in one place for easy review and management.

4.25.2 Stimulus/Response Sequences

Use Case Name: View Car Listing Requests.
Description: Admins can view a list of all car listings awaiting approval or rejection.
Actors: Admin.
Goal: Provide admins with an overview of pending car listings for efficient processing.
Pre-Conditions: The admin must be logged in.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** Admin navigates to the "Pending Car Listings" section. **System Response:** The system displays all car listings awaiting admin action.
Alternate Flow (No Pending Listings): **User Action**: There are no pending car listings to review. **System Response**: The system displays a message like "No pending car listings at the moment."
Exception Flow (System Errors) **System Failure:** The system encounters an error while loading the list of pending requests. **System Response:** The system displays an error message and asks the admin to try again.
Post Conditions: The admin successfully views all pending car listings.

Table 48 View Car Listing Requests (S/RS)

4.25.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall display a list of all car listings awaiting admin action.	High	Pending
Req 2	The system shall notify the admin if no pending listings are available.	Medium	Pending

Table 49 View Car Listing Requests (FR)

4.26 Adjusting Item Quantity in Cart

4.26.1 Description and Priority

This use case enables users to increase or decrease the quantity of items in their shopping cart. The system updates the total cost dynamically.

4.26.2 Stimulus/Response Sequences

Use Case Name: Adjusting Item Quantity in Cart.
Description: This use case allows users to increase or decrease the quantity of an item in their shopping cart.
Actors: Users.
Goal: Enable users to adjust item quantities in the cart while updating the total price dynamically.
Pre-Conditions: The user must be logged in and the user must have items in their cart.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** User navigates to the cart page. **System Response:** The system displays the list of items with their current quantities. 2. **User Action (Step 2):** User clicks the "+" or "-" button next to an item. **System Response:** The system adjusts the quantity and updates the total price.
Alternate Flow (Exceeds Stock): **User Action**: The user tries to increase the quantity beyond available stock. **System Response**: The system displays an error message: "Only X items available."
Exception Flow (Database Error) **System Failure:** The system fails to update the quantity in the database. **System Response:** The system displays an error message and prompts the user to retry.
Post Conditions: The cart is updated with the correct quantity and total price.

Table 50 Adjusting Item Quantity in Cart (S/RS)

4.26.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall allow users to increase the quantity of items in the cart.	High	Pending
Req 2	The system shall allow users to decrease the quantity of items in the cart.	High	Pending
Req 3	The system shall update the cart total dynamically based on quantity changes.	High	Pending

Table 51 Adjusting Item Quantity in Cart (FR)

4.27 View Received Orders

4.27.1 Description and Priority

This use case allows the shopkeeper to view a list of all received orders, including basic information such as order date, user details, and total amount.

4.27.2 Stimulus/Response Sequences

Use Case Name: View Received Orders
Description: This use case allows shopkeepers to view all received orders, including order details such as user information, items, and status.
Actors: Shopkeeper.
Goal: Provide shopkeepers with a list of received orders for efficient order management.
Pre-Conditions: The shopkeeper must be logged in and the shopkeeper must have received at least one order.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** Shopkeeper navigates to the "Orders" section in their dashboard. **System Response:** The system displays a list of all received orders.
Alternate Flow (No Orders): **User Action**: The shopkeeper has not received any orders. **System Response**: The system displays a message: "No orders received yet."
Exception Flow (System Errors) **System Failure:** The system fails to fetch order data. **System Response:** The system displays an error message and asks the shopkeeper to try again later.
Post Conditions: The shopkeeper successfully views the list of received orders.

Table 52 View Received Orders (S/RS)

4.27.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall display a list of all orders received by the shopkeeper.	High	Pending
Req 2	The system shall notify the shopkeeper if no orders are available.	Medium	Pending

Table 53 View Received Orders (FR)

4.28 View Order Details

4.28.1 Description and Priority

This use case enables the shopkeeper to view detailed information about a specific order, such as item details, quantity, user contact information, and payment status.

4.28.2 Stimulus/Response Sequences

Use Case Name: View Order Details.
Description: This use case enables shopkeepers to view detailed information about a specific order.
Actors: Shopkeeper.
Goal: Allow shopkeepers to access all necessary information for fulfilling an order.
Pre-Conditions: The shopkeeper must be logged in and the shopkeeper must have received at least one order.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** Shopkeeper selects an order from the "Orders" section. **System Response:** The system retrieves and displays detailed information about the selected order, including items, quantity, and user contact details. 2. **User Action (Step 2):** Shopkeeper reviews the order details for processing. **System Response:** The system provides a button to return to the list of orders.
Alternate Flow (Order Not Found): **User Action**: The shopkeeper selects an order that no longer exists. **System Response**: The system displays an error: "Order not found."
Exception Flow (System Errors) **System Failure:** The system fails to fetch detailed order data. **System Response:** The system displays an error message and asks the shopkeeper to try again later.
Post Conditions: The shopkeeper successfully views detailed information for the selected order.

Table 54 View Order Details (S/RS)

4.28.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall allow shopkeepers to view detailed information about a selected order.	High	Pending
Req 2	The system shall display all order-related data, including item details, quantities, and user contact.	High	Pending

Table 55 View Order Details (FR)

4.29 Cancel an Order

4.29.1 Description and Priority

This use case allows the shopkeeper to cancel an order due to stock unavailability or other issues, updating the order status and notifying the user about the cancellation.

4.29.2 Stimulus/Response Sequences

Use Case Name: Cancel an Order.
Description: This use case allows shopkeepers to cancel an order due to stock unavailability or other issues.
Actors: Shopkeeper.
Goal: Enable shopkeepers to notify users and cancel orders they cannot fulfill.
Pre-Conditions: The shopkeeper must be logged in and the shopkeeper must have received at least one order.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** Shopkeeper selects an order from the "Orders" section. **System Response:** The system retrieves and displays the order details. 2. **User Action (Step 2):** Shopkeeper clicks the "Cancel Order" button. **System Response:** The system updates the order status to "Cancelled" and notifies the user.
Alternate Flow (Order Already Cancelled): **User Action**: The shopkeeper tries to cancel an order that has already been cancelled. **System Response**: The system displays a message: "Order has already been cancelled."
Exception Flow (System Errors) **System Failure:** The system fails to update the order status. **System Response:** The system displays an error message and asks the shopkeeper to try again.
Post Conditions: The order is successfully cancelled, and the user is notified.

Table 56 Cancel an Order (S/RS)

4.29.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall allow shopkeepers to cancel active orders.	High	Pending
Req 2	The system shall update the order status to "Cancelled."	High	Pending
Req 3		High	Pending

Table 57 Cancel an Order (FR)

4.30 Delete User

4.30.1 Description and Priority

This use case allows the admin to delete a user account from the platform due to violations of terms, inactivity, or other administrative reasons.

4.30.2 Stimulus/Response Sequences

Use Case Name: Delete User.
Description: This use case allows the admin to delete a user account from the platform due to violations of terms, inactivity, or other administrative reasons.
Actors: Admin.
Goal: Enable the admin to remove users from the system and maintain platform integrity.
Pre-Conditions: The admin must be logged in, and the user to be deleted must exist in the system.
Basic Course of Events /Main Flow: 1. **User Action (Step 1):** Admin navigates to the "User Management" section. **System Response:** The system displays a list of all registered users. 2. **User Action (Step 2):** Admin selects a specific user and clicks the "Delete User" button. **System Response:** The system displays a confirmation prompt. 3. **User Action (Step 3):** Admin confirms the deletion. **System Response:** The system deletes the user account and all associated data, and displays a success message.
Alternate Flow (User Already Deleted): **User Action**: Admin attempts to delete a user who has already been removed. **System Response**: The system displays a message: "User does not exist or has already been deleted."
Exception Flow (System Errors): **System Failure:** The system encounters an error during the deletion process. **System Response:** The system displays an error message and asks the admin to try again.
Post Conditions: The selected user is removed from the platform, and a success notification is shown to the admin.

Table 58 Delete User (S/RS)

4.30.3 Functional Requirements

Req.ID	Description	Priority	Status
Req 1	The system shall allow the admin to view all registered users.	High	Pending
Req 2	The system shall allow the admin to delete a user from the platform.	High	Pending
Req 3	The system shall remove all associated data of the deleted user.	High	Pending

Table 59 Delete User (FR)

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- **System Load**: The system must be able to handle up to 500 concurrent users without performance degradation.
- **Response Time**: Search queries for car listings and maintenance logs must return results within 2 seconds under normal load.
- **Page Load Time**: The web app's page load time should not exceed 3 seconds for standard operations, such as loading the user dashboard or car listings.
- **Database Queries**: Database queries related to car listings, user profiles, and maintenance logs should execute within 1 second under normal conditions to maintain overall system efficiency.

5.2 Safety Requirements

Regular backups must be conducted to prevent data loss in case of server failure or cyberattacks. The system must safeguard against unauthorized users accessing restricted areas, such as the admin panel, user profiles, or payment systems.

5.3 Security Requirements

User's personal data including payment details must be secured.

5.4 Software Quality Attributes

App Quality Assurance is necessary for any system as it is a set of activities for ensuring quality.

- Usability: The interface is designed to make it user-friendly so it will be easy to for the users to use this app.
- Maintainability: The coding is done in a structured way so in future it will be easy for the developers to make changes or update the code.
- Correctness: The app design should work as expected and contain minimal errors and bugs.

5.5 Business Rules

Protection of copy rights.

6. Other Requirements

N/A

Software Design Specifications

1. Introduction

1.1 Purpose of this document

Software Design Specification (SDS) focuses on specifying a high-level view of architecture and components of our system. SDS also focuses on detailing a low-level view of architecture and components of the system and the way these components interact with each other.

1.2 Scope of the development project

Digital Car-Yard is a web-based platform designed to streamline the buying, selling, and maintenance of vehicles. It allows users to manage car listings, track maintenance tasks, and access a marketplace for car parts and services. The app offers personalized reminders for vehicle upkeep and supports communication between users and service providers. Its goal is to simplify vehicle ownership and provide a comprehensive solution for car-related transactions and maintenance.

1.3 Definitions, acronyms, and abbreviations

N/A.

1.4 References

See page no. 139.

1.5 Overview of document

This document will provide an overview of the system's major components and architecture. In section 2, it will describe low level classes, components, and function. In section 3, it will describe the description of components used in the project. GUI designs will be provided in section 4.

2. System architecture description

2.1 Section Overview

This section describes the constraints and usability of the system. Data design is used to explain the major features of the system. It describes the complete structure of the software and detail description of each individual features. It also describes the architecture design with major components of the software.

2.2 General Constraints

The main thing which is to be kept in mind is to provide flow in the performance of the system. The system will be easy to understand, easy to use, responsive, secure, and the most important thing is it will be so simple so that the user can use it smoothly.

The database of this app is MongoDB, which requires internet connection to fetch and store data.

- Hardware Interfaces

  This app does not have any direct hardware interface. Although for optimal performance and faster results, the user is required to use a high-speed internet.

- Software Interfaces

  The application interacts with the following software systems to deliver its functionalities:

  Web Browser: The system is compatible with modern web browsers, including Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari (latest versions recommended).

  Backend: The backend uses Node.js for handling server-side operations, with data stored and managed using MongoDB, hosted on cloud servers.

  Frontend: The frontend is developed using Flutter or similar web technologies to provide a seamless user interface.

  Other Components: Integration with third-party payment gateway APIs ensures secure and efficient processing of transactions within the marketplace.

2.3 Data Design

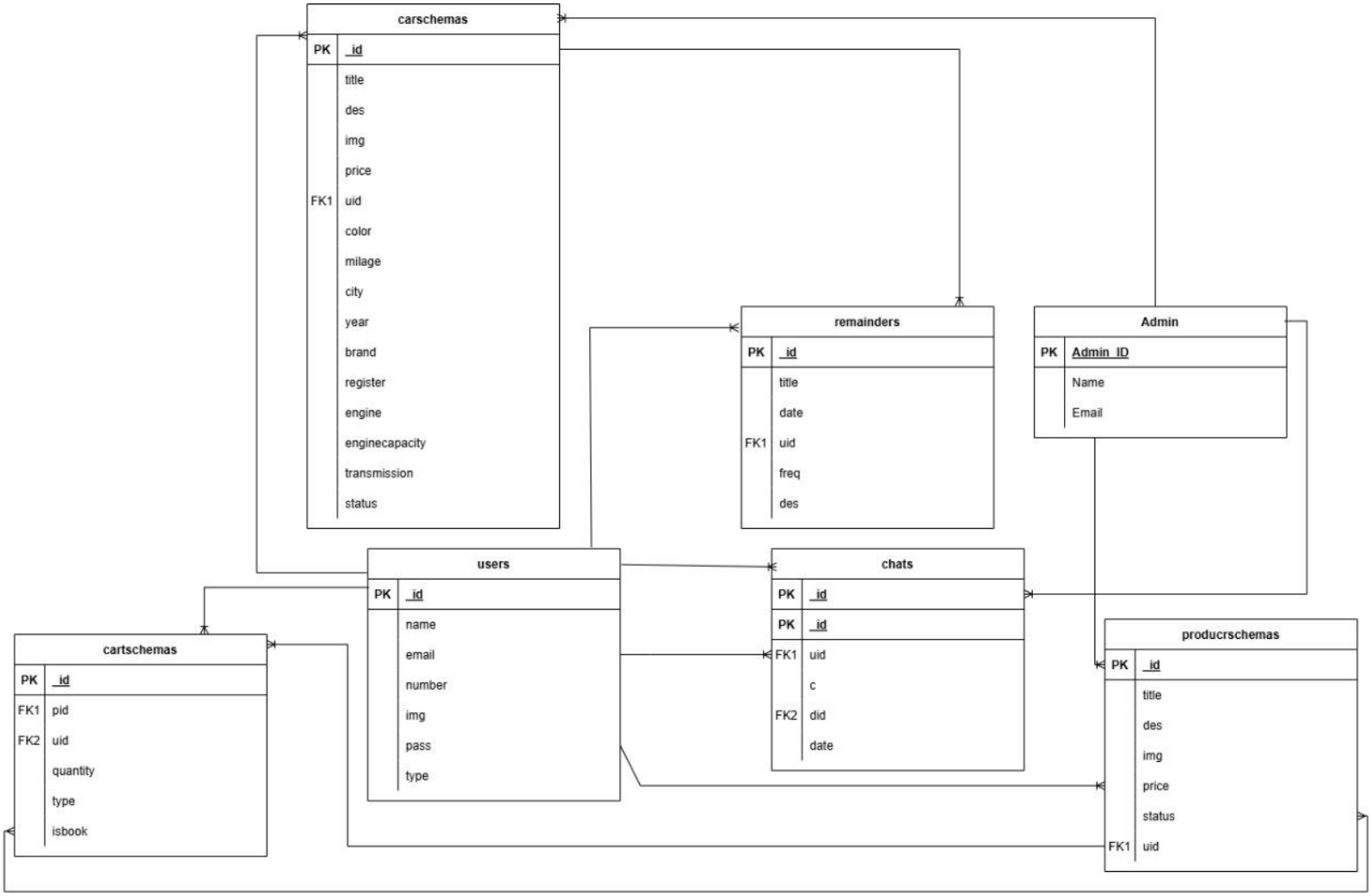

Figure 4 Data Design

2.4 Program Structure

The architectural model chosen for the "Digital Car-Yard" project is a **three-tier architecture**, which separates the application into three main layers: the **frontend**, **backend**, and **database**. This approach ensures scalability, modularity, and maintainability.

1. **Frontend**:
   The frontend layer is responsible for providing the user interface (UI) for the application. Built using **Flutter** or other web technologies, it ensures a responsive and interactive user experience compatible with various devices and platforms, including Windows, macOS, Linux, Android, and iOS.
2. **Backend**:
   The backend serves as the application's core logic and business processing layer. Using **Node.js**, it handles API requests, processes transactions, and communicates with the database. It also integrates with third-party APIs, such as payment gateways, to facilitate secure transactions.
3. **Database**:
   Data is stored in **MongoDB**, a NoSQL database that ensures flexibility and scalability for managing structured and unstructured data, including car listings, user information, and transaction records.
4. **Other Components**:
   - **Web Browsers**: The application is optimized for all modern browsers, including Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari.
   - **Third-Party Integrations**: Includes APIs for payment gateways, enhancing the marketplace's secure transaction functionality.

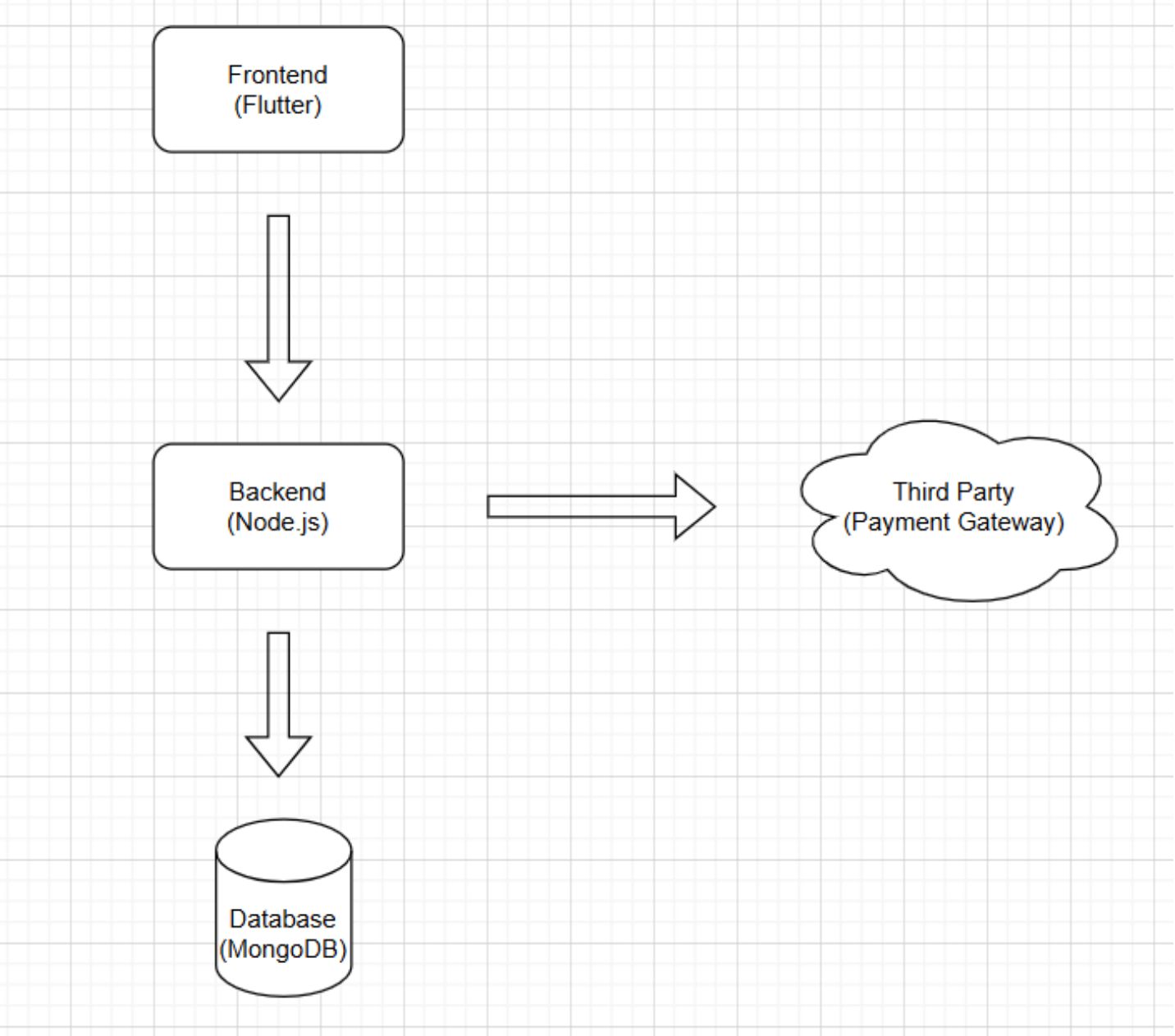

Figure 5 Program Structure

2.5 Alternatives Considered

N/A.

3. Detailed description of components

3.1 Section Overview

This section describes the complete description of all components in detail with their function's dependencies, their relationships with other components. It describes the interface of each component with input and output requirements.

3.2 Component and Detail (include a sub-section for each component)

APIs used in this project are:

- Login
- Signup
- Forget Password
- Save Reminder
- Payment Gateway

The components of this project are:

- Login
- Forgot Password
- Logout
- Chat
- Chat (Admin, Shopkeeper)
- Maintenance Reminders
- Navigate to marketplace
- Search Cars/Car Parts
- View Car/Car Part Details
- Payment Gateway
- List Cars
- View Car Listings
- Delete Car Listings
- Add to Cart
- Delete from cart
- View Cart
- Checkout
- Sign up
- Add Car Part
- Delete Car Part
- View Car Part
- View System Analytics
- Approve Car Listing
- Reject Car Listing
- View Car Listing Requests
- Adjusting Item Quantity in Cart
- View Orders
- View Order Details
- Cancel an Order.

Identification	Login
Type	Component/App Page
Purpose	Login is a component that will be used by every user and admin to access their account.
Function	To log in to our system.
Subordinates	The screen contains links to the following pages: • Registration Page • Forgot Password Page
Dependencies	User needs to have a registered account on this app in order to log in.
Interfaces	A login page consisting of email and password fields.
Resources	Database access requirement. The system will check the email and password in the database and if the user exists, it will log the user into the app.
Processing	After entering the email and password, the system will check for the user in the database. If the user exists, they will be logged in; if not, the system will show an error.
Data	Data type "String" is used for all fields.

Table 60 Login (CT)

Identification	Forgot Password
Type	Component/App Page
Purpose	Allows users to reset their password in case they forget it.
Function	To enable password recovery for registered users.
Subordinates	Contains links to: • Login Page
Dependencies	User must provide a valid registered email.
Interfaces	Input field for email and submit button.
Resources	Database connection for verifying email and sending a reset link.
Processing	The system checks the provided email. If it matches a registered account, a reset link is sent to the email.
Data	Registered email data in the system.

Table 61 Forget password (CT)

Identification	Logout
Type	Component/App Functionality
Purpose	Provides users with the ability to securely exit their account.
Function	Ends the user session.
Subordinates	None.
Dependencies	Requires an active user session.
Interfaces	Logout button.
Resources	Session data storage.
Processing	Clears user session data and redirects to the login page.
Data	Session ID or token.

Table 62 Logout (CT)

Identification	Chat
Type	Component/App Page
Purpose	Enables real-time communication between users.
Function	To allow users to communicate within the app.
Subordinates	Chat history.
Dependencies	Internet connection.
Interfaces	Chat window with input fields and send button.
Resources	Backend server for handling messages.
Processing	User messages are sent to the server and relayed to the recipient in real time.
Data	Messages stored temporarily or persistently.

Table 63 Chat (CT)

Identification	Chat (Admin, Shopkeeper)
Type	Component/App Page
Purpose	Facilitates communication between admin and shopkeepers.
Function	To enable discussions and issue resolutions.
Subordinates	Admin dashboard integration.
Dependencies	Requires valid admin and shopkeeper accounts.
Interfaces	Chat interface with a list of active users.
Resources	Messaging server.
Processing	Messages exchanged through secure communication protocols.
Data	Message history linked to user accounts.

Table 64 Chat (Admin, Shopkeeper) (CT)

Identification	Maintenance Reminders
Type	Component/App Functionality
Purpose	Reminds users of scheduled vehicle maintenance tasks.
Function	Sends notifications for vehicle upkeep.
Subordinates	User profile settings.
Dependencies	Requires user-configured reminder settings.
Interfaces	Notification interface.
Resources	Backend system for scheduling and notifications.
Processing	The system schedules reminders based on user input and sends alerts.
Data	Maintenance schedule data.

Table 65 Maintenance Reminder (CT)

Identification	Navigate to Marketplace
Type	Component/App Page
Purpose	Allows users to access the car parts marketplace.
Function	Displays available car parts.
Subordinates	Search and filter components.
Dependencies	Database of car parts.
Interfaces	Search bar and category filters.
Resources	API to fetch marketplace data.
Processing	Queries the database and displays results.
Data	Marketplace inventory.

Table 66 Navigate to Marketplace (CT)

Identification	Search Cars/Car Parts
Type	Component/App Functionality
Purpose	Allows users to search for specific cars or parts.
Function	Filters data based on search queries.
Subordinates	Search results page.
Dependencies	Search algorithm.
Interfaces	Input field for keywords.
Resources	Database query engine.
Processing	Processes user input and fetches relevant results.
Data	Search query and result data.

Table 67 Search Cars/ Car Parts (CT)

Identification	View Car/Car Part Details
Type	Component/App Page
Purpose	Displays detailed information about a car or car part.
Function	Provides comprehensive data for informed decisions.
Subordinates	Related products section.
Dependencies	Database with detailed product information.
Interfaces	Details page with images and specifications.
Resources	API to fetch product details.
Processing	Fetches and displays data from the database.
Data	Product details.

Table 68 View Car/Car part details (CT)

Identification	Payment Gateway
Type	Component/App Functionality
Purpose	Facilitates secure online transactions.
Function	Processes user payments.
Subordinates	Checkout page.
Dependencies	Integration with third-party payment providers.
Interfaces	Payment form with fields for card details.
Resources	Secure API for payment processing.
Processing	Encrypts payment details and processes transactions.
Data	Payment information.

Table 69 Payment Gateway (CT)

Identification	List Cars
Type	Component/App Page
Purpose	Enables users to add their cars for sale.
Function	Adds car listings to the marketplace.
Subordinates	Add Car Details Form.
Dependencies	User must have an account.
Interfaces	Form fields for car information.
Resources	Database to store car listings.
Processing	Validates input and stores the car details.
Data	Car listing details.

Table 70 List Cars (CT)

Identification	View Car Listings
Type	Component/App Page
Purpose	Displays all car listings to users.
Function	Shows available cars for purchase.
Subordinates	Filters and sorting options.
Dependencies	Database with car listing data.
Interfaces	List view of cars with images and prices.
Resources	API to fetch listings.
Processing	Queries database and displays results.
Data	Car listing data.

Table 71 View Car Listings (CT)

Identification	Delete Car Listings
Type	Component/App Functionality
Purpose	Allows users to remove their car listings.
Function	Deletes selected car listings.
Subordinates	Confirmation prompt.
Dependencies	User account validation.
Interfaces	Delete button on listing page.
Resources	Backend API for deletion.
Processing	Verifies user permissions and deletes the listing.
Data	Car ID and user details.

Table 72 Delete Car Listing (CT)

Identification	Add to Cart
Type	Component/App Functionality
Purpose	Allows users to add items to their shopping cart.
Function	Adds selected products to the cart.
Subordinates	Cart summary page.
Dependencies	User session.
Interfaces	Add to cart button.
Resources	Backend storage for cart items.
Processing	Updates cart data with selected items.
Data	Product ID and quantity.

Table 73 Add to Cart (CT)

Identification	Delete from Cart
Type	Component/App Functionality
Purpose	Enables users to remove items from their cart.
Function	Deletes selected items from the cart.
Subordinates	Cart summary page.
Dependencies	User session.
Interfaces	Delete button on cart items.
Resources	Backend storage for cart updates.
Processing	Removes specified items from the cart.
Data	Product ID.

Table 74 Delete from Cart (CT)

Identification	View Cart
Type	Component/App Page
Purpose	Displays all items added to the cart.
Function	Shows cart contents.
Subordinates	Checkout button.
Dependencies	User session.
Interfaces	Cart page with item details.
Resources	Backend API for fetching cart data.
Processing	Queries database for cart items and displays them.
Data	Cart item details.

Table 75 View Cart (CT)

Identification	Checkout
Type	Component/App Functionality
Purpose	Enables users to complete their purchase.
Function	Processes order and payment.
Subordinates	Order summary page.
Dependencies	Payment gateway.
Interfaces	Checkout form.
Resources	Secure API for payment and order processing.
Processing	Validates order details and processes the transaction.
Data	Order and payment information.

Table 76 Checkout (CT)

Identification	Sign Up
Type	Component/App Page
Purpose	Allows new users to create an account.
Function	Registers user details in the system.
Subordinates	None.
Dependencies	Valid user information.
Interfaces	Registration form.
Resources	Database for storing user data.
Processing	Validates input and creates a new user record.
Data	User details.

Table 77 Sign up (CT)

Identification	Add Car Part
Type	Component/App Functionality
Purpose	Enables shopkeepers to add new car parts to the marketplace.
Function	Adds car part details to the system.
Subordinates	Add Car Part form.
Dependencies	Valid shopkeeper account.
Interfaces	Form fields for part information.
Resources	Database to store car part details.
Processing	Validates input and stores car part details in the database.
Data	Car part details.

Table 78 Add Car Part (CT)

Identification	Delete Car Part
Type	Component/App Functionality
Purpose	Allows shopkeepers to remove car parts from the marketplace.
Function	Deletes car parts from the system.
Subordinates	Confirmation prompt.
Dependencies	Valid shopkeeper account.
Interfaces	Delete button on the car part page.
Resources	Backend API for deletion.
Processing	Verifies shopkeeper permissions and deletes car part details from the database.
Data	Car part ID.

Table 79 Delete Car Part (CT)

Identification	View Car Parts
Type	Component/App Page
Purpose	Displays a list of all car parts listed by the shopkeeper.
Function	Allows shopkeepers to view and manage their car part listings.
Subordinates	Sorting and filtering options.
Dependencies	Database with shopkeeper-specific car part details.
Interfaces	Car parts listing page with management options.
Resources	API to fetch and manage car part listings.
Processing	Queries database to fetch car parts linked to the shopkeeper's account and displays them.
Data	Car part listing details.

Table 80 View Car Part (CT)

Identification	View System Analytics
Type	Component/App Functionality
Purpose	Provides admins with data insights and performance metrics.
Function	Displays system analytics.
Subordinates	Dashboard charts and graphs.
Dependencies	Database of system metrics.
Interfaces	Analytics dashboard.
Resources	Backend API for fetching analytics data.
Processing	Queries system metrics and displays charts/graphs.
Data	System performance data.

Table 81 View System Analytics (CT)

Identification	Approve Car Listing
Type	Component/App Functionality
Purpose	Allows admins to approve new car listings.
Function	Marks car listings as approved.
Subordinates	Approval dashboard.
Dependencies	Valid admin account.
Interfaces	Approve button on listing page.
Resources	Backend API for updating listing status.
Processing	Updates car listing status to approved in the database.
Data	Car listing details.

Table 82 Approve Car Listing (CT)

Identification	Reject Car Listing
Type	Component/App Functionality
Purpose	Allows admins to reject car listings.
Function	Marks car listings as rejected.
Subordinates	Rejection reason prompt.
Dependencies	Valid admin account.
Interfaces	Reject button on listing page.
Resources	Backend API for updating listing status.
Processing	Updates car listing status to rejected in the database.
Data	Car listing details.

