National Academy of Science and Technology

(Affiliated to Pokhara University)

Accredited by University Grants Commission (UGC), Nepal (2022)

Uttarbehedi 4 Dhangadhi, Kailali, Nepal

A

Project Report

On

Vehicle Rental Services

For the partial fulfillment of requirements for the degree of Bachelor of Computer Engineering Under Pokhara University

Submitted to

Department of Computer Engineering

National Academy of Science and Technology

Under the Supervision of

Mr.Sunil Bahadur Bist

Lecturer, Department of Computer Engineering

Submitted by

Arjun Chaudhary (20070237)

Sagar Chaudhary (20070263)

Sangam Chaudhary (20070264)

Saroj Kumar Dangaura (20070266)

BE. Computer, 8th Semester

July, 2024

COPYRIGHT

DECLERATION

We, Arjun Chaudhary , Sagar Chaudhary , Sangam Chaudhary and Saroj Kumar Dangaura students of Computer Engineering ,semester 8th, National Academy of Science and Technology college affiliated to Pokhara University, hereby declare that the project work entitled "Vehicle Rental Service" is the original place of work under the supervision of Mr. Sunil Bahadur Bist, Department of Computer Engineering and submitted in partial fulfillment of the requirement of Bachelor of Computer Engineering. This project report work has not been submitted by other university or institution for the award of any degree or diploma.

…….. ……..……

Arjun Chaudhary (20070237) Sagar Chaudhary(20070263)

Date: Date:

…….. ……

Sangam Chaudhary (20070264) Saroj Kumar Dangaura (20070266)

Date: Date:

ACKNOWLEDGEMENT

The project has been benefited from the joint efforts of many individuals. It has been a pleasure for us to acknowledge the assistance and contributions that were very important and supportive throughout the project. We would like to extend our sincere thanks to all of them. We would also like to thank **Department of Computer Engineering and Head of Department, Er. Ravi Khadka** for giving us permission to develop a project in this instance. We would like to provide our special thanks to **Mr. Sunil Bahadur Bist**, Supervisor for providing technical support with enormous suggestions during project development.

Furthermore, we wish to express heartfelt gratitude to our teachers, classmates and families who have always served as the strongest source of inspiration and have been knowingly the part of this project and lent support and views during the entire development time.

Project Team

Arjun Chaudhary (20070237)

Sagar Chaudhary (20070263

Sangam Chaudhary (20070264)

Saroj Kumar Dangaura (20070266)

ABSTRACT

In the age of digital era, mobile applications have become tool in revolutionizing various industries, businesses, including vehicle rental services. This Report outlines the development of mobile app focusing mainly for improving the rental experiences, managed operations and customer engagement within the vehicle rental services sector.

The Primary objectives of this project are to design and develop user-friendly mobile app that smooth booking process, real time vehicle availability tracking and secure payment options. It also includes robust communication channel between users and rental providers for instant support and feedback.

The key features of the project mobile app include easy user interface with personalized profiles, GPS navigation for convenient pick-up and drop-off locations and digital documentation for rental agreements. Furthermore, this project focuses the significance of data security and privacy compliance to protect sensitive user information.

Overall, this mobile app project represents a positive step towards embracing innovation, digitalization and customer focus in the vehicle rental services.

Keywords:- Vehicle rent, Vehicle Rental Services, Rent services, Vehicle Rent App.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATION

API: Application Programming Interface

BE: Bachelor in Engineering

CRUD: Create Read Update and Delete

DB: Database

ER-Diagram: Entity Relationship Diagram

Er: Engineer

GUI: Graphical User Interface

HTML: Hyper Text Markup Language

IDE: Integrated Development Environment

JS: Java Script

MVC: Model View Controller

MySQL: My Structure Query Language

OS: Operating System

SQL: Structure Query Language

VS-Code: Visual Studio Code

CHAPTER 1
INTRODUCTION

1.1 Introduction

The vehicle rental services industry has gone a significant transformation in recent year, due to technological advancements. The introduction of a dedicated mobile app for rental services represents how opportunity will meet the demands of modern-day customer.

The Project **Vehicle Rental Services** includes information about vehicle, owner, customer, etc. storing their details in the database. It includes search facility to know the current status and location of each vehicle. It also includes real time vehicle availability tracking and secure payment options. The Vehicle Rental Services is a Mobile Application designed to find vehicle for rent. We used agile methodology in our project. This methodology involves breaking the project into phases and emphasizes collaboration and improvement. We were follow cycle of planning, executing and evaluating for the complete of our project.

This mobile apps user has three role i.e. customer, owner and admin. The Customer will request for signup to take services from rental services and the signup request will approve by admin manually. The owner will add their vehicle in the system. The customer and owner would be same person. Firstly sign up as customer then people can request for owner. Once they became owner they are able to add their vehicle and fix price as they want. For booking, Customer can request for booking any vehicle according to their needs then owner can accept their booking request and conform the booking. After booking conform customer can pick-up the vehicle at pick-up location.

The key features of the project mobile app include easy user interface with personalized profiles, GPS navigation for convenient pick-up and drop-off locations and digital documentation for rental agreements.

1.2 Problem Statement

- Traditional process of vehicle booking
- Paper work to keep record.
- Difficult to find vehicle for rent.

1.3 Objectives

The main objective of this project is:

- To develop a vehicle rental system for vehicle services through digital platform for customer and owner.

1.4 Significance of study

- Customer will take advantages of rental services.
- Time saving
- Better Security

1.5 Scope and Limitations

1.5.1 Scope

- **Vehicle Types**: It has wide range of vehicle types including Bike Car, Trucks, Buses, Tractors, etc.
- **Technological Integration**: Advancements in technology aloe rental services to integrate features like GPS tracking, digital documentation, etc.

1.5.2 Limitations

- **Vehicle Availability:** Specific vehicle types or models may be limited.
- **License Issues:** Valid driving license.

