COMSATS University Islamabad (CUI)

Software Design Description
(SDS DOCUMENT)

for

SafarRehnuma

Version 1.0

By

Muhammad Qasim Tariq **CIIT/SP21-BCS-038/ISB**
Muhammad Usman Ahmad **CIIT/SP21-BCS-071/ISB**

Supervisor
Mr. Qasim Malik

Bachelor of Science in Computer Science (2021-2025)

Table of Contents

1. Introduction 1
1.1 Modules 1
1.1.1 User Management 1
1.1.2 Vehicle Registration 1
1.1.3 Ride Management 1
1.1.4 Dynamic Pricing 1
1.1.5 Location Tracking 1
1.1.6 Route Optimization and Suggestions 1
1.1.7 Booking Management 1
1.1.8 Payments 1
1.1.9 Reviews and Ratings 1
1.1.10 Notifications 2
1.1.11 Rewards 2
2. Design Methodology and Software Process Model 2
2.1 Design Methodology 2
2.2 Software Process Model 2
3. System Overview 3
3.1 Functionality 3
3.2 Context 3
3.3 Design 3
3.4 Architectural Design 3
4. Design Models 6
4.1 Activity Diagrams 6
4.1.1 Activity Diagram for Module 1: User Management 6
4.1.2 Activity Diagram for Module 2: Vehicle Registration 7
4.1.3 Activity Diagram for Module 3: Ride Management 8
4.1.4 Activity Diagram for Module 4: Dynamic Pricing 9
4.1.5 Activity Diagram for Module 5: Location Tracking 10
4.1.6 Activity Diagram for Module 6: Route Optimization and Suggestions 11
4.1.7 Activity Diagram for Module 7: Booking Management 12
4.1.8 Activity Diagram for Module 8: Payments 13
4.1.9 Activity Diagram for Module 9: Reviews and Ratings 14
4.1.10 Activity Diagram for Module 10: Notifications 15

4.1.11 Activity Diagram for Module 11: Rewards 16
4.2 Class Diagram 17
4.3 Sequence Diagrams 18
4.3.1 Module 1: User Management 18
4.3.2 Module 2: Vehicle Registration 20
4.3.3 Module 3: Ride Management 21
4.3.4 Module 4: Dynamic Pricing 22
4.3.5 Module 5: Location Tracking 23
4.3.6 Module 6: Route Optimization and Suggestions 25
4.3.7 Module 7: Booking Management 26
4.3.8 Module 8: Payments 28
4.3.9 Module 9: Reviews and Ratings 29
4.3.10 Module 10: Notifications 31
4.3.11 Module 11: Rewards 33
4.4 State Transition Diagrams 35
4.4.1 State Transition Diagram for Dynamic Pricing 35
4.4.2 State Transition Diagram for Route Suggestion 35
4.4.3 State Transition Diagram for Ride Booking 35
5. Data Design 36
5.1 Data Dictionary 36
5.2 Schema 37
5.2.1 User Schema 37
5.2.2 Rides Schema 38
5.2.3 Bookings Schema 38
5.2.4 Payments Schema 39
5.2.5 Vehicles Schema 39
5.2.6 Feedback Schema 40
6. Human Interface Design 42
6.1 Screen Images 42
6.1.1 Splash Screen 42
6.1.2 Welcome Screen 43
6.1.3 Sign Up Screen 44
6.1.4 Home Screen 45
6.1.5 Search Vehicles Screen 46
6.1.6 Available Vehicles Screen 47
6.1.7 Request Ride Screen 48

6.1.8 Review Screen 49
6.1.9 Wallet Screen 50
6.1.10 Contact Us Screen 51
6.1.11 Registered Vehicles Screen 52
6.1.12 Admin Dashboard Screen (Web) 53
6.2 Screen Objects and Actions 54
6.2.1 Splash Screen 54
6.2.2 Welcome Screen 54
6.2.3 Signup Screen 54
6.2.4 Home Screen 54
6.2.5 Search Vehicles Screen 54
6.2.6 Available Vehicles Screen 54
6.2.7 Request Ride Screen 54
6.2.8 Review Screen 54
6.2.9 Wallet Screen 55
6.2.10 Contact Us Screen 55
6.2.11 Registered Vehicles Screen 55
6.2.12 Admin Dashboard Screen (Web) 55
7. Implementation 56
7.1 Algorithms 56
7.1.1 Algorithm: Dynamic Pricing 56
7.1.2 Algorithm: Route Optimization and Suggestions 58
7.2 External APIs 60
7.3 Deployment 60
8. Testing and Evaluation 61
8.1 Unit Testing 61
8.1.1 Unit Testing 1: Home Screen 61
8.1.2 Unit Testing 2: User Management 62
8.1.3 Unit Testing 3: Vehicle Registration 62
8.1.4 Unit Testing 4: Ride Management 63
8.1.5 Unit Testing 5: Payments 64
8.2 Functional Testing 64
8.2.1 Functional Testing 1: User Registration 64
8.2.2 Functional Testing 2: User Login 65
8.2.3 Functional Testing 3: Register a Vehicle 65
8.2.4 Functional Testing 4: Book a Ride 66

8.2.5 Functional Testing 5: Cancel Booking 67
8.2.6 Functional Testing 6: Make Payment 67
8.2.7 Functional Testing 7: Submit Review 68
8.3 Integration Testing 69
8.3.1 Integration Testing 1: Vehicle Registration 69
8.3.2 Integration Testing 2: Dynamic Pricing 69
8.3.3 Integration Testing 3: Location Tracking 70
8.3.4 Integration Testing 4: Route Optimization and Suggestions 71

Revision History

Name	Date	Reason for changes	Version

Application Evaluation History

Comments (by committee)	Action Taken

Supervised by

Mr. Qasim Malik

Signature_____

1. Introduction

The SafarRehnuma project aims to create an advanced, integrated urban transportation system tailored for Pakistani cities. The primary goal is to streamline public and local transportation through a user-friendly platform that addresses the inefficiencies, safety concerns, and income instability prevalent in the current system.

1.1 Modules

This subsection lists the modules of SafarRehnuma, along with their functionalities.

1.1.1 User Management

Manages registration, authentication, and profile updates for passengers and drivers.

1.1.2 Vehicle Registration

Allows drivers to register and manage their vehicles on the platform.

1.1.3 Ride Management

Enables drivers to manage their availability, accept bookings, and navigate routes.

1.1.4 Dynamic Pricing

Adjusts fare prices dynamically based on demand, distance, and time factors.

1.1.5 Location Tracking

Provides real-time GPS tracking and location sharing for vehicles.

1.1.6 Route Optimization and Suggestions

Offers optimized and alternative routes using historical data and current traffic conditions.

1.1.7 Booking Management

Manages the selection, booking, and modification of vehicle reservations.

1.1.8 Payments

Facilitates secure transactions through various payment methods and supports refund processing.

1.1.9 Reviews and Ratings

Collects and manages feedback from users to maintain service standards.

1.1.10 Notifications

Sends real-time updates, trip alerts, and promotional messages to users.

1.1.11 Rewards

Implements reward programs to incentivize and retain users through points and benefits.

2. Design Methodology and Software Process Model

This section outlines the chosen design methodology and software process model.

2.1 Design Methodology

SafarRehnuma employs Object-Oriented Programming (OOP) as its design methodology due to its ability to manage complex software systems effectively. OOP facilitates modularity, reusability, and maintainability by organizing software components into reusable objects, each encapsulating data and behavior. This approach promotes clear code structures, code reusability, and scalability, making it suitable for SafarRehnuma's interconnected modules and entities. By representing real-world entities as objects with attributes and methods, OOP ensures a structured and intuitive design that aligns well with the project's complexity and requirements.

2.2 Software Process Model

SafarRehnuma follows the Agile Development process model for its flexibility, adaptability, and iterative nature. Agile emphasizes collaboration, customer involvement, and incremental delivery, ensuring early and continuous delivery of valuable software. It accommodates changes in requirements and facilitates quick response to evolving needs, enabling efficient adaptation to new requirements. By breaking down the project into small, manageable iterations, Agile promotes collaboration, flexibility, and customer satisfaction throughout the software development lifecycle.

This figure illustrates the agile development process model and its stages.

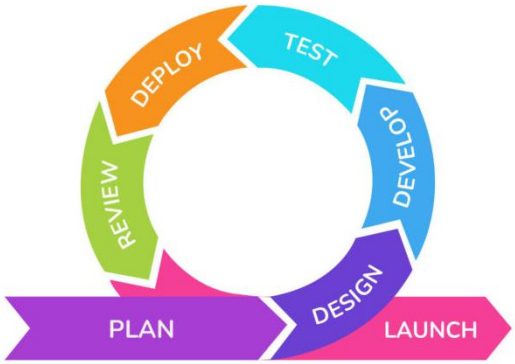


Figure 1 Agile Development Model

3. System Overview

This section provides a high-level summary of the overall system architecture and functionalities of SafarRehnuma.

3.1 Functionality

SafarRehnuma is a robust urban transportation platform designed to streamline commuting in Pakistani cities by providing features such as user management for passengers, drivers, and contractors; ride booking, tracking, and fare calculation with optimized routes; dynamic pricing based on real-time demand and supply; support for multiple payment methods including card, wallet, and cash; real-time notifications about ride status and traffic conditions; a reward system for accumulating and redeeming points; and comprehensive admin tools for overseeing system operations and user management.

3.2 Context

Operating within the urban transportation ecosystem of Pakistan, SafarRehnuma addresses challenges like congestion, safety, and pricing transparency by integrating with existing transportation infrastructure and offering additional functionalities. The platform aims to enhance the commuting experience for both passengers and drivers by leveraging advanced technologies for route optimization and dynamic pricing, ensuring efficient, safe, and transparent transportation services.

3.3 Design

SafarRehnuma utilizes a multi-tiered architecture comprising the Presentation Layer for user interfaces, the Business Logic Layer for core functionalities like user and ride management, and the Data Access Layer for secure data interactions. This design promotes modularity, scalability, and security, with APIs facilitating seamless module communication. Advanced algorithms, powered by machine learning, enable dynamic pricing and route optimization, allowing the system to adapt to user behavior and real-time conditions effectively.

3.4 Architectural Design

The SafarRehnuma platform is designed with a modular architecture to ensure scalability, maintainability, and ease of integration. The system is decomposed into several key modules: User Management, Ride Management, Payment Processing, Notification System, Reward System, and Admin Management. These modules collaborate through well-defined interfaces to provide a seamless user experience. The architecture follows a multi-tiered pattern, separating the

presentation layer, business logic layer, and data access layer. This separation ensures that changes in one layer do not affect others, facilitating easier updates and maintenance.

The User Management module handles user registration, authentication, and profile management. Ride Management oversees ride booking, ride tracking, and fare calculation. Payment Processing deals with various payment methods and ensures secure transactions. The Notification System sends real-time alerts and updates to users. The Reward System manages the accumulation and redemption of reward points. Admin Management allows administrators to oversee system operations and manage users and rides.

.

This figure illustrates the box and line diagram to represent the SafarRehnuma system.

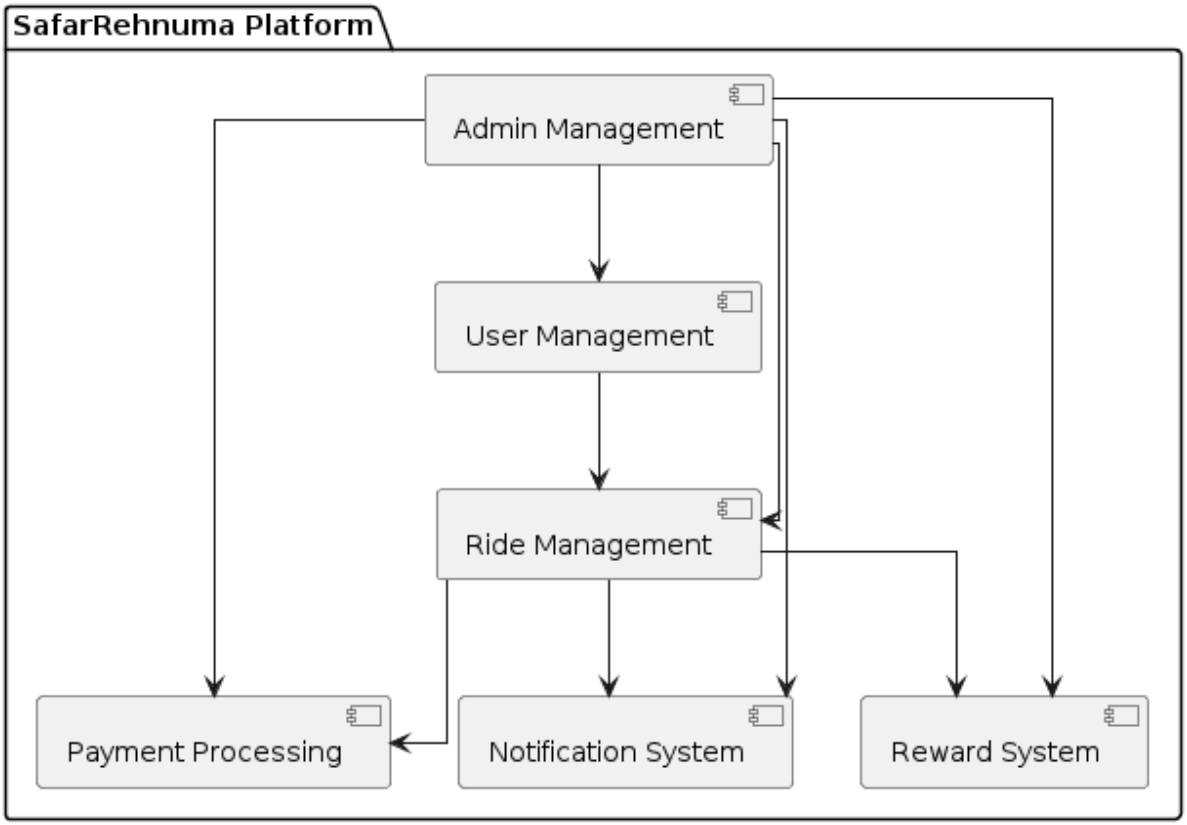


Figure 2 Box and Line Diagram for SafarRehnuma

This figure illustrates the architecture style diagram showing the multi-tiered pattern that is followed by the SafarRehnuma system.

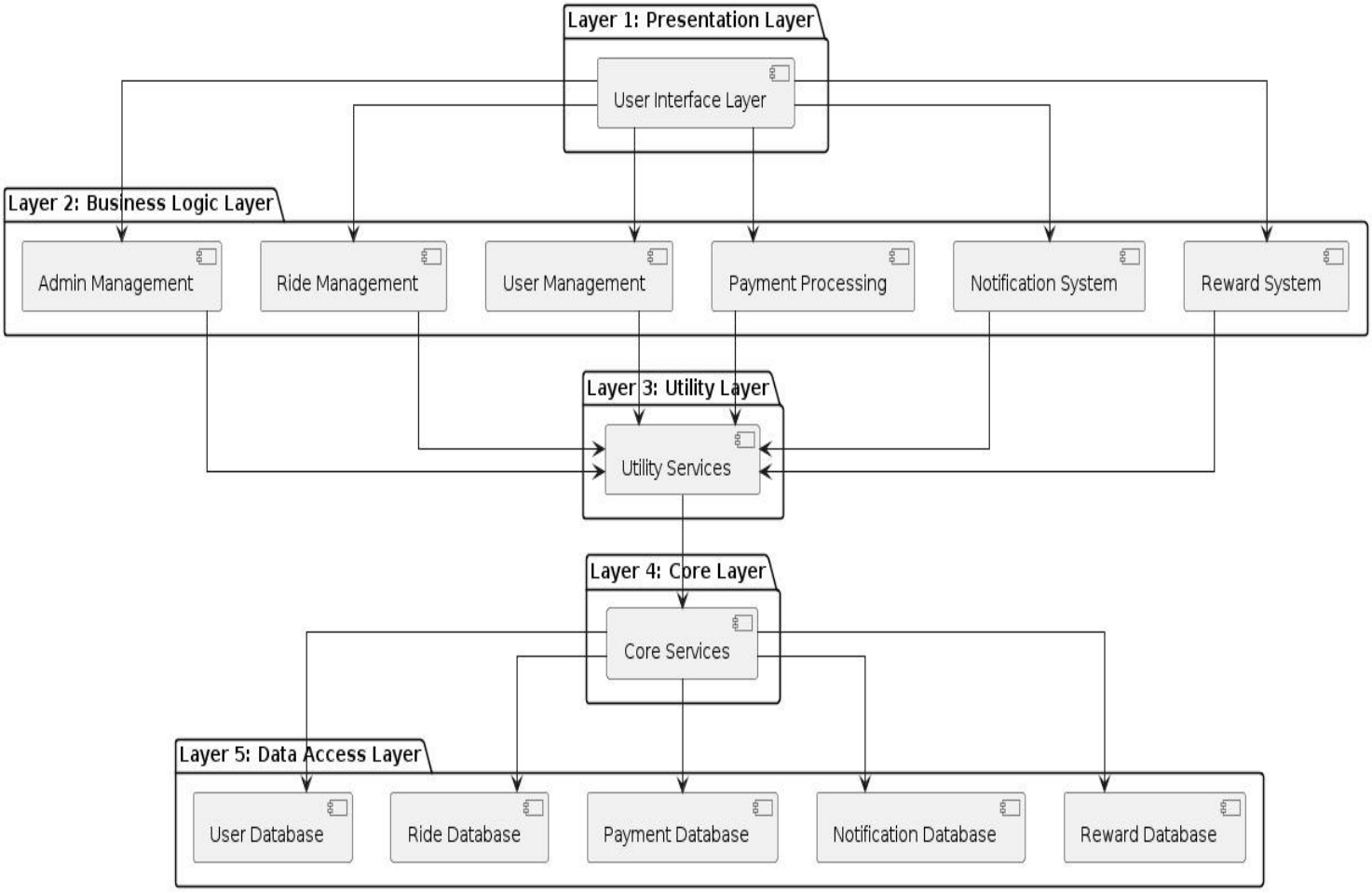


Figure 3 Architectural Style Diagram for SafarRehnuma

4. Design Models

This section details the design methodologies and models used to structure and implement the SafarRehnuma system.

4.1 Activity Diagrams

This subsection presents activity diagrams illustrating the workflow of key processes within SafarRehnuma.

4.1.1 Activity Diagram for Module 1: User Management

This figure illustrates the activity diagram detailing the flow of operations involved in the User Management module.

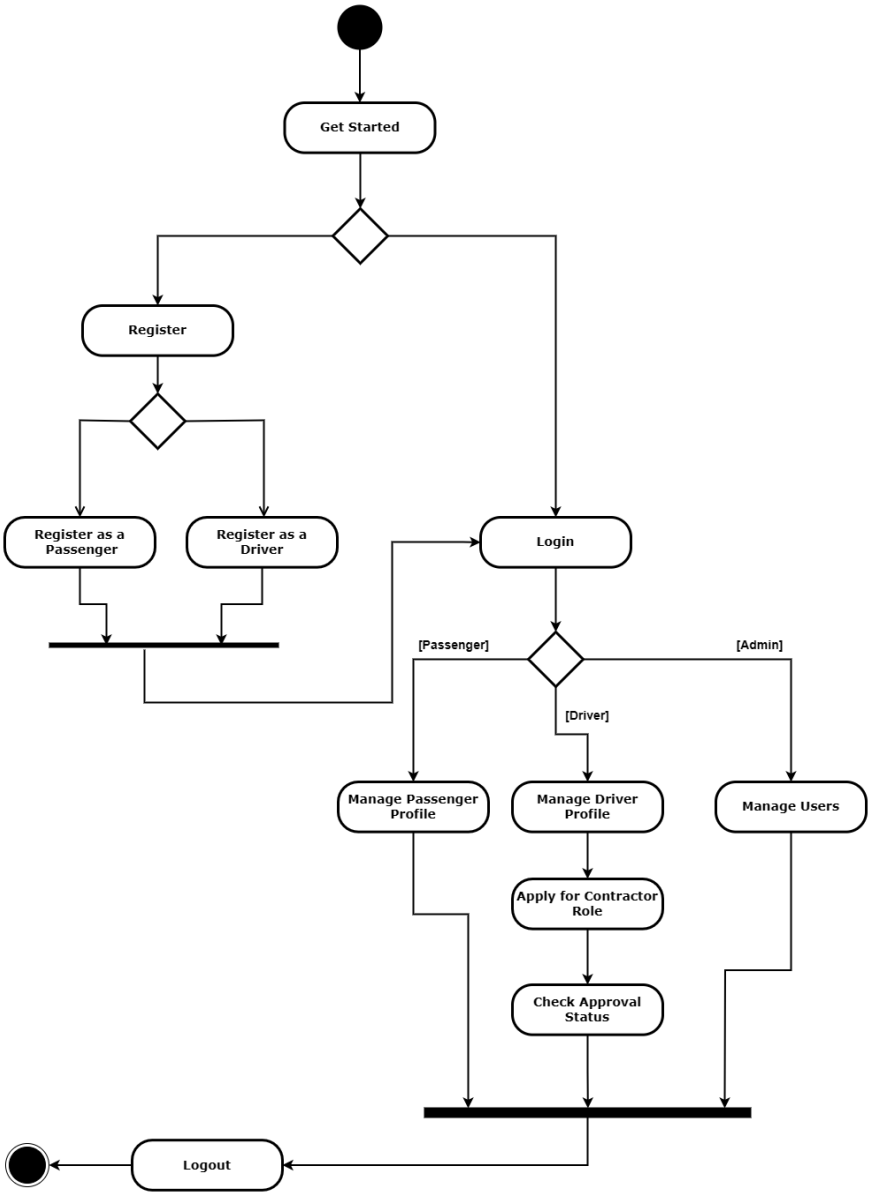


Figure 4 Activity Diagram for User Management

4.1.2 Activity Diagram for Module 2: Vehicle Registration

This figure illustrates the activity diagram detailing the flow of operations involved in Vehicle Registration module.