Table 83 Reject Car Listing (CT)

Identification	View Car Listing Requests
Type	Component/App Page
Purpose	Allows admins to review new car listing submissions.
Function	Displays pending car listing requests.
Subordinates	Approval and rejection options.
Dependencies	Valid admin account.
Interfaces	Request list page.
Resources	Backend API for fetching requests.
Processing	Queries database for pending car listings and displays them.
Data	Car listing request details.

Table 84 View Car Listing Requests (CT)

Identification	Adjusting Item Quantity in Cart
Type	Component/App Functionality
Purpose	Enables users to update the quantity of items in their cart.
Function	Modifies item quantities in the cart.
Subordinates	Cart summary page.
Dependencies	User session.
Interfaces	Quantity input field and update button.
Resources	Backend storage for cart updates.
Processing	Updates the cart with new quantities in the database.
Data	Product ID and updated quantity.

Table 85 Adjusting Item Quantity (CT)

Identification	View Orders
Type	Component/App Page
Purpose	Allows shopkeepers to see all the orders they have received.
Function	Displays a list of all received orders.
Subordinates	Order details page.
Dependencies	Valid shopkeeper account.
Interfaces	Received orders page.
Resources	API to fetch received orders.
Processing	Queries database and displays received orders for the logged-in shopkeeper.
Data	Order details including product, buyer information, and status.

Table 86 View Orders (CT)

Identification	View Order Details
Type	Component/App Page
Purpose	Allows shopkeepers to view detailed information about a specific order they have received.
Function	Provides order-specific data for shopkeepers.
Subordinates	None.
Dependencies	Valid shopkeeper account.
Interfaces	Order details page.
Resources	API for fetching order details.
Processing	Fetches and displays details of the selected order received by the shopkeeper.
Data	Order details including buyer information, order items, and status.

Table 87 View Order Details (CT)

Identification	Cancel an Order
Type	Component/App Functionality
Purpose	Allows shopkeepers to cancel orders they have received if necessary.
Function	Updates the order status to canceled.
Subordinates	Confirmation prompt.
Dependencies	Valid shopkeeper account.
Interfaces	Cancel button on the order details page.
Resources	Backend API for order status update.
Processing	Verifies shopkeeper permissions and updates the order status to canceled in the database.
Data	Order ID and shopkeeper details.

Table 88 Cancel an Order (CT)

Identification	Delete a User
Type	Component/App Functionality
Purpose	Allows the admin to remove a user from the platform due to violations, inactivity, or other valid reasons.
Function	Removes the user's data from the system and updates the platform database.
Subordinates	Confirmation prompt for deletion.
Dependencies	Valid admin account; the user must exist in the database.
Interfaces	Delete button on the user management panel.
Resources	Backend API for user deletion.
Processing	Verifies admin permissions, checks if the user exists, and deletes the user record from the database.
Data	User ID, user details, and deletion timestamp.

Table 89 Delete a User (CT)

4. User Interface Design

4.1 Section Overview

This section provides a view into the set of standards which we have made use of to build a user-friendly Digital Car-Yard Web App. They have been applied in the best possible way to make the interface appear pleasing to the user and not seem too difficult to understand at the same time.

4.2 Interface Design Rules

Users should always be informed of system operations with easy to understand and highly visible status displayed on the screen within a reasonable amount of time. Interface designers should ensure that both the graphic elements and terminology are maintained across similar platforms. For example, an icon that represents one category or concept should not represent a different concept when used on a different screen. Designers should assume that users are unable to understand technical terminology, therefore, error messages should almost always be expressed in plain language to ensure nothing gets lost in translation.

4.3 GUI Components

- db.js
- car.modal.js
- chat.modal.js
- remainder.modal.js
- user.model.js
- car.route.js
- chat.route.js
- remainder.route.js
- user.route.js
- app.js
- index.js
- package.json
- package-lock.json

4.4 Detailed Description

- **db.js:** It consists of database configurations and connections. It handles the setup and initialization of the MongoDB database, ensuring smooth communication between the application and the database. It is used for performing CRUD operations and maintaining the integrity of data throughout the application.
- **car.modal.js:** It defines the schema for car data, including attributes like model, brand, and price, and manages database operations like adding, or retrieving car details.
- **chat.modal.js:** It defines the schema for chat data, including attributes like sender, receiver, message content, and timestamps, and handles database operations for chat functionality.
- **remainder.modal.js:** It defines the schema for reminders, including attributes like title, date, and time, and manages database operations for scheduling and tracking reminders.

- **user.model.js:** It defines the schema for user data, including attributes like name, email, password, and manages database operations for user authentication and profiles.
- **car.route.js:** It defines API endpoints for car-related operations, such as adding retrieving, or deleting car details, and routes requests to the appropriate controller functions.
- **chat.route.js:** It defines API endpoints for chat functionality, handling operations like sending, receiving, and retrieving messages, and routes requests to the corresponding chat controller.
- **remainder.route.js:** It defines API endpoints for reminder functionality, handling operations like creating, updating, retrieving, or deleting reminders, and routes requests to the relevant controller.
- **user.route.js:** It defines API endpoints for user-related operations, such as registration, login, profile management, and user data retrieval, routing requests to the appropriate controllers.
- **app.js:** It serves as the main entry point of the application, where the server is initialized, middleware is configured, and routes are connected. It also handles error management and application-wide settings.
- **index.js:** It serves as the main entry point for the application, initializing the server, importing necessary modules, and setting up routes and middleware for the app's functionality.
- **package.json:** It contains main data about the project, including the project's name, version, dependencies, and scripts. It is used to manage and install the necessary packages and libraries for the application.
- **package-lock.json:** It records the exact versions of dependencies and their sub-dependencies installed in the project. It ensures consistent and reproducible installations across different environments by locking the dependency tree.

5. Reuse and relationships with other products

Reusability allows developers to be more efficient because the same code can be developed once and used in many different applications. Second, reliability can be improved by reusing previously developed, and previously tested, components. The development of new code entails additional costs in time and money for testing, validation, and verification. Much of these expenses can be avoided by using already manufactured external components.

6. Design decisions and tradeoffs

N/A.

7. Pseudocode for components

Pseudocode for component Login:

```
Begin
FUNCTION Login(email, password):
   FETCH email FROM Database WHERE user.email == email
   IF user EXISTS AND user.password == password:
      RETURN 'Login Successful'
   ELSE:
      RETURN 'Invalid Email or Password'
End
```

Pseudocode for component Forgot Password:

```
Begin
FUNCTION ForgotPassword(email):
   FETCH user FROM Database WHERE user.email == email
   IF user EXISTS:
      GENERATE reset_link
      SEND reset_link TO email
      RETURN 'Reset Link Sent'
   ELSE:
      RETURN 'Email Not Found'
End
```

Pseudocode for component Logout:

```
Begin
FUNCTION Logout(session_id):
   DELETE session_id FROM ActiveSessions
   RETURN 'Logged Out'
End
```

Pseudocode for component Chat:

```
Begin
FUNCTION SendMessage(sender, receiver, message):
   STORE message IN Database WITH sender, receiver
   NOTIFY receiver
   RETURN 'Message Sent'
End
```

Pseudocode for component Chat (Admin, Shopkeeper):

```
Begin
FUNCTION AdminShopkeeperChat(admin_id, shopkeeper_id, message):
   STORE message IN Database WITH admin_id, shopkeeper_id
   NOTIFY shopkeeper
   RETURN 'Message Sent'
End
```

Pseudocode for component Maintenance Reminders:

```
Begin
FUNCTION SetMaintenanceReminder(user_id, schedule):
   STORE schedule IN Database WITH user_id
   SCHEDULE notification BASED ON schedule
   RETURN 'Reminder Set'
End
```

Pseudocode for component Navigate to Marketplace:

```
Begin
FUNCTION NavigateToMarketplace():
   DISPLAY MarketplacePage
   RETURN 'Marketplace Loaded'
End
```

Pseudocode for component Search Cars/Car Parts:

```
Begin
FUNCTION SearchItems(query):
   IF query IS NOT EMPTY:
      FETCH items FROM Database WHERE query MATCHES item.name PARTIALLY
      IF items ARE FOUND:
         RETURN items
      ELSE:
         RETURN 'No results found'
   ELSE:
      RETURN 'Please enter a valid search term'
End
```

Pseudocode for component View Car/Car Part Details:

```
Begin
FUNCTION ViewItemDetails(item_id):
   FETCH item FROM Database WHERE item.id == item_id
   RETURN item
End
```

Pseudocode for component Payment Gateway:

```
Begin
FUNCTION ProcessPayment(order_id, payment_details):
   VALIDATE payment_details
   IF VALID:
      UPDATE order.status TO 'Paid'
      RETURN 'Payment Successful'
   ELSE:
      RETURN 'Payment Failed'
End
```

Pseudocode for component List Cars:

```
Begin
FUNCTION ListCar(user_id, car_details):
   STORE car_details IN TemporaryDatabase WITH user_id AND status AS 'Pending Approval'
   NOTIFY admin OF new listing request
   RETURN 'Car Listing Request Sent'
End
```

Pseudocode for component View Car Listings:

```
Begin
FUNCTION ViewCarListings():
   FETCH car_listings FROM Database
   RETURN car_listings
End
```

Pseudocode for component Delete Car Listing:

```
Begin
FUNCTION DeleteCarListing(user_id, car_id):
   DELETE car FROM Database WHERE car.id == car_id AND car.user_id == user_id
   RETURN 'Car Listing Deleted'
End
```

Pseudocode for component Add to Cart:

```
Begin
FUNCTION AddToCart(user_id, item_id):
   STORE item_id IN Cart WHERE cart.user_id == user_id
   RETURN 'Item Added to Cart'
End
```

Pseudocode for component Delete from Cart:

```
Begin
FUNCTION DeleteFromCart(user_id, item_id):
   DELETE item_id FROM Cart WHERE cart.user_id == user_id
   RETURN 'Item Removed from Cart'
End
```

Pseudocode for component View Cart:

```
Begin
FUNCTION ViewCart(user_id):
   FETCH cart_items FROM Cart WHERE cart.user_id == user_id
   RETURN cart_items
End
```

Pseudocode for component Checkout:

```
Begin
FUNCTION Checkout(user_id):
   FETCH cart_items FROM Cart WHERE cart.user_id == user_id
   CREATE order WITH cart_items
   RETURN 'Order Created'
End
```

Pseudocode for component Sign up:

```
Begin
FUNCTION SignUp(user_details):
   STORE user_details IN Database
   RETURN 'User Registered'
End
```

Pseudocode for component Add Car Part:

```
Begin
FUNCTION AddCarPart(shopkeeper_id, part_details):
   STORE part_details IN Database WITH shopkeeper_id
   RETURN 'Car Part Added'
End
```

Pseudocode for component Delete Car Part:

```
Begin
FUNCTION DeleteCarPart(shopkeeper_id, part_id):
   DELETE part FROM Database WHERE part.id == part_id AND part.shopkeeper_id == shopkeeper_id
   RETURN 'Car Part Deleted'
End
```

Pseudocode for component View Car Parts:

```
Begin
FUNCTION ViewCarParts(shopkeeper_id):
   FETCH parts FROM Database WHERE parts.shopkeeper_id == shopkeeper_id
   RETURN parts
End
```

Pseudocode for component View System Analytics:

```
Begin
FUNCTION ViewSystemAnalytics(admin_id):
   FETCH analytics_data FROM Database
   RETURN analytics_data
End
```

Pseudocode for component Approve Car Listing:

```
Begin
FUNCTION ApproveCarListing(admin_id, listing_id):
   UPDATE listing.status TO 'Approved' WHERE listing.id == listing_id
   RETURN 'Car Listing Approved'
End
```

Pseudocode for component Reject Car Listing:

```
Begin
FUNCTION RejectCarListing(admin_id, listing_id):
   UPDATE listing.status TO 'Rejected' WHERE listing.id == listing_id
   RETURN 'Car Listing Rejected'
End
```

Pseudocode for component View Car Listing Requests:

```
Begin
FUNCTION ViewCarListingRequests(admin_id):
   FETCH pending_listings FROM Database WHERE listing.status == 'Pending'
   RETURN pending_listings
End
```

Pseudocode for component Adjusting Item Quantity in Cart:

```
Begin
FUNCTION AdjustItemQuantity(user_id, item_id, quantity):
   UPDATE cart.item.quantity TO quantity WHERE cart.user_id == user_id AND cart.item_id
== item_id
   RETURN 'Item Quantity Updated'
End
```

Pseudocode for component View Orders:

```
Begin
FUNCTION ViewOrders(shopkeeper_id):
   FETCH orders FROM Database WHERE orders.shopkeeper_id == shopkeeper_id
   RETURN orders
End
```

Pseudocode for component View Order Details:

```
Begin
FUNCTION ViewOrderDetails(order_id):
   FETCH order FROM Database WHERE order.id == order_id
   RETURN order
End
```

Pseudocode for component Cancel an Order:

```
Begin
FUNCTION CancelOrder(shopkeeper_id, order_id):
   UPDATE order.status TO 'Canceled' WHERE order.id == order_id AND order.shopkeeper_id
== shopkeeper_id
   RETURN 'Order Canceled'
End
```

Pseudocode for component Delete User:

```
Begin
FUNCTION DeleteUser(admin_id, user_id):
   IF VALIDATE Admin(admin_id) THEN
      IF EXISTS(User WHERE user.id == user_id) THEN
         DELETE FROM Users WHERE user.id == user_id
         RETURN 'User Deleted Successfully'
      ELSE
         RETURN 'User Not Found'
      ENDIF
   ELSE
      RETURN 'Unauthorized Action'
   ENDIF
End
```

8. Appendices

8.1 Class Diagram

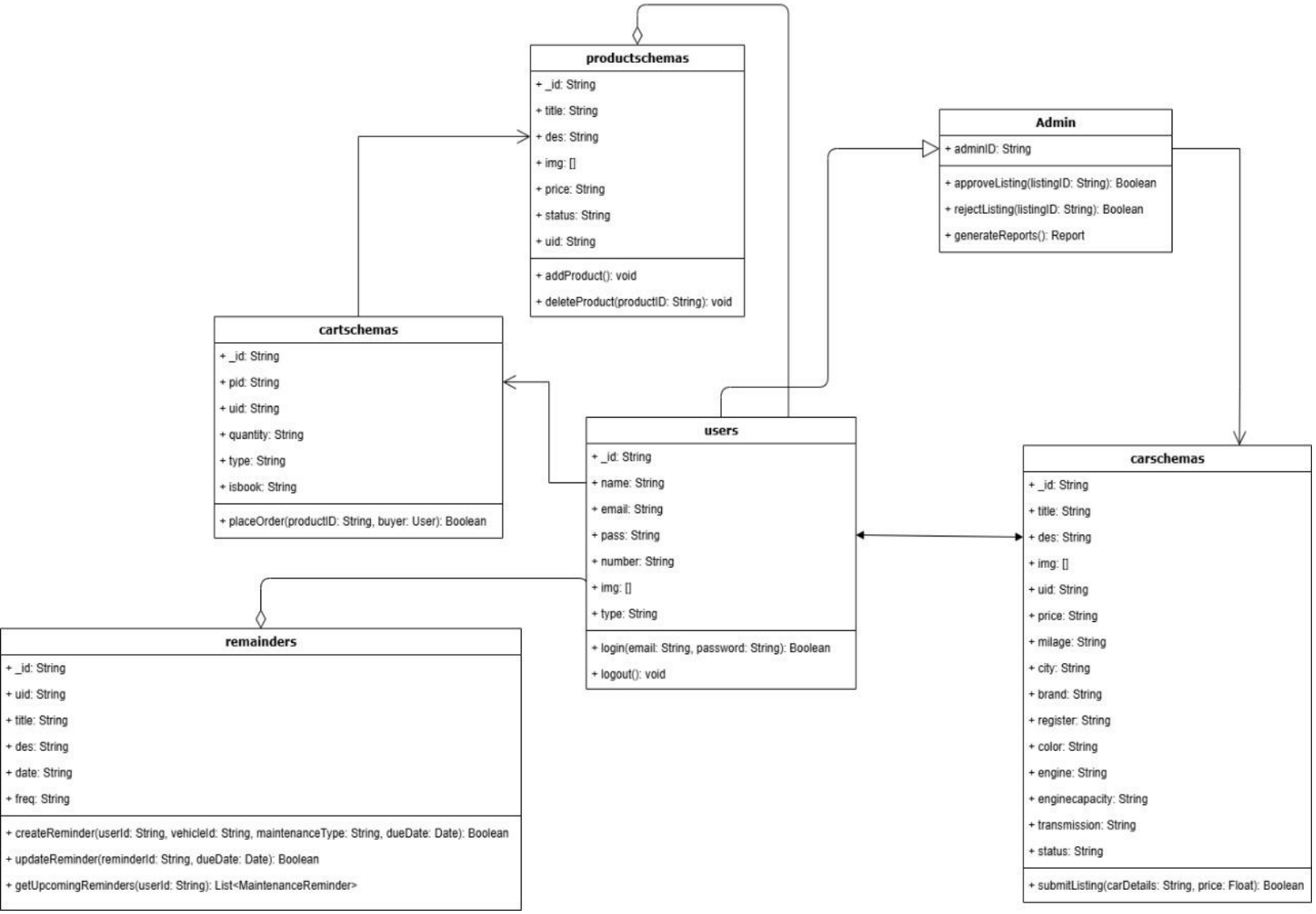

Figure 6 Class Diagram

8.2 Object Diagram

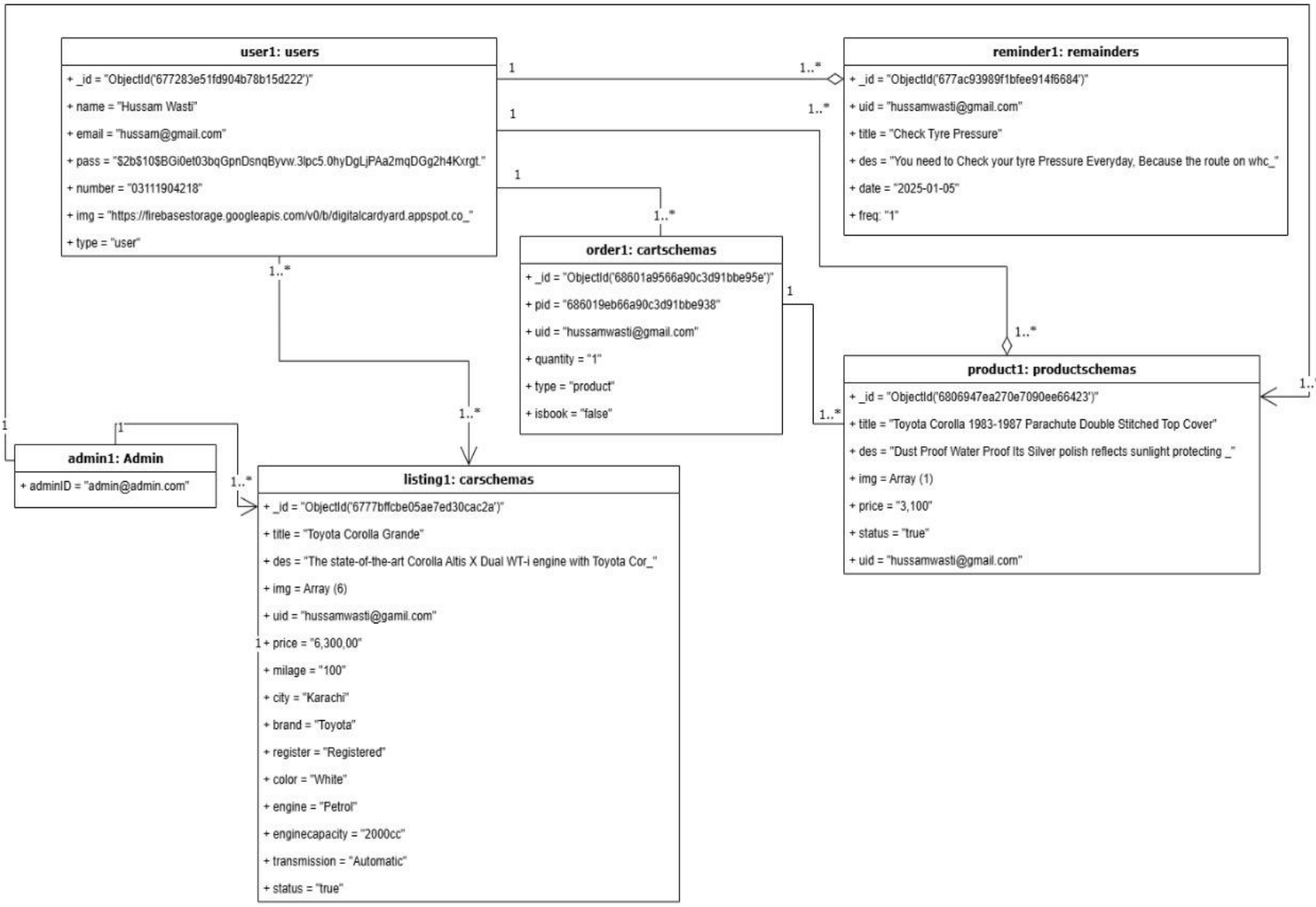

Figure 7 Object Diagram

8.3 State Machine Diagrams

8.3.1 Maintenance Reminders

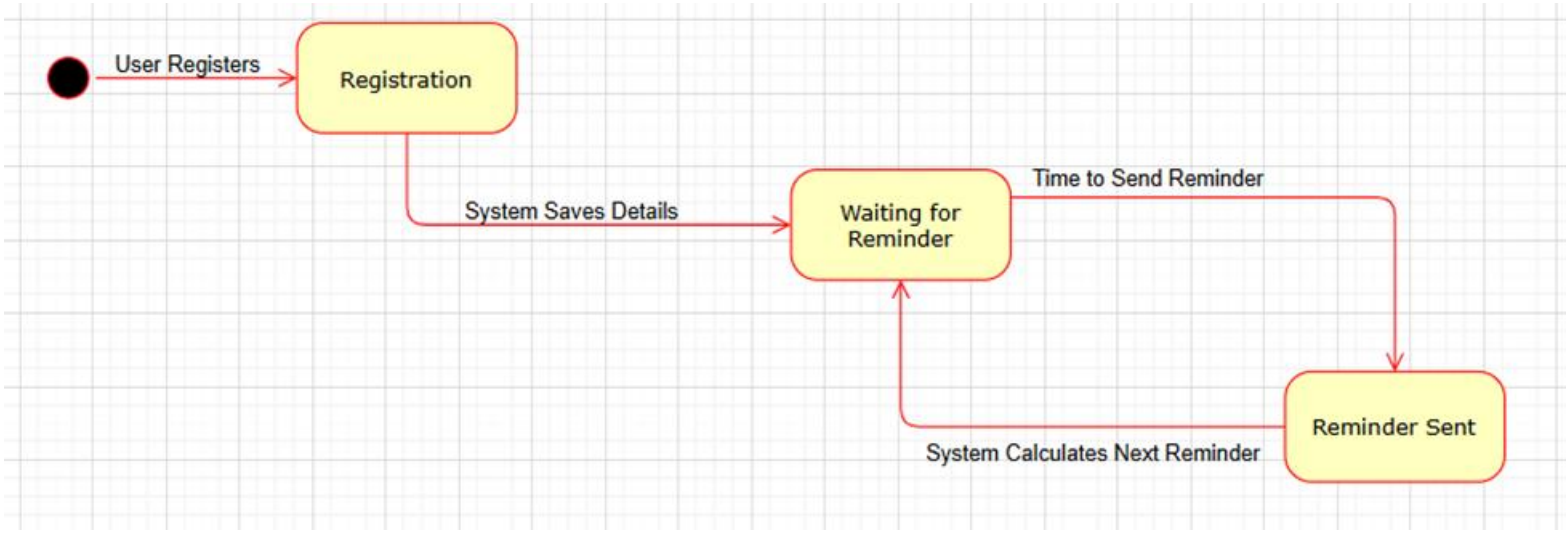

Figure 8 Maintenance Reminders (SMD)

8.3.2 Checkout

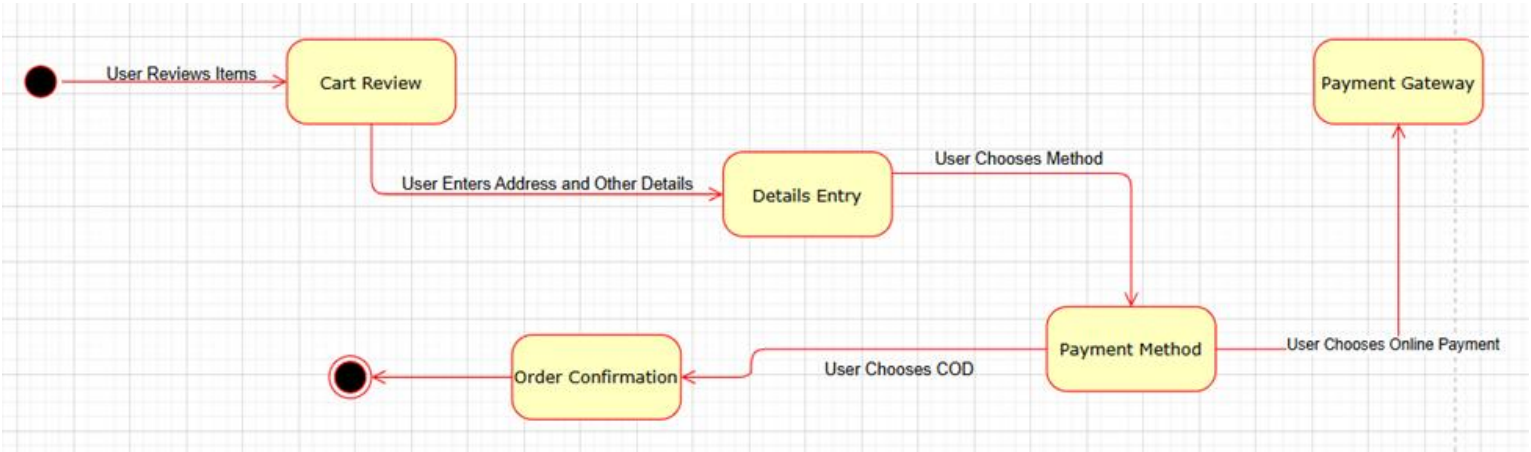

Figure 9 Checkout (SMD)

8.3.3 Payment Gateway

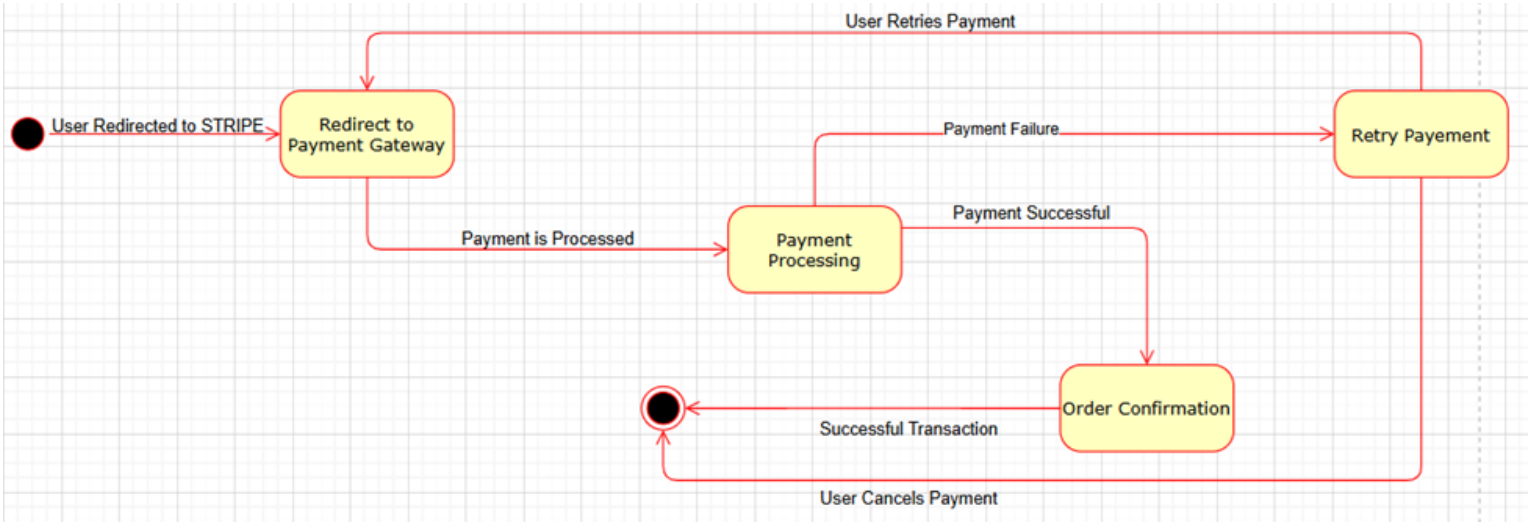

Figure 10 Payment Gateway (SMD)

8.3.4 List Cars

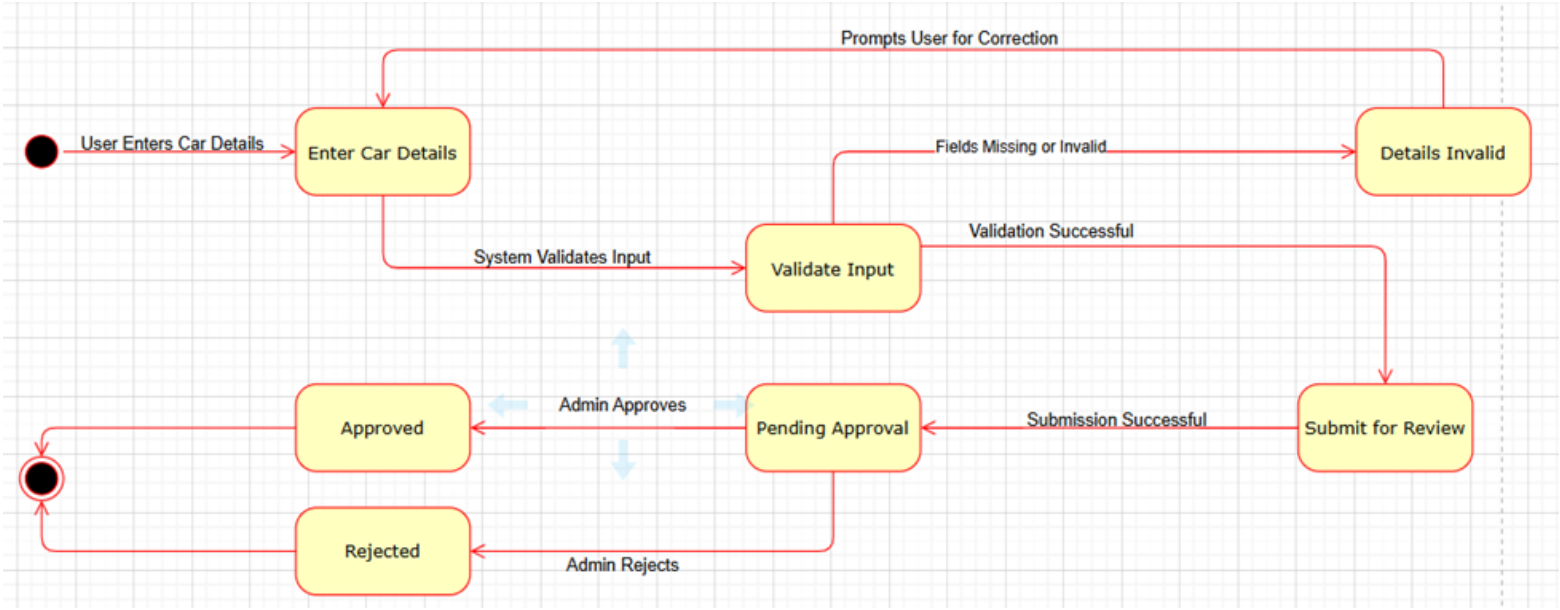

Figure 11 List Cars (SMD)