CHAPTER 2
LITERATURE REVIEW

There are several mobile app that are already available in the market.

2.1 Turo

Turo allows individuals to rent out their personal vehicles to others, creating a peer-to-peer car rental marketplace. Turo is an innovative peer-to-peer car-sharing platform that has transformed the way people rent vehicles. Launched in 2009 by Shelby Clark, Turo operates on a simple premise: allowing car owners to rent out their vehicles to others when they're not using them, creating a marketplace that benefits both parties.

Turo offers a diverse selection of vehicles, from everyday cars to luxury, exotic, and specialty vehicles. Renters can choose based on their needs, whether they require a budget-friendly option or a high-end sports car. It provides a robust online platform and mobile app that makes browsing, booking, and managing rentals straightforward. Users can search for vehicles using various filters, including location, price range, vehicle type, and features.

2.2 Bird

Bird is another popular electric scooter rental app that operates in many cities globally. The Bird app, also known simply as Bird, is a popular electric scooter rental service that has gained significant traction in urban transportation solutions since its founding in 2017 by Travis Vander Zanden. Bird operates on the principle of providing convenient, eco-friendly, and affordable last-mile transportation options through its fleet of electric scooters. Bird offers electric scooters that are designed for short-distance travel, making them ideal for urban commuting and last-mile transportation.

The Bird app, available on both iOS and Android, allows users to locate, unlock, and rent electric scooters. It also facilitates payment and provides real-time scooter availability. The app features a map that shows the locations of available scooters and designated parking areas. Local regulations regarding scooter usage, including speed limits and parking rules, must be followed. Users should be aware of and comply with these rules to avoid fines or penalties.

2.3 Zipcar

Zipcar offers a car-sharing service where users can vehicles by the hour or day. Zipcar is a car-sharing service that provides a flexible alternative to traditional car ownership or rental. Founded in 2000 by Antje Danielson and Robin Chase, Zipcar has grown to become one of the largest and most recognizable names in the car-sharing industry. Zipcar has built a community of users who appreciate the convenience and economic benefits of car sharing. It promotes a sharing economy mindset and contributes to reducing congestion and environmental impact in urban areas.

Users can reserve cars through the Zipcar app or website, selecting the vehicle type, location, and rental duration. The app provides features for unlocking and locking vehicles, managing bookings, and viewing rental history. Charges are generally based on hourly or daily rates, with prices including fuel, insurance, and maintenance. Some plans offer reduced rates or additional benefits for frequent users. Rental rates typically include the cost of fuel and basic insurance coverage. Users do not need to worry about refueling the car before returning it, though they are encouraged to refuel if necessary.

2.4 Sparkcar

Sparkcar was established with the goal of making car rentals more accessible and user-friendly, offering an alternative to traditional car rental models. While detailed historical information on Sparkcar's founding is less documented compared to more prominent services, it operates with similar principles to other car-sharing platforms. It offers car rental for various purposes, including self-drive options and vehicles for weddings with drivers. Sparkcar offers a variety of car rental services, including:

- Rental of vehicles for weddings, ranging from economy cars to luxury vehicles.
- Premium electric car rentals in some Indian cities and Toronto.
- General car rentals in Nepal, including services with drivers.

Overall, Sparkcar appears to be a good option for those looking to rent a car in Nepal, particularly for weddings or if you'd prefer an electric vehicle. It's important to note that since their website focuses on Nepal. It is typically available in various urban and suburban areas, with coverage depending on regional demand and partnerships.

CHAPTER 3
METHODOLOGY

The Vehicle Rental Services is a Mobile Application designed to find vehicle for rent. We used agile methodology in our project. This methodology involves breaking the project into phases and emphasizes collaboration and improvement. We were follow cycle of planning, executing and evaluating for the complete of our project.

To a design Vehicle Rental Services, we are following these steps:

3.1 Agile Methodology

Agile methodology is a project management and product development approach that emphasizes flexibility, collaboration, and customer feedback. Agile is guided by the Agile Manifesto, which consists of four fundamental values and twelve principles. The core values are:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