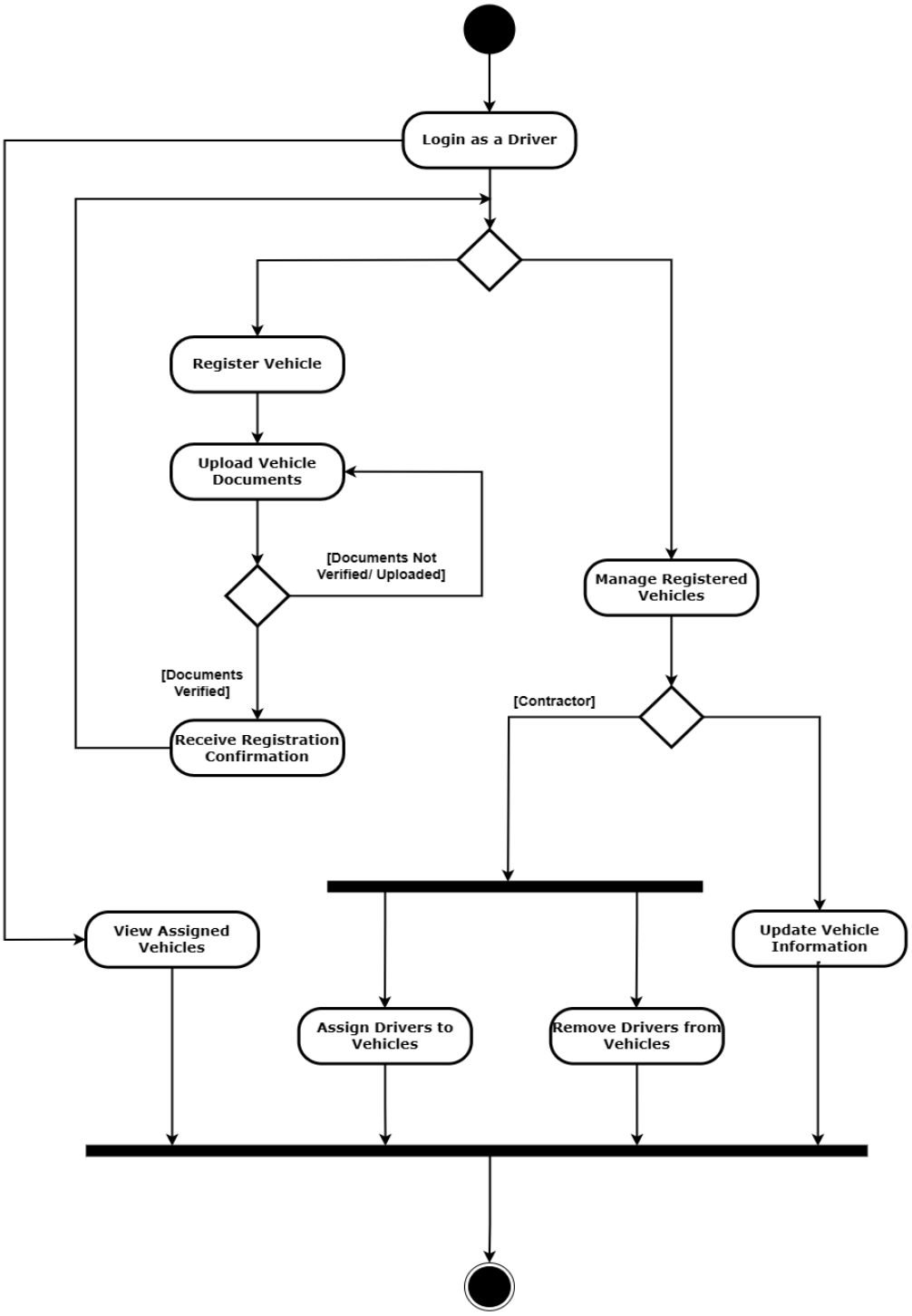


Figure 5 Activity Diagram for Vehicle Registration

4.1.3 Activity Diagram for Module 3: Ride Management

This figure illustrates the activity diagram detailing the flow of operations involved in Ride Management module.

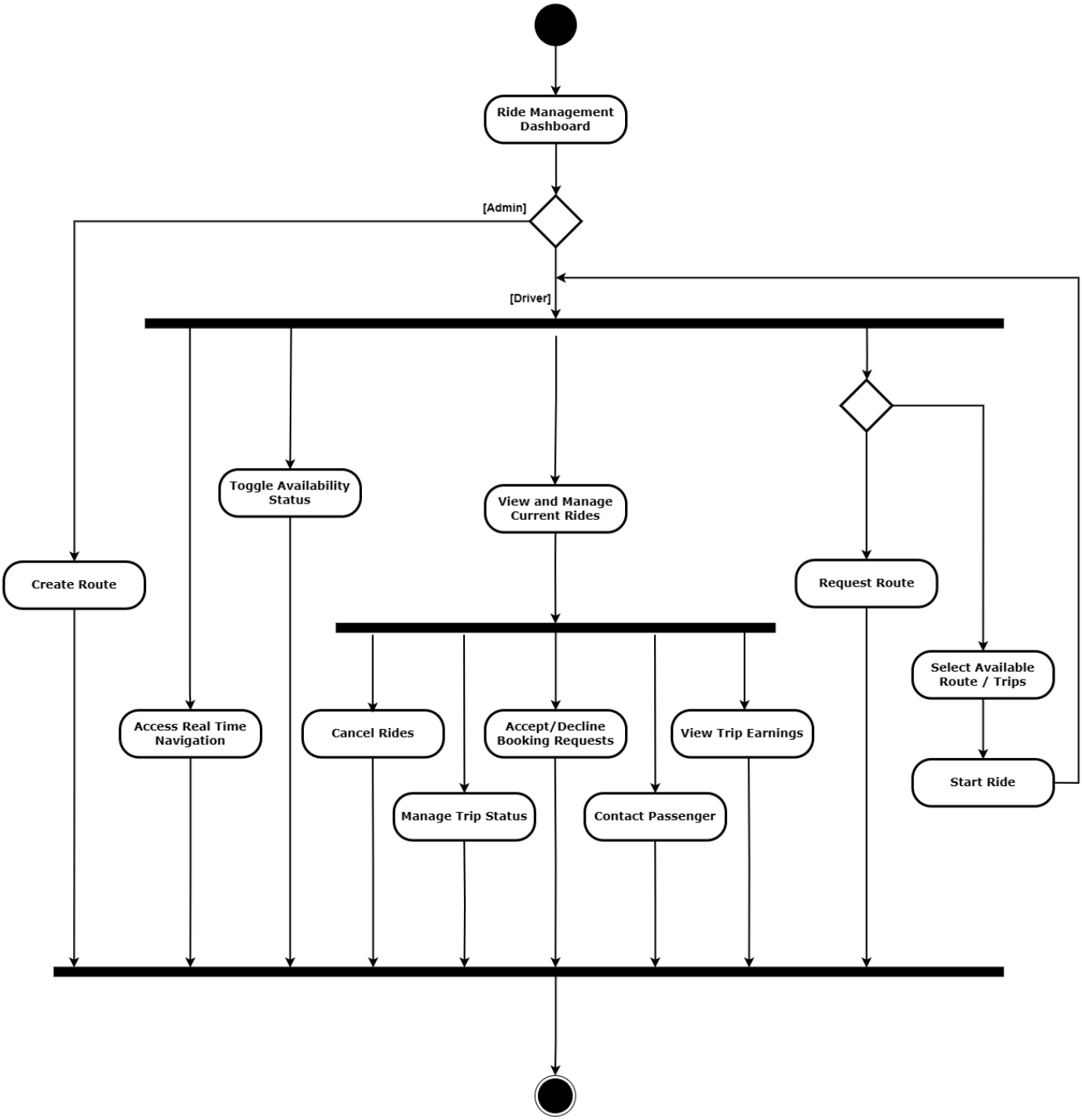


Figure 6 Activity Diagram for Ride Management

4.1.4 Activity Diagram for Module 4: Dynamic Pricing

This figure illustrates the activity diagram detailing the flow of operations involved in Dynamic Pricing module.

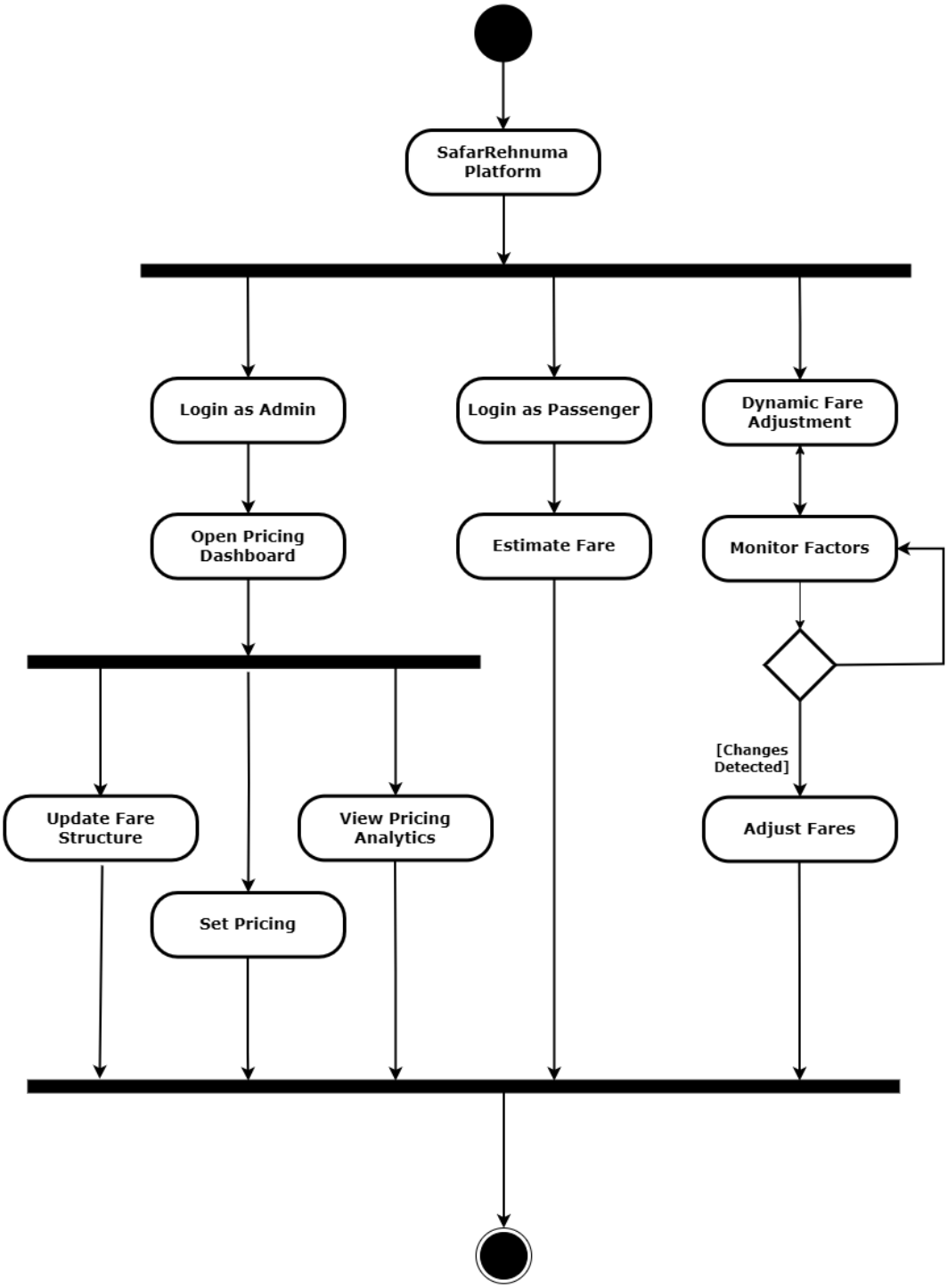


Figure 7 Activity Diagram for Dynamic Pricing

4.1.5 Activity Diagram for Module 5: Location Tracking

This figure illustrates the activity diagram detailing the flow of operations involved in Location Tracking module.

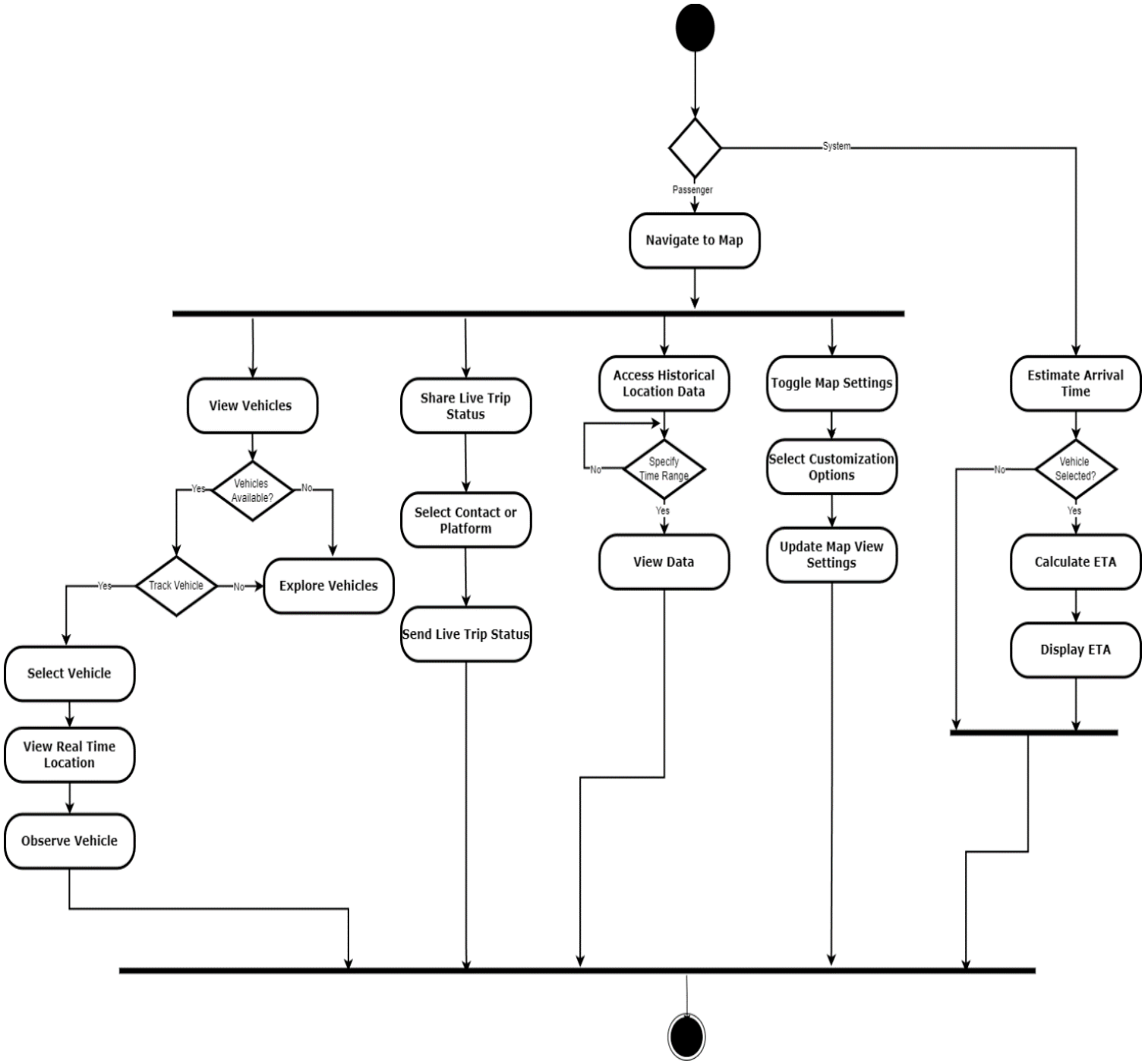


Figure 8 Activity Diagram for Location Tracking

4.1.6 Activity Diagram for Module 6: Route Optimization and Suggestions

This figure illustrates the activity diagram detailing the flow of operations involved in Route Optimization and Suggestions module.

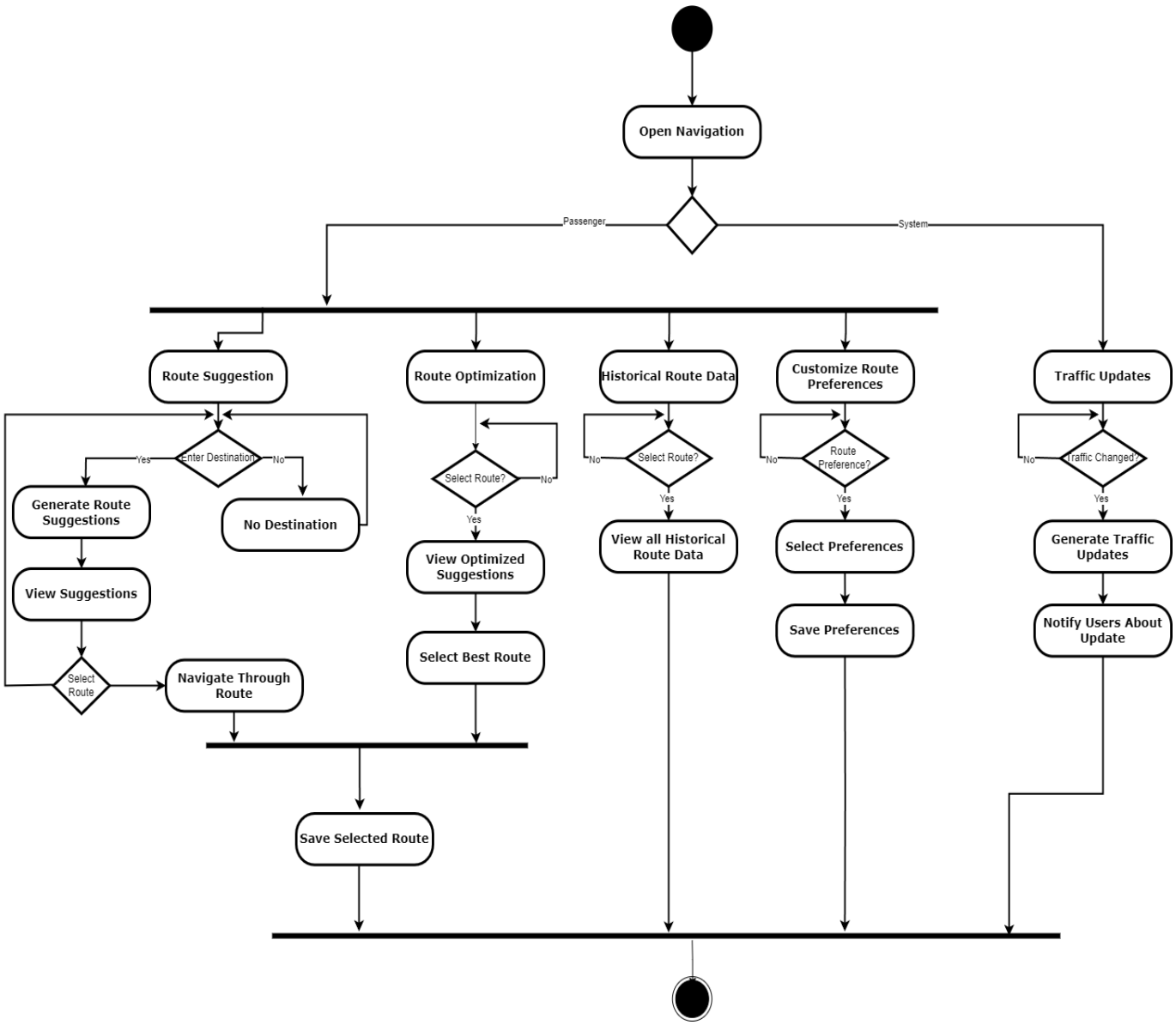


Figure 9 Activity Diagram for Route Optimization and Suggestions

4.1.7 Activity Diagram for Module 7: Booking Management

This figure illustrates the activity diagram detailing the flow of operations involved in Booking Management module.

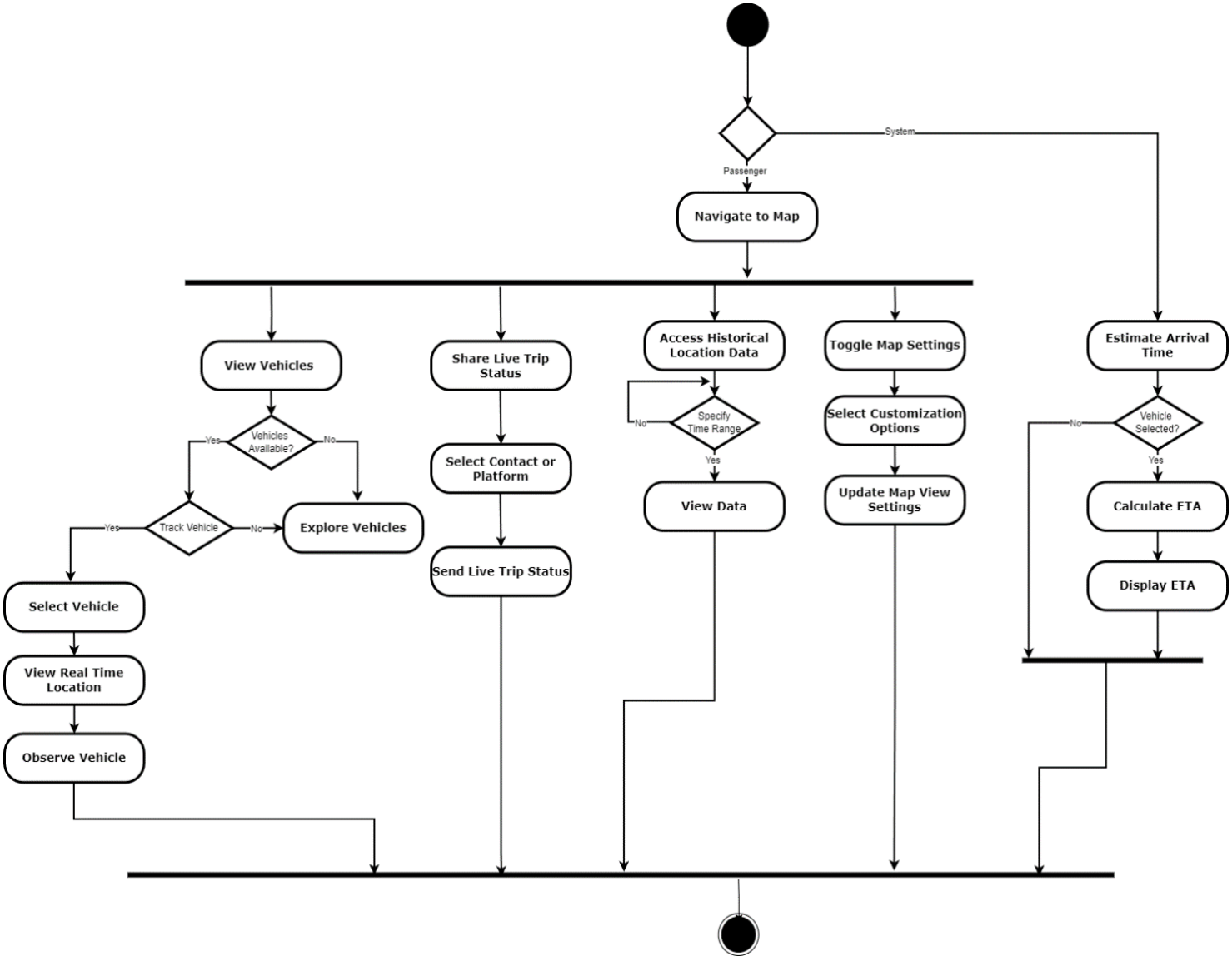


Figure 10 Activity Diagram for Booking Management

4.1.8 Activity Diagram for Module 8: Payments

This figure illustrates the activity diagram detailing the flow of operations involved in Payments module.

