ROAD GUARDIAN

A Roadside Assistance providing platform

Mini Project Report

Submitted by

ROSHAN VARGHESE

Reg. No.: AJC23MCA-2052

In Partial fulfillment for the Award of the Degree of

MASTER OF COMPUTER APPLICATIONS (MCA)

AMAL JYOTHI COLLEGE OF ENGINEERING AUTONOMOUS

KANJIRAPPALLY

[Approved by AICTE, Accredited by NAAC, Accredited by NBA.
Koovappally, Kanjirappally, Kottayam, Kerala – 686518]

2024-2025

DEPARTMENT OF COMPUTER APPLICATIONS

AMAL JYOTHI COLLEGE OF ENGINEERING AUTONOMOUS

KANJIRAPPALLY

CERTIFICATE

This is to certify that the Project report, "**ROAD GUARDIAN"** is the bonafide work of **ROSHAN VARGHESE (Regno: AJC23MCA-2052)** in partial fulfillment of the requirements for the award of the Degree of Master of Computer Applications under **Amal Jyothi College of Engineering Autonomous, Kanjirappally** during the year 2024-25.

Ms.Gloriya Mathew
Internal Coordinator

Mr.Binumon Joseph
Coordinator

Rev. Fr. Dr. Rubin Thottupurathu Jose
Head of the Department

DECLARATION

I hereby declare that the project report **“ROAD GUARDIAN”** is a bonafide work done at Amal Jyothi College of Engineering, towards the partial fulfillment of the requirements for the award of the Master of Computer Applications (MCA) from Amal Jyothi College of Engineering Autonomous during the academic year 2024-2025.

Date:	**ROSHAN VARGHESE**
KANJIRAPPALLY	**Reg: AJC23MCA-2052**

ACKNOWLEDGEMENT

First and foremost, I thank God almighty for his eternal love and protection throughout the project. I take this opportunity to express my gratitude to all who helped me in completing this project successfully. It has been said that gratitude is the memory of the heart. I wish to express my sincere gratitude to our Director (Administration) **Rev. Fr. Dr. Roy Abraham Pazhayaparampil** and Principal **Dr. Lillykutty Jacob** for providing good faculty for guidance.

I own a great depth of gratitude towards our Head of the Department **Rev. Fr. Dr. Rubin Thottupurathu Jose** for helping us. I extend my whole hearted thanks to the project coordinator **Mr.Binumon Joseph** for his valuable suggestions and for overwhelming concern and guidance from the beginning to the end of the project. I would also express sincere gratitude to my guide **Ms.Gloriya Mathew** for her inspiration and helping hand.

I thank our beloved teachers for their cooperation and suggestions that helped me throughout the project. I express my thanks to all my friends and classmates for their interest, dedication, and encouragement shown towards the project. I convey my hearty thanks to my family for the moral support, suggestions, and encouragement to make this venture a success.

ROSHAN VARGHESE

CONTENT

List of Abbreviations

- UML - Unified Modelling Language
- ORM - Object-Relational Mapping
- MVT - Model-View-Template
- MVC - Model-View-Controller
- RDBMS - Relational Database Management System
- 3NF - Third Normal Form
- IDE - Integrated Development Environment
- HTML - HyperText Markup Language
- JS - JavaScript
- CSS - Cascading Style Sheets
- API - Application Programming Interface
- UI - User Interface
- HTTP - Hypertext Transfer Protocol
- URL - Uniform Resource Locator
- Django: Python Framework

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

The Road Guardian project is a comprehensive web-based platform designed to provide real-time support to vehicle owners facing roadside emergencies such as breakdowns, accidents, or other unexpected issues. The system offers a streamlined and efficient method for users to request immediate assistance, helping reduce wait times and ensuring timely, reliable service delivery. By integrating modern technologies, such as GPS-based location tracking, real-time updates, and secure digital payment systems, the platform enhances the overall user experience. The ultimate goal is to connect vehicle owners with service providers seamlessly, improving service accessibility and convenience.

Built using Django for the back-end and HTML/CSS for the front-end, the system integrates modern technologies like real-time tracking, automated service dispatch, and digital payment systems, ensuring a fast, secure, and reliable experience. The admin dashboard allows system operators to monitor performance, manage users, and generate reports. Overall, the platform aims to enhance efficiency, reduce response times, and improve service reliability, delivering a seamless and modern roadside assistance experience.

The system is designed to cater to three distinct user groups:

1. **Vehicle Owners (Users)**: Individuals who require roadside assistance services such as towing, repairs, or fuel delivery.
2. **Service Providers (Towing services, mechanics, etc.)**: Professionals who offer roadside assistance services and use the platform to manage requests, update status, and receive payments.
3. **Administrators**: System operators responsible for overseeing platform functionality, managing users and service providers, and generating performance analytics.

The platform is built using the Django framework for the back-end and HTML/CSS for the front-end, ensuring scalability, security, and ease of use. Its architecture supports key features like user registration, service request management, real-time tracking, and an administrative dashboard to monitor and report on system performance.

1.2 PROJECT SPECIFICATION

The Road Guardian system is a web-based platform designed to provide real-time support to vehicle owners during emergencies such as vehicle breakdowns or accidents. Users can request assistance through a mobile-friendly interface, track the location of service providers in real-time, and make secure digital payments. The system enhances communication between vehicle owners and service providers, reducing response times and improving service quality.

Platform & Technology Stack:

- Back-end: Django (Python)
- Front-end: HTML, CSS, JavaScript
- Database: MySQL (for user, service provider, and service request data)
- Payment Gateway: Integrated digital payment system for secure transactions

Modules & Features:

- User Registration and Authentication: Allows users and service providers to create accounts and securely log in.
- Service Request Management: Users can submit and track requests for roadside assistance.
- Service Provider Management: Service providers can manage their availability and respond to requests.
- Real-time Tracking: GPS-based tracking of service provider locations.
- Admin Dashboard: Administrators can manage users, service providers, and generate system performance reports.

Target Audience:

- Vehicle owners in need of emergency assistance.
- Service providers offering roadside help (towing, mechanical repairs, etc.).
- System administrators overseeing the platform’s operations.

CHAPTER 2
SYSTEM STUDY

2.1 INTRODUCTION

System study is a critical phase in the development of any project, involving a comprehensive analysis of the existing processes, identifying inefficiencies, and proposing solutions through a new system. In the case of the Road Guardian project, the system study focuses on understanding the current methods of providing roadside assistance, evaluating their shortcomings, and designing a modern, technology-driven solution to improve the overall user experience.

The existing roadside assistance process is largely manual, relying on phone calls to request help, which often leads to delays, miscommunication, and customer frustration. There is no real-time tracking of service providers, limited integration with GPS technology, and no digital payment options, making the process cumbersome and inefficient.

The system study aims to analyze these gaps in the current system and propose a web-based platform that automates and streamlines the roadside assistance process. The new system leverages modern technologies like GPS tracking, real-time updates, and digital payments, ensuring quicker response times and enhanced service quality. Through this study, the proposed system is designed to meet user needs more effectively, reduce operational inefficiencies, and provide a scalable solution for future growth.

2.2 EXISTING SYSTEM

The current roadside assistance system relies on manual methods such as phone calls to connect users with service providers. This process often results in long wait times and inaccuracies, as there is no real-time tracking of service vehicles, leading to customer frustration and inefficiencies.

2.2.1 NATURAL SYSTEM STUDIED

In the context of the Road Guardian project, the natural system refers to the traditional, manual process that currently exists for providing roadside assistance. This system primarily relies on human interaction and manual operations, which involve the following steps:

Manual Service Requests: Vehicle owners in need of roadside assistance typically make phone calls to service providers or roadside assistance companies. The process of requesting help is entirely manual, often leading to delays and miscommunication.

Information Gathering: During the call, the service provider manually gathers information such as the location of the vehicle, the nature of the issue (e.g., flat tire, engine trouble), and the contact details of the vehicle owner. This information is often collected verbally, increasing the potential for errors.

Dispatching Assistance: After the service request is received, the service provider dispatches a vehicle or technician to the breakdown location. The process of dispatching is largely manual and often based on the availability of service providers, without leveraging technology like GPS for optimization.

No Real-Time Tracking: Vehicle owners are usually left waiting without knowing the exact status or location of the dispatched service provider. This lack of transparency leads to uncertainty and frustration.

Payment Process: Payment for the services is typically handled in person and is often cash-based. There is no integration of digital payment systems, which can lead to inefficiencies and inconveniences for both users and service providers.

2.2.2 DESIGNED SYSTEM STUDIED

The designed system for the Road Guardian project is a comprehensive, technology-driven solution created to address the inefficiencies of the traditional, manual roadside assistance system. This designed system integrates modern technologies such as GPS, real-time tracking, digital payments, and automation to streamline the process and provide a more efficient and user-friendly service.

Key Components of the Designed System:

Automated Service Request: Vehicle owners can request roadside assistance through a web-based platform or mobile app. This eliminates the need for phone calls, reducing the risk of miscommunication. Users can specify the type of issue (e.g., flat tire, engine trouble) through an intuitive interface.

GPS-Based Location Tracking: The system automatically captures the user's location using GPS technology. This enables accurate and efficient dispatching of the nearest available service provider, reducing response times significantly. Users can also view the real-time location of the service provider on a map.

Real-Time Updates: Once the service request is initiated, users receive real-time updates on the status of their request, including the estimated time of arrival and the live location of the service provider. This transparency enhances the overall user experience and reduces uncertainty.

Service Provider Management: Service providers can manage their availability, accept requests, and update their service status via the platform. This helps optimize their schedules and increase their efficiency in handling multiple service requests.

Digital Payments: The system integrates secure digital payment options, allowing users to pay for services online without the need for cash transactions. This enhances convenience and ensures smoother financial operations.

Admin Dashboard: Administrators can manage the entire system from a centralized dashboard. They can monitor user activity, manage service providers, track system performance, and generate reports on various metrics such as service response times and customer satisfaction.

Feedback and Ratings: After each service is completed, users can rate the service provider and leave feedback. This system helps maintain service quality and holds providers accountable for their performance.

2.3 DRAWBACKS OF EXISTING SYSTEM

- **Manual Process:** The lack of automation slows down the assistance process.

- **Limited Technology Integration:** Modern technologies such as GPS or digital payments are absent.

- **No Real-Time Tracking:** Users cannot track service providers, leading to uncertainty

- **Customer Frustration:** Delays in service delivery lead to dissatisfaction.

2.4 PROPOSED SYSTEM

The Road Guardian system aims to revolutionize the traditional roadside assistance process by integrating modern technologies to improve response times, service quality, and user convenience. The new system is designed to eliminate the inefficiencies present in the current manual process and provide a seamless, automated solution for vehicle owners in need of emergency roadside help.

Key Objectives:

- Automated Service Requests: Vehicle owners can easily request assistance through a web-based platform or mobile app. Users can describe their issue (e.g., flat tire, engine problem) and submit their request in just a few clicks.

- GPS Location Tracking: The system automatically captures the user's location using GPS. This enables real-time tracking of the assistance vehicle and ensures accurate dispatching of the nearest available service provider, reducing response times significantly.

- Real-Time Updates: After a service request is made, users receive real-time updates about the status of their request, including the estimated time of arrival of the service provider. This transparency increases trust and reduces anxiety for the user.

- Service Provider Management: Service providers can register on the platform, manage their availability, and respond to requests. They can also view and track their service history, ratings, and feedback to maintain high service standards.

- Digital Payments: The system integrates secure digital payment options, allowing users to pay for services conveniently without cash. This simplifies the transaction process and enhances user convenience.

- Admin Dashboard: System administrators can monitor the platform's performance, manage users and service providers, and generate reports on service request volumes, response times, and overall system efficiency.

- Feedback and Ratings: Users can provide feedback and rate service providers after the completion of a service, allowing the system to maintain high-quality standards and ensure accountability.

2.5 ADVANTAGES OF PROPOSED SYSTEM

- **Increased Efficiency**: Automated processes and GPS-based dispatching lead to quicker response times and a more efficient service model.
- **Improved Accuracy**: GPS integration ensures precise location tracking, minimizing the chances of errors in dispatching assistance.
- **Enhanced User Experience**: A user-friendly interface combined with real-time updates and secure payments ensures a smooth experience for users.

- **Scalability**: The system is designed to handle a growing user base, making it scalable for future expansions and new features.
- **Service Quality Control**: Feedback and rating systems ensure that service providers maintain high standards and are accountable for their services.

CHAPTER 3
REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDY

The feasibility study evaluates the technical, economic, and behavioral viability of the proposed Road Guardian system.

Economic Feasibility: The system is financially viable with initial investments in technology, development, and maintenance offset by increased efficiency, customer satisfaction, and potential revenue from service charges. The system's ability to reduce response times and improve service quality ensures a positive return on investment (ROI).

Technical Feasibility: The necessary technologies—Django for web development, GPS for location tracking, real-time updates, and digital payments—are readily available. The system is scalable and can handle growing user demand, ensuring long-term reliability.

Behavioral Feasibility: Indicate high user acceptance, as vehicle owners prefer faster response times and real-time tracking. Service providers are also inclined to adopt the system due to its ease of use and efficiency improvements. Proper training and support will further boost acceptance.

3.1.1 Economical Feasibility

Economic feasibility examines the costs associated with developing, implementing, and maintaining the system, while assessing its potential to generate revenue and return on investment (ROI).

- Initial Costs: Development costs include software development (Django back-end, HTML/CSS front-end), hardware (servers, GPS integration), database management (MySQL), and digital payment integration. Maintenance and support will also incur ongoing costs.

- Revenue Generation: The platform can generate revenue through service fees charged to users for roadside assistance and potential subscription models for frequent users. Partnerships with service providers can also create revenue streams. Additionally, service providers may pay to be listed on the platform.

- Return on Investment (ROI): By improving service efficiency and reducing response times, the system is expected to attract more users, resulting in higher service fees. The enhanced user experience will likely result in repeat customers and positive word-of-mouth, ensuring long-term financial viability. The predicted ROI is positive due to the system’s ability to optimize service delivery and increase user satisfaction.

3.1.2 Technical Feasibility

Technical feasibility assesses whether the proposed system can be built using the available technology and expertise.

- Technology Availability: The system will be developed using Django, a scalable and secure framework, with HTML/CSS for the front-end, and MySQL for the database. These are widely used, well-supported technologies that can easily handle the requirements of the project.

- GPS Integration: Real-time GPS tracking, a core feature of the system, is achievable through existing APIs such as Google Maps. This ensures that service providers can be accurately tracked and dispatched, reducing response times.

- Real-Time Communication: Technologies to implement real-time updates, such as WebSockets or third-party messaging APIs, are readily available and can be integrated seamlessly into the system.

- Scalability: The system is designed to scale as more users and service providers join the platform, ensuring smooth operation even with increasing demand.

- Security: Secure payment gateways and encrypted data transmission ensure that users' financial and personal information is protected.

3.1.3 Behavioral Feasibility

Behavioral feasibility assesses whether the target users—vehicle owners, service providers, and administrators—are likely to accept and adopt the system.

- User Acceptance: Vehicle owners are expected to readily adopt the system, as it addresses major pain points, including delays, lack of tracking, and manual payment processes. The convenience of real-time updates, digital payments, and automated service requests aligns with modern user expectations.

- Service Provider Readiness: Service providers are likely to embrace the system as it simplifies the management of service requests, helps them track service history, and provides a platform for receiving payments. The system also offers them visibility and potential business growth through positive ratings and feedback.

- Training and Support: Minimal training will be required for users and service providers, as the system is designed with a user-friendly interface. However, support resources will be available to ensure smooth adoption and to address any issues that arise.

- Cultural and Behavioral Factors: With the increasing reliance on technology for convenience and service optimization, users are likely to adapt well to the digital platform. This shift aligns with trends in other service industries, such as ride-sharing or food delivery, where similar platforms are widely accepted.

3.1.4 Feasibility Study Questionnaire

1.What are the most common types of roadside assistance services required by vehicle owners?

-Towing services, battery jump-starts, fuel delivery, flat tire changes, and lockout services.

2.How quickly do vehicle owners expect assistance after requesting help?

-Typically within 30 to 60 minutes, depending on the urgency of the situation and proximity of service providers.

3.What information should be collected from users when they request roadside assistance?

-User’s name, contact number, vehicle type, location (GPS coordinates or address), nature of the problem, and any special instructions.

4.How do vehicle owners currently request roadside assistance, and what are the main challenges they face?

-Vehicle owners currently request assistance via phone calls or through existing apps. Challenges include long wait times, difficulty in tracking the service provider, and inconsistent service quality.

5.What features would make a roadside assistance system more user-friendly?

-Real-time tracking of service providers, easy-to-use interface, quick and simple request process, clear communication channels, and updates on estimated arrival times.

6.What are the key factors that influence the choice of a service provider for roadside assistance?

-Service provider's response time, reputation, availability, cost of services, and proximity to the vehicle owner's location.

7.How important is real-time tracking of service requests to users?

-Highly important, as it provides transparency, reduces anxiety, and allows users to plan accordingly while waiting for assistance.

8.What are the expectations for service provider response times?

-Users expect service providers to acknowledge the request within 10-15 minutes and to arrive on-site within 30 to 60 minutes.

9.How can the system ensure the accuracy and reliability of service provider information?

-By implementing a verification process during registration, gathering user reviews, and regularly updating service provider details based on performance and feedback.

10.What type of reporting or analytics would be useful for administrators to monitor system performance?

-Metrics on request response times, user satisfaction ratings, service provider performance, system uptime, and overall request volume and resolution rates.

3.2 SYSTEM SPECIFICATION

3.2.1 Hardware Specification

Processor: Intel i5 or higher (for server setup) / AMD equivalent.
RAM: Minimum 8 GB (for development and server-side processing).
Hard Disk: Minimum 500 GB HDD or 256 GB SSD (for server data storage and backups).

3.2.2 Software Specification

- **Front End:** HTML5, CSS3, JavaScript, AJAX, J Query.
- **Back End:** Django (Python Framework).

- **Database:** MySQL.
- **Client on PC:** Windows 7 and above (for client-side interactions and testing).
- **Technologies Used:**
    - JavaScript (for interactive elements and client-side validation).
    - HTML5 (for structuring the web pages).
    - CSS3 (for styling and layouts).
    - AJAX (for asynchronous page updates without reloading).
    - J Query (for JavaScript simplification).
    - PHP (for additional dynamic functionalities in certain modules).

3.3 SOFTWARE DESCRIPTION

3.3.1 Django (Python Framework)

Django is a high-level Python web framework used for building the back-end of the Road Guardian system. It provides built-in features for database management, user authentication, and secure handling of data. Django's scalability ensures that the platform can handle growing numbers of users and requests without performance issues.

Key Features:

- Provides robust ORM (Object-Relational Mapping) for database operations.
- Built-in user authentication system.
- Secure handling of data and integration with third-party APIs (such as GPS tracking services).

3.3.2 MySQL (Database)

MySQL is the relational database management system used to store and manage data for the Road Guardian project. It stores information such as user profiles, service provider details, service requests, transaction history, and feedback. MySQL is widely used for web applications due to its performance and scalability.

Key Features:

- Supports large-scale data storage and efficient querying.
- Provides indexing for faster data retrieval.
- Allows integration with Django using its built-in ORM.

3.3.3 JavaScript and jQuery (Front-End Interaction)

JavaScript is used for making the web pages interactive by handling client-side operations such as form validation, event handling, and dynamic content updates. jQuery simplifies JavaScript operations by offering easy-to-use functions for handling HTML elements, animations, and asynchronous updates (AJAX).

Key Features:

- Enhances user experience with interactive and dynamic web pages.
- Supports asynchronous operations, reducing page reload times.
- Cross-browser compatibility for consistent performance across different browsers.

3.3.4 HTML5 & CSS3 (Web Page Structuring and Styling)

HTML5 is used for structuring web pages, ensuring proper layout and readability, while CSS3 is used for styling and positioning elements on the web pages. Together, they form the foundation of the front-end design.

Key Features:

- Responsive design to ensure optimal viewing on various devices (desktops, tablets, mobiles).
- CSS3 transitions and animations for better user experience.
- HTML5 semantic tags improve accessibility and SEO.

CHAPTER 4

SYSTEM DESIGN

4.1 INTRODUCTION

System design is a critical phase in the development of any software project, where the theoretical concepts and requirements are translated into a practical and workable structure. The objective of the system design phase is to create a blueprint that outlines how the system's various components and functionalities will be implemented to meet the project's requirements efficiently.

For the Road Guardian project, the system design involves defining the architecture, data flow, user interactions, and the relationships between various system modules. The goal is to ensure that the system is scalable, secure, and user-friendly while meeting the specific needs of vehicle owners, service providers, and administrators.

The system design is divided into two key phases:

1. High-Level Design (HLD):

This phase focuses on the overall architecture of the system, defining the major components and how they interact with one another. In the case of Road Guardian, the high-level design includes:

- User Interface Design: Designing the layout and structure of web pages for users, service providers, and administrators.
- System Architecture: Defining the relationship between the front-end, back-end, and database, including how data flows between them
- Module Design: Outlining the key system modules, such as user registration, service request management, real-time tracking, and payment processing.

2. Low-Level Design (LLD):

This phase delves into the finer details of how each module will function. It includes:

- Detailed Component Design: Specifying the exact functionalities of each system module (e.g., login system, service provider dashboard, admin panel).
- Database Schema Design: Designing the structure of the database to efficiently store and manage data such as user profiles, service requests, and transaction history.
- Algorithm Design: Defining algorithms for tasks such as matching users with the nearest available service providers and calculating estimated arrival times.

4.2 UML DIAGRAM

Unified Modeling Language (UML) diagrams are essential tools in system design that visually represent the structure, components, and behavior of a software system. For the Road Guardian project, several UML diagrams can be used to model the system effectively. Below are the key UML diagrams commonly used in the project:

4.2.1 USE CASE DIAGRAM

A use case diagram provides an overview of the interactions between users (actors) and the system. It highlights the functionalities that each user role can perform.

Actors:

- Vehicle Owner (User)
- Service Provider
- Administrator

Use Cases:

- Register/Login
- Request Roadside Assistance
- Track Service Provider (Real-Time)
- Update Service Status (Service Provider)
- Manage Users and Providers (Admin)
- View Service History
- Provide Feedback

User

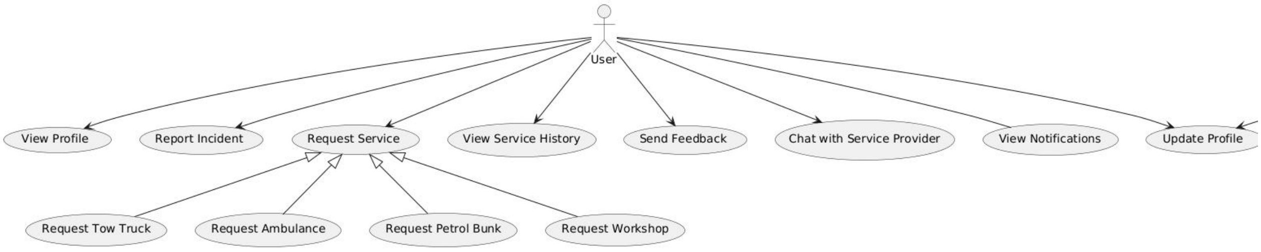


Fig:4.2.1.1

Service Provider

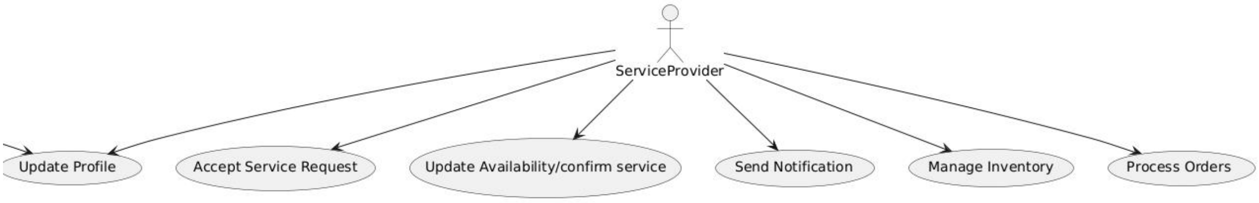


Fig:4.2.1.2

Admin

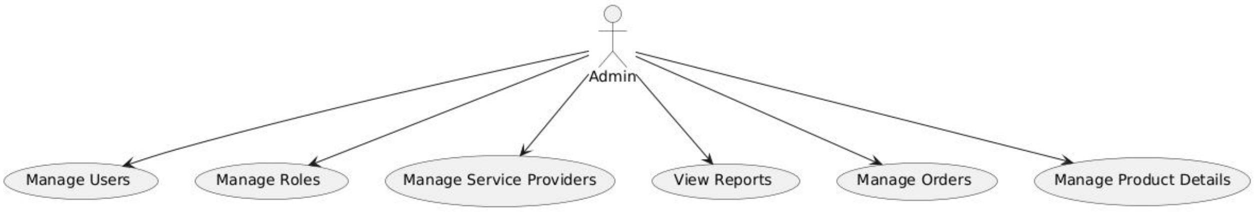


Fig:4.2.1.3

4.2.2 SEQUENCE DIAGRAM

A sequence diagram illustrates how objects in the system interact over time to carry out a specific function, such as requesting roadside assistance.

Scenario: Requesting Roadside Assistance

Steps:

- The vehicle owner logs in.
- The vehicle owner submits a service request, specifying the issue and location.
- The system assigns the nearest available service provider.
- The service provider accepts and begins the service.
- Real-time updates are sent to the vehicle owner.
- The service is completed, and the user provides feedback.

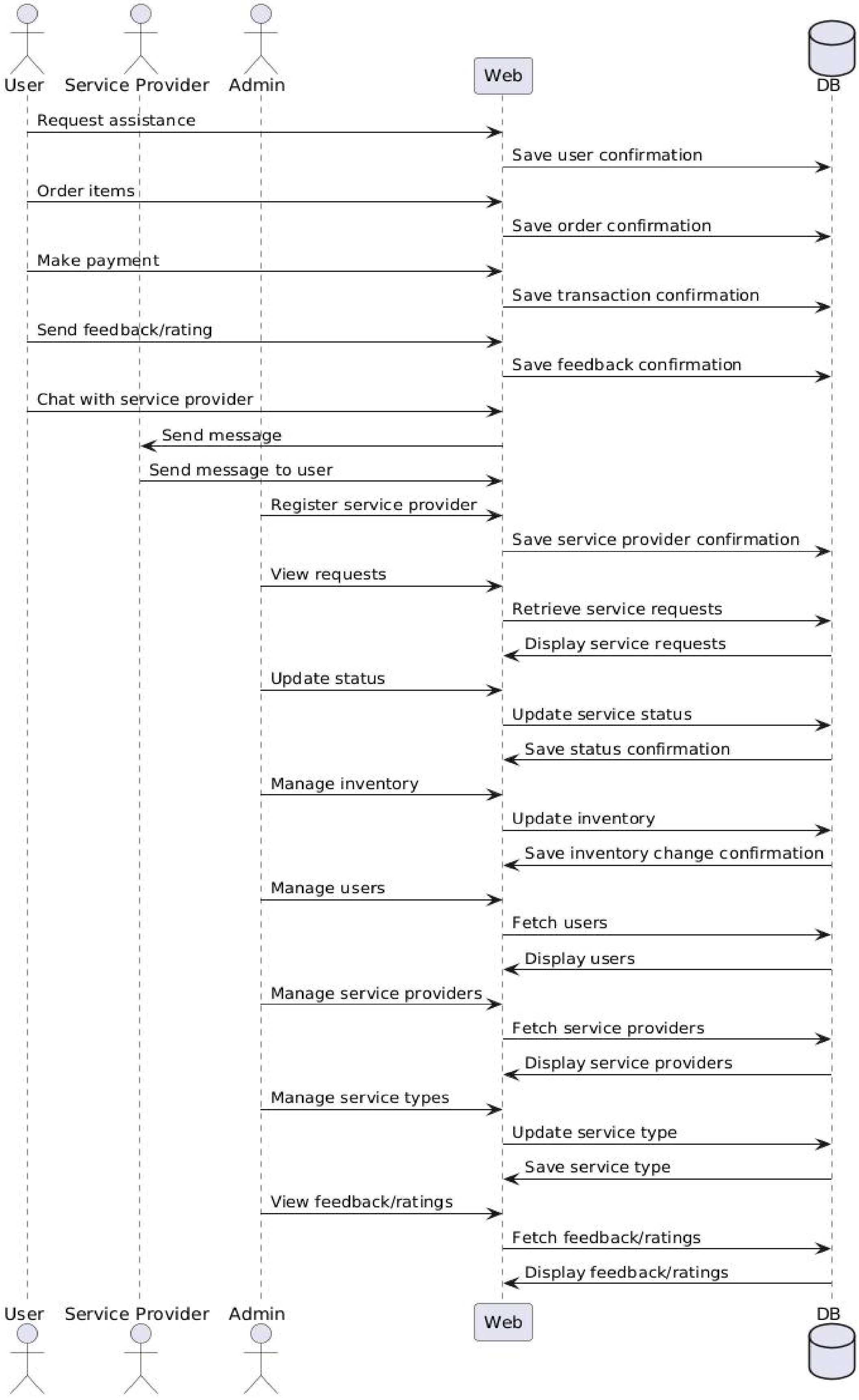

User
Service Provider
Admin
Web
DB
Request assistance
Save user confirmation
Order items
Save order confirmation
Make payment
Save transaction confirmation
Send feedback/rating
Save feedback confirmation
Chat with service provider
Send message
Send message to user
Register service provider
Save service provider confirmation
View requests
Retrieve service requests
Display service requests
Update status
Update service status
Save status confirmation
Manage inventory
Update inventory
Save inventory change confirmation
Manage users
Fetch users
Display users
Manage service providers
Fetch service providers
Display service providers
Manage service types
Update service type
Save service type
View feedback/ratings
Fetch feedback/ratings
Display feedback/ratings
User
Service Provider
Admin
Web
DB

Fig:4.2.2

4.2.3 State Chart Diagram

States:

- Idle: Initial state when no service request has been made.
- Requested: The user submits a service request.
- Assigned: A service provider has been assigned to the request.
- In Progress: The service provider is en route to assist the user.
- Completed: The service has been completed successfully.
- Canceled: The user cancels the request before it is completed.
- Failed: The request could not be completed due to various issues (e.g., service provider unavailable).

Transitions:

- Submit Request: From Idle to Requested when the user submits a service request.
- Assign Provider: From Requested to Assigned when the system assigns a service provider.
- Start Service: From Assigned to In Progress when the service provider accepts the request.
- Complete Service: From In Progress to Completed when the service is finished.
- Cancel Request: From Requested or In Progress to Canceled when the user cancels the request.
- Fail Request: From Assigned or In Progress to Failed if the service provider cannot fulfill the request.

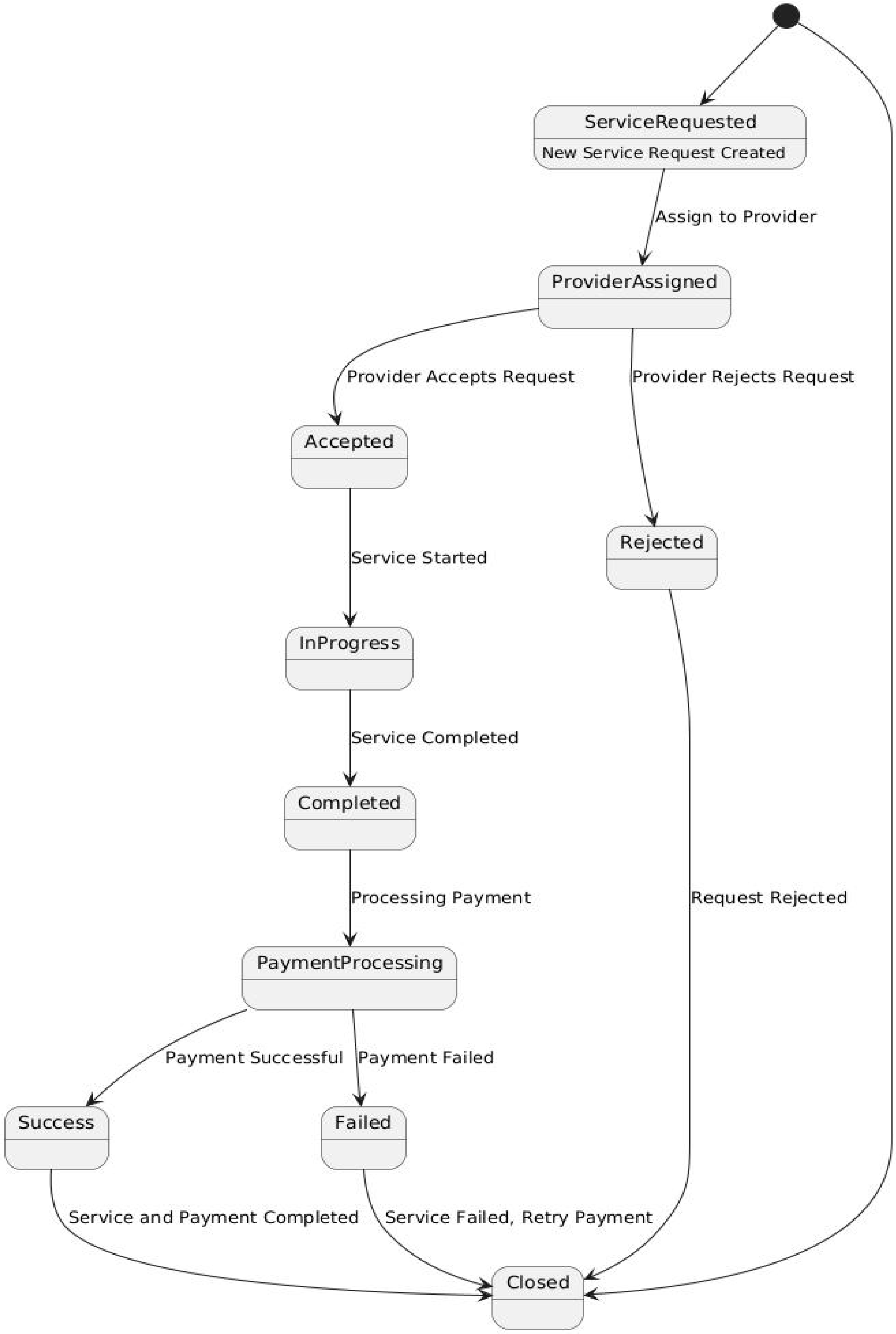

ServiceRequested
New Service Request Created
Assign to Provider
ProviderAssigned
Provider Accepts Request
Provider Rejects Request
Accepted
Rejected
Service Started
InProgress
Service Completed
Completed
Processing Payment
Request Rejected
PaymentProcessing
Payment Successful
Payment Failed
Success
Failed
Service and Payment Completed
Service Failed, Retry Payment
Closed

Fig:4.2.3.1

4.2.4 Activity Diagram

An activity diagram represents the workflow of activities within the system. It is useful for showing the step-by-step flow of a particular process.

Activity: Handling a Service Request

Steps:

- User submits a service request.
- The system verifies location and assigns a service provider.
- Service provider receives and accepts the request.
- The service provider moves to the location.
- The user tracks the service provider in real-time.
- Service is completed, and the request is marked as resolved.

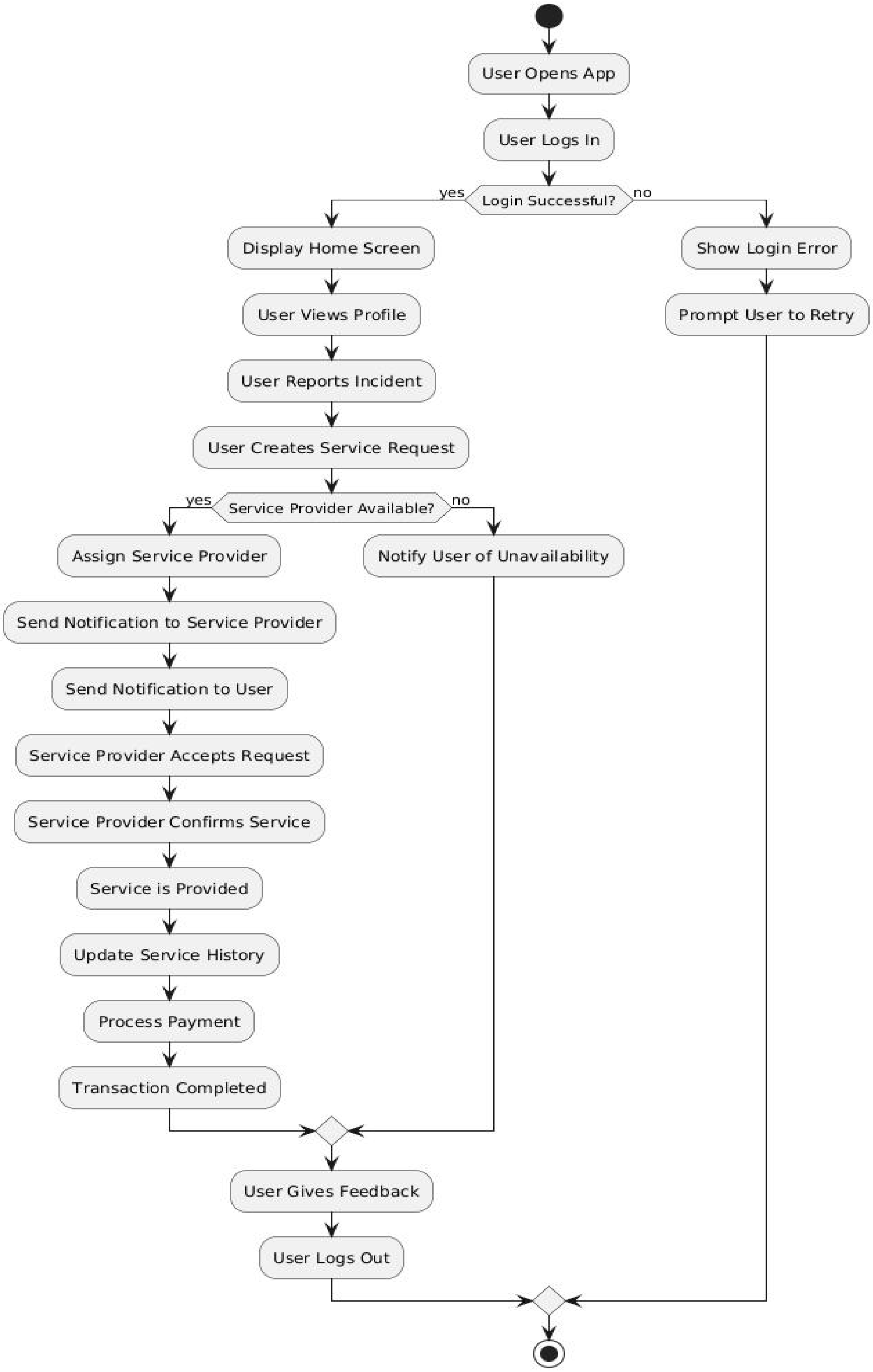

User Opens App
User Logs In
Login Successful?
yes
no
Display Home Screen
Show Login Error
User Views Profile
Prompt User to Retry
User Reports Incident
User Creates Service Request
Service Provider Available?
yes
no
Assign Service Provider
Notify User of Unavailability
Send Notification to Service Provider
Send Notification to User
Service Provider Accepts Request
Service Provider Confirms Service
Service is Provided
Update Service History
Process Payment
Transaction Completed
User Gives Feedback
User Logs Out

Fig:4.2.4.1

4.2.5 Class Diagram

The class diagram models the structure of the system by showing the system's classes, their attributes, methods, and relationships between them.

- **Classes**:
  - User
    - Attributes: username, password, email, phone, role
    - Methods: register(), login(), updateProfile()
  - ServiceRequest
    - Attributes: request_id, user_id, location, status, service_type
    - Methods: createRequest(), trackRequest()
  - ServiceProvider
    - Attributes: provider_id, name, service_type, availability, location
    - Methods: updateStatus(), respondToRequest()
  - Admin
    - Attributes: admin_id, username
    - Methods: manageUsers(), generateReports()

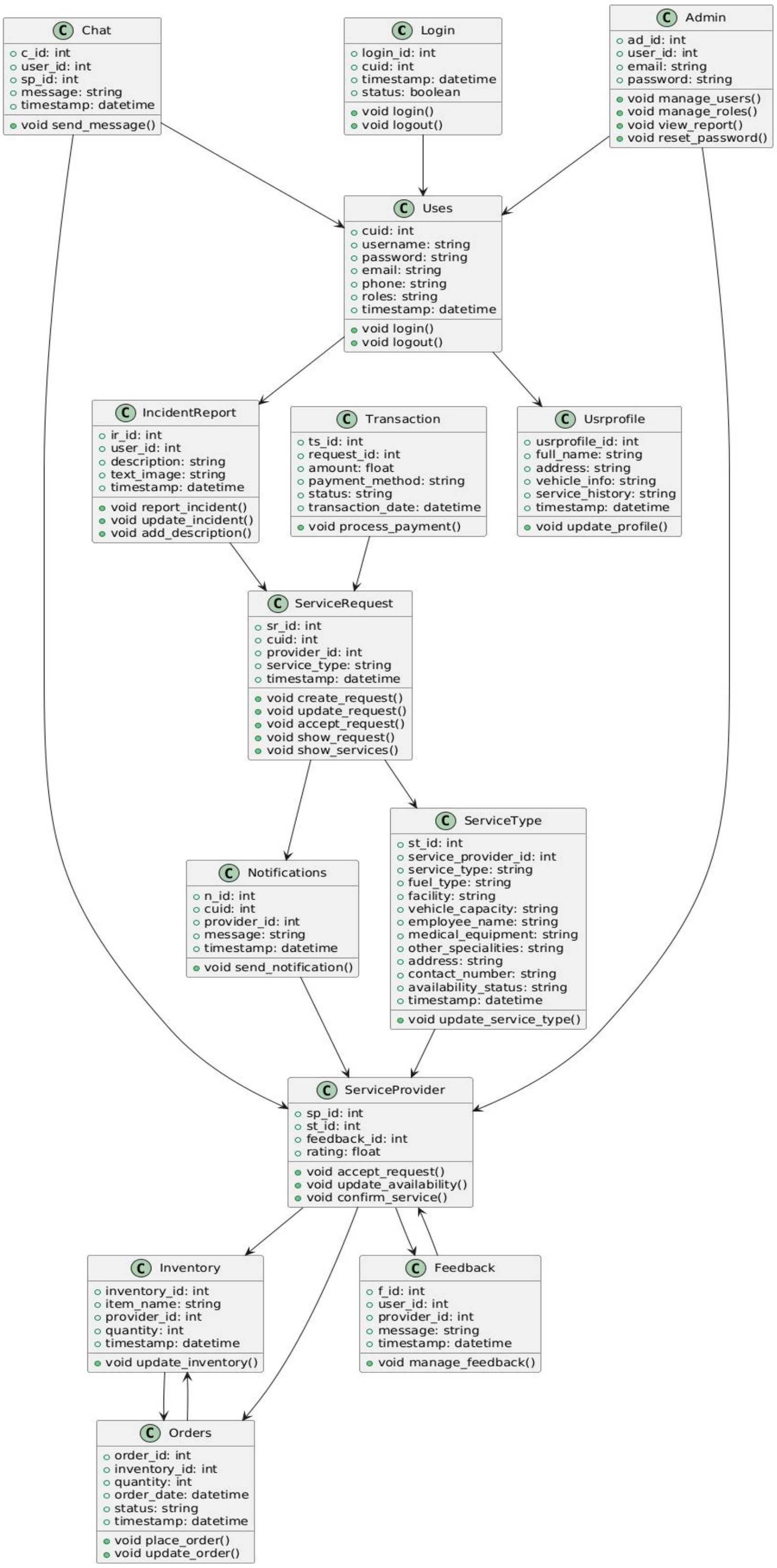

Chat
c_id: int
user_id: int
sp_id: int
message: string
timestamp: datetime
void send_message()
Login
login_id: int
cuid: int
timestamp: datetime
status: boolean
void login()
void logout()
Admin
ad_id: int
user_id: int
email: string
password: string
void manage_users()
void manage_roles()
void view_report()
void reset_password()
Uses
cuid: int
username: string
password: string
email: string
phone: string
roles: string
timestamp: datetime
void login()
void logout()
IncidentReport
ir_id: int
user_id: int
description: string
text_image: string
timestamp: datetime
void report_incident()
void update_incident()
void add_description()
Transaction
ts_id: int
request_id: int
amount: float
payment_method: string
status: string
transaction_date: datetime
void process_payment()
Usrprofile
usrprofile_id: int
full_name: string
address: string
vehicle_info: string
service_history: string
timestamp: datetime
void update_profile()
ServiceRequest
sr_id: int
cuid: int
provider_id: int
service_type: string
timestamp: datetime
void create_request()
void update_request()
void accept_request()
void show_request()
void show_services()
ServiceType
st_id: int
service_provider_id: int
service_type: string
fuel_type: string
facility: string
vehicle_capacity: string
employee_name: string
medical_equipment: string
other_specialities: string
address: string
contact_number: string
availability_status: string
timestamp: datetime
void update_service_type()
Notifications
n_id: int
cuid: int
provider_id: int
message: string
timestamp: datetime
void send_notification()
ServiceProvider
sp_id: int
st_id: int
feedback_id: int
rating: float
void accept_request()
void update_availability()
void confirm_service()
Inventory
inventory_id: int
item_name: string
provider_id: int
quantity: int
timestamp: datetime
void update_inventory()
Feedback
f_id: int
user_id: int
provider_id: int
message: string
timestamp: datetime
void manage_feedback()
Orders
order_id: int
inventory_id: int
quantity: int
order_date: datetime
status: string
timestamp: datetime
void place_order()
void update_order()

Fig:4.2.5.1

4.2.6 Object Diagram

An object diagram is a type of UML diagram that shows a snapshot of the instances of classes (objects) at a particular moment in time, along with their relationships. In the context of your Road Guardian project, an object diagram can illustrate the relationships between different objects in the system, such as users, service requests, service providers, and administrators.

Key Objects and Their Relationships

- User (Vehicle Owner)
  - Attributes:

    username: "john_doe"

    phone: "1234567890"

    role: "Vehicle Owner"

- ServiceRequest
  - Attributes:

    request_id: "R123"

    user_id: "U001"

    location: "123 Main St"

    status: "Requested"

    service_type: "Towing"

- ServiceProvider
  - Attributes:

    provider_id: "P001"

    name: "Fast Tow Services"

    service_type: "Towing"

    availability: "Available"
- Admin
  - Attributes:

    admin_id: "A001"

    username: "admin_user"

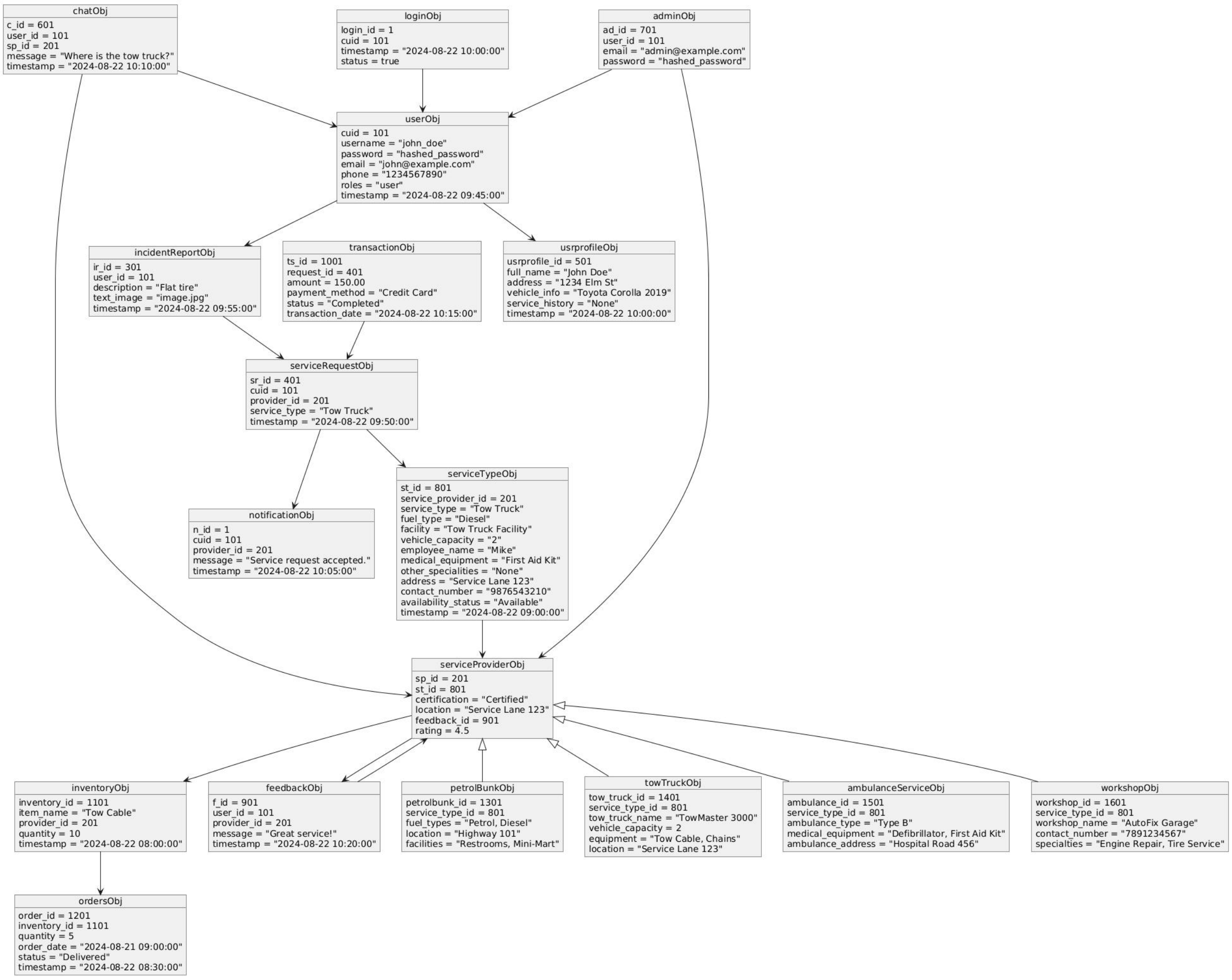

chatObj
c_id = 601
user_id = 101
sp_id = 201
message = "Where is the tow truck?"
timestamp = "2024-08-22 10:10:00"
loginObj
login_id = 1
cuid = 101
timestamp = "2024-08-22 10:00:00"
status = true
adminObj
ad_id = 701
user_id = 101
email = "admin@example.com"
password = "hashed_password"
userObj
cuid = 101
username = "john_doe"
password = "hashed_password"
email = "john@example.com"
phone = "1234567890"
roles = "user"
timestamp = "2024-08-22 09:45:00"
incidentReportObj
ir_id = 301
user_id = 101
description = "Flat tire"
text_image = "image.jpg"
timestamp = "2024-08-22 09:55:00"
transactionObj
ts_id = 1001
request_id = 401
amount = 150.00
payment_method = "Credit Card"
status = "Completed"
transaction_date = "2024-08-22 10:15:00"
usrprofileObj
usrprofile_id = 501
full_name = "John Doe"
address = "1234 Elm St"
vehicle_info = "Toyota Corolla 2019"
service_history = "None"
timestamp = "2024-08-22 10:00:00"
serviceRequestObj
sr_id = 401
cuid = 101
provider_id = 201
service_type = "Tow Truck"
timestamp = "2024-08-22 09:50:00"
notificationObj
n_id = 1
cuid = 101
provider_id = 201
message = "Service request accepted."
timestamp = "2024-08-22 10:05:00"
serviceTypeObj
st_id = 801
service_provider_id = 201
service_type = "Tow Truck"
fuel_type = "Diesel"
facility = "Tow Truck Facility"
vehicle_capacity = "2"
employee_name = "Mike"
medical_equipment = "First Aid Kit"
other_specialities = "None"
address = "Service Lane 123"
contact_number = "9876543210"
availability_status = "Available"
timestamp = "2024-08-22 09:00:00"
serviceProviderObj
sp_id = 201
st_id = 801
certification = "Certified"
location = "Service Lane 123"
feedback_id = 901
rating = 4.5
inventoryObj
inventory_id = 1101
item_name = "Tow Cable"
provider_id = 201
quantity = 10
timestamp = "2024-08-22 08:00:00"
feedbackObj
f_id = 901
user_id = 101
provider_id = 201
message = "Great service!"
timestamp = "2024-08-22 10:20:00"
petrolBunkObj
petrolbunk_id = 1301
service_type_id = 801
fuel_types = "Petrol, Diesel"
location = "Highway 101"
facilities = "Restrooms, Mini-Mart"
towTruckObj
tow_truck_id = 1401
service_type_id = 801
tow_truck_name = "TowMaster 3000"
vehicle_capacity = 2
equipment = "Tow Cable, Chains"
location = "Service Lane 123"
ambulanceServiceObj
ambulance_id = 1501
service_type_id = 801
ambulance_type = "Type B"
medical_equipment = "Defibrillator, First Aid Kit"
ambulance_address = "Hospital Road 456"
workshopObj
workshop_id = 1601
service_type_id = 801
workshop_name = "AutoFix Garage"
contact_number = "7891234567"
specialties = "Engine Repair, Tire Service"
ordersObj
order_id = 1201
inventory_id = 1101
quantity = 5
order_date = "2024-08-21 09:00:00"
status = "Delivered"
timestamp = "2024-08-22 08:30:00"

Fig:4.2.6.1

4.2.7 Component Diagram

A component diagram provides a high-level view of the components in a system and how they interact with each other. For your Road Guardian project, we can create a component diagram that illustrates the key components and their relationships.

Component Diagram for Road Guardian

Key Components

User Interface

- Represents the front-end interface for users to interact with the system.

User Management

- Handles user registration, login, and profile management.

Service Request Management

- Manages the creation, assignment, and tracking of service requests.

Service Provider Management

- Handles service provider registration, updates, and availability.

Administrator Panel

- Allows administrators to manage users and service providers.

Notification System

- Sends notifications and real-time updates to users and service providers.

Database

- Stores all user, service request, and service provider information.

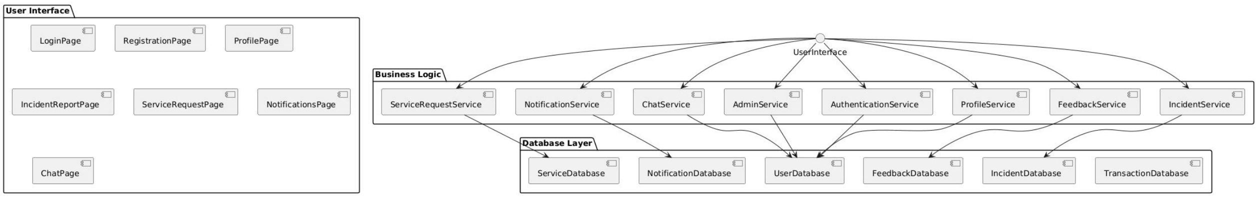

User Interface
LoginPage
RegistrationPage
ProfilePage
IncidentReportPage
ServiceRequestPage
NotificationsPage
ChatPage
UserInterface
Business Logic
ServiceRequestService
NotificationService
ChatService
AdminService
AuthenticationService
ProfileService
FeedbackService
IncidentService
Database Layer
ServiceDatabase
NotificationDatabase
UserDatabase
FeedbackDatabase
IncidentDatabase
TransactionDatabase

Fig:4.2.7.1

4.2.8 Deployment Diagram

A deployment diagram illustrates the physical architecture of a system, showing how software components are deployed on hardware. For the 247 Roadside Assistance project, a deployment diagram will highlight the servers, devices, and connections used to run and interact with the application.

Key Nodes:

Client Devices

- Devices used by users (Vehicle Owners, Service Providers, Administrators) to interact with the system.
- Typically includes laptops, smartphones, and tablets.
- Uses a web browser or mobile app to access the system.

Web Server

- Hosts the Django-based web application.
- Handles HTTP requests from client devices.
- Runs the application logic for user management, service requests, etc.

Database Server

- Stores data related to users, service requests, service providers, feedback, etc.
- Uses MySQL for data storage.

Notification Server (Optional)

- Handles real-time notifications and updates.
- Uses WebSocket or similar technology to provide real-time tracking updates.

Admin Workstation

- A specialized device or computer for system administrators.
- Accesses the Admin Panel to manage users and service providers.

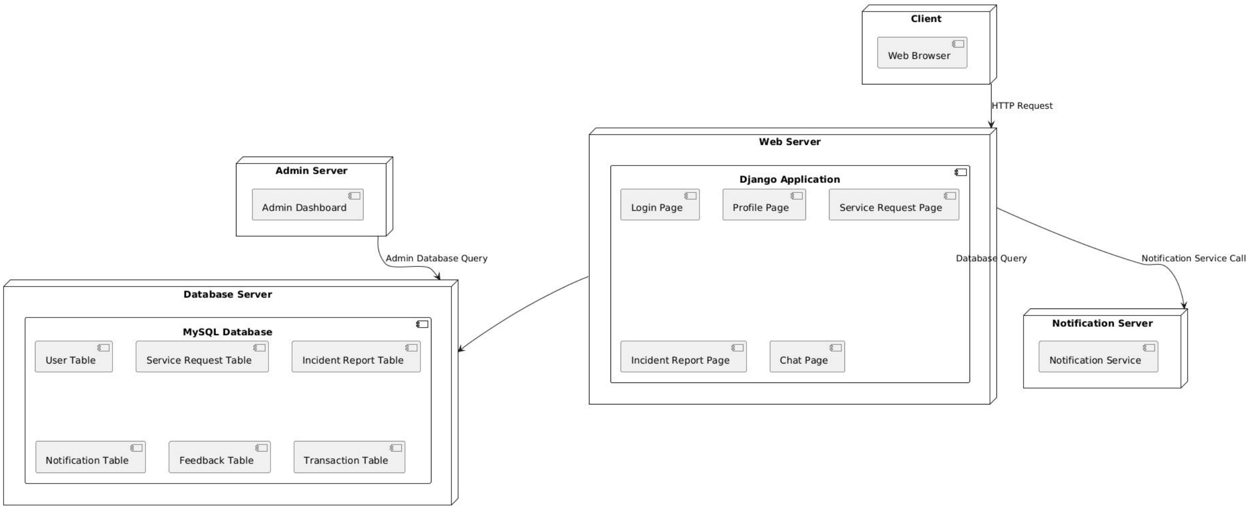


Fig:4.2.8.1

4.2.9 Collaboration Diagram

A collaboration diagram (also known as an interaction or communication diagram) is a type of UML (Unified Modeling Language) diagram that visualizes the interactions between objects or components in a system to achieve a specific outcome. Unlike sequence diagrams, which emphasize the time order of messages, collaboration diagrams focus on the relationships and roles of objects during the interaction.

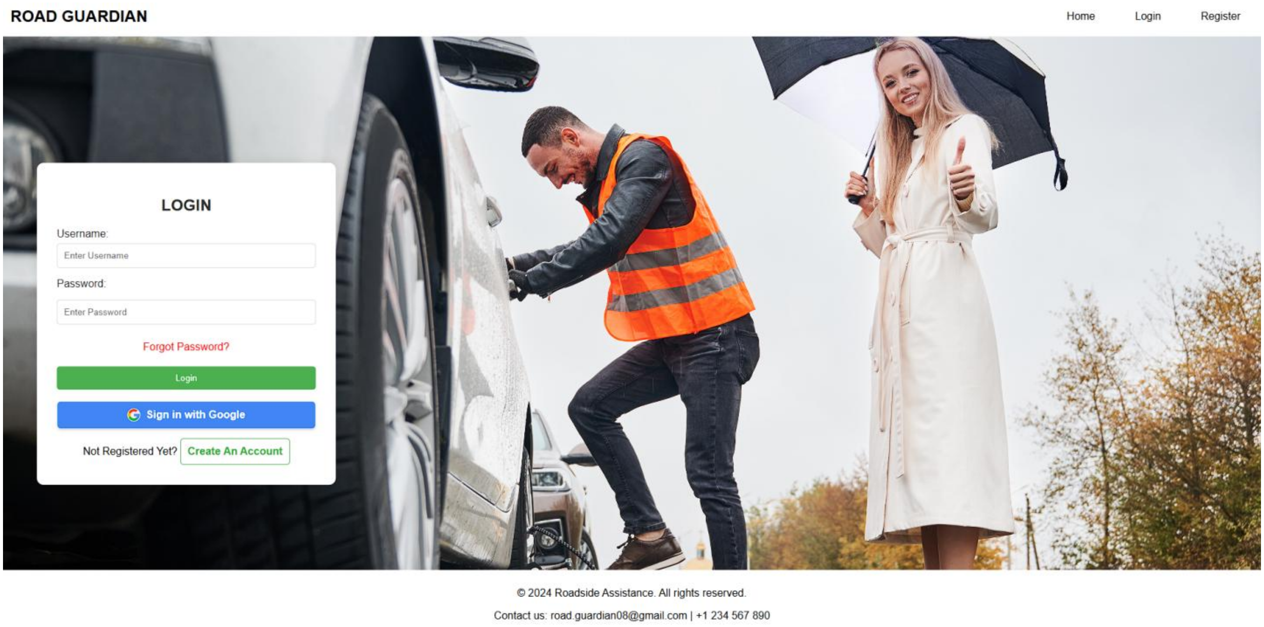
Key characteristics of a collaboration diagram include:

- Objects and Roles: Represents the objects involved in the interaction and their roles.
- Links: Shows the connections between these objects, indicating how they communicate.
- Messages: Labeled arrows between objects that show the flow of information or function calls, with numbers indicating the sequence of interactions.

Purpose: Collaboration diagrams are used to understand and model complex interactions in a system, emphasizing how objects are linked and collaborate to perform a task. They help visualize the structural organization and the exchange of messages between objects to fulfill system requirements.

4.3 USER INTERFACE DESIGN USING FIGMA

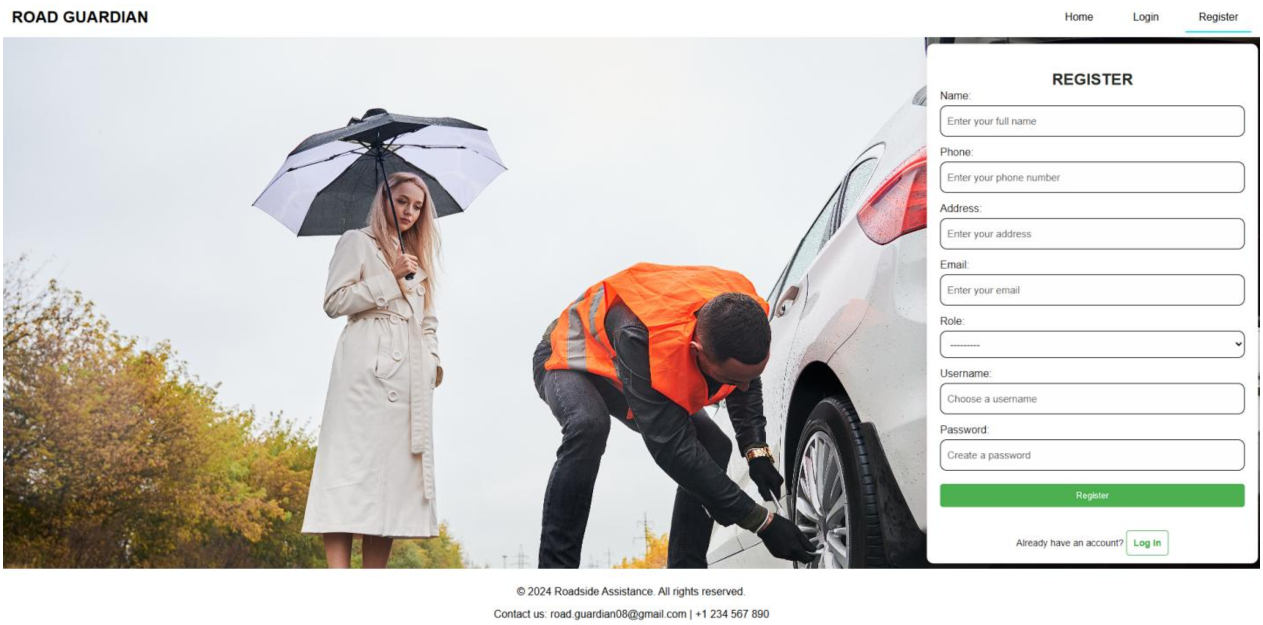
4.3.1 Login Page:



© 2024 Roadside Assistance. All rights reserved.

Contact us: road.guardian08@gmail.com | +1 234 567 890

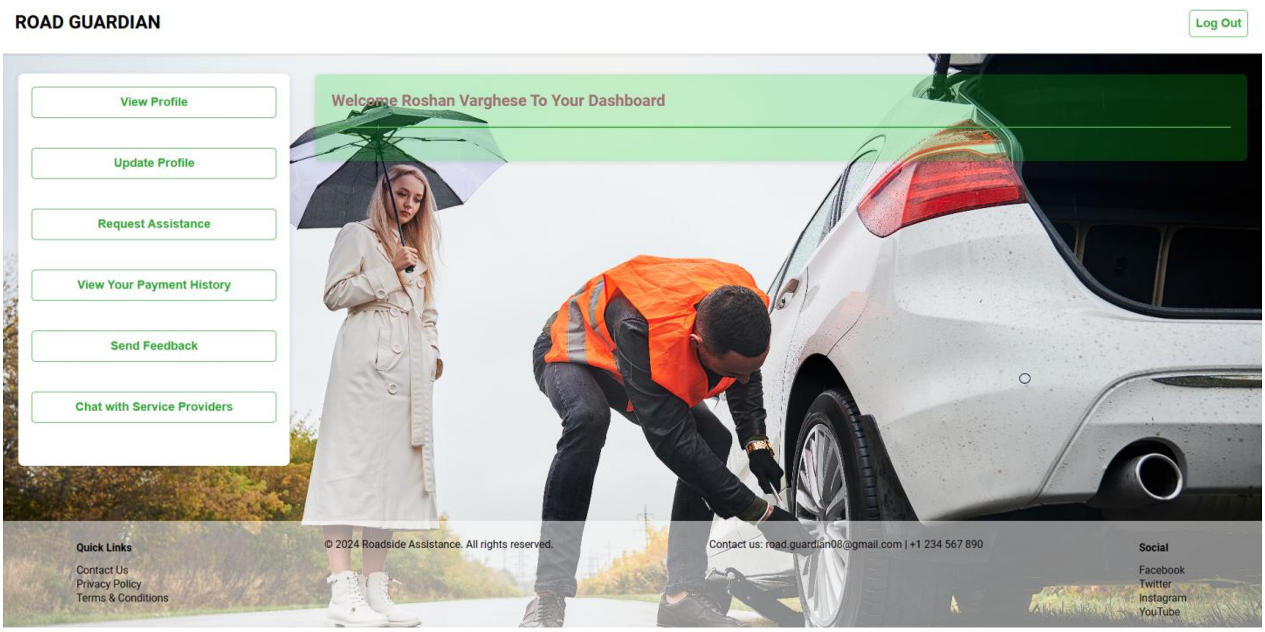
4..3.2 Registration Page:



© 2024 Roadside Assistance. All rights reserved.

Contact us: road.guardian08@gmail.com | +1 234 567 890

4.3.4 User Dashboard:

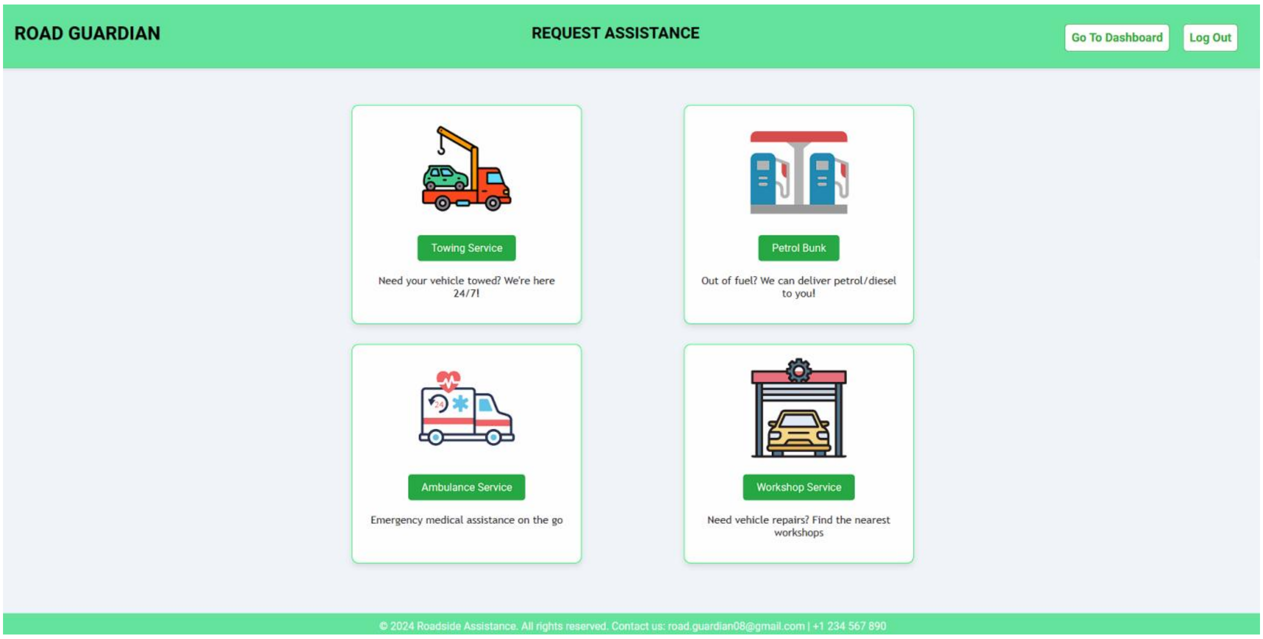


Welcome Roshan Varghese To Your Dashboard

© 2024 Roadside Assistance. All rights reserved.

Contact us: road.guardian08@gmail.com | +1 234 567 890

4.3.4 Request Assistance Page:



Need your vehicle towed? We're here 24/7!

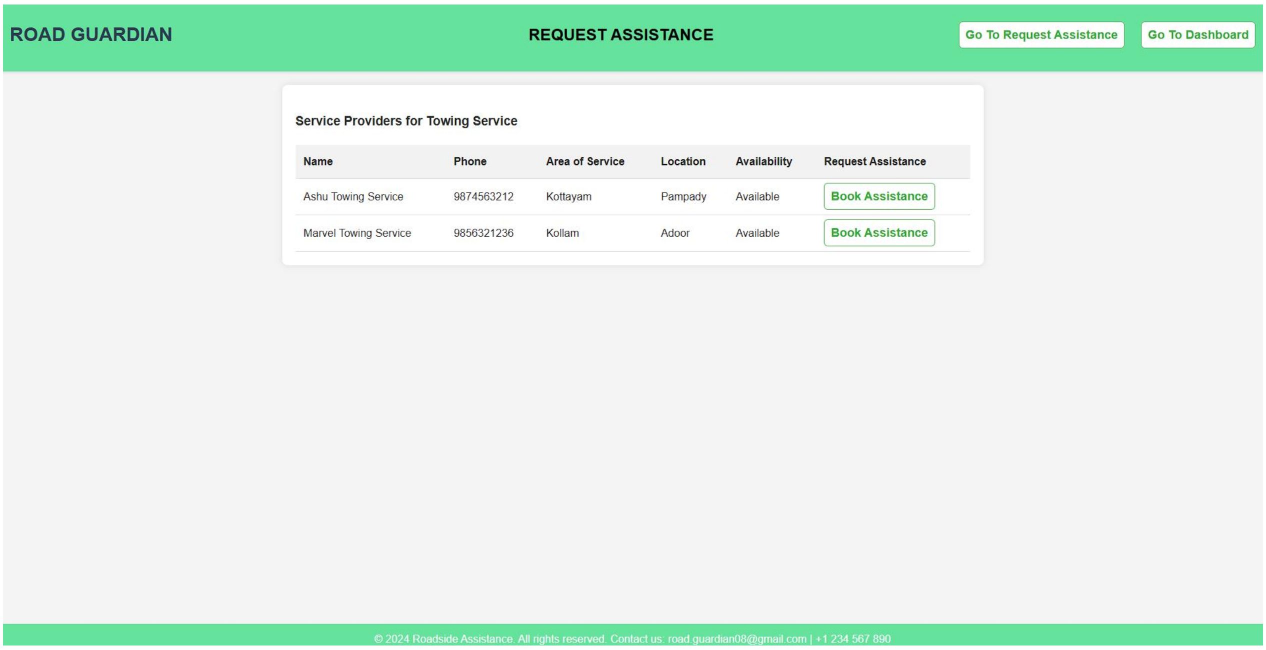
Out of fuel? We can deliver petrol/diesel to you!

Emergency medical assistance on the go

Need vehicle repairs? Find the nearest workshops

© 2024 Roadside Assistance. All rights reserved. Contact us: road.guardian08@gmail.com | +1 234 567 890

4.3.5 Service Providers List Page:

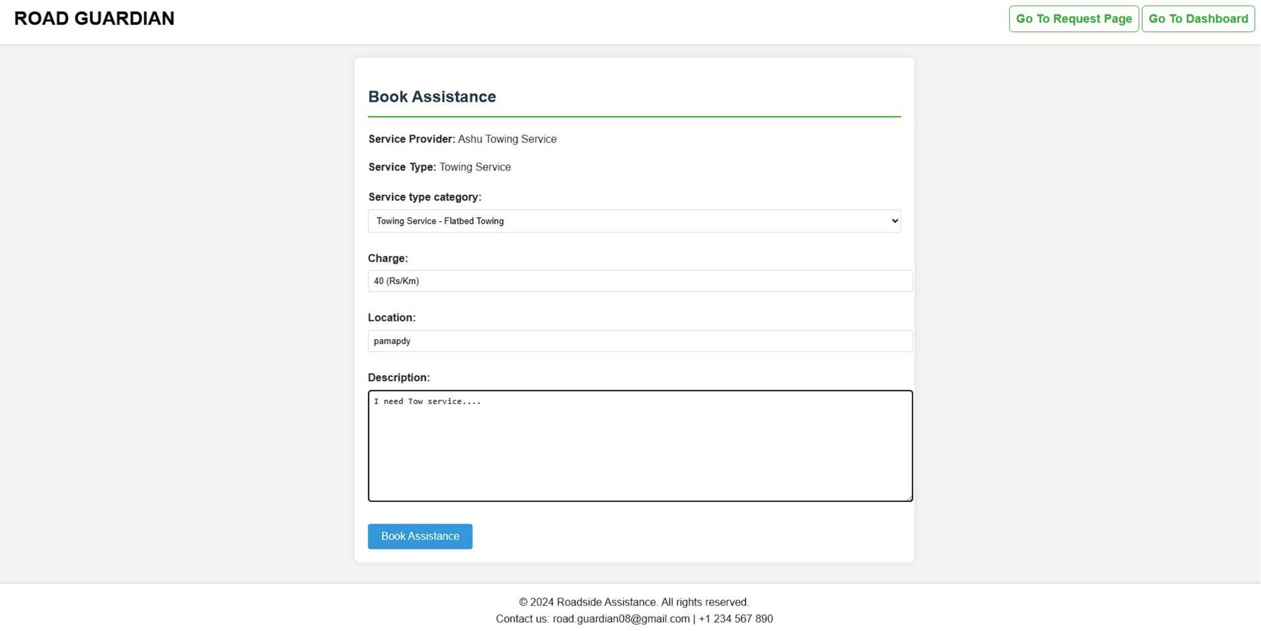


Service Providers for Towing Service

Name	Phone	Area of Service	Location	Availability	Request Assistance
Ashu Towing Service	9874563212	Kottayam	Pampady	Available	Book Assistance
Marvel Towing Service	9856321236	Kollam	Adoor	Available	Book Assistance

© 2024 Roadside Assistance. All rights reserved. Contact us: road.guardian08@gmail.com | +1 234 567 890

4.3.6 Booking Page:



Book Assistance

Service Provider: Ashu Towing Service

Service Type: Towing Service

Service type category: Towing Service - Flatbed Towing

Charge: 40 (Rs/Km)

Location: pamapdy

Description: I need Tow service....

© 2024 Roadside Assistance. All rights reserved.
Contact us: road.guardian08@gmail.com | +1 234 567 890

4.4 DATABASE DESIGN

4.4.1 Relational Database Management System (RDBMS)

For the Road Guardian project, a Relational Database Management System (RDBMS) is used to store and manage data efficiently. The database chosen for this project is MySQL, which is a popular open-source RDBMS known for its reliability, scalability, and robust support for relational data.

Key Tables:

User Table

- Stores information about users, including Vehicle Owners, Service Providers, and Administrators.
- Columns: user_id, username, password, email, phone, role, date_joined.

ServiceRequest Table

- Stores service request details created by users.
- Columns: request_id, user_id (foreign key to User table), location, status, service_type, request_time, completion_time.

ServiceProvider Table

- Contains information about registered service providers.
- Columns: provider_id, user_id (foreign key to User table), service_type, availability, location, certificate.

Feedback Table

- Stores feedback provided by users after service completion.
- Columns: feedback_id, user_id (foreign key to User table), request_id (foreign key to ServiceRequest table), rating, comments, feedback_date.

Admin Table

- Logs administrative actions performed in the system.
- Columns: action_id, admin_id (foreign key to User table), action_type, details, action_date

4.4.2 Normalization

Normalization is the process of structuring a relational database to reduce redundancy and improve data integrity. Road Guardian project uses normalization up to the Third Normal Form (3NF) to ensure that the database is optimized.

Normalization Steps:

First Normal Form (1NF):

- Ensure that all tables have unique rows and each column contains only atomic (indivisible) values.
- Example: In the User table, the phone number should not contain multiple values in a single cell.

Second Normal Form (2NF):

- Ensure that all non-key attributes are fully dependent on the primary key.
- Example: In the ServiceRequest table, all attributes are related to a specific service request, and each row is identified by a unique request_id.

Third Normal Form (3NF):

- Ensure that there are no transitive dependencies, meaning non-key attributes should not depend on other non-key attributes.
- Example: In the Feedback table, the feedback information is directly linked to the request_id, not indirectly through another non-key column.

4.4.3 Sanitization

Sanitization is the process of cleaning input data to prevent malicious attacks such as SQL injection. In the Road Guardian project, sanitization is critical to ensure data security and integrity.

Sanitization Techniques:

Input Validation:

- Validate user inputs to ensure they meet the expected format. For example, checking if an email input follows a standard email format.

Parameterized Queries:

- Use parameterized queries in SQL to avoid SQL injection. This ensures that user input is treated as data and not as executable code.

Escape Special Characters:

- Escape special characters in user inputs (like quotes, slashes) to prevent them from interfering with SQL commands.

XSS Protection:

- Use HTML sanitization to protect against Cross-Site Scripting (XSS) attacks, especially when displaying user-provided data on web pages.

4.4.4 Indexing

Indexing is a technique to improve the speed of data retrieval operations on a database. In the Road Guardian project, indexes are used to enhance the performance of frequently queried data.

Types of Indexes Used:

Primary Index:

- Automatically created on the primary key of each table. For example, in the User table, an index is created on the user_id field to allow quick lookup of user data.

Foreign Key Index:

- Created on columns that are foreign keys. For example, an index is created on user_id in the ServiceRequest table to quickly fetch user-related service requests.

Unique Index:

- Used to enforce the uniqueness of a field. For example, creating a unique index on the username column in the User table ensures no two users have the same username.

Composite Index:

- An index on multiple columns to optimize multi-column searches. For example, a composite index on location and service_type in the ServiceRequest table can speed up location-specific service requests.

Full-Text Index:

- Allows for efficient searching of large text fields, such as feedback comments in the Feedback table, making text-based searches faster.

4.5 TABLE DESIGN

4.5.1 Tbl_users_login

Primary key: **login_id**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	login_id	Int(10)	Auto incremet	Unique Id for every login
2	username	CharField(30)	Primary	Unique identifier for each login attempt
3	User	Int(10)	ForeignKey	Reference to CustomUser (User who is attempting to log in)
4	timestamp	DateTimeField	Auto-Added	Date and time when the login attempt was made

4.5.2 Tbl_User

Primary key: **login_id**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	username	CharField(30)	Primary,Unique	Unique username for the user
2	name	CharField(200	Not Null	Full name of the user
3	phone	CharField(15)	Not Null	Phone number of the user

4	address	CharField(255)	Not Null	Address of the user
5	Email	EmailField(150)	Unique	Email Address of the user
6	role	CharField(20)	Not Null	Role of the user
7	Password	CharField(120)	Not Null	Hashed Password
8	Is_active	BooleanField	Not Null	Shows that the user is active or not

4.5.3 Tbl_ServiceType

Primary key: servicetype_id

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	servicetype_id	Int(20)	Primary Key	Unique identifier for the service type
2	servicetype_name	CharField(100)	Not Null	Name of the service provider
3	description	TextField(100)	Not Null	Description of the service type
4	Image	ImageField	Not Null	Image of the service type

4.5.4 Tbl_ServiceProvider

Primary key: serviceprovider_id

Foreign key:servicetype

Foreignkey:username(ReferenceTable:Tbl_User)

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	serviceprovider_id	AutoField(10)	Primary Key	Autogenerated Primary key
2	username	CharField(100)	Foreign Key	Link to CustomUser
3	service_type	Int(10)	Foreign Key	Link to Service type
4	certificate	ImageField	Not Null	Certifiacte uploaded by service providers
5	area_of_service	CharField	Not Null	Service area description
6	location	Location	Not Null	Location Of service providerss
7	availability_status	BooleanField	Default=TRUE	Avaliablity status of service providers

4.5.5 Tbl_servicetype_category

Primary key: category_id

Foreign key:servicetype

Foreignkey:username(ReferenceTable:Tbl_User)

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	category_id	AutoField(10)	Primary Key	Auto-generated primary key
2	service_type	CharField(100)	ForeignKey	The service type associated with this category
3	category_name	CharField(100)	Not Null	Name of the service category
4	charge	CharField(100)	Not Null	Charge for the service category (optional)

4.5.6 Tbl_Booking

Primary key: booking_id

Foreignkey:username(ReferenceTable:Tbl_User)

Foreignkey:service_provider(ReferenceTable:Tbl_User)

ForeignKey:service_type_category(ReferenceTable:service _type_category)

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	booking_id	AutoField(10)	Primary Key	Auto-generated primary key
2	username	Int(10)	ForeignKey	The user making the booking
3	service_provider	Int(10)	ForeignKey	The service provider linked to the booking
4	service_type_category	Int(10)	ForeignKey	The category of the service requested
5	location	CharField(100)	Not Null	Booking location
6	description	TextField(100)	Not Null	Additional description for the booking
7	status	CharField(100)	Not Null	Status of the booking

4.5.7 Tbl_Feedback

Eg.Primary key: feedback_id

Foreignkey:service_provider_id(ReferenceTable:Tbl_User)

ForeignKey:service_type_category(ReferenceTable:service _provider_id)

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	feedback_id	Int(10)	Primary Key	Auto-generated primary key
2	user_id	Int(10)	ForeignKey	The user giving feedback
3	service_provider_id	Int(10)	ForeignKey	The service provider receiving feedback
4	booking_id	Int(10)	ForeignKey	Booking related to the feedback
5	rating	IntegerField	Not Null	Rating from 1 to 5
6	timestamp	date	DateTimeField	Timestamp of feedback creation

4.5.8 Tbl_payment

Eg.Primary key: payment_id

Foreignkey:service_provider_id(ReferenceTable:Tbl_User)

ForeignKey:user_id(ReferenceTable:service_type_category)

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	payemant_id	Int(10)	Primary Key	Auto-generated primary key
2	user_id	Int(10)	ForeignKey	The user making the payment
3	service_provider_id	Int(10)	ForeignKey	The service provider receiving the payment
4	amount	DecimalField(10)	Not Null	Payment amount (up to two decimal places)
5	date	date	DateTimeField	Timestamp of when the payment was made
6	description	TextField(100)	TextField	Optional description for the payment

CHAPTER 5
SYSTEM TESTING

5.1 INTRODUCTION

Software testing is an essential procedure employed to verify if a computer program functions as intended. It is conducted to ensure that the software performs its designated tasks accurately and complies with the prescribed requirements and standards. Validation is the process of inspecting and assessing software to confirm its adherence to the specified criteria. Software testing is a method

for assessing the performance of a program, often in conjunction with techniques like code inspection and program walk through. Validation ensures that the software aligns with the user's expectations and requirements.

Several principles and objectives guide the process of software testing, including:

1. Testing is the practice of executing a program with the primary aim of identifying errors.
2. An effective test case is one that has a high likelihood of uncovering previously undiscovered errors.
3. A successful test is one that exposes previously undiscovered errors.

When a test case operates effectively and accomplishes its objectives, it can detect flaws within the

software. This demonstrates that the computer program is functioning as intended and is performing

well. The process of evaluating a computer program encompasses three primary aspects:

1. Correctness assessment
2. Evaluation of implementation efficiency
3. Examination of computational complexity

5.1 TEST PLAN

The test plan outlines the approach, scope, resources, and schedule for testing activities. It includes identifying the testing methods, tools, and environment to ensure that each module works independently and in conjunction with other components. This project follows a structured test plan, beginning with Unit Testing and progressing to Integration, Validation, Output, and Automation Testing.

5.2.1 Unit Testing

Unit testing checks the smallest part of a software design - the software component or module. Testing important control paths within a module using the design guide to find errors. This means how difficult the tests are for each small part of a program and what parts of the program haven't been tested yet. Unit testing is a type of testing that looks at how the code works inside and can be done at the same time for different parts of the program.

Before starting any other test, we need to check if the data flows correctly between different parts of the computer program. If the information doesn't move in and out correctly, all other checks are pointless. When designing something, it's important to think about what could go wrong and make a plan for how to deal with those problems. This can mean redirecting the process or stopping it completely.

The Sell-Soft System was tested by looking at each part by itself and trying different tests on it. Some mistakes in the design of the modules were discovered and then fixed. After writing the instructions for different parts, each part is checked and tried out separately. We got rid of extra code and made sure everything works the way it should.

5.2.2 Integration Testing

Integration testing is a critical process in software development that involves constructing a program while simultaneously identifying errors in the interaction between different program components. The primary objective is to utilize tested individual parts and assemble them into a program according to the initial plan. This comprehensive testing approach assesses the entire program to ensure its proper and correct functionality. As issues are identified and resolved during integration testing, it's not uncommon for new problems to surface, leading to an ongoing cycle of testing and refinement. After each individual component of the system is thoroughly examined, these components are integrated to ensure they function harmoniously. Additionally, efforts are made to standardize all programs to ensure uniformity rather than having disparate versions.

5.2.3 Validation Testing or System Testing

The final phase of testing involves a comprehensive examination of the entire system to ensure the correct interaction of various components, including different types of instructions and building blocks. This testing approach is referred to as Black Box testing or System testing.

Black Box testing is a method employed to determine if the software functions as intended. It assists

software engineers in identifying all program issues by employing diverse input types. Black Box testing encompasses the assessment of errors in functions, interfaces, data access, performance, as well as initialization and termination processes. It is a vital technique to verify that the software meets its intended requirements and performs its functions correctly.

5.2.4 Output Testing or User Acceptance Testing

System testing is conducted to assess user satisfaction and alignment with the company's requirements. During the development or update of a computer program, it's essential to maintain a connection with the end-users. This connection is established through the following elements:

- Input Screen Designs.
- Output Screen Designs.

To perform system testing, various types of data are utilized. The preparation of test data plays a crucial role in this phase. Once the test data is gathered, it is used to evaluate the system under investigation. When issues are identified during this testing, they are addressed by following established procedures. Records of these corrections are maintained for future reference and improvement. This process ensures that the system functions effectively and meets the needs of both users and the organization

5.2.5 Automation Testing

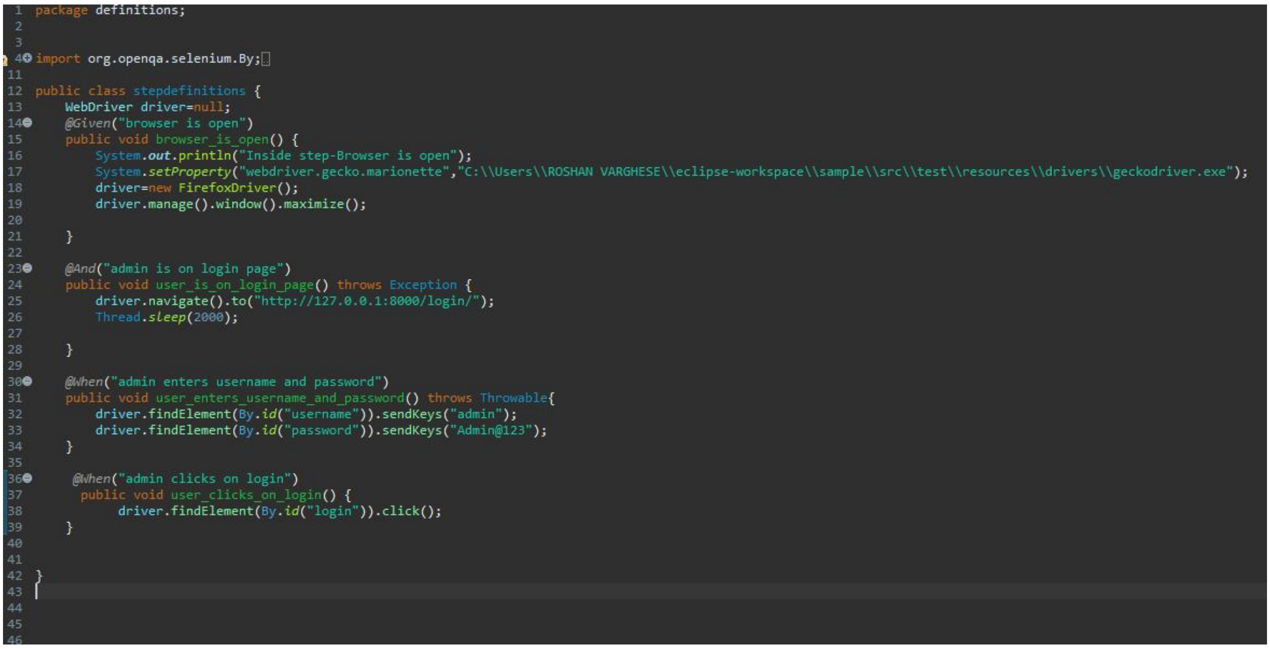
Automated testing is a method employed to verify that software functions correctly and complies with established standards before it is put into official use. This type of testing relies on written instructions that are executed by testing tools. Specifically, UI automation testing involves the use of specialized tools to automate the testing process. Instead of relying on manual interactions where individuals click through the application to ensure its proper functioning, scripts are created to automate these tests for various scenarios. Automating testing is particularly valuable when it is necessary to conduct the same test across multiple computers simultaneously, streamlining the testing process and ensuring consistency

5.2.6 Selenium Testing

Selenium is a valuable and free tool designed for automating website testing. It plays a crucial role for web developers as it simplifies the testing process. Selenium automation testing refers to the practice of using Selenium for this purpose. Selenium isn't just a single tool; it's a collection of tools, each serving distinct functions in the realm of automation testing. Manual testing is a necessary aspect of application development, but it can be monotonous and repetitive. To alleviate these challenges, Jason Huggins, an employee at ThoughtWorks, devised a method for automating testing procedures, replacing manual tasks. He initially created a tool named the JavaScriptTestRunner to facilitate automated website testing, and in 2004, it was rebranded as Selenium.

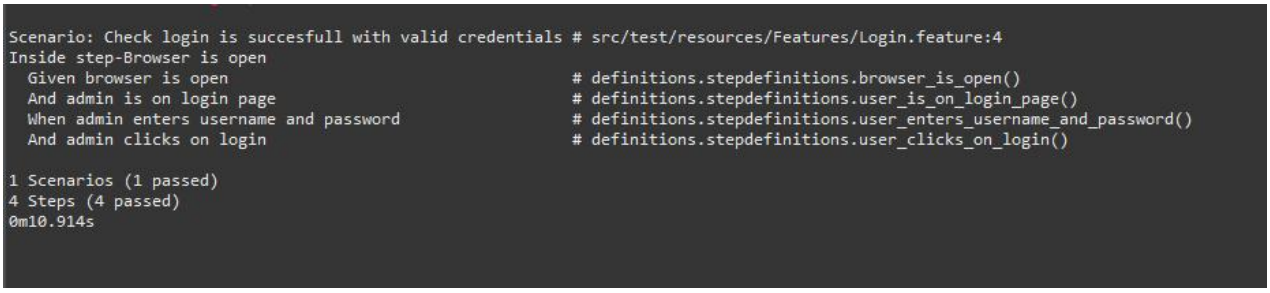
Test Case 1 - Admin Login

Code



```java
package definitions;


import org.openqa.selenium.By;

public class stepdefinitions {
    WebDriver driver=null;
    @Given("browser is open")
    public void browser_is_open() {
        System.out.println("Inside step-Browser is open");
        System.setProperty("webdriver.gecko.marionette","C:\\Users\\ROSHAN VARGHESE\\eclipse-workspace\\sample\\src\\test\\resources\\drivers\\geckodriver.exe");
        driver=new FirefoxDriver();
        driver.manage().window().maximize();

    }

    @And("admin is on login page")
    public void user_is_on_login_page() throws Exception {
        driver.navigate().to("http://127.0.0.1:8000/login/");
        Thread.sleep(2000);

    }

    @When("admin enters username and password")
    public void user_enters_username_and_password() throws Throwable{
        driver.findElement(By.id("username")).sendKeys("admin");
        driver.findElement(By.id("password")).sendKeys("Admin@123");
    }

     @When("admin clicks on login")
      public void user_clicks_on_login() {
           driver.findElement(By.id("login")).click();
    }


}
```

Screenshot



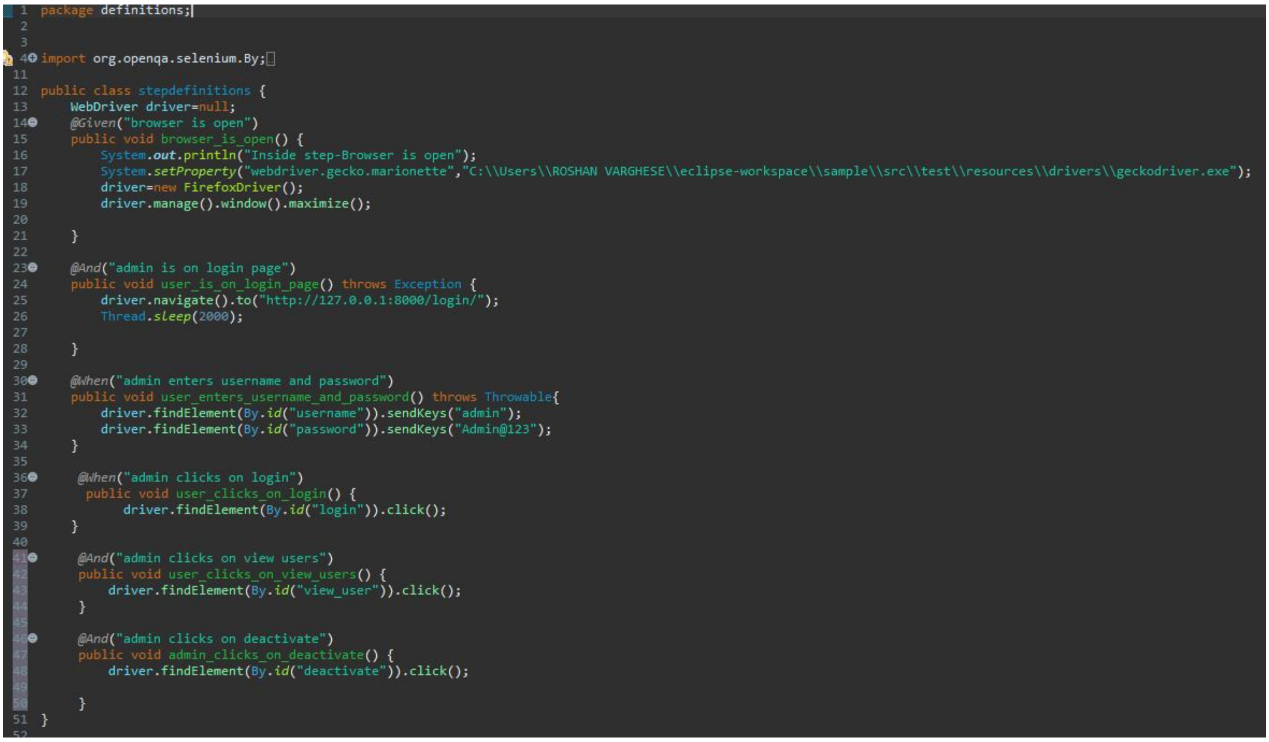
```
Scenario: Check login is succesfull with valid credentials # src/test/resources/Features/Login.feature:4
Inside step-Browser is open
  Given browser is open                                   # definitions.stepdefinitions.browser_is_open()
  And admin is on login page                              # definitions.stepdefinitions.user_is_on_login_page()
  When admin enters username and password                 # definitions.stepdefinitions.user_enters_username_and_password()
  And admin clicks on login                               # definitions.stepdefinitions.user_clicks_on_login()

1 Scenarios (1 passed)
4 Steps (4 passed)
0m10.914s
```

Test Case 1

Project Name: ROAD GUARDIAN					
Login Test Case					
Test Case ID: Test 1			**Test Designed By:Roshan Varghese**		
Test Priority(Low/Medium/High) : Medium			**Test Designed Date: 20/10/2024**		
Module Name:Login module			**Test Executed By :Ms Gloriya Mathew**		
Test Title : Admin Login			**Test Execution Date: 21/10/2024**		
Description: : Admin has a valid Username/password					
Pre-Condition: Admin has valid username and password					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fail)**
1	Navigate to login page		Login form should be displayed	Login form is displayed	pass
2	Provide valid username	Username: admin	Admin should be able to login	User logged in	pass
2	Provide valid password	Password: Admin@123			
4	Click on login button				
Post-Condition: User is validated with database and successfully logs into account. The account session details are logged in database					

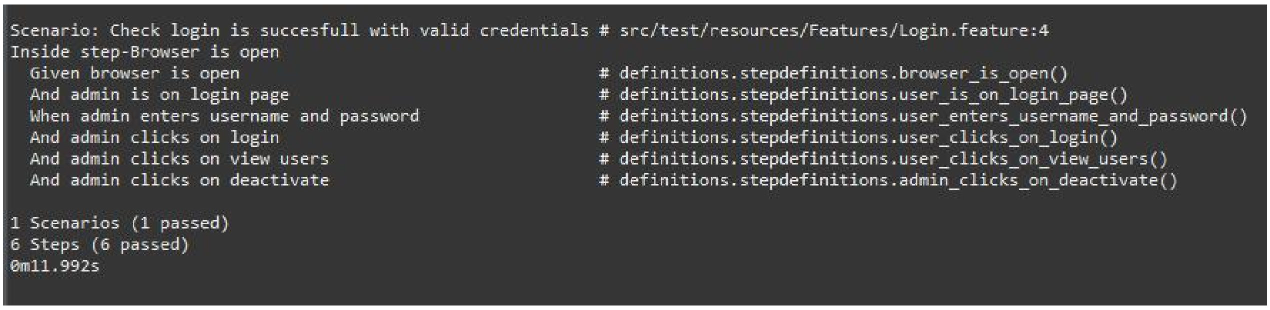
Test Case 2: Admin Activate/Deactivate A User

Code



```java
package definitions;


import org.openqa.selenium.By;

public class stepdefinitions {
    WebDriver driver=null;
    @Given("browser is open")
    public void browser_is_open() {
        System.out.println("Inside step-Browser is open");
        System.setProperty("webdriver.gecko.marionette","C:\\Users\\ROSHAN VARGHESE\\eclipse-workspace\\sample\\src\\test\\resources\\drivers\\geckodriver.exe");
        driver=new FirefoxDriver();
        driver.manage().window().maximize();

    }

    @And("admin is on login page")
    public void user_is_on_login_page() throws Exception {
        driver.navigate().to("http://127.0.0.1:8000/login/");
        Thread.sleep(2000);

    }

    @When("admin enters username and password")
    public void user_enters_username_and_password() throws Throwable{
        driver.findElement(By.id("username")).sendKeys("admin");
        driver.findElement(By.id("password")).sendKeys("Admin@123");
    }

     @When("admin clicks on login")
      public void user_clicks_on_login() {
           driver.findElement(By.id("login")).click();
    }

     @And("admin clicks on view users")
     public void user_clicks_on_view_users() {
         driver.findElement(By.id("view_user")).click();
     }

     @And("admin clicks on deactivate")
     public void admin_clicks_on_deactivate() {
         driver.findElement(By.id("deactivate")).click();

     }
}
```

Screenshot



```
Scenario: Check login is succesfull with valid credentials # src/test/resources/Features/Login.feature:4
Inside step-Browser is open
  Given browser is open                                    # definitions.stepdefinitions.browser_is_open()
  And admin is on login page                               # definitions.stepdefinitions.user_is_on_login_page()
  When admin enters username and password                  # definitions.stepdefinitions.user_enters_username_and_password()
  And admin clicks on login                                # definitions.stepdefinitions.user_clicks_on_login()
  And admin clicks on view users                           # definitions.stepdefinitions.user_clicks_on_view_users()
  And admin clicks on deactivate                           # definitions.stepdefinitions.admin_clicks_on_deactivate()

1 Scenarios (1 passed)
6 Steps (6 passed)
0m11.992s
```

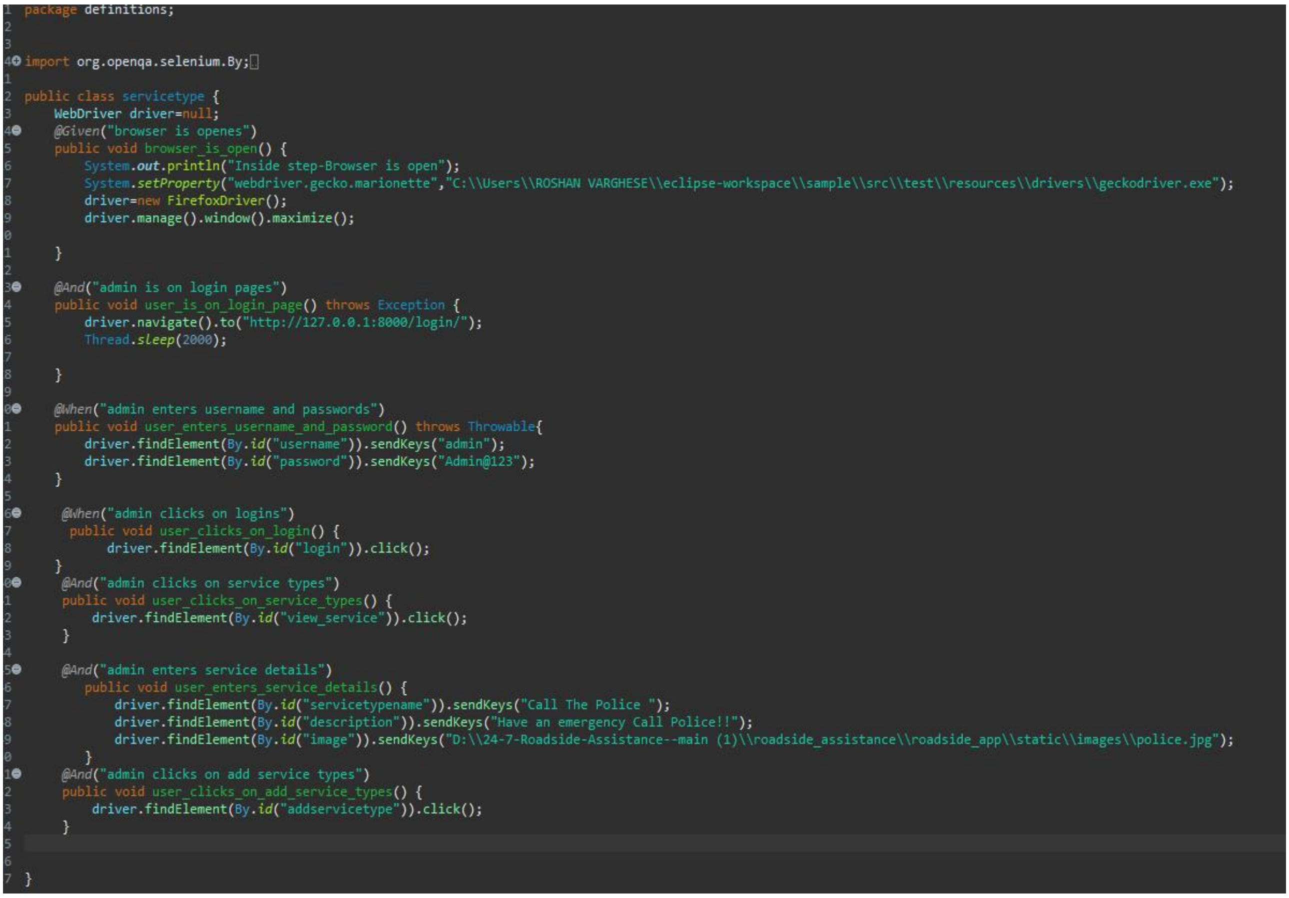
Test Case 2

Project Name: ROAD GUARDIAN					
Activate/Deactivate A User Test Case					
Test Case ID: Test_2			**Test Designed By:Roshan Varghese**		
Test Priority(Low/Medium/High) :Medium			**Test Designed Date: 23/10/2024**		
Module Name: Status changing module			**Test Executed By :Ms Gloriya Mathew**		
Test Title : Admin Activate/Deactivate			**Test Execution Date: 24/10/2024**		
Description: : Admin change status					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fail)**
1	Navigate to login page		Login form should be displayed	Login form is displayed	pass
2	Provide valid username	Username: rosh	User should be able to login	User logged in	pass
2	Provide valid password	Password: Rosh@123			
4	Click on login button				
5	Click on manage users		List of users should be displayed	List of users displayed	Pass
6	Click on deactivate		When click it shoulde be deactivate	When clicked the user is deactivated	Pass

Post-Condition: Admin perform a activity to view user and deactivate the corresponding user

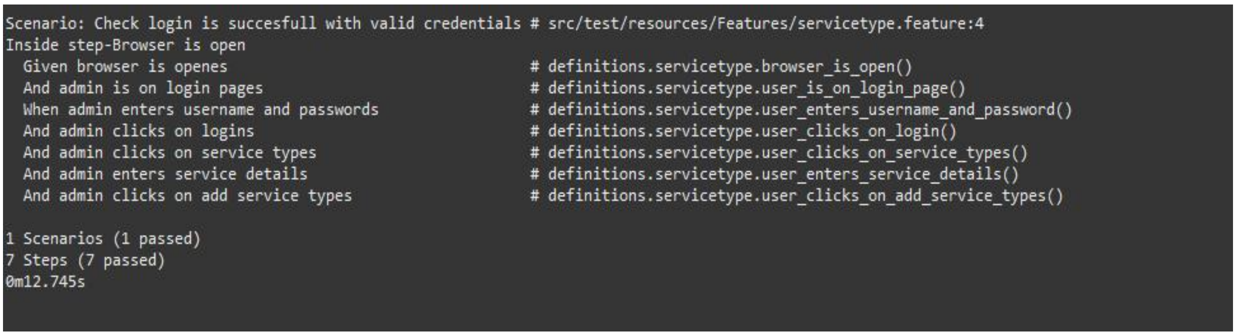
Test Case 3: Admin Add Service Type

Code



```java
package definitions;


import org.openqa.selenium.By;□

public class servicetype {
    WebDriver driver=null;
    @Given("browser is openes")
    public void browser_is_open() {
        System.out.println("Inside step-Browser is open");
        System.setProperty("webdriver.gecko.marionette","C:\\Users\\ROSHAN VARGHESE\\eclipse-workspace\\sample\\src\\test\\resources\\drivers\\geckodriver.exe");
        driver=new FirefoxDriver();
        driver.manage().window().maximize();

    }

    @And("admin is on login pages")
    public void user_is_on_login_page() throws Exception {
        driver.navigate().to("http://127.0.0.1:8000/login/");
        Thread.sleep(2000);

    }

    @When("admin enters username and passwords")
    public void user_enters_username_and_password() throws Throwable{
        driver.findElement(By.id("username")).sendKeys("admin");
        driver.findElement(By.id("password")).sendKeys("Admin@123");
    }

     @When("admin clicks on logins")
      public void user_clicks_on_login() {
           driver.findElement(By.id("login")).click();
    }
     @And("admin clicks on service types")
     public void user_clicks_on_service_types() {
         driver.findElement(By.id("view_service")).click();
     }

     @And("admin enters service details")
        public void user_enters_service_details() {
            driver.findElement(By.id("servicetypename")).sendKeys("Call The Police ");
            driver.findElement(By.id("description")).sendKeys("Have an emergency Call Police!!");
            driver.findElement(By.id("image")).sendKeys("D:\\24-7-Roadside-Assistance--main (1)\\roadside_assistance\\roadside_app\\static\\images\\police.jpg");
        }
     @And("admin clicks on add service types")
     public void user_clicks_on_add_service_types() {
         driver.findElement(By.id("addservicetype")).click();
     }


}
```

Screenshot



```
Scenario: Check login is succesfull with valid credentials # src/test/resources/Features/servicetype.feature:4
Inside step-Browser is open
  Given browser is openes                                  # definitions.servicetype.browser_is_open()
  And admin is on login pages                              # definitions.servicetype.user_is_on_login_page()
  When admin enters username and passwords                 # definitions.servicetype.user_enters_username_and_password()
  And admin clicks on logins                               # definitions.servicetype.user_clicks_on_login()
  And admin clicks on service types                        # definitions.servicetype.user_clicks_on_service_types()
  And admin enters service details                         # definitions.servicetype.user_enters_service_details()
  And admin clicks on add service types                    # definitions.servicetype.user_clicks_on_add_service_types()

1 Scenarios (1 passed)
7 Steps (7 passed)
0m12.745s
```

Test Case 3

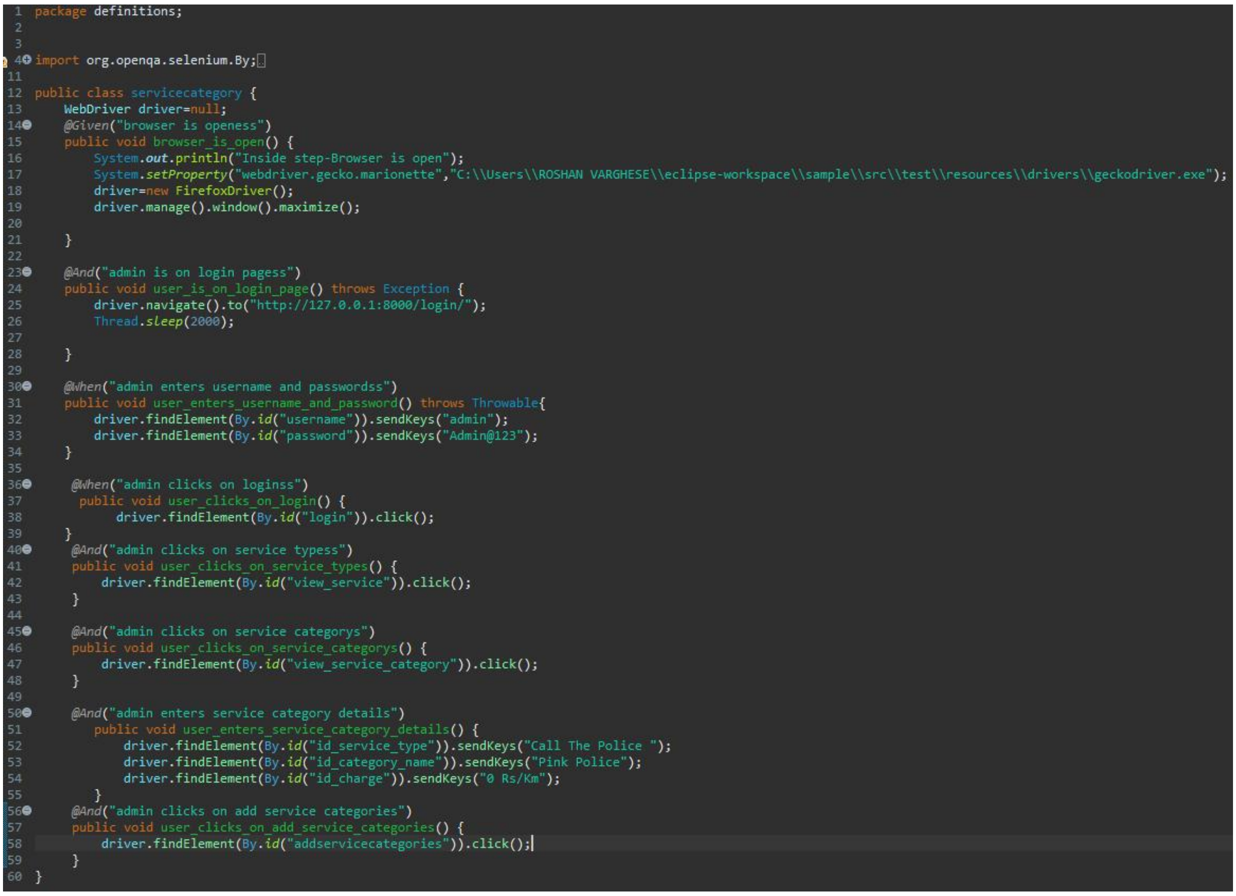
Project Name: ROAD GUARDIAN					
Admin Add Service type					
Test Case ID: Test 3			**Test Designed By:Roshan Varghese**		
Test Priority(Low/Medium/High) : Medium			**Test Designed Date: 26/10/2024**		
Module Name:AdminAdd Service type			**Test Executed By :Ms Gloriya Mathew**		
Test Title : Admin Add Service type			**Test Execution Date: 27/10/2024**		
Description: : Admin add service type					
Pre-Condition : Admin perform activity add new service					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fail)**
1	Navigate to login page		Login form should be displayed	Login form is displayed	pass
2	Provide valid username	Username: rosh	User should be able to login	User logged in	pass
3	Provide valid password	Password: Rosh@123			
4	Click on login button				
5	Click on manage service types		Area to add a new service type should be displayed	Area to add a new service type displayed	Pass

6	Provide a service type name	Service type name:call the police	Admin should be able to Add service type	Admin added service type	pass
7	Provide a service type image	Image: Police.png			
8	Click on add service type				

Post-Condition: Admin perform a activity to add new service type

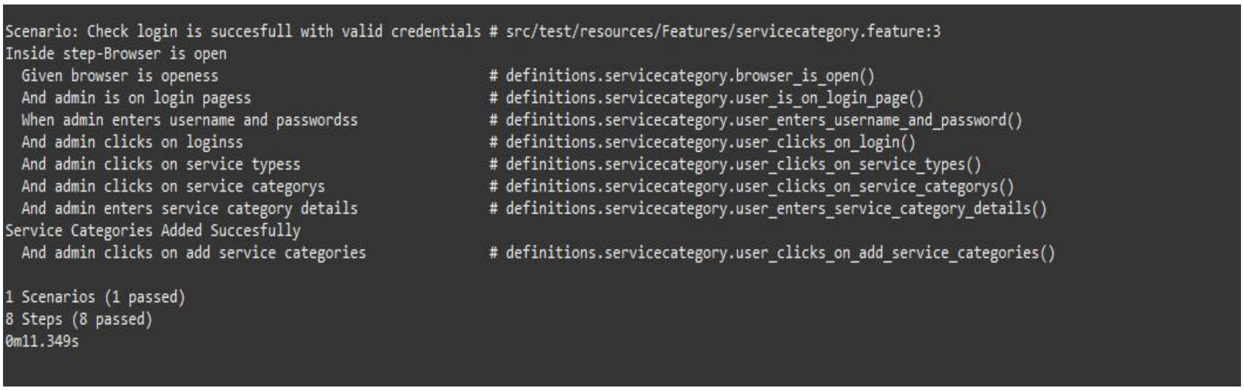
Test Case 4: Admin Add Service Categories

Code



```java
package definitions;


import org.openqa.selenium.By;

public class servicecategory {
    WebDriver driver=null;
    @Given("browser is openess")
    public void browser_is_open() {
        System.out.println("Inside step-Browser is open");
        System.setProperty("webdriver.gecko.marionette","C:\\Users\\ROSHAN VARGHESE\\eclipse-workspace\\sample\\src\\test\\resources\\drivers\\geckodriver.exe");
        driver=new FirefoxDriver();
        driver.manage().window().maximize();

    }

    @And("admin is on login pagess")
    public void user_is_on_login_page() throws Exception {
        driver.navigate().to("http://127.0.0.1:8000/login/");
        Thread.sleep(2000);

    }

    @When("admin enters username and passwordss")
    public void user_enters_username_and_password() throws Throwable{
        driver.findElement(By.id("username")).sendKeys("admin");
        driver.findElement(By.id("password")).sendKeys("Admin@123");
    }

     @When("admin clicks on loginss")
      public void user_clicks_on_login() {
            driver.findElement(By.id("login")).click();
    }
     @And("admin clicks on service typess")
     public void user_clicks_on_service_types() {
         driver.findElement(By.id("view_service")).click();
     }

     @And("admin clicks on service categorys")
     public void user_clicks_on_service_categorys() {
         driver.findElement(By.id("view_service_category")).click();
     }

     @And("admin enters service category details")
        public void user_enters_service_category_details() {
            driver.findElement(By.id("id_service_type")).sendKeys("Call The Police ");
            driver.findElement(By.id("id_category_name")).sendKeys("Pink Police");
            driver.findElement(By.id("id_charge")).sendKeys("0 Rs/Km");
        }
     @And("admin clicks on add service categories")
     public void user_clicks_on_add_service_categories() {
         driver.findElement(By.id("addservicecategories")).click();
     }
}
```

Screenshot:



```
Scenario: Check login is succesfull with valid credentials # src/test/resources/Features/servicecategory.feature:3
Inside step-Browser is open
  Given browser is openess                                 # definitions.servicecategory.browser_is_open()
  And admin is on login pagess                             # definitions.servicecategory.user_is_on_login_page()
  When admin enters username and passwordss                # definitions.servicecategory.user_enters_username_and_password()
  And admin clicks on loginss                              # definitions.servicecategory.user_clicks_on_login()
  And admin clicks on service typess                       # definitions.servicecategory.user_clicks_on_service_types()
  And admin clicks on service categorys                    # definitions.servicecategory.user_clicks_on_service_categorys()
  And admin enters service category details                # definitions.servicecategory.user_enters_service_category_details()
Service Categories Added Succesfully
  And admin clicks on add service categories               # definitions.servicecategory.user_clicks_on_add_service_categories()

1 Scenarios (1 passed)
8 Steps (8 passed)
0m11.349s
```

Test Case 4

Project Name: **ROAD GUARDIAN**					
Admin add service type category					
Test Case ID: Test 3			**Test Designed By:Roshan Varghese**		
Test Priority(Low/Medium/High) : Medium			**Test Designed Date:29/10/2024**		
Module Name: Admin add service type category			**Test Executed By :Ms Gloriya Mathew**		
Test Title : Admin Add Service type Category			**Test Execution Date: 30/10/2024**		
Description: : Admin add service type category					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fail)**
1	Navigate to login page		Login form should be displayed	Login form is displayed	pass
2	Provide valid username	Username: rosh	User should be able to login	User logged in	pass
3	Provide valid password	Password: Rosh@123			
4	Click on login button				

5	Click on manage service types		Area to add a new service type should be displayed	Area to add a new service type displayed	Pass
6	Click on manage Servicetype categories		Area to add a new service type category should be displayed	Area to add a new service type category is displayed	Pass
7	Provide a service type category name	Service type category name:pink police	Admin should be able to Add service type category	Admin added service type category	pass
8	Provide a service type category charge	Charge:free			
9	Click on add service type category				
Post-Condition: Admin perform a activity toadd new service type category					

CHAPTER 6
IMPLEMENTATION

6.1 INTRODUCTION

This section outlines the implementation of the Road Guardian Platform, detailing the steps taken to bring the platform from development to deployment. Implementation involves configuring the application, setting up essential software and database integrations, and conducting thorough testing to ensure a stable and user-friendly experience.

6.1 IMPLEMENTATION PROCEDURES

The implementation procedures for the platform include setting up the development and production environments, configuring databases, and deploying the web application. The core steps also involve integrating the machine learning models for predictive analytics, setting up user authentication, and configuring communication channels for notifications and live support.

Environment Setup:

- Configure Django and MySQL databases in the production environment.
- Install necessary libraries and dependencies for frontend and backend functionalities.

Data Integration:

- Load initial data for service providers, types of services, and test data for predictive analytics.

Testing and Debugging

- Conduct unit and integration testing to ensure each module functions correctly.
- Fix any issues and fine-tune the database queries and routing algorithms for optimal performance.

Deployment:

- Deploy the application on a server or cloud platform with necessary configurations for security and scalability.

- Set up monitoring tools to ensure system health and track user activity.

6.2.1 User Training

User training is essential to familiarize users with the functionalities of the platform, including how to request assistance, track service providers, and update personal profiles. Training includes:

- **Guided Tours**: An introductory guide through the main features upon first login.
- **User Manual**: A step-by-step document outlining how to use the platform effectively
- **Support Access**: Users are given access to help sections and FAQs within the application.

6.1.1 Training on the Application Software

Once the fundamental computer skills have been covered, the next step is to instruct individuals on using a new software program. This training should provide a comprehensive understanding of the new system, including navigation through screens, accessing help resources, error management, and resolution procedures. The training aims to equip users or groups with the knowledge and skills necessary to effectively utilize the system or its components. It's important to note that training may be tailored differently for various groups of users and individuals in different roles within the organization to cater to their specific needs and requirements.Training on the application software is provided for administrators and service providers who will manage services and handle user requests. This includes:

- **Admin Training**: Focus on managing service requests, accessing analytics dashboards, and overseeing the communication channels
- **Service Provider Training**: Training for service providers on managing their availability, updating their profiles, and tracking assignments.

6.1.2 System Maintenance

System maintenance is the process of keeping a software system up-to-date and functioning properly after it has been deployed. It involves a variety of activities that are designed to ensure that the system remains stable, secure, and efficient over time. Some of the key activities involved in system maintenance include:

- Updates and patches: One of the most important aspects of system maintenance is keeping the system up-to-date with the latest software updates and security patches. These updates and patches are released periodically by software vendors to fix bugs, add new features, and address security vulnerabilities. Applying these updates in a timely manner is essential to ensure that the system remains secure and reliable.
- Performance monitoring: Regular performance monitoring is another key aspect of system maintenance. This involves tracking system performance metrics such as CPU usage,memory utilization, and network traffic to identify potential issues and optimize system performance.
- Backup and recovery: Backing up system data and ensuring that it can be recovered in the event of a system failure or data loss is an important part of system maintenance. This involves creating regular backups of system data and testing the recovery process to ensure that it works as expected.
- Security management: Maintaining system security is critical to protecting sensitive data and preventing unauthorized access to the system. This involves implementing security controls such as firewalls, antivirus software, and access controls, regularly monitoring the system for potential security breaches.
- Hardware maintenance: Ensuring that hardware components such as servers, storage devices, and network devices are functioning properly is another key aspect of system maintenance. This involves performing regular maintenance tasks such as cleaning, repairs, and upgrades to ensure that the hardware components remain in good working condition.
- Effective system maintenance is critical to ensuring that a software system remains reliable, secure, and efficient over time. By implementing a proactive system maintenance strategy, organizations can minimize downtime, prevent system failures, and optimize system performance.

6.1.3 Hosting

Hosting refers to the process of storing and serving website files on a remote server, which can be accessed by visitors over the internet. When you create a website, you need to have it hosted on a server so that it can be available for others to view. There are various types of hosting options available such as shared hosting, dedicated hosting, VPS hosting, cloud hosting, etc. The choice of hosting depends on the size of the website, its traffic volume, and the level of control and flexibility required by

Render

Render is a cloud hosting service that provides a streamlined platform for deploying web applications, APIs, static sites, and databases. It simplifies the process of hosting applications by automating tasks such as server setup, scaling, and load balancing, which allows developers to focus on building their applications rather than managing infrastructure. Render supports various languages and frameworks, including Django, and offers seamless integration with Git for continuous deployment

Procedure for hosting a website on render:

Step 1: Login to your render account

Step 2: Create a new project

Step 3: Create a requirements.txt file which will include all your dependencies to be installed.

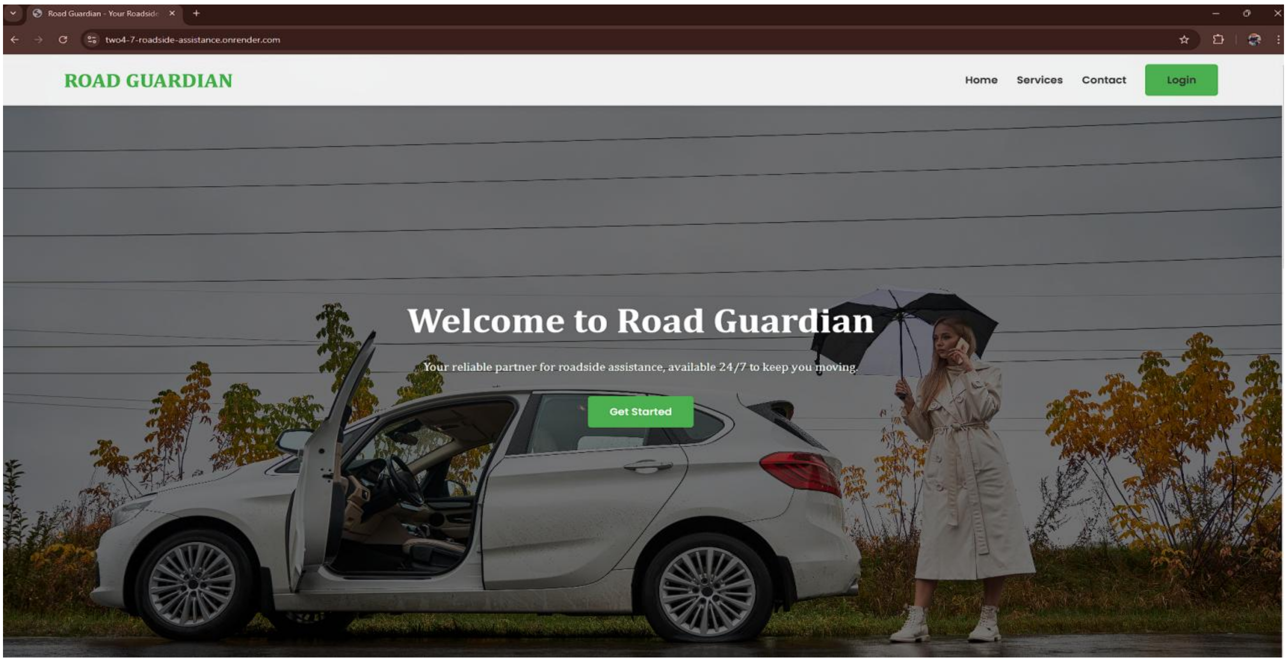
Step 4: Upload the content which is to be hosted into GitHub

Step 5: Change the setting of your project to include the host also and then deploy.