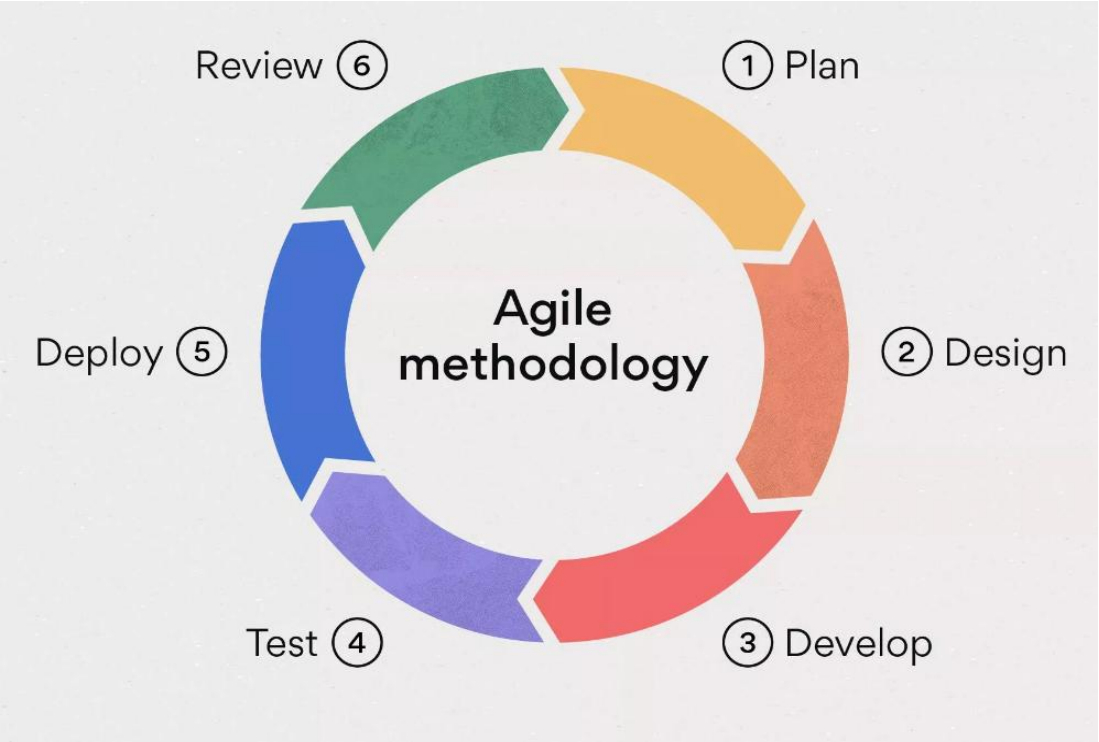


Figure 3. 1 Agile Methdology

3.2 Initial Research and Planning

The vehicle rental services industry has gone a significant transformation in recent year, due to technological advancements. The introduction of a dedicated mobile app for rental services represents how opportunity will meet the demands of modern-day customer. The Vehicle Rental Services was a Mobile Application designed to find vehicle for rent. We used agile methodology in our project. This methodology involves breaking the project into phases and emphasizes collaboration and improvement.

Study the vehicle rental industry to understand current trends, market size, and growth potential. Look into different segments like short-term rentals, long-term leases, or peer-to-peer rentals. Analyze existing vehicle rental systems and platforms. Identify their strengths, weaknesses, and unique features. Examine user reviews and feedback to spot areas for improvement.

By exploring different Websites and Mobile Application we had collect different ideas that were help us in developing this Mobile app. The Websites and Mobile App are Turo, Bird SparkCar, etc. The details about these Websites and Mobile app are given above in Literature Review.

Initial research for a vehicle renting services involves a thorough understanding of the market, users, technology, and regulatory environment. By focusing on these areas, you can develop a robust plan that addresses user needs, leverages current technology, and navigates regulatory requirements effectively. This foundational research will set the stage for a successful project and help in creating a vehicle renting system that is both functional and competitive.

3.2.1 Schedule Feasibility

Our team contains four members. Each member did more hours of work per day. In one week our total working hours was 25 hours. In this way, in 8 week we did 200 hours of work.

Our project includes the following schedule task:

Task	May 2024			June 2024				July 2024			
	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4
Problem Identification	■										
Requirement Collection	■										
System Design		■									
Proposal Defense			■								
Coding & Testing				■	■	■	■	■	■	■	
Mid-Term Defense							■				
Maintenance										■	
Documentation	■	■	■	■	■	■	■	■	■	■	■
Final Defense											■

Table 3.2. 1 Schedule

3.2.2 Technical Feasibility

The technical feasibility determination should be made to determine whether or not the system can be developed by using existing technology. Our team worked hard hence we didn't face much problem in this department. The technical feasibility area can be divided into 2 sections: Hardware and Software.

Hardware Recommendations

- Laptop
- RAM (Minimum 8 GB recommended)
- SSD (Minimum 256 GB recommnded)

Software Recommendations

- Operating system (64 bit or latest)
- Java
- MySQL
- IDE (Environment)

3.2.3 Economic Feasibility

Economic feasibility is a kind of cost-benefit analysis of the examined project, which assesses whether it is possible to implement it. For any system if the expected benefits equal or exceed the expected costs, the system can be judged to be economically feasible. During the project, we analyzed equipment cost, operating cost, development cost, etc.

S.N.	Elements	Price
1.	Hosting	Rs. 3485
2.	Domain	Rs. 1345
3.	Documentations	Rs. 3000
	Total	Rs. 7830

Table 3.2. 3 Economic Feasibility

3.3 System Design

The vehicle rental system is designed to facilitate the process of renting vehicles to customers efficiently. It involves managing vehicle availability, processing reservations, handling payments, and ensuring proper vehicle maintenance. The system should be scalable, reliable, and secure to handle varying loads and protect sensitive information. It allows users to create accounts, log in, and manage their profiles. There are different roles for customers, administrators, and owner with varying permissions.