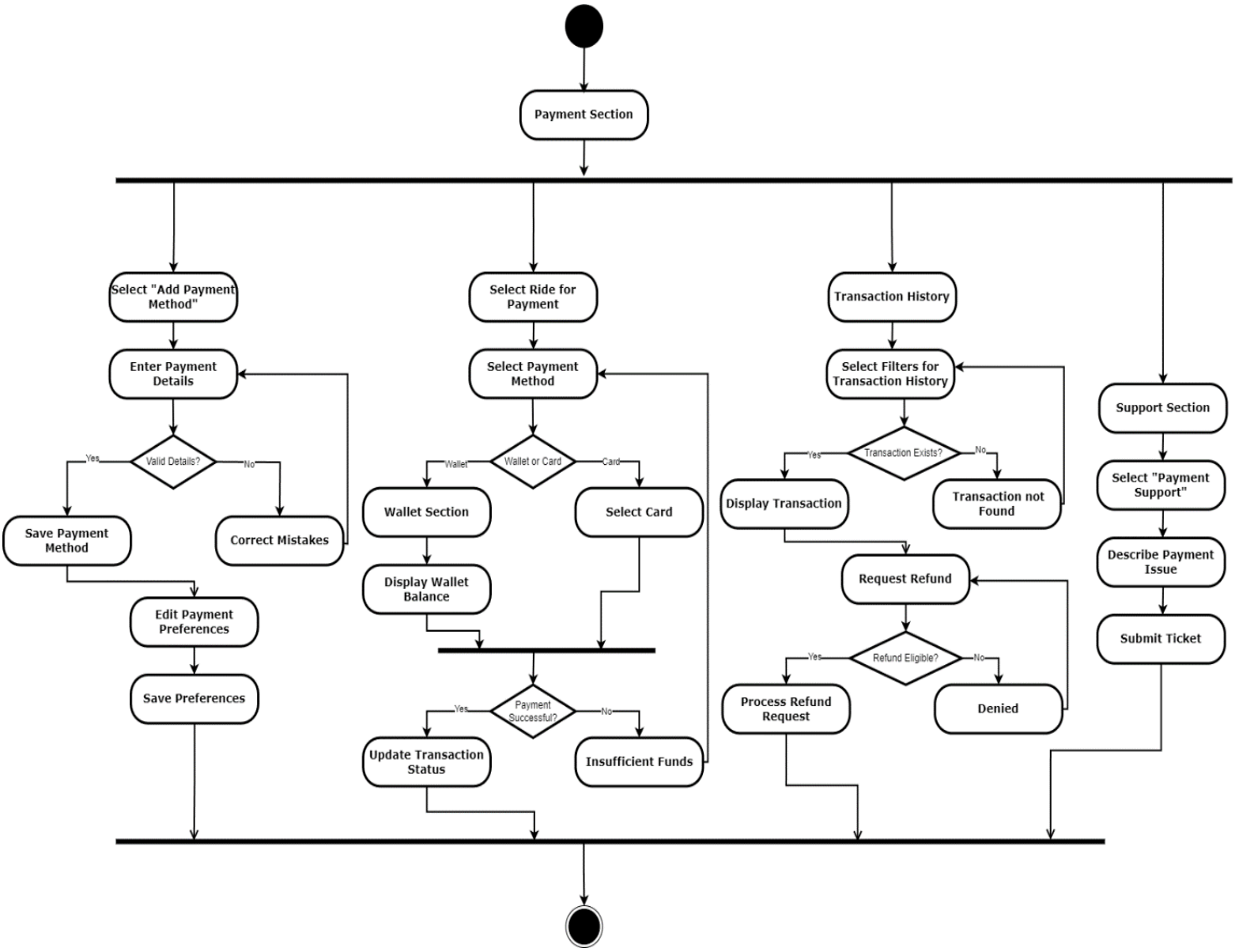


Figure 11 Activity Diagram for Payments

4.1.9 Activity Diagram for Module 9: Reviews and Ratings

This figure illustrates the activity diagram detailing the flow of operations involved in Reviews and Ratings module.

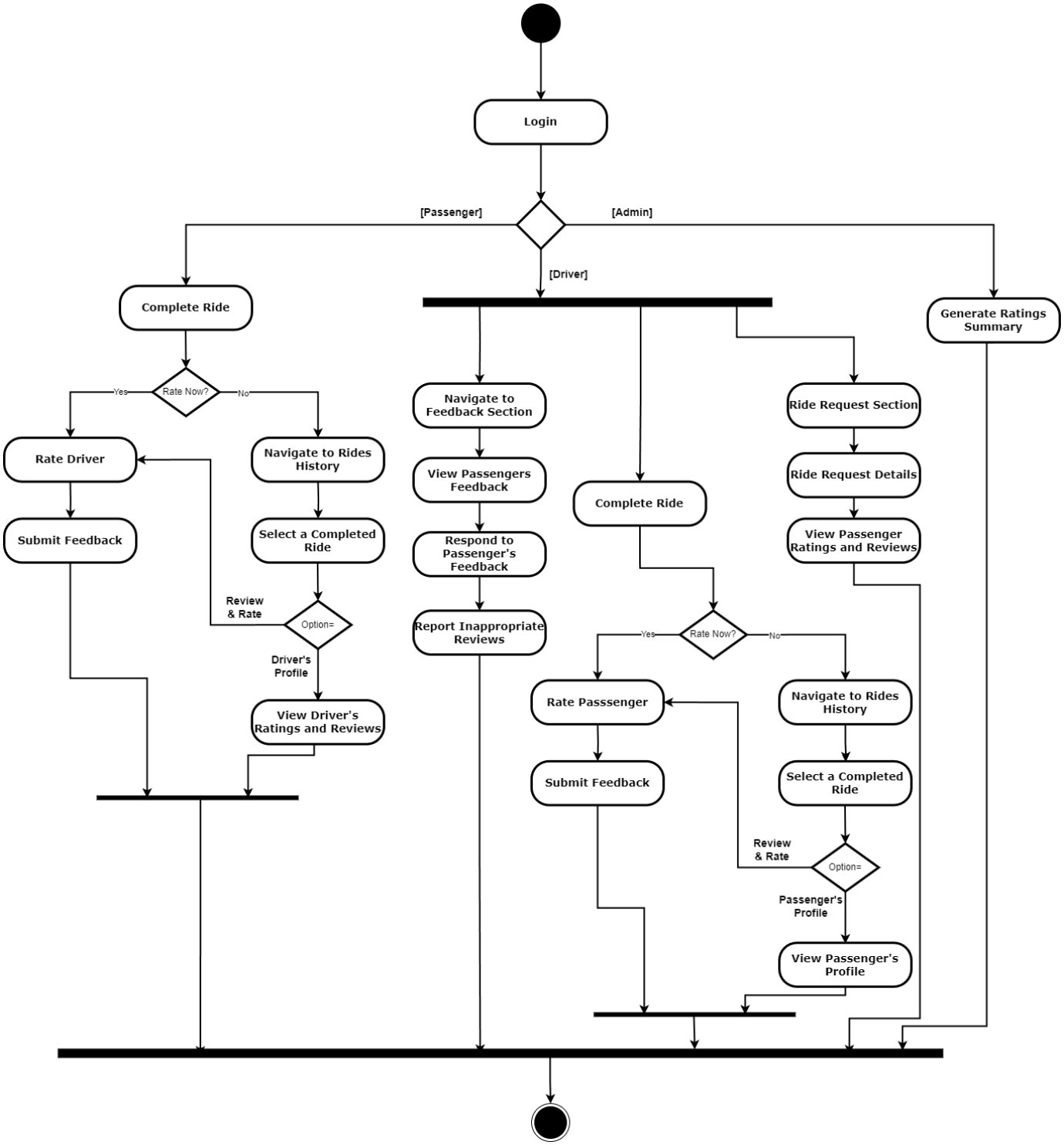


Figure 12 Activity Diagram for Reviews and Ratings

4.1.10 Activity Diagram for Module 10: Notifications

This figure illustrates the activity diagram detailing the flow of operations involved in Notifications module.

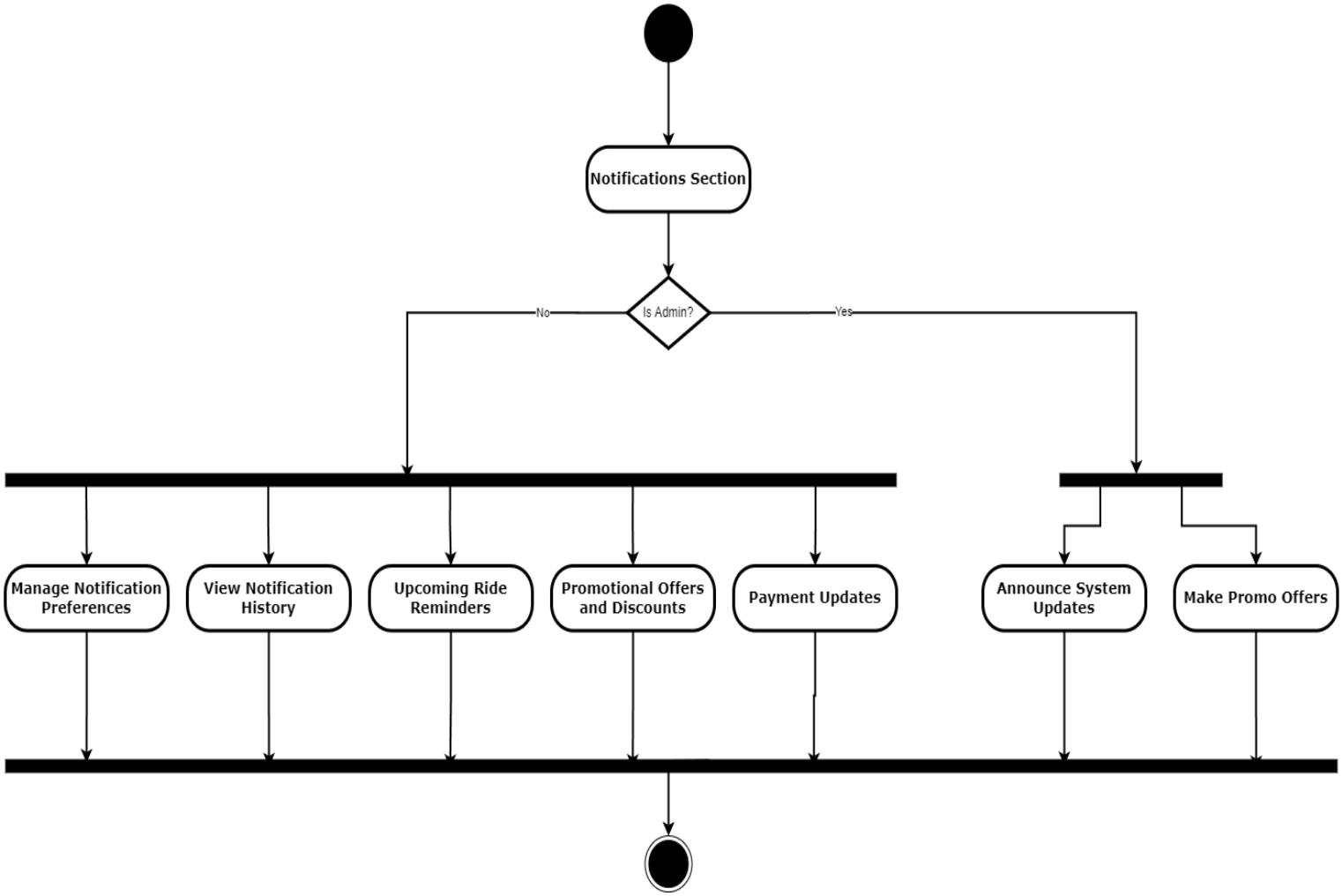


Figure 13 Activity Diagram for Notifications

4.1.11 Activity Diagram for Module 11: Rewards

This figure illustrates the activity diagram detailing the flow of operations involved in Rewards module.

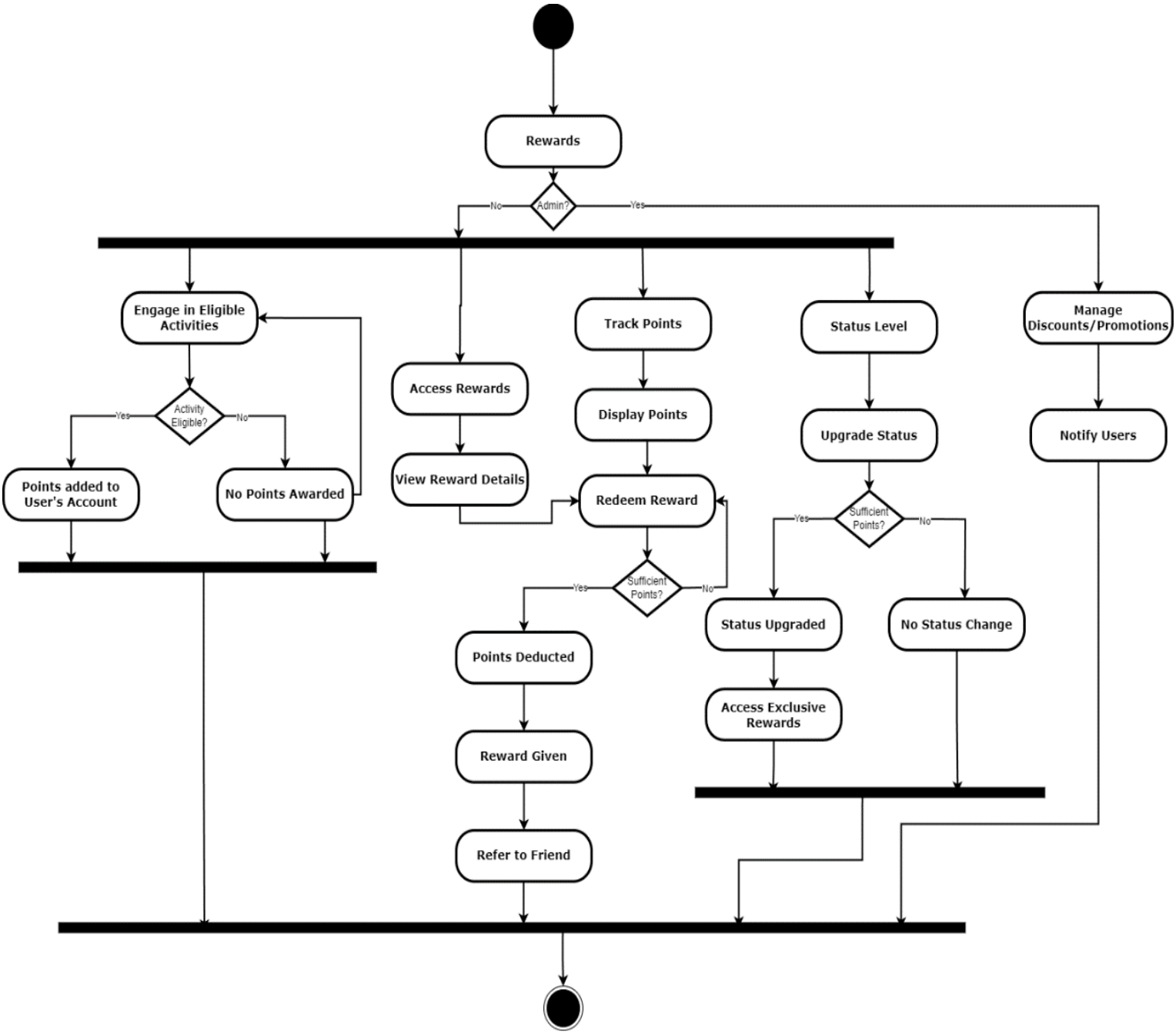


Figure 14 Activity Diagram for Rewards

4.2 Class Diagram

This figure shows the class diagram representing the structure of the SafarRehnuma system, illustrating the various classes, their attributes, methods, and relationships.

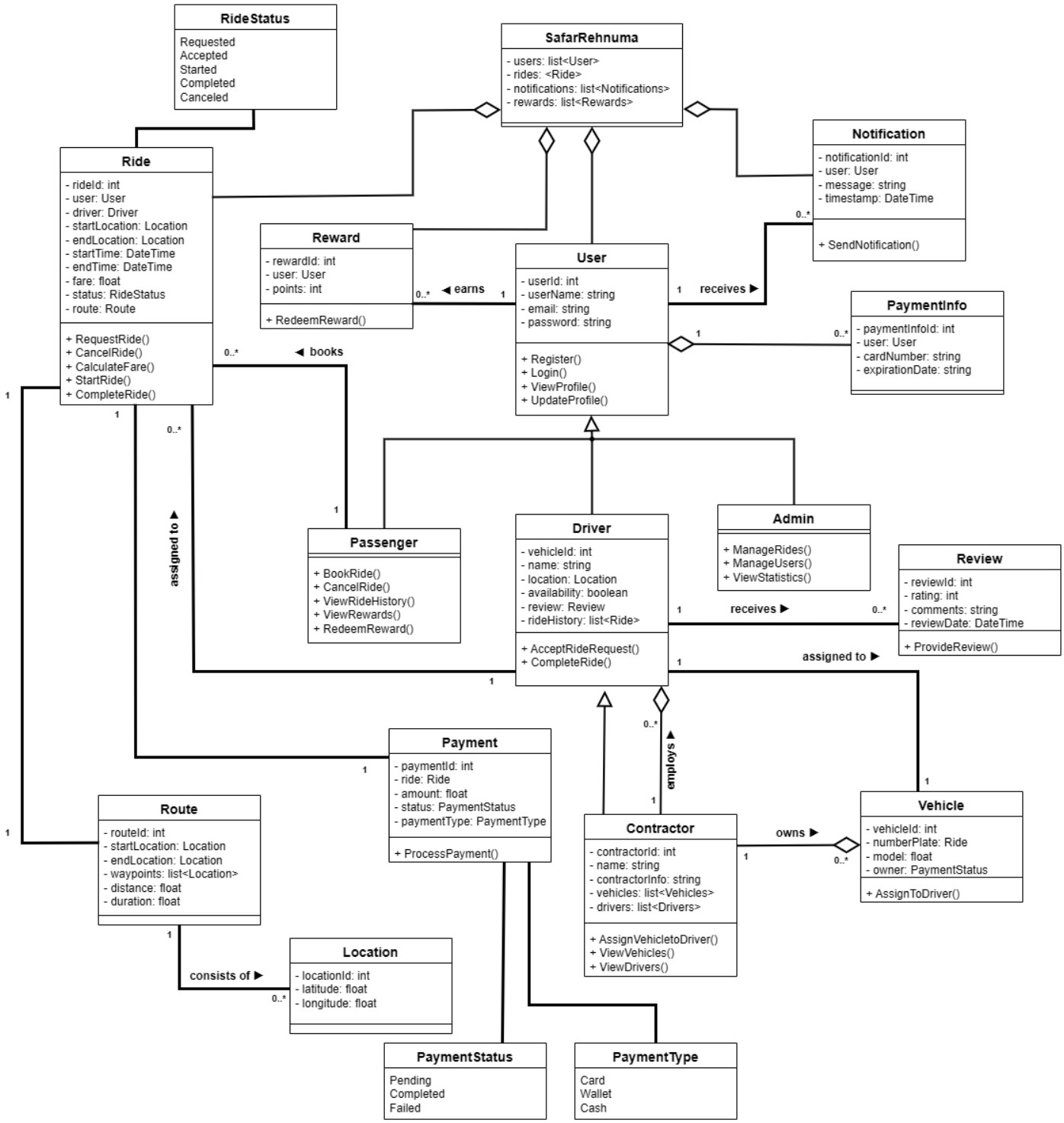


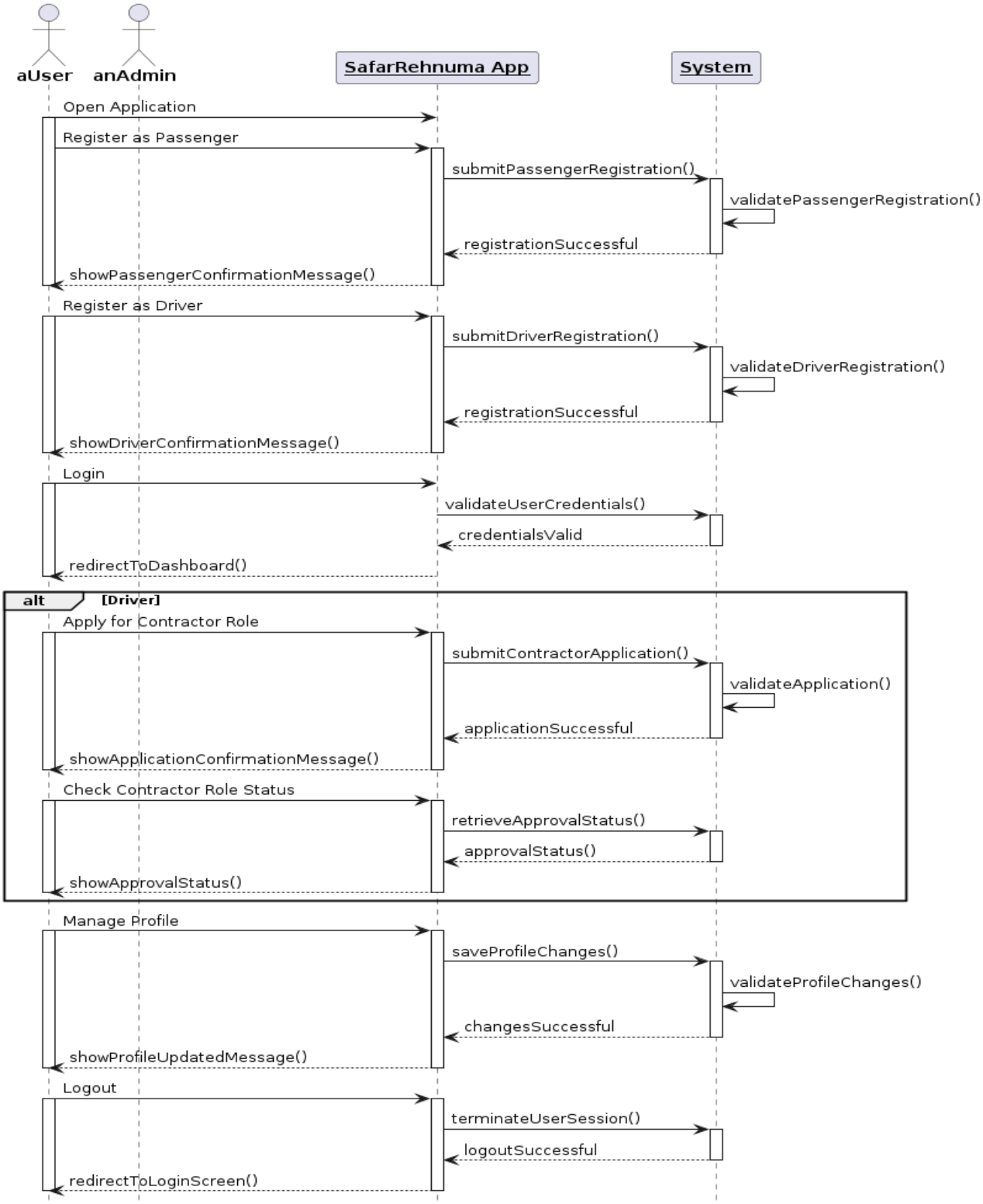
Figure 15 Class Diagram for SafarRehnuma

4.3 Sequence Diagrams

This subsection presents sequence diagrams depicting the sequence of interactions and messages exchanged between objects within SafarRehnuma's system during key processes.

4.3.1 Module 1: User Management

This figure shows the sequence diagram for User Management module.



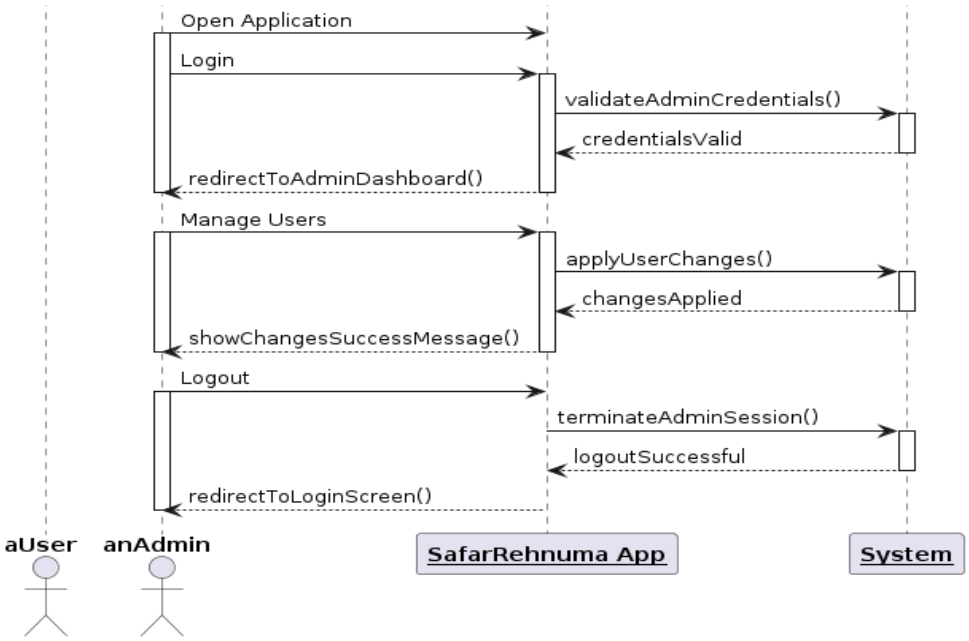


Figure 16 Sequence Diagram for User Management

4.3.2 Module 2: Vehicle Registration

This figure shows the sequence diagram for Vehicle Registration module.

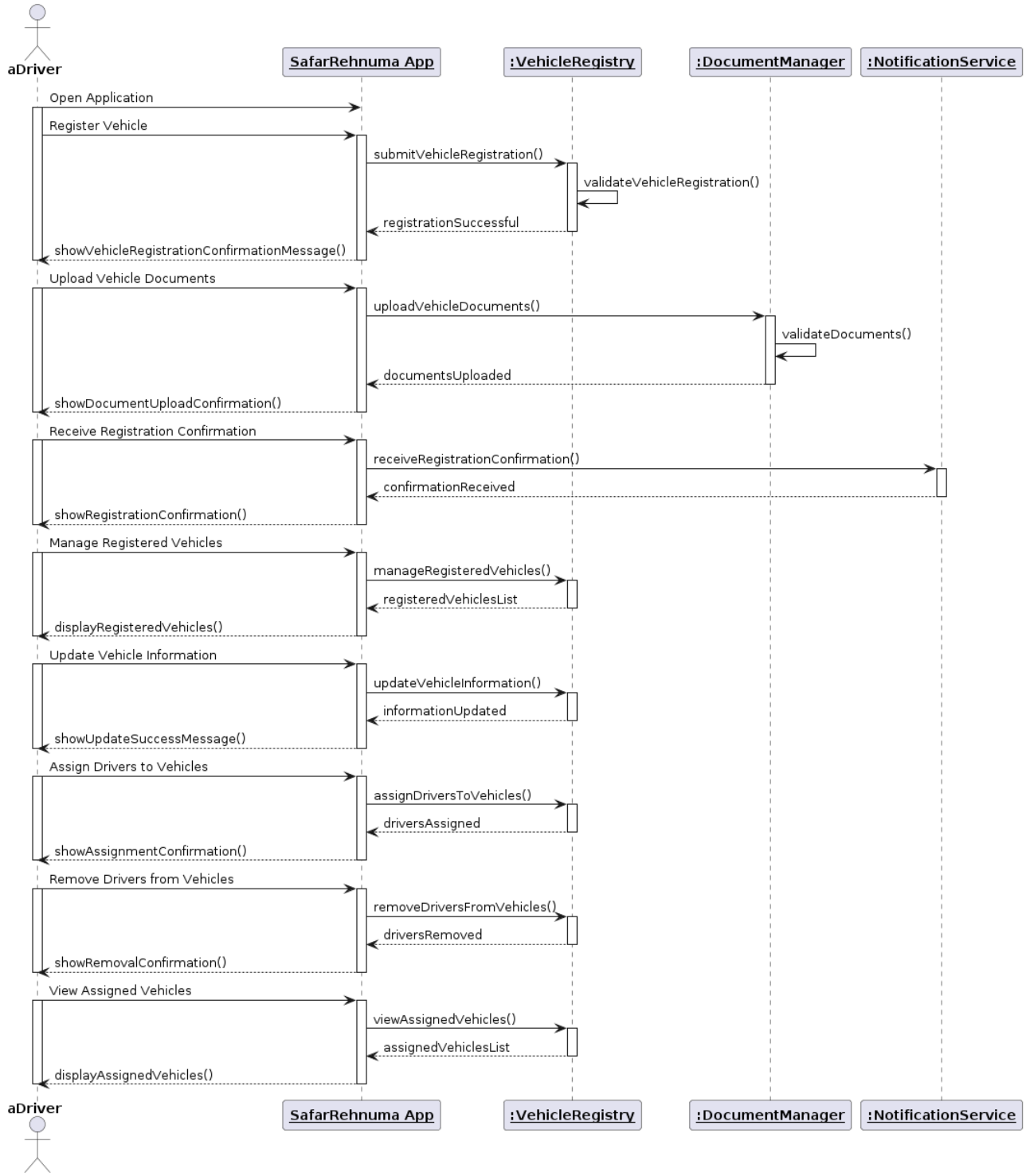


Figure 17 Sequence Diagram for Vehicle Registration

4.3.3 Module 3: Ride Management

This figure shows the sequence diagram for Ride Management module.

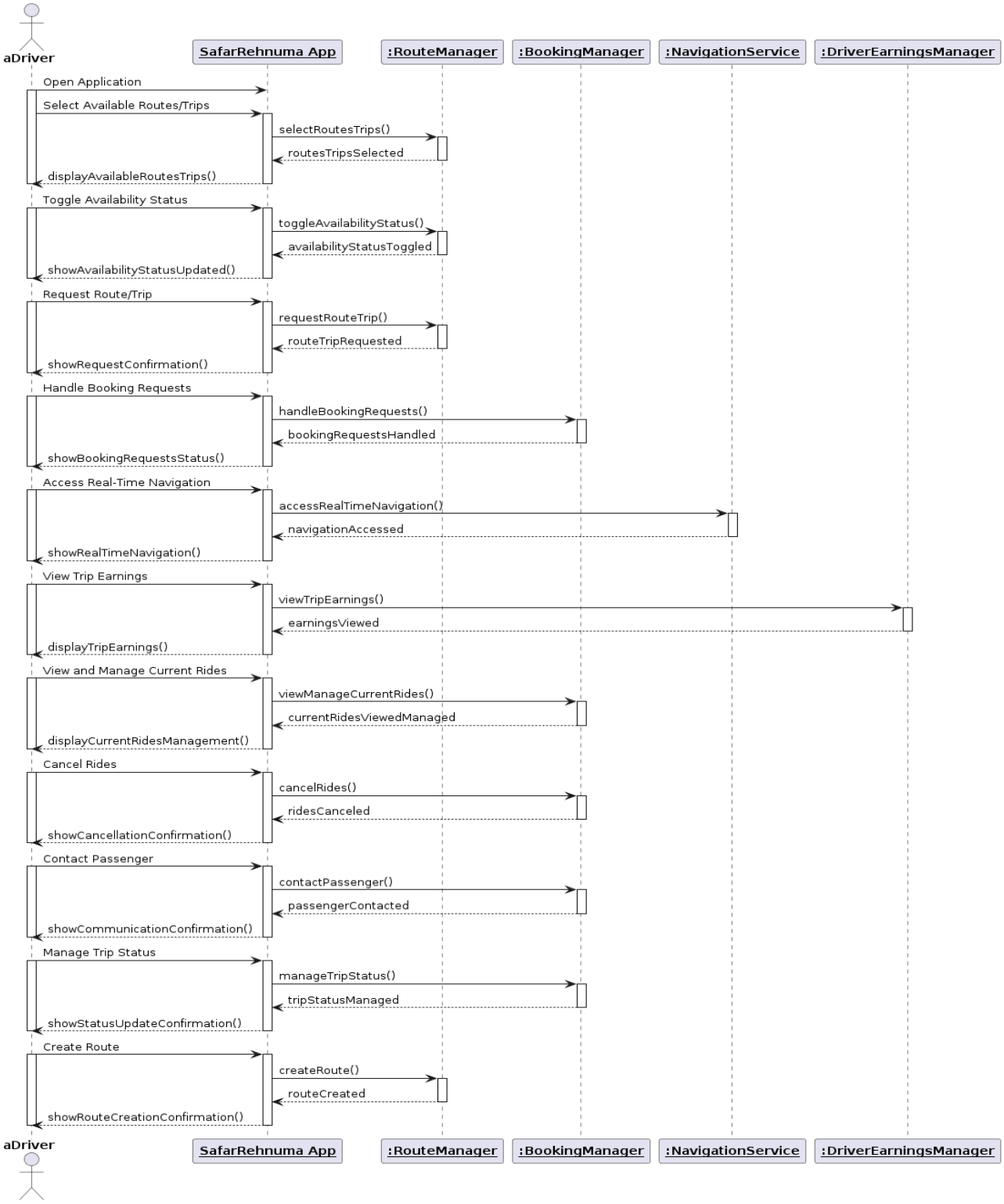


Figure 18 Sequence Diagram for Ride Management

4.3.4 Module 4: Dynamic Pricing

This figure shows the sequence diagram for Dynamic Pricing module.

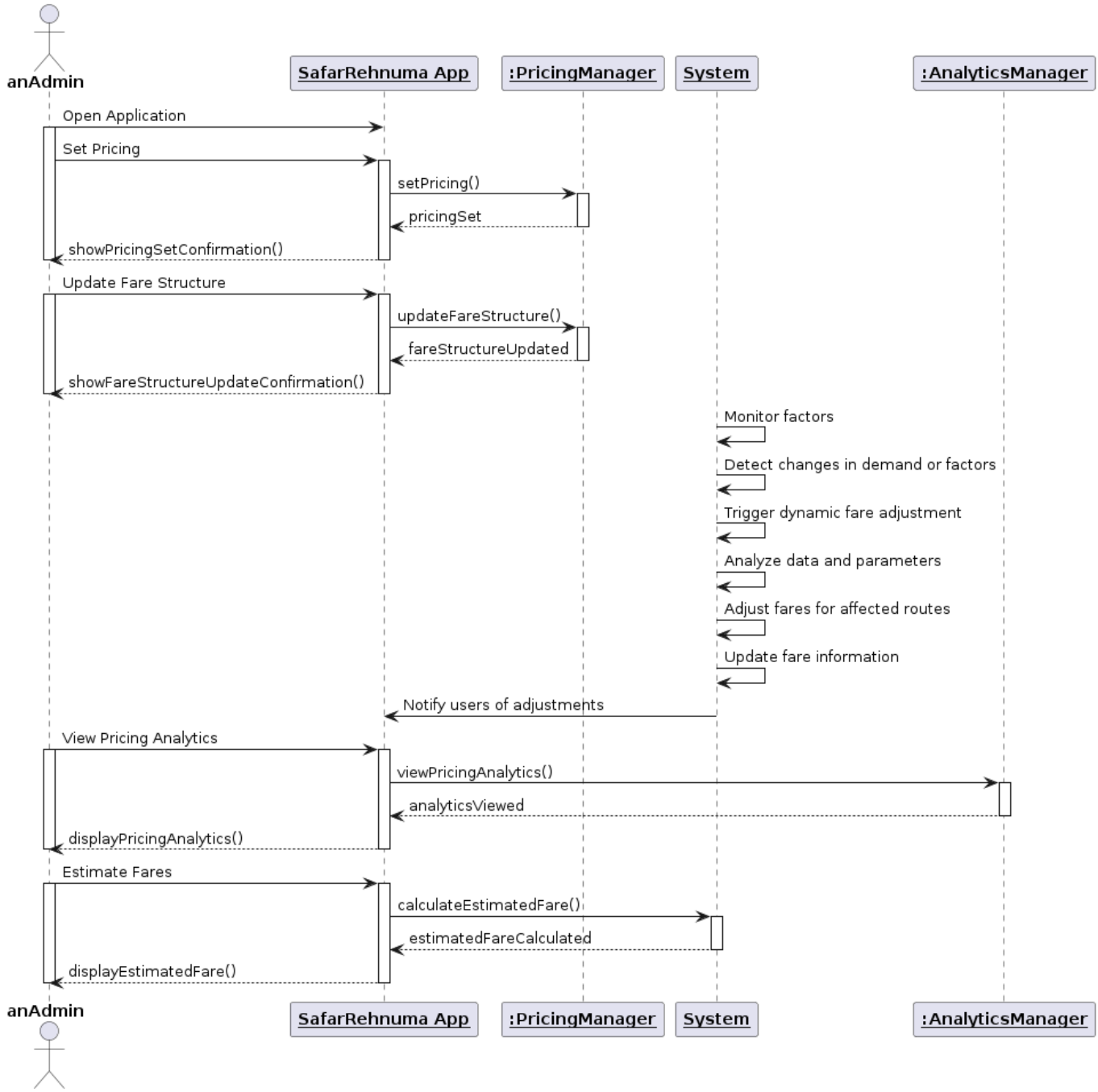
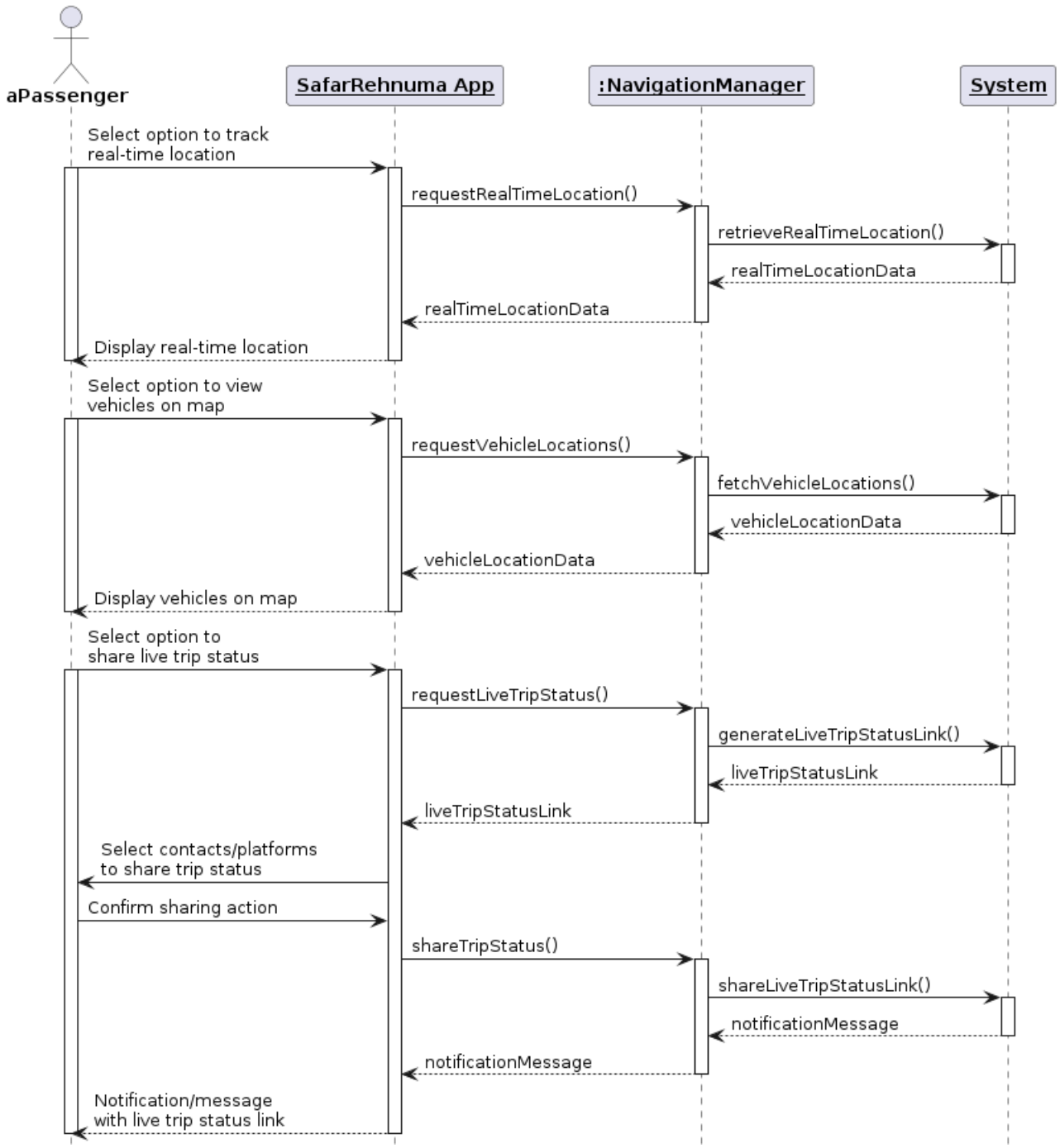


Figure 19 Sequence Diagram for Dynamic Pricing

4.3.5 Module 5: Location Tracking

This figure shows the sequence diagram for Location Tracking module.



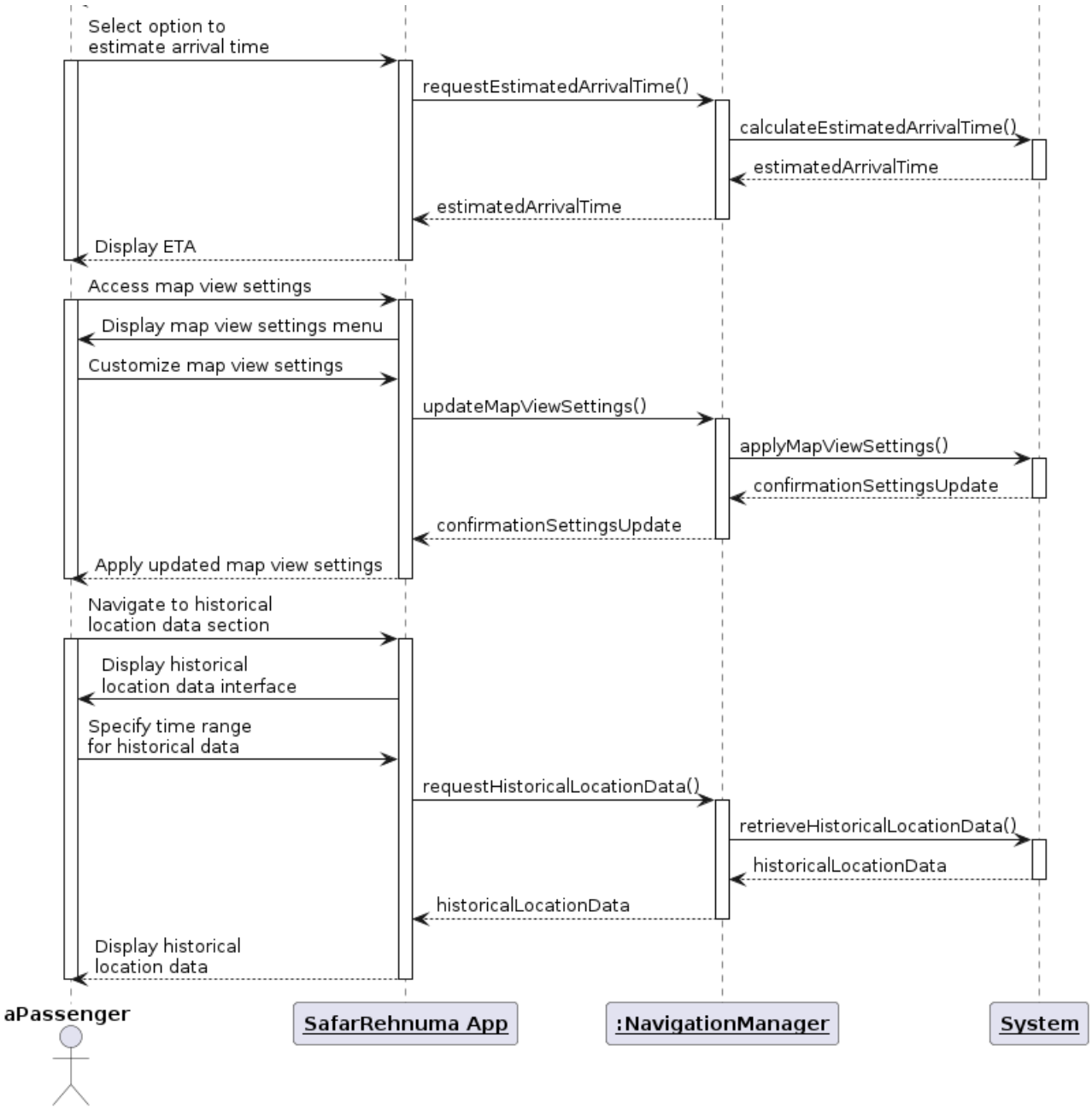


Figure 20 Sequence Diagram for Location Tracking

4.3.6 Module 6: Route Optimization and Suggestions

This figure shows the sequence diagram for Location Tracking module.

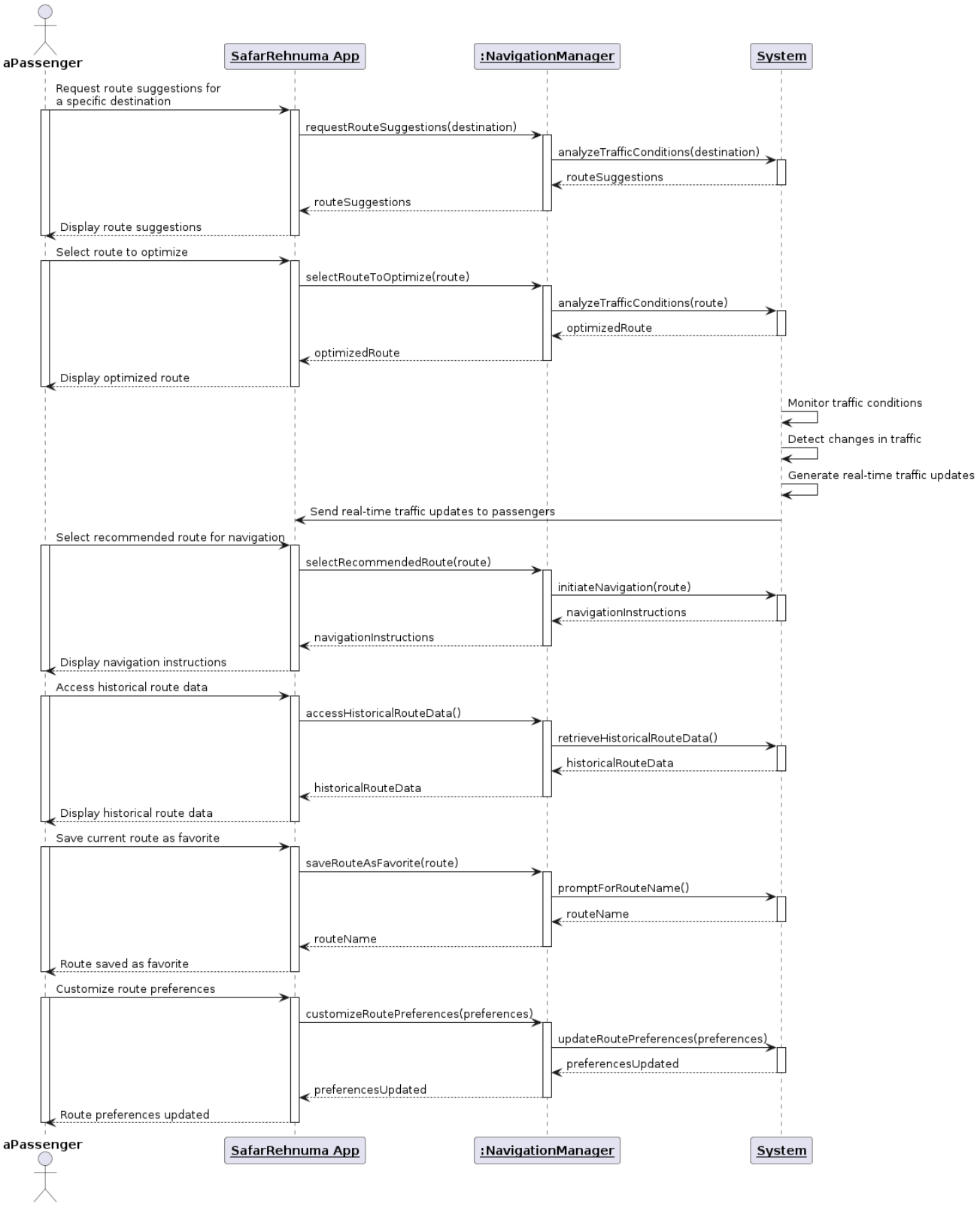
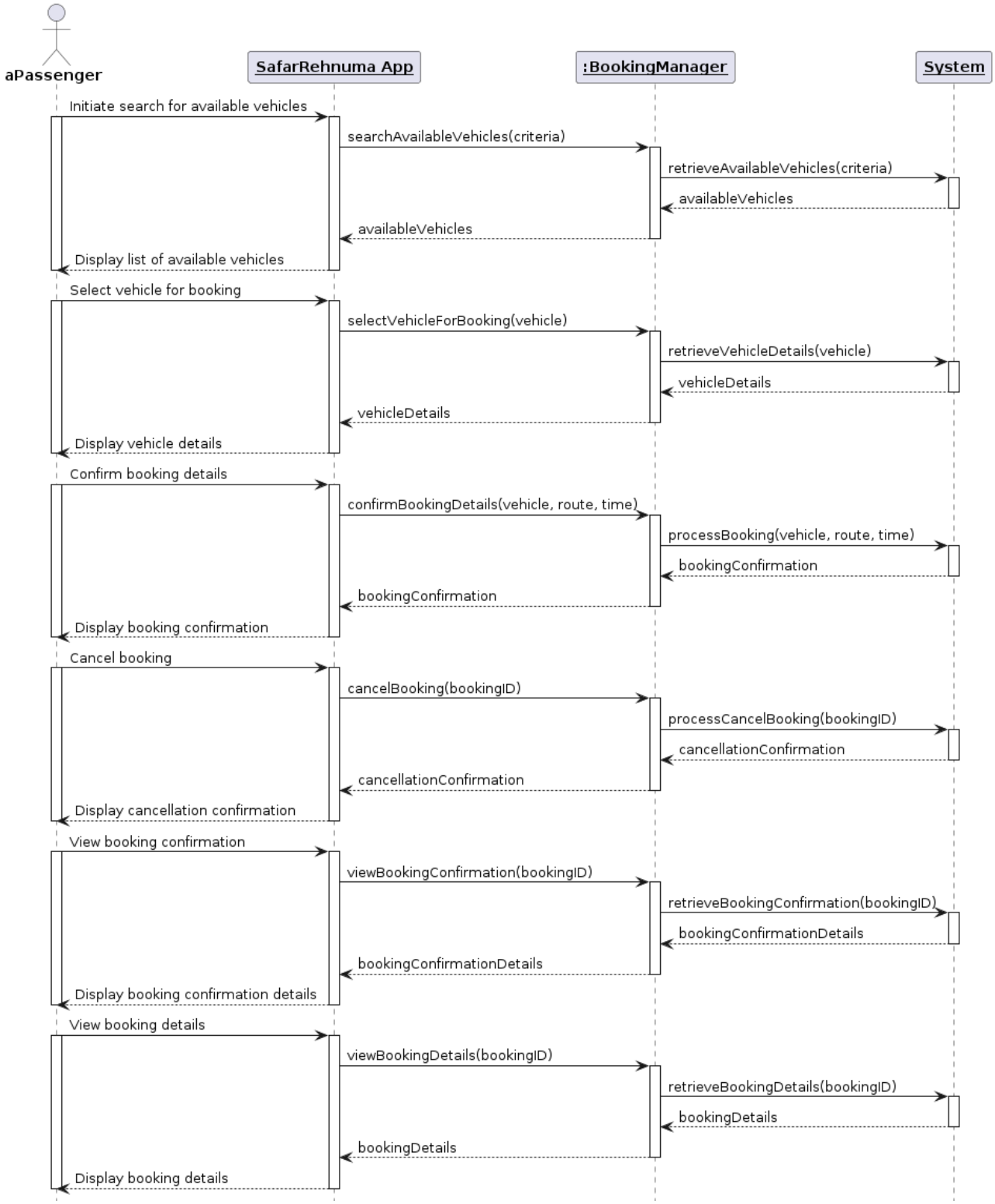


Figure 21 Sequence Diagram for Route Optimizations and Suggestions

4.3.7 Module 7: Booking Management

This figure shows the sequence diagram for Booking Management module.



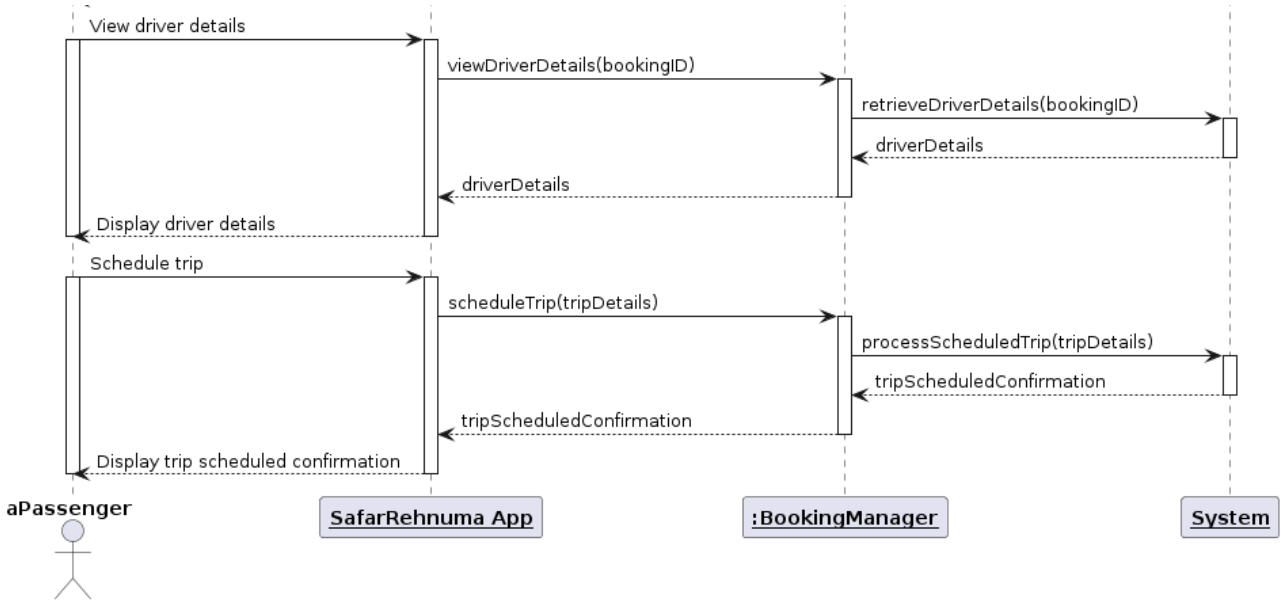


Figure 22 Sequence Diagram for Booking Management

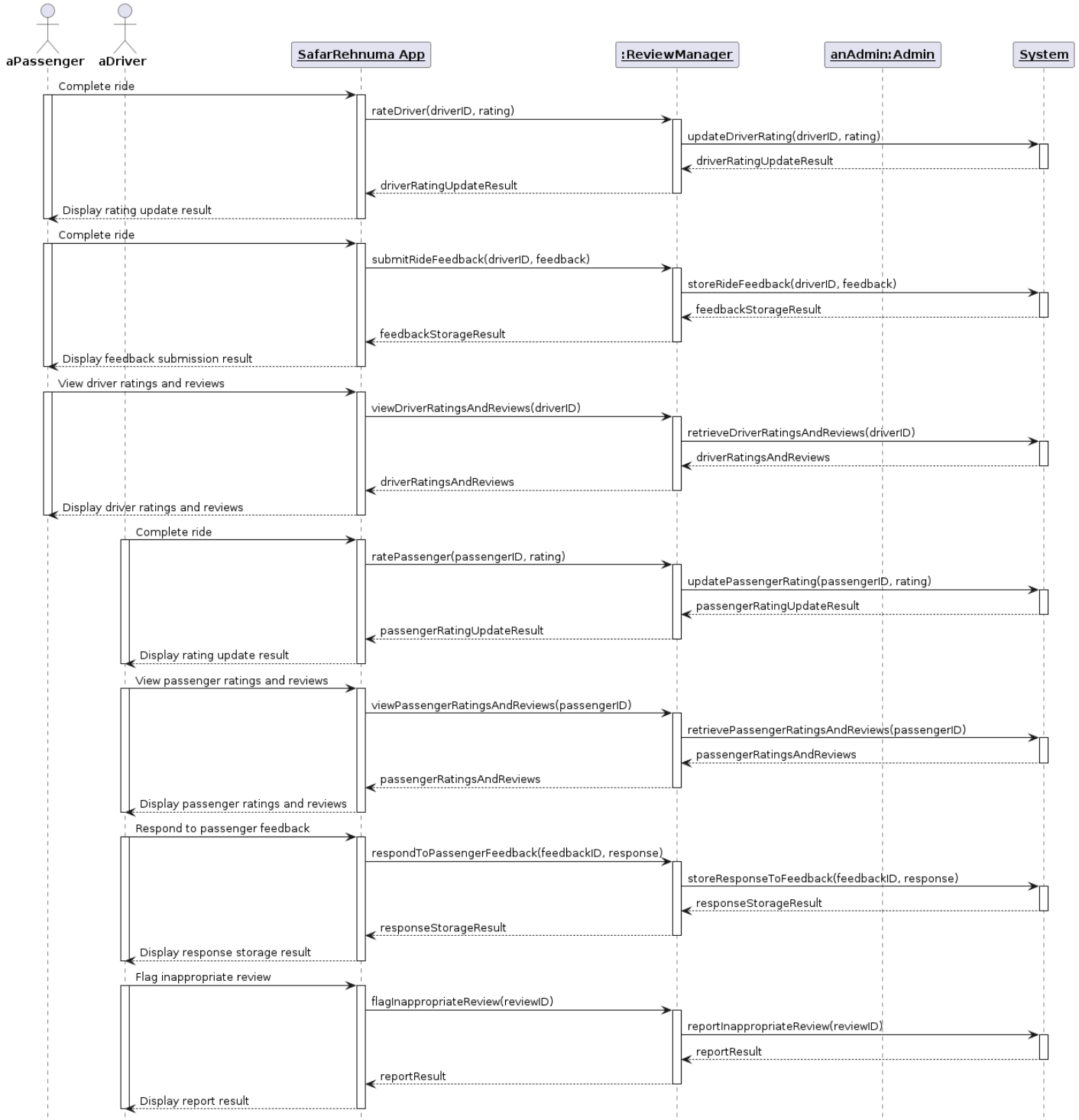
4.3.8 Module 8: Payments

This figure shows the sequence diagram for Payments module.

Figure 23 Sequence Diagram for Payments

4.3.9 Module 9: Reviews and Ratings

This figure shows the sequence diagram for Reviews and Ratings module.



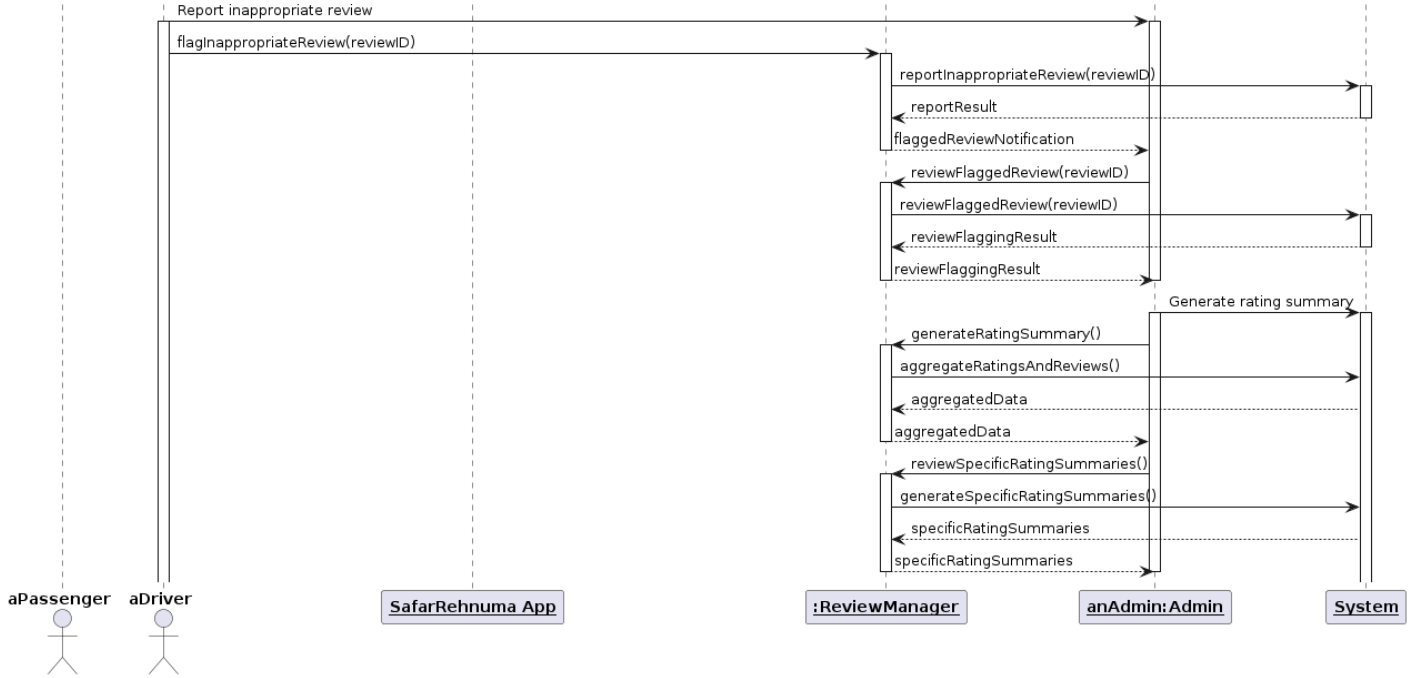


Figure 24 Sequence Diagram for Ratings and Reviews

4.3.10 Module 10: Notifications

This figure shows the sequence diagram for Notifications module.

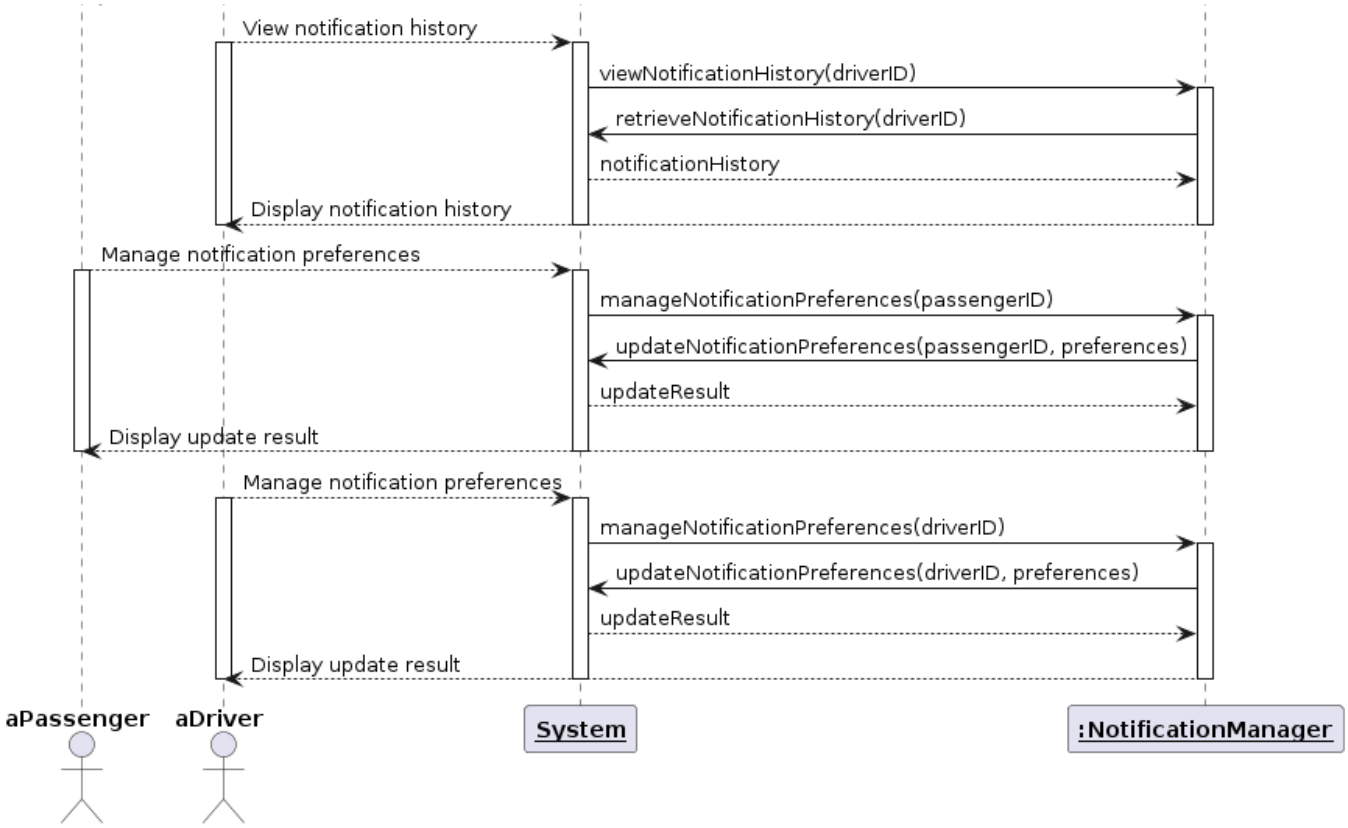
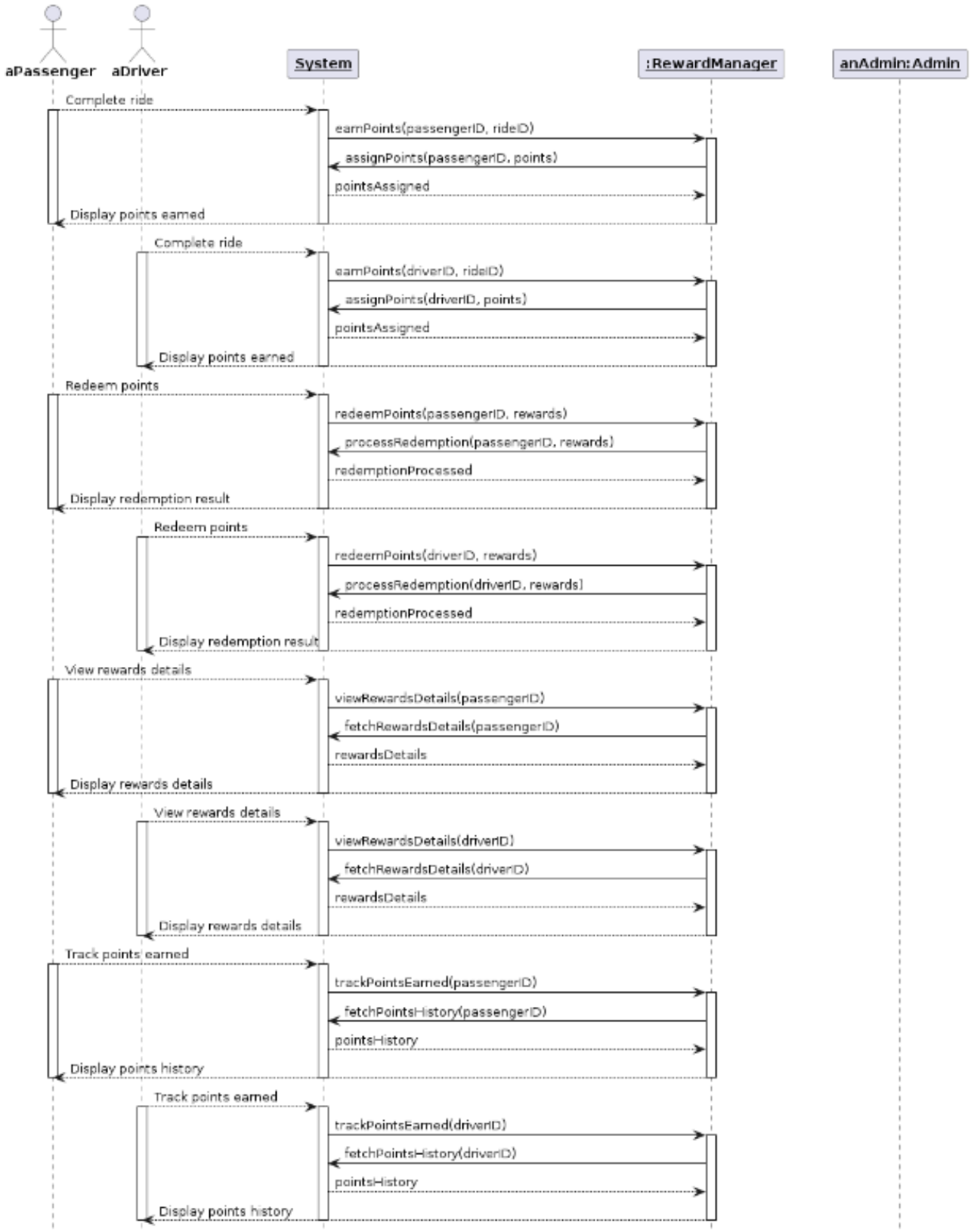


Figure 25 Sequence Diagram for Notifications

4.3.11 Module 11: Rewards

This subsection further shows the sequence diagram for the module “Rewards”.



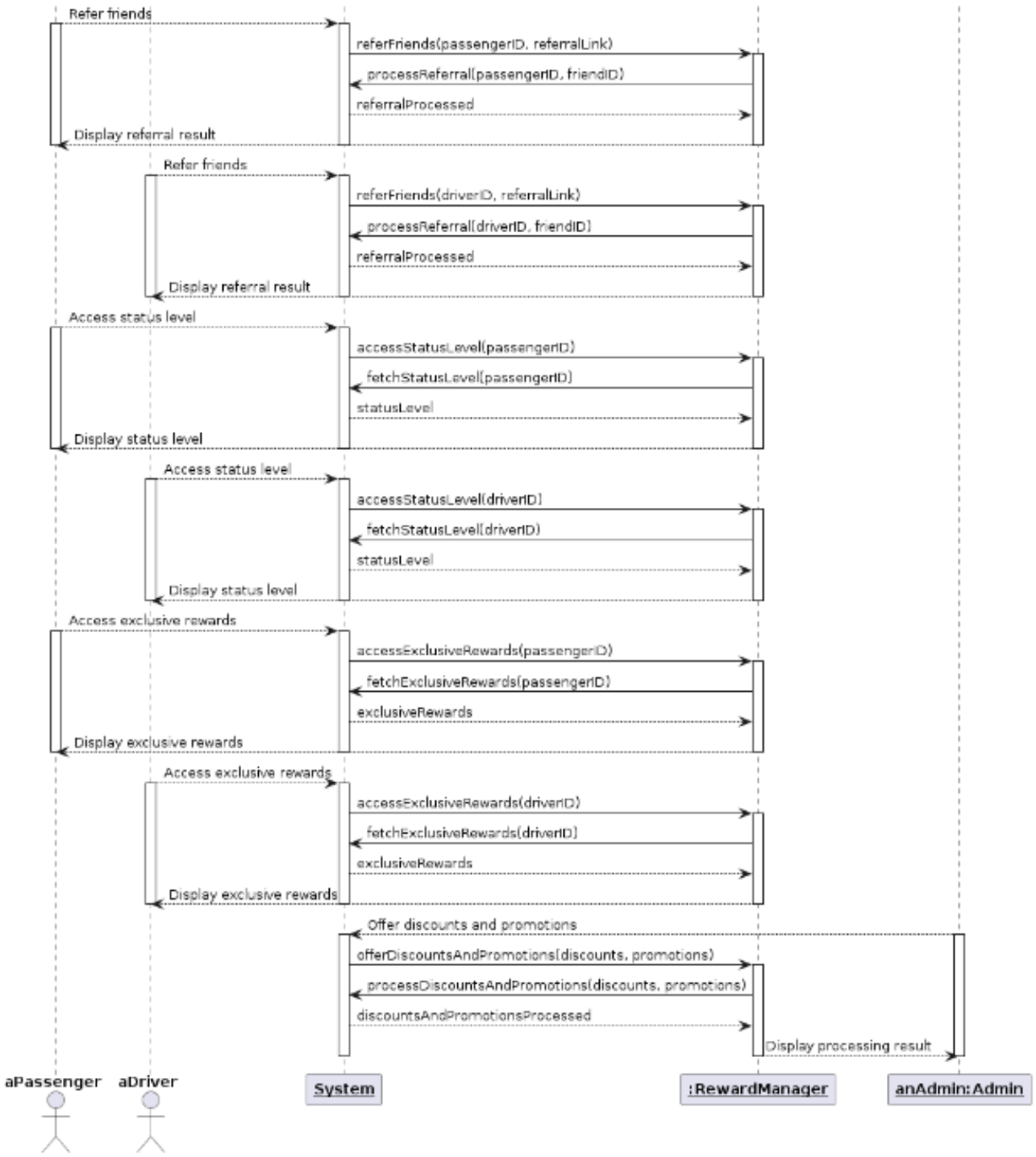


Figure 26 Sequence Diagram for Rewards

4.4 State Transition Diagrams

The State Transition diagrams for our system Safar Rehnuma are described here.

4.4.1 State Transition Diagram for Dynamic Pricing

This figure shows the state transition diagram for Dynamic Pricing.

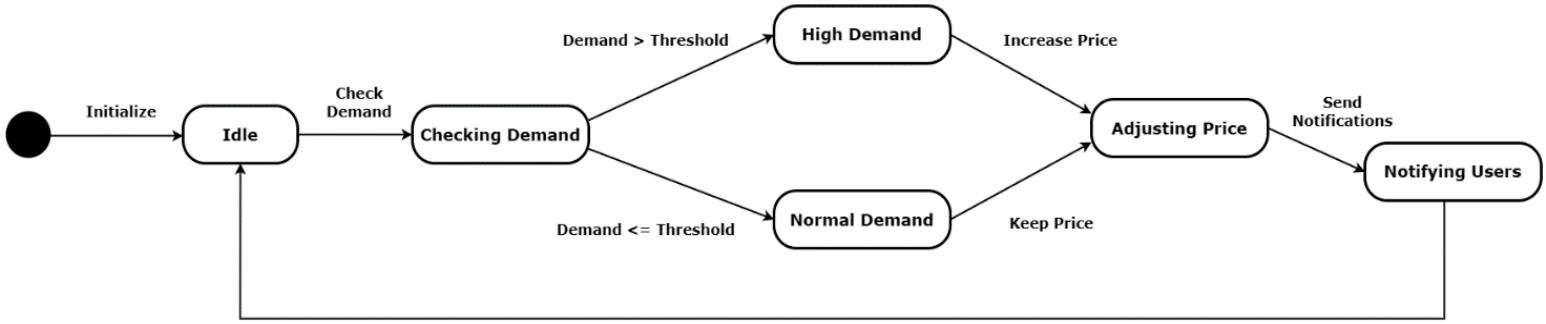


Figure 27 State Transition Diagram for Dynamic Pricing

4.4.2 State Transition Diagram for Route Suggestion

This figure shows the state transition diagram for Route Suggestion.

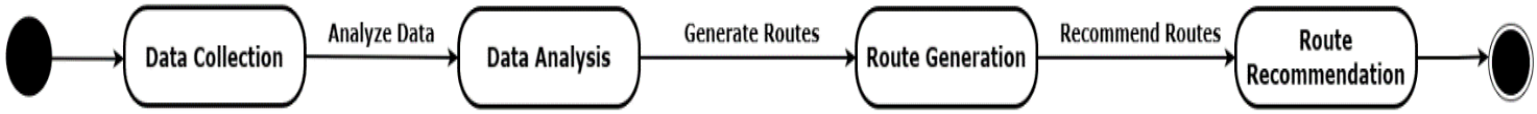


Figure 28 State Transition Diagram for Route Suggestion

4.4.3 State Transition Diagram for Ride Booking

This figure shows the state transition diagram for Ride Booking.

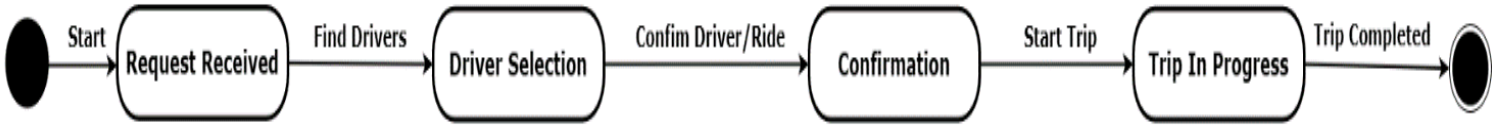


Figure 29 State Transition Diagram for Ride Booking

5. Data Design

This section outlines the structure and organization of the data used in SafarRehnuma, detailing the schema, data models, and relationships essential for efficient data management and retrieval.

5.1 Data Dictionary

This table shows the data dictionary used in the SafarRehnuma system.

Type	Username	String	Name of User
Users	Email	String	Email of User
	Password	String	Password for Authentication
	Photo	String	User Profile Photo
	Role	String	Role of User
	Position	String	User Current Position
	Reset Password	Token	String
	Token Expiry Date	Expiry Date	Expiry Date of Reset Password Token
	Position	String	User Current Position
Driver	Driver ID	String	Unique identifier for each driver
	User	String	Name of the user who is a driver
	License Number	String	Driver's license number
	Vehicle	String	Vehicle assigned to the driver
	Rating	Number	Driver's rating
	Availability	Boolean	Current availability status of the driver
Passenger	Passenger ID	String	Unique identifier for each passenger
	User	String	Name of the user who is a passenger
	Trip History	String	History of trips taken by the passenger
	Payment Info	String	Payment information of the passenger
	Rewards	Number	Reward points earned by the passenger
Rides	Ride ID	String	Unique identifier for each trip
	Passenger	String	Passenger's Name
	Driver	String	Driver's Name
	Origin	String	Origin location of the trip
	Destination	String	Destination location of the trip
	Fare	Number	Fare for the trip
	Start Time	DateTime	Start time of the ride
	End Time	DateTime	End time of the ride
	Status	String	Status of the ride

	Route	String	Route taken during the ride
Bookings	Booking ID	String	Unique identifier for each booking
	User	String	Name of the user making the booking
	Trip	String	ID of the trip associated with the booking
	Date	Date	Date of the booking
	Status	String	Status of the booking
	Payment Method	String	Payment method used for the booking
Payments	Payment ID	String	Unique identifier for each payment
	User	String	Name of the user making the payment
	Amount	Number	Amount of the payment
	Date	Date	Date of the payment
	Payment Type	String	Type of payment (Card, Wallet, Cash)
	Status	String	Status of the payment
Reviews	Review ID	String	Unique identifier for each review
	User	String	Name of the user giving the review
	Driver	String	Name of the driver being reviewed
	Rating	Number	Rating given to the driver

5.2 Schema

This subsection outlines the schema of the SafarRehnuma platform, detailing the structure of the database tables, their relationships, and attributes.

5.2.1 User Schema

```
const userSchema = new mongoose.Schema(
 {
   username: {
      type: String,
      required: [true, "Please Enter Your Username"],
      maxLength: [50, "Username cannot be more than 50 characters"],
   },
   email: {
      type: String,
      required: [true, "Please Enter Your Email"],
      unique: true,
      validate: [validator.isEmail, "Please Enter A Valid Email"],
   },
   password: {
      type: String,
      required: [true, "Please Enter Your Password"],
      minLength: [8, "Password must be at least 8 characters"],
      select: false,
   },
   photo: {
```

```
        type: String,
    },
    role: {
        type: String,
        enum: ["user", "admin"],
        default: "user",
    },
    position: {
        type: String,
        enum: ["driver", "passenger"],
        default: "passenger",
    },
    resetPasswordToken: String,
    resetPasswordExpire: Date,
  },
  { timestamps: true }
);
```

5.2.2 Rides Schema

```
const rideSchema = new mongoose.Schema(
  {
    rideId: {
        type: String,
        required: true,
        unique: true,
    },
    passenger: {
        type: String,
        required: true,
    },
    driver: {
        type: String,
        required: true,
    },
    origin: {
        type: String,
        required: true,
    },
    destination: {
        type: String,
        required: true,
    },
    fare: {
        type: Number,
        required: true,
    },
  },
  { timestamps: true }
);
```

5.2.3 Bookings Schema

```
const bookingSchema = new mongoose.Schema(
 {
   bookingId: {
      type: String,
      required: true,
      unique: true,
   },
   user: {
      type: String,
      required: true,
   },
   trip: {
      type: String,
      required: true,
   },
   date: {
      type: Date,
      default: Date.now,
   },
 },
 { timestamps: true }
);
```

5.2.4 Payments Schema

```
const paymentSchema = new mongoose.Schema(
 {
   paymentId: {
      type: String,
      required: true,
      unique: true,
   },
   user: {
      type: String,
      required: true,
   },
   amount: {
      type: Number,
      required: true,
   },
   date: {
      type: Date,
      default: Date.now,
   },
 },
 { timestamps: true }
); },
 { timestamps: true }
);
```

5.2.5 Vehicles Schema

```
const mongoose = require('mongoose');
```

```
const vehicleSchema = new mongoose.Schema(
  {
    owner: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User',
      required: true,
    },
    licensePlate: {
      type: String,
      required: [true, "Please enter the vehicle's license plate"],
      unique: true,
    },
    make: {
      type: String,
      required: [true, "Please enter the vehicle's make"],
    },
    model: {
      type: String,
      required: [true, "Please enter the vehicle's model"],
    },
    year: {
      type: Number,
      required: [true, "Please enter the vehicle's year of manufacture"],
    },
    color: {
      type: String,
      required: [true, "Please enter the vehicle's color"],
    },
    capacity: {
      type: Number,
      required: [true, "Please enter the vehicle's seating capacity"],
    },
    status: {
      type: String,
      enum: ['available', 'unavailable', 'in maintenance'],
      default: 'available',
    },
    location: {
      type: String,
      required: [true, "Please enter the vehicle's current location"],
    },
  },
  { timestamps: true }
);

module.exports = mongoose.model('Vehicle', vehicleSchema);;
```

5.2.6 Feedback Schema

```
const mongoose = require('mongoose');

const feedbackSchema = new mongoose.Schema(
  {
    user: {
```

```
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User',
      required: true,
    },
    trip: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Trip',
      required: true,
    },
    rating: {
      type: Number,
      required: [true, "Please provide a rating"],
      min: 1,
      max: 5,
    },
    comment: {
      type: String,
      required: [true, "Please provide feedback"],
      maxLength: [500, "Feedback cannot exceed 500 characters"],
    },
    createdAt: {
      type: Date,
      default: Date.now,
    },
  },
  { timestamps: true }
);

module.exports = mongoose.model('Feedback', feedbackSchema);
```

6. Human Interface Design

This section outlines the user interface design principles and components for SafarRehnuma, ensuring an intuitive and seamless experience for both mobile app users and admin dashboard operators.

6.1 Screen Images

This subsection presents the screen images showing the user interface for the SafarRehnuma platform.

6.1.1 Splash Screen

This figure shows the splash screen of the SafarRehnuma application.

Figure 30 Splash Screen

6.1.2 Welcome Screen

This figure shows the welcome screen of the SafarRehnuma application.

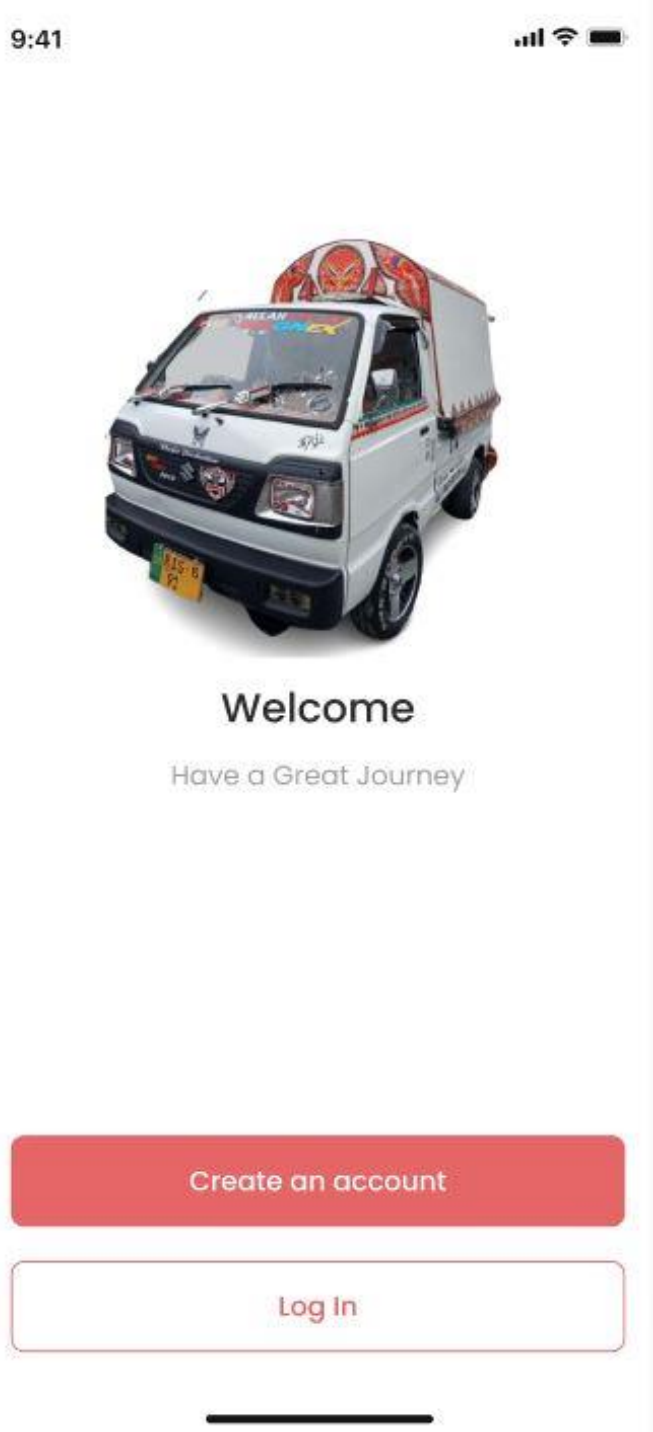


Figure 31 Welcome Screen

6.1.3 Sign Up Screen

This figure shows the Sign-Up screen of the SafarRehnuma application.

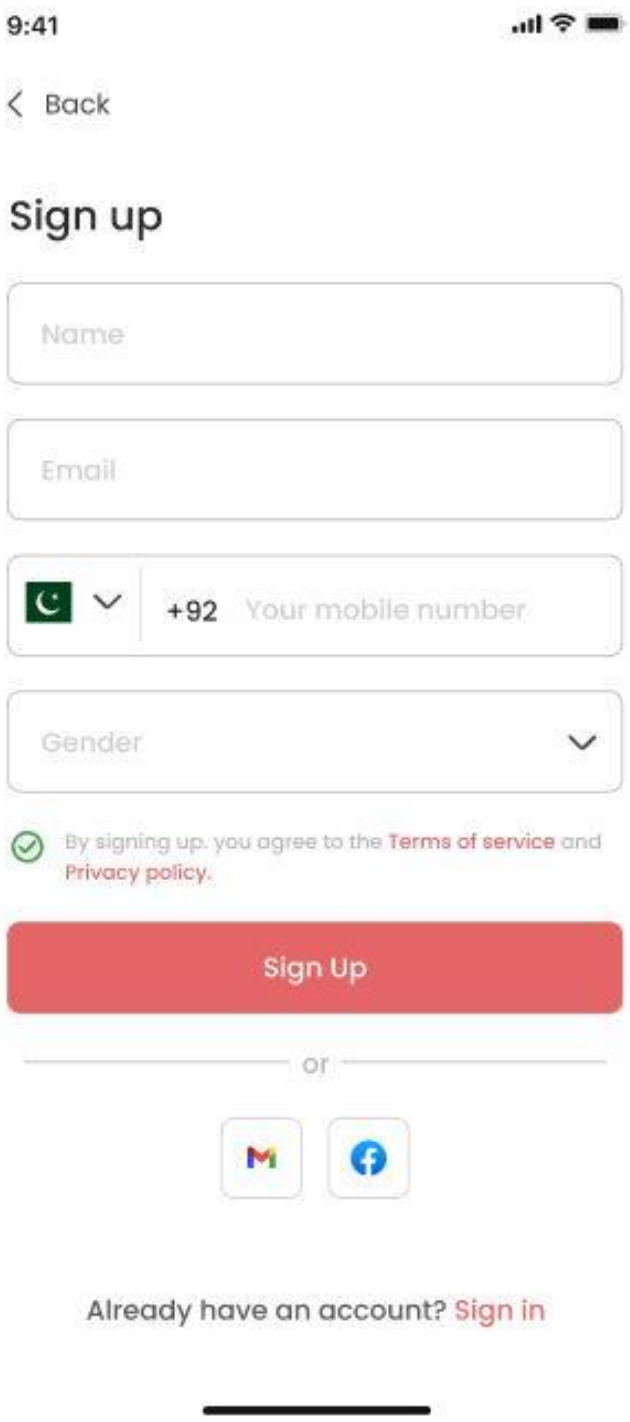


Figure 32 Sign Up Screen

6.1.4 Home Screen

This figure shows the Sign-Up screen of the SafarRehnuma application.

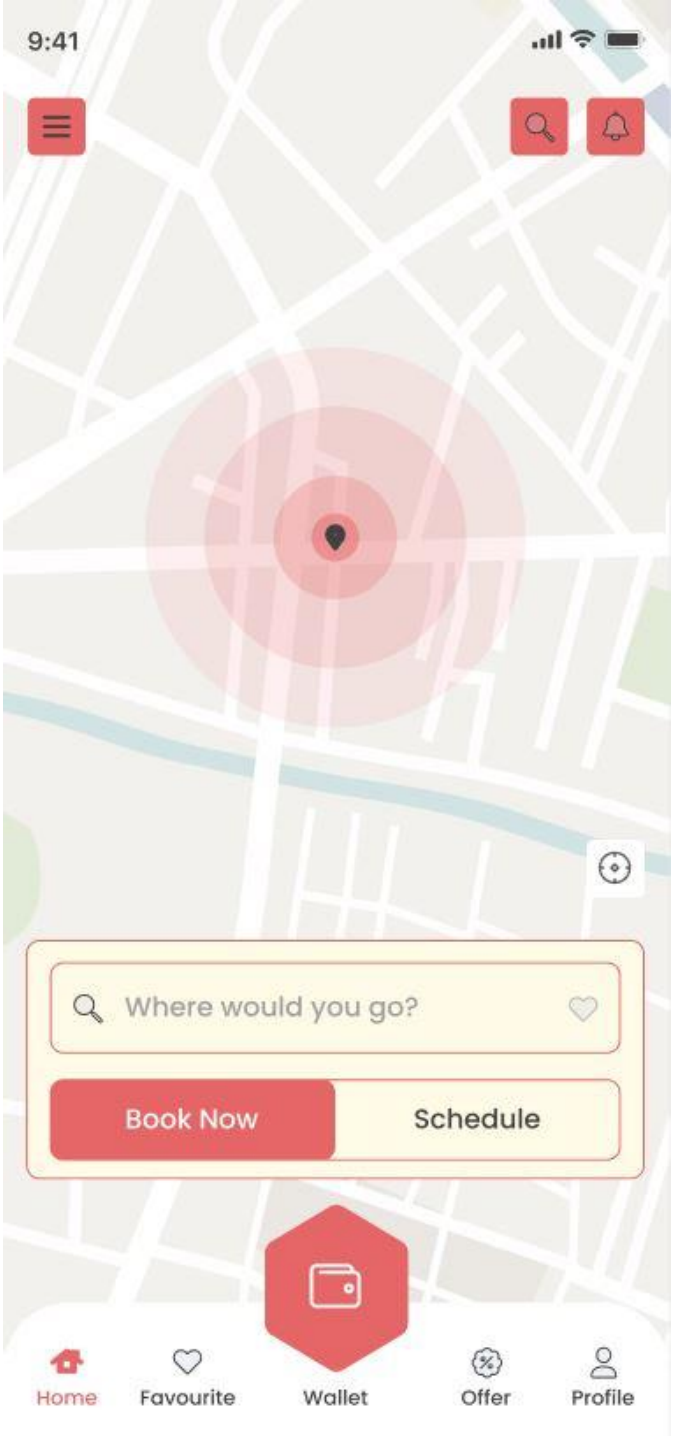


Figure 33 Home Screen

6.1.5 Search Vehicles Screen

This figure shows the Search Vehicles screen of the SafarRehnuma application.

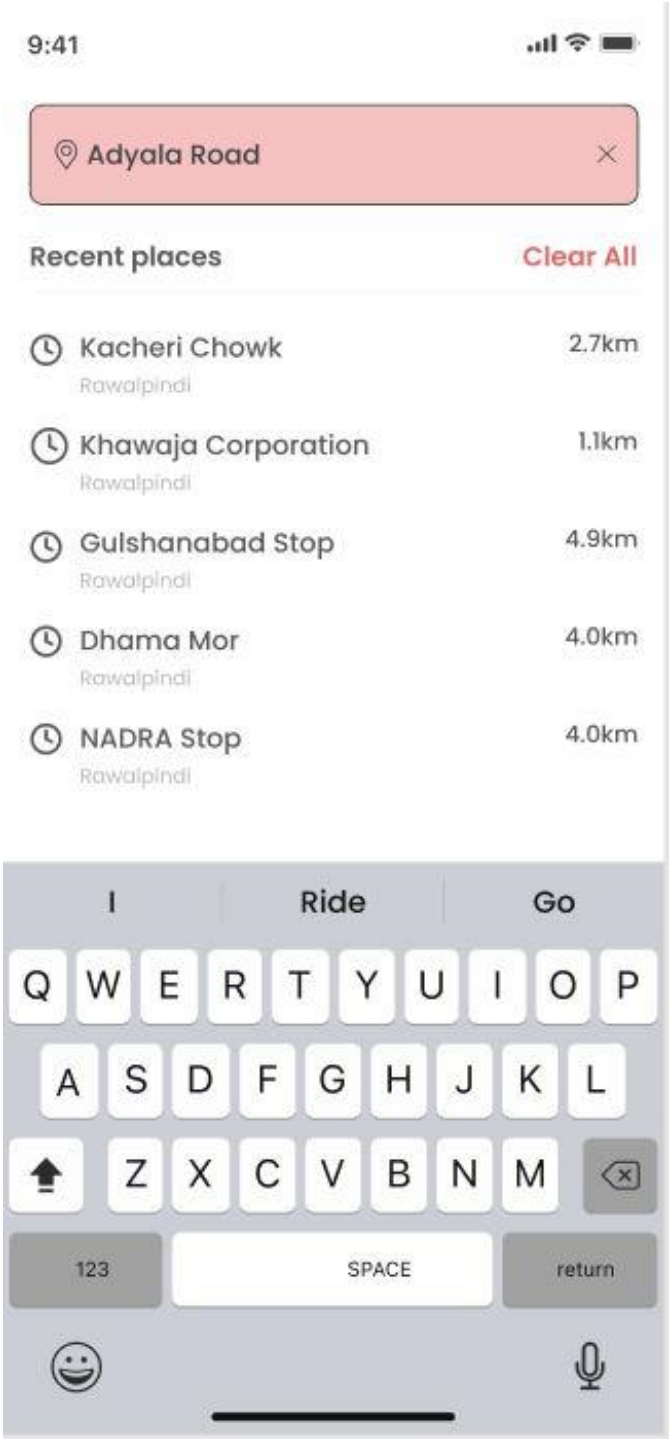


Figure 34 Search Vehicles

6.1.6 Available Vehicles Screen

This figure shows the Available Vehicles screen of the SafarRehnuma application.

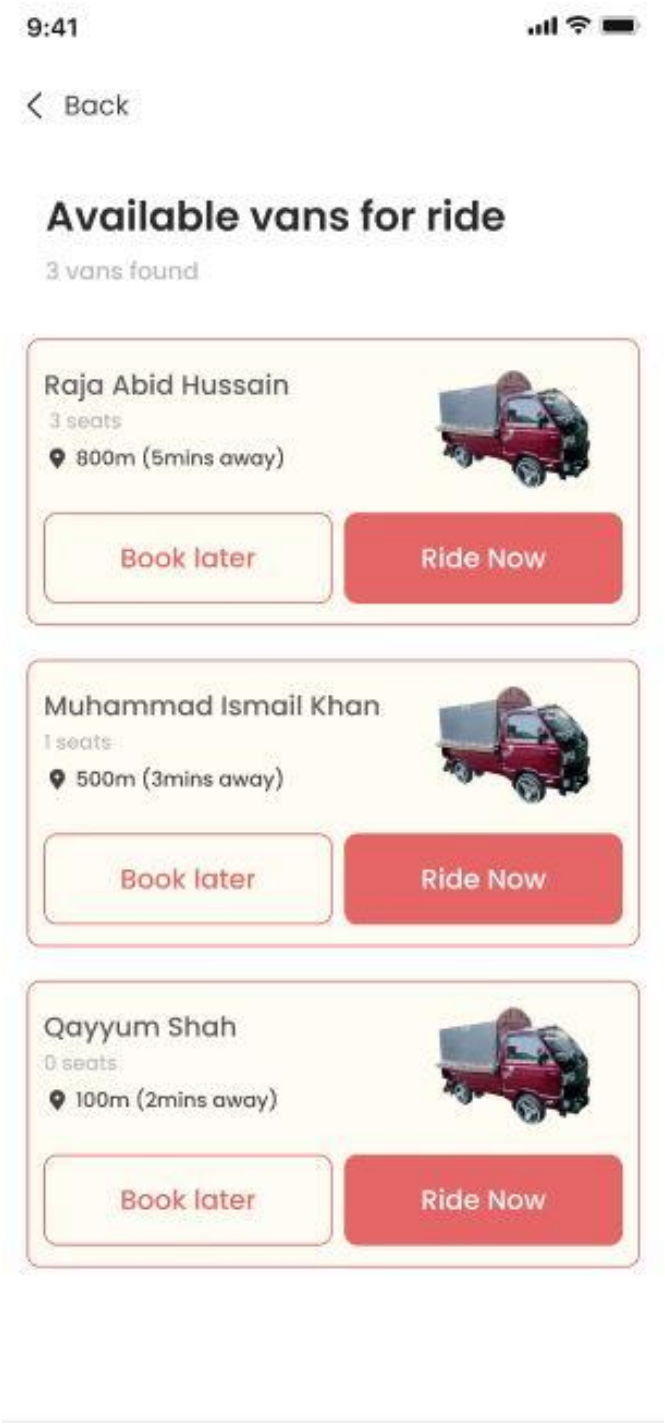


Figure 35 Available Vehicles

6.1.7 Request Ride Screen

This figure shows the Request Ride screen of the SafarRehnuma application.

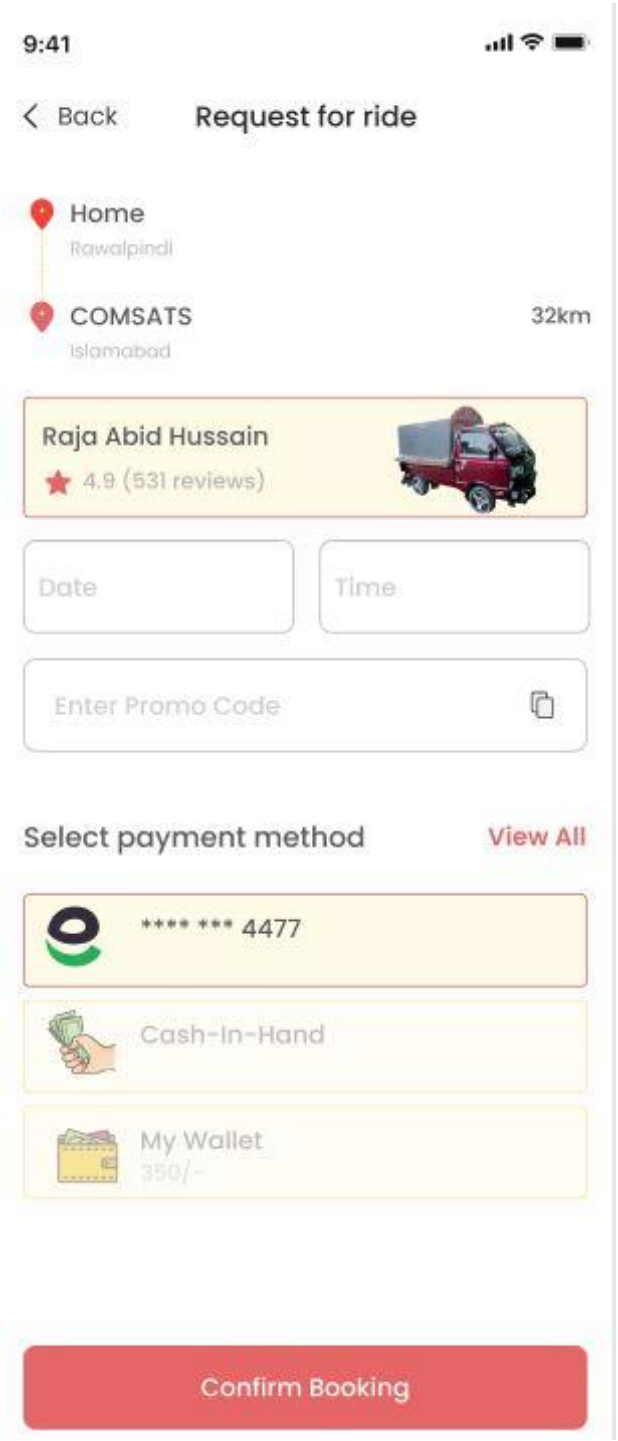


Figure 36 Request Ride

6.1.8 Review Screen

This figure shows the Review screen of the SafarRehnuma application.

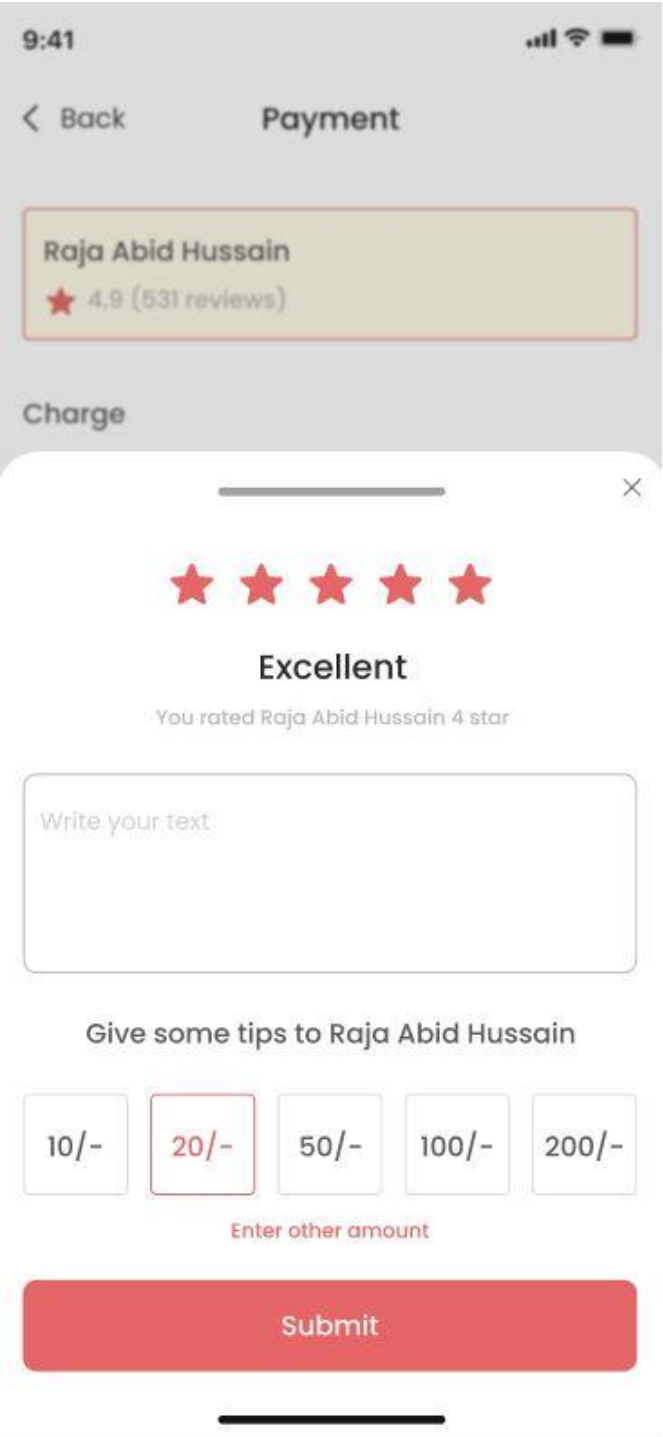


Figure 37 Review Screen

6.1.9 Wallet Screen

This figure shows the Wallet screen of the SafarRehnuma application.

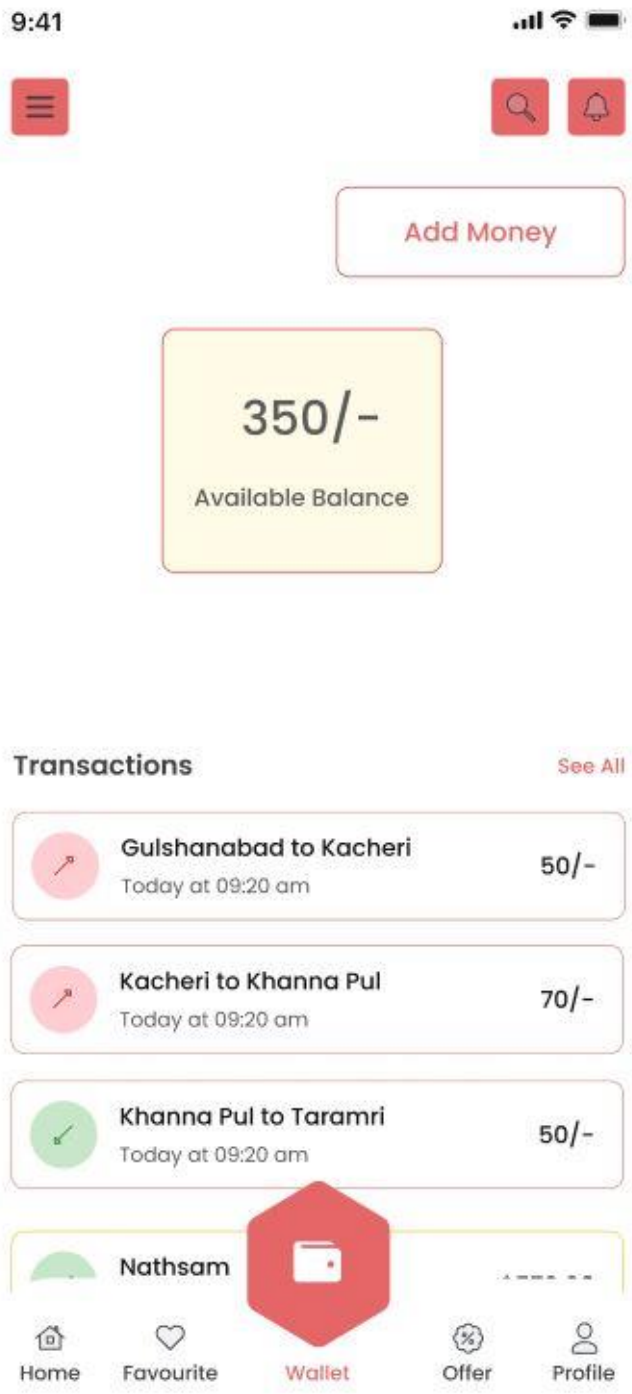


Figure 38 Wallet Screen

6.1.10 Contact Us Screen

This figure shows the Contact Us screen of the SafarRehnuma application.

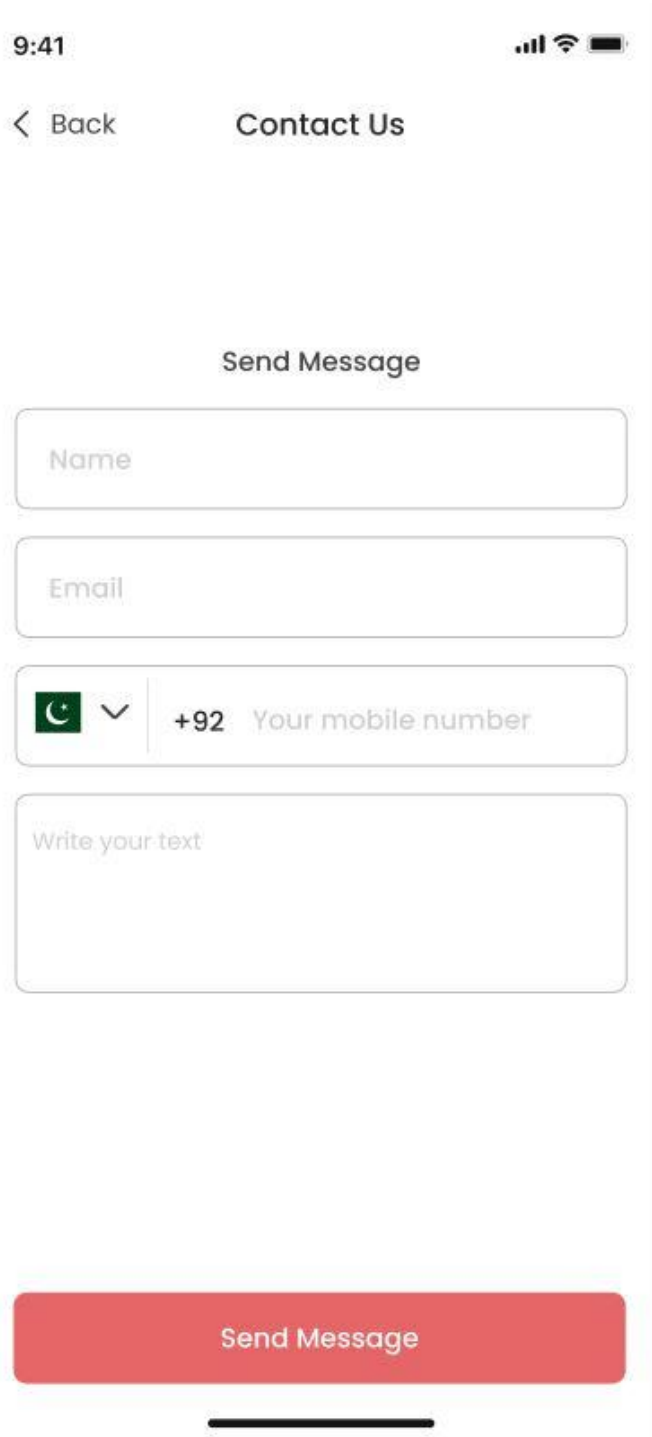


Figure 39 Contact Us

6.1.11 Registered Vehicles Screen

This figure shows the Registered Vehicles screen of the SafarRehnuma application.

Figure 40 Registered Vehicles

6.1.12 Admin Dashboard Screen (Web)

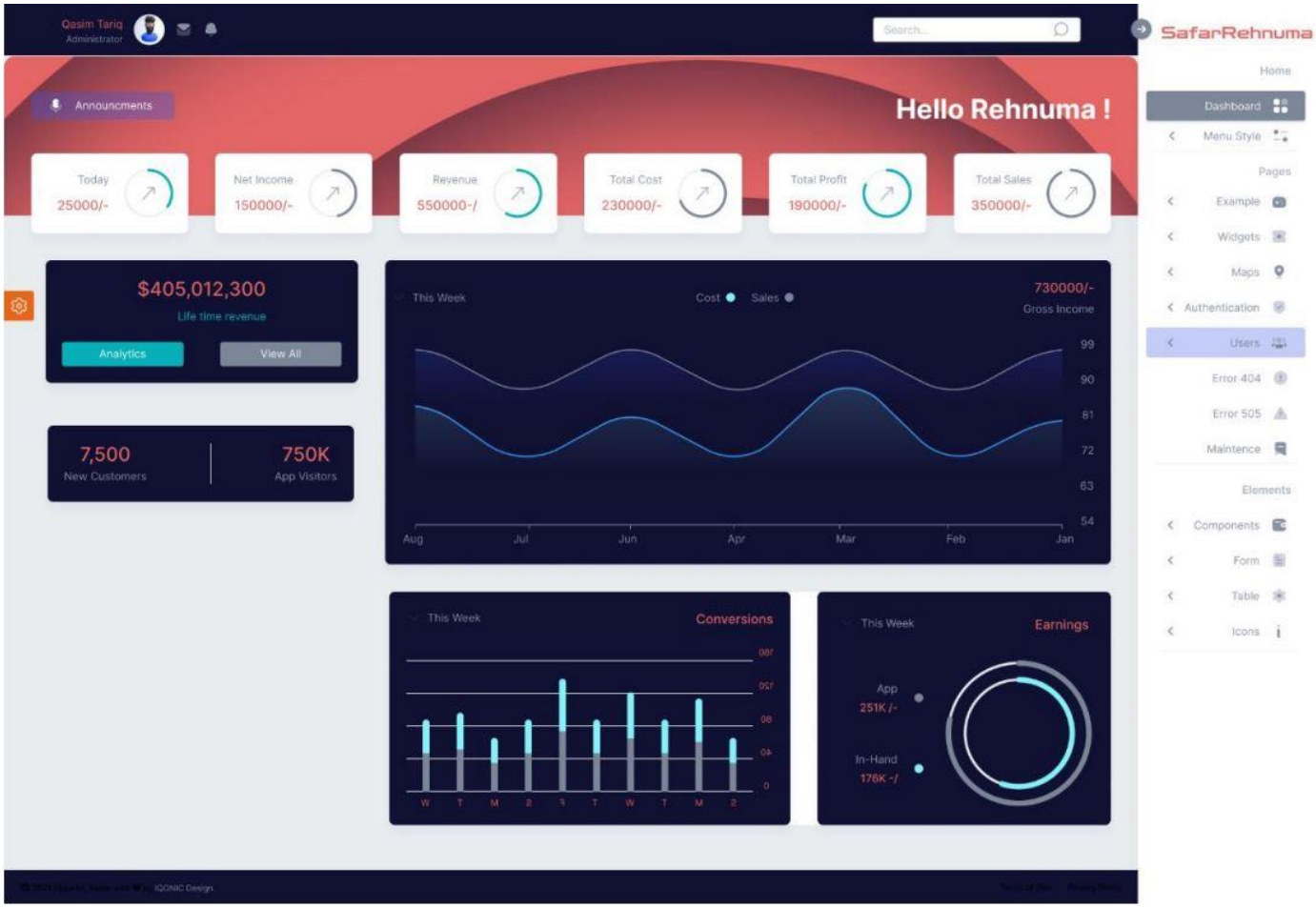


Figure 41 Admin Dashboard

6.2 Screen Objects and Actions

This subsection outlines the different screens and user interactions present in the SafarRehnuma application.

6.2.1 Splash Screen

The splash screen of Safar Rehnuma displays the logo and serves as an introductory screen while the application loads.

6.2.2 Welcome Screen

The welcome screen greets users after the splash screen, providing an overview of the app's features and options to log in or sign up.

6.2.3 Signup Screen

On the signup screen, new users can create accounts by entering details like username, email and password. It includes form fields and a submission button for user registration.

6.2.4 Home Screen

The home screen serves as the main hub of Safar Rehnuma, offering access to essential features like booking a ride, viewing ride history, managing profile, etc.

6.2.5 Search Vehicles Screen

In the search vehicles screen, users can search for available vehicles based on preferences like location, vehicle type, etc. It provides search filters and options to refine results.

6.2.6 Available Vehicles Screen

The available vehicles screen presents users with a list of vehicles ready for booking, displaying details such as vehicle type, driver info, and availability status.

6.2.7 Request Ride Screen

Users can request rides on the request ride screen by specifying pickup and drop-off locations, along with other ride preferences. It offers options to customize the request and confirm booking.

6.2.8 Review Screen

After completing a trip, users can rate and review their ride experience on the review screen. It includes form fields for feedback, driver rating, etc., and a submission option.

6.2.9 Wallet Screen

The wallet screen showcases users' digital wallets, including account balance, transaction history, and options for adding or withdrawing funds.

6.2.10 Contact Us Screen

Users can contact customer support or service representatives via the contact us screen, which provides contact details like phone numbers, email addresses, and a feedback form.

6.2.11 Registered Vehicles Screen

Accessible to administrators, the registered vehicles screen displays a list of vehicles registered on Safar Rehnuma, along with details like vehicle type, registration status, owner info, etc.

6.2.12 Admin Dashboard Screen (Web)

The web-based admin dashboard offers administrators comprehensive control over Safar Rehnuma, with functionalities such as user management, ride monitoring, analytics, etc.

7. Implementation

This section covers the practical implementation aspects of SafarRehnuma, encompassing algorithmic implementations, integration with external interfaces, and the development of user interfaces for seamless interaction.

7.1 Algorithms

This subsection explains the algorithms implemented within SafarRehnuma, highlighting their roles in optimizing routes, calculating dynamic pricing, and managing ride requests efficiently.

7.1.1 Algorithm: Dynamic Pricing

Algorithm 1.1 Set Pricing

Input: Admin ID, Route ID, Fare

Output: Success message or error message

```
begin
function (on("connection", (socket) ) => {
  socket.on("set-pricing", (data) => {
    if (data) {
      const { adminId, routeId, fare } = data;
      const admin = getAdmin(adminId);
      if (admin && admin.hasPermission("setPricing")) {
        const route = getRoute(routeId);
        setPricing(route, fare);
        socket.emit("pricing-set", {
          message: "Pricing for the specified route has been set successfully",
        });
      } else {
        socket.emit("error", {
          message: "Admin does not have permission to set pricing",
        });
      }
    }
  });
}
end
```

Algorithm 1.2 Dynamic Fare Adjustment

Input: None

Output: Confirmation message indicating dynamic fare adjustments have been made, if applicable

```
begin
function (on("connection", (socket) ) => {
  const dynamicFareAdjustment = () => {
    const changesDetected = detectChangesInDemand();
    if (changesDetected) {
      const fareAdjustments = calculateDynamicFareAdjustments();
```

```
    applyFareAdjustments(fareAdjustments);
    io.emit("fare-adjustments", {
      message: "Fares have been dynamically adjusted",
    });
  }
};

setInterval(dynamicFareAdjustment, 60000); // Check every minute
}
end
```

Algorithm 1.3 Update Fare Structure	**Algorithm 1.4 View Pricing Analytics**
Input: Admin ID, New fare structure	**Input:** Admin ID
Output: Confirmation message indicating successful update of the fare structure, or an error message if the admin lacks permission.	**Output:** Analytics data, if the admin has permission to view analytics; otherwise, an error message.

```
begin
function (on("connection", (socket) ) => {
  socket.on("update-fare-structure", (data) => {
    if (data) {
      const { adminId, newFareStructure } = data;
      const admin = getAdmin(adminId);
      if (admin && admin.hasPermission("updateFareStructure")) {
        updateFareStructure(newFareStructure);
        socket.emit("fare-structure-updated", {
          message: "Fare structure has been updated successfully",
        });
      } else {
        socket.emit("error", {
          message: "Admin does not have permission to update fare structure",
        });
      }
    }
  });
}
end
```

```
begin
function (on("connection", (socket) ) => {
  socket.on("view-pricing-analytics", (data) => {
    if (data) {
      const { adminId } = data;
      const admin = getAdmin(adminId);
      if (admin && admin.hasPermission("viewAnalytics")) {
        const analytics = getPricingAnalytics();
        socket.emit("display-analytics", {
          analytics: analytics,
        });
      } else {
        socket.emit("error", {
          message: "Admin does not have permission to view analytics",
        });
      }
    }
  });
}
end
```

Algorithm 1.5 Estimate Fares

Input: Passenger ID, Origin, Destination

Output: Estimated fare for the journey, or an error message if the passenger is not found.

```
begin
function (on("connection", (socket) ) => {
  socket.on("estimate-fare", (data) => {
    if (data) {
      const { passengerId, origin, destination } = data;
      const passenger = getPassenger(passengerId);
      if (passenger) {
```

```
      const estimatedFare = calculateFareEstimate(origin, destination);
      socket.emit("fare-estimate", {
        estimate: estimatedFare,
      });
    } else {
      socket.emit("error", {
        message: "Passenger not found",
      });
    }
  }
 });
}
end
```

7.1.2 Algorithm: Route Optimization and Suggestions

Algorithm 2.1 Suggest Routes

Input: Passenger ID, Destination

Output: Route suggestions for the given destination, or an error message if the passenger is not found.

```
begin
function (on("connection", (socket) ) => {
 socket.on("suggest-routes", (data) => {
   if (data) {
     const { passengerId, destination } = data;
     const passenger = getPassenger(passengerId);
     if (passenger) {
       const routeSuggestions = generateRouteSuggestions(destination);
       socket.emit("route-suggestions", {
         suggestions: routeSuggestions,
       });
     } else {
       socket.emit("error", {
         message: "Passenger not found",
       });
     }
   }
 });
}
end
```

Algorithm 2.2 Optimize Route

Input: Passenger ID, Route ID

Output: Optimized route for the given passenger and route ID, or an error message if the passenger is not found.

```
begin
function (on("connection", (socket) ) => {
 socket.on("optimize-route", (data) => {
   if (data) {
     const { passengerId, routeId } = data;
     const passenger = getPassenger(passengerId);
```

```
      if (passenger) {
        const optimizedRoute = optimizeRoute(routeId);
        socket.emit("optimized-route", {
          route: optimizedRoute,
        });
      } else {
        socket.emit("error", {
          message: "Passenger not found",
        });
      }
    }
  });
}
end
```

Algorithm 2.3 Get Real-Time Traffic Updates	**Algorithm 2.4 Navigate Through Recommended Routes**
Input: None	**Input:** Passenger ID, Route ID
Output: Real-time traffic updates emitted to all connected clients.	**Output:** Navigation instructions for the recommended route, or an error message if the passenger is not found.

```
begin
function (on("connection", (socket) ) => {
  const realTimeTrafficUpdates = () => {
    const trafficUpdates = getRealTimeTrafficUpdates();
    if (trafficUpdates) {
      io.emit("traffic-updates", {
        updates: trafficUpdates,
      });
    }
  };

  setInterval(realTimeTrafficUpdates, 30000); // Check every 30 seconds
}
end
```

```
begin
function (on("connection", (socket) ) => {
  socket.on("navigate-recommended-route", (data) => {
    if (data) {
      const { passengerId, routeId } = data;
      const passenger = getPassenger(passengerId);
      if (passenger) {
        const navigationInstructions = getNavigationInstructions(routeId);
        socket.emit("navigation-instructions", {
          instructions: navigationInstructions,
        });
      } else {
        socket.emit("error", {
          message: "Passenger not found",
        });
      }
    }
  });
}
end
```

7.2 External APIs

This table lists the external APIs used in the SafarRehnuma project.

Table 1 External APIs in SafarRehnuma

Name of API	Description of API	Purpose of usage	API Endpoint/Function/Class
Google Maps API	Provides geolocation and mapping services.	To enable real-time location tracking, route optimization, and navigation features.	googleMapsClient.geocode googleMapsClient.directions
Twilio	Offers messaging and communication services.	To send SMS notifications and alerts to users about trip status, payment confirmations, and other updates.	twilio.messages.create
Stripe	Integrates credit card payment processing.	To handle ride payment transactions securely.	stripe.paymentMethods.create stripe.charges.create
MongoDB Atlas (v4.2)	A cloud database service that provides scalable, secure, and globally accessible databases.	To store and manage the application's data, including users, vehicles, trips, bookings, and payments.	mongoose.connect mongoose.model

7.3 Deployment

The deployment environments for Safar Rehnuma encompass various hosting and cloud services to ensure a seamless and efficient operation of all its subsystems.

- The mobile application, developed using Flutter SDK version 3.7.0 and Dart version 2.18.0, is hosted on both Google Play Store and Apple App Store. Firebase services, including Authentication, Firestore, and Cloud Messaging, are integrated to manage backend functionalities such as user authentication, real-time data updates, and notifications.
- The backend services are hosted on Amazon Web Services (AWS), utilizing Node.js version 16.14.0 and Express version 4.17.1 for server-side operations. MongoDB Atlas version 4.2, managed through Mongoose version 6.1.2, is used for database management. Additionally, the backend integrates various third-party APIs such as Stripe (version 2020-08-27) for payment processing, Twilio for communication services,

- The admin dashboard, developed using the MERN stack, is hosted on AWS Amplify and Heroku. The dashboard utilizes React.js version 17.0.2 and Redux version 4.1.0 for frontend state management, along with Node.js and Express for backend services, ensuring seamless administration and monitoring of the system.
- MongoDB Atlas serves as the primary database, offering robust and scalable data management capabilities. Continuous Integration/Continuous Deployment (CI/CD) processes are managed through GitHub Actions and AWS CodePipeline, facilitating automated testing, deployment, and updates. For real-time location services, the application leverages Google Cloud Platform's Google Maps API version 3, and Firebase Realtime Database, ensuring accurate and efficient location tracking.

8. Testing and Evaluation

Once the system has been successfully developed, testing must be performed to ensure that the system working as intended. This is also to check that the system meets the requirements stated. earlier.

8.1 Unit Testing

This subsection lists the unit testing and its details for the SafarRehnuma system.

8.1.1 Unit Testing 1: Home Screen

Testing Objective: To ensure the Home Screen component renders correctly and displays the expected content.

No.	Test case/Test script	Attribute and value	Expected result	Result
1	Check if the HomeScreen component renders without any errors.	Component should render without errors	Component should render without errors	Pass
2	Check if the header displays the correct text	Header Text: 'Welcome to Safar Rehnuma'	The header displays the provided text	Pass
3	Check if the SearchBar component is visible	SearchBar should be visible on the screen	SearchBar is visible on the screen	Pass
4	Check if the "Recent Rides" section is	The section should be visible on the screen	"Recent Rides" section is visible	Pass

	visible		on the screen	
5	Check if the "Available Vehicles" section is visible	The section should be visible on the screen	"Available Vehicles" section is visible on the screen	Pass
6	Check if the ride cards are displayed correctly	Verify the rendering of ride cards with sample data	Ride cards should be displayed with the provided data	Pass
7	Check if the vehicle components are displayed correctly	Verify the rendering of vehicle components with sample data	Vehicle components should be displayed with the provided data	Pass

8.1.2 Unit Testing 2: User Management

Testing Objective: To ensure the User Management screens render correctly and perform the expected actions.

No.	Test case/Test script	Attribute and value	Expected result	Result
1	Check if the Registration screen renders without any errors.	Component should render without errors	Component should render without errors	Pass
2	Check if the Login screen renders without any errors.	Component should render without errors	Component should render without errors	Pass
3	Check if the profile information is displayed correctly after login	Profile information should be visible and accurate	Profile information is displayed correctly	Pass
4	Check if the user can update profile information	Enter new profile information and save	Profile information is updated successfully	Pass
5	Check if the user can log out successfully	Click on "Logout" button	User is logged out and redirected to the Home Screen	Pass

8.1.3 Unit Testing 3: Vehicle Registration

Testing Objective: To ensure the Vehicle Registration screens render correctly and perform the expected actions.

No.	Test case/Test script	Attribute and value	Expected result	Result
1	Check if the Vehicle Registration screen renders without any errors.	Component should render without errors	Component should render without errors	Pass
2	Check if the registered vehicles list is displayed correctly	List of registered vehicles should be visible	Registered vehicles list is displayed correctly	Pass
3	Check if the user can register a new vehicle	Enter vehicle information and save	Vehicle is registered successfully	Pass
4	Check if the user can update vehicle information	Enter updated vehicle information and save	Vehicle information is updated successfully	Pass
5	Check if the user can delete a registered vehicle	Click on "Delete" button next to a vehicle	Vehicle is deleted successfully	Pass

8.1.4 Unit Testing 4: Ride Management

Testing Objective: To ensure the Ride Management screens render correctly and perform the expected actions.

No.	Test case/Test script	Attribute and value	Expected result	Result
1	Check if the Ride Booking screen renders without any errors.	Component should render without errors	Component should render without errors	Pass
2	Check if the ride details are displayed correctly	Ride details should be visible and accurate	Ride details are displayed correctly	Pass
3	Check if the user can book a ride	Enter ride details and confirm booking	Ride is booked successfully	Pass
4	Check if the user can cancel a ride	Click on "Cancel Ride" button	Ride is canceled successfully	Pass
5	Check if the user can track current ride	Click on "Track Ride" button	Current ride tracking is	Pass

			displayed correctly	

8.1.5 Unit Testing 5: Payments

Testing Objective: To ensure the Payments screens render correctly and perform the expected actions.

No.	Test case/Test script	Attribute and value	Expected result	Result
1	Check if the Payment Processing screen renders without any errors.	Component should render without errors	Component should render without errors	Pass
2	Check if the payment history is displayed correctly	Payment history should be visible and accurate	Payment history is displayed correctly	Pass
3	Check if the user can add a new payment method	Enter payment method details and save	Payment method is added successfully	Pass
4	Check if the user can make a payment	Enter payment details and confirm payment	Payment is processed successfully	Pass
5	Check if the user can delete a payment method	Click on "Delete" button next to a payment method	Payment method is deleted successfully	Pass

8.2 Functional Testing

This subsection lists the functional testing for the SafarRehnuma system which will take place after the unit testing is concluded. In functional testing, the functionality of each of the modules is tested, this is to ensure that the system produced meets the specifications and requirements.

8.2.1 Functional Testing 1: User Registration

Objective: To ensure that the user is registered into the system after successfully creating an account.

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Render Registration	Component rendering	Component renders without errors	Component renders without	Pass

	Form			errors	
2	Input Fields	Username, email, password	All fields are visible and functional	All fields are visible and functional	Pass
3	Form Submission	Valid input data	User is registered successfully	User is registered successfully	Pass
4	Error Handling	Invalid email format	Error message "Invalid email format" is displayed	Error message "Invalid email format" is displayed	Pass
5	Password Validation	Password < 6 characters	Error message "Password too short" is displayed	Error message "Password too short" is displayed	Pass

8.2.2 Functional Testing 2: User Login

Objective: To ensure that the user has logged in to the system after successfully creating an account.

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Render Login Form	Component rendering	Component renders without errors	Component renders without errors	Pass
2	Input Fields	Username and password	All fields are visible and functional	All fields are visible and functional	Pass
3	Form Submission	Valid credentials	User is logged in successfully	User is logged in successfully	Pass
4	Error Handling	Invalid credentials	Error message "Invalid username or password" displayed	Error message "Invalid username or password" displayed	Pass
5	Session Management	Post login	User session is maintained	User session is maintained	Pass

8.2.3 Functional Testing 3: Register a Vehicle

Objective: To ensure that the vehicle registration process works correctly.

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Render Registration Form	Component rendering	Component renders without errors	Component renders without errors	Pass
2	Input Fields	Vehicle details	All fields are visible and functional	All fields are visible and functional	Pass
3	Form Submission	Valid input data	Vehicle is registered successfully	Vehicle is registered successfully	Pass
4	Error Handling	Missing input data	Error message "Please fill all fields" is displayed	Error message "Please fill all fields" is displayed	Pass
5	Confirmation Message	Post registration	Confirmation message "Vehicle registered successfully" displayed	Confirmation message "Vehicle registered successfully" displayed	Pass

8.2.4 Functional Testing 4: Book a Ride

Objective: To ensure that the ride booking process works correctly.

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Render Booking Form	Component rendering	Component renders without errors	Component renders without errors	Pass
2	Input Fields	Pickup and destination	All fields are visible and functional	All fields are visible and functional	Pass
3	Form Submission	Valid input data	Ride is booked successfully	Ride is booked successfully	Pass
4	Error Handling	Missing input data	Error message "Please fill all fields" is displayed	Error message "Please fill all fields" is displayed	Pass

5	Confirmation Message	Post booking	Confirmation message "Ride booked successfully" displayed	Confirmation message "Ride booked successfully" displayed	Pass

8.2.5 Functional Testing 5: Cancel Booking

Objective: To ensure that the booking cancellation process works correctly.

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Render Cancellation Form	Component rendering	Component renders without errors	Component renders without errors	Pass
2	Input Fields	Booking ID	All fields are visible and functional	All fields are visible and functional	Pass
3	Form Submission	Valid booking ID	Booking is cancelled successfully	Booking is cancelled successfully	Pass
4	Error Handling	Invalid booking ID	Error message "Invalid booking ID" is displayed	Error message "Invalid booking ID" is displayed	Pass
5	Confirmation Message	Post cancellation	Confirmation message "Booking cancelled successfully" displayed	Confirmation message "Booking cancelled successfully" displayed	Pass

8.2.6 Functional Testing 6: Make Payment

Objective: To ensure that the payment process works correctly.

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Render Payment	Component rendering	Component renders without errors	Component renders without	Pass

	Form			errors	
2	Input Fields	Card details	All fields are visible and functional	All fields are visible and functional	Pass
3	Form Submission	Valid card details	Payment is processed successfully	Payment is processed successfully	Pass
4	Error Handling	Invalid card details	Error message "Invalid card details" is displayed	Error message "Invalid card details" is displayed	Pass
5	Confirmation Message	Post payment	Confirmation message "Payment successful" displayed	Confirmation message "Payment successful" displayed	Pass

8.2.7 Functional Testing 7: Submit Review

Objective: To ensure that the review submission process works correctly.

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Render Review Form	Component rendering	Component renders without errors	Component renders without errors	Pass
2	Input Fields	Review text	All fields are visible and functional	All fields are visible and functional	Pass
3	Form Submission	Valid input data	Review is submitted successfully	Review is submitted successfully	Pass
4	Error Handling	Missing input data	Error message "Please fill all fields" is displayed	Error message "Please fill all fields" is displayed	Pass
5	Confirmation Message	Post submission	Confirmation message "Review submitted successfully"	Confirmation message "Review submitted successfully"	Pass

			displayed	displayed	

8.3 Integration Testing

This subsection lists the integration testing for the SafarRehnuma system. Integration tests assess whether a set of classes that must work together do so without error. They ensure that the interfaces and linkages between different parts of the system work properly.

8.3.1 Integration Testing 1: Vehicle Registration

Testing Objective: To ensure the integration between vehicle registration, viewing registered vehicles, and updating vehicle information screens works correctly.

No.	Test case/Test script	Attribute and value	Expected result	Actual result	Result
1	Register New Vehicle	Click on "Register Vehicle" button	Successfully navigate to the Vehicle Registration Screen	Successfully navigate to the Vehicle Registration Screen	Pass
2	View Registered Vehicles	Click on "My Vehicles" tab	Successfully navigate to the Registered Vehicles Screen	Successfully navigate to the Registered Vehicles Screen	Pass
3	Update Vehicle Information	Click on "Edit" button next to a vehicle	Successfully navigate to the Vehicle Information Update Screen	Successfully navigate to the Vehicle Information Update Screen	Pass
4	Delete Vehicle	Click on "Delete" button next to a vehicle	Successfully delete the vehicle and update the vehicle list	Successfully delete the vehicle and update the vehicle list	Pass
5	Return to Home from Vehicle Registration	Click on "Home" tab	Successfully navigate back to the Home Screen	Successfully navigate back to the Home Screen	Pass

8.3.2 Integration Testing 2: Dynamic Pricing

Testing Objective: To ensure the integration between ride fare estimation, applying dynamic pricing, and viewing fare breakdown screens works correctly.

No.	Test case/Test script	Attribute and value	Expected result	Actual result	Result
1	Estimate Ride Fare	Enter ride details and click "Estimate Fare"	Successfully display estimated fare based on entered details	Successfully display estimated fare based on entered details	Pass
2	Apply Dynamic Pricing	View fare with dynamic pricing applied	Successfully apply and display dynamic pricing	Successfully apply and display dynamic pricing	Pass
3	View Fare Breakdown	Click on "Fare Breakdown" button	Successfully navigate to the Fare Breakdown Screen	Successfully navigate to the Fare Breakdown Screen	Pass
4	Confirm Fare and Book Ride	Click on "Confirm Fare" and "Book Ride"	Successfully confirm fare and navigate to Ride Booking Screen	Successfully confirm fare and navigate to Ride Booking Screen	Pass
5	Return to Home from Dynamic Pricing	Click on "Home" tab	Successfully navigate back to the Home Screen	Successfully navigate back to the Home Screen	Pass

8.3.3 Integration Testing 3: Location Tracking

Testing Objective: To ensure the integration between real-time location tracking, viewing current location, and updating location preferences screens works correctly.

No.	Test case/Test script	Attribute and value	Expected result	Actual result	Result
1	Track Current Location	Click on "Track Location" button	Successfully display real-time current location	Successfully display real-time current location	Pass
2	View Location History	Click on "Location History" tab	Successfully navigate to the Location History Screen	Successfully navigate to the Location History Screen	Pass
3	Update Location Preferences	Click on "Preferences" in Location Tracking	Successfully navigate to the Location Preferences Screen	Successfully navigate to the Location	Pass

				Preferences Screen	
4	Save Updated Location Preferences	Click "Save" on Location Preferences Screen	Successfully save and apply updated location preferences	Successfully save and apply updated location preferences	Pass
5	Return to Home from Location Tracking	Click on "Home" tab	Successfully navigate back to the Home Screen	Successfully navigate back to the Home Screen	Pass

8.3.4 Integration Testing 4: Route Optimization and Suggestions

Testing Objective: To ensure the integration between route suggestion, route optimization, and real-time traffic updates screens works correctly.

No.	Test case/Test script	Attribute and value	Expected result	Actual result	Result
1	Suggest Routes	Enter destination and click "Suggest Routes"	Successfully display optimized route suggestions	Successfully display optimized route suggestions	Pass
2	Optimize Route	Select a route and click "Optimize"	Successfully apply optimization and display optimized route	Successfully apply optimization and display optimized route	Pass
3	Get Real-Time Traffic Updates	Click on "Traffic Updates" button	Successfully display real-time traffic updates	Successfully display real-time traffic updates	Pass
4	Navigate Through Recommended Routes	Select a recommended route and click "Navigate"	Successfully initiate navigation through recommended route	Successfully initiate navigation through recommended route	Pass
5	Return to Home from Route Optimization	Click on "Home" tab	Successfully navigate back to the Home Screen	Successfully navigate back to the Home Screen	Pass