8.3.5 Approve Car Listing

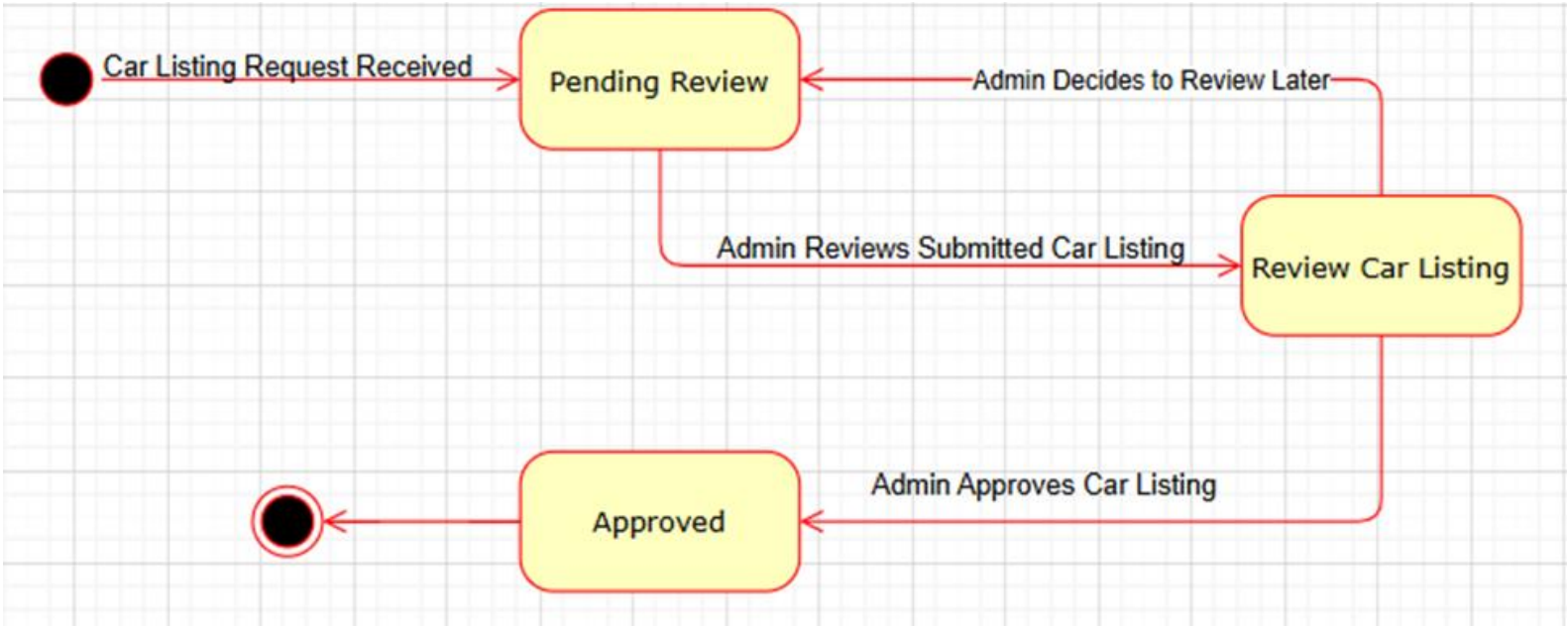

Figure 12 Approve Car Listing (SMD)

8.3.6 Reject Car Listing

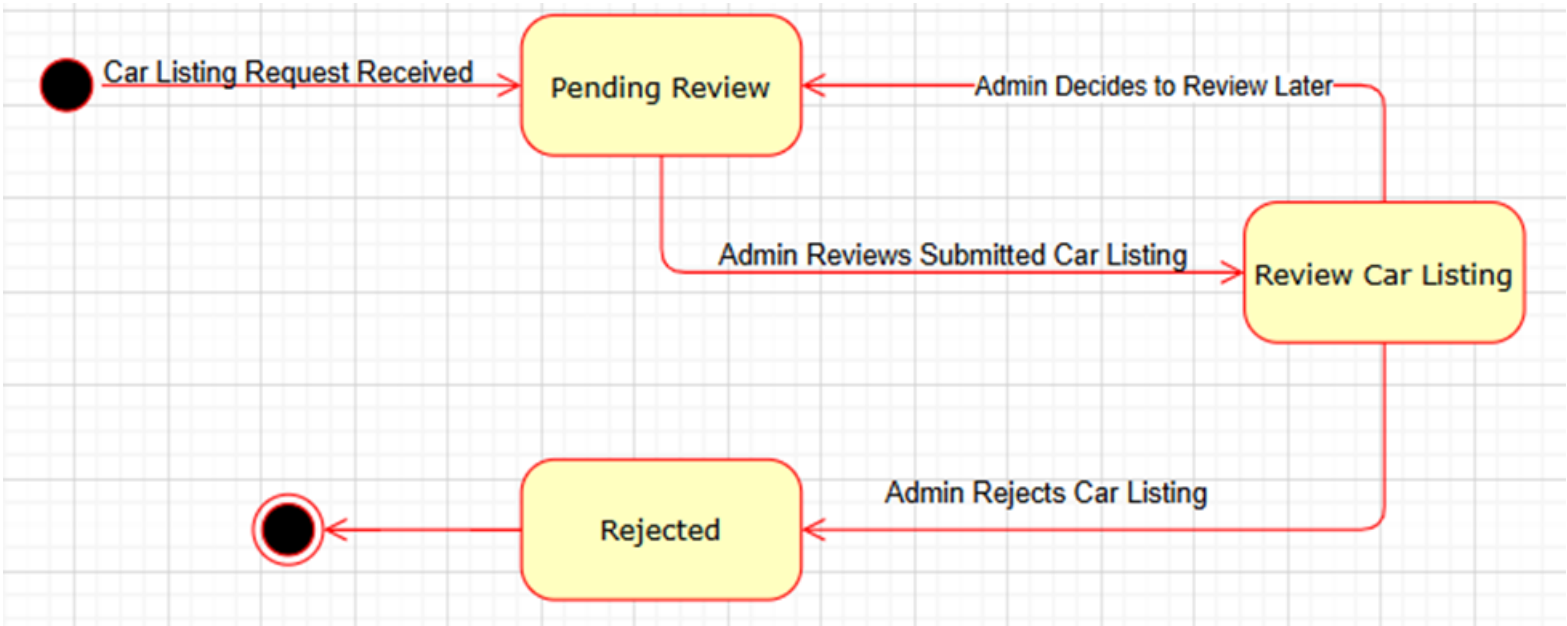

Figure 13 Reject Car Listing (SMD)

8.3.7 Cancel an Order

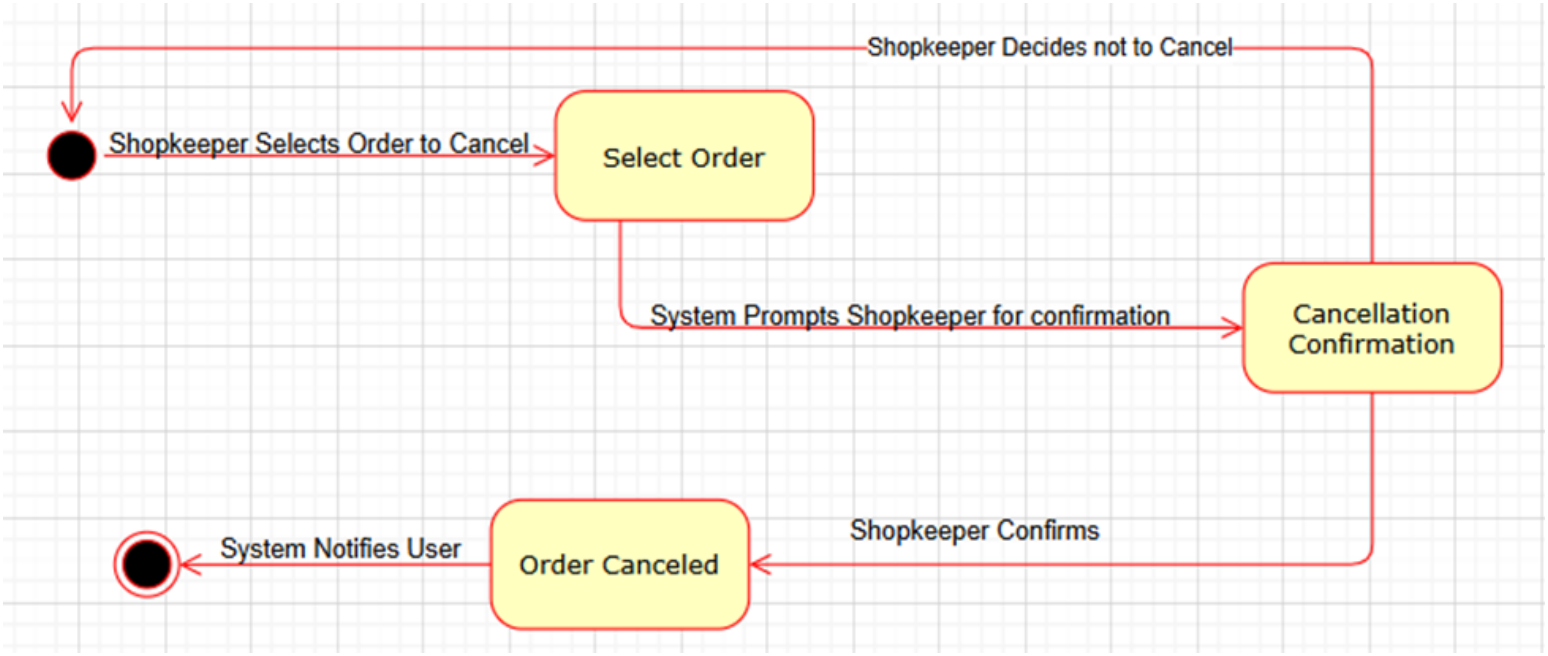

Figure 14 Cancel an Order (SMD)

8.3.8 View Orders

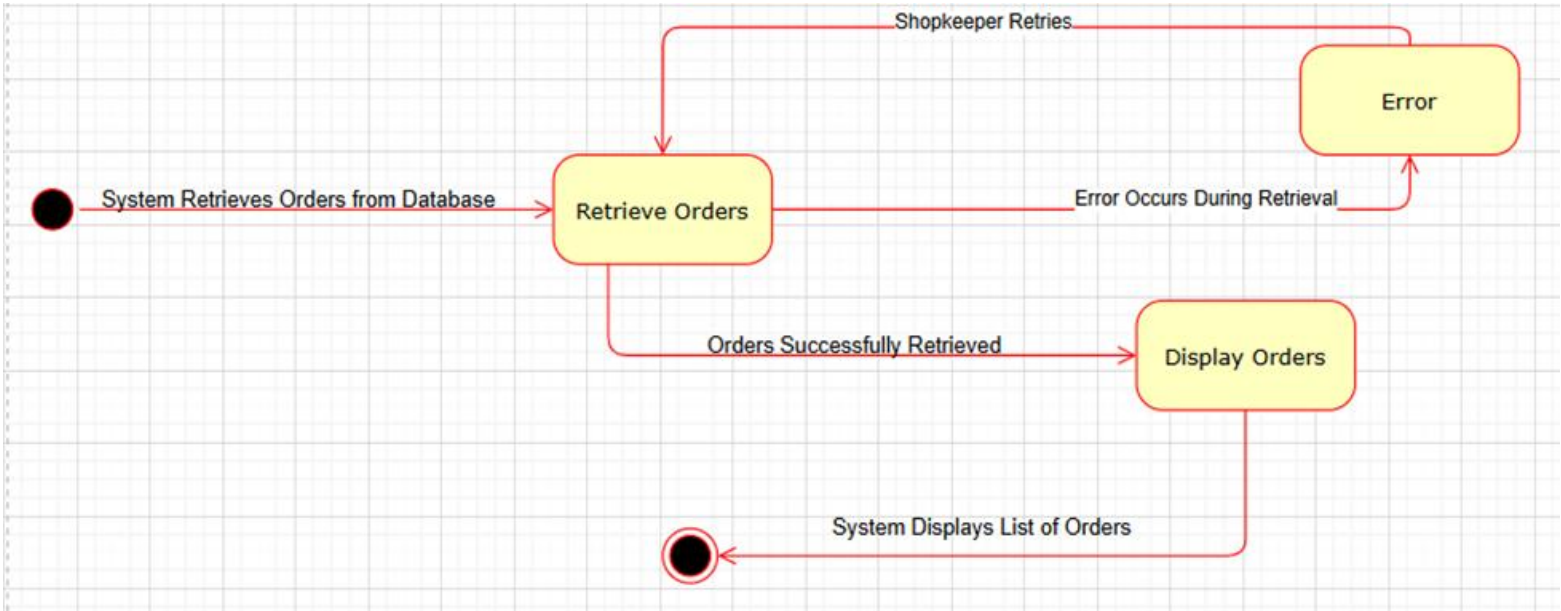

Figure 15 View Orders (SMD)

8.4 Activity Diagrams

8.4.1 Login

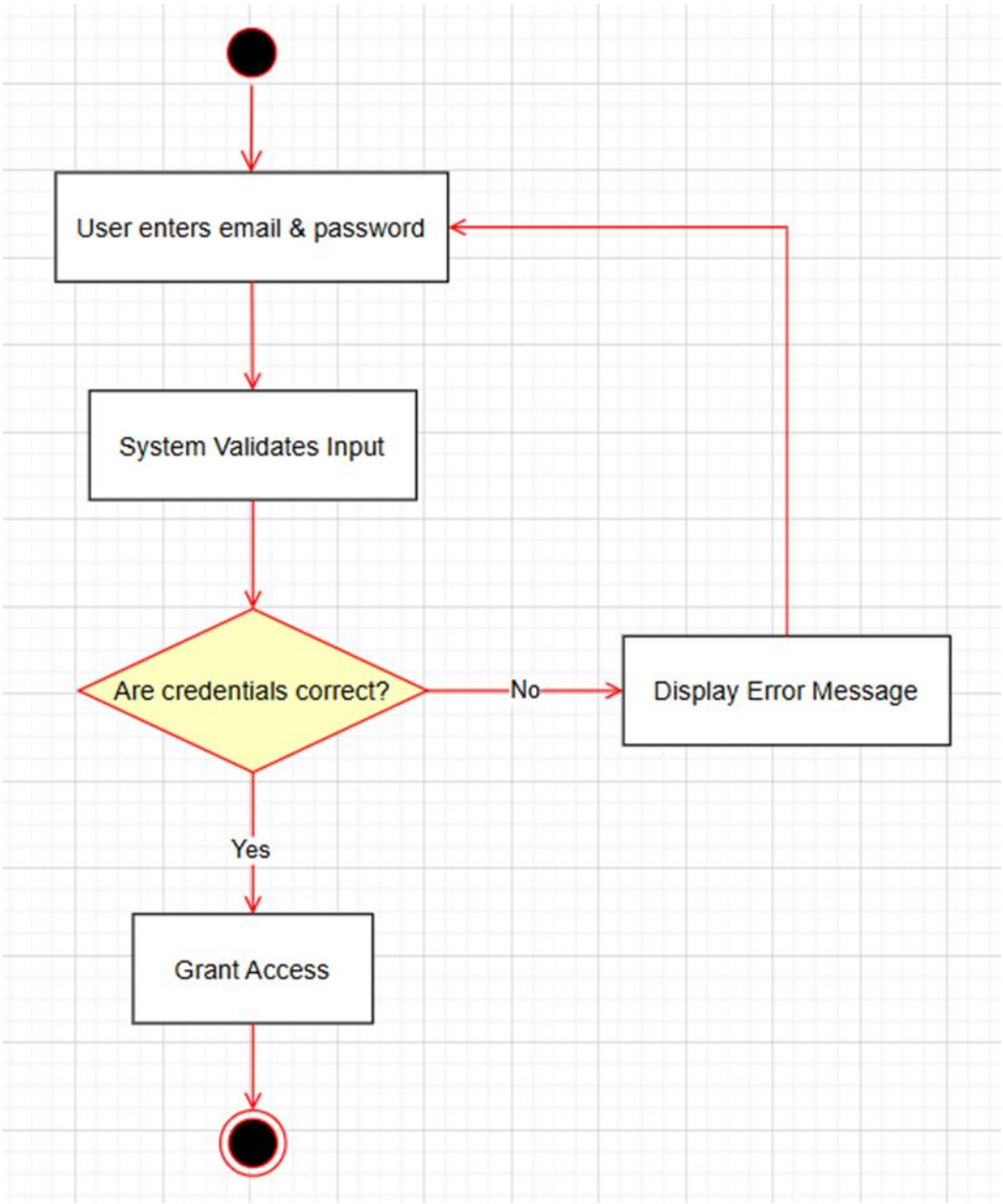

Figure 16 Login (AD)

8.4.2 Signup

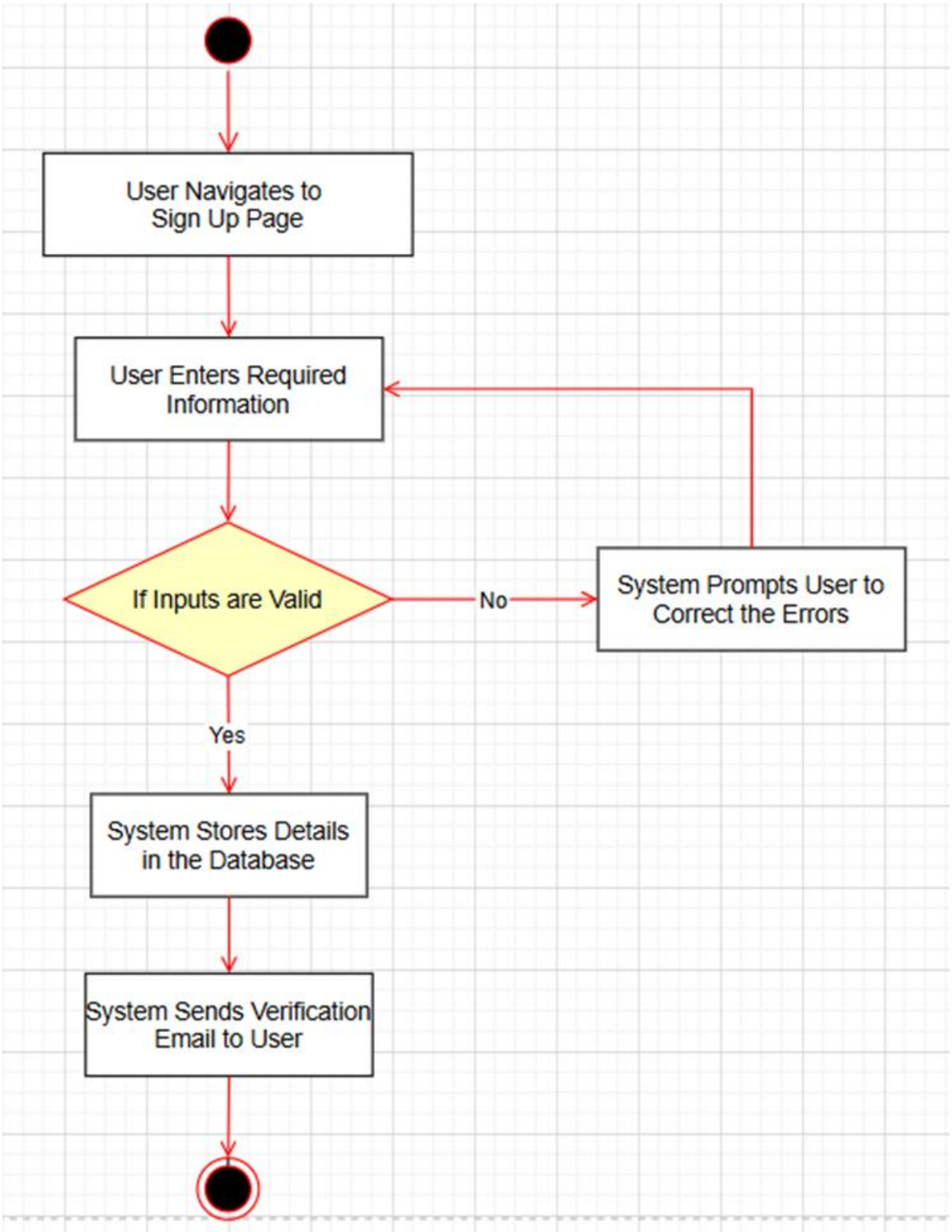

Figure 17 Signup (AD)

8.4.3 Maintenance Reminders

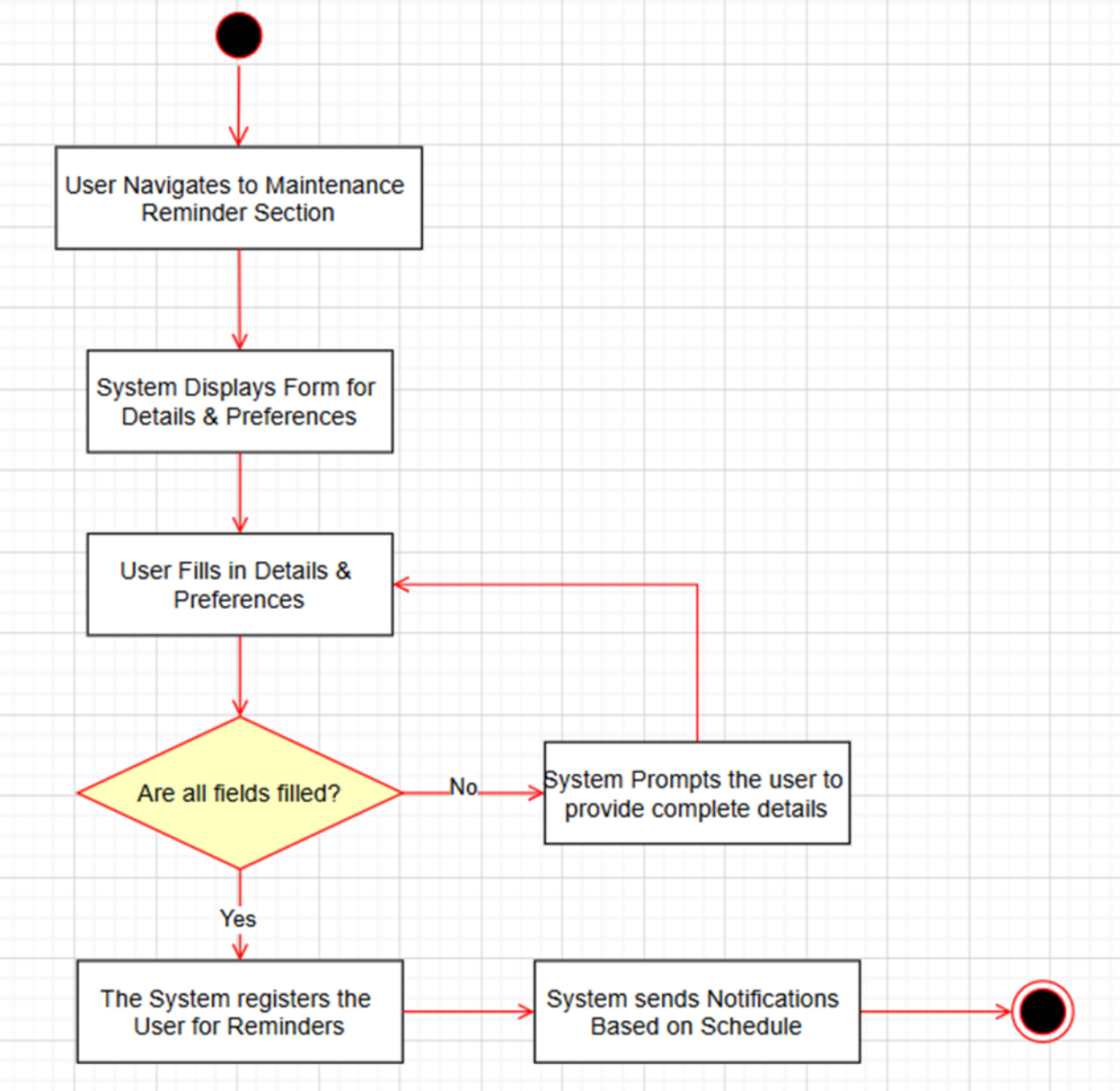

Figure 18 Maintenance Reminders (AD)

8.4.4 Search Cars/Car Parts

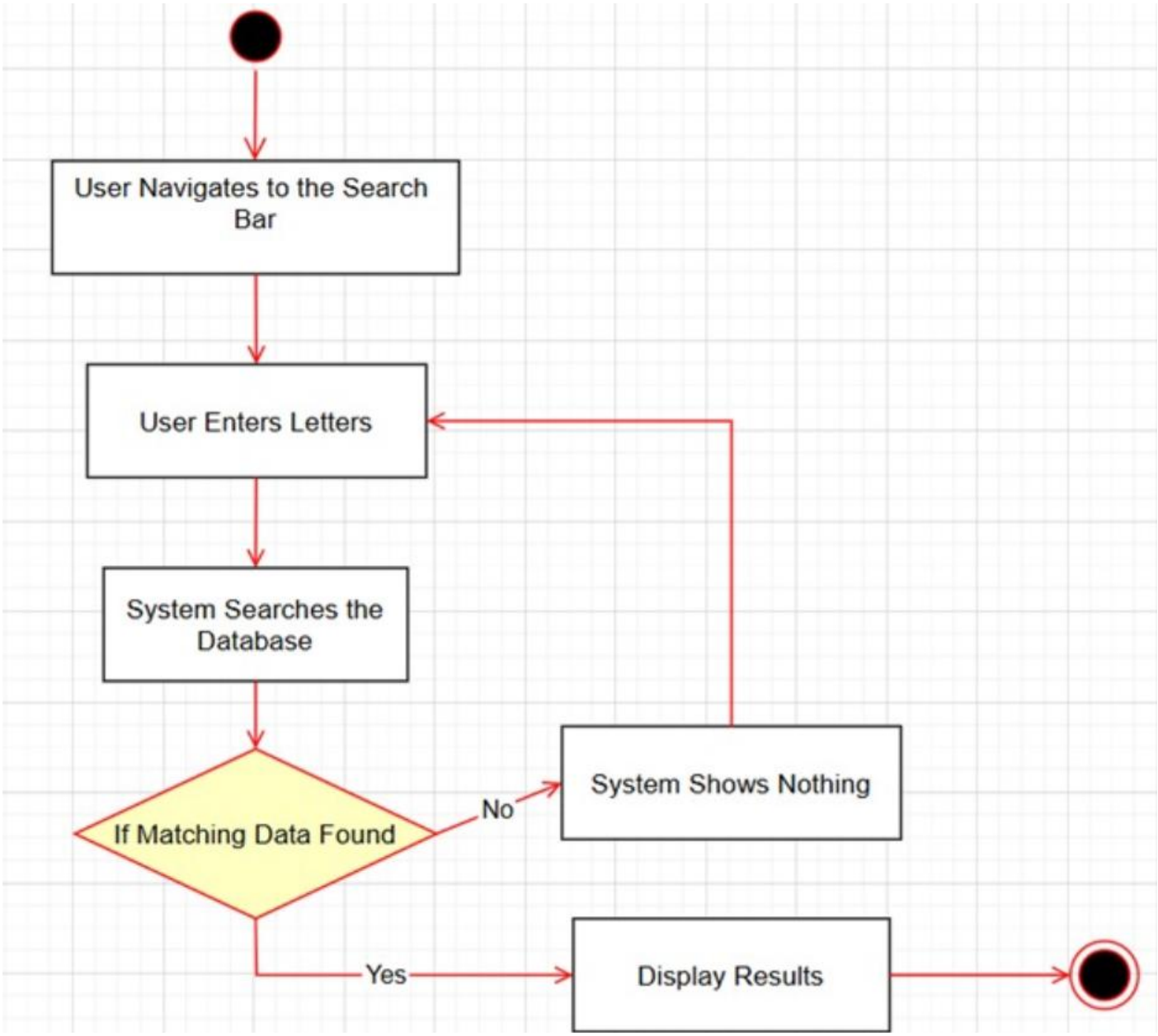

Figure 19 Search Cars/Car Parts (AD)