3.3.1 System Design

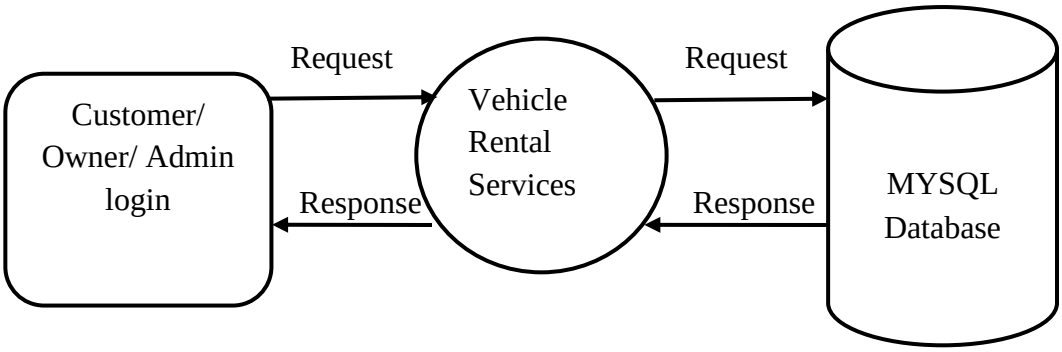


Figure 3.3. 1 System Design

3.3.2 Zero Level Data Flow Diagram

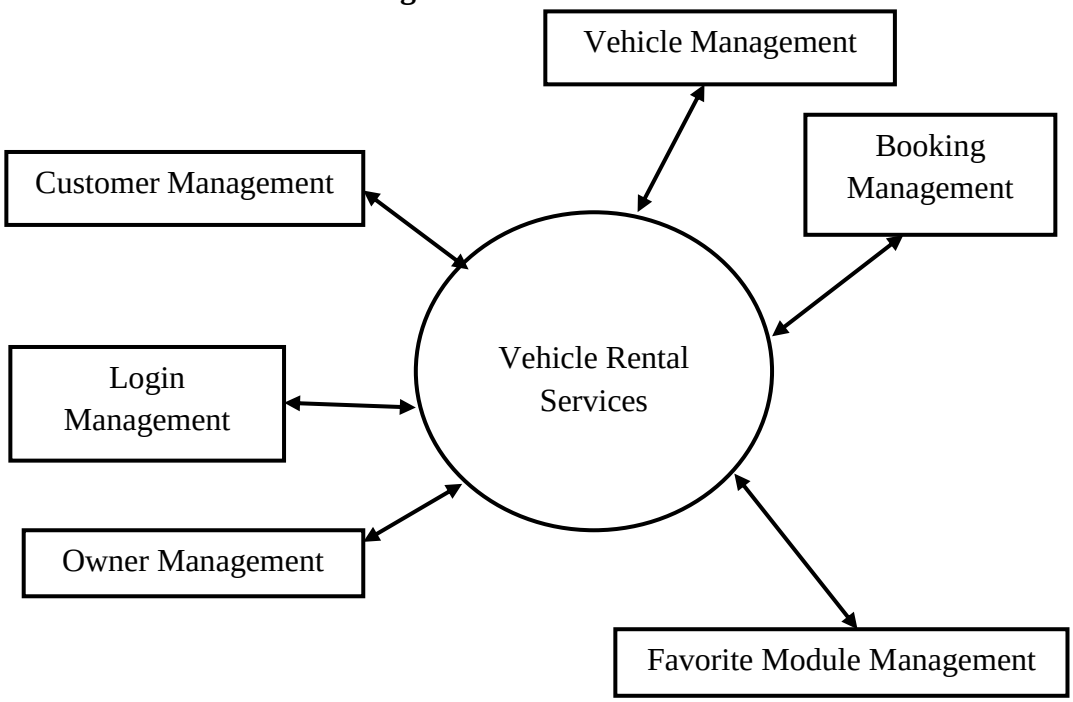


Figure 3.3. 2 Zero-Level DFD

3.3.3 One Level Data Flow Diagram

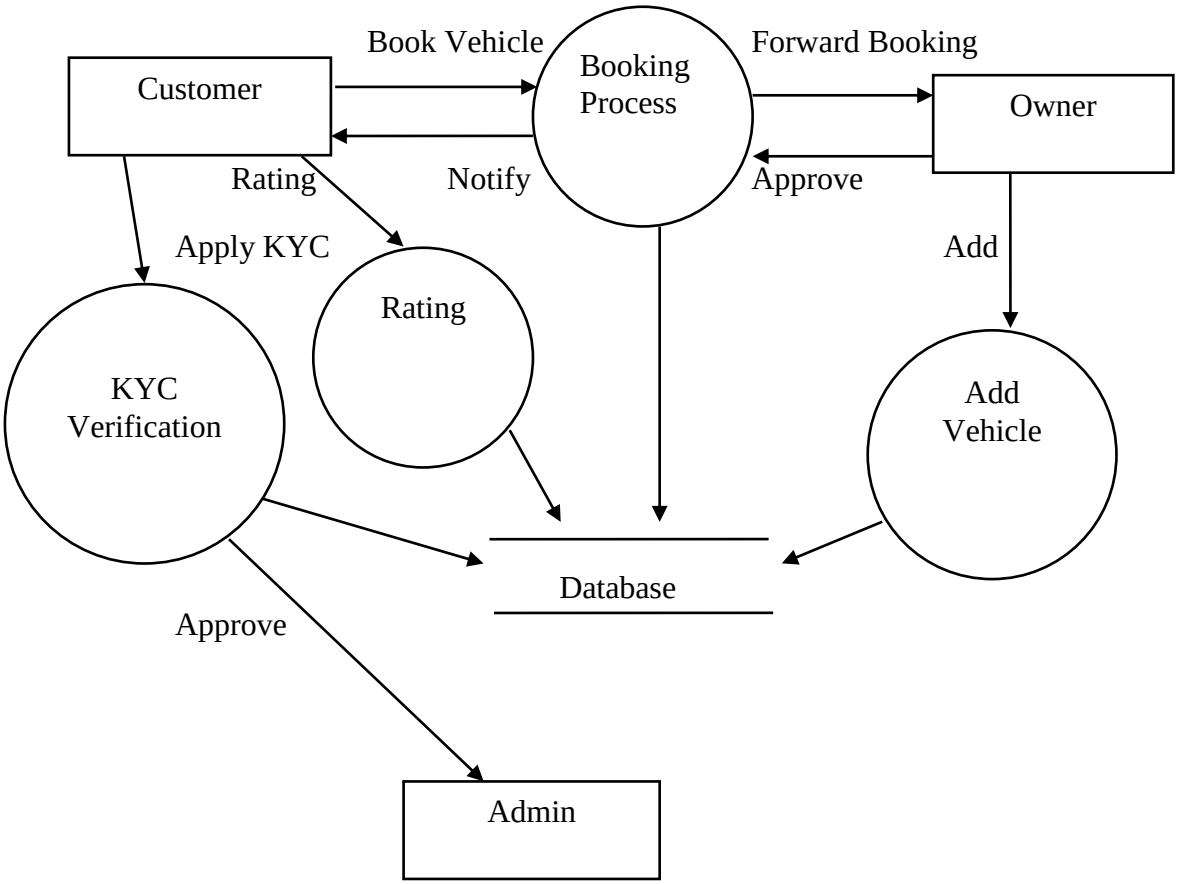


Figure 3.3. 3 1-Level DFD

3.3.4 Flow Chart

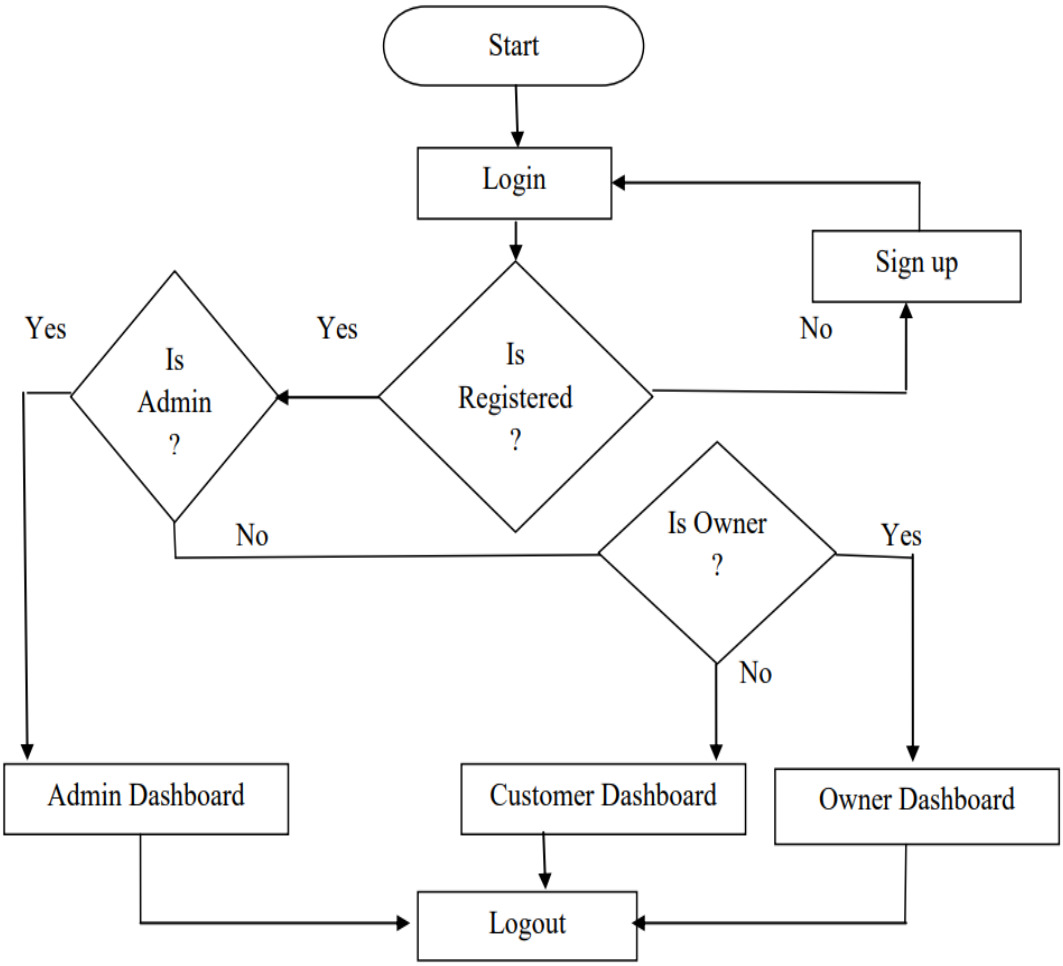


Figure 3.3. 4 Flow chart

3.3.5 Schema Diagram

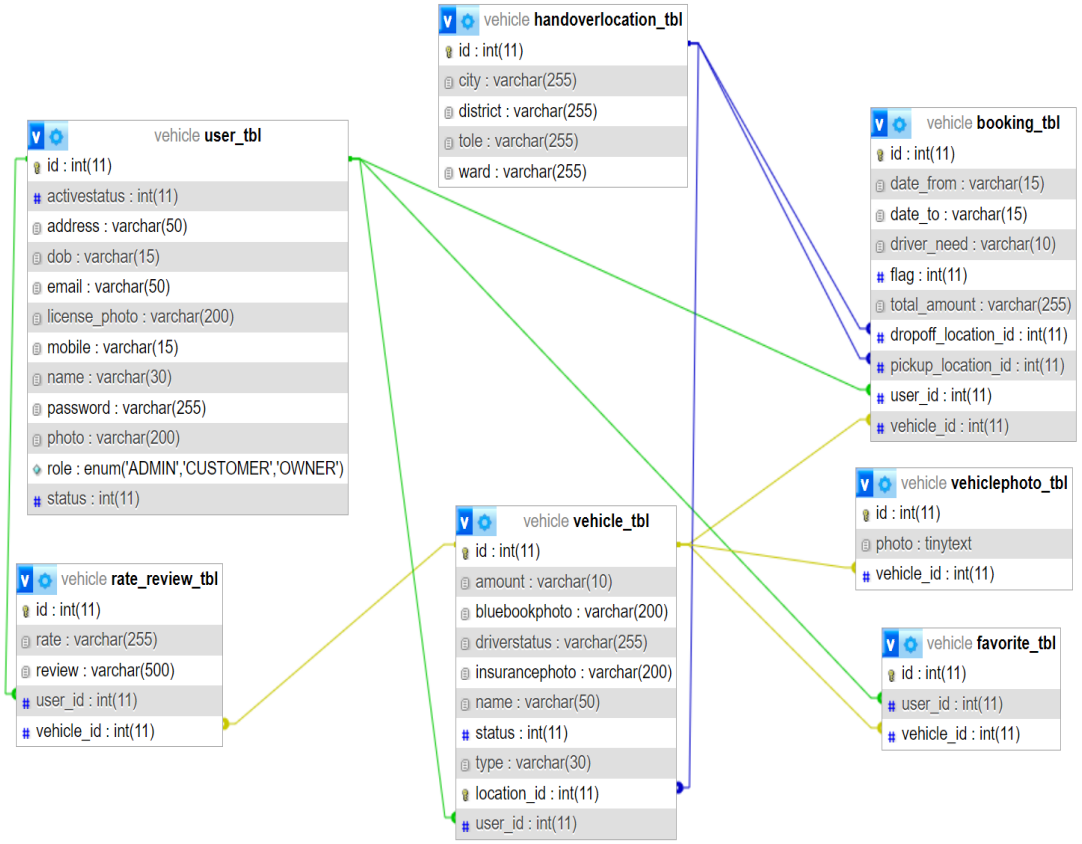


Figure 3.3. 5 Schema Diagram

3.3.6 Use Case Diagram

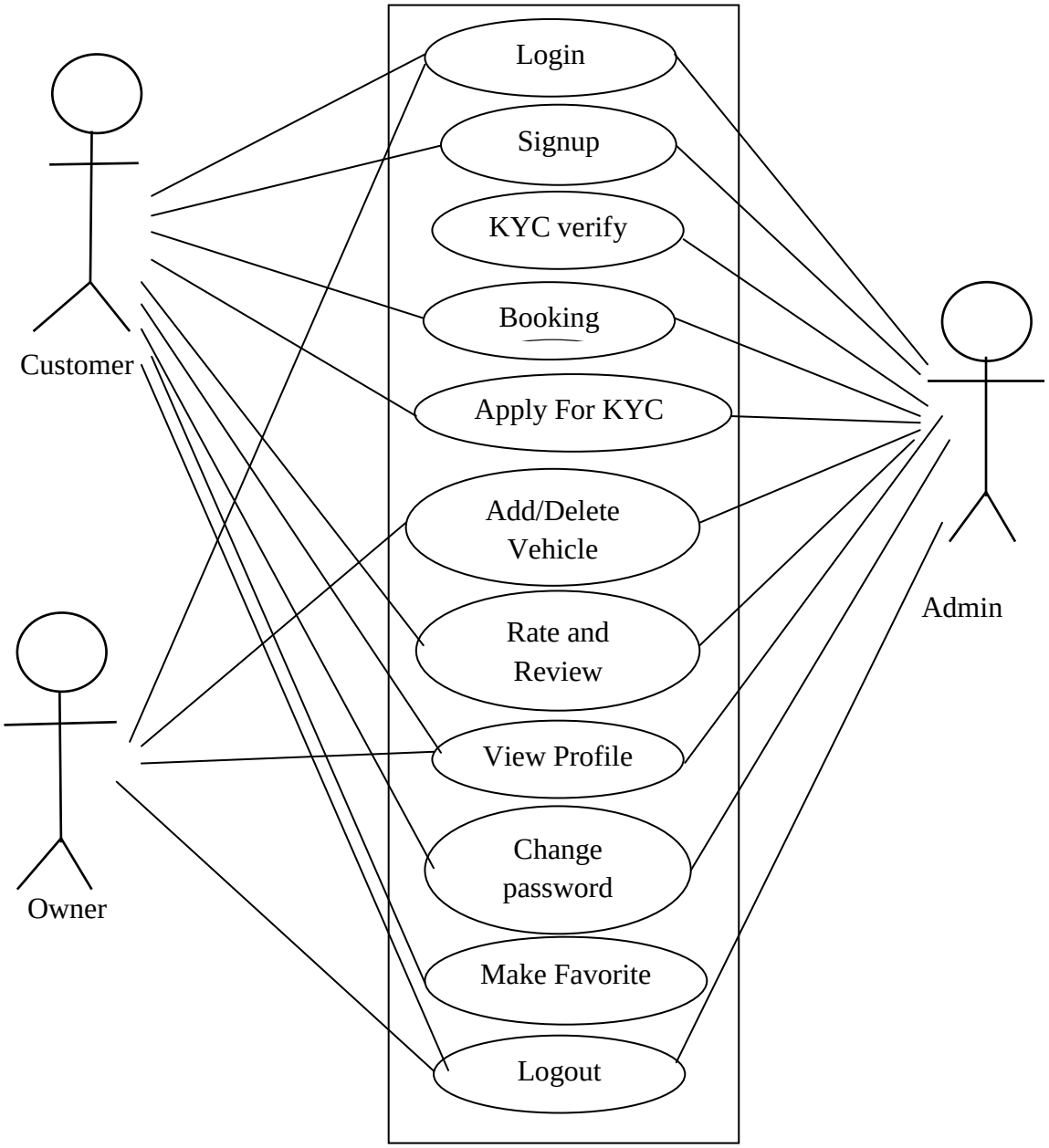


Figure 3.3. 6 Use case diagram

3.4 Design and Development

The Vehicle Rental Services is a Mobile Application designed to find vehicle for rent. We used agile methodology in our project. This methodology involves breaking the project into phases and emphasizes collaboration and improvement. We were follow cycle of planning, executing and evaluating to complete of our project.

The Front end development was done by using Android Studio with java programming language. Volley Library is use to call Rest APIs. The Rest APIs was developed by using Spring Boot Framework which is java's Framework in Eclipse IDE. The MySQL Database is used in this project.

The Login, Sign Up, Dashboard (Home), Vehicle adding, Vehicle Booking, Approve Booking, User Profile, KYC Section, Help Support, Privacy and Terms, Change Password, etc are the UI design. In Rest API includes the module like User module, Vehicle module, Booking Module, Vehicle Photo module, Location module, Favorite module, Rate and Review module are used in this project.

3.4.1 Module List

1. Login Module
2. Signup Module
3. Admin Module
4. Owner Module
5. Customer Module
6. Vehicle Module
7. Booking Module
8. Favorite Module

1. Login Module

A login module is a fundamental component of many software applications and websites, designed to authenticate users and grant them access to the system based on their credentials. The primary function of a login module is to verify a user's identity by checking their credentials—usually a username and password—against stored data. It ensures that only authorized users can access certain features or data. Once

authenticated, the module may also handle authorization, determining what actions the user is permitted to perform within the application.

This includes the login form where users input their credentials. Typically, it consists of fields for a username/email and password, along with a "login" button. Some systems may also include additional elements like "Forgot Password" links or "Sign up" buttons.

2. Signup Module

A signup module is an essential component of many applications and websites, enabling new users to create an account and gain access to the system. The signup module allows users to register by providing necessary information, creating a unique account for them in the system. It collects essential details required for user identification and account setup, such as E-mail/Username and Password. Users may need to agree to terms and privacy policies.

Our Mobile app includes fields for entering information such as username, email address, password, and details like phone number, address. It also contains Signup button and terms and condition button to complete the registration process. A signup module is crucial for user involving collecting and validating user information, ensuring security, and providing a smooth user experience.

3. Admin Module

An admin module is a critical component in applications and websites designed for system administrators or managers who need to oversee and control various aspects of the system. This module provides tools and interfaces to manage users, settings, data, and other system functionalities. The admin module is designed to give administrators control over the application, including user management, system configuration, and monitoring. It provides the capability to manage system resources, review and manage content, and ensure the system runs smoothly.

It provides tools to verify user accounts in our mobile application. It also provides services to approve or reject content before it becomes publicly visible.

4. Owner Module

An owner module is a specialized component of an application or system tailored for individuals or entities who own or have the highest level of authority within a platform. This module typically provides comprehensive tools for overseeing and managing various aspects of the system, often with more extensive control and capabilities compared to standard admin functions.

Our mobile application provide services as a owner and he or she has an permission to add vehicles, and have access to explore all vehicles and vehicles details. He/she can have access of exploring profile, edit details, approved booking request, change password and Logout. An owner must provide vehicle model, type, renting price, vehicle photo, bluebook details and insurance details and so on while adding vehicles.

5. Customer Module

A customer module is an essential part of an application or system designed to manage interactions with customers. It focuses on providing tools and features to enhance the customer experience, streamline service delivery, and handle various customer-related tasks efficiently. The primary goal of the customer module is to facilitate and manage interactions between the business and its customers, ensuring a positive experience and addressing their needs effectively. It helps in delivering services, processing requests, and managing customer accounts, contributing to overall customer satisfaction and retention.

Our mobile application provide services as a customer and he or she has an permission to book vehicles, and have access to explore all vehicles and vehicles details. He/she can have access of exploring profile, edit details, place booking, change password and Logout as well as add vehicle to favorite section. A customer can explore his booking history, search vehicles. A customer must fill KYC including mainly license number and photo and other details. And all filled details are verified by admin.

6. Vehicle Module

Vehicle details encompass a range of information about a vehicle that can be used for various purposes, including registration, maintenance, insurance, and rent. It includes

manufacturer or brand of the vehicle, specific model of the vehicle. An owner must provide vehicle model, type, renting price, vehicle photo, bluebook details and insurance details and so on while adding vehicles.

The UX (User Experience) design of a vehicle details module focuses on creating an intuitive, user-friendly interface for managing and accessing comprehensive information about vehicles. Effective UX design ensures that users—whether they are vehicle owners, potential buyers, or service providers—can easily navigate and interact with the system. The interface should present vehicle information in a clear and organized manner. Key details like make, model, and VIN should be immediately visible, with additional information accessible through well-labeled sections or tabs.

7. Booking Module

A booking module is a crucial component in applications or systems that handle reservations or appointments for various services, such as travel, events, accommodations, or professional services. This module facilitates the process of scheduling and managing bookings, ensuring a smooth and efficient experience for users. The booking module enables users to book services, appointments, or resources, and helps businesses manage these reservations effectively.

Effective UX design focuses on creating an intuitive, seamless, and enjoyable experience for users throughout the booking process. The interface should present booking options clearly. Key elements like booking vehicles and booking forms should be straightforward and easy to understand.

8. Favorite Module

A favorite module is a feature often found in various applications and systems that allows users to mark items, content, or services as favorites for easy access and management. This module enhances user experience by helping users quickly find and manage their preferred or frequently used items. It enables users to personalize their experience by bookmarking or highlighting items they like or use frequently. It provides quick access to favored items, improving navigation and efficiency for users.

3.4.2 Monitoring and Evaluation

- **Performance Monitoring**: Tracked the performance of the implemented solution against predefined metrics and benchmarks.
- **Feedback Collection**: Gathered feedback from users and stakeholders to assess satisfaction and effectiveness.
- **Evaluation Report**: Compiled evaluation results to determine if project goals were met. Analyzed any discrepancies and areas for improvement.

3.4.3 Documentation and Reporting

- **Documentation**: Created detailed documentation of the project processes, methodologies, and outcomes. Ensured that all procedures and results were accurately recorded.
- **Final Report**: Compiled a comprehensive project report, including an executive summary, methodology, findings, conclusions, and recommendations.

3.5 Testing

The manual testing was done by our team member. The UI/UX and working too good and the Rest APIs are also working well. In manual testing, our team member manually reviewing and testing the application and simulating the behavior of real user to identify the error, bugs and other issues.

3.5.1 Manual Testing

Manual testing is a software testing process where test cases are executed manually by testers without the use of automated tools. This approach involves interacting with the software application in the same way a real user would, to identify bugs, ensure functionality, and verify that the software meets its requirements. In manual testing, manually reviewing and testing the application, simulating the behavior of real user to identify the error, bugs and other issues. Its main purposes are to ensure that the software performs its intended functions correctly and meets user requirements. Detect defects or issues in the software that might not be easily identified through automated testing. Assess the software's usability and user experience, ensuring it is intuitive and user-friendly.

In manual testing process first of all define the scope, objectives, and resources required for testing. Create a test plan that outlines the testing strategy, scope, and schedule. Then, develop test cases based on requirements and use cases. Test cases include the test steps, expected results, and any necessary test data. Manually execute the test cases by interacting with the software, following the predefined steps to verify functionality. Document any defects or issues discovered during testing, including detailed information to help developers understand and address the problem. Review and analyze the testing process, document results, and provide feedback on the overall quality of the software.

Best practices are maintain detailed and well-documented test cases, including steps, expected results, and test data to ensure consistency and reproducibility. Review test cases and results regularly to ensure accuracy and effectiveness, and update them based on changes in requirements or application functionality. Maintain clear communication with developers and other stakeholders to ensure that defects are understood and addressed promptly.

Manual testing remains an essential aspect of the software development process, particularly for evaluating user experience, usability, and complex scenarios that automated tests might not fully cover. Despite the rise of automation, manual testing provides valuable insights and ensures that software meets user expectations and quality standards.

3.5.2 Unit Testing

Unit testing is a software testing technique where individual components or "units" of a software application are tested in isolation to ensure they function correctly. A unit is the smallest testable part of an application, typically a single function or method. In object-oriented programming, a unit might be a class or a module. Units are tested in isolation from the rest of the application to ensure that the test results are not influenced by other parts of the code. Dependencies are often mocked or stubbed to achieve this.

Unit tests are usually automated, meaning they are written in code and executed by a testing framework. This allows tests to be run frequently and consistently with minimal manual intervention. Unit testing is a fundamental practice in modern software development that helps ensure the correctness of individual units of code,

supports better design, and aids in the maintenance of high-quality software.

3.5.3 Integration Testing

In this type of testing the components of this project application are gradually integrated and then tested as a unified group. Integration testing is a key phase in the software development lifecycle focused on ensuring that different components or systems work together as expected. It involves testing the interactions between multiple systems or applications, often in a production-like environment. It Validates the complete flow of data and functionality from start to finish, simulating real user scenarios.

Integration testing aims to detect interface defects and ensure that different modules or systems work together seamlessly. By testing integrations early in the development process, teams can identify and address issues before they escalate. All components are integrated simultaneously, and the entire system is tested as a whole. This can be risky as identifying the root cause of failures can be challenging. Tools like Selenium, JUnit, and TestNG can be used to automate integration tests, improving efficiency and consistency. By focusing on how different parts of the system work together, integration testing helps ensure that the final product is robust and reliable, ultimately leading to a more stable and user-friendly application.

3.5.4 White Box Testing

White box testing, also known as clear box testing, glass box testing, or structural testing, is a software testing methodology that examines the internal logic and structure of the code. It is a form of testing that provides the tester with complete knowledge of the application being tested, including access to source code and design documents.

It requires a deep understanding of the code and its logic, which can be complex and time-consuming. As the code changes, the test cases and scripts may need to be updated to remain effective. It focuses on the internal workings of the code, which might not fully address integration issues or interactions with external systems. By focusing on the internal structure and logic of the code, white box testing provides valuable insights into the functioning of a software application, helping to ensure that it performs correctly and efficiently.

It focuses on individual components or functions of the software. It verifies that each unit of the code performs as expected. Tools like JUnit (for Java) or NUnit (for .NET) are often used. It measures how much of the code is executed during testing. Common coverage metrics include statement coverage, branch coverage, and path coverage.

- **Statement Coverage:** Ensures that every line of code is executed at least once.
- **Branch Coverage:** Ensures that every branch (decision point) in the code is tested.
- **Path Coverage:** Ensures that all possible paths through the code are tested.

3.5.5 Black Box Testing

Black box testing is a software testing methodology that evaluates the functionality of an application without any knowledge of its internal code structure or implementation details. This approach focuses on testing the software's external behavior and its interaction with users or other systems based on specified requirements and use cases. It is a form of testing that is performed with no knowledge of internal of the system**.** It can be carried out to evaluate the functionality, security, performance and other aspect of the project application**.**

The primary goal of black box testing is to ensure that the software meets its functional requirements and performs as expected from an end-user perspective. It helps in identifying discrepancies between the actual behavior of the software and its expected behavior, such as bugs or missing functionalities. It divides the input data into equivalent partitions that are expected to produce similar results. This reduces the number of test cases while ensuring coverage. It focuses on testing the boundaries of input values to identify errors at the edges of input ranges, as boundary conditions are often where defects occur.

Testers do not need to understand the internal code structure, making it accessible to non-developers or those with a focus on functionality rather than implementation. It can be used at different levels of testing, including system, integration, and acceptance testing. Tools like JMeter and LoadRunner can be used to test how the software performs under different load conditions, focusing on its responsiveness and stability.

The best practices are to develop a wide range of test scenarios to cover various aspects of the software's functionality and interactions. Ensure that requirements and specifications are well-documented and understood, as these form the basis for creating effective test cases. Regularly review and update test cases based on changes in requirements or application functionality to maintain relevance and effectiveness. Black box testing is crucial for validating that an application functions correctly and meets user expectations, making it an integral part of a comprehensive testing strategy.

3.6 Validation

- Gather feedback from including owner of Vehicle and customer to validate the usability, responsiveness, and reliability of the system in real world scenarios.
- Verify that all functionalities of this system work correctly according to the defined requirements.

3.7 Role and Responsibilities

The details about who is responsible for each part of work are given below:

- **Idea, Research and Planning :** Arjun Chaudhary, Sagar Chaudhary, Sangam Chaudhary and Saroj Kumar Dangaura
- **UI Design :** Arjun Chaudhary, Sagar Chaudhary, Sangam Chaudhary and Saroj Kumar Dangaura
- **UX and Backend :** Saroj Kumar Dangaura
- **Rest APIs :** Saroj Kumar Dangaura
- **Database :** Saroj Kumar Dangaura
- **Manual Testing :** Arjun Chaