ZERELO

A PROJECT REPORT

Submitted by

SREENIVAS NITHIN [RA2211026010145]
K GURU CHARAN [RA2211026010141]
G SAI VARSHITH [RA2211026010151]

Under the Guidance of

Dr. S.KRISHNAVENI

Associate Professor
Department Of Computational Intelligence

in partial fulfillment of the requirementsfor the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING
with specialization in Artificial Intelligence And Machine Learning

DEPARTMENT OF COMPUTATIONAL INTELLIGENCE COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY

KATTANKULATHUR- 603 203

MAY 2025

Department of Computational Intelligence

SRM Institute of Science & Technology

Own Work Declaration Form

This sheet must be filled in (each box ticked to show that the condition has been met). It must be signed and dated along with your student registration number and included with all assignments you submit – work will not be marked unless this is done.

To be completed by the student for all assessments

Degree/ Course : B. TECH (CSE AIML)

Student Name : SREENIVAS NITHIN, K GURU CHARAN, G SAI VARSHITH

Registration Number : RA2211026010145, RA2211026010141, RA2211026010151

Title of Work : ZERELO

I / We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this assessment is my / our own except where indicated, and that I / We have met the following conditions:

- Clearly referenced / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

SRM INSTITUTE OF SCIENCE AND TECHNOLOGYKATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that 21CSP302L - Project report titled "**ZERELO**" is the bonafide work of "**SREENIVAS NITHIN [RA2211026010145], K GURU CHARAN[RA2211026010141], G SAI VARSHITH [RA2211026010151]"** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE	**SIGNATURE**
Dr. S.Krishnaveni	Dr. R.Annie Uthra
Supervisor	**Head of the Department**
Associate Professor	Professor
Department of Computational Intelligence	Department of Computational Intelligence
SRM Institute of Science and Technology	SRM Institute of Science and Technology
Kattankulathur	Kattankulathur

ACKNOWLEDGEMENTS

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to **Dr. Leenus Jesu Martin M,** Dean-CET, SRM Institute of Science and Technology, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor and Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We encompass our sincere thanks to, **Dr. M. Pushpalatha**, Professor and Associate Chairperson – CS, School of Computing and **Dr. Lakshmi,** Professor and Associate Chairperson -AI, School of Computing, SRM Institute of Science and Technology, for their invaluable support.

We are incredibly grateful to our Head of the Department, **Dr. R.Annie Uthra**, Professor, Computational Intelligence, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Project Coordinators, Panel Head, and Panel Members Department of Computational Intelligence, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor, **Dr.Prithi S**, Assistant Professor, Department of Computational Intelligence, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr.S.Krishnaveni**, Associate Professor, Department of Computational Intelligence, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under her mentorship. She provided us with the freedom and support to explore the research topics of our interest. Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank all the staff members of Computational Intelligence, School of Computing, S.R.M Institute of Science and Technology, for their help during our project. Finally, we would like to thank our parents, family members, and friends for their unconditional love, constant support and encouragement

Sreenivas Nithin

K Guru Charan

G Sai Varshith

ABSTRACT

Vehicle breakdowns can occur unexpectedly, often leaving drivers stranded in inconvenient or remote locations. Traditional service centers are not always within immediate reach, leading to delays, stress, and potential safety concerns. To address this challenge, **Zerelo** is developed as a smart vehicle breakdown assistance and home service platform. The system allows users to book on-site repair services or request emergency help directly through a user-friendly interface. By integrating real-time location tracking, service scheduling, and nearest mechanic dispatch features, Zerelo ensures timely support for both urban and remote area users. It enhances road safety, improves service accessibility, and reduces downtime through a reliable, technology-driven solution. This report outlines the system's design, architecture, and practical implementation, aiming to modernize roadside assistance and provide a seamless customer experience.

Zerelo operates through a hybrid model combining scheduled home services and on-demand roadside assistance. Users can access the platform via a mobile app or web portal to request routine vehicle maintenance at their doorstep or get immediate help during a breakdown. The system uses GPS to locate the user and automatically assigns the nearest available mechanic, reducing response time significantly. By leveraging cloud-based infrastructure and scalable APIs, Zerelo ensures smooth communication between users, service providers, and administrators. This project aims to bridge the gap between traditional garages and modern mobility needs, offering convenience, safety, and efficiency to vehicle owners.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

Abbreviation	Full Form
AI	Artificial Intelligence
UX	User Experience
UI	User Interface
DB	Database
API	Application Programming Interface
JWT	JSON Web Token
SQL	Structured Query Language
SRS	Software Requirements Specification
SDG	Sustainable Development Goals
CRUD	Create, Read, Update, Delete
SSL	Secure Sockets Layer
GDPR	General Data Protection Regulation
AWS	Amazon Web Services
CSV	Comma-Separated Values
ML	Machine Learning
KPI	Key Performance Indicator
ETL	Extract, Transform, Load
SaaS	Software as a Service
MVC	Model-View-Controller
OTP	One-Time Password
HTTPS	HyperText Transfer Protocol Secure
SMTP	Simple Mail Transfer Protocol
ROI	Return on Investment
CI/CD	Continuous Integration / Continuous Deployment
CDN	Content Delivery Network

CHAPTER 1

INTRODUCTION

1.1 Introduction to Project:

The rise in vehicle ownership has led to increased demand for reliable maintenance and emergency support services. Despite advancements in automotive technology, unexpected vehicle breakdowns continue to be a significant inconvenience for drivers, particularly in remote or rural areas where help is not readily available. Traditional service centers may not offer immediate assistance, and current roadside help services are often fragmented or limited in reach. Recognizing this gap, **Zerelo** is introduced as an intelligent, location-aware system that offers both home-based vehicle services and on-spot breakdown assistance through a digital platform.

Zerelo aims to deliver a seamless experience by connecting users with verified mechanics and service providers in real time, ensuring faster resolution and enhanced customer satisfaction. This platform is designed to be accessible, efficient, and scalable, making it a viable solution for urban dwellers and travelers in less accessible locations.

1.2 Motivation

- The idea behind **Zerelo** is driven by the everyday problems vehicle owners encounter, especially when they are caught off guard by unexpected breakdowns. Whether it's a flat tire on a highway, engine failure in a remote village, or a dead battery at home, the lack of quick and reliable service can turn a minor inconvenience into a serious issue. These incidents are not only frustrating but can also pose safety risks—particularly when they occur at night or in unfamiliar locations.
- Moreover, in many regions, roadside assistance services are either too costly, unorganized, or entirely unavailable. Traditional garages often do not offer emergency support, and drivers are left relying on informal, unverified mechanics, which can compromise service quality and vehicle safety. Even in urban settings, scheduling home servicing usually involves long wait times, lack of communication, and unclear pricing.

- Zerelo was envisioned as a direct response to these challenges. The motivation is to build a **technology-powered solution** that bridges the gap between vehicle owners and trustworthy service providers, eliminating uncertainty and delays. It seeks to empower users with control and peace of mind—knowing that reliable help is just a few clicks away, regardless of their location.
- Additionally, the platform aims to uplift local mechanics by giving them digital visibility and structured work opportunities, contributing to skill-based employment in the automotive service sector. By addressing both user-side pain points and service-side inefficiencies, **Zerelo** positions itself as a socially impactful and practically essential system in today's mobility ecosystem.

1.3 Sustainable Development Goal of the Project

Zerelo aligns with the following United Nations Sustainable Development Goals:

- **SDG 9: Industry, Innovation and Infrastructure** –

  Promoting innovative infrastructure to enhance mobility support systems.

- **SDG 11: Sustainable Cities and Communities** –

  Supporting smart urban transport services that are inclusive and safe.

- **SDG 8: Decent Work and Economic Growth** –

  Creating employment opportunities for local mechanics and service technicians through a digitally managed platform.

By contributing to these goals, Zerelo not only serves immediate user needs but also promotes long-term sustainability in transport and community development.

1.4 Product Vision Statement

The vision for **Zerelo** is to become a trusted, go-to digital platform for vehicle maintenance and emergency breakdown support that functions efficiently across urban and rural regions. The platform aspires to combine real-time location tracking, intelligent service allocation, and user-friendly interfaces to deliver fast, reliable, and transparent vehicle services.

Target Audience

- Individual vehicle owners (2-wheelers, 4-wheelers)
- Commercial vehicle operators and fleet managers
- Elderly drivers and families who need safer, at-home service options
- Commuters traveling long distances or through remote areas

User Needs

- Immediate support in the event of a vehicle breakdown, especially in isolated locations
- Reliable doorstep service for regular vehicle maintenance
- Access to trusted and verified service providers
- Real-time service updates and transparent pricing
- Easy-to-use mobile and web interfaces

Product Description

Zerelo is a service-oriented platform accessible via a mobile app and web interface. It enables users to:

- Schedule regular home servicing appointments
- Get real-time assistance during vehicle breakdowns
- Track service provider location and ETA using GPS
- Receive service history and digital billing records
- Give feedback and rate service providers

Mechanics and service partners can register on the platform, accept tasks based on location and availability, and manage their services and payments through an integrated dashboard.

Core Values and Benefits

- **Convenience**: On-demand help and maintenance scheduling from anywhere.
- **Speed**: Fast service allocation using location data and real-time availability.
- **Transparency**: Upfront service pricing and live tracking to build user trust.
- **Safety**: Verified mechanics and optional SOS alert system for added security.
- **Inclusivity**: Designed for use in both well-connected cities and remote towns.
- **Employment Generation**: Provides a digital platform for local mechanics to gain visibility and stable income.

Vision Statement

"To revolutionize vehicle service accessibility by delivering prompt, transparent, and location-aware breakdown and home maintenance support—anywhere, anytime."

1.5 Product Goal

1. Reliable Emergency Support
Develop a robust, GPS-enabled system that can quickly locate users during vehicle breakdowns and assign the nearest available mechanic or service provider. This ensures minimal waiting time and provides reassurance during stressful scenarios.
2. Home-Based Maintenance Scheduling
Enable users to schedule regular vehicle check-ups, servicing, and repairs at their convenience and location. The goal is to eliminate the need to visit service centers for routine issues, promoting time efficiency and customer comfort.
3. Smart Service Allocation
Implement intelligent service-matching algorithms that consider user location, mechanic availability, service type, and urgency to allocate the best possible resource dynamically and efficiently.
4. Transparent and User-Centric Experience

Provide clear service pricing, estimated time of arrival, real-time tracking, and post-service digital receipts to improve trust and accountability between users and service providers.

5. Verified and Trained Service Providers

Onboard skilled mechanics and technicians through a verification process and provide them with digital tools to manage service requests, track jobs, and receive customer feedback. This ensures quality control and professional service delivery.

6. Service Accessibility in Remote Areas

Ensure the platform is functional even in areas with limited service coverage by using lightweight app design, offline alerts, and network fallback mechanisms to reach users in remote or low-signal zones.

7. Feedback and Continuous Improvement

Integrate a feedback and rating mechanism that allows users to review their experience and service providers. This data can be used to improve service standards and identify high-performing mechanics.

8. Scalable and Modular Architecture

Design the platform backend to be modular and cloud-ready, allowing it to scale easily with increased users, locations, and new service categories such as towing, battery jump-start, or insurance linkage.

9. Employment Generation and Mechanic Empowerment

Support local economies by creating work opportunities for freelance and garage-based mechanics. Empower them with a digital presence and steady client flow through Zerelo.

10. Sustainability and Impact

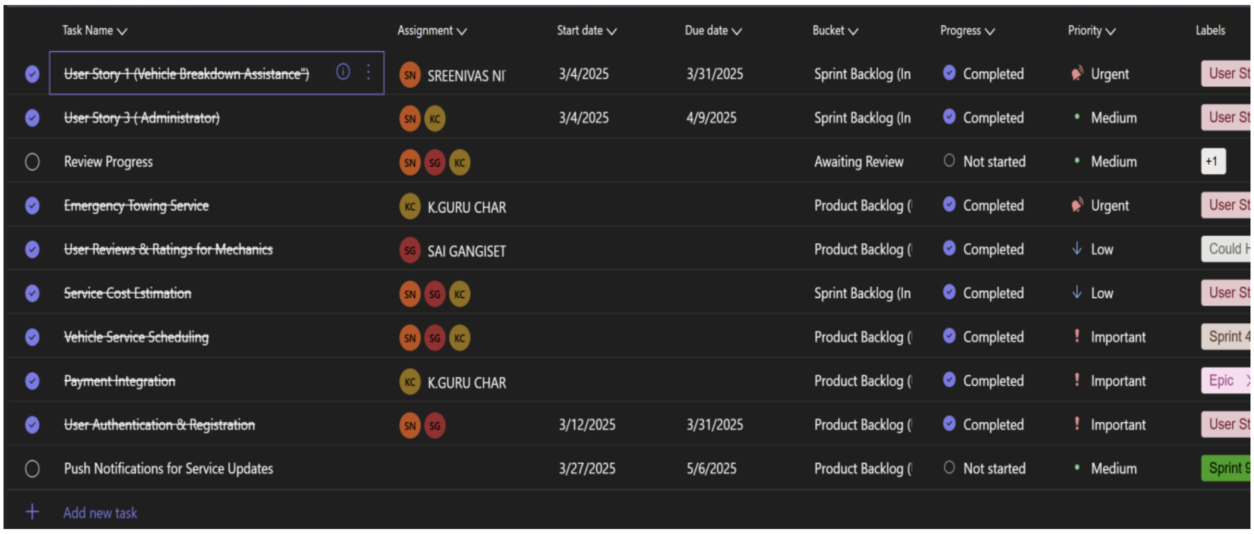
Contribute to reducing fuel consumption and environmental impact by minimizing unnecessary vehicle movement for servicing. Promote sustainable practices through digital receipts, scheduled maintenance reminders, and eco-friendly service options.

1.6 Product Backlog

Table 1.1 User Stories

S.NO.	USER STORIES OF DAILY EXPENSE TRACKER
1	As a user, I want to securely log in and register using my email or phone number.
2	As a driver, I want to request breakdown assistance and track the service provider in real time.
3	As a car owner, I want to schedule periodic maintenance and repairs at authorized service centers.
4	As a user, I want to see the location of my assigned mechanic on a map in real time.
5	As a stranded driver, I want to request a tow truck to take my vehicle to the nearest service station.
6	As a user, I want to rate and review mechanics and service centers based on my experience.
7	As a car owner, I want to receive an estimated cost for repairs and services before booking.
8	As a car owner, I want to manage multiple vehicles under one account to easily schedule services for each.
9	As a user, I want to receive real-time push notifications for service updates, reminders, and promotions.
10	As a car owner, I want an AI-powered tool to help diagnose basic vehicle issues based on symptoms I enter.

From Table 1.1,The product backlog for the Zerelo consists of a prioritized list of features and improvements, including expense data integration, implementing the ARIMA forecasting model, creating an interactive user dashboard, providing real time location, building notifications, and ensuring secure user authentication. This backlog is continuously refined based on user feedback to ensure the development aligns with the project's goals and delivers valuable functionality to users.



	Task Name	Assignment	Start date	Due date	Bucket	Progress	Priority	Labels
✓	~~User Story 1 (Vehicle Breakdown Assistance")~~	SN SREENIVAS NI	3/4/2025	3/31/2025	Sprint Backlog (In	Completed	Urgent	User St
✓	~~User Story 3 (Administrator)~~	SN KC	3/4/2025	4/9/2025	Sprint Backlog (In	Completed	Medium	User St
○	Review Progress	SN SG KC			Awaiting Review	Not started	Medium	+1
✓	~~Emergency Towing Service~~	KC K.GURU CHAR			Product Backlog (	Completed	Urgent	User St
✓	~~User Reviews & Ratings for Mechanics~~	SG SAI GANGISET			Product Backlog (	Completed	Low	Could H
✓	~~Service Cost Estimation~~	SN SG KC			Sprint Backlog (In	Completed	Low	User St
✓	~~Vehicle Service Scheduling~~	SN SG KC			Product Backlog (	Completed	Important	Sprint 4
✓	~~Payment Integration~~	KC K.GURU CHAR			Product Backlog (	Completed	Important	Epic
✓	~~User Authentication & Registration~~	SN SG	3/12/2025	3/31/2025	Product Backlog (	Completed	Important	User St
○	Push Notifications for Service Updates		3/27/2025	5/6/2025	Product Backlog (	Not started	Medium	Sprint 9

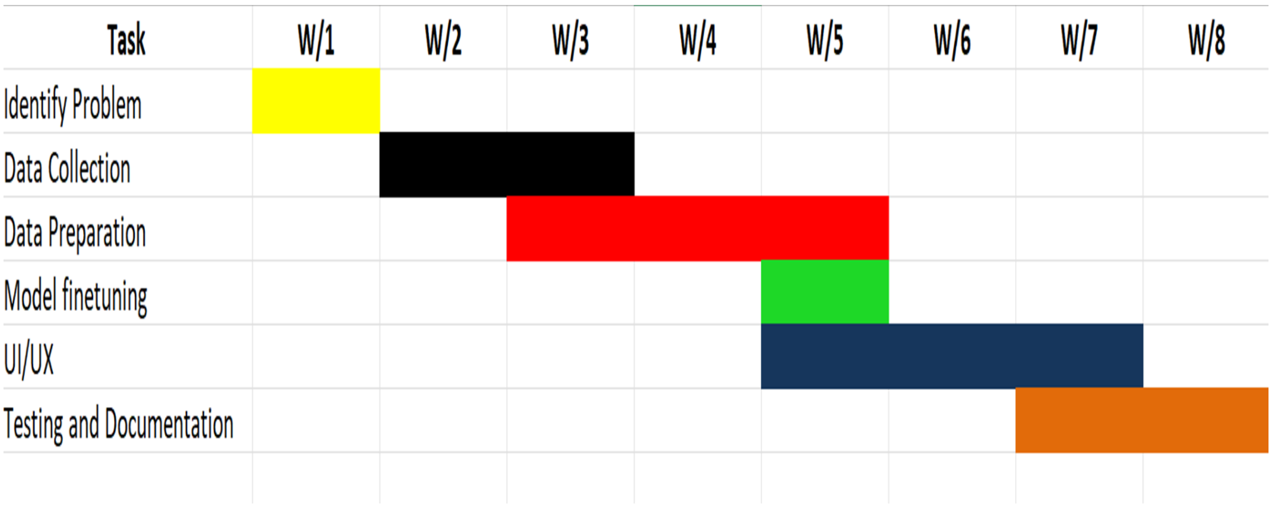
+ Add new task

Figure 1.1 MS Planner Board of daily expense tracker

Figure 1.1 shows the Sprint planning which we have done using MS planner

1.7 Product Release Plan

The following Figure 1.2 depicts the release plan of the project



Task	W/1	W/2	W/3	W/4	W/5	W/6	W/7	W/8
Identify Problem	🟨							
Data Collection		⬛	⬛					
Data Preparation			🟥	🟥	🟥			
Model finetuning					🟩			
UI/UX					🟦	🟦	🟦	
Testing and Documentation							🟧	🟧

Figure 1.2 Release plan Figure

1.2 represents the expected Time line of the entire Project. The Project as per figure 1.2 is expected to be completed in a span of 8 weeks

CHAPTER 2

SPRINT PLANNING AND EXECUTION

2.1 Sprint 1

2.1.1 Sprint Goal with User Stories of Sprint 1

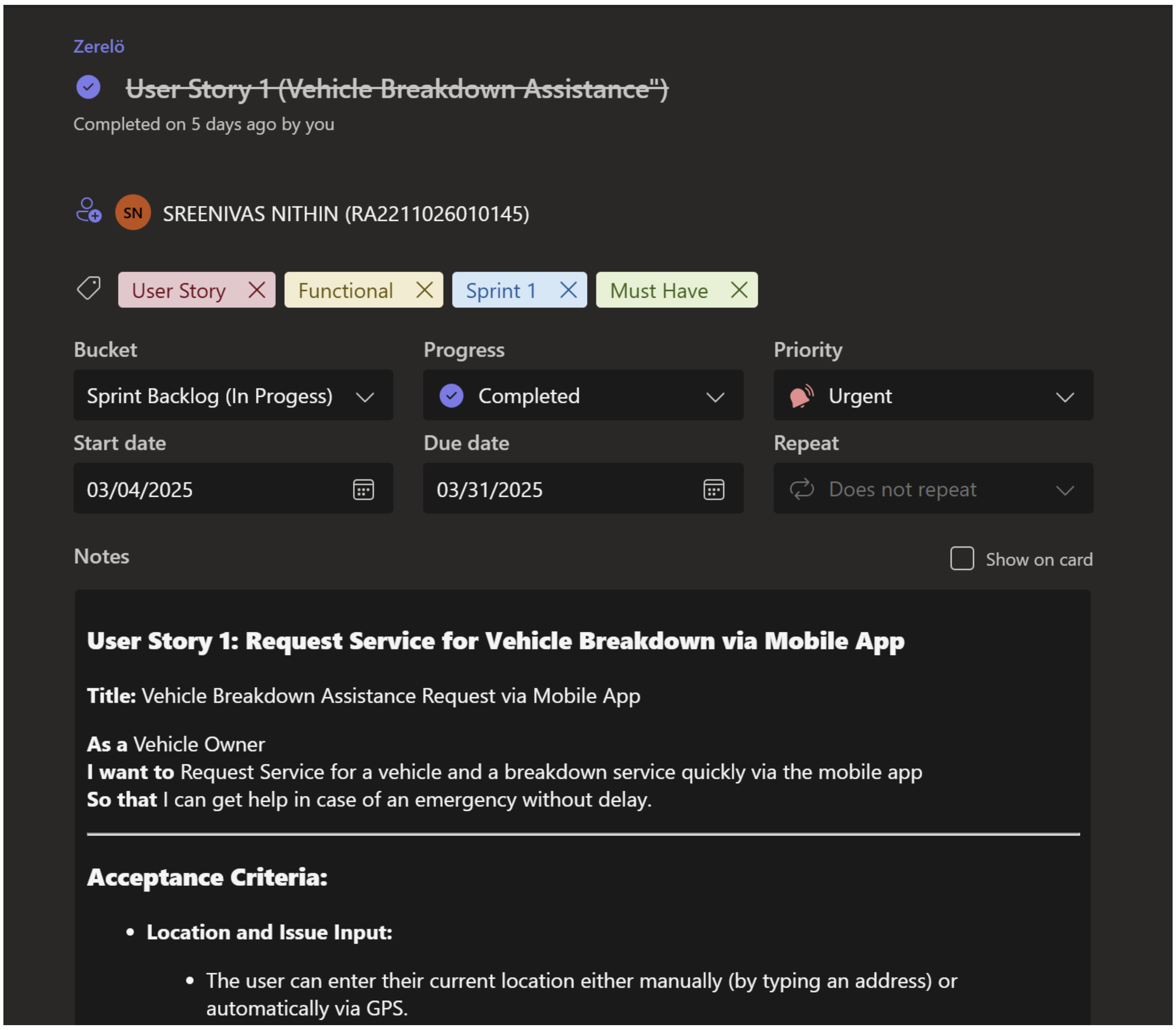
The goal of the first sprint is to lay the foundation for the "Daily Expense Tracker" application, focusing on user registration, profile creation, and setting up essential functionalities for expense management. The sprint will also include developing the user interface and essential features for adding and categorizing expenses.The following table 2.1 represents the detailed user stories of the sprint 1

Table 2.1 Detailed User Stories of sprint 1

S.NO	**Detailed User Stories**
US #1	As a user, I want to register and log in securely using email or phone so that I can access the app.
US #2	As a user, I want to update my profile and vehicle details so that service providers know my needs.
US #3	As a user, I want to request instant breakdown assistance using GPS so that I get help on the spot.
US #4	As a developer, I want to integrate Google Maps API so that users can share accurate locations.
US #5	As a developer, I want to secure user data with encryption and safe login mechanisms.

Table 2.1 represents the user stories which are considered for sprint 1.It Consists of 5 user stories which mainly focused on user registration and management.

Planner Board representation of user stories are mentioned below figures 2.1,2.2 and 2.3



Zerelö

~~User Story 1 (Vehicle Breakdown Assistance")~~

Completed on 5 days ago by you

SN SREENIVAS NITHIN (RA2211026010145)

User Story ✕ Functional ✕ Sprint 1 ✕ Must Have ✕

Bucket	Progress	Priority
Sprint Backlog (In Progess)	Completed	Urgent
Start date	**Due date**	**Repeat**
03/04/2025	03/31/2025	Does not repeat

Notes ☐ Show on card

User Story 1: Request Service for Vehicle Breakdown via Mobile App

Title: Vehicle Breakdown Assistance Request via Mobile App

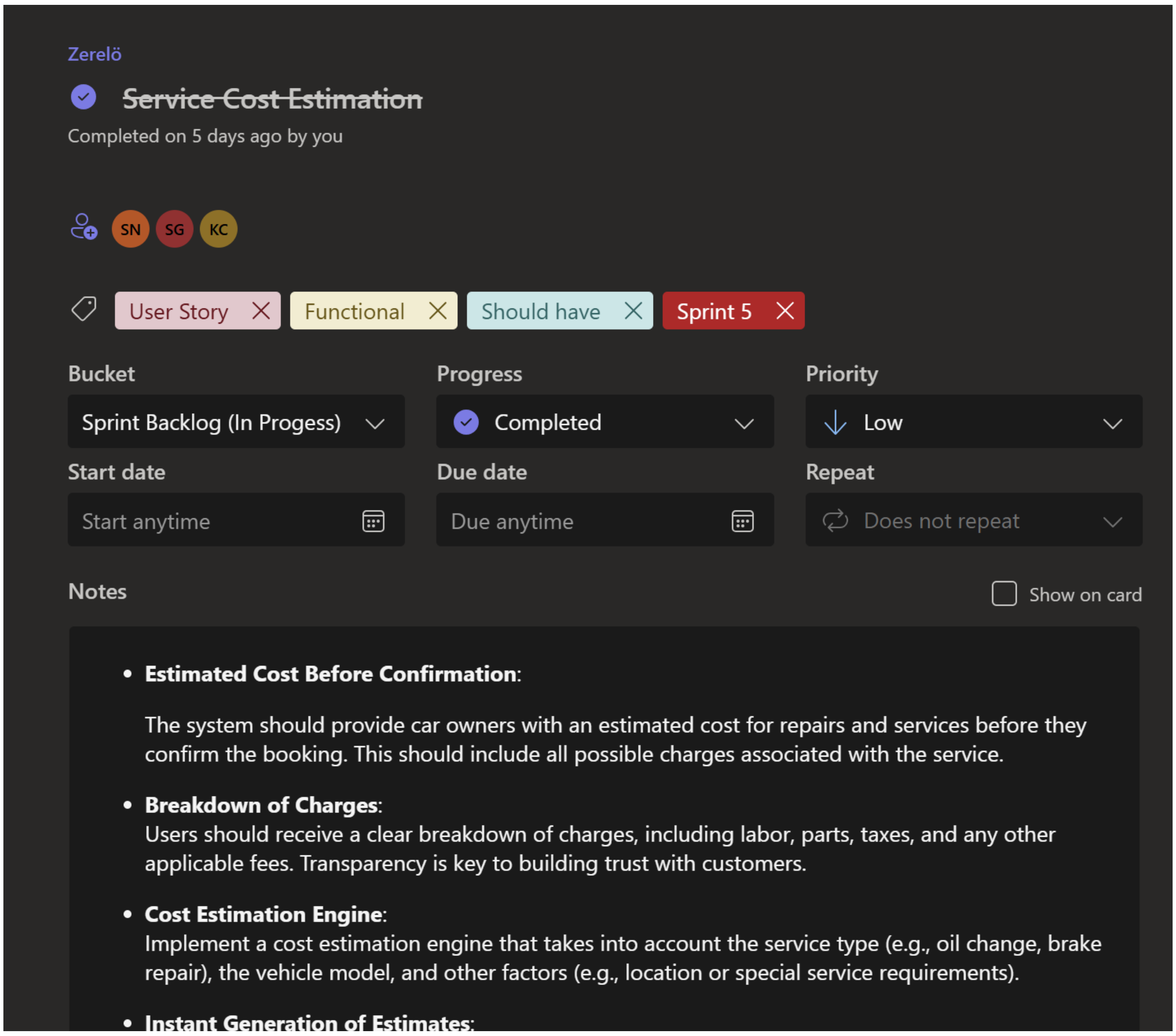
As a Vehicle Owner
I want to Request Service for a vehicle and a breakdown service quickly via the mobile app
So that I can get help in case of an emergency without delay.

Acceptance Criteria:

- **Location and Issue Input:**
  - The user can enter their current location either manually (by typing an address) or automatically via GPS.

Figure 2.1 bucket for user registration

Figure 2.1 represents the Bucket For user registration which have sub tasks of authentication and profile management.


Zerelö

~~Service Cost Estimation~~

Completed on 5 days ago by you

SN SG KC

User Story Functional Should have Sprint 5

Bucket: Sprint Backlog (In Progess)

Progress: Completed

Priority: Low

Start date: Start anytime

Due date: Due anytime

Repeat: Does not repeat

Notes

Show on card

- **Estimated Cost Before Confirmation**:

  The system should provide car owners with an estimated cost for repairs and services before they confirm the booking. This should include all possible charges associated with the service.

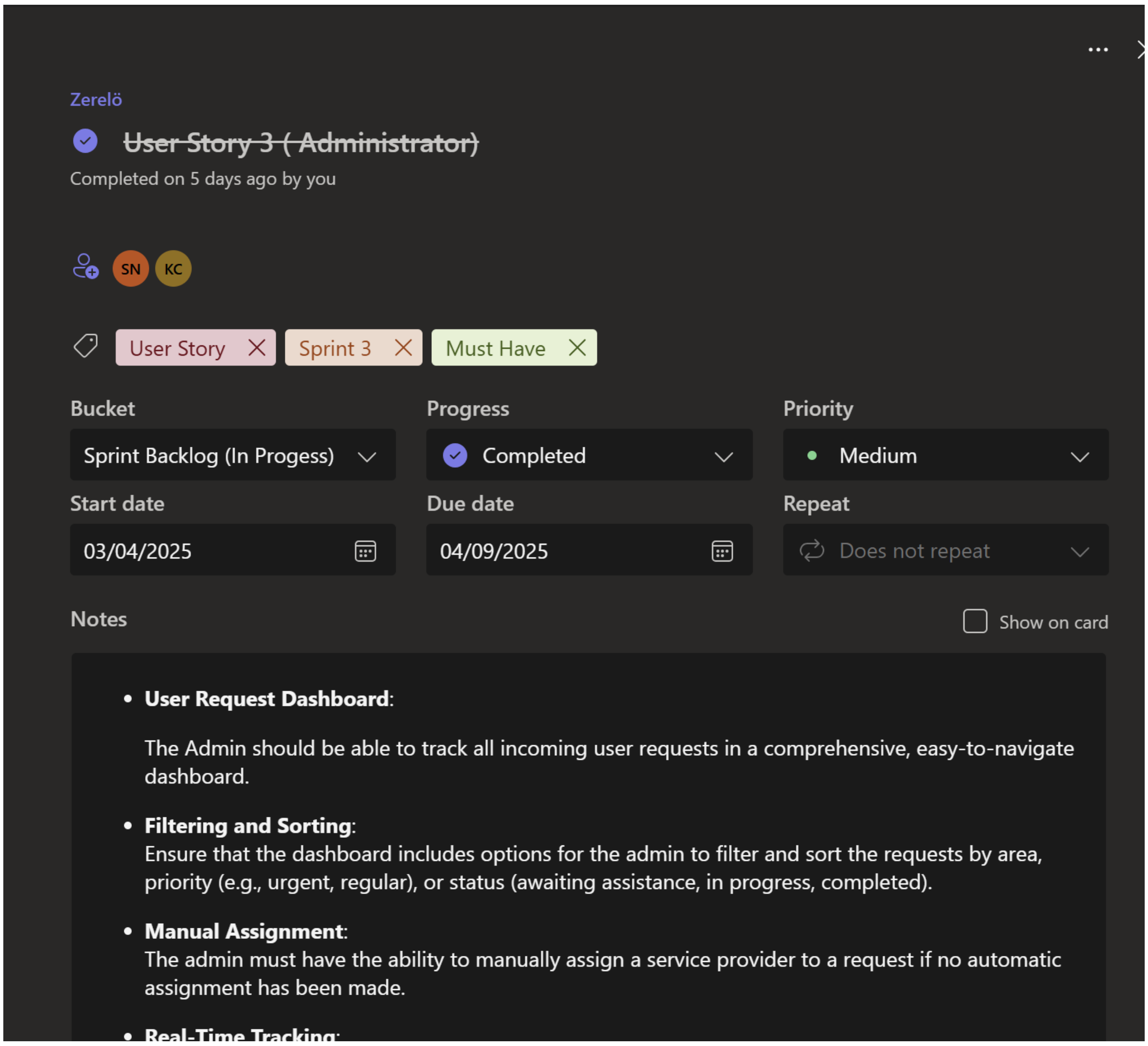
- **Breakdown of Charges**:
  Users should receive a clear breakdown of charges, including labor, parts, taxes, and any other applicable fees. Transparency is key to building trust with customers.

- **Cost Estimation Engine**:
  Implement a cost estimation engine that takes into account the service type (e.g., oil change, brake repair), the vehicle model, and other factors (e.g., location or special service requirements).

- **Instant Generation of Estimates**:

Figure 2.2 Service cost estimation

Figure 2.2 depicts the bucket created in MS planner to manage the expense logging process.


Zerelö

~~User Story 3 (Administrator)~~

Completed on 5 days ago by you

User Story | Sprint 3 | Must Have

Bucket	Progress	Priority
Sprint Backlog (In Progess)	Completed	Medium
Start date	Due date	Repeat
03/04/2025	04/09/2025	Does not repeat

Notes

Show on card

- **User Request Dashboard**:

  The Admin should be able to track all incoming user requests in a comprehensive, easy-to-navigate dashboard.
- **Filtering and Sorting**:
  Ensure that the dashboard includes options for the admin to filter and sort the requests by area, priority (e.g., urgent, regular), or status (awaiting assistance, in progress, completed).
- **Manual Assignment**:
  The admin must have the ability to manually assign a service provider to a request if no automatic assignment has been made.
- **Real-Time Tracking**:

Figure 2.3 User request

Figure 2.3 represents the Bucket for User request

2.1.2 Functional Document

2.1.2.1 Introduction

Zerelo is a smart vehicle service management platform designed to streamline service bookings, manage breakdown assistance, and provide real-time vehicle support to users. The first sprint focuses on delivering core features like user authentication, slot booking, and live breakdown tracking for a seamless service experience.

2.1.2.2 Product goal

The primary goal of this sprint is to enable users to book service slots efficiently, request instant breakdown support, and track service status, aiming to provide a reliable and responsive digital vehicle service experience.

2.1.2.3 Demography (users, location)

Users

Target Users: Vehicle owners, mechanics, service center admins

User Characteristics: Tech-savvy users looking for on-demand service options, predominantly aged between 25-55, familiar with mobile apps, often owning two-wheelers or four-wheelers.

Locations

Target Locations: Initially focused on urban Indian cities with planned scalability to tier-2 towns

Service Coverage: Metro cities (e.g., Delhi, Mumbai, Bengaluru), with future expansion to semi-urban areas where access to real-time breakdown assistance is limited.

Deployment: Android and web platforms primarily, with iOS support in later sprints

2.1.2.4 BUSINESS PROCESSES

The key business processes for **Zerelo** are as follows:

- **User Registration and Authentication**:
  - Users can register on the platform through email, phone number, or social media accounts.
  - Authentication ensures secure access to the platform, protecting sensitive user data like location, service history, and payment details.
- **Breakdown Assistance Request**:

    - Users can request breakdown assistance through GPS-based location services.
    - Requests are automatically routed to the nearest available mechanic based on location and type of service required.
- **Home Service Scheduling**:
    - Users can schedule home-based vehicle servicing (e.g., oil change, tire replacement) through the app.
    - Scheduling includes time selection, service type, and preferred mechanic (if applicable).
- **Service Payment and Invoicing**:
    - Users can pay for services either before or after service completion via secure payment gateways.
    - Invoices are generated and shared with users electronically, detailing the service performed and the cost.
- **Mechanic Feedback and Rating**:
    - After the service is completed, users can rate and leave feedback for mechanics to ensure service quality and transparency.

his project focuses on implementing the following key features:

Feature 1: User Registration and Profile Creation

1. **Description**:
   The platform allows users to securely register and create a personalized profile for managing breakdown requests, service history, and payment preferences.
2. **User Story**:
   As a user, I want to register on the platform and create my profile so I can request assistance and manage my vehicle services efficiently.

Feature 2: Breakdown Assistance Request

1. **Description**:
   Users can request immediate breakdown assistance via GPS, specifying the nature of the breakdown and their location.

2. **User Story**:
   As a user, I want to request immediate breakdown assistance, providing my current location to get help as quickly as possible.

Feature 3: Home Service Scheduling

1. **Description**:
   Users can schedule home-based maintenance and servicing for their vehicles, selecting the time, service type, and mechanic preference.
2. **User Story**:
   As a user, I want to schedule a home-based service for my vehicle at my convenience so I can avoid going to a service center.

Feature 4: Mechanic Matching and Assignment

1. **Description**:
   The platform intelligently matches users with the most appropriate mechanics based on service type, location, and availability.
2. **User Story**:
   As a user, I want the system to automatically assign a mechanic near me who is capable of providing the service I need.

Feature 5: Service Payment and Invoice Generation

1. **Description**:
   Users can make payments through secure channels, and digital invoices are generated and shared after service completion.
2. **User Story**:
   As a user, I want to make payments for services securely and receive an invoice for my records.

Feature 6: Mechanic Feedback and Rating

1. **Description**:
   After each service, users can rate the mechanic and provide feedback to improve the quality of service and maintain transparency.

2. **User Story**:
   As a user, I want to leave feedback and rate the mechanic's performance to help improve future services.

2.1.2.6 Authorization matrix

Table 2.2: Access Level Authorization Matrix

Role	Access Level
Administrator	Full access to all platform features, including user management, service management, mechanic onboarding, and platform settings.
User	Access to request breakdown assistance, schedule home services, make payments, view service history, and provide feedback/rating.
Guest User	Limited access to view basic platform information (e.g., types of services offered, FAQs) without interacting with the service directly.
Mechanic	Limited access to view basic platform information (e.g., types of services offered, FAQs) without interacting with the service directly.

Table 2.2 Represents the access level authorization matrix for the roles of administrator, User, Guest User, Mechanic.

2.1.2.7 Assumptions

 Data Availability:
User data, such as service requests, feedback, and payment history, will be reliably stored and available for analysis to improve platform performance.

 Cloud Infrastructure:
The platform will utilize cloud-based servers for scalable and efficient data storage, ensuring high availability and minimal downtime.

 User Feedback:
Users will provide valuable feedback through ratings, reviews, and service experiences, which will be used to refine and improve the platform.

 Regulatory Compliance:
The platform will adhere to relevant data protection laws (e.g., GDPR, CCPA) to ensure that users' personal and financial data is handled securely.

 Service Availability:
The platform assumes the availability of skilled mechanics in various regions to provide prompt and efficient service to users, especially in remote or underserved areas.

2.1.3 Architecture Document

2.1.3.1 Application

Zerelo leverages Microservices Architecture to enable modular and independently deployable services. Future enhancements may include:

1. Microservices

o Decoupled services for authentication, booking, breakdown, and feedback.

o Services communicate via REST APIs.

2. Event-Driven (Planned)

o Events for booking confirmation, mechanic assignment, and service completion.

o Use of pub/sub or message queues in later sprints for asynchronous updates.

3. Serverless (Optional Future Consideration)

o Serverless functions for sending notifications, feedback collection, or payment processing during low-traffic hours.

2.1.3.2 System Architecture

1. ER Diagram

o ER diagram consists of entities like Users, Bookings, Mechanics, Service Centers, Breakdown Requests, and Payments.

2. Schema Design

o Users Table: id, name, contact, vehicle_info, login_credentials

o Bookings Table: booking_id, user_id, service_center_id, date, status

o Mechanics Table: mechanic_id, name, center_id, availability_status

o Breakdown Requests Table: request_id, user_id, location, time, assigned_mechanic_id

o Payments Table: payment_id, user_id, booking_id, amount, status

o Service Centers Table: center_id, name, location, operating_hours

3 Data Exchange Contract

1. Frequency of Data Exchanges

o Real-time for breakdown assignment, mechanic location updates, notifications.

o Periodic for summary reports, admin analytics, and usage logs.

2. Data Sets

o User Profiles and Preferences

o Booking and Breakdown History

o Mechanic Availability and Assignments

o Payments and Feedback Records

3. Mode of Exchanges

o API: REST APIs for frontend-backend communication

o File: Downloadable CSV reports for admin usage

o Queue: Kafka or RabbitMQ planned for handling notification delivery and assignment workflows

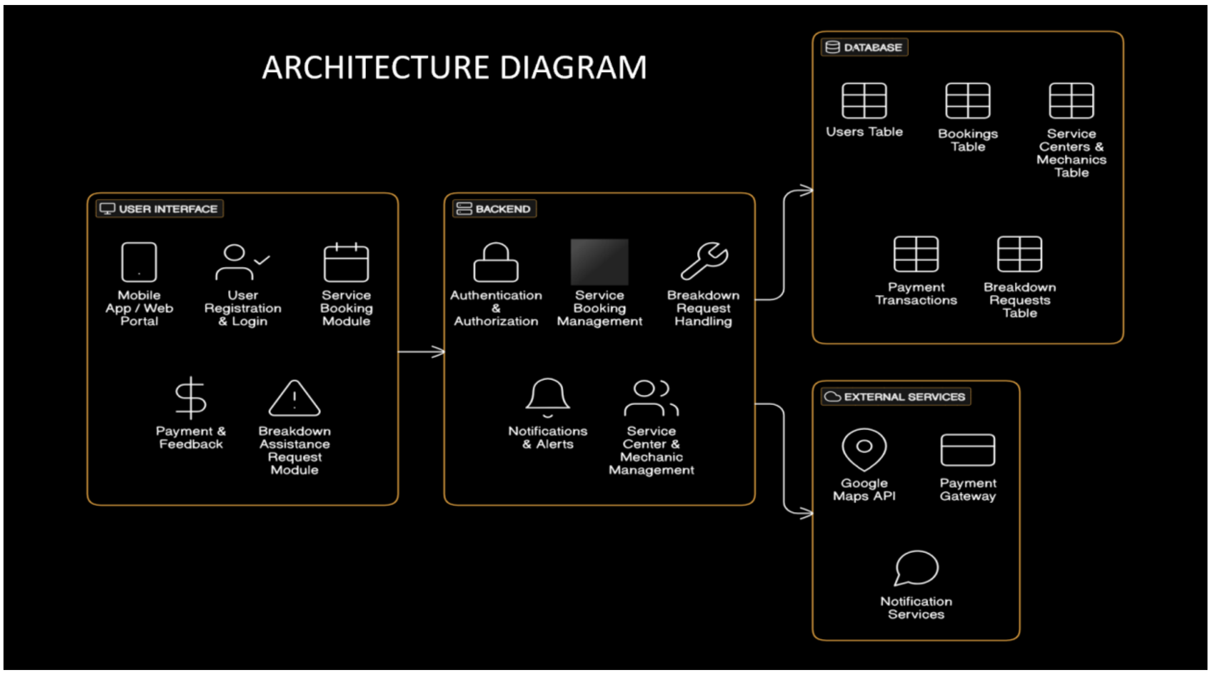


Figure 2.4: System Architecture Diagram

2.1.3.3 Data Exchange Contract

Frequency of Data Exchanges:

Data exchanges in Zerelo are structured to support **real-time responsiveness** for critical services (e.g., breakdown assistance) and **periodic synchronization** for reporting and analytics.

Real-Time Exchanges:

- User authentication and authorization
  → Occurs during login, sign-up, and session validation to ensure secure access.
- Breakdown and home service requests
  → Service requests are sent and processed instantly using geolocation and mechanic assignment logic.
- Mechanic location updates
  → GPS-based updates are exchanged in real time to track the mechanic's route and ETA.
- Notifications and alerts
  → Real-time notifications (e.g., mechanic assigned, service started, service completed) are pushed to users and mechanics.

Periodic Syncs:

- Service history and analytics
  → Service data is periodically synchronized for generating service history, reports, and performance metrics.
- Backup of user, mechanic, and service data
  → Periodic backups are performed to ensure data integrity and business continuity.

Data Sets:

Zerelo handles several critical data sets, each requiring specific exchange mechanisms:

- User Data
  → Includes registration info, contact details, vehicle details, login credentials, and preferences.
  → Exchanged during sign-up, login, and profile updates.
- Service Request Data
  → Contains information like service type (breakdown or home service), vehicle info, location, and request time.
  → Exchanged in real-time during service request and mechanic assignment.
- Mechanic Data
  → Includes mechanic profiles, skills, current status, and location.
  → Exchanged dynamically for assignment logic and location tracking.
- Feedback and Ratings
  → User-submitted ratings and comments about service quality.
  → Exchanged after each service session for quality assurance.
- Notification Data
  → Contains alerts for both users and mechanics (e.g., request accepted, delay alerts, payment reminders).
  → Triggered in real time by system events.

Mode of Exchanges

Different technologies are employed based on the data's frequency and nature:

- RESTful APIs
  → Used for real-time interactions between frontend and backend such as:
    - User registration, login
    - Requesting and tracking services
    - Fetching service history and ratings
- Message Queues (e.g., RabbitMQ, AWS SQS)
  → Used for asynchronous processes like:
    - Sending service notifications
    - Matching and dispatching mechanics
    - Background rating analysis and performance scoring

2.1.4 UI design

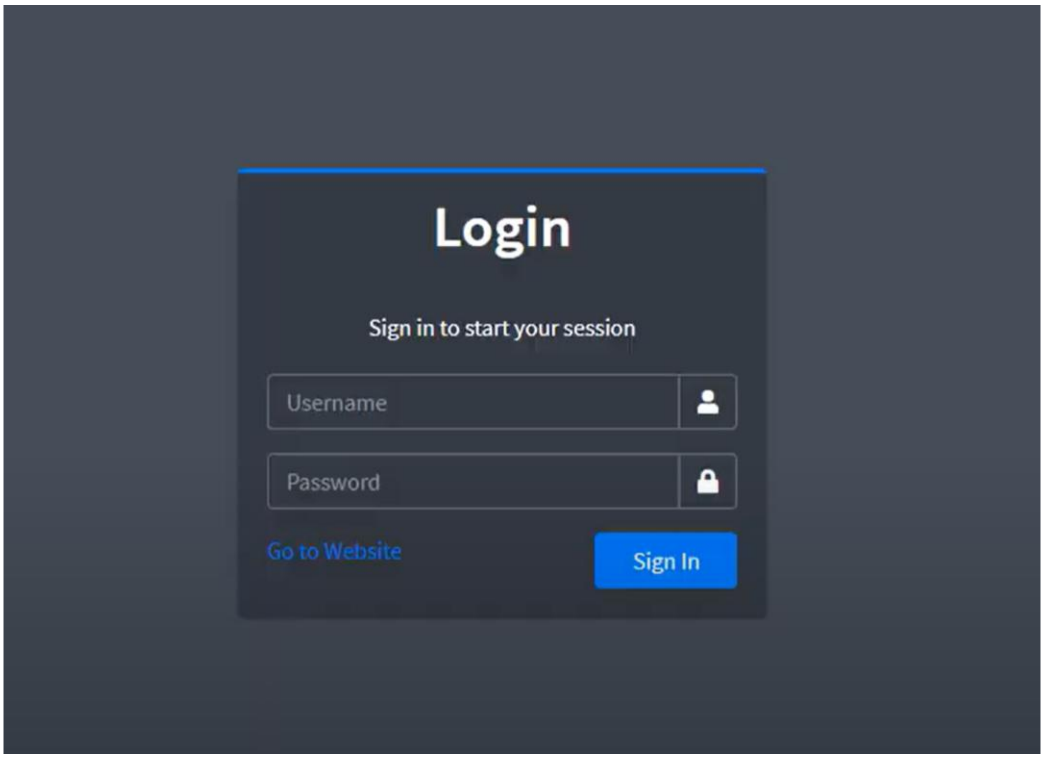


Figure 2.5 UI Design for Login page

Figure 2.5 represents the UI design of Login page. It contains the spaces to enter the correct username and password.

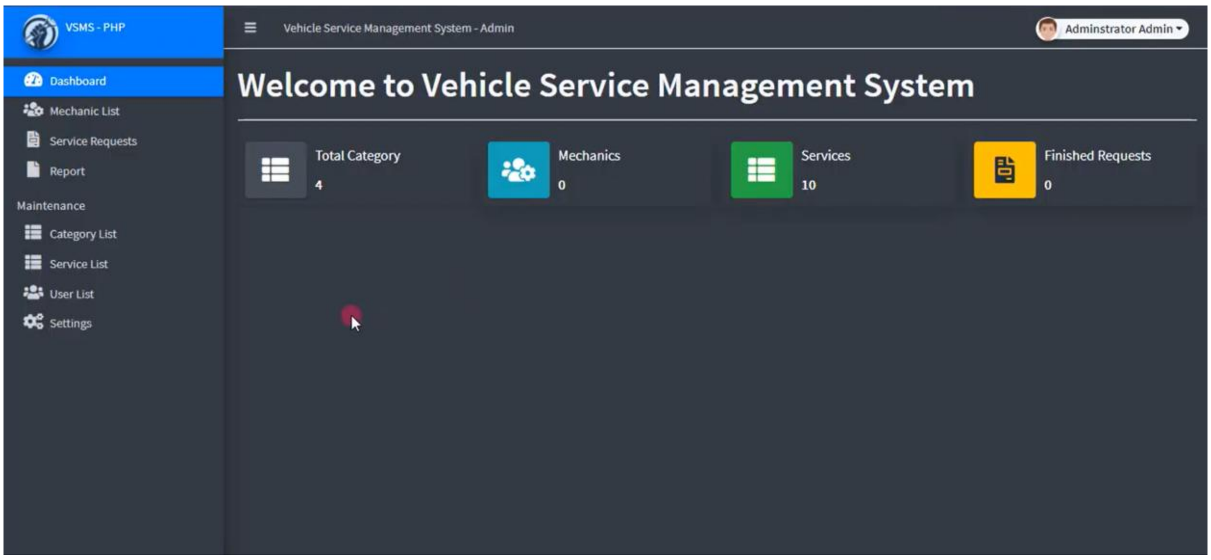


Figure 2.6 UI design for Admin Dashboard

Figure 2.6 represents the UI design for managing the expense logging .

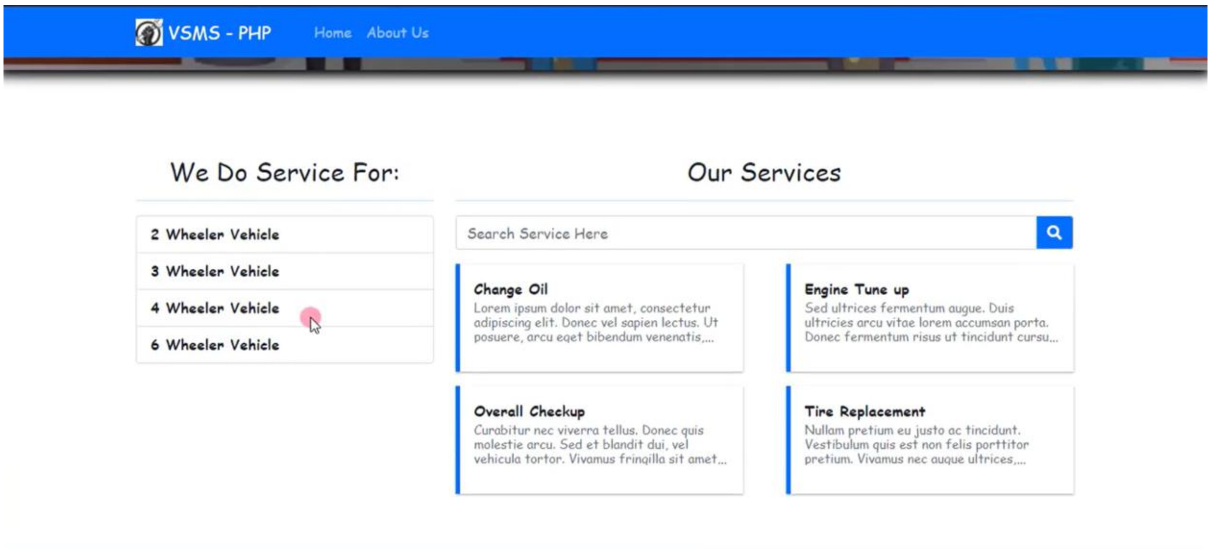
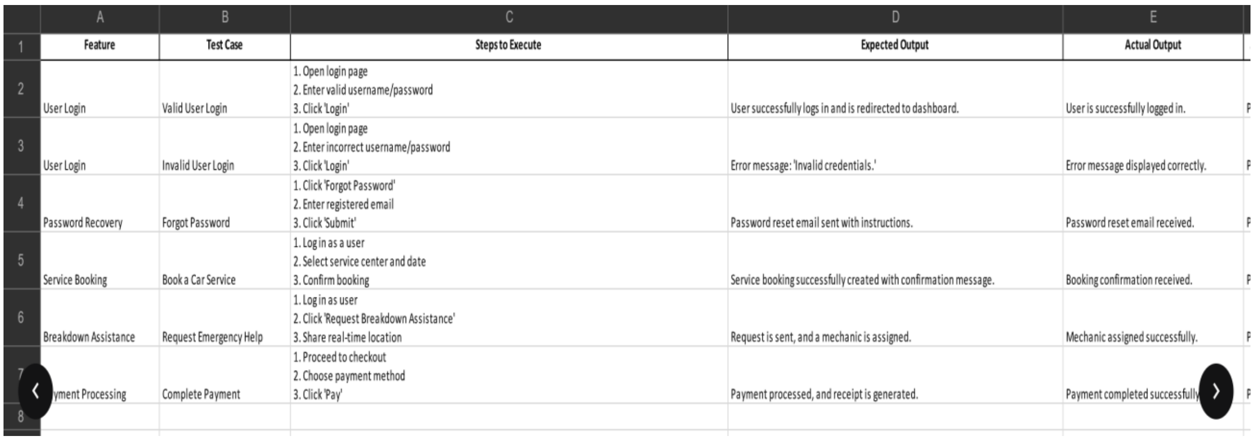

VSMS - PHP
Home
About Us
We Do Service For:
2 Wheeler Vehicle
3 Wheeler Vehicle
4 Wheeler Vehicle
6 Wheeler Vehicle
Our Services
Search Service Here
Change Oil
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec vel sapien lectus. Ut posuere, arcu eget bibendum venenatis,...
Engine Tune up
Sed ultrices fermentum augue. Duis ultricies arcu vitae lorem accumsan porta. Donec fermentum risus ut tincidunt cursu...
Overall Checkup
Curabitur nec viverra tellus. Donec quis molestie arcu. Sed et blandit dui, vel vehicula tortor. Vivamus fringilla sit amet...
Tire Replacement
Nullam pretium eu justo ac tincidunt. Vestibulum quis est non felis porttitor pretium. Vivamus nec augue ultrices,...

Figure 2.7 UI design for selecting preferance

Figure 2.7 represents the UI design to create a income tracking dashboard.It contains a graph tacking the income and separate sections for income logging over separate period of times.

2.1.5 Functional Test Cases

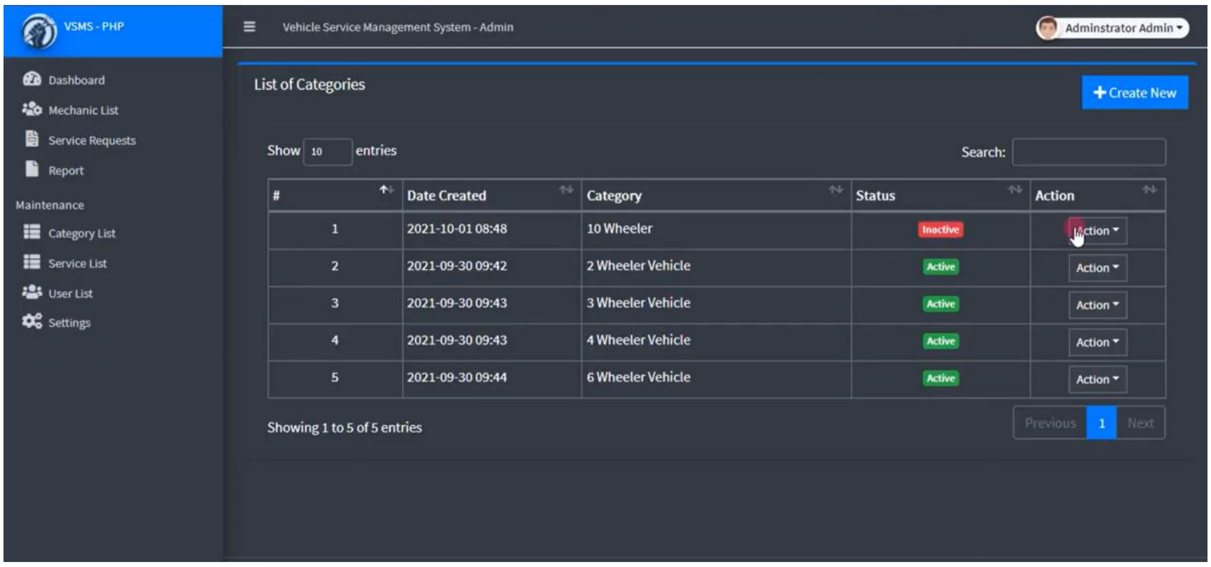
Table 2.3 Detailed Functional Test Case



	A	B	C	D	E	
1	Feature	Test Case	Steps to Execute	Expected Output	Actual Output	
2	User Login	Valid User Login	1. Open login page 2. Enter valid username/password 3. Click 'Login'	User successfully logs in and is redirected to dashboard.	User is successfully logged in.	P
3	User Login	Invalid User Login	1. Open login page 2. Enter incorrect username/password 3. Click 'Login'	Error message: 'Invalid credentials.'	Error message displayed correctly.	P
4	Password Recovery	Forgot Password	1. Click 'Forgot Password' 2. Enter registered email 3. Click 'Submit'	Password reset email sent with instructions.	Password reset email received.	P
5	Service Booking	Book a Car Service	1. Log in as a user 2. Select service center and date 3. Confirm booking	Service booking successfully created with confirmation message.	Booking confirmation received.	P
6	Breakdown Assistance	Request Emergency Help	1. Log in as user 2. Click 'Request Breakdown Assistance' 3. Share real-time location	Request is sent, and a mechanic is assigned.	Mechanic assigned successfully.	P
7	yment Processing	Complete Payment	1. Proceed to checkout 2. Choose payment method 3. Click 'Pay'	Payment processed, and receipt is generated.	Payment completed successfully	P
8						

Table 2.3 represents the Detailed Functional test Cases of the sprint 1.

2.1.6 Selecting category



List of Categories

Show 10 entries

Search:

#	Date Created	Category	Status	Action
1	2021-10-01 08:48	10 Wheeler	Inactive	Action
2	2021-09-30 09:42	2 Wheeler Vehicle	Active	Action
3	2021-09-30 09:43	3 Wheeler Vehicle	Active	Action
4	2021-09-30 09:43	4 Wheeler Vehicle	Active	Action
5	2021-09-30 09:44	6 Wheeler Vehicle	Active	Action

Showing 1 to 5 of 5 entries

Figure 2.8 Category

2.1.7 Committed Vs Completed User Stories

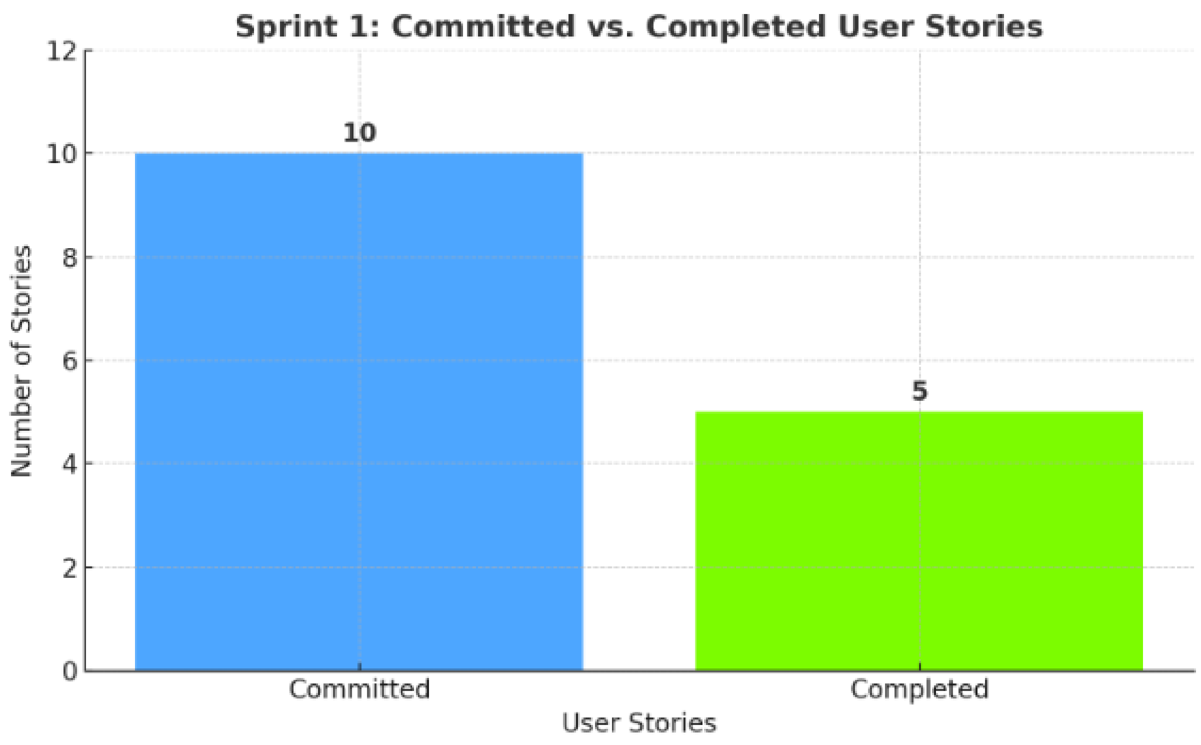
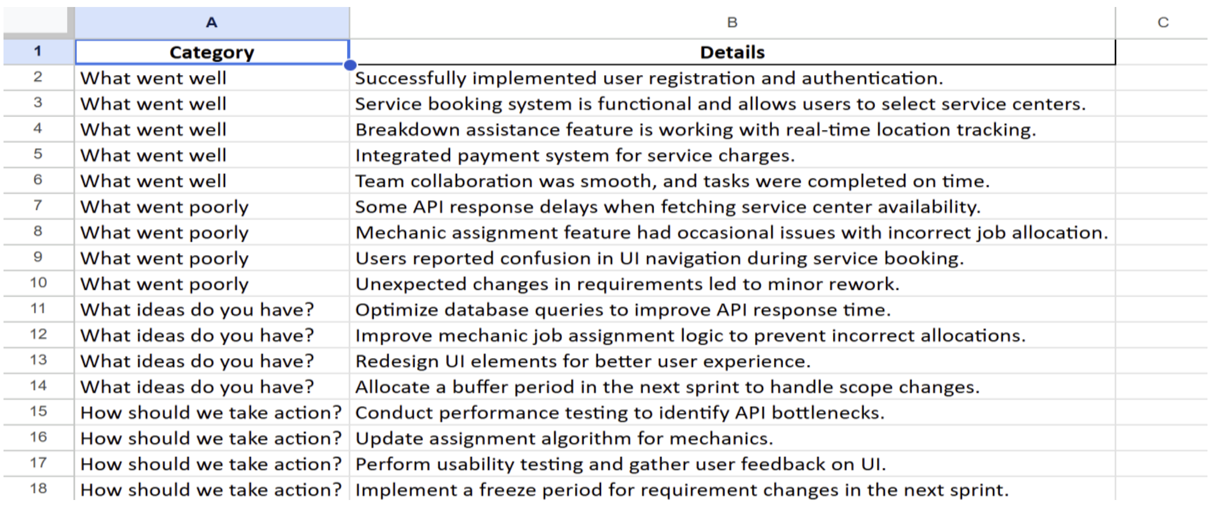


Figure 2.9 Bar graph for Committed Vs Completed User Stories

From figure 2.9 we can say that out of 10 user stories we managed to complete 5 user stories successfully.

2.1.8 Sprint Retrospective

Table 2.4 Sprint Retrospective for the Sprint 1



Category	Details
What went well	Successfully implemented user registration and authentication.
What went well	Service booking system is functional and allows users to select service centers.
What went well	Breakdown assistance feature is working with real-time location tracking.
What went well	Integrated payment system for service charges.
What went well	Team collaboration was smooth, and tasks were completed on time.
What went poorly	Some API response delays when fetching service center availability.
What went poorly	Mechanic assignment feature had occasional issues with incorrect job allocation.
What went poorly	Users reported confusion in UI navigation during service booking.
What went poorly	Unexpected changes in requirements led to minor rework.
What ideas do you have?	Optimize database queries to improve API response time.
What ideas do you have?	Improve mechanic job assignment logic to prevent incorrect allocations.
What ideas do you have?	Redesign UI elements for better user experience.
What ideas do you have?	Allocate a buffer period in the next sprint to handle scope changes.
How should we take action?	Conduct performance testing to identify API bottlenecks.
How should we take action?	Update assignment algorithm for mechanics.
How should we take action?	Perform usability testing and gather user feedback on UI.
How should we take action?	Implement a freeze period for requirement changes in the next sprint.

Table 2.4 represents the sprint retrospective for the sprint 1

2.2 SPRINT 2

2.2.1 Sprint Goal with User Stories of Sprint 2

The goal of Sprint 2 is to enhance the *Zerelo* platform with intelligent service recommendations, collaborative service tracking, visual analytics, and multi-currency support. This sprint aims to make Zerelo smarter and more globally accessible, providing users with AI-based service predictions, cost-sharing capabilities for group services, visual dashboards, and support for service billing in various currencies.

Table 2.5 User stories for sprint 2

S.NO	Detailed User Stories
US #1	As a user, I want to see AI-based insights on service usage so that I can schedule and budget better.
US #2	As a user, I want to set a service budget and get alerts if I'm close to my limit, so I don't overspend.
US #3	AAs a user, I want to share service costs with others (e.g., roommates or fleet members) to track shared services.
US #4	As a user, I want to view service costs in different currencies so I can manage expenses across countries.
US #5	As a user, I want to see visual reports of service usage over time and by type to better understand maintenance trends.
US #6	As a user, I want to categorize services and see where most of my vehicle or home expenses go.

Table 2.5 represents the user stories which are considered for sprint 2

2.2.2 Functional Document

2.2.2.1 Introduction

Sprint 2 of Zerelo focuses on implementing advanced functionality like AI-based service recommendations, collaborative service cost sharing, multi-currency billing, and graphical service analytics. These features will provide users with predictive maintenance suggestions, efficient group tracking, and greater transparency in spending.

2.2.2.2 Product Goal

- Integrate AI to offer predictive service alerts and maintenance suggestions.

- Let users define budget caps and receive threshold alerts.
- Enable service cost sharing between users in shared households, travel groups, or fleet scenarios.
- Support currency conversion for cross-border users or travelers.
- Provide interactive visualizations (charts, graphs) for service frequency, cost, and category distribution.

2.2.2.3 Demography (Users, Location)

- Users:
    - Individuals (vehicle owners, renters, homeowners), fleet managers, logistics teams, and service coordinators.
    - Users with varying levels of technical and financial awareness.
- Location:
    - Global, with key markets in the US, Europe, India, and Southeast Asia, supporting users with different currencies and cross-border services.

2.2.2.4 Business Processes

- Service Tracking:
    - Users add service requests (breakdown or home), categorize them, and track updates in real-time.
- AI-Based Recommendations:
    - The system reviews past service data and usage history to suggest upcoming maintenance or frequently required services.
- Collaborative Cost Sharing:
    - Users can invite others to split service costs (e.g., shared taxi fleet, household repairs).
    - Contributions are automatically calculated and logged.
- Multi-Currency Support:

  - Real-time conversion allows service charges and payments in different currencies to be standardized into a preferred local currency.

2.2.2.5 Features

Feature 1: AI-Based Service Recommendations

- Description:
  AI engine analyzes user service history to forecast recurring breakdowns or required home maintenance.
- User Story:
  As a user, I want to get service predictions so I can prepare for upcoming breakdowns or maintenance.

Feature 2: Collaborative Service Tracking

- Description:
  Users can share service tickets with other users (e.g., co-owners, roommates, fleet users) to divide costs and monitor jointly.
- User Story:
  As a user, I want to share my service costs with others to track shared services like household plumbing or company vehicles.

Feature 3: Multi-Currency Support

- Description:
  Supports logging and converting costs in different currencies, depending on the user's region or travel requirements.
- User Story:
  As a user, I want to view and log services in different currencies so I can manage my international expenses.

Feature 4: Visual Service Reports

- Description:
  Offers interactive dashboards with charts on types of services used, cost over time, frequency of breakdowns, etc.

- User Story:
  As a user, I want to see visual breakdowns of my service data to understand how and where I'm spending.

2.2.2.6 Authorization Matrix

Table 2.6 Access Level Authorization Matrix

Role	**Access Level**
Administrator	Full access manage users, services, insights, and platform data..
User	Access to personal service data, budgeting, AI suggestions, and sharing features.
Guest User	View-only access to general service information without the ability to request or manage services.

Table 2.6 represents the authorization matrix for the roles of administrator, User and guest user.

2.2.2.7 Assumptions

- AI-based insights require sufficient service history to offer accurate forecasts.
- Multi-currency conversion depends on real-time exchange rate APIs (e.g., OpenExchangeRates).
- Internet connectivity is available to sync live updates and receive notifications.
- The system complies with GDPR, CCPA, and other data protection regulations.

2.2.3 Architecture Document

2.2.3.1 Application

Microservices (Sprint 2 Specific):

- Service Management Service: Handles requests, updates, and categorization of services.
- AI Recommendation Service: Processes service logs to generate future service predictions.
- Cost Sharing Service: Tracks contributions and shared payments between users.
- Currency Service: Fetches live rates and converts service costs accordingly.

- Notification Service: Sends updates and alerts for AI suggestions, cost sharing, and budget warnings.

2.2.3.2 System Architecture

1. Frontend Layer:

- Web UI or mobile app used to interact with Zerelo for service requests, cost tracking, sharing, and insights.

2. API Gateway:

- Directs user interactions to the appropriate backend microservices.

3. Backend Layer:

- Microservices: Encapsulate all core services—modular, scalable, and independently deployable.
- Databases: Store users, services, budget limits, shared cost records, and insight data.
- AI Engine: Trained on user service data to make future predictions and suggest preventive maintenance.

4. Third-Party Integrations:

- Currency Conversion API: Real-time exchange rate provider (e.g., Fixer, OpenExchangeRates).
- Notification System: SMS, email, or in-app push notifications for events like nearing budgets or sharing acceptance.

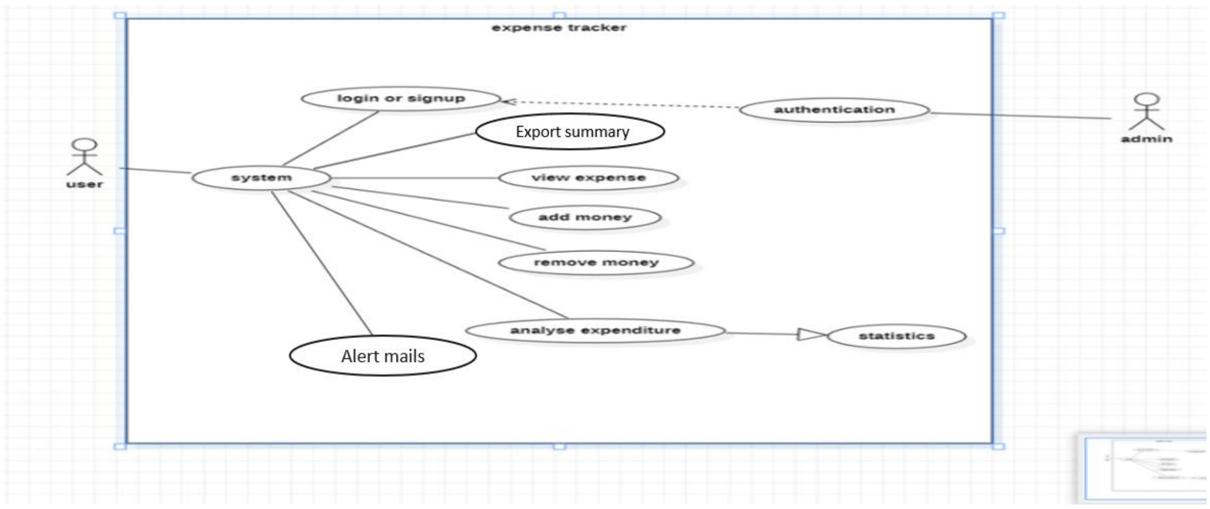


Fig 2.10 use case diagram

Figure 2.10 represents the use case diagram of the entire project.

- **Real-Time Exchanges:**
  - Service Booking and Updates: When a user books or updates a breakdown or home service.
  - AI-Based Service Insights: AI modules analyze data and provide real-time suggestions for preventive maintenance or service optimization.
  - Notifications and Alerts: Budget threshold alerts, shared service confirmations, service completions, etc.
- Periodic Syncs:
  - User Activity Logs: Logged periodically for analytics, reporting, and AI model improvement.
  - Shared Services and Budget Trends: Aggregated and synced at intervals to update dashboards and collaborative tracking modules.

Data Sets Involved

- User Data:
  Includes user profile info (name, contact, address), preferences (e.g., preferred mechanic or plumber), authentication data, and location settings.

- Service Request Data:
  Includes service type (vehicle breakdown, home repair), location, timestamps, assigned professional, cost details, and service status.
- Insight Data:
  AI-generated insights such as:
  - Frequently requested services
  - Predicted breakdowns or maintenance needs
  - Cost-saving suggestions
  - Usage trends by time or location
- Currency Data:
  Live currency exchange rates pulled from third-party APIs, used for converting and standardizing service charges across countries.
- Shared Service Data:
  Data related to service cost sharing among users including:
  - Participants
  - Contribution breakdown
  - Confirmation status
  - Payment records

Mode of Exchanges

- API (RESTful):
  - Used for synchronous interactions between the frontend and backend services (e.g., booking services, fetching insights, updating user preferences).
  - Also used for integration with third-party APIs (e.g., currency converters, SMS/email services).
- Message Queues (e.g., RabbitMQ/Kafka):
  - Employed for handling asynchronous operations, such as:
    - Service completion notifications

        - Budget alerts
        - Service reminder broadcasts
- File-Based Exchanges:
    - Used primarily for:
        - Bulk upload of service records (e.g., CSV import for fleet owners)
        - Historical service logs for analytics
        - Admin-level reporting/export

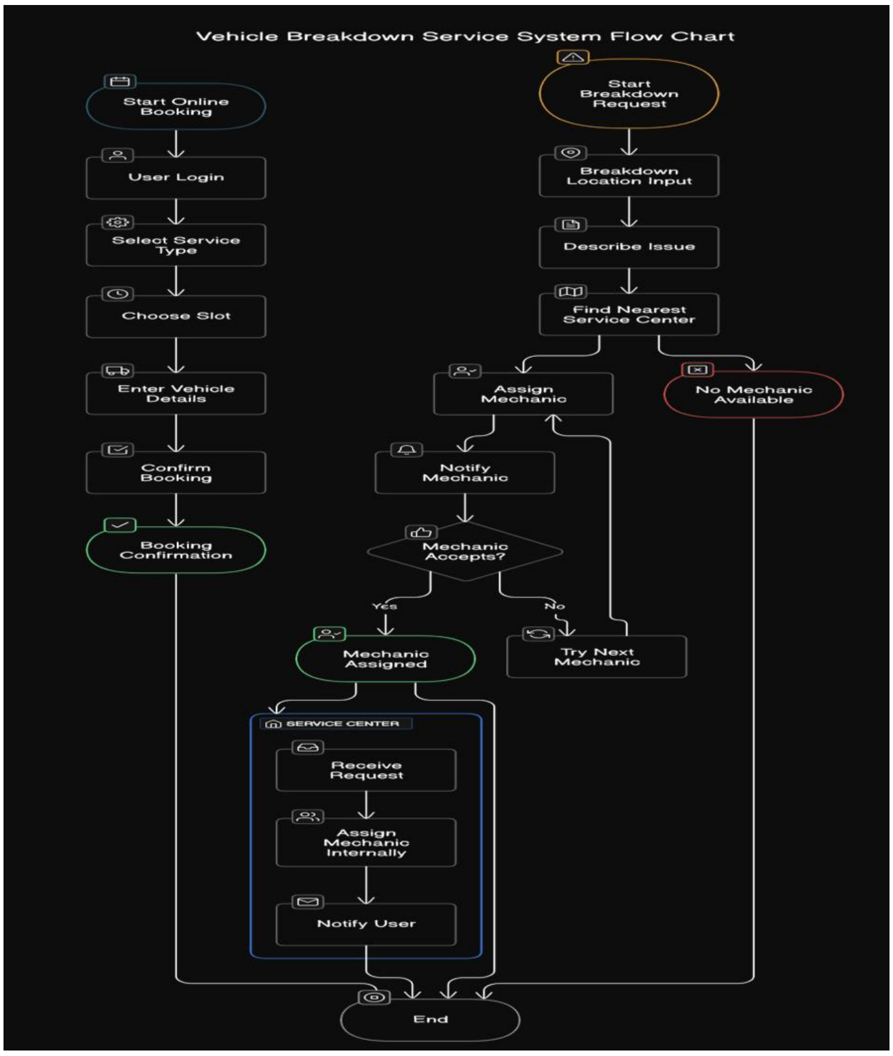
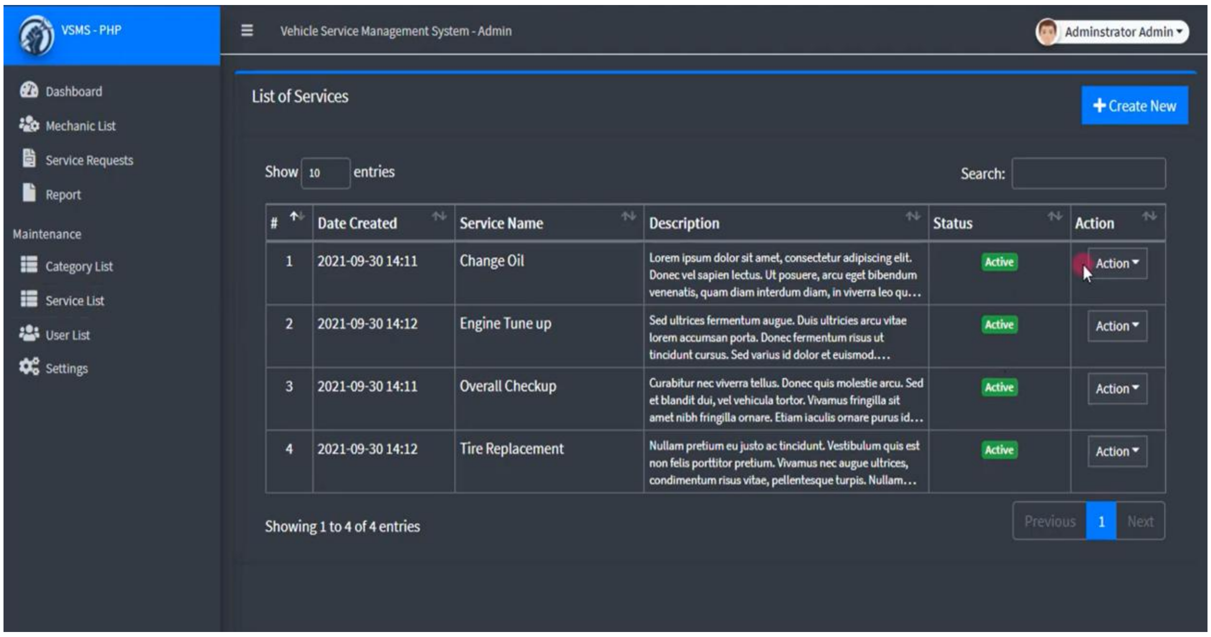


Fig 2.11 ER diagram

Figure 2.11 represents the ER diagram for the entire project explaining about the data transfers.

2.2.4 UI Design



List of Services

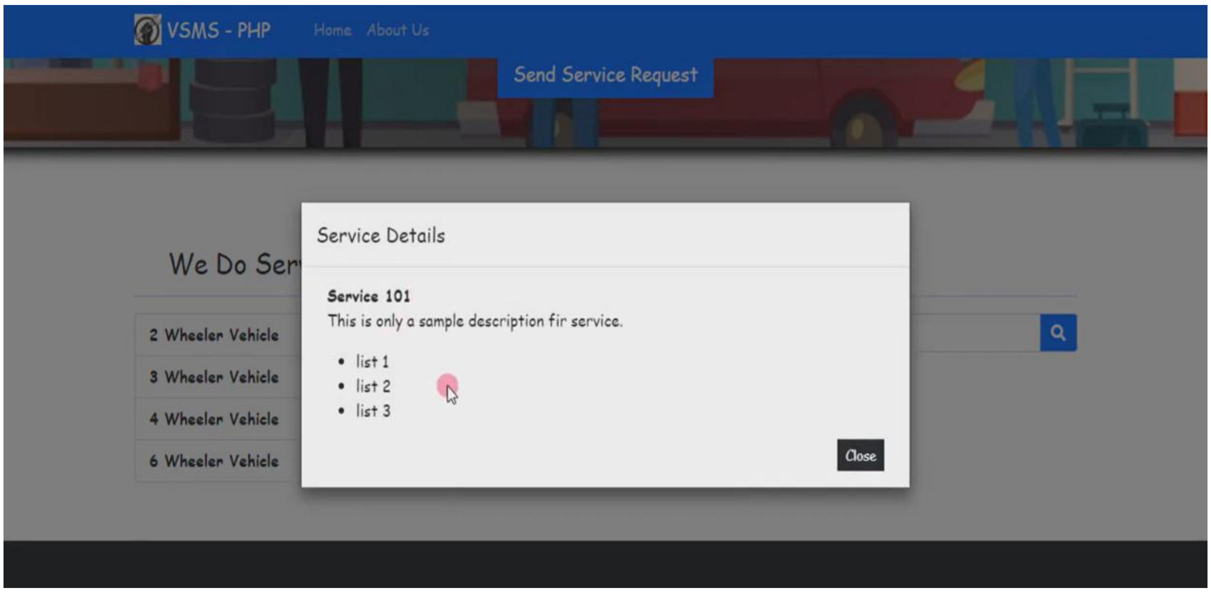
Show 10 entries   Search:

#	Date Created	Service Name	Description	Status	Action
1	2021-09-30 14:11	Change Oil	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec vel sapien lectus. Ut posuere, arcu eget bibendum venenatis, quam diam interdum diam, in viverra leo qu...	Active	Action
2	2021-09-30 14:12	Engine Tune up	Sed ultrices fermentum augue. Duis ultricies arcu vitae lorem accumsan porta. Donec fermentum risus ut tincidunt cursus. Sed varius id dolor et euismod....	Active	Action
3	2021-09-30 14:11	Overall Checkup	Curabitur nec viverra tellus. Donec quis molestie arcu. Sed et blandit dui, vel vehicula tortor. Vivamus fringilla sit amet nibh fringilla ornare. Etiam iaculis ornare purus id...	Active	Action
4	2021-09-30 14:12	Tire Replacement	Nullam pretium eu justo ac tincidunt. Vestibulum quis est non felis porttitor pretium. Vivamus nec augue ultrices, condimentum risus vitae, pellentesque turpis. Nullam...	Active	Action

Showing 1 to 4 of 4 entries   Previous 1 Next

Fig 2.12 UI design for service checkup

Figure 2.12 represents the UI design for service checkup.



Service Details

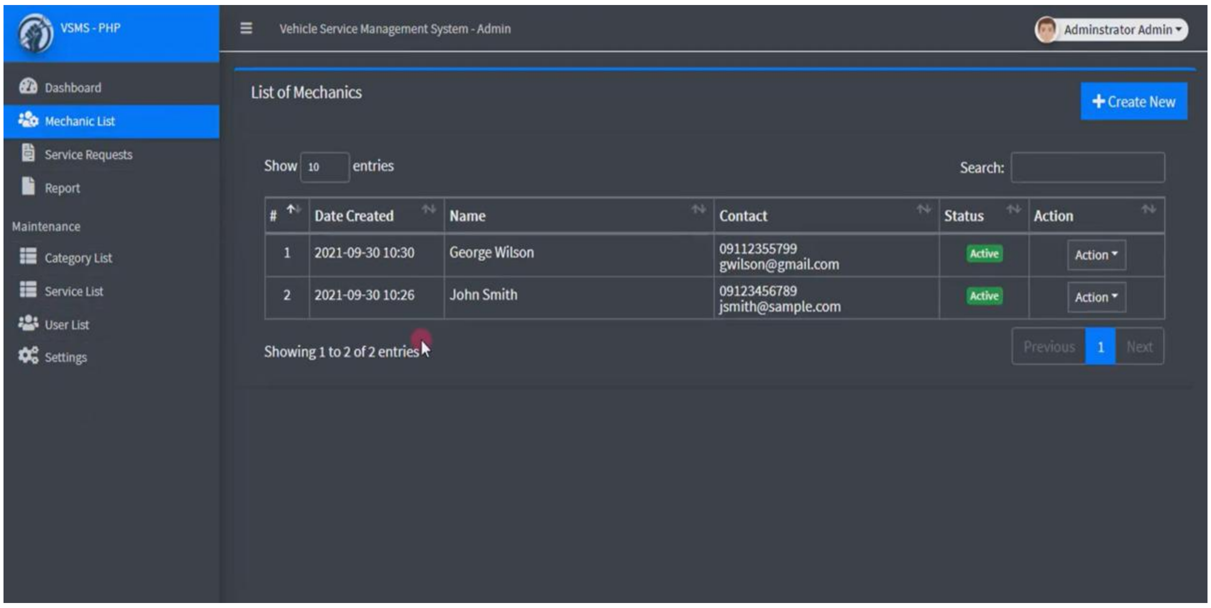
Service 101

This is only a sample description fir service.

- list 1
- list 2
- list 3

Fig 2.13 UI design for Service details

Figure 2.13 represents the UI design for Service details.



#	Date Created	Name	Contact	Status	Action
1	2021-09-30 10:30	George Wilson	09112355799 gwilson@gmail.com	Active	Action
2	2021-09-30 10:26	John Smith	09123456789 jsmith@sample.com	Active	Action

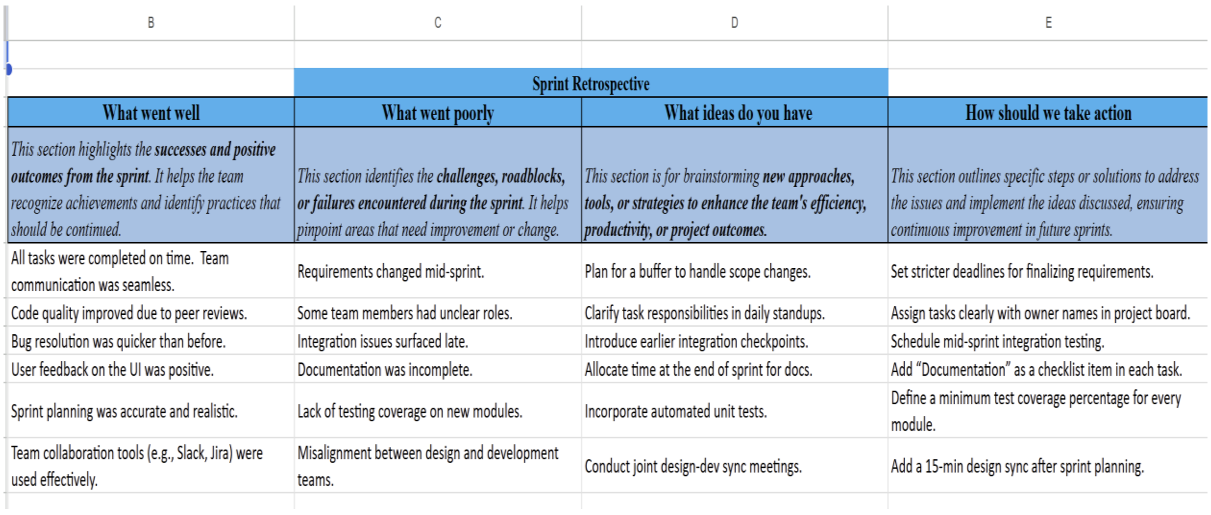
Showing 1 to 2 of 2 entries

Fig 2.14 UI design for Mechanic List

Figure 2.14 represents the UI design for Mechanic List

2.2.5 Functional Test Cases

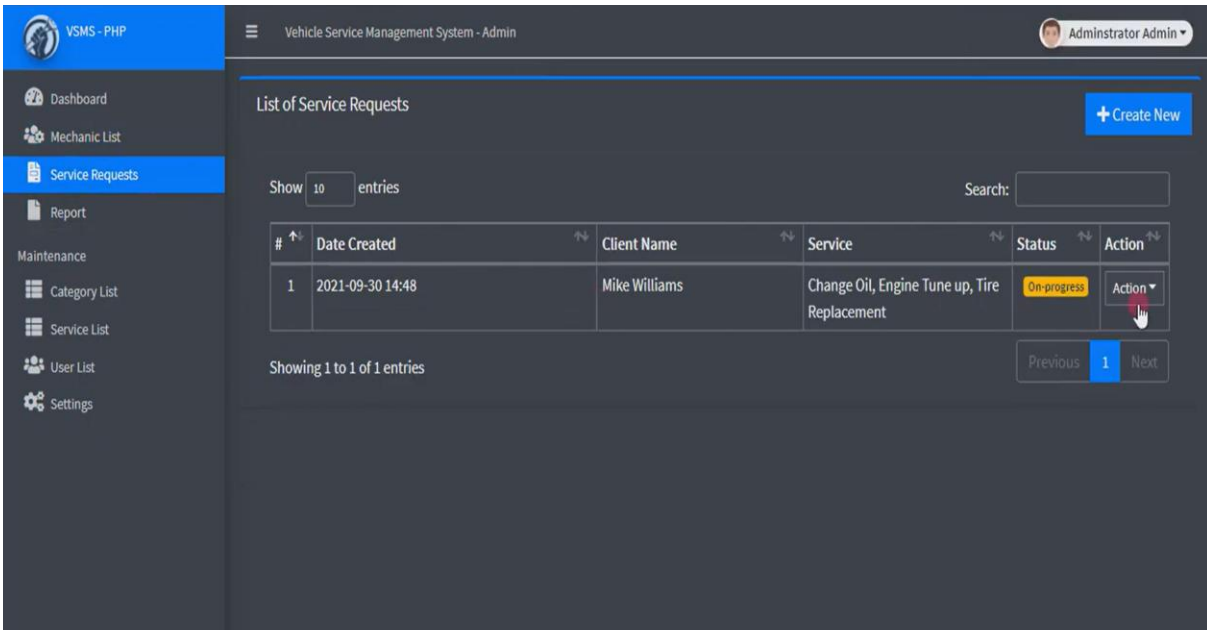
Table 2.7 Functional test cases sprint 2



B	C	D	E
	Sprint Retrospective		
What went well	What went poorly	What ideas do you have	How should we take action
*This section highlights the **successes and positive outcomes from the sprint**. It helps the team recognize achievements and identify practices that should be continued.*	*This section identifies the **challenges, roadblocks, or failures encountered during the sprint**. It helps pinpoint areas that need improvement or change.*	*This section is for brainstorming **new approaches, tools, or strategies to enhance the team's efficiency, productivity, or project outcomes.***	*This section outlines specific steps or solutions to address the issues and implement the ideas discussed, ensuring continuous improvement in future sprints.*
All tasks were completed on time. Team communication was seamless.	Requirements changed mid-sprint.	Plan for a buffer to handle scope changes.	Set stricter deadlines for finalizing requirements.
Code quality improved due to peer reviews.	Some team members had unclear roles.	Clarify task responsibilities in daily standups.	Assign tasks clearly with owner names in project board.
Bug resolution was quicker than before.	Integration issues surfaced late.	Introduce earlier integration checkpoints.	Schedule mid-sprint integration testing.
User feedback on the UI was positive.	Documentation was incomplete.	Allocate time at the end of sprint for docs.	Add "Documentation" as a checklist item in each task.
Sprint planning was accurate and realistic.	Lack of testing coverage on new modules.	Incorporate automated unit tests.	Define a minimum test coverage percentage for every module.
Team collaboration tools (e.g., Slack, Jira) were used effectively.	Misalignment between design and development teams.	Conduct joint design-dev sync meetings.	Add a 15-min design sync after sprint planning.

Table 2.7 represents the functional test cases for our sprint 2.

2.2.6 Service Request



List of Service Requests

Show 10 entries

Search:

#	Date Created	Client Name	Service	Status	Action
1	2021-09-30 14:48	Mike Williams	Change Oil, Engine Tune up, Tire Replacement	On-progress	Action

Showing 1 to 1 of 1 entries

Fig 2.15 **Service Request**

Figure 2.15 gives an idea about the service requests

2.2.7 COMMITTED Vs COMPLETED USER STORIES

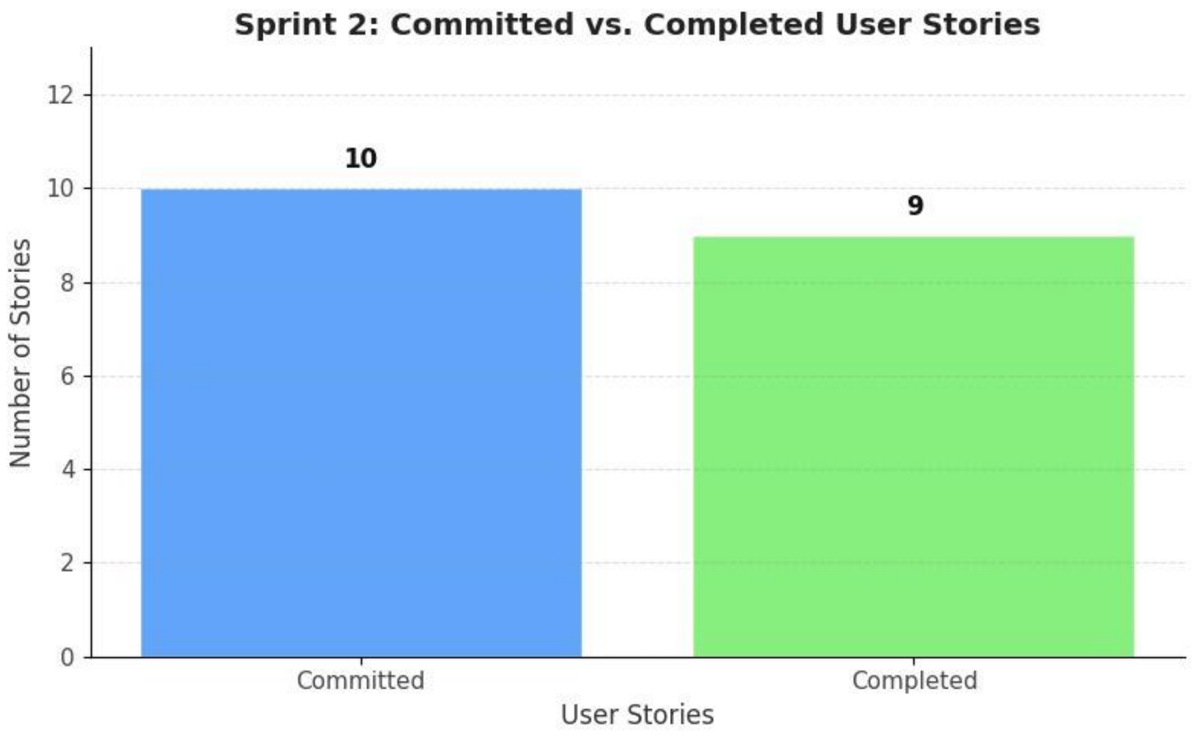
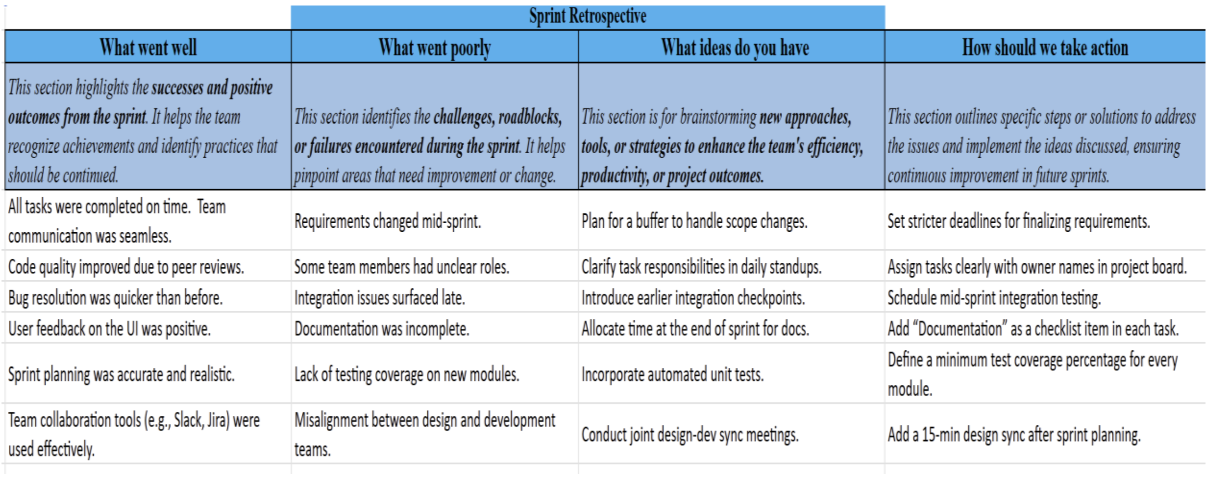


Fig 2.16 Committed vs Completed user stories sprint 2

From Figure 2.16, we can say that 9 out of 10 user stories execution has been done during the time of sprint 2.

2.2.8 Sprint Retrospective

Table 2.8 Sprint retrospective for sprint 2



	Sprint Retrospective		
What went well	**What went poorly**	**What ideas do you have**	**How should we take action**
This section highlights the ***successes and positive outcomes from the sprint****. It helps the team recognize achievements and identify practices that should be continued.*	*This section identifies the* ***challenges, roadblocks, or failures encountered during the sprint****. It helps pinpoint areas that need improvement or change.*	*This section is for brainstorming* ***new approaches, tools, or strategies to enhance the team's efficiency, productivity, or project outcomes.***	*This section outlines specific steps or solutions to address the issues and implement the ideas discussed, ensuring continuous improvement in future sprints.*
All tasks were completed on time. Team communication was seamless.	Requirements changed mid-sprint.	Plan for a buffer to handle scope changes.	Set stricter deadlines for finalizing requirements.
Code quality improved due to peer reviews.	Some team members had unclear roles.	Clarify task responsibilities in daily standups.	Assign tasks clearly with owner names in project board.
Bug resolution was quicker than before.	Integration issues surfaced late.	Introduce earlier integration checkpoints.	Schedule mid-sprint integration testing.
User feedback on the UI was positive.	Documentation was incomplete.	Allocate time at the end of sprint for docs.	Add "Documentation" as a checklist item in each task.
Sprint planning was accurate and realistic.	Lack of testing coverage on new modules.	Incorporate automated unit tests.	Define a minimum test coverage percentage for every module.
Team collaboration tools (e.g., Slack, Jira) were used effectively.	Misalignment between design and development teams.	Conduct joint design-dev sync meetings.	Add a 15-min design sync after sprint planning.

Table 2.8 represents the sprint retrospective we have done for the sprint 2

CHAPTER 3

RESULTS AND DISCUSSION

3.1 PROJECT OUTCOMES

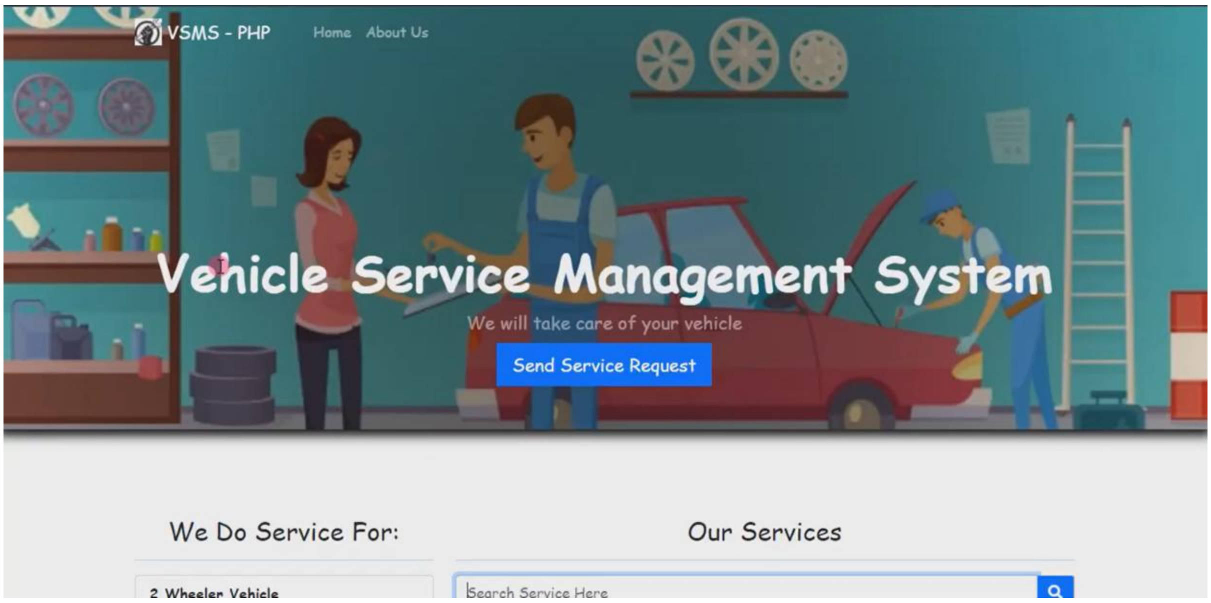


Fig 3.1 Home page

Figure 3.1 shows thw login page of our project which have fields to enter correct username and password

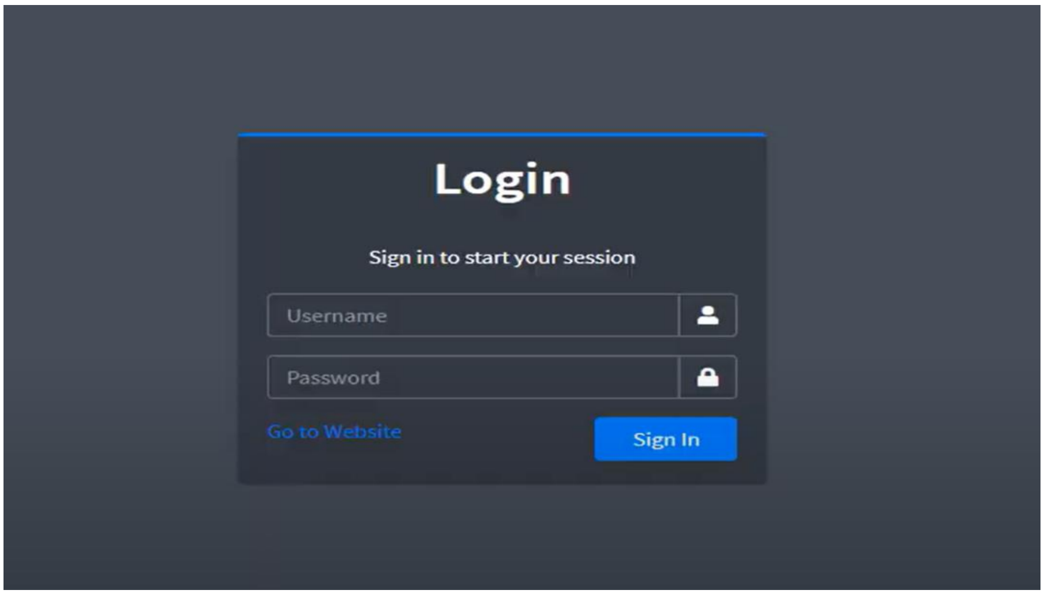
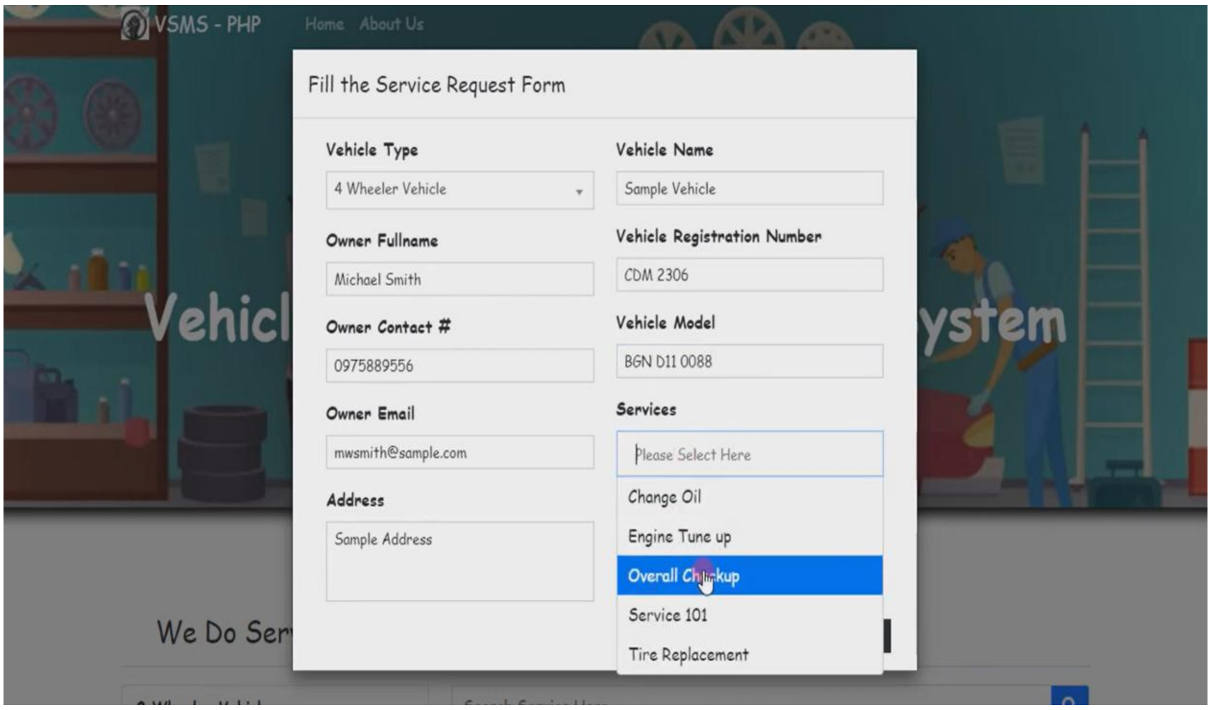


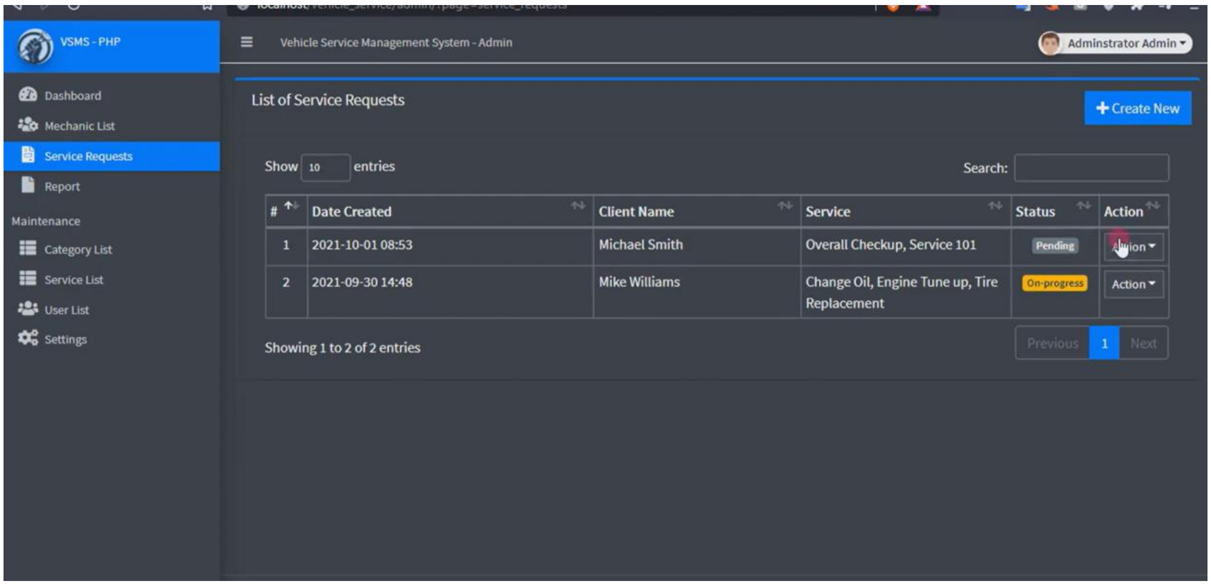
Fig 3.2 Login Page


Fill the Service Request Form

Vehicle Type: 4 Wheeler Vehicle	Vehicle Name: Sample Vehicle
Owner Fullname: Michael Smith	Vehicle Registration Number: CDM 2306
Owner Contact #: 0975889556	Vehicle Model: BGN D11 0088
Owner Email: mwsmith@sample.com	Services: Please Select Here
Address: Sample Address	

Fig 3.3 Service Request Form

From figure 3.3 we can see the Service request form where we can fill the services

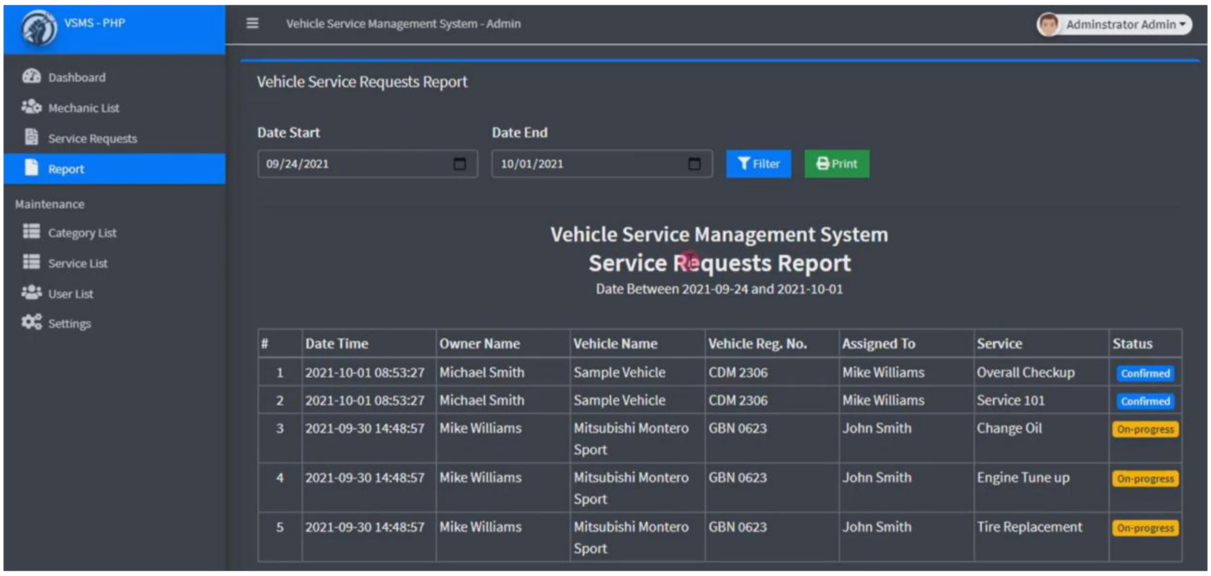

List of Service Requests

#	Date Created	Client Name	Service	Status	Action
1	2021-10-01 08:53	Michael Smith	Overall Checkup, Service 101	Pending	Action
2	2021-09-30 14:48	Mike Williams	Change Oil, Engine Tune up, Tire Replacement	On-progress	Action

Showing 1 to 2 of 2 entries

Fig 3.4 Service Requests

Figure 3.4 represents the Service Requests that are entered



Vehicle Service Requests Report

Date Start: 09/24/2021

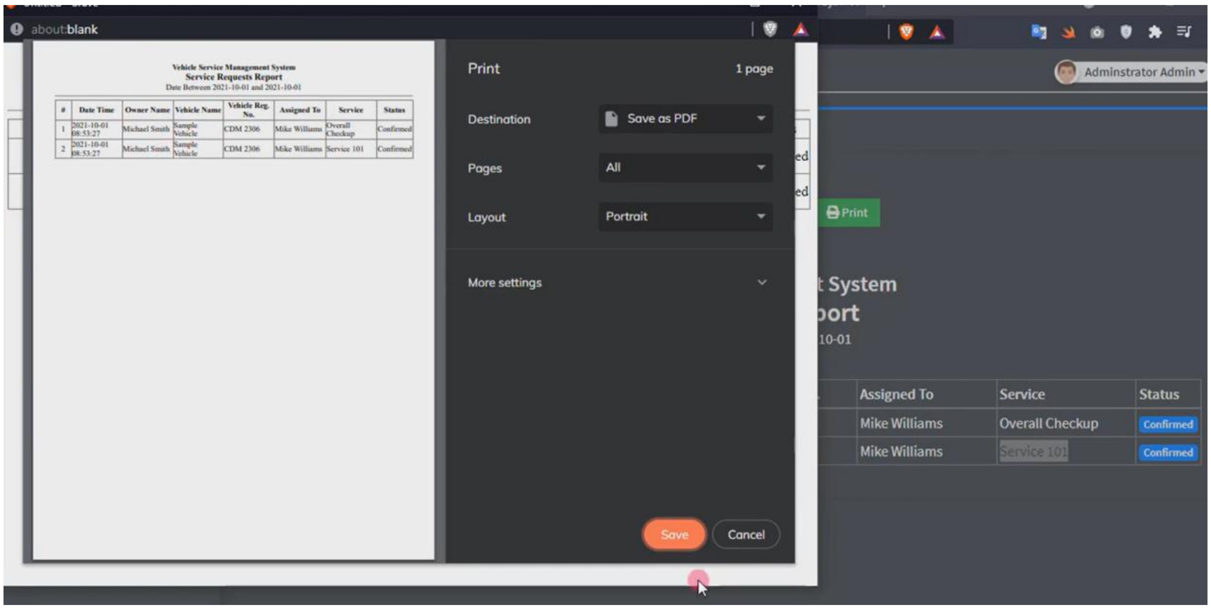
Date End: 10/01/2021

Vehicle Service Management System
Service Requests Report
Date Between 2021-09-24 and 2021-10-01

#	Date Time	Owner Name	Vehicle Name	Vehicle Reg. No.	Assigned To	Service	Status
1	2021-10-01 08:53:27	Michael Smith	Sample Vehicle	CDM 2306	Mike Williams	Overall Checkup	Confirmed
2	2021-10-01 08:53:27	Michael Smith	Sample Vehicle	CDM 2306	Mike Williams	Service 101	Confirmed
3	2021-09-30 14:48:57	Mike Williams	Mitsubishi Montero Sport	GBN 0623	John Smith	Change Oil	On-progress
4	2021-09-30 14:48:57	Mike Williams	Mitsubishi Montero Sport	GBN 0623	John Smith	Engine Tune up	On-progress
5	2021-09-30 14:48:57	Mike Williams	Mitsubishi Montero Sport	GBN 0623	John Smith	Tire Replacement	On-progress

Fig 3.5 Report

From figure 3.5, we can see the reports in the report section



Destination: Save as PDF

Pages: All

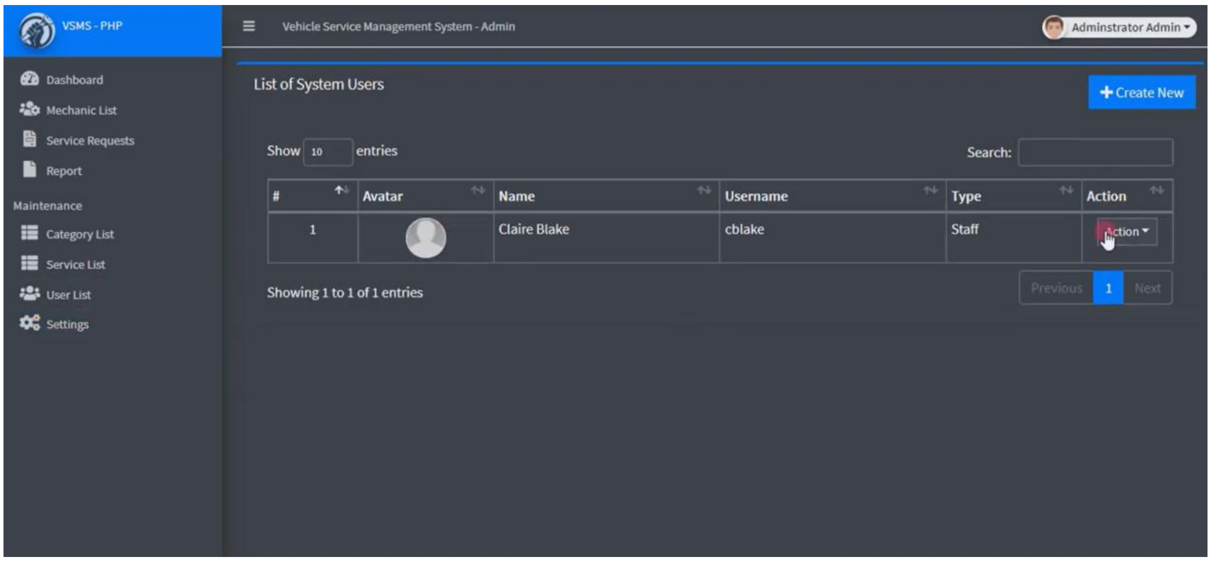
Layout: Portrait

Vehicle Service Management System
Service Requests Report
Date Between 2021-10-01 and 2021-10-01

#	Date Time	Owner Name	Vehicle Name	Vehicle Reg. No.	Assigned To	Service	Status
1	2021-10-01 08:53:27	Michael Smith	Sample Vehicle	CDM 2306	Mike Williams	Overall Checkup	Confirmed
2	2021-10-01 08:53:27	Michael Smith	Sample Vehicle	CDM 2306	Mike Williams	Service 101	Confirmed

Fig 3.6 Saving report

From Figure 3.6, we can save the report to our files



List of System Users

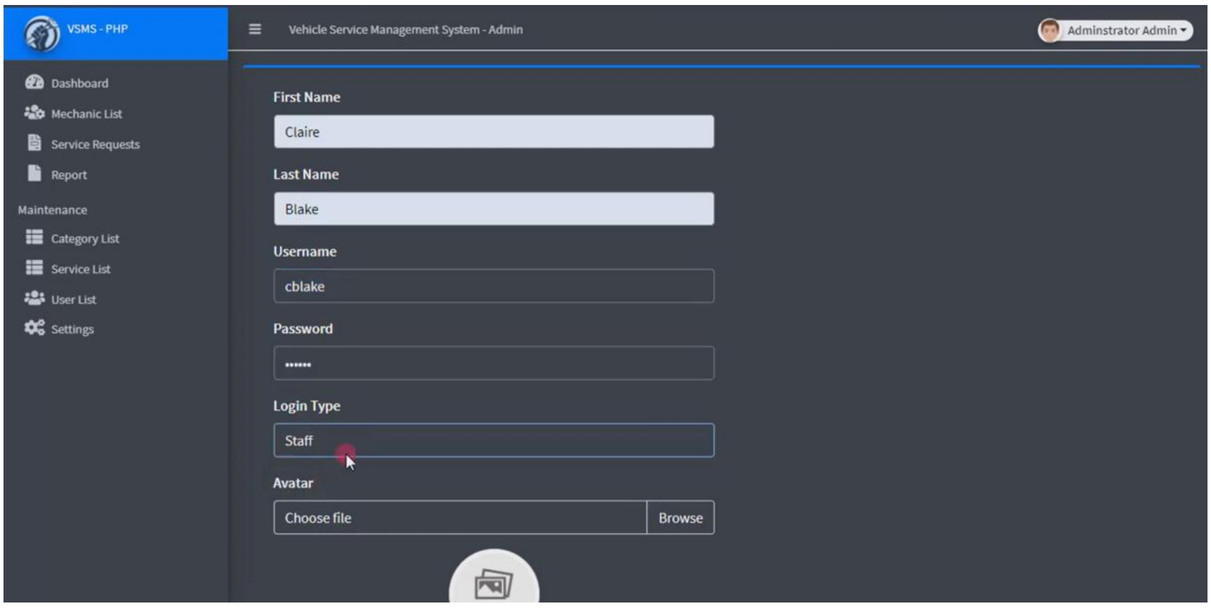
Show 10 entries

Search:

#	Avatar	Name	Username	Type	Action
1		Claire Blake	cblake	Staff	Action

Showing 1 to 1 of 1 entries

Fig 3.7 User list



First Name: Claire

Last Name: Blake

Username: cblake

Password: ••••••

Login Type: Staff

Avatar: Choose file

Fig 3.8 Adding Users

From fig 3.8 we can notice the different options for preferred currencies which help to track your expenses when you are out of the country.

DISSCUSSIONS

• Functional Outcomes:

Zerelo successfully implemented real-time breakdown assistance, home service booking, mechanic tracking, and automated mechanic assignment using location-based logic. Core microservices like user management, service scheduling, and notification dispatch worked in harmony to support on-demand vehicle services. Features such as mechanic status updates and service history tracking helped users monitor and manage their vehicle servicing lifecycle effectively.

• User Experience Outcomes:

Users found the booking process intuitive and quick, with high satisfaction around real-time mechanic tracking and transparent status updates. The breakdown assistance feature was particularly well-received by users in emergency situations. However, some users suggested enhancements in response times during peak hours and additional support channels (e.g., live chat) for smoother communication.

• Technical Outcomes:

Zerelo's microservices-based architecture enabled modular development, better maintainability, and horizontal scalability. Real-time data exchange using APIs and queues (e.g., for notifications and mechanic updates) worked efficiently under normal load. However, during peak traffic (e.g., weekends or evenings), some latency and API throttling issues were reported, indicating the need for backend optimization and autoscaling infrastructure support (e.g., Kubernetes, Load Balancers).

• Business Outcomes:

Zerelo demonstrated strong potential as a reliable, user-friendly alternative to traditional garage services. The blend of on-demand assistance, AI-driven mechanic dispatching, and user-mechanic transparency created a competitive edge. This positions Zerelo for strategic partnerships with insurance companies, automobile manufacturers, and roadside assistance providers, paving the way for revenue diversification and market expansion.

3.2 COMMITTED VS COMPLETED USER STORIES

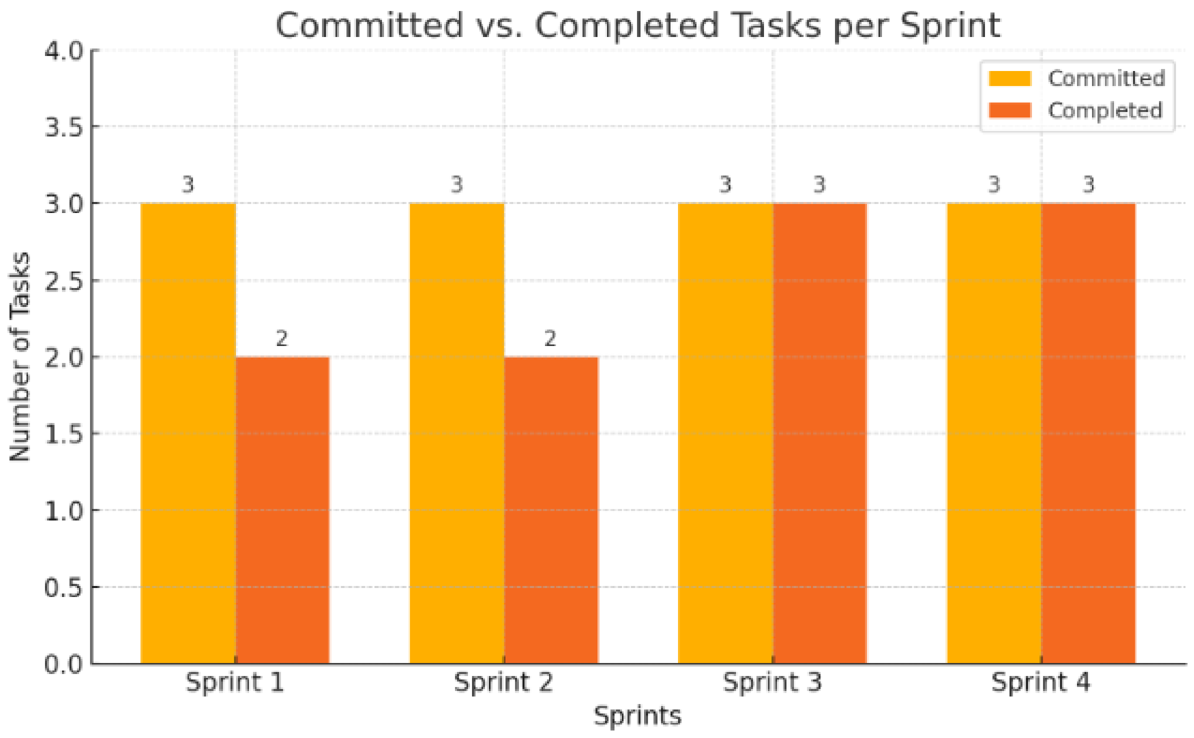


Fig 3.9 Committed Vs Completed User stories

From figure 3.9, we can see number of user stories which are aimed to commit and number of user stories completed over different sprints.

CHAPTER 4

CONCLUSION & FUTURE ENHANCEMENTS

CONCLUSION

The Zerelo project has successfully achieved its mission of delivering an intelligent, reliable, and user-centric platform for managing vehicle breakdowns and service appointments. Through the integration of real-time breakdown assistance, AI-driven mechanic dispatching, slot booking, and live status tracking, Zerelo has streamlined the traditionally manual process of vehicle servicing. The microservices architecture ensured modular scalability, while real-time notifications, secure authentication, and a responsive UI provided users with transparency and control over their vehicle's service lifecycle. Collaborative features like mechanic-user chat and live tracking have elevated user trust and satisfaction, setting Zerelo apart in the domain of roadside assistance and vehicle service platforms.FUTURE ENHANCEMENTS

 Predictive Maintenance Insights:
Introduce AI models to predict potential breakdowns based on service history, vehicle usage, and environmental factors—enabling proactive maintenance recommendations for users.

 Mobile App Integration:
Launch Android and iOS versions of Zerelo, giving users the convenience of booking, tracking, and communicating with mechanics on the go.

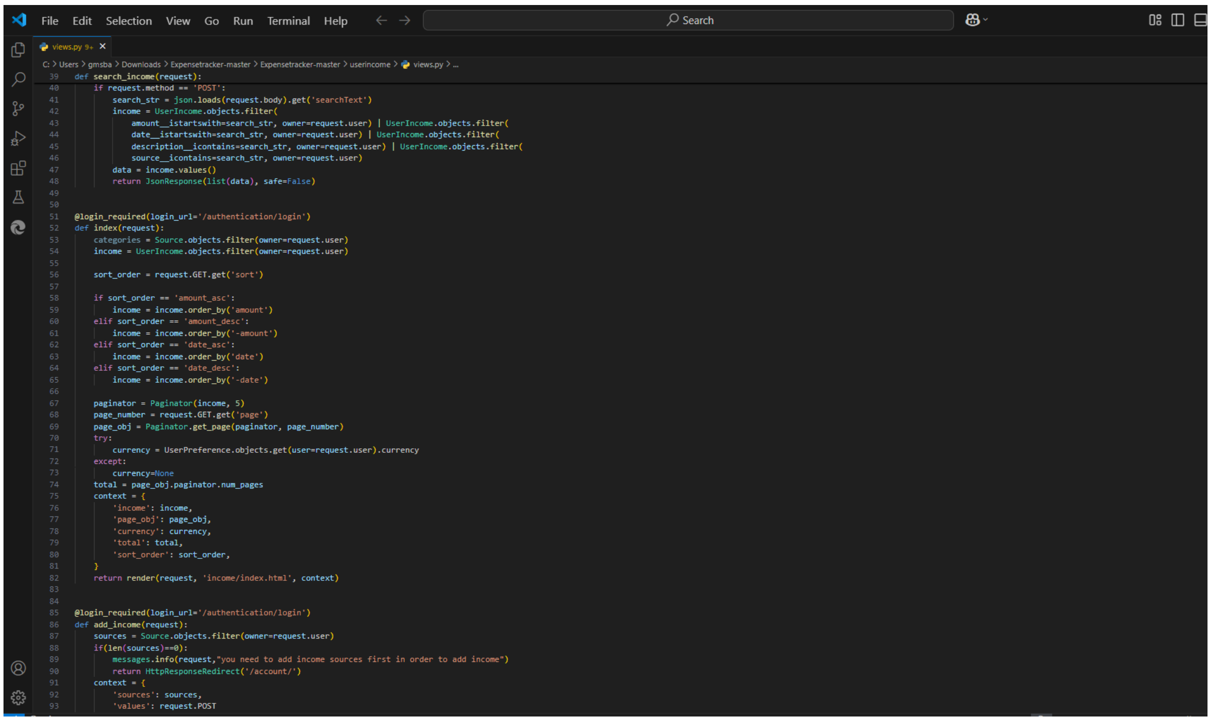
 Voice-Based Booking Assistant:
Implement voice recognition for users to book services or report breakdowns using simple voice commands—ideal for emergency hands-free situations.

 Fleet Management Module:
Extend the platform to support commercial and logistics fleets, offering tools like bulk booking, performance tracking, and analytics for multiple vehicles.

 IoT Integration:
Integrate with vehicle sensors or OBD devices to auto-detect faults and trigger service requests or warnings in real-time.

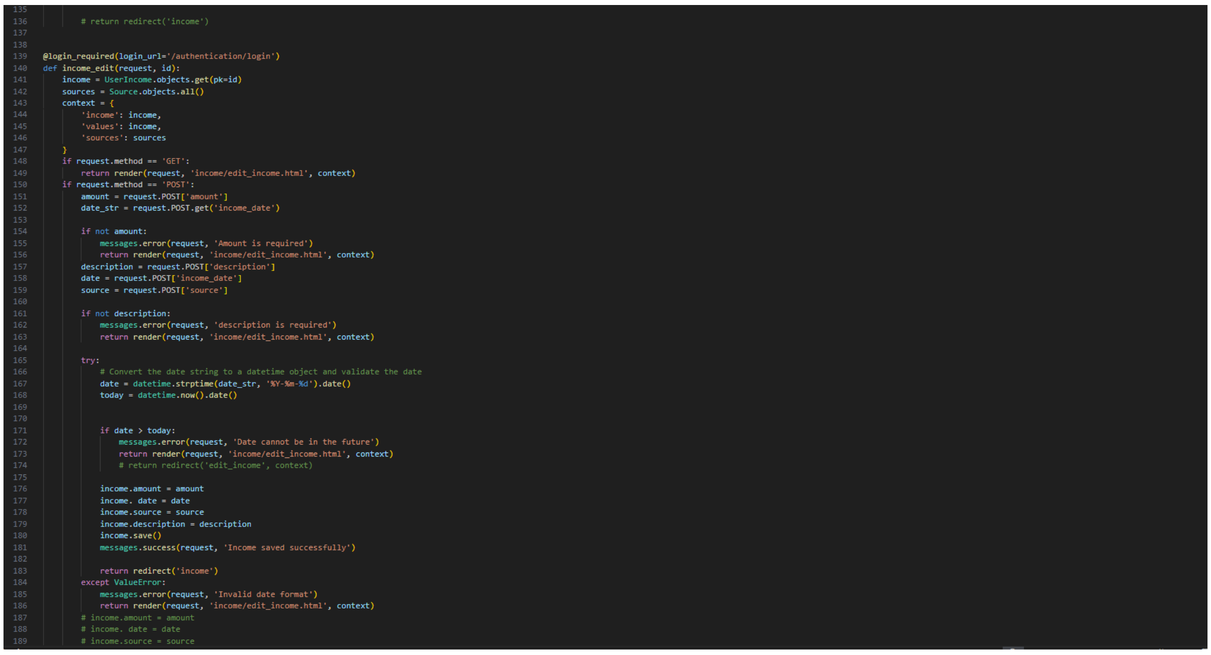
APPENDIX

A.SAMPLE CODING

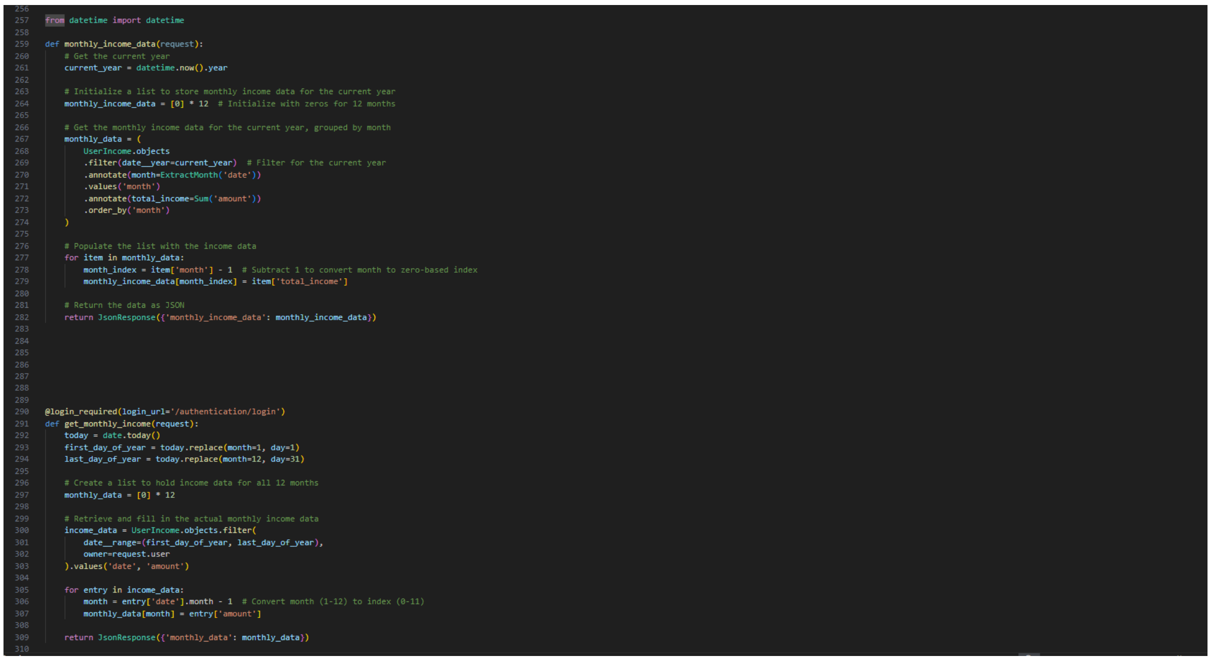


File Edit Selection View Go Run Terminal Help
Search
views.py 9+
C: > Users > gmsba > Downloads > Expensetracker-master > Expensetracker-master > userincome > views.py > ...

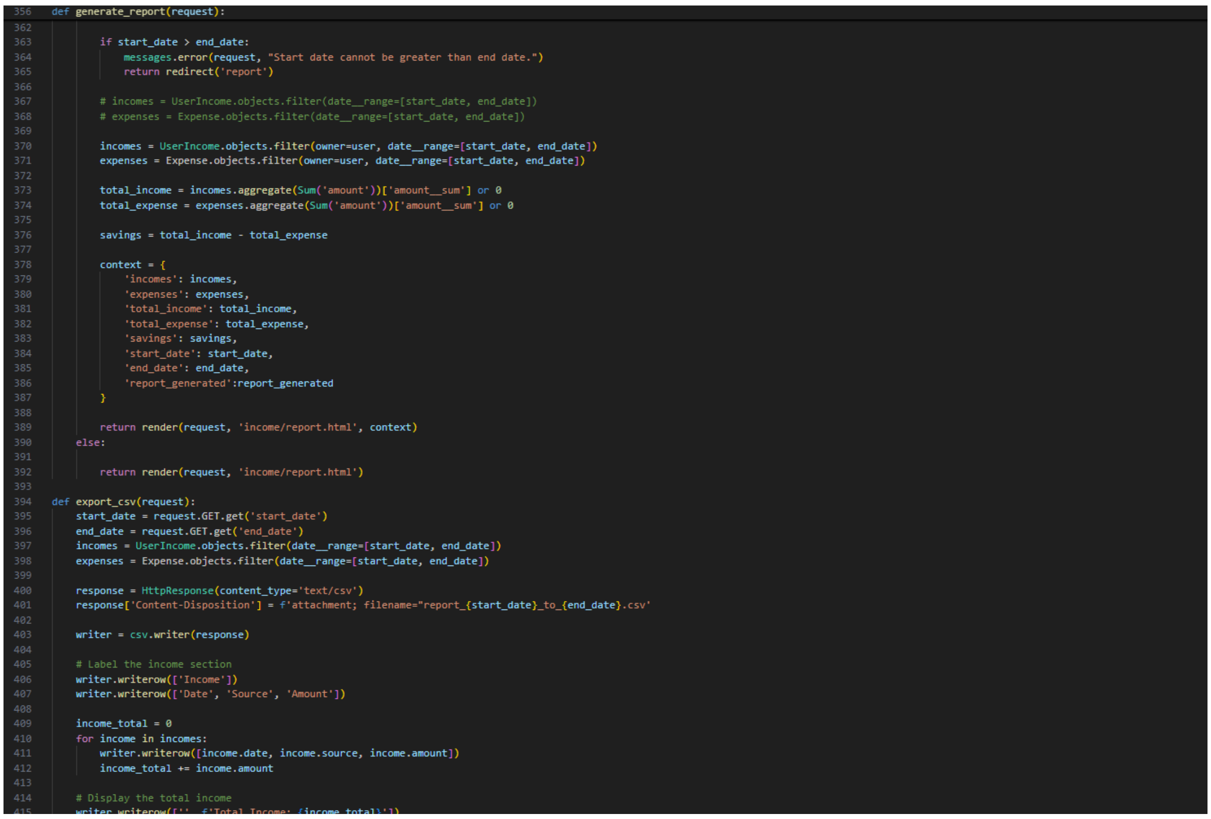
```python
def search_income(request):
    if request.method == 'POST':
        search_str = json.loads(request.body).get('searchText')
        income = UserIncome.objects.filter(
            amount__istartswith=search_str, owner=request.user) | UserIncome.objects.filter(
            date__istartswith=search_str, owner=request.user) | UserIncome.objects.filter(
            description__icontains=search_str, owner=request.user) | UserIncome.objects.filter(
            source__icontains=search_str, owner=request.user)
        data = income.values()
        return JsonResponse(list(data), safe=False)


@login_required(login_url='/authentication/login')
def index(request):
    categories = Source.objects.filter(owner=request.user)
    income = UserIncome.objects.filter(owner=request.user)

    sort_order = request.GET.get('sort')

    if sort_order == 'amount_asc':
        income = income.order_by('amount')
    elif sort_order == 'amount_desc':
        income = income.order_by('-amount')
    elif sort_order == 'date_asc':
        income = income.order_by('date')
    elif sort_order == 'date_desc':
        income = income.order_by('-date')

    paginator = Paginator(income, 5)
    page_number = request.GET.get('page')
    page_obj = Paginator.get_page(paginator, page_number)
    try:
        currency = UserPreference.objects.get(user=request.user).currency
    except:
        currency=None
    total = page_obj.paginator.num_pages
    context = {
        'income': income,
        'page_obj': page_obj,
        'currency': currency,
        'total': total,
        'sort_order': sort_order,
    }
    return render(request, 'income/index.html', context)


@login_required(login_url='/authentication/login')
def add_income(request):
    sources = Source.objects.filter(owner=request.user)
    if(len(sources)==0):
        messages.info(request,"you need to add income sources first in order to add income")
        return HttpResponseRedirect('/account/')
    context = {
        'sources': sources,
        'values': request.POST
```



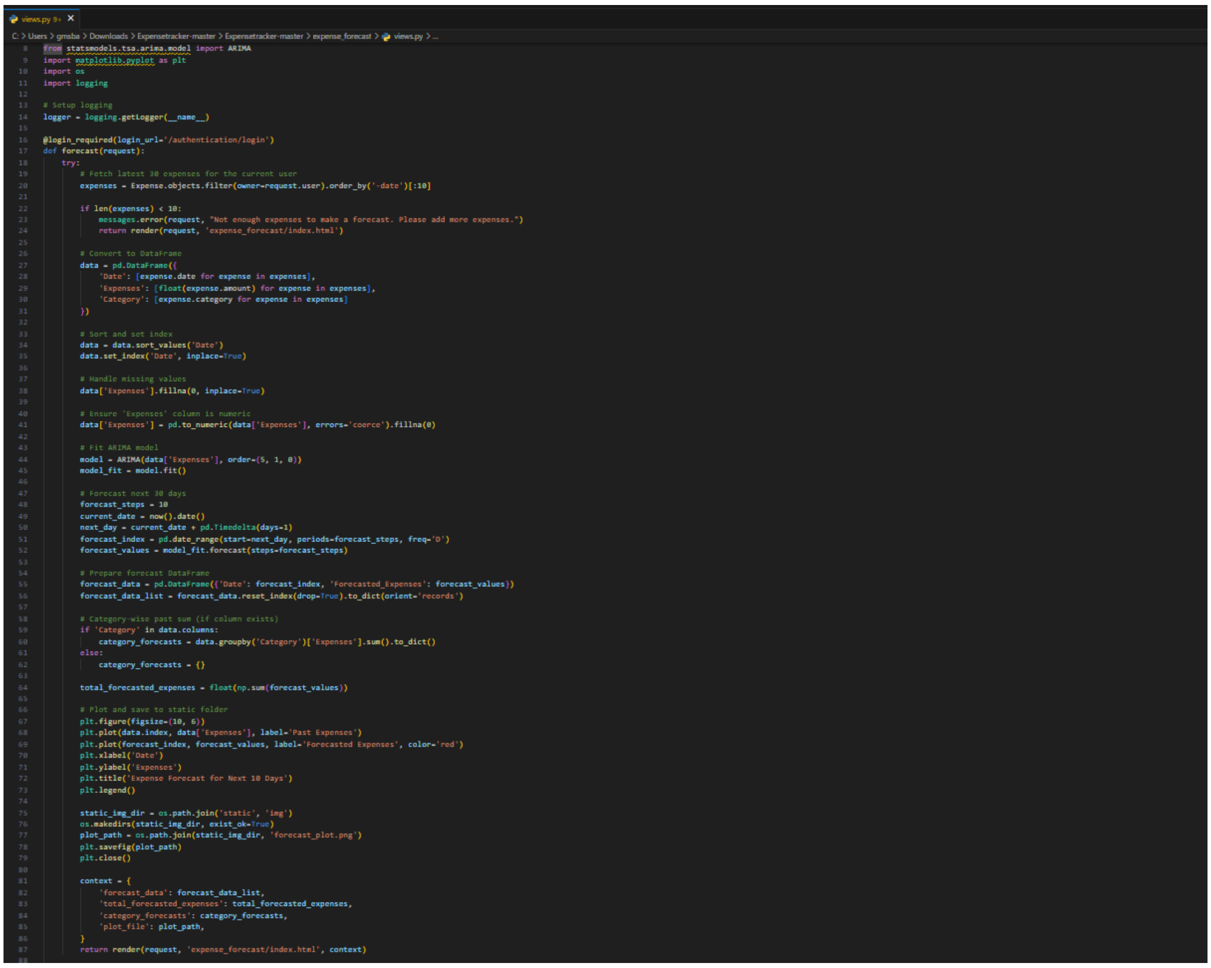
```python

        # return redirect('income')


@login_required(login_url='/authentication/login')
def income_edit(request, id):
    income = UserIncome.objects.get(pk=id)
    sources = Source.objects.all()
    context = {
        'income': income,
        'values': income,
        'sources': sources
    }
    if request.method == 'GET':
        return render(request, 'income/edit_income.html', context)
    if request.method == 'POST':
        amount = request.POST['amount']
        date_str = request.POST.get('income_date')

        if not amount:
            messages.error(request, 'Amount is required')
            return render(request, 'income/edit_income.html', context)
        description = request.POST['description']
        date = request.POST['income_date']
        source = request.POST['source']

        if not description:
            messages.error(request, 'description is required')
            return render(request, 'income/edit_income.html', context)

        try:
            # Convert the date string to a datetime object and validate the date
            date = datetime.strptime(date_str, '%Y-%m-%d').date()
            today = datetime.now().date()


            if date > today:
                messages.error(request, 'Date cannot be in the future')
                return render(request, 'income/edit_income.html', context)
                # return redirect('edit_income', context)

            income.amount = amount
            income. date = date
            income.source = source
            income.description = description
            income.save()
            messages.success(request, 'Income saved successfully')

            return redirect('income')
        except ValueError:
            messages.error(request, 'Invalid date format')
            return render(request, 'income/edit_income.html', context)
        # income.amount = amount
        # income. date = date
        # income.source = source
```



```python
from datetime import datetime

def monthly_income_data(request):
    # Get the current year
    current_year = datetime.now().year

    # Initialize a list to store monthly income data for the current year
    monthly_income_data = [0] * 12  # Initialize with zeros for 12 months

    # Get the monthly income data for the current year, grouped by month
    monthly_data = (
        UserIncome.objects
        .filter(date__year=current_year)  # Filter for the current year
        .annotate(month=ExtractMonth('date'))
        .values('month')
        .annotate(total_income=Sum('amount'))
        .order_by('month')
    )

    # Populate the list with the income data
    for item in monthly_data:
        month_index = item['month'] - 1  # Subtract 1 to convert month to zero-based index
        monthly_income_data[month_index] = item['total_income']

    # Return the data as JSON
    return JsonResponse({'monthly_income_data': monthly_income_data})







@login_required(login_url='/authentication/login')
def get_monthly_income(request):
    today = date.today()
    first_day_of_year = today.replace(month=1, day=1)
    last_day_of_year = today.replace(month=12, day=31)

    # Create a list to hold income data for all 12 months
    monthly_data = [0] * 12

    # Retrieve and fill in the actual monthly income data
    income_data = UserIncome.objects.filter(
        date__range=(first_day_of_year, last_day_of_year),
        owner=request.user
    ).values('date', 'amount')

    for entry in income_data:
        month = entry['date'].month - 1  # Convert month (1-12) to index (0-11)
        monthly_data[month] = entry['amount']

    return JsonResponse({'monthly_data': monthly_data})

```



```python
def generate_report(request):

        if start_date > end_date:
            messages.error(request, "Start date cannot be greater than end date.")
            return redirect('report')

        # incomes = UserIncome.objects.filter(date__range=[start_date, end_date])
        # expenses = Expense.objects.filter(date__range=[start_date, end_date])

        incomes = UserIncome.objects.filter(owner=user, date__range=[start_date, end_date])
        expenses = Expense.objects.filter(owner=user, date__range=[start_date, end_date])

        total_income = incomes.aggregate(Sum('amount'))['amount__sum'] or 0
        total_expense = expenses.aggregate(Sum('amount'))['amount__sum'] or 0

        savings = total_income - total_expense

        context = {
            'incomes': incomes,
            'expenses': expenses,
            'total_income': total_income,
            'total_expense': total_expense,
            'savings': savings,
            'start_date': start_date,
            'end_date': end_date,
            'report_generated':report_generated
        }

        return render(request, 'income/report.html', context)
    else:

        return render(request, 'income/report.html')

def export_csv(request):
    start_date = request.GET.get('start_date')
    end_date = request.GET.get('end_date')
    incomes = UserIncome.objects.filter(date__range=[start_date, end_date])
    expenses = Expense.objects.filter(date__range=[start_date, end_date])

    response = HttpResponse(content_type='text/csv')
    response['Content-Disposition'] = f'attachment; filename="report_{start_date}_to_{end_date}.csv'

    writer = csv.writer(response)

    # Label the income section
    writer.writerow(['Income'])
    writer.writerow(['Date', 'Source', 'Amount'])

    income_total = 0
    for income in incomes:
        writer.writerow([income.date, income.source, income.amount])
        income_total += income.amount

    # Display the total income
    writer.writerow(['', f'Total Income: {income_total}'])
```


views.py 9+
C: > Users > gmsba > Downloads > Expensetracker-master > Expensetracker-master > expense_forecast > views.py > ...

```python
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt
import os
import logging

# Setup logging
logger = logging.getLogger(__name__)

@login_required(login_url='/authentication/login')
def forecast(request):
    try:
        # Fetch latest 30 expenses for the current user
        expenses = Expense.objects.filter(owner=request.user).order_by('-date')[:10]

        if len(expenses) < 10:
            messages.error(request, "Not enough expenses to make a forecast. Please add more expenses.")
            return render(request, 'expense_forecast/index.html')

        # Convert to DataFrame
        data = pd.DataFrame({
            'Date': [expense.date for expense in expenses],
            'Expenses': [float(expense.amount) for expense in expenses],
            'Category': [expense.category for expense in expenses]
        })

        # Sort and set index
        data = data.sort_values('Date')
        data.set_index('Date', inplace=True)

        # Handle missing values
        data['Expenses'].fillna(0, inplace=True)

        # Ensure 'Expenses' column is numeric
        data['Expenses'] = pd.to_numeric(data['Expenses'], errors='coerce').fillna(0)

        # Fit ARIMA model
        model = ARIMA(data['Expenses'], order=(5, 1, 0))
        model_fit = model.fit()

        # Forecast next 30 days
        forecast_steps = 10
        current_date = now().date()
        next_day = current_date + pd.Timedelta(days=1)
        forecast_index = pd.date_range(start=next_day, periods=forecast_steps, freq='D')
        forecast_values = model_fit.forecast(steps=forecast_steps)

        # Prepare forecast DataFrame
        forecast_data = pd.DataFrame({'Date': forecast_index, 'Forecasted_Expenses': forecast_values})
        forecast_data_list = forecast_data.reset_index(drop=True).to_dict(orient='records')

        # Category-wise past sum (if column exists)
        if 'Category' in data.columns:
            category_forecasts = data.groupby('Category')['Expenses'].sum().to_dict()
        else:
            category_forecasts = {}

        total_forecasted_expenses = float(np.sum(forecast_values))

        # Plot and save to static folder
        plt.figure(figsize=(10, 6))
        plt.plot(data.index, data['Expenses'], label='Past Expenses')
        plt.plot(forecast_index, forecast_values, label='Forecasted Expenses', color='red')
        plt.xlabel('Date')
        plt.ylabel('Expenses')
        plt.title('Expense Forecast for Next 10 Days')
        plt.legend()

        static_img_dir = os.path.join('static', 'img')
        os.makedirs(static_img_dir, exist_ok=True)
        plot_path = os.path.join(static_img_dir, 'forecast_plot.png')
        plt.savefig(plot_path)
        plt.close()

        context = {
            'forecast_data': forecast_data_list,
            'total_forecasted_expenses': total_forecasted_expenses,
            'category_forecasts': category_forecasts,
            'plot_file': plot_path,
        }
        return render(request, 'expense_forecast/index.html', context)
```