Hosted Website: Render

Hosted Link:https://two4-7-roadside-assistance.onrender.com

Screenshot:


Road Guardian - Your Roadside
two4-7-roadside-assistance.onrender.com
ROAD GUARDIAN
Home
Services
Contact
Login
Get Started

Welcome to Road Guardian

Your reliable partner for roadside assistance, available 24/7 to keep you moving.

CHAPTER 7
CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

The Road Guardian project has successfully addressed the need for a reliable, round-the-clock support system for motorists. The project has demonstrated how modern technology can be harnessed to create a user-friendly platform that bridges the gap between stranded drivers and nearby service providers. This solution not only enhances convenience for users but also provides an effective way for service providers to reach their target audience. Through the integration of real-time booking, service tracking, and a feedback system, the platform ensures transparency, efficiency, and customer satisfaction.

The choice of Django as the web framework, complemented by a MySQL database, has allowed the system to maintain a robust backend structure while delivering a seamless frontend experience. Additional features like role-based access, service type management, and comprehensive booking records have added versatility to the system, ensuring that it caters to a broad spectrum of user needs. Security, a vital concern for any digital platform, has been addressed with custom user authentication and authorization mechanisms, further enhancing trustworthiness.

The project has also highlighted the importance of adaptability and user feedback in system development. Iterative improvements and user-centered design choices have contributed to a product that is both functional and scalable. As technology and user expectations evolve, this project provides a solid foundation for future enhancements, ensuring that it remains a relevant and valuable tool for years to come.

7.1 FUTURE SCOPE

The Road Guardian project has a solid foundation but leaves room for several enhancements and expansions in the future:

- **Mobile Application Development**: Creating a dedicated mobile app would make the platform even more accessible, allowing users to request services instantly from their mobile devices.

- **Real-time GPS Tracking**: Integrating real-time GPS tracking for service providers can enhance the user experience by allowing them to monitor the service provider's location and estimated arrival time.

- **AI-based Assistance**: Incorporating AI-driven chatbots could provide users with immediate guidance and answers to frequently asked questions, reducing response times during emergencies.

- **Automated Dispatch System**: Implementing an automated dispatch system for service providers based on proximity and availability can optimize service allocation and reduce wait times.

- **Service Expansion**: The platform could extend its offerings to include additional services like minor on-site repairs, battery replacements, and more.

- **Multi-Language Support**: Adding multi-language support would make the platform accessible to a broader audience, ensuring that language barriers do not hinder usability.

- **Partnership with Insurance Companies**: Establishing partnerships with insurance companies could provide users with more comprehensive assistance options, like integrating roadside assistance into insurance packages.

- **Data Analytics and Insights**: Adding analytics to track common roadside incidents can help predict peak times for services, leading to better resource allocation and targeted service improvements.

- **User Reviews and Ratings**: Enhancing the feedback system with detailed ratings and reviews could provide users with a better understanding of service quality and help service providers improve their offerings.

.

CHAPTER 8
BIBLIOGRAPHY

REFERENCES:

- **Smith, J.**, *Advanced Django Web Development*. Springer Publishing, 2022.
- **Brown, L. & Miller, D.**, *Machine Learning for Predictive Analytics*. Pearson Education, 2021.
- **Jones, R.**, *Database Management with MySQL*. O'Reilly Media, 2023.
- **Baker, T.**, *User Interface Design with HTML, CSS, and JavaScript*. Wiley, 2020.
- **Green, S. & White, A.**, *Roadside Assistance Systems and Technologies*. Elsevier, 2022.

WEBSITES:

- www.onstar.com
- www.allstate.com/roadside-assistance.aspx:
- www.goodsamroadside.com
- www.aaa

CHAPTER 9
APPENDIX

9.1 Sample Code

user_dashboard.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>User Dashboard - Road Guardian</title>
        <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;600;700&display
=swap" rel="stylesheet">
        <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
  awesome/6.0.0-beta3/css/all.min.css">
    <style>
        :root {
            --primary-color: #4CAF50;
            --secondary-color: #45a049;
            --background-color: #ffffff;
            --text-color: #333333;
            --shadow-color: rgba(0,0,0,0.1);
        }
        * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }
        body, html {
            font-family: 'Poppins', sans-serif;
            line-height: 1.6;
            color: var(--text-color);
      height: 100%;
      overflow-x: hidden;
```

```
        background-color: var(--background-color);
    }
    header {
        background-color: #ffffff;
        padding: 1rem 5%;
        display: flex;
        justify-content: space-between;
        align-items: center;
        position: fixed;
        top: 0;
        left: 0;
        right: 0;
        z-index: 1000;
        box-shadow: 0 2px 10px var(--shadow-color);
      }
     header h1 {
        color: var(--primary-color);
        font-size: 1.8rem;
     }
     .user-info {
        display: flex;
        align-items: center;
      }
      .user-name {
        margin-right: 1rem;
        color: var(--text-color);
      }
      .user-menu {
          position: relative;
      }
        .user-menu-btn {
            background: none;
                    border: none;
                    color: var(--text-color);
                    font-size: 1.2rem;
                    cursor: pointer;
            }
            .user-dropdown {
                    position: absolute;
                    right: 0;
                    top: 100%;
                    background-color: #ffffff;
                    box-shadow: 0 2px 10px var(--shadow-color);
                    border-radius: 5px;
                    display: none;
                    width: 1100%;
            }
            .user-dropdown a {
                    display: block;
                    padding: 10px 20px;
                    color: var(--text-color);
                    text-decoration: none;
                    transition: background-color 0.3s;
            }
            .user-dropdown a:hover {
                    background-color: #f0f0f0;
            }
            .dashboard-container {
                    display: flex;
                    margin-top: 80px;
                    min-height: calc(100vh - 80px);
            }
            .sidebar {
                    width: 250px;
                    background-color: #f0f0f0;
                    padding: 2rem;
                    box-shadow: 2px 0 10px var(--shadow-color);
            }
```

```
          .sidebar button {
               width: 100%;
               padding: 12px;
               margin-bottom: 1rem;
               font-size: 1rem;
               border: none;
               background-color: #ffffff;
               color: var(--text-color);
               cursor: pointer;
               transition: all 0.3s;
               border-radius: 5px;
               text-align: left;
          }
          .sidebar button:hover {
               background-color: var(--primary-color);
               color: #ffffff;
               transform: translateY(-2px);
               box-shadow: 0 4px 6px rgba(76,175,80,0.2);
          }
          .content {
               flex-grow: 1;
               background-color: #ffffff;
               border-radius: 10px;
               box-shadow: 0 4px 6px var(--shadow-color);
               padding: 2rem;
               margin: 1rem;
          }
          .content h2 {
               color: var(--primary-color);
               margin-bottom: 1.5rem;
               font-size: 2rem;
               border-bottom: 2px solid var(--primary-color);
     padding-bottom: 0.5rem;
}
```

```
.update-profile, .view-profile {
        background-color: #f9f9f9;
        padding: 2rem;
        border-radius: 10px;
        box-shadow: 0 0 15px var(--shadow-color);
        display: none;
}
.update-profile h3, .view-profile h3 {
        color: var(--primary-color);
        margin-bottom: 1.5rem;
        font-size: 1.5rem;
}
form label {
        display: block;
        margin-bottom: 0.5rem;
        color: var(--text-color);
}
form input, form select {
        width: 100%;
        padding: 0.8rem;
        margin-bottom: 1rem;
        border: 1px solid #ddd;
        background-color: #ffffff;
        color: var(--text-color);
        border-radius: 5px;
}
form button {
        background-color: var(--primary-color);
        color: #fff;
        padding: 0.8rem 1.5rem;
        border: none;
        border-radius: 5px;
                cursor: pointer;
                transition: background-color 0.3s;
}
form button:hover {
        background-color: var(--secondary-color);
}
.error-message {
        color: #f44336;
        font-size: 0.9rem;
        margin-top: 0.25rem;
}
.alert {
        padding: 1rem;
        margin-bottom: 1rem;
        border-radius: 5px;
        background-color: #d4edda;
        color: #155724;
}
.alert-success {
        background-color: #d4edda;
        color: #155724;
}
footer {
        background-color: #f0f0f0;
        color: var(--text-color);
        text-align: center;
        padding: 2rem 0;
        margin-top: 2rem;
}
.footer-content {
        display: flex;
        justify-content: space-around;
        max-width: 1200px;
        margin: 0 auto;
                padding: 0 2rem;
            }
```

```
.footer-section {
        text-align: left;
}
.footer-section h4 {
        color: var(--primary-color);
        margin-bottom: 1rem;
}
.footer-section a {
        color: var(--text-color);
        text-decoration: none;
        display: block;
        margin-bottom: 0.5rem;
        transition: color 0.3s;
}
.footer-section a:hover {
        color: var(--primary-color);
}
@media (max-width: 768px) {
        .dashboard-container {
                flex-direction: column;
        }
        .sidebar {
                width: 100%;
                margin-bottom: 1rem;
        }
        .content {
                margin: 0;
        }
        .footer-content {
                flex-direction: column;
        }
        .footer-section {
    margin-bottom: 1.5rem;
    }
```

```
            }
        </style>
    </head>
    <body>
        <header>
            <h1 style="font-family: Cambria, Cochin, Georgia, Times, 'Times New Roman',    serif;">ROAD GUARDIAN</h1>
            <div class="user-info">
                <span class="user-name">{{ user.name }}</span>
                <div class="user-menu">
                    <button class="user-menu-btn"><i class="fas fa-user"></i></button>
                    <div class="user-dropdown">
                        <a href="#" onclick="toggleViewProfile()"><i class="fas fa-user"></i> View Profile</a>
                        <a href="#" onclick="toggleUpdateProfile()"><i class="fas fa-edit"></i> Update Profile</a>
                        <form action="{% url 'logout' %}" method="post" style="display:    inline;">
                            {% csrf_token %}
                            <a href="#" onclick="this.parentNode.submit()"><i class="fas fa-sign-out-alt"></i> Log Out</a>
                        </form>
                    </div>
                </div>
            </div>
        </header>

        <div class="dashboard-container">
            <div class="sidebar">

                <button onclick=window.location.href='{% url 'request_assistance' %}'><i class="fas fa-hands-helping"></i> Request Assistance</button>
              <button><i class="fas fa-history"></i> View Payment History</button>
              <button onclick=window.location.href='{% url 'submit_feedback' %}'><i
```

```
class="fas fa-comment-alt"></i> Send Feedback</button>
                <button><i class="fas fa-comments"></i> Chat with Service Providers</button>
            </div>

            <div class="content">
                <h2 style="font-family: Cambria, Cochin, Georgia, Times, 'Times New Roman',
    serif;">Welcome, {{ user.name }}!</h2>
                {% if messages %}
                    {% for message in messages %}
                        <div class="alert alert-success">{{ message }}</div>
                    {% endfor %}
                {% endif %}

                <div class="update-profile" id="updateProfileSection">
                    <h3>Update Your Profile</h3>
                    <form method="post" action="{% url 'user_update_profile' %}"
        id="profileForm">
                        {% csrf_token %}
                        {{ form.non_field_errors }}

                        <label for="name">Name:</label>
                        {{ form.name }}
                        <span id="nameError" class="error-message"></span>

                        <label for="username">Username:</label>
                        {{ form.username }}
                        <span id="usernameError" class="error-message"></span>

                        <label for="address">Address:</label>
                        {{ form.address }}
                        <span id="addressError" class="error-message"></span>

                        <label for="phone">Phone:</label>
                        {{ form.phone }}
                        <span id="phoneError" class="error-message"></span>

                        <label for="email" id="emailLabel">Email:</label>
                        <input type="email" name="email" id="emailField"
      value="{{ form.email.value }}" readonly>
                        <span id="emailTooltip" class="tooltip" style="display: none;">Can't
            change email due to security reasons</span>
                        <span id="emailError" class="error-message"></span>

                        <button type="submit">Save Changes</button>
                    </form>
                </div>

                <div class="view-profile" id="viewProfileSection">
                    <h3>Your Profile Information</h3>
                    <p><strong>Username:</strong> {{ user.username }}</p>
                    <p><strong>Name:</strong> {{ user.name }}</p>
                    <p><strong>Email:</strong> {{ user.email }}</p>
                    <p><strong>Phone:</strong> {{ user.phone }}</p>
                    <p><strong>Address:</strong> {{ user.address }}</p>
                    <p><strong>Role:</strong> {{ user.role }}</p>
                </div>
            </div>
            </div>

            <footer>
            <div class="footer-content">
                <div class="footer-section">
                    <h4>Quick Links</h4>
                    <a href="#">Home</a>
                    <a href="#">Services</a>
                    <a href="#">About Us</a>
                    <a href="#">Contact</a>
                </div>
                <div class="footer-section">
                    <h4>Contact Us</h4>
                    <p>Email: road.guardian08@gmail.com</p>
                    <p>Phone: +1 234 567 890</p>
                </div>
                <div class="footer-section">
                    <h4>Follow Us</h4>
                    <a href="#"><i class="fab fa-facebook"></i> Facebook</a>
                    <a href="#"><i class="fab fa-twitter"></i> Twitter</a>
                    <a href="#"><i class="fab fa-instagram"></i> Instagram</a>
                    <a href="#"><i class="fab fa-linkedin"></i> LinkedIn</a>
                </div>
            </div>
            <p>&copy; 2024 Road Guardian. All rights reserved.</p>
            </footer>

            <script>
            document.addEventListener('DOMContentLoaded', function () {
                const successMessages = document.querySelectorAll('.alert-success');
                successMessages.forEach(function(message) {
                    setTimeout(function () {
                        message.classList.add('fade-out');
                        setTimeout(function () {
                            message.style.display = 'none';
                        }, 1000);
                    }, 3000);
                });

                // User dropdown menu
                const userMenuBtn = document.querySelector('.user-menu-btn');
                const userDropdown = document.querySelector('.user-dropdown');
                userMenuBtn.addEventListener('click', function(event) {
                    event.stopPropagation();
                    userDropdown.style.display = userDropdown.style.display === 'block' ?
```

```
'none' : 'block';
            });

            // Close dropdown when clicking outside
            document.addEventListener('click', function(event) {
                if (!userMenuBtn.contains(event.target)
  && !userDropdown.contains(event.target)) {
                    userDropdown.style.display = 'none';
                }
            });

            // Email tooltip
            const emailField = document.getElementById('emailField');
            const emailTooltip = document.getElementById('emailTooltip');
            emailField.addEventListener('focus', function() {
                emailTooltip.style.display = 'block';
            });
            emailField.addEventListener('blur', function() {
                emailTooltip.style.display = 'none';
            });

            // Form validation
            const form = document.getElementById('profileForm');
            const name = document.getElementById('id_name');
            const username = document.getElementById('id_username');
            const address = document.getElementById('id_address');
            const phone = document.getElementById('id_phone');
            const email = document.getElementById('id_email');

            const nameError = document.getElementById('nameError');
            const usernameError = document.getElementById('usernameError');
            const addressError = document.getElementById('addressError');
            const phoneError = document.getElementById('phoneError');
            const emailError = document.getElementById('emailError');
```

```
function validateName() {
    const nameRegex = /^[A-Z][a-zA-Z\s]*$/;
    if (!nameRegex.test(name.value)) {
        nameError.textContent = "Name must start with a capital letter and
contain only alphabets.";
        return false;
    }
    nameError.textContent = "";
    return true;
}

function validateAddress() {
    const addressRegex = /^[a-zA-Z\s]*$/;
    if (!addressRegex.test(address.value)) {
        addressError.textContent = "Address must contain only alphabets.";
        return false;
    }
    addressError.textContent = "";
    return true;
}

function validatePhone() {
    const phoneRegex = /^\d{10}$/;
    const repeatedNumbersRegex = /(.)\1{5,}/;
    if (!phoneRegex.test(phone.value) || repeatedNumbersRegex.test(phone.value))
{
        phoneError.textContent = "Phone must be 10 digits and not contain
continuous repeated numbers.";
        return false;
    }
    phoneError.textContent = "";
    return true;
}
```

```
        function validateEmail() {
            const emailRegex = /^[a-zA-Z0-9._%+-]+@gmail\.com$/;
            if (!emailRegex.test(email.value)) {
                emailError.textContent = "Email must be a valid @gmail.com address.";
                return false;
            }
            emailError.textContent = "";
            return true;
        }

        form.addEventListener('submit', function (event) {
            let valid = true;
            if (!validateName()) valid = false;
            if (!validateAddress()) valid = false;
            if (!validatePhone()) valid = false;
            if (!validateEmail()) valid = false;

            if (!valid) {
                event.preventDefault();
            }
        });

        name.addEventListener('input', validateName);
        address.addEventListener('input', validateAddress);
        phone.addEventListener('input', validatePhone);
        email.addEventListener('input', validateEmail);
    });

    function toggleUpdateProfile() {
        const updateSection = document.getElementById('updateProfileSection');
        const viewSection = document.getElementById('viewProfileSection');

        if (updateSection.style.display === 'none') {
            updateSection.style.display = 'block';
```