8.4.5 Checkout

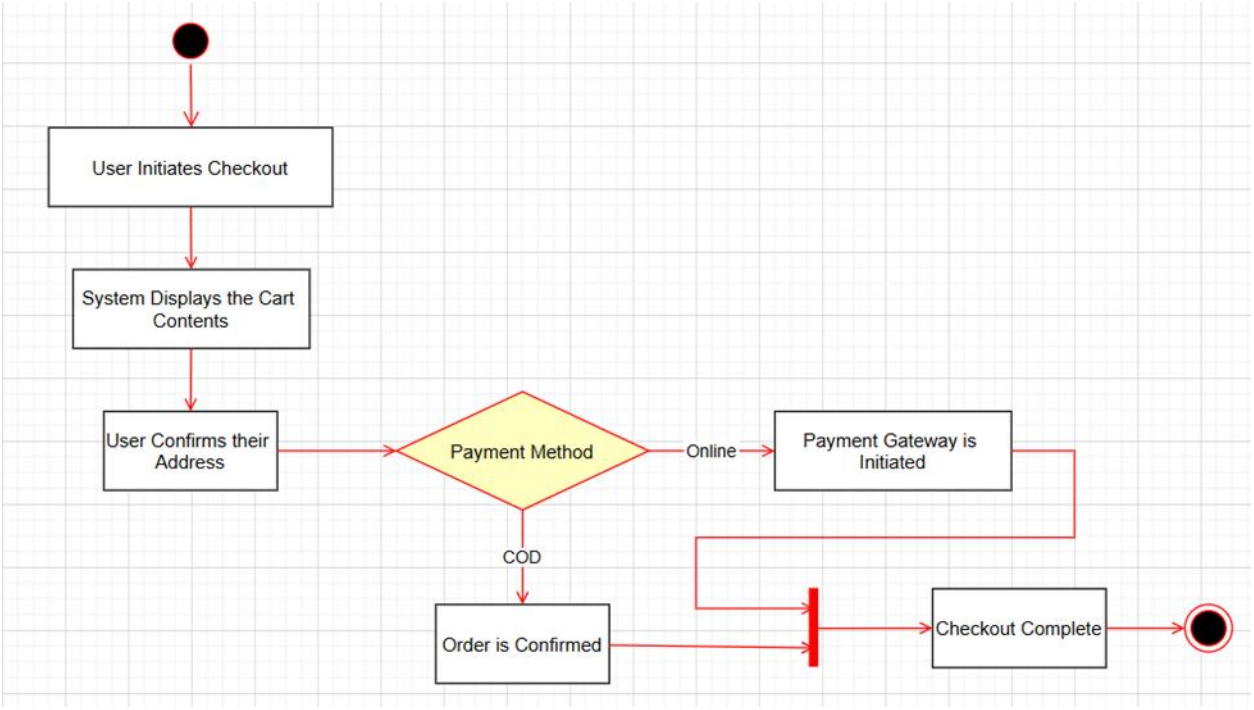

Figure 20 Checkout (AD)

8.4.6 Payment Gateway

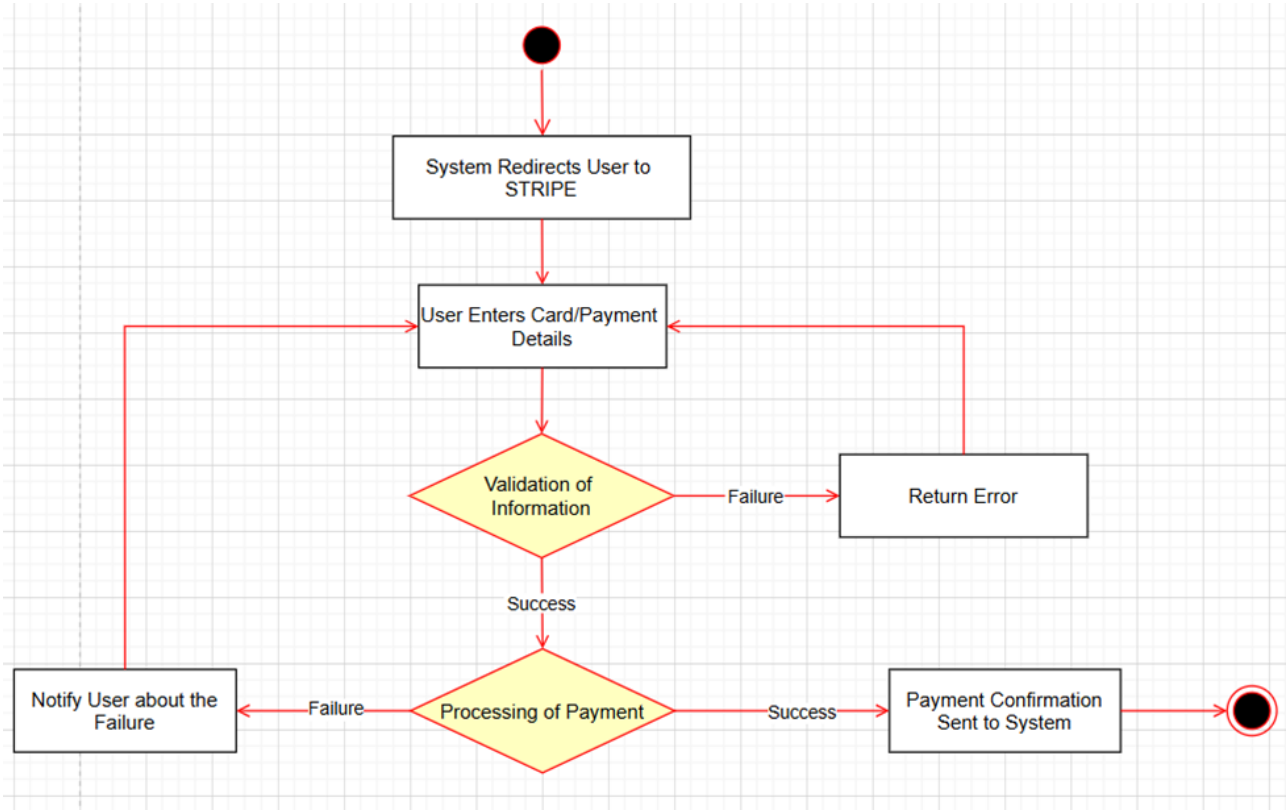

Figure 21 Payment Gateway (AD)

8.4.7 Delete Car Listing

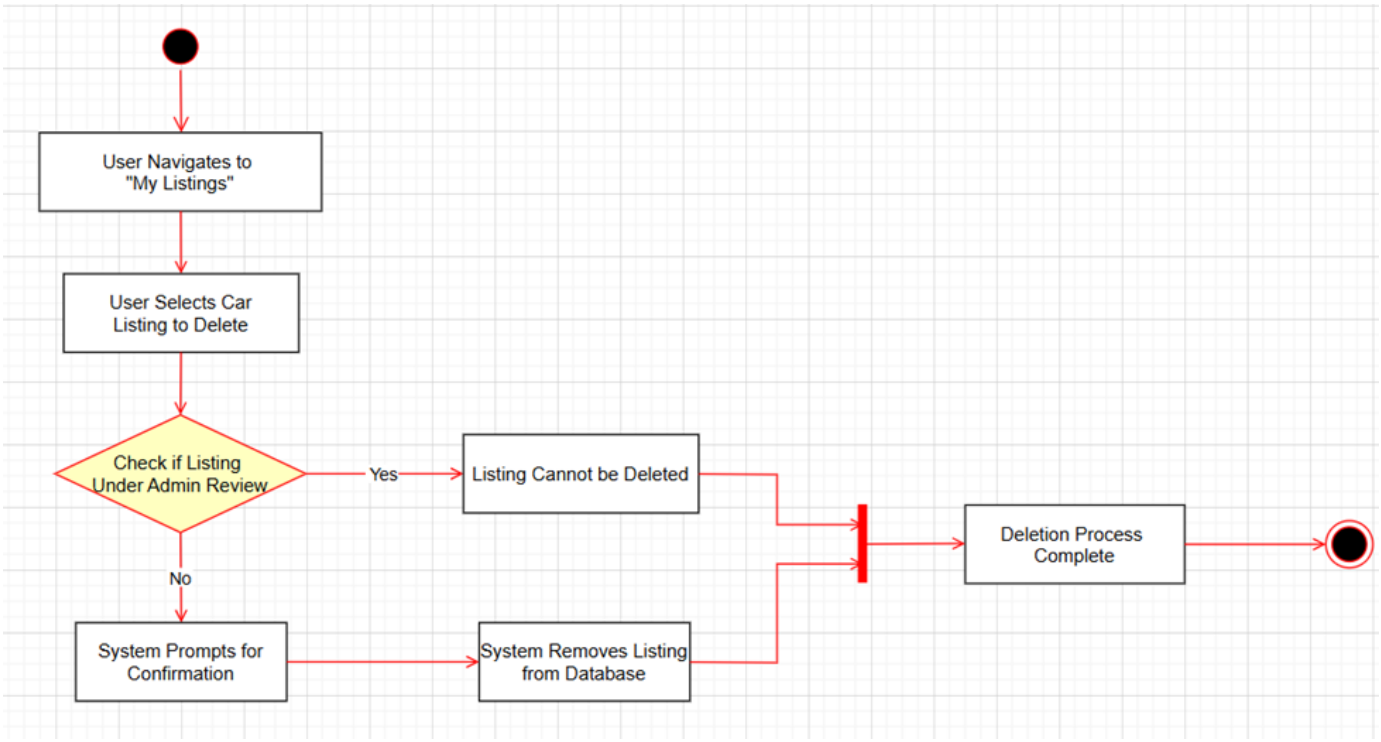

Figure 22 Delete Car Listing (AD)

8.5 Sequence Diagrams

8.5.1 Login

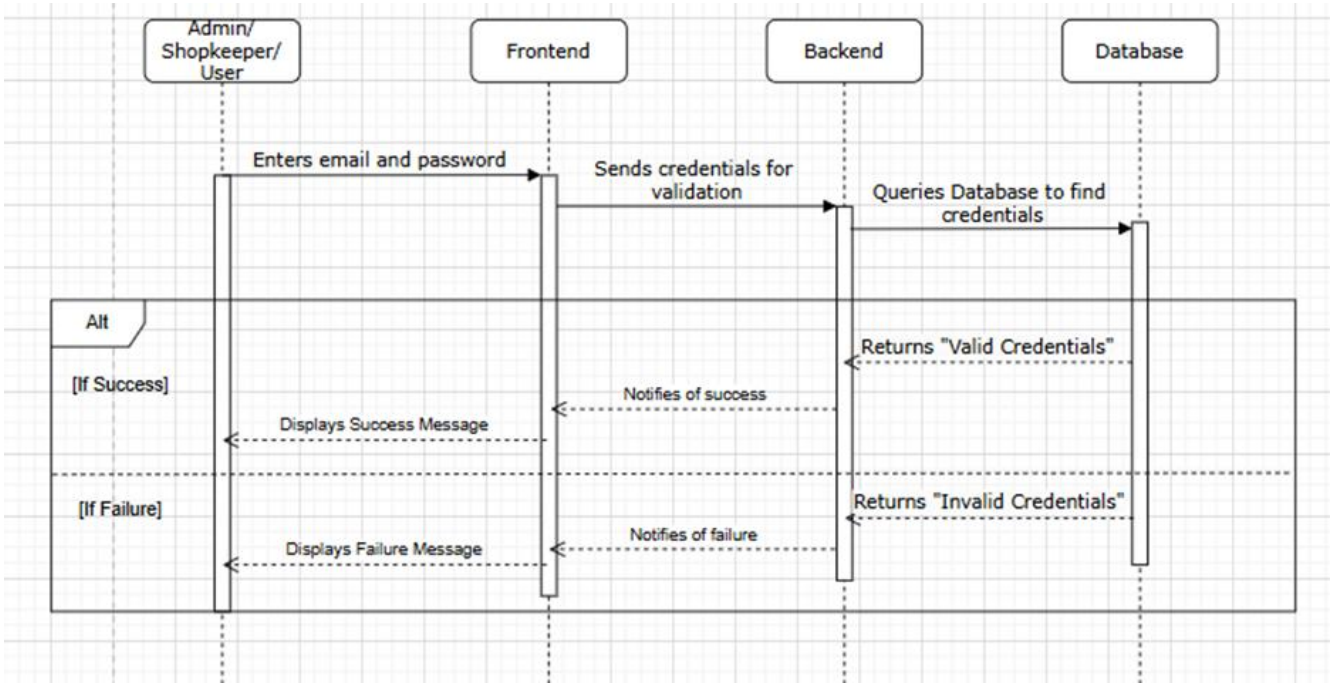

Figure 23 Login (SD)

8.5.2 Signup

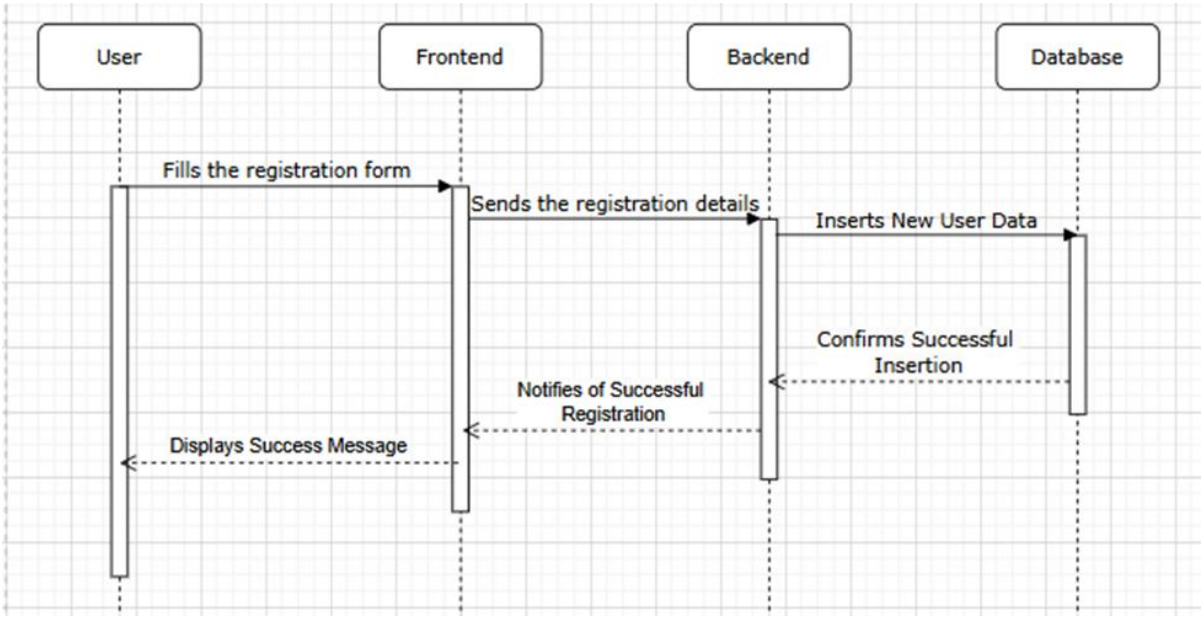

Figure 24 Signup (SD)

8.5.3 Forgot Password

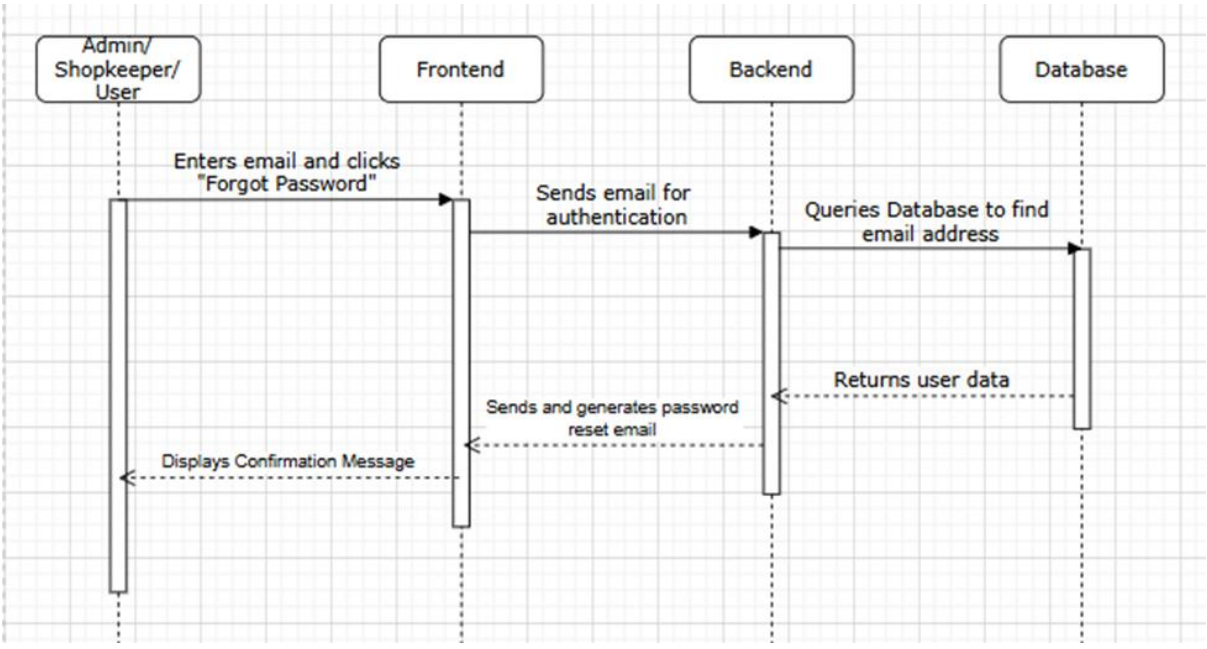

Figure 25 Forgot Password (SD)

8.5.4 Maintenance Reminders

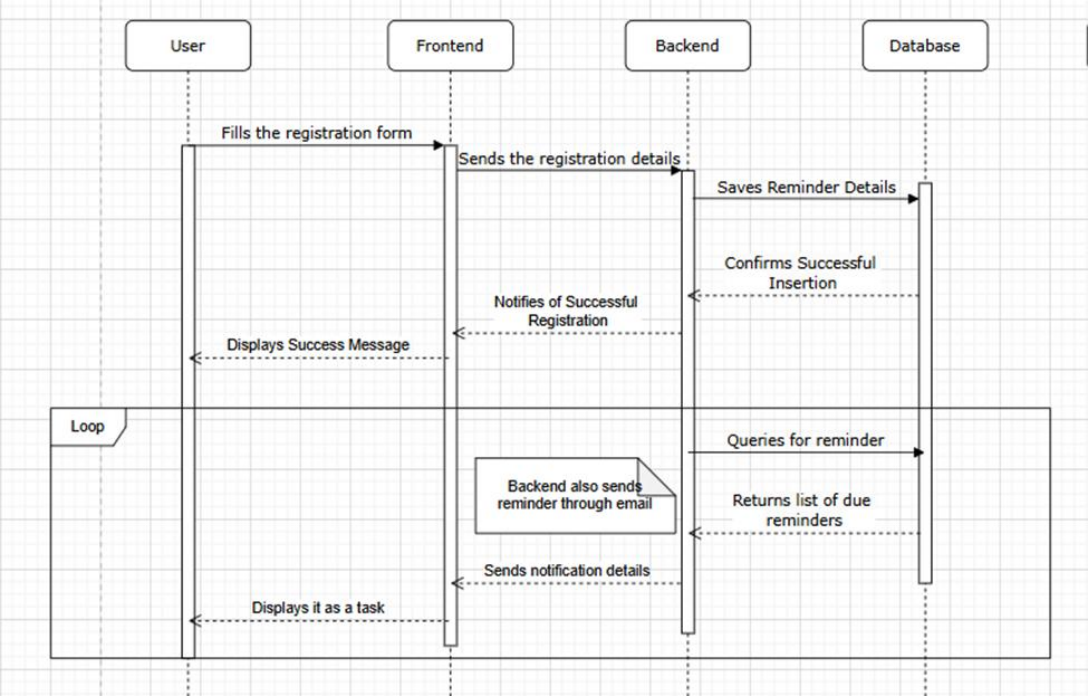

Figure 26 Maintenance Reminders (SD)

8.5.5 Add Car Part

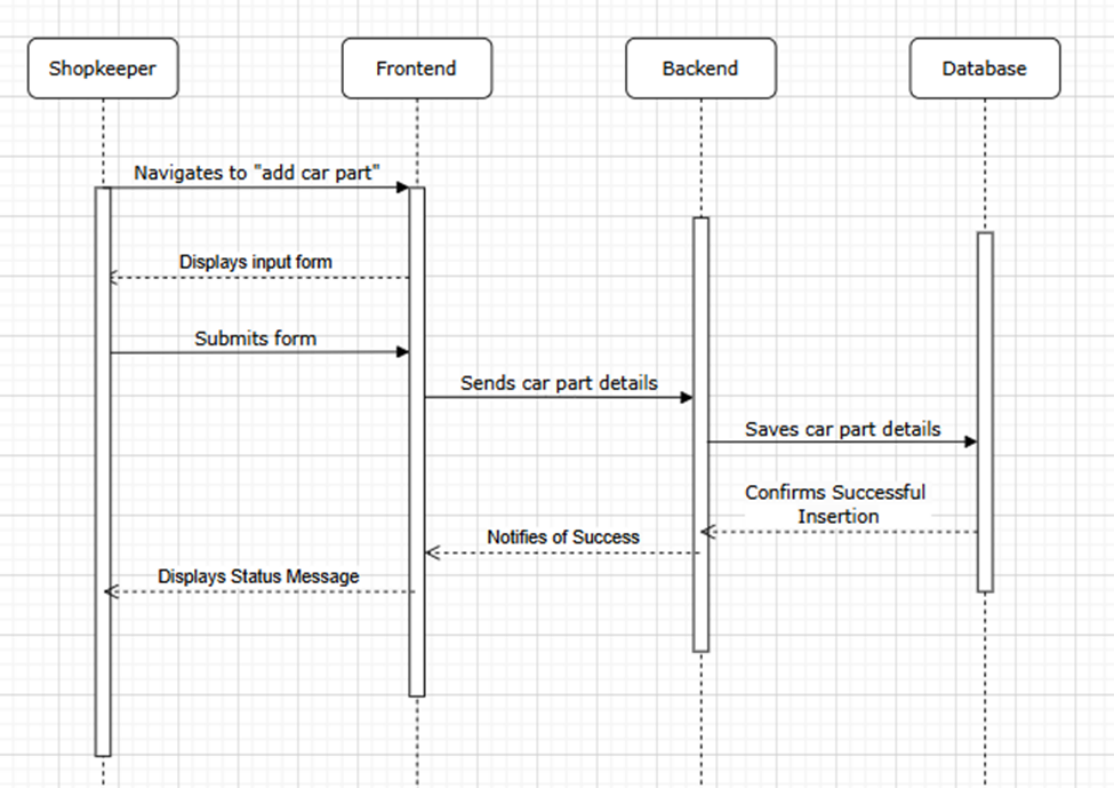

Figure 27 Add Car Part (SD)

8.5.6 Delete Car Part

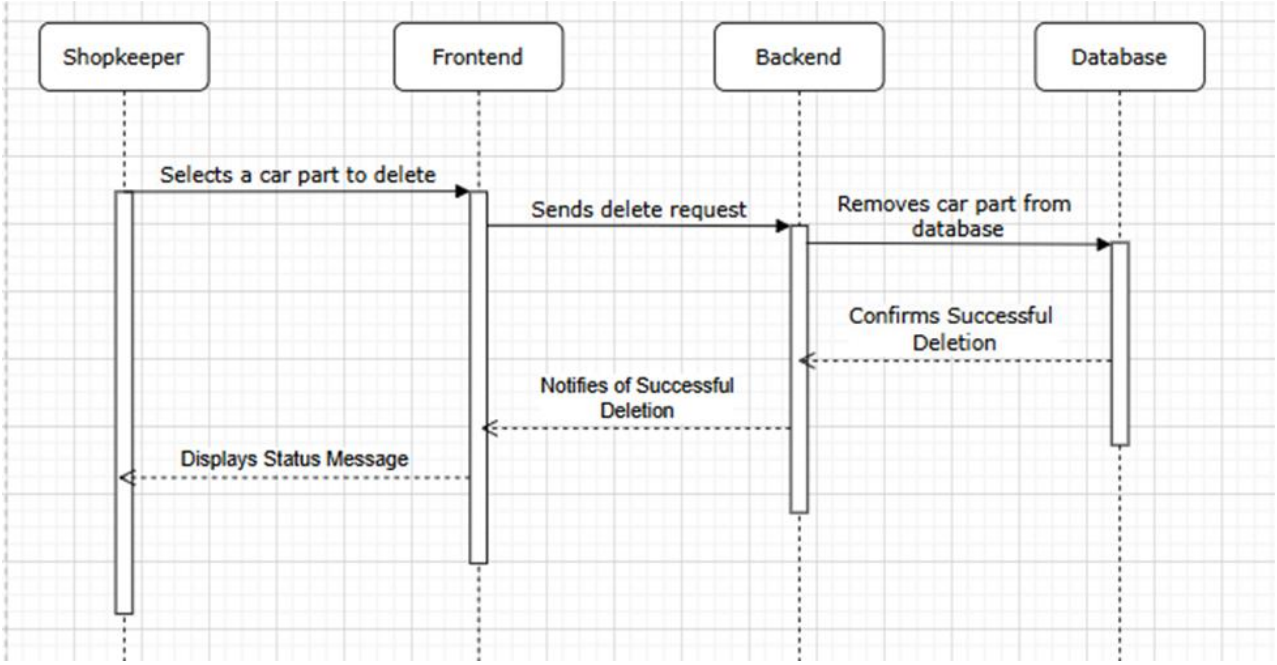

Figure 28 Delete Car Part (SD)

8.5.7 View Orders

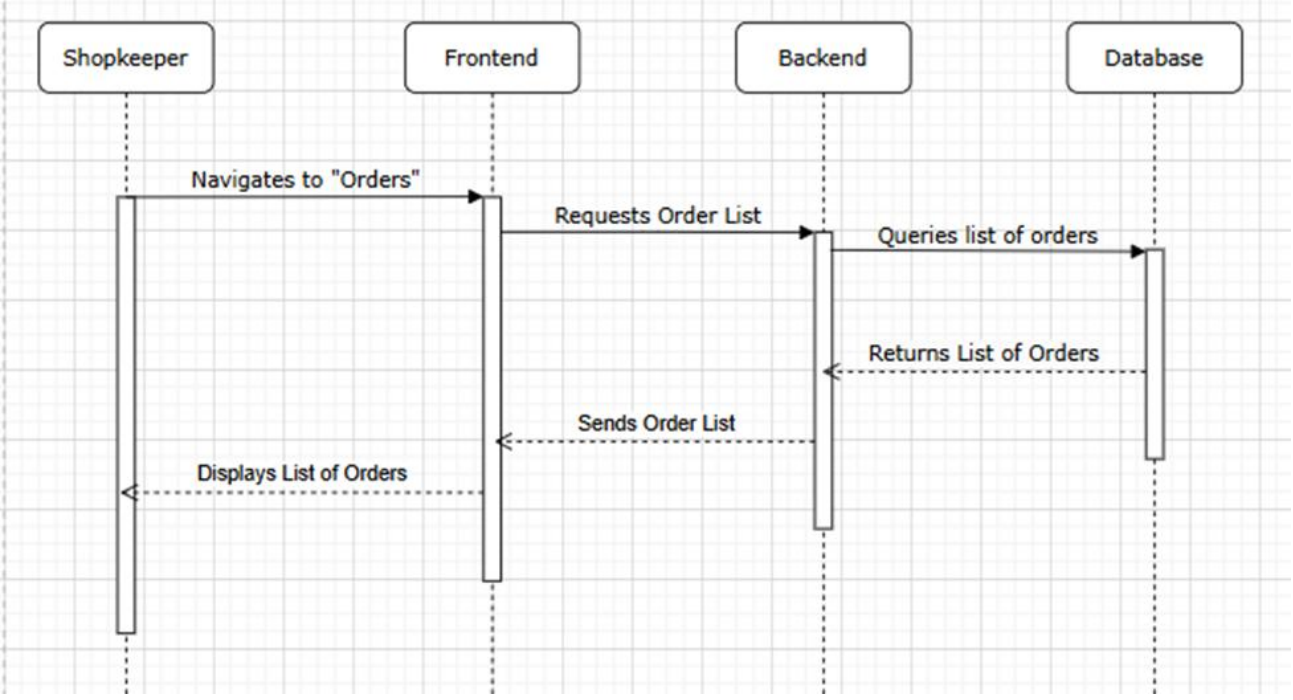

Figure 29 View Orders (SD)

8.5.8 View Order Details

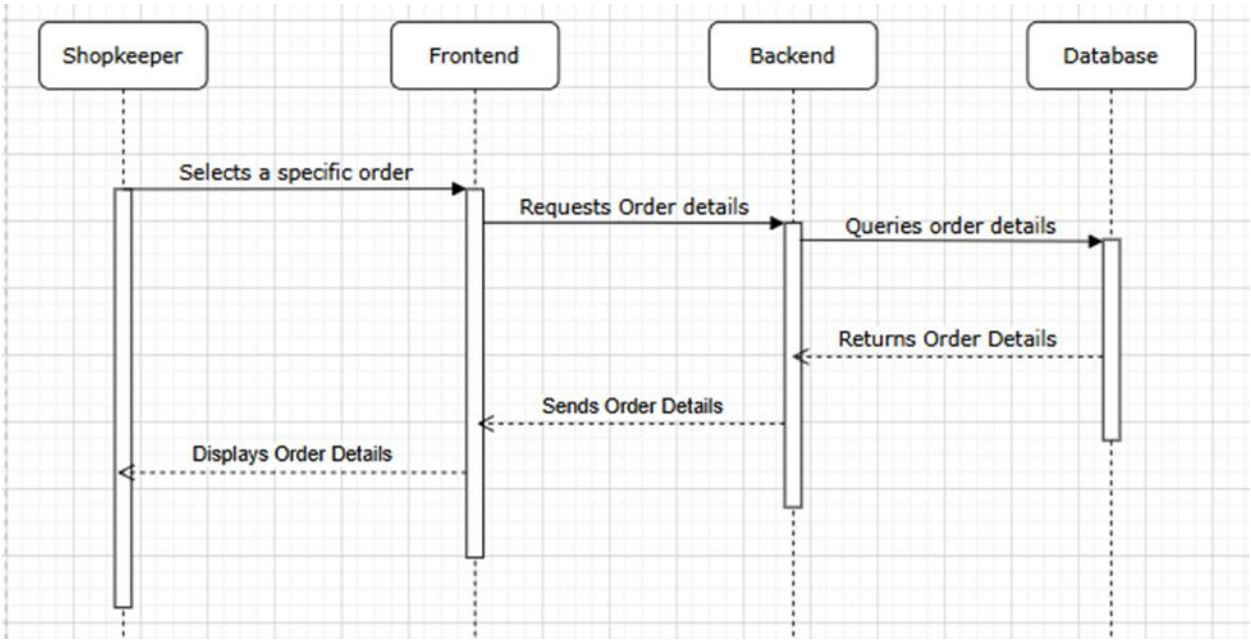

Figure 30 View Order Details (SD)

8.5.9 Cancel an Order

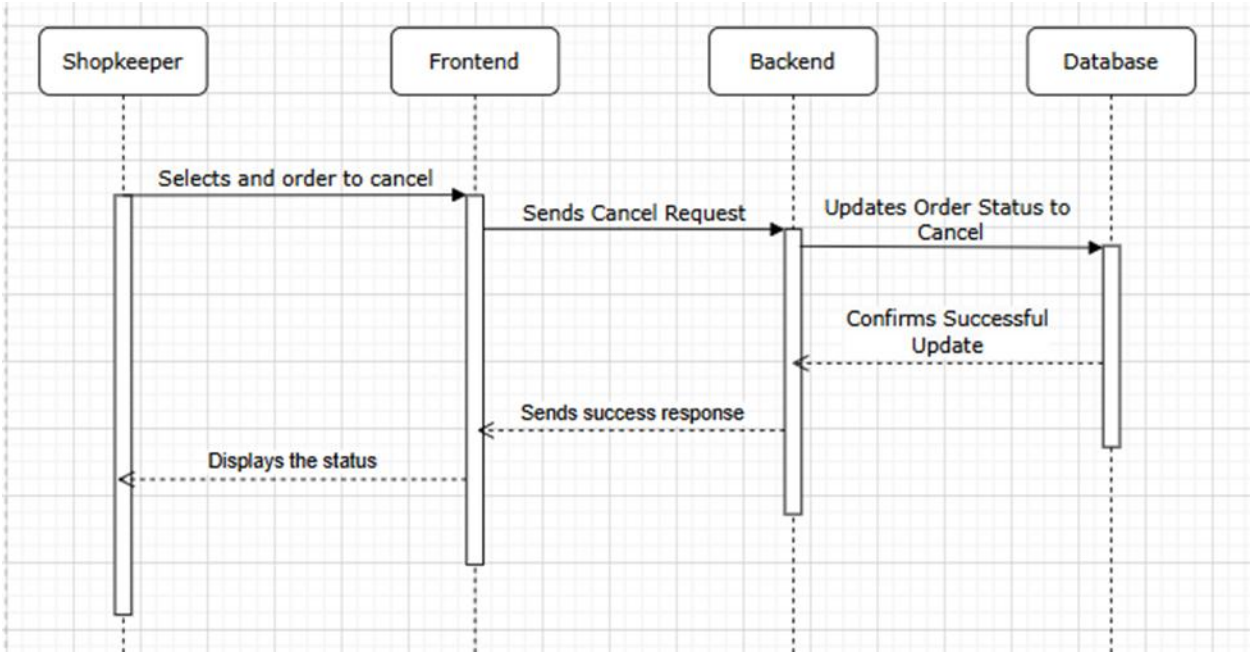

Figure 31 Cancel an Order (SD)

8.5.10 Search Car/Car Part

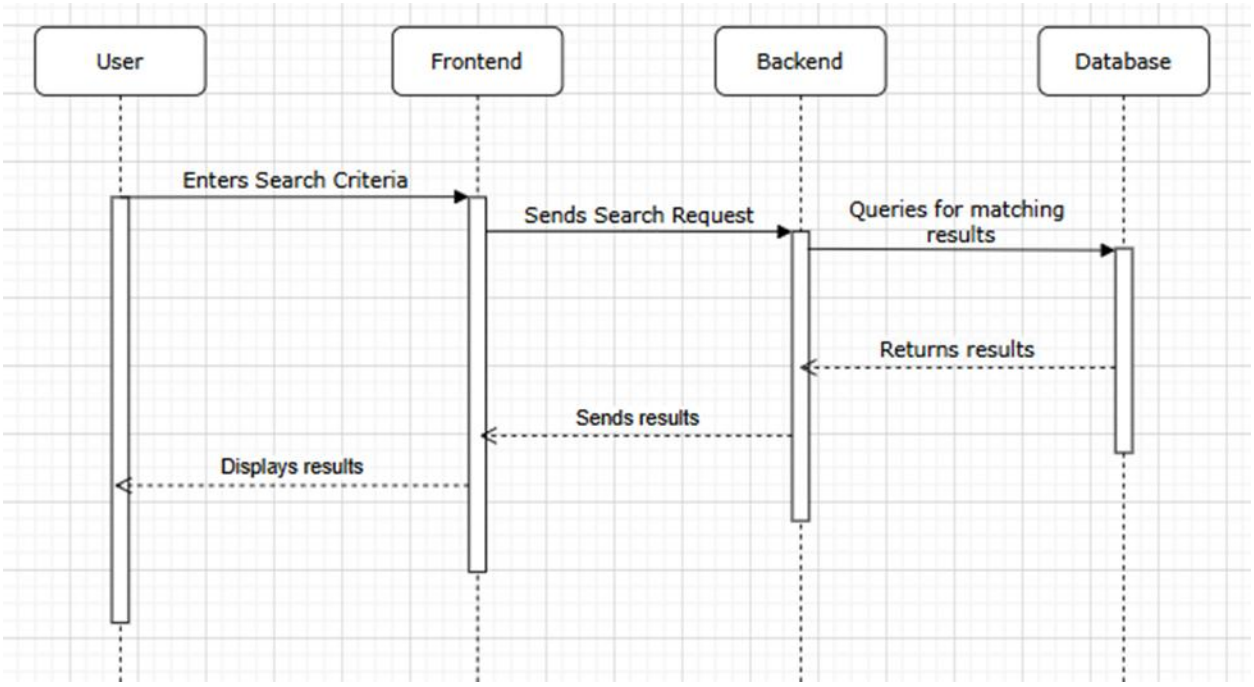

Figure 32 Search Car/Car Part (SD)

8.5.11 View Car/Car Part Details

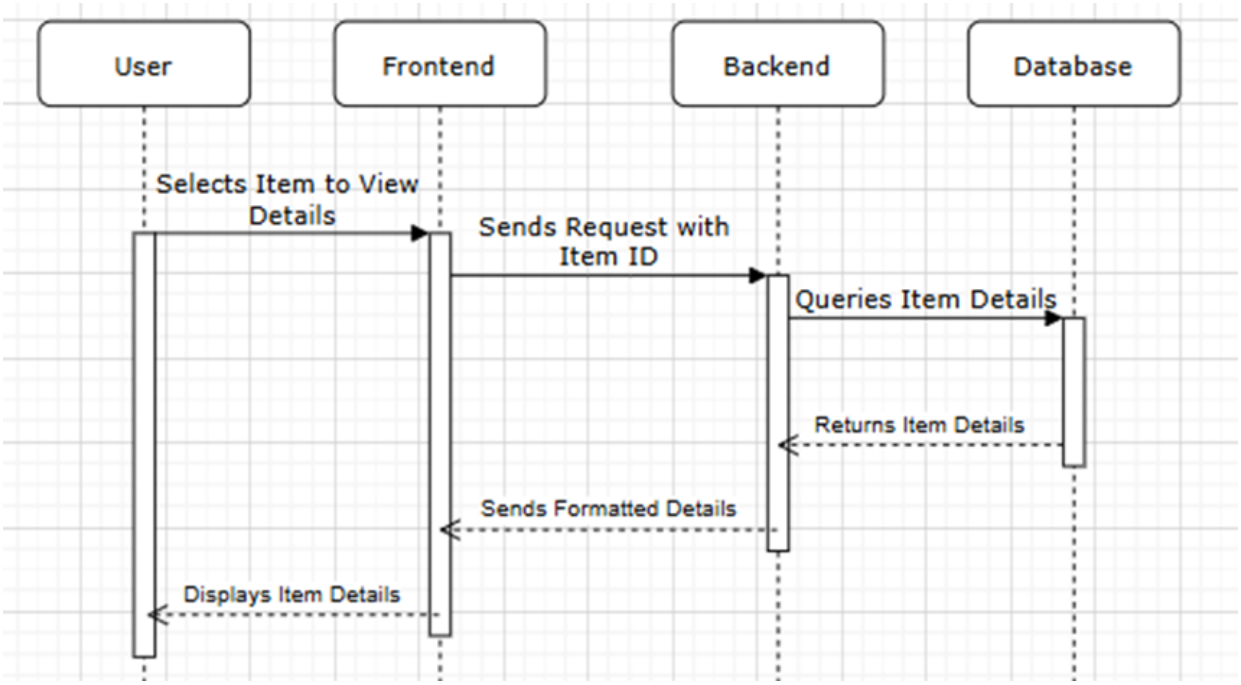

Figure 33 View Car/Car Part (SD)

8.6 Collaboration Diagrams

8.6.1 Login

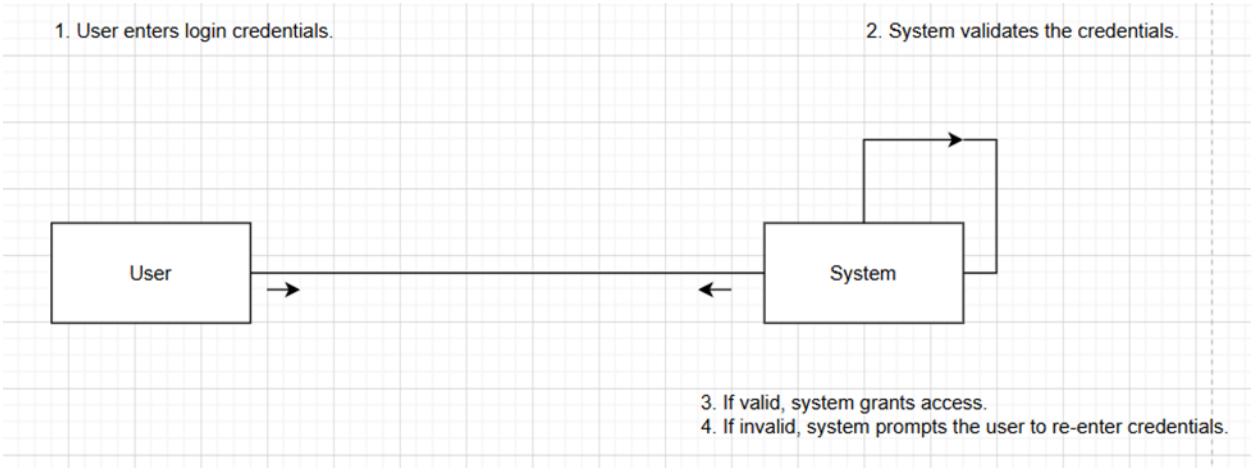

Figure 34 Login (CD)

8.6.2 Signup

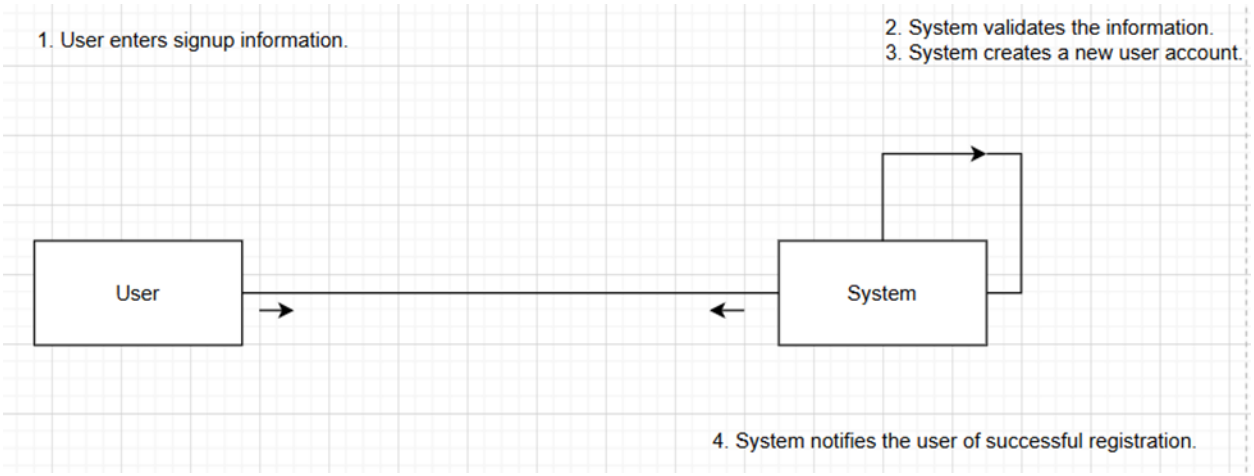

Figure 35 Signup (CD)

8.6.3 Forgot Password

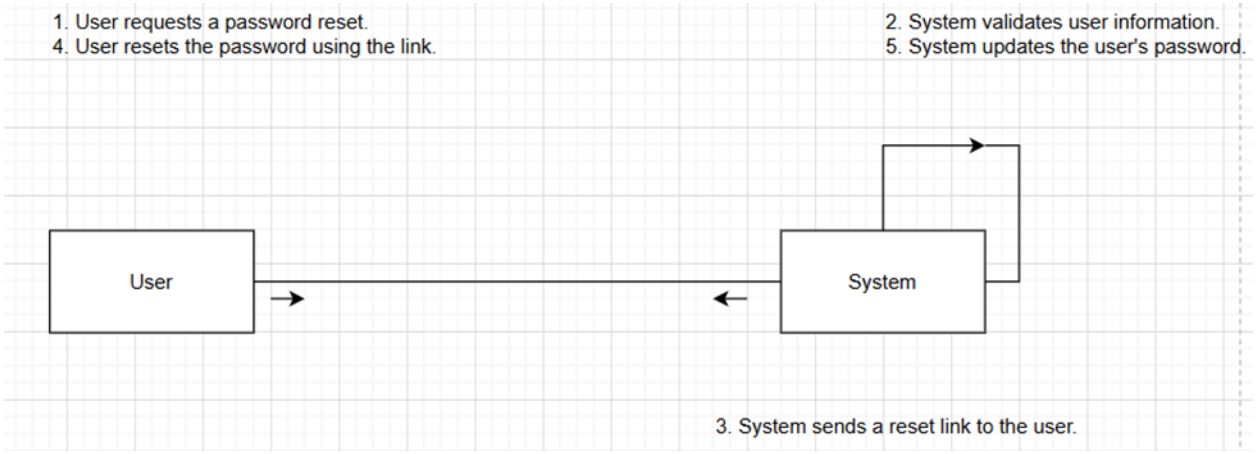

Figure 36 Forgot Password (CD)

8.6.4 List Cars

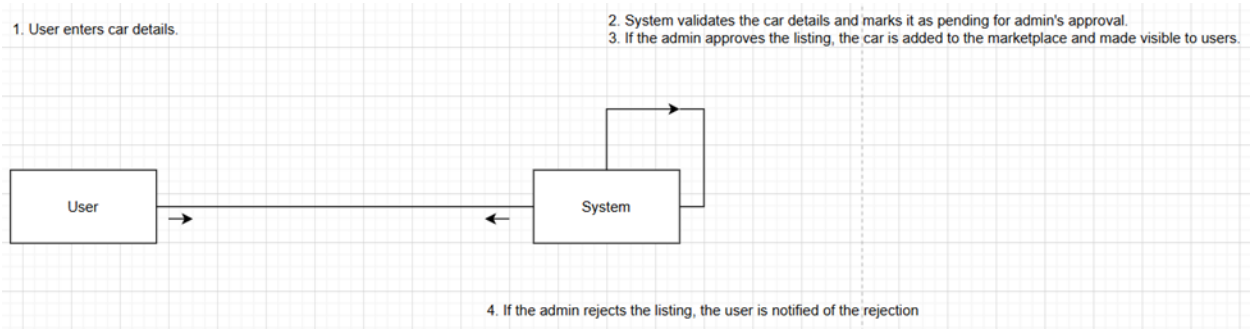

Figure 37 List Cars (CD)

8.6.5 Approve Car Listing

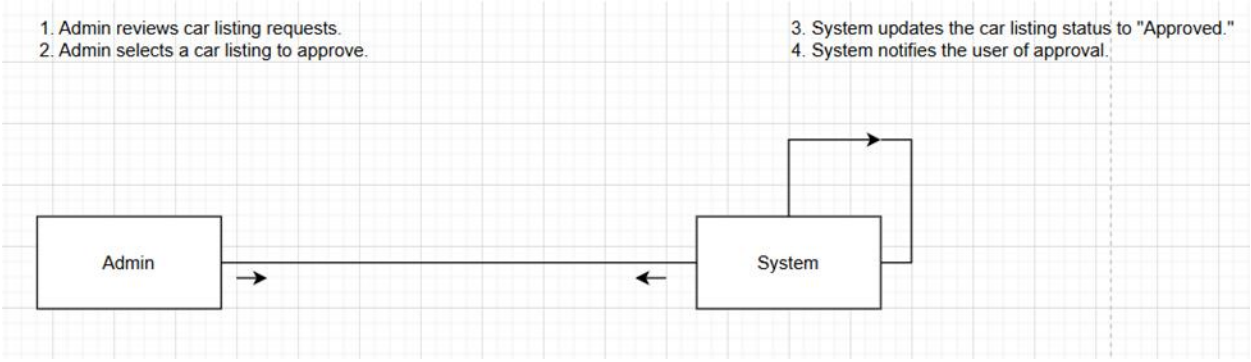

Figure 38 Approve Car Listing (CD)

8.6.6 Maintenance Reminders

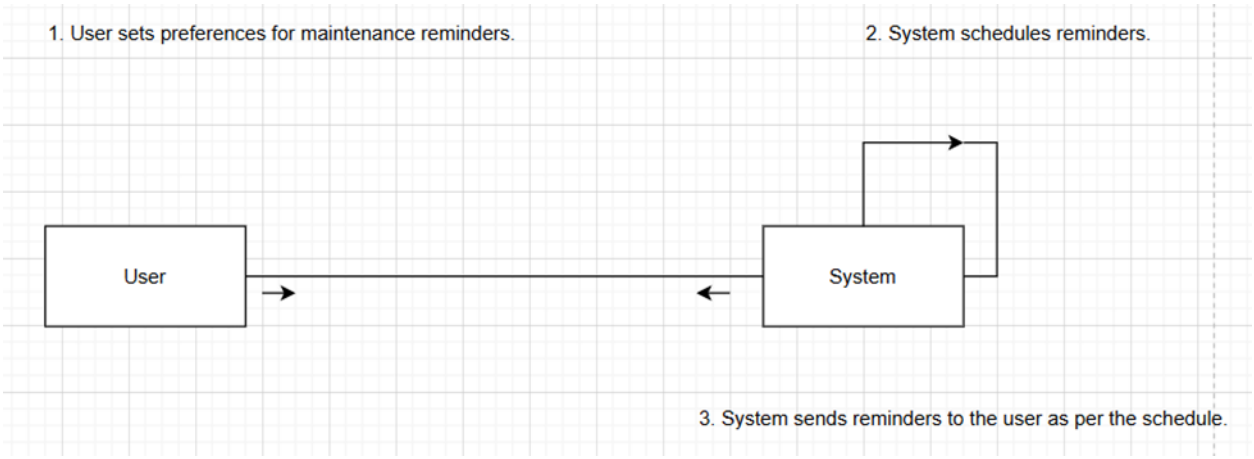

Figure 39 Maintenance Reminders (CD)

8.6.7 Checkout

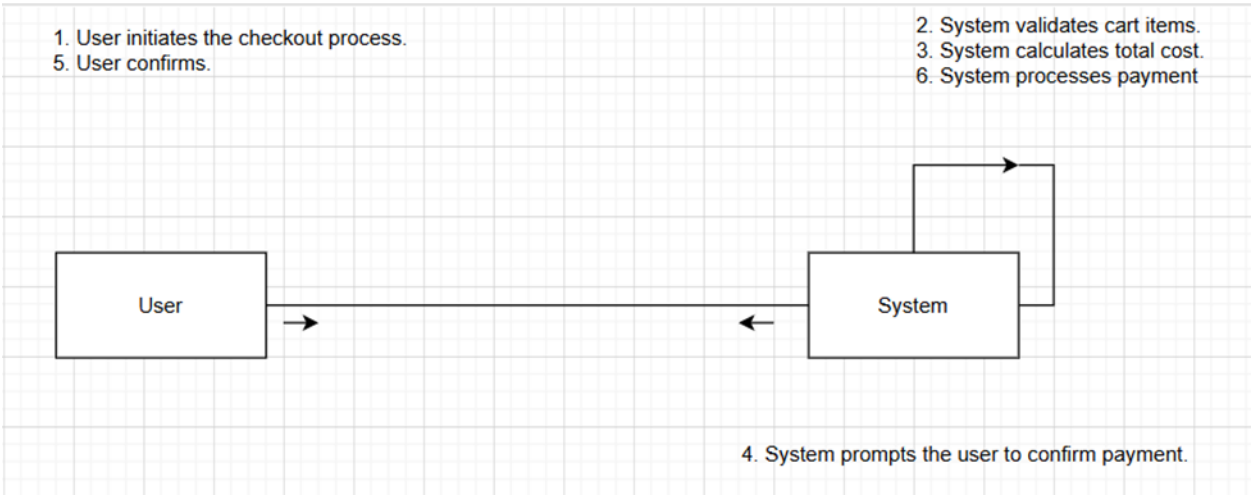

Figure 40 Checkout (CD)

8.6.8 Payment Gateway

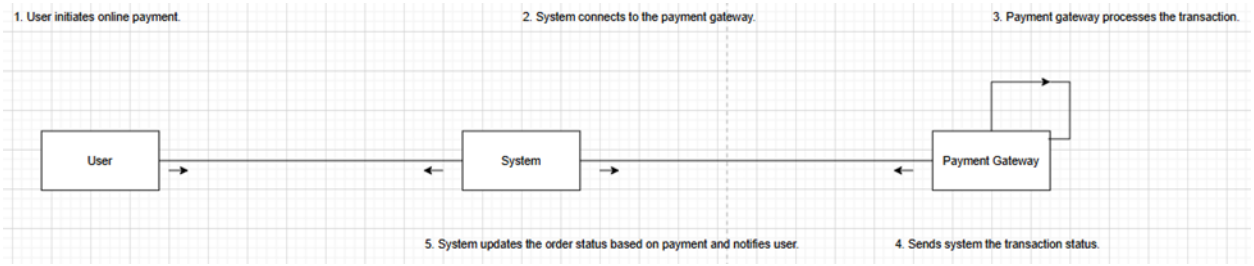

Figure 41 Payment Gateway (CD)

8.7 Use-Case Diagrams

8.7.1 User

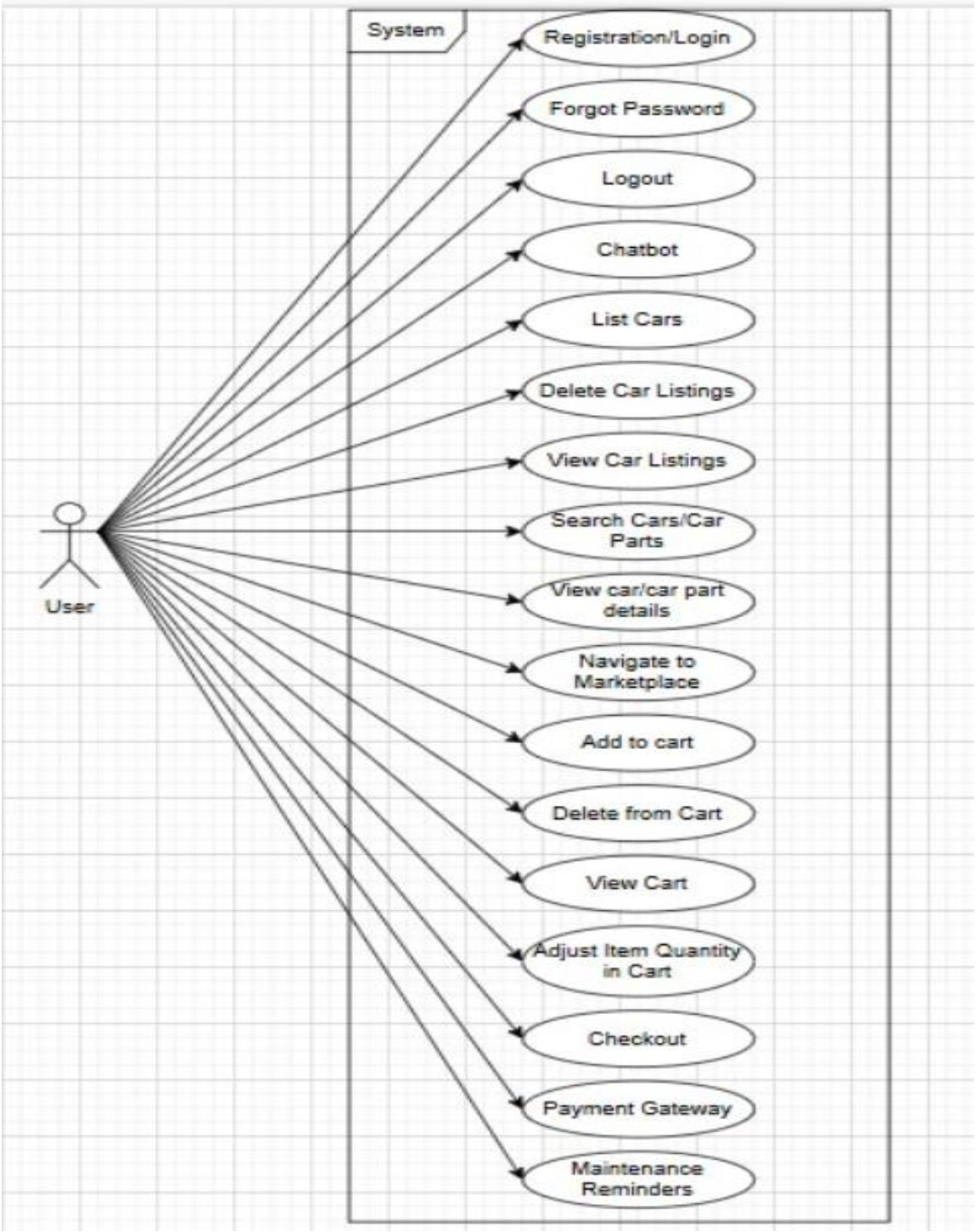

Figure 42 User (Use Case)

8.7.2 Shopkeeper

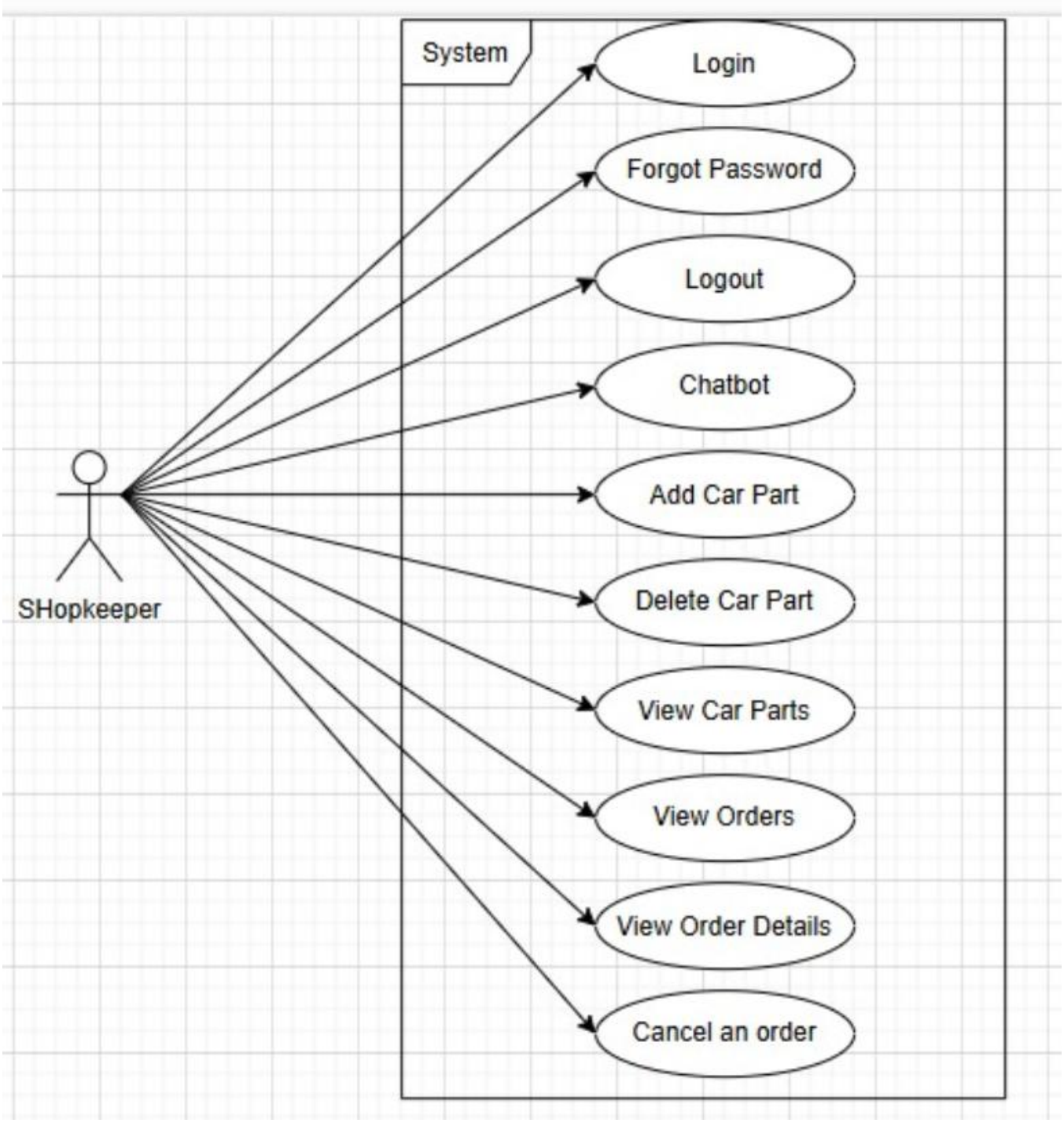

Figure 43 Shopkeeper (Use-Case)

8.7.3 Admin

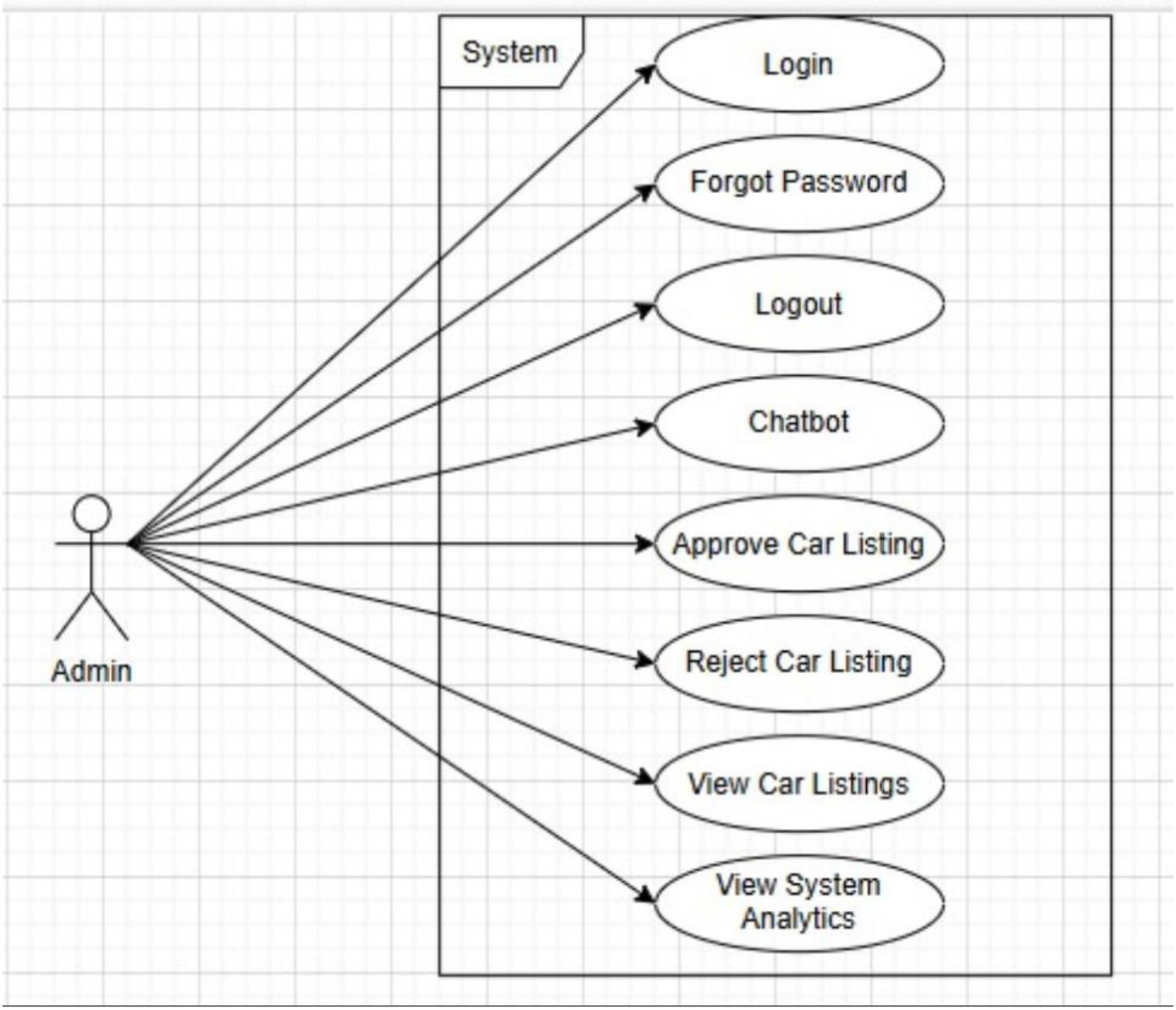

Figure 44 Admin (Use-Case)

8.8 Component Diagrams

8.8.1 User

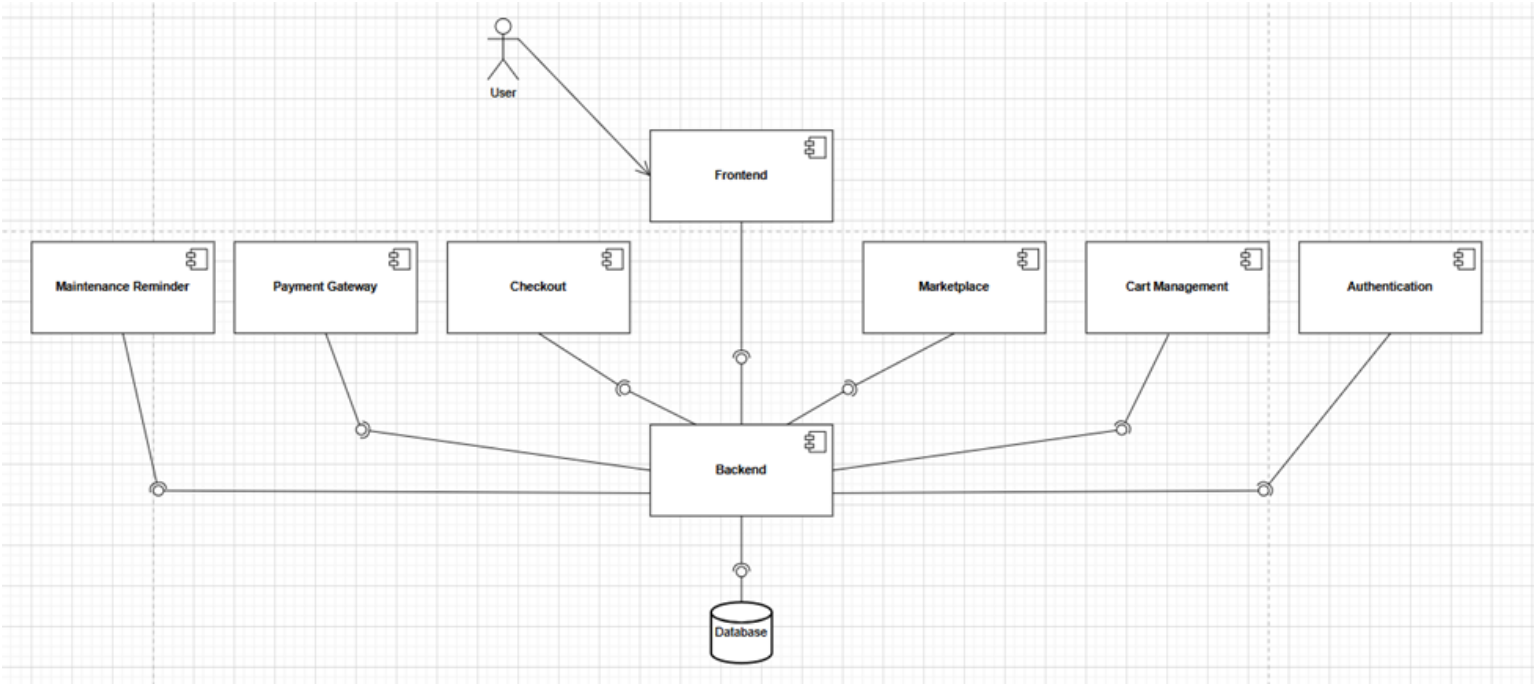

Figure 45 User (Component)

8.8.2 Shopkeeper

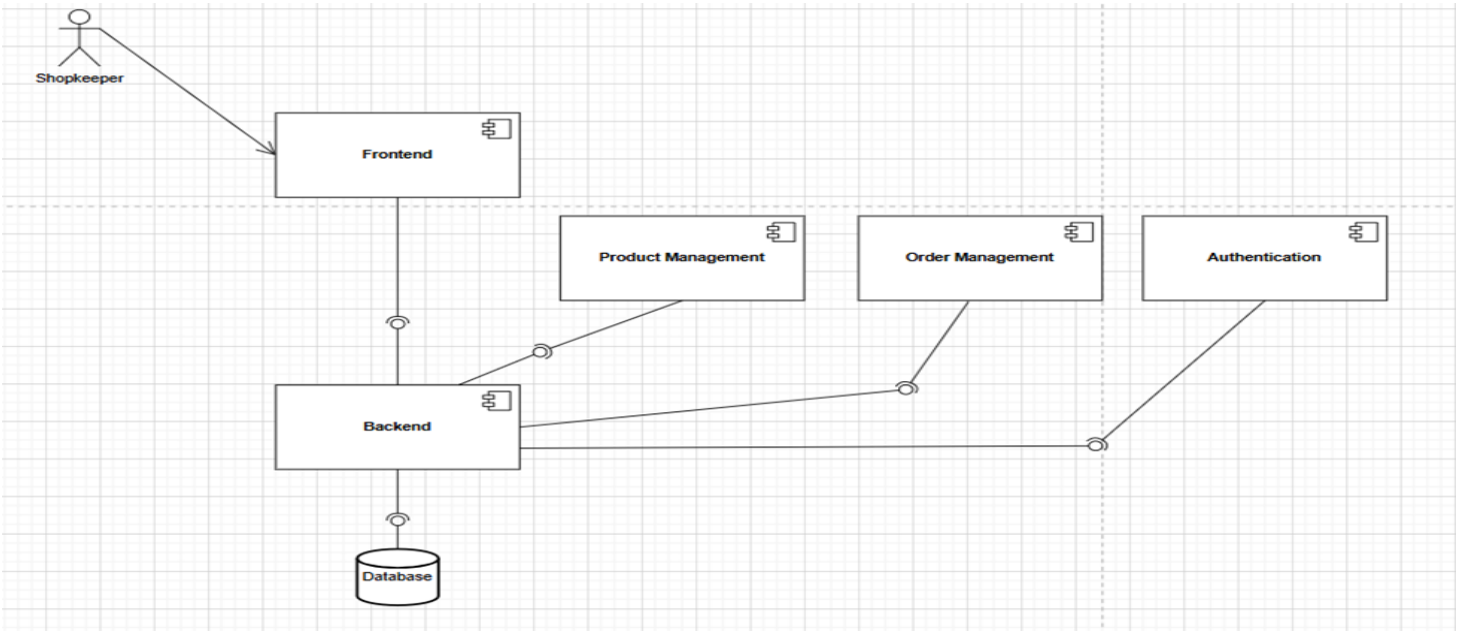

Figure 46 Shopkeeper (Component)

8.8.3 Admin

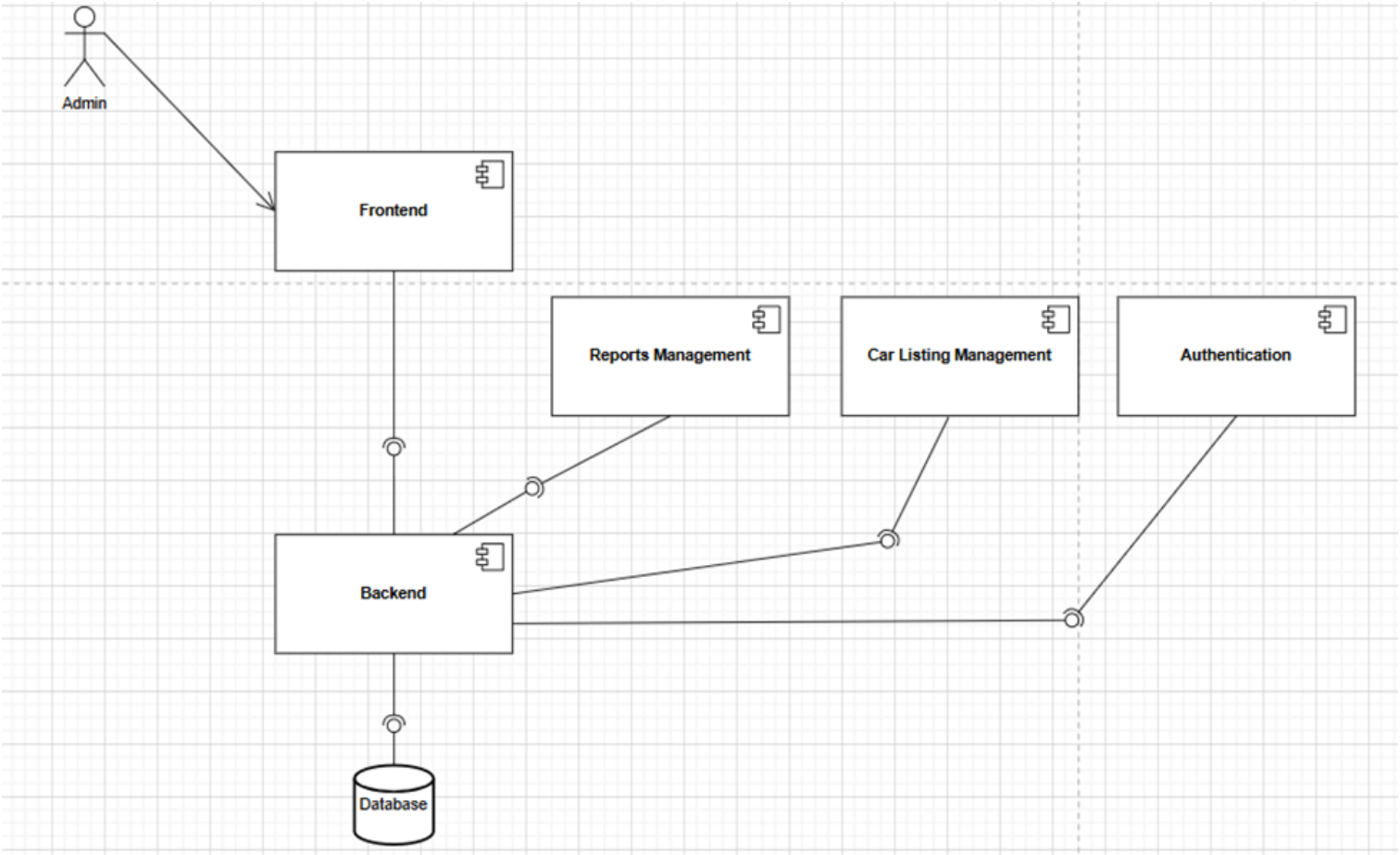

Figure 47 Admin (Component)

8.9 Deployment Diagram

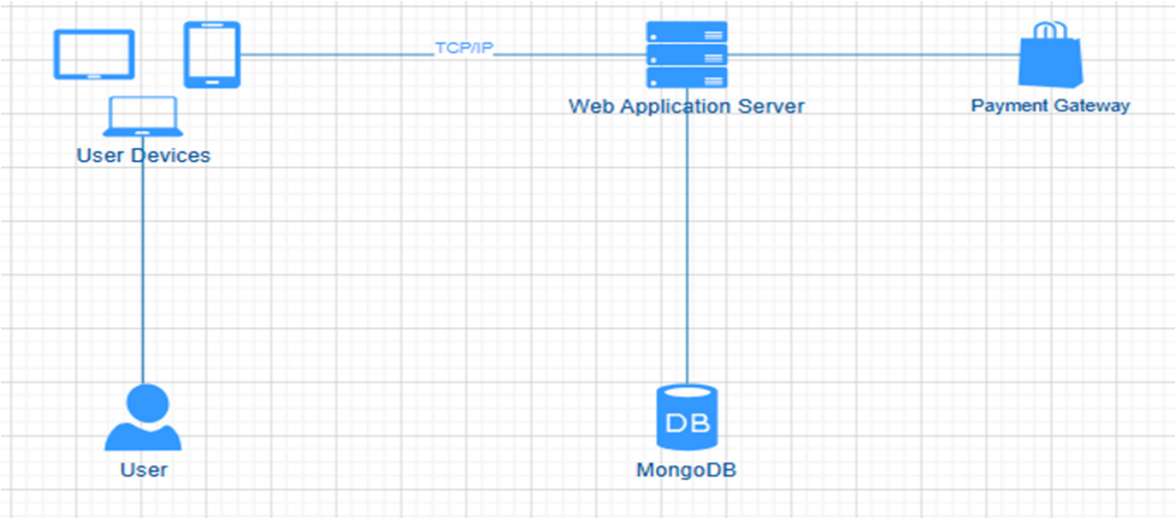

Figure 48 Deployment Diagram

8.10 System Block Diagram

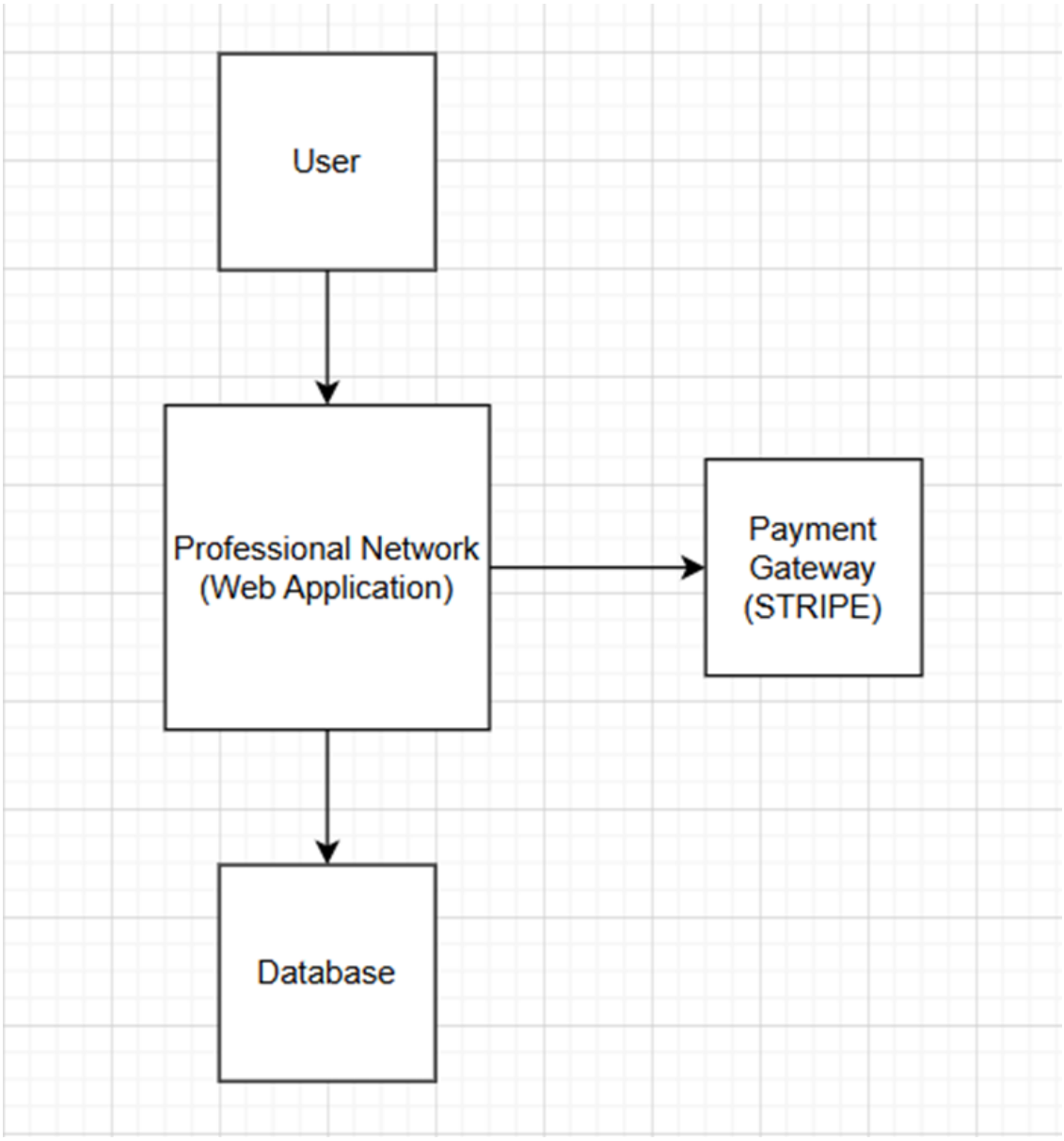

Figure 49 System Block Diagram

Test Cases

9. Test cases

9.1 Login

1	**Test Case ID**	LOGIN_001				
2	**Test Case Name**	Login	**Test Case Description**	To test the functionality of the login feature		
3	**Created By**	Hussam	**Version**	1.1	**Date**	19 Jan 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	User must have an active account.				
7	2	Credentials (email and password) must be registered.				
8						
9	**Test Scenario**	Verify that the user can log in with valid credentials and receives appropriate error messages for invalid inputs.				
10						
11	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**	
12	1	Enter incorrect username and correct password	System will display a message: "No such user exists."	As Expected	Pass	
13	2	Enter correct username and incorrect password	System will display a message: "Incorrect password entered."	As Expected	Pass	
14	3	Enter correct username and password	System will redirect the user to the homepage.	As Expected	Pass	
15	4	Leave both fields blank and submit	System will display a message: "Fill all fields"	As Expected	Pass	

Figure 50 Login (TC)

9.2 Forgot Password

Test Case ID	FP_001				
Test Case Name	Forgot Password	**Test Case Description**	To test the forgot password functionality		
Created By	Roha Ali	**Version**	1.1	**Date**	19 Jan 2025

S.no	**Prerequisites:**
1	User email must be registered in the system.

Test Scenario	Verify that the system can handle password reset requests and send recovery links to the registered email.

Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Enter unregistered email	System displays a message: "Email not found."	Not As Expected	Fail
2	Enter registered email	System sends a password reset link to the email.	As Expected	Pass
3	Leave the email field blank and submit	System displays a message: "Email is required."	As Expected	Pass
4	Login with the new password	System will redirect the user to the homepage.	Not As Expected	Suspended

Figure 51 Forget Password (TC)

1	**Test Case ID**	DCY_003				
2	**Test Case Name**	Forgot Password	**Test Case Description**	To test the forgot password functionality		
3	**Created By**	Roha Ali	**Version**	1.2	**Date**	19 Jan 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	User email must be registered in the system.				
7						
8						
9	**Test Scenario**	Verify that the system can handle password reset requests and send recovery links to the registered email.				
10						
11	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**	
12	1	Enter unregistered email	System displays a message: "Email not found."	System displays a message: "Verify on your email"	Fail	
13	2	Enter registered email	System sends a password reset link to the email.	System sends a password reset link to the email.	Pass	
14	3	Leave the email field blank and submit	System displays a message: "Email is required."	System displays a message: "Email is required."	Pass	
15	4	Login with the new password	System will redirect the user to the homepage.	System redirects the user to the homepage.	Pass	
16						

Figure 52 Forget Password-2 (TC)

9.3 Logout

1	**Test Case ID**	LOGOUT_001				
2	**Test Case Name**	Logout	**Test Case Description**	To test the functionality of the logout feature		
3	**Created By**	Hussam	**Version**	1.1	**Date**	19 Jan 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	User must be logged into the system.				
8						
9	**Test Scenario**	Verify that the user can log out successfully and cannot access protected pages afterward.				
10						
11	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**	
12	1	Click the logout button	System logs the user out and redirects to the login page.	As Expected	Pass	

Figure 53 Logout (TC)

9.4 Signup

1	**Test Case ID**	SIGNUP_001				
2	**Test Case Name**	Sign Up	**Test Case Description**	To test the user registration process.		
3	**Created By**	Hussam	**Version**	1.1	**Date**	19 Jan 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	User must have access to the registration page.				
7						
8						
9	**Test Scenario**	Verify that users can create a new account successfully.				
10						

11	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
12	1	User navigates to the sign-up page.	System displays the registration form.	As Expected	Pass
13	2	User fills out the form with valid information.	System validates the input and creates the account successfully.	As Expected	Pass
14	3	User submits the form with missing data.	System will display a message: "Fill all fields"	As Expected	Pass
15	4	User submits the form with an email that is already registered.	System displays error: "Email already in use."	As Expected	Pass
16	5	User submits the form with a number that is already registered.	System displays error: "Phone Number already in use."	Not As Expected	Fail

Figure 54 Signup (TC)

1	**Test Case ID**	DCY_006				
2	**Test Case Name**	Sign Up	**Test Case Description**	To test the user registration process.		
3	**Created By**	Hussam	**Version**	1.2	**Date**	19 May 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	User must have access to the registration page.				
7						
8						
9	**Test Scenario**	Verify that users can create a new account successfully.				
10						

11	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
12	1	User navigates to the sign-up page.	System displays the registration form.	System displays the registration form.	Pass
13	2	User fills out the form with valid information.	System validates the input and creates the account successfully.	System validates the input and creates the account successfully.	Pass
14	3	User submits the form with missing data.	System will display a message: "Fill all fields"	System displays a message: "Fill all fields"	Pass
15	4	User submits the form with an email that is already registered.	System displays error: "Email already in use."	System displays error: "Email already in use."	Pass
16	5	User submits the form with a number that is already registered.	System displays error: "Phone Number already in use."	System displays error: "Phone Number already in use."	Pass

Figure 55 Signup (TC)

9.5 Maintenance Reminder

1	**Test Case ID**	MAINT_001				
2	**Test Case Name**	Maintenance Reminders	**Test Case Description**	To test the functionality of setting and receiving maintenance reminders		
3	**Created By**	Roha Ali	**Version**	1.1	**Date**	19 Jan 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	User must be logged in.				
7						
8	**Test Scenario**	Verify that users can set maintenance reminders and receive notifications.				
9						
10	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**	
11	1	User sets a reminder for oil change	System saves the reminder successfully.	As Expected	Pass	
12	2	Reminder time is reached	System sends a email to the user.	Not As Expected	Fail	
13	3	Reminder time is reached again	System calculates next reminder date automatically	Not As Expected	Fail	

Figure 56 Maintenance Reminder (TC)

1	**Test Case ID**	DCY_008				
2	**Test Case Name**	Maintenance Reminders	**Test Case Description**	To test the functionality of setting and receiving maintenance reminders		
3	**Created By**	Roha Ali	**Version**	1.2	**Date**	19 Jan 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	User must be logged in.				
7						
8	**Test Scenario**	Verify that users can set maintenance reminders and receive notifications.				
9						
10	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**	
11	1	User sets a reminder for oil change	System saves the reminder successfully.	System saves the reminder successfully.	Pass	
12	2	Reminder time is reached	System sends a email to the user.	System does not send any email.	Fail	
13	3	Reminder time is reached again	System calculates next reminder date automatically	System calculates next reminder date automatically.	Pass	

Figure 57 Maintenance Reminder-2 (TC)

9.6 Search Cars/ Car Parts

1	**Test Case ID**	SEARCH_001				
2	**Test Case Name**	Search Cars/Car Parts	**Test Case Description**	To test the search functionality for cars and car parts		
3	**Created By**	Roha Ali	**Version**	1.1	**Date**	19 Jan 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	User must be logged in.				
7	2	Database must contain car or car part listings.				
8						
9	**Test Scenario**	Verify that the search functionality retrieves results by matching letters in the car or car part names.				
10						
11	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**	
12	1	User enters a full name of a car or car part in the search bar (e.g., "Toyota Corolla").	System displays the exact match results for the query.	As Expected	Pass	
13	2	User enters a partial name of a car or car part (e.g., "Cor").	System displays all results containing the partial name (e.g., "Corolla," "Core Filter").	As Expected	Pass	
14	3	User enters a single letter (e.g., "T").	System displays all results containing the letter "T" (e.g., "Toyota," "Tire").	As Expected	Pass	
15	4	User enters a letter or sequence that does not match any records (e.g., "XYZ").	System displays nothing	As Expected	Pass	
16	5	User enters a search term with mixed case (e.g., "toyota").	System performs a case-insensitive search and displays relevant results (e.g., "Toyota").	As Expected	Pass	

Figure 58 Search Cars/Car Parts (TC)

9.7 View Car/ Car Part Details

1	**Test Case ID**	VIEW_DETAILS_001				
2	**Test Case Name**	View Car/Car Part Details	**Test Case Description**	To test viewing details of a car or car part		
3	**Created By**	Hussam	**Version**	1.1	**Date**	19 Jan 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	User must be logged in.				
7	2	Listing must be available.				
8						
9	**Test Scenario**	Verify that users can view detailed information about a car or car part.				
10						
11	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**	
12	1	Click on a car listing	System displays all details of the selected car.	Not As Expected	Fail	
13	2	Click on a car part listing	System displays all details of the selected car part.	Not Executed	Deferred	

Figure 59 View Car/ Car Part Details (TC)

Test Case ID	DCY_011					
Test Case Name	View Car/Car Part Details	**Test Case Description**	To test viewing details of a car or car part			
Created By	Hussam	**Version**	1.2	**Date**	19 Jan 2025	

S.no	**Prerequisites:**
1	User must be logged in.
2	Listing must be available.

Test Scenario	Verify that users can view detailed information about a car or car part.

Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Click on a car listing	System displays all details of the selected car.	System displays all details of the selected car.	Pass
2	Click on a car part listing	System displays all details of the selected car part.	System displays all details of the selected car part.	Pass

Figure 60 View Car/ Car Part Details-2 (TC)

9.8 Chat (User with Admin/Shopkeeper)

Test Case ID	CHAT_USER_001					
Test Case Name	Chat (User with Admin/Shopkeeper)	**Test Case Description**	To test how a user initiates and communicates with the admin or shopkeeper via chat.			
Created By	Roha Ali	**Version**	1.1	**Date**	19 Jan 2025	

S.no	**Prerequisites:**
1	User must be logged in.
2	Admin or shopkeeper must have an active account.

Test Scenario	Verify that users can initiate and communicate with the admin or shopkeeper.

Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	User initiates a chat with admin/shopkeeper.	Chat window opens, allowing the user to send messages.	As Expected	Pass
2	User sends a message to admin/shopkeeper.	Message appears in the chat window for both user and admin/shopkeeper.	As Expected	Pass
3	Admin/shopkeeper replies to the user's message.	Reply appears in the user's chat window.	As Expected	Pass
4	User sends a blank message.	System displays an error: "Message cannot be empty."	Not As Expected	Fail
5	User navigates away from chat during an ongoing session.	Chat session remains active and resumes when the user returns.	As Expected	Pass

Figure 61 Chat (User with Admin/Shopkeeper) (TC)

9.9 Chat (reverse)

Test Case ID	CHAT_ADMIN_SHOP_001				
Test Case Name	Chat (Admin/Shopkeeper with Users	**Test Case Description**	To test how an admin or shopkeeper communicates with users via chat.		
Created By	Roha Ali	**Version**	1.1	**Date**	19 Jan 2025

S.no	Prerequisites:
1	Admin or shopkeeper must have an active account.
2	Users must have valid accounts.

Test Scenario	Verify that admin/shopkeeper can communicate with users effectively.

Step no.	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Admin/shopkeeper opens a chat session with a user.	Chat window opens, showing the list of all messages exchanged.	As Expected	Pass
2	Admin/shopkeeper sends a message to the user.	Message appears in both admin/shopkeeper and user's chat window.	As Expected	Pass
3	User replies to the admin/shopkeeper's message.	Reply appears in admin/shopkeeper's chat window.	As Expected	Pass
4	Admin/shopkeeper sends a blank message.	System displays an error: "Message cannot be empty."	Not As Expected	Fail
5	Admin/shopkeeper ends the chat session.	System logs the chat session for future reference.	As Expected	Pass
6	Admin/shopkeeper clicks on the chat icon	System shows a list of all the chats	As Expected	Pass

Figure 62 Chat (Admin/Shopkeeper with Users) (TC)

9.10 List Car

Test Case ID	LIST_001				
Test Case Name	List Cars	**Test Case Description**	To test the functionality of listing cars for sale.		
Created By	Hussam	**Version**	1.1	**Date**	19 Jan 2025

S.no	Prerequisites:
1	User must have an active account.
2	User must have car details available.

Test Scenario	Verify that users can list cars for sale.

Step no.	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	User navigates to the "List Cars" section.	System displays a form to add car details.	As Expected	Pass
2	User fills out the form with valid car details.	System saves the car listing as pending in my cars.	Not As Expected	Fail
3	User submits the form with incomplete details.	System displays an error: "All fields are required."	As Expected	Pass
4	Listing is rejected by the admin	System saves the car listing as rejected and it can only be seen by the user in my cars	Not As Expected	Fail
5	Listing is approved by the admin	System saves the car listing as rejected and only on approval it is shown to other users	Not As Expected	Fail

Figure 63 List Cars (TC)

1	**Test Case ID**	DCY_015				
2	**Test Case Name**	List Cars	**Test Case Description**	To test the functionality of listing cars for sale.		
3	**Created By**	Hussam	**Version**	1.2	**Date**	19 Jan 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	User must have an active account.				
7	2	User must have car details available.				
8						
9	**Test Scenario**	Verify that users can list cars for sale.				
10						

11	Step no.	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
12	1	User navigates to the "List Cars" section.	System displays a form to add car details.	System displays a form to add car details.	Pass
13	2	User fills out the form with valid car details.	System saves the car listing as pending in my cars.	System saves the car listing as pending in my cars.	Pass
14	3	User submits the form with incomplete details.	System displays an error: "All fields are required."	System displays an error: "All fields are required."	Pass
15	4	Listing is rejected by the admin	System saves the car listing as rejected and it can only be seen by the user in my cars.	System saves the car listing as rejected and it can only be seen by the user in my cars.	Pass
16	5	Listing is approved by the admin	System saves the car listing as rejected and only on approval it is shown to other users.	System saves the car listing as rejected and only on approval it is shown to other users.	Pass

Figure 64 List Cars-2 (TC)

9.11 Reject Car Listing

1	**Test Case ID**	REJECT_001				
2	**Test Case Name**	Reject Car Listing	**Test Case Description**	To test the admin's ability to reject car listings.		
3	**Created By**	Roha Ali	**Version**	1.1	**Date**	19 Jan 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	Admin must be logged in.				
7	2	Pending car listings must exist.				
8						
9	**Test Scenario**	Verify that the admin can reject car listings.				
10						

11	Step no.	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
12	1	Admin views pending car listings.	List of all pending car listings is displayed.	As Expected	Pass
13	2	Admin rejects a car listing.	System marks the listing as "Rejected".	As Expected	Pass

Figure 65 Reject Car Listing (TC)

9.12 Approve Car Listing

1	**Test Case ID**	APPROVE_001				
2	**Test Case Name**	Approve Car Listing	**Test Case Description**	To test the admin's ability to approve car listings.		
3	**Created By**	Roha Ali	**Version**	1.1	**Date**	19 Jan 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	Admin must be logged in.				
7	2	Pending car listings must exist.				
8						
9	**Test Scenario**	Verify that the admin can approve car listings.				
10						
11	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**	
12	1	Admin views pending car listings.	List of all pending car listings is displayed.	As Expected	Pass	
13	2	Admin approves a car listing.	System marks the listing as "Approved".	As Expected	Pass	

Figure 66 Approve Car Listing (TC)

9.13 View Car Listing (User)

1	**Test Case ID**	VIEW_CARS_001				
2	**Test Case Name**	View Car Listings (User)	**Test Case Description**	To test if a user can view their own car listings.		
3	**Created By**	Hussam	**Version**	1.1	**Date**	19 Jan 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	User must be logged in.				
7	2	User must have listed at least one car.				
8						
9	**Test Scenario**	Verify that users can view their own car listings.				
10						
11	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**	
12	1	User navigates to "My Cars" section.	System displays a list of all cars listed by the user.	As Expected	Pass	
13	2	User views details of a specific listing.	Detailed information about the selected listing is displayed.	As Expected	Pass	
14	3	User has no car listings.	System displays a message: "You have no active listings."	As Expected	Pass	

Figure 67 View Car Listing (User) (TC)

9.14 Delete Car Listing

Test Case ID	DELETE_CAR_001					
Test Case Name	Delete Car Listings	**Test Case Description**	To test if a user can delete their own car listings.			
Created By	Roha Ali	**Version**	1.1	**Date**	19 Jan 2025	

S.no	Prerequisites:
1	User must be logged in.
2	User must have at least one active car listing.

Test Scenario	Verify that users can delete their own car listings.

Step no.	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	User navigates to "My Cars" section.	System displays a list of all cars listed by the user.	As Expected	Pass
2	User selects a car listing and clicks on the delete icon.	System removes the car listing.	As Expected	Pass

Figure 68 Delete Car Listings (TC)

9.15 View Car Listing Requests

Test Case ID	VIEW_REQUESTS_001					
Test Case Name	View Car Listing Requests	**Test Case Description**	To test if an admin can view all car listing requests.			
Created By	Hussam	**Version**	1.1	**Date**	19 Jan 2025	

S.no	Prerequisites:
1	Admin must be logged in.
2	At least one car listing request must exist in the system.

Test Scenario	Verify that the admin can view all car listing requests.

Step no.	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Admin views pending car listings.	List of all pending car listings is displayed.	As Expected	Pass
2	Admin clicks on a specific car listing request.	Detailed information about the car listing request is displayed.	As Expected	Pass
3	There are no pending requests.	System displays nothing.	As Expected	Pass

Figure 69 View Car Listing Requests (TC)

9.16 Delete User

Test Case ID	DCY_021					
Test Case Name	Delete User	**Test Case Description**	To test if an admin can delete a user from the platform.			
Created By	Roha Ali	**Version**	1.1	**Date**	19 Mar 2025	

S.no	**Prerequisites:**
1	Admin must be logged in.
2	At least one user must be present in the system.

Test Scenario	Verify that the admin can view users and delete a selected user successfully.

Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Admin navigates to the User Management Panel.	List of all registered users is displayed.	List of all registered users is displayed.	Pass
2	Admin selects a user and clicks "Delete."	System prompts for confirmation of deletion.	System prompts for confirmation of deletion.	Pass
3	Admin confirms deletion.	User is removed from the system, and confirmation message is shown.	User is removed from the system, and confirmation message is shown.	Pass
4	Admin attempts to delete a non-existent user.	System displays error message: "User does not exist."	System displays error message: "User does not exist."	Pass

Figure 70 Delete User (TC)

9.17 Shopkeeper Sign-up

Test Case ID	DCY_022				
Test Case Name	Shopkeeper Sign-Up	**Test Case Description**	To test if a shopkeeper can successfully sign up.		
Created By	Hussam	**Version**	1.1	**Date**	19 Mar 2025

S.no	**Prerequisites:**
1	The shopkeeper must not already have an account.
2	The system must be connected to the database.

Test Scenario	Verify that a shopkeeper can successfully register on the system.

Step no.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Shopkeeper enters valid details, selects shopkeeper option submits the sign-up form.	Shopkeeper is successfully registered and redirected to the dashboard.	Shopkeeper is successfully registered and redirected to the dashboard.	Pass
2	Shopkeeper enters an already registered email.	System displays an error: "Email already in use."	System displays an error: "Email already in use."	Pass
3	Shopkeeper submits the form with missing required fields.	System displays an error: "All fields are required."	System displays an error: "All fields are required."	Pass

Figure 71 Shopkeeper Sign-up (TC)

9.18 Add Car Part

1	**Test Case ID**	DCY_023				
2	**Test Case Name**	Add Car Part	**Test Case Description**	To test if a shopkeeper can add a car part listing.		
3	**Created By**	Roha Ali	**Version**	1.1	**Date**	19 Mar 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	The shopkeeper must be logged in.				
7	2	The shopkeeper must have access to the "Add Car Part" page.				
8						
9	**Test Scenario**	Verify that a shopkeeper can successfully add a car part listing.				
10						
11	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**	
12	1	Shopkeeper enters valid details and submits the form.	System saves the product listing as pending.	System saves the product listing as pending.	Pass	
13	2	Listing is rejected by the admin	System saves the Product listing as rejected and it can only be seen by the shopkeeper.	System saves the Product listing as rejected and it can only be seen by the shopkeeper.	Pass	
14	3	Shopkeeper submits the form with missing required fields.	System displays an error: "All fields are required."	System displays an error: "All fields are required."	Pass	
15	4	Listing is approved by the admin	System saves the product listing as approved and only on approval it is shown to other users.	System saves the product listing as approved and only on approval it is shown to other	Pass	
16						

Figure 72 Add Car Part (TC)

9.19 View Car Parts

	A	B	C	D	E	F	G	H	I	J
1	**Test Case ID**		DCY_024							
2	**Test Case Name**		View Car Parts	**Test Case Description**		To test if a shopkeeper can view all added car parts.				
3	**Created By**		Roha Ali	**Version**		1.1		**Date**	19 Mar 2025	
4										
5	**S.no**	**Prerequisites:**								
6	1	The shopkeeper must be logged in.								
7	2	At least one car part must be listed.								
8										
9	**Test Scenario**	Verify that a shopkeeper can successfully view all added car parts.								
10										
11	**Step no.**	**Step Details**		**Expected Results**		**Actual Results**		**Pass / Fail / Not executed / Suspended**		
12	1	Shopkeeper navigates to the inventory page.		System displays a list of all added car parts.		System displays a list of all added car parts.		Pass		
13	2	No car parts are listed in the inventory.		System displays a message: "No car parts found."		System displays a message: "No data."		Pass		
14	3	System encounters an error while fetching data.		System displays an error: "Unable to fetch inventory. Please try again."		System displays an error: "Unable to fetch inventory. Please try again."		Pass		

Figure 73 View Car Parts (TC)

9.20 Delete Car Part

Test Case ID	DCY_025					
Test Case Name	Delete Car Part	**Test Case Description**	To test if a shopkeeper can delete a car part listing.			
Created By	Roha Ali	**Version**	1.1	**Date**	19 Mar 2025	

S.no	Prerequisites:
1	The shopkeeper must be logged in.
2	At least one car part must be listed in the inventory.

Test Scenario	Verify that a shopkeeper can successfully delete a car part listing.

Step no.	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Shopkeeper selects a car part and clicks "Delete".	System removes the car part from the inventory.	System does not remove the car part from the inventory.	Fail
2	Shopkeeper cancels the deletion after confirmation prompt.	Car part remains in the inventory.	No confirmation shown.	Fail
3	System encounters an error during deletion.	System displays an error: "Error deleting car part. Please try again."	System does nothing.	Fail

Figure 74 Delete Car Part (TC)

Test Case ID	DCY_026					
Test Case Name	Delete Car Part	**Test Case Description**	To test if a shopkeeper can delete a car part listing.			
Created By	Roha Ali	**Version**	1.2	**Date**	02 Apr 2025	

S.no	Prerequisites:
1	The shopkeeper must be logged in.
2	At least one car part must be listed in the inventory.

Test Scenario	Verify that a shopkeeper can successfully delete a car part listing.

Step no.	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Shopkeeper selects a car part and clicks "Delete".	System removes the car part from the inventory.	System does not remove the car part from the inventory.	Fail
2	Shopkeeper cancels the deletion after confirmation prompt.	Car part remains in the inventory.	No confirmation shown.	Fail
3	System encounters an error during deletion.	System displays an error: "Error deleting car part. Please try again."	System does nothing.	Fail

Figure 75 Delete Car Part-2 (TC)

1	**Test Case ID**	DCY_027				
2	**Test Case Name**	Delete Car Part	**Test Case Description**	To test if a shopkeeper can delete a car part listing.		
3	**Created By**	Roha Ali	**Version**	1.3	**Date**	16 Apr 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	The shopkeeper must be logged in.				
7	2	At least one car part must be listed in the inventory.				
8						
9	**Test Scenario**	Verify that a shopkeeper can successfully delete a car part listing.				
10						

11	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
12	1	Shopkeeper selects a car part and clicks "Delete".	System removes the car part from the inventory.	System removes the car part from the inventory.	Pass
13	2	Shopkeeper cancels the deletion after confirmation prompt.	Car part remains in the inventory.	Car part remains in the inventory.	Pass
14	3	System encounters an error during deletion.	System displays an error: "Error deleting car part. Please try again."	System displays an error: "Error deleting car part. Please try again."	Pass

Figure 76 Delete Car Part-3 (TC)

9.21 Delete from Cart

1	**Test Case ID**	DCY_029				
2	**Test Case Name**	Delete from Cart	**Test Case Description**	To test if a user can remove a car part from their cart.		
3	**Created By**	Roha Ali	**Version**	1.1	**Date**	23 May 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	The user must be logged in.				
7	2	The cart must contain at least one item.				
8						
9	**Test Scenario**	Verify that a user can remove items from the cart.				
10						

11	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
12	1	User selects an item in cart and clicks "Delete".	System removes the car part from the inventory.		
13	2	System error occurs.	Error message is shown.		

Figure 77 Delete from Cart (TC)

9.22 Adjust Quantity in Cart

Test Case ID		DCY_030							
Test Case Name		Adjust Quantity in Cart	**Test Case Description**		To test if a user can change the quantity of items in the cart.				
Created By		Roha Ali	**Version**		1.1		**Date**	23 May 2025	
S.no	**Prerequisites:**								
1	The user must be logged in.								
2	The cart must contain items.								
Test Scenario	Verify that a user can adjust the quantity of items in their cart.								
Step no.	**Step Details**		**Expected Results**		**Actual Results**		**Pass / Fail / Not executed / Suspended**		
1	User increases quantity of item.		Quantity increases.						
2	User decreases quantity.		Quantity decreases or item removed if 0.						
3	Error occurs.		Error message is shown.						

Figure 78 Adjust Quantity in Cart (TC)

9.23 View Cart

Test Case ID		DCY_031							
Test Case Name		View Cart	**Test Case Description**		To test if a user can view the items in their cart.				
Created By		Roha Ali	**Version**		1.1		**Date**	23 May 2025	
S.no	**Prerequisites:**								
1	The user must be logged in.								
Test Scenario	Verify that a user can view the items they have added to their cart.								
Step no.	**Step Details**		**Expected Results**		**Actual Results**		**Pass / Fail / Not executed / Suspended**		
1	User navigates to cart page.		Items in cart are displayed.						
2	Cart is empty.		Message "Cart is empty" displayed.						

Figure 79 View Cart (TC)

Test Case ID		DCY_033							
Test Case Name		View Cart	**Test Case Description**		To test if a user can view the items in their cart.				
Created By		Roha Ali	**Version**		1.2		**Date**	23 May 2025	
S.no	**Prerequisites:**								
1	The user must be logged in.								
Test Scenario	Verify that a user can view the items they have added to their cart.								
Step no.	**Step Details**		**Expected Results**		**Actual Results**		**Pass / Fail / Not executed / Suspended**		
1	User navigates to cart page.		Items in cart are displayed.		Items in cart are displayed.		Pass		
2	Cart is empty.		Message "Cart is empty" displayed.		Message "Cart is empty" displayed.		Pass		

Figure 80 View Cart-2 (TC)

9.24 Checkout (COD)

1	**Test Case ID**	DCY_032				
2	**Test Case Name**	Checkout (COD)	**Test Case Description**	To test if a user can checkout using Cash on Delivery.		
3	**Created By**	Roha Ali	**Version**	1.1	**Date**	23 May 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	User must be logged in.				
7	2	Cart must have at least one item.				
8						
9	**Test Scenario**	Verify that a user can place an order using Cash on Delivery.				
10						
11	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**	
12	1	User selects "Checkout".	Checkout page opens.			
13	2	User provides delivery address and any additional information.	Address and info saved for order.			
14	3	User selects COD.	Order is placed with COD selected.			

Figure 81 Checkout (COD) (TC)

9.25 View all Orders

1	**Test Case ID**	DCY_033				
2	**Test Case Name**	View All Orders	**Test Case Description**	To test if shopkeeper can view all received orders.		
3	**Created By**	Roha Ali	**Version**	1.1	**Date**	23 May 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	Shopkeeper must be logged in.				
7						
8						
9	**Test Scenario**	Verify that a shopkeeper can see all received orders.				
10						
11	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**	
12	1	Shopkeeper opens orders section.	List of orders is displayed.			
13	2	There are no orders.	System displays a message: "No orders."			

Figure 82 View all Orders (TC)

9.26 View Order Details

1	**Test Case ID**	DCY_034				
2	**Test Case Name**	View Order Details	**Test Case Description**	To test if shopkeeper can view specific order details.		
3	**Created By**	Roha Ali	**Version**	1.1	**Date**	23 May 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	Shopkeeper must be logged in.				
7	2	There must be at least one order.				
8						
9	**Test Scenario**	Verify that shopkeeper can see complete order info.				
10						
11	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**	
12	1	Shopkeeper selects an order.	Order details are displayed.			

Figure 83 View Order Details (TC)

9.27 Add to cart

	A	B	C	D	E	F	G	H	I	J
1	**Test Case ID**		DCY_028							
2	**Test Case Name**		Add to Cart	**Test Case Description**		To test if a user can add a car part to their cart.				
3	**Created By**		Roha Ali	**Version**		1.1		**Date**	23 May 2025	
4										
5	**S.no**	**Prerequisites:**								
6	1	The user must be logged in.								
7	2	At least one car part must exist in the marketplace.								
8										
9	**Test Scenario**	Verify that a user can successfully add a car part to their cart.								
10										
11	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**					
12	1	User selects a car part and clicks "Add to Cart".	Car part is added to the cart.	Car part is not added to the cart.	Fail					
13	2	User tries to add the same part again.	Quantity in cart is updated.	Quantity in cart is not updated.	Fail					
14	3	System error during add.	Error message is displayed.	No error message.	Fail					

Figure 84 Add to Cart (TC)

1	**Test Case ID**	DCY_029				
2	**Test Case Name**	Add to Cart	**Test Case Description**	To test if a user can add a car part to their cart.		
3	**Created By**	Roha Ali	**Version**	1.2	**Date**	23 May 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	The user must be logged in.				
7	2	At least one car part must exist in the marketplace.				
8						
9	**Test Scenario**	Verify that a user can successfully add a car part to their cart.				
10						

11	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
12	1	User selects a car part and clicks "Add to Cart".	Car part is added to the cart.	Car part is added to the cart.	Pass
13	2	User tries to add the same part again.	Quantity in cart is updated.	Quantity in cart is updated.	Pass
14	3	System error during add.	Error message is displayed.	Error message is displayed.	Pass

Figure 85 Add to Cart-2 (TC)

9.28 Cancel an Order

1	**Test Case ID**	DCY_037				
2	**Test Case Name**	Cancel an Order	**Test Case Description**	To test if shopkeeper can cancel an order.		
3	**Created By**	Roha Ali	**Version**	1.1	**Date**	23 May 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	Shopkeeper must be logged in.				
7	2	At least one order must exist.				
8						
9	**Test Scenario**	Verify that shopkeeper can cancel order and notify user.				
10						

11	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
12	1	Shopkeeper selects an order an clicks cancel.	Order is marked as cancelled.	No option to cancel order.	Fail
13	2	User is notified.	User is shown a message for cancellation.	No message	Fail
14	3	Error during cancellation.	Error message is shown.	No error	Fail

Figure 86 Cancel an Order (TC)

1	**Test Case ID**	DCY_038				
2	**Test Case Name**	Cancel an Order	**Test Case Description**	To test if shopkeeper can cancel an order.		
3	**Created By**	Roha Ali	**Version**	1.2	**Date**	23 May 2025
4						
5	**S.no**	**Prerequisites:**				
6	1	Shopkeeper must be logged in.				
7	2	At least one order must exist.				
8						
9	**Test Scenario**	Verify that shopkeeper can cancel order and notify user.				
10						
11	**Step no.**	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**	
12	1	Shopkeeper selects an order an clicks cancel.	Order is marked as cancelled.	Order is marked as cancelled.	Pass	
13	2	User is notified.	User is shown a message for cancellation.	User is shown a message for cancellation.	Pass	
14	3	Error during cancellation.	Error message is shown.	Error message is shown.	Pass	

Figure 87 Cancel an Order-2 (TC)

User Manual

User Guide

1. Introduction

Welcome to the official user guide for Digital Car-Yard. This guide is designed to provide comprehensive information about the functionalities, features, and services offered by Digital Car-Yard. Whether you're a new user or returning, this guide will help you understand how to navigate and utilize the full scope of our platform to simplify vehicle- related activities such as buying, selling, maintaining, and upgrading vehicles.

2. Overview of Services

1. Core Services

At Digital Car-Yard, we focus on delivering a broad range of services tailored to meet diverse customer needs. Our core services include:

Service	Description	Availability	Pricing Model	Customer Support
Vehicle Listings	Sell and view new/used cars with detailed information	24/7	Free	Chat & Email Support
Maintenance Reminder	Automated notifications for oil change, tire rotation, etc.	24/7	Free	Chat Support
Car Parts Marketplace	Buy/sell vehicle parts with integrated checkout and cart	24/7	Pay per Item	Chat & Email Support

Figure 88 Core Services Table - (UM)

These services are continuously updated to ensure that we meet the evolving demands of our customers in 2025.

In addition to our core offerings, Digital Car-Yard supports:

• **Real-Time Chat** with shopkeepers and admins for inquiries
• **System Analytics** for admins to monitor platform activity
• **Inventory Management** tools for shopkeepers
• **Vehicle Maintenance History Tracking**

3. Getting Started

1. Account Creation

To begin using Digital Car-Yard:

• Visit our website.
• Click the "Sign Up" button.
• Provide your name, email, phone number, and password.
• Verify your email to activate the account.
• Log in using your email and password.

Once verified, you can log in and access the full suite of services offered by Digital Car- Yard.

2. Setting Up Your Profile

Once registered, personalize your profile by:

• Adding a profile picture
• Providing vehicle ownership details
• Setting up maintenance preferences

4. Using the Platform

1. Navigation

The **Digital Car-Yard** platform is designed with user experience in mind. The dashboard includes:

Section	Functionality	Icon
Home	Access listings and reminders	
Marketplace	Browse car parts and accessories	
Profile	Manage your information	
Chat	Real-time support from admins/shopkeepers	

Figure 89 Navigation Table - (UM)

2. Accessing Services

- Use the top navigation bar or side panel to switch between features.
- Click List Car to sell a vehicle, Marketplace to shop parts, or maintenance to set reminders.
- Services to reflect your role (User, Admin, Shopkeeper) upon login.

5. Troubleshooting and Support

1. Common Issues

Should you encounter any issues while using **Digital Car-Yard,** refer to the table below for common solutions:

Issue	Description	Solution
Login Failure	Invalid email/password	Use "Forgot Password" to reset credentials
Marketplace Empty	No items listed yet	Check back later or refresh the page
Missing Car Info	Incomplete car listing	Edit and complete required fields
Cart Errors	Issues adding/removing items from cart	Retry or contact support

Figure 90 Common Issues Table - (UM)

2. Contacting Support

For unresolved issues

- Email us at info@digitalcaryard.com
- Call +92-111-11111
- Use the in-app chat support

We aim to respond within 24 hours

6. Legal and Privacy Information

By using **Digital Car-Yard,** user agree to our **terms of services** and **privacy policy.** We implement industry-standard security measures, including **data encryption**, to safeguard user data and transactions. For full legal information, visit the **private policy** section on our website.

7. Conclusion

we hope this guide has helped you understand the features and usage of Digital Car-yard. Our mission is to offer a seamless, user-friendly solution for all your vehicle related needs. For further assistance, reach out to our support.

USER INTERFACES

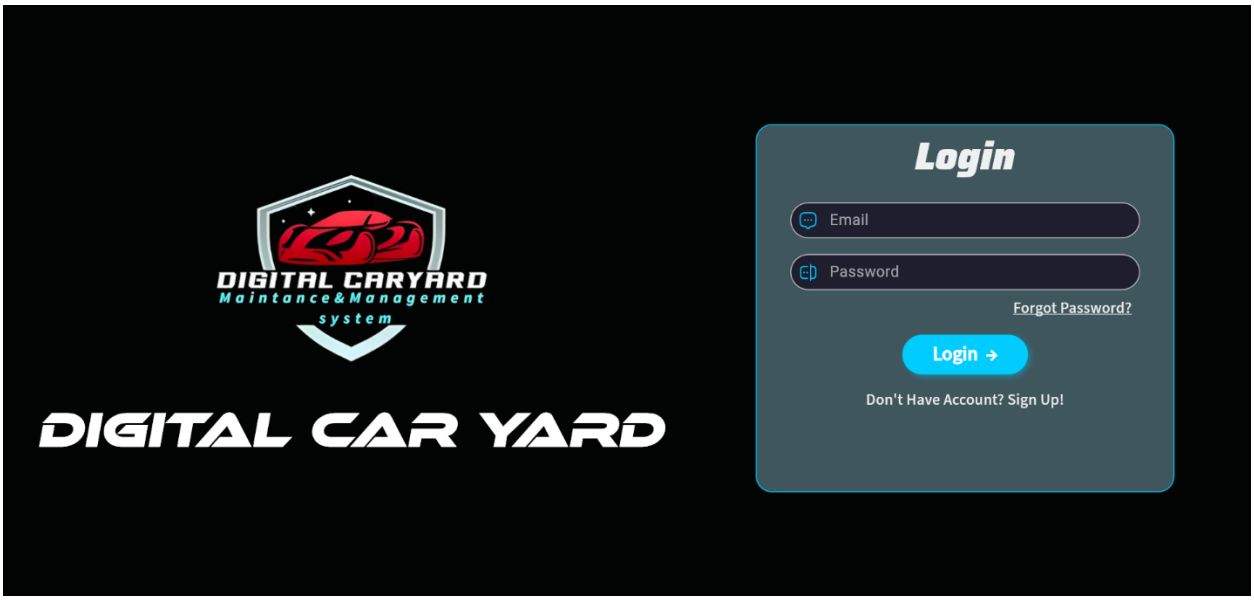

Figure 91 Login (UI)

Figure 92 Signup (UI)

Figure 93 Add Reminder (UI)

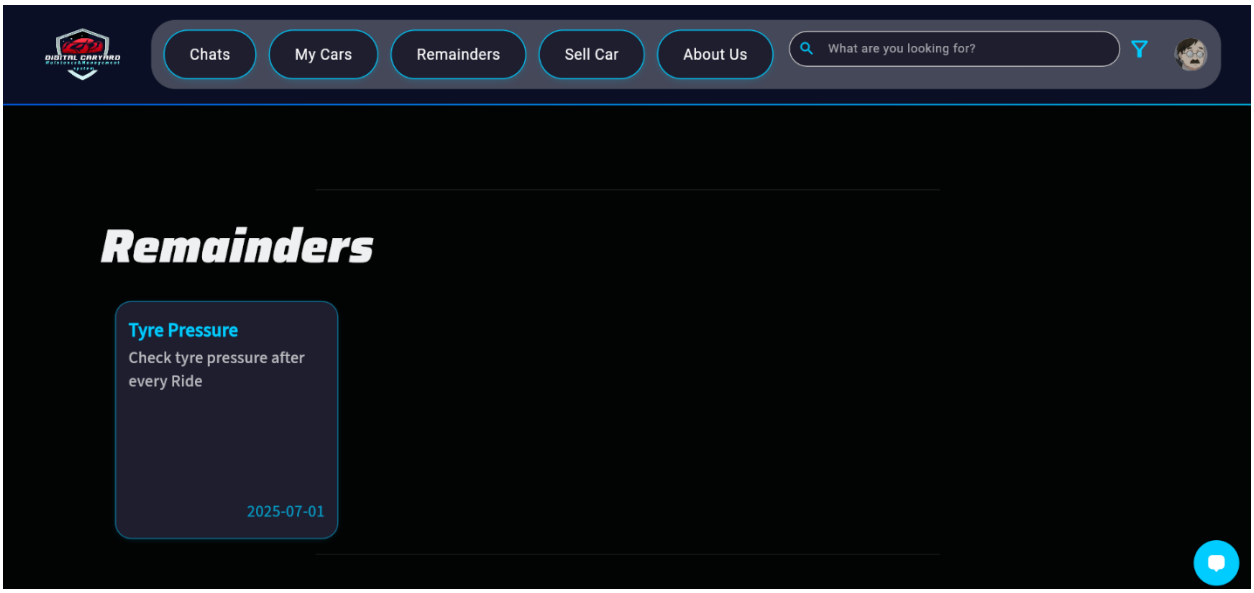

Figure 94 Reminder (UI)

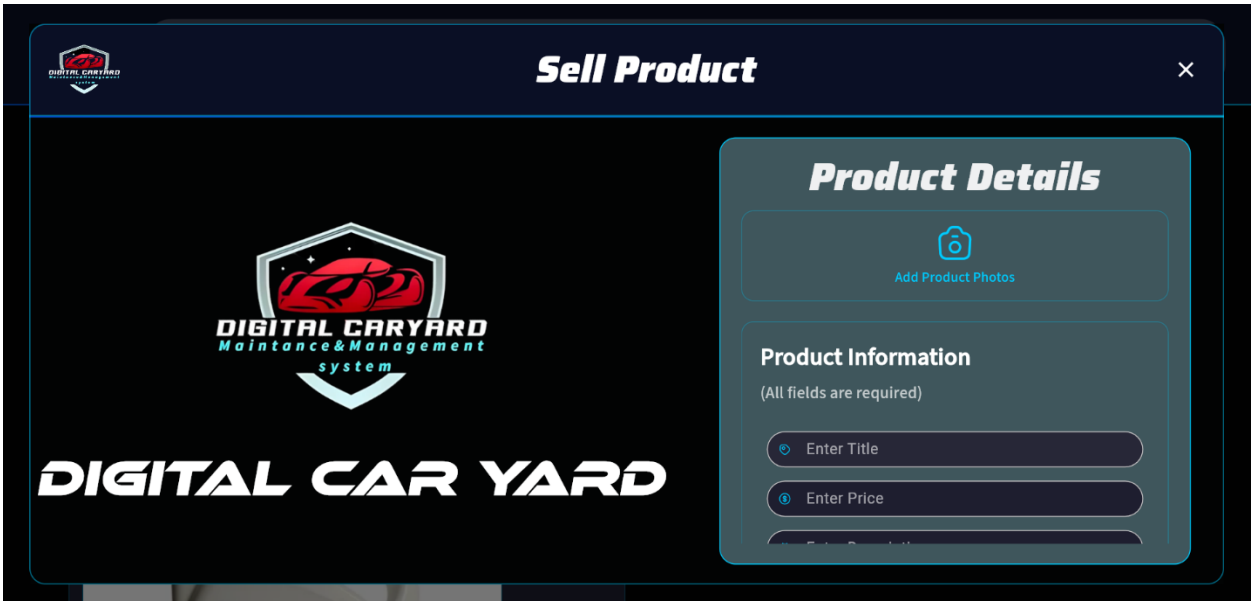

Figure 95 Sell Product (UI)

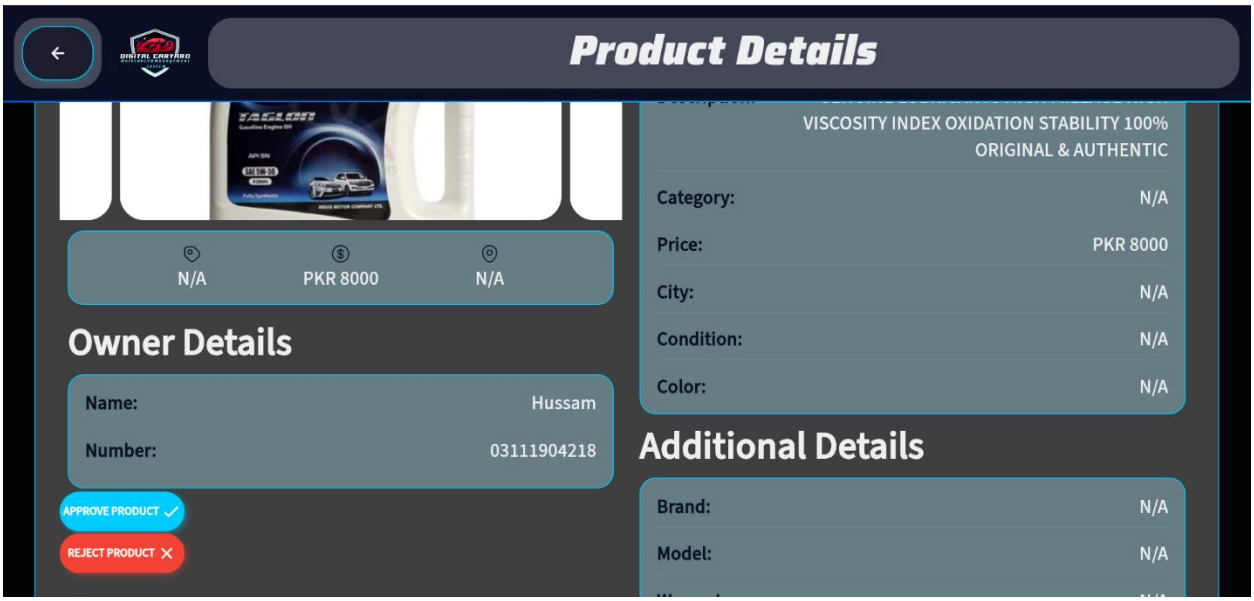

Figure 96 Product View (UI)

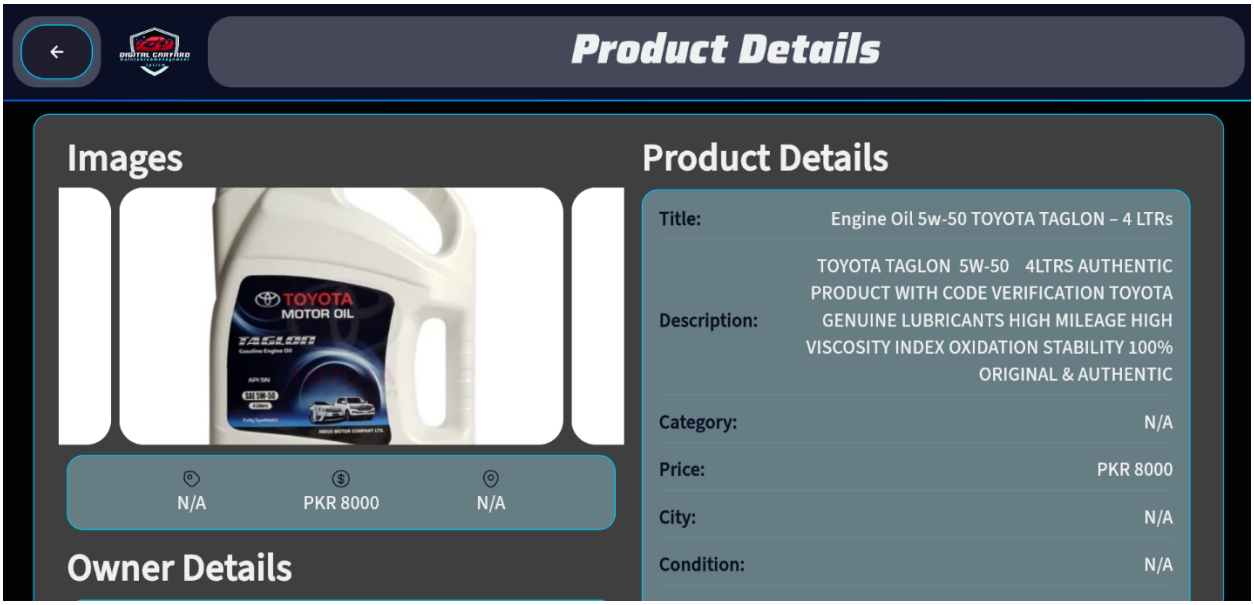

Figure 97 Product Details (UI)

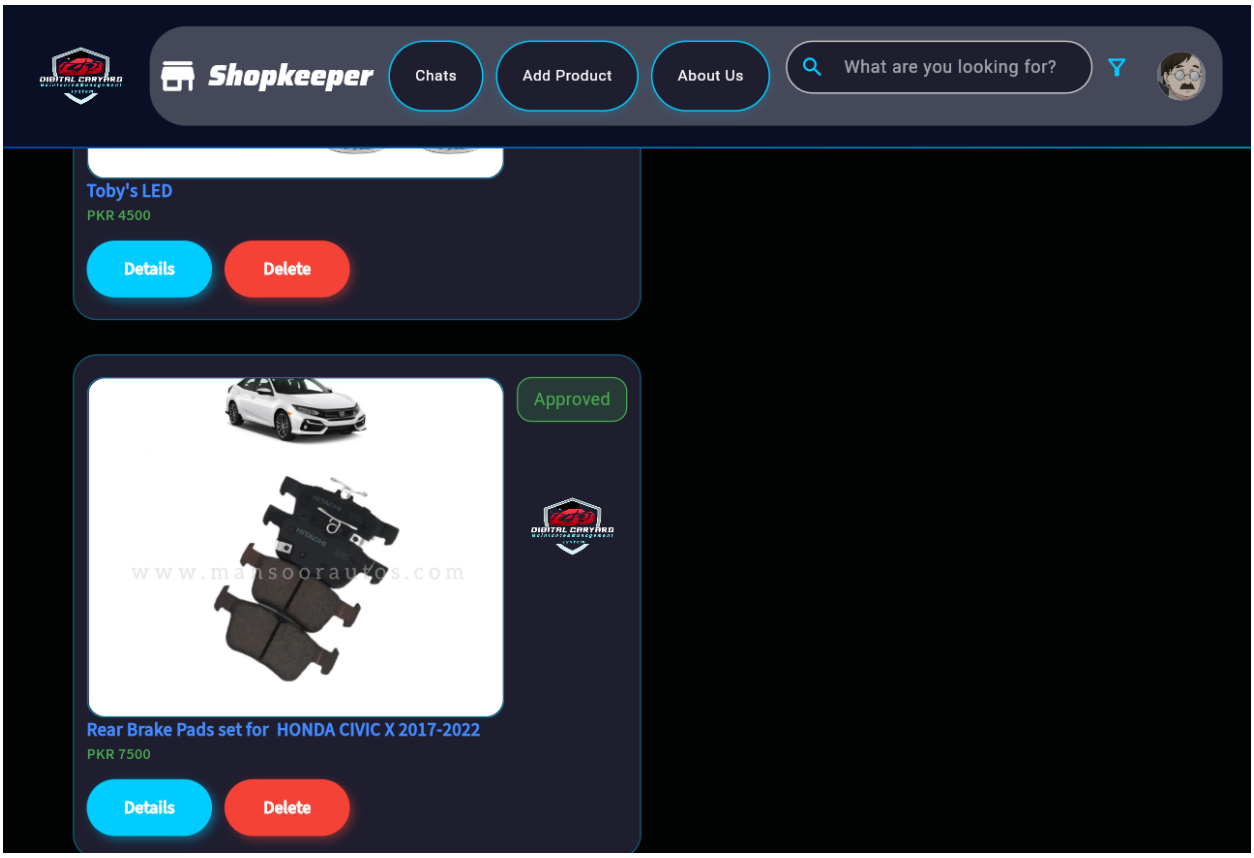

Figure 98 Shopkeeper View (UI)

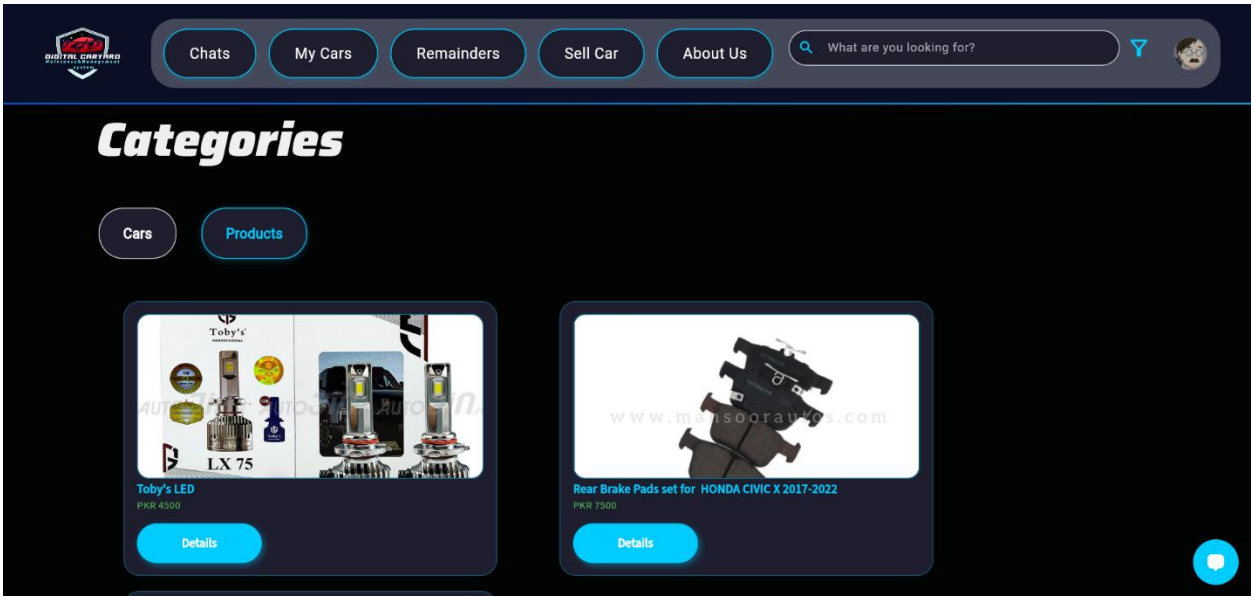

Figure 99 Product Categories (UI)

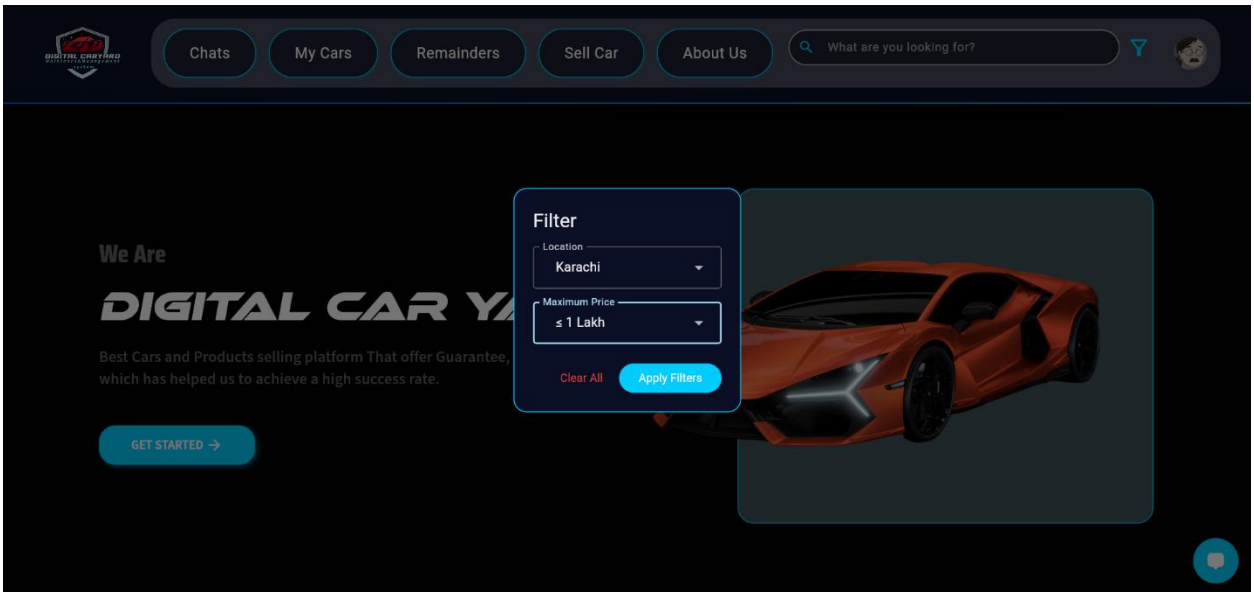

Figure 100 Filter (UI)

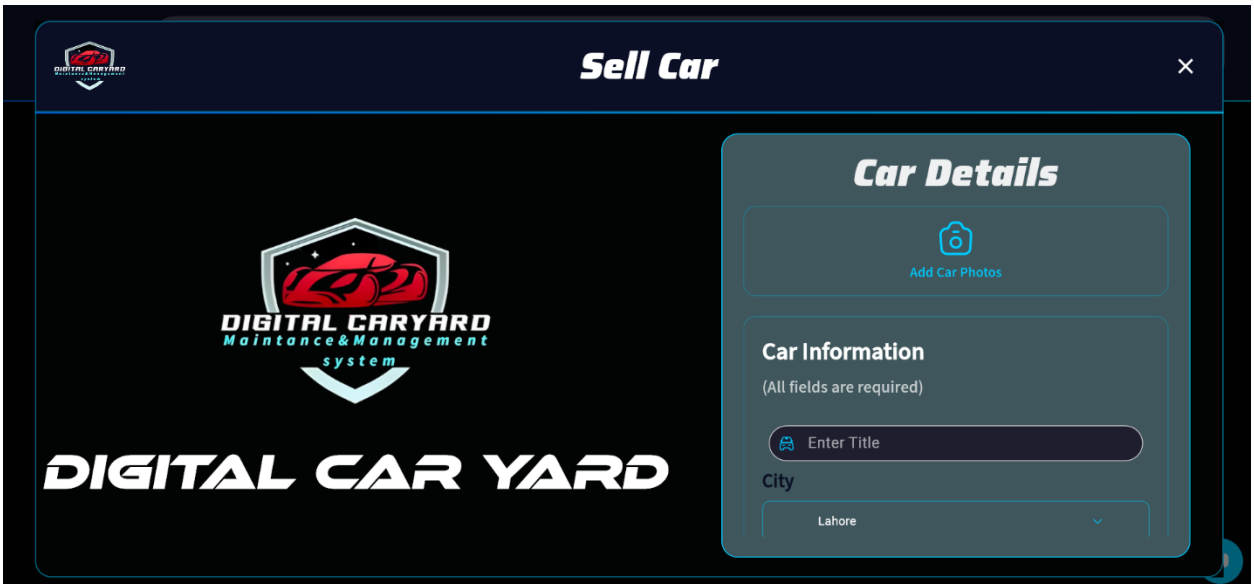

Figure 101 Car Details (UI)

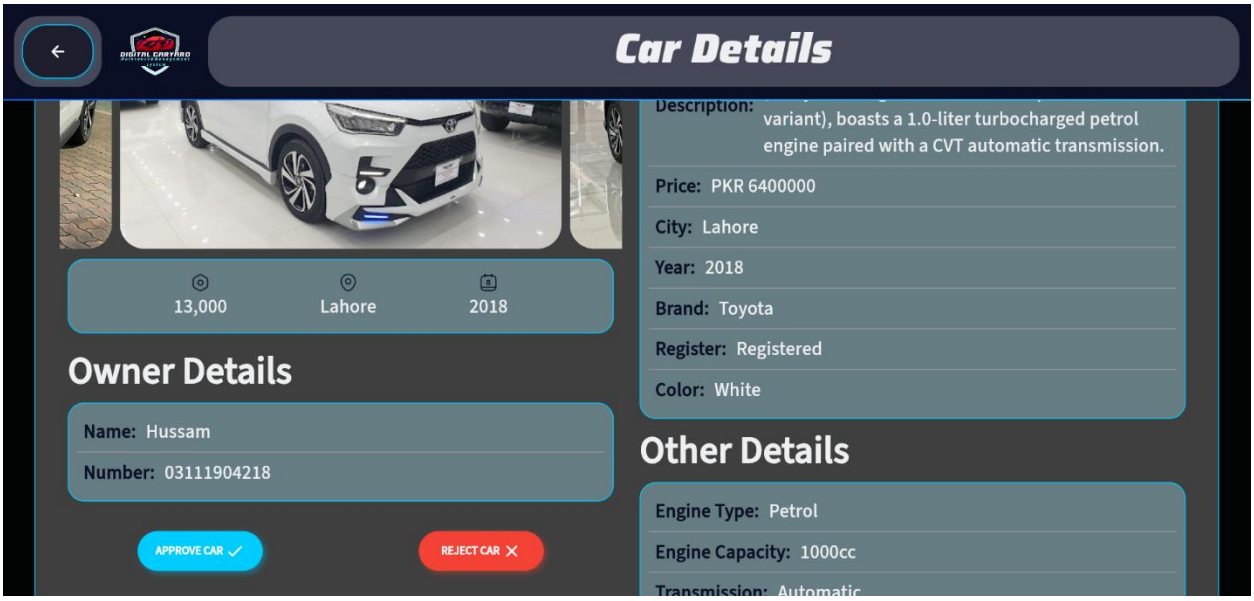

Figure 102 Car Details (UI)

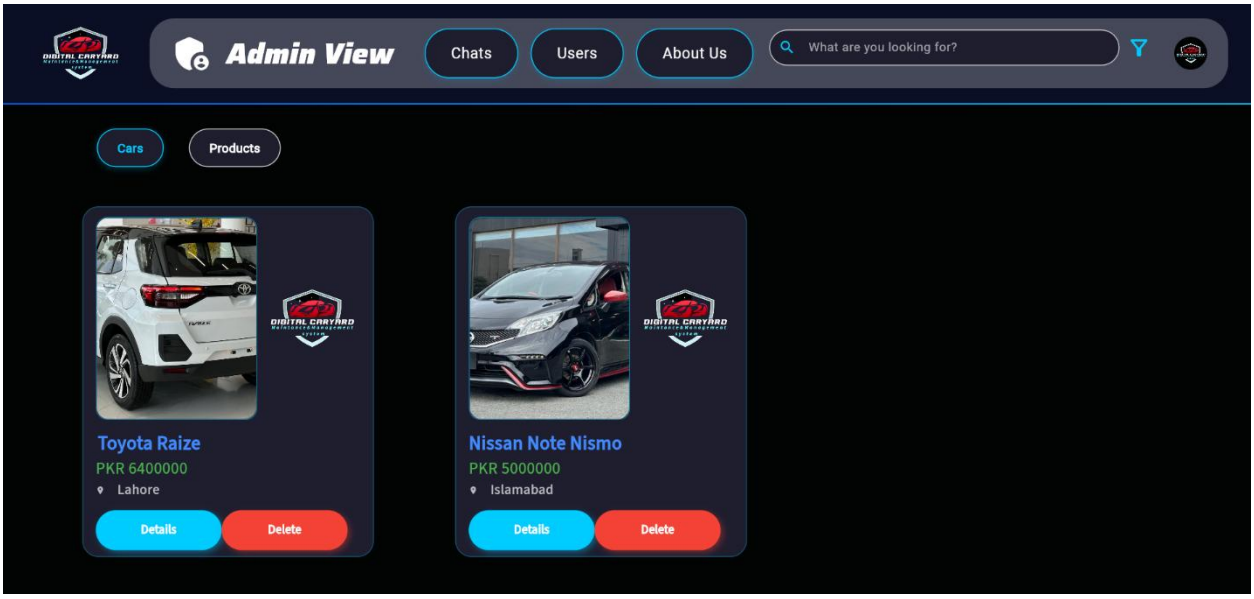

Figure 103 Approval Admin View (UI)

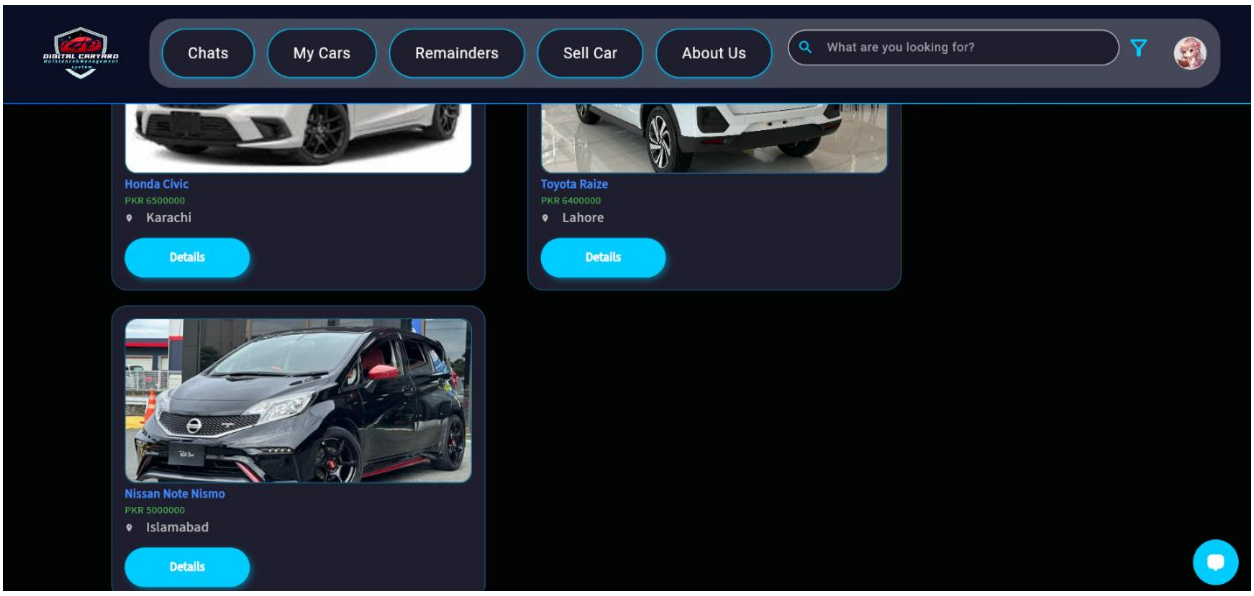

Figure 104 Car View (UI)

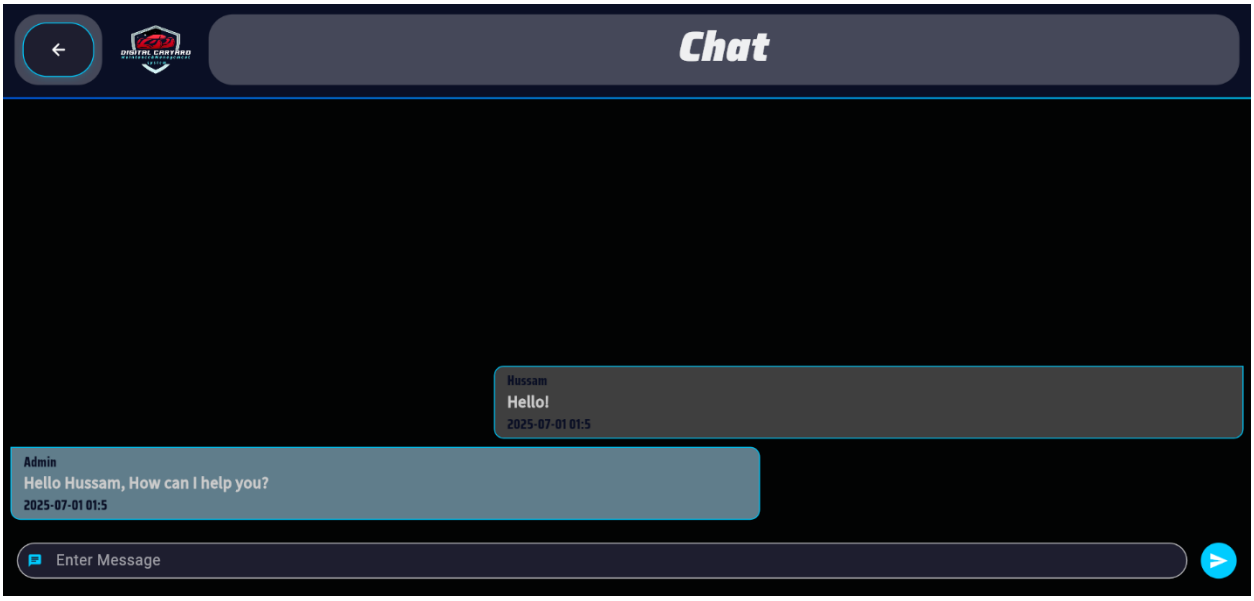

Figure 105 Chat (UI)

Appendix A: Glossary

Sr.no	Terms	Description
1	API	Application Programming Interface – A set of protocols and tools for building software applications and facilitating communication between different systems.
2	MongoDB	A relational database management system used for storing and managing application data.
3	RESTful API	Representational State Transfer – A web service architecture that allows interaction with REST-based web services.
4	HTTPS	Hypertext Transfer Protocol Secure – A secure version of HTTP used for encrypted communication over a computer network.
5	UI	User Interface – Refers to the design and layout through which users interact with the app.
6	UX	User Experience – Refers to the overall experience of a user when interacting with the app.
7	S/RS	Stimulus/Response Sequence – Refers to the stimulus/response sequence tables.
8	FR	Functional Requirements – Refers to the functional requirements tables.
9	CT	Component Table – Refers to the component tables that describe the components of this project in detail.
10	SMD	State Machine Diagram – Refers to the state machine diagrams for different system features.
11	AD	Activity Diagram – Refers to the activity diagrams for different system features.
12	SD	Sequence Diagram – Refers to the sequence diagrams for different system features.
13	CD	Collaboration Diagram – Refers to the collaboration diagrams for different system features.
14	TC	Test Case – Refers to the test cases done manually for all the system features completed.

Table 90 Glossary

Appendix B: Design Diagram

ERD for FYP-| & FYP-||

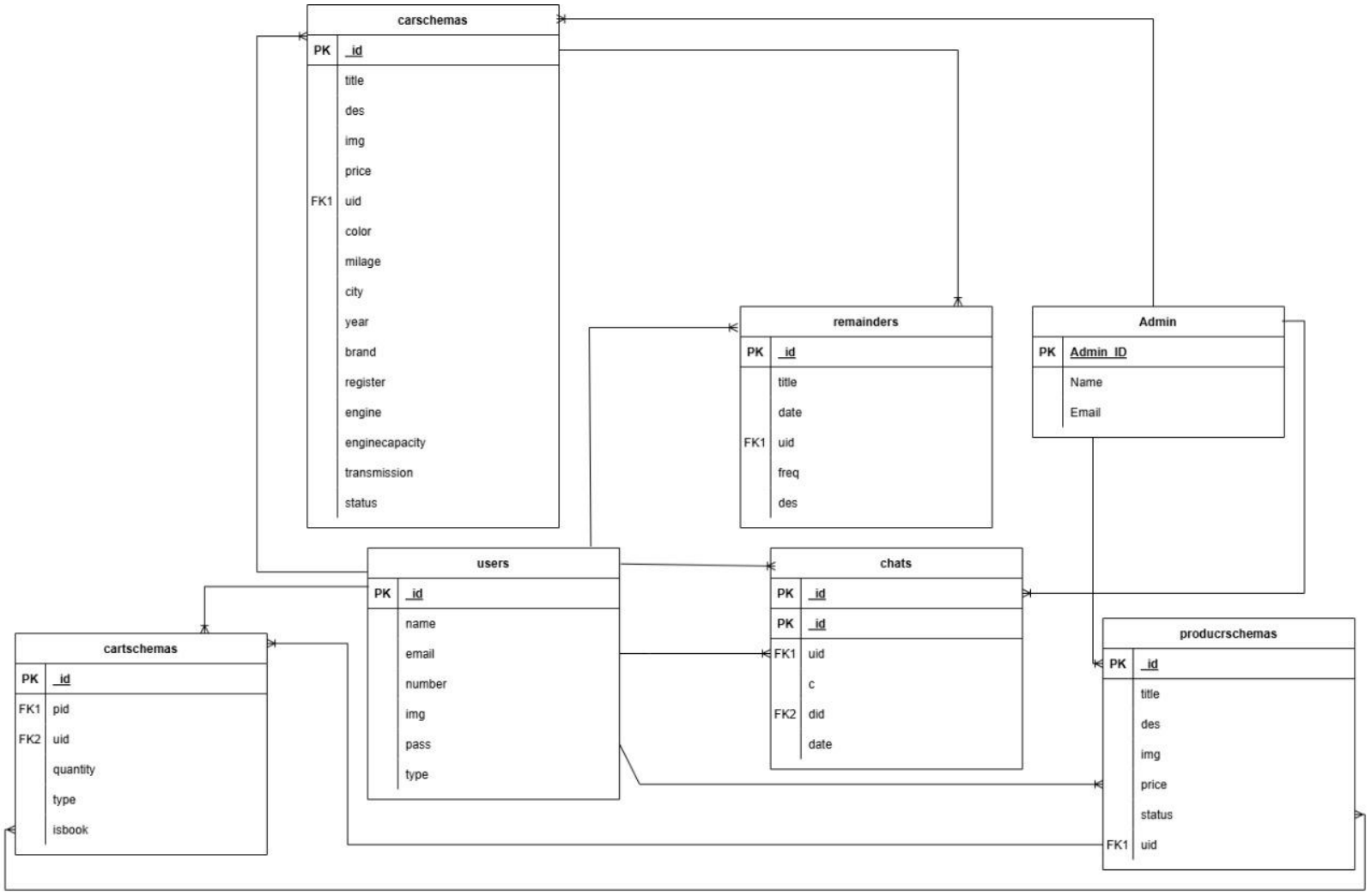

Figure 106 ERD

References

1. Oracle Corporation. "MySQL Documentation". September 1, 2023. Oracle. September 30, 2024. https://dev.mysql.com/doc/

2. **Automotive News**. “Subscription Models”. 2023. Automotive News. September 17, 2024. http://www.automotivenews.com/subscription-models

3. NestJS. "A progressive Node.js framework". August 30, 2023. NestJS. September 29, 2024. https://nestjs.com/

4. Google Developers. "Responsive Web Design Fundamentals". September 1, 2023. Google. October 1, 2024. https://developers.google.com/web/fundamentals/design-and-ux/responsive

5. IBM Cloud Education. "Chatbots in the Automotive Industry". August 25, 2023. IBM. September 29, 2024. https://www.ibm.com/cloud/learn/chatbots-explained

6. Heroku. "Cloud Application Platform". September 5, 2023. Salesforce. October 1, 2024. https://www.heroku.com/

7. Atlassian. "Trello: Manage Your Team's Projects". August 20, 2023. Atlassian. October 1, 2024. https://trello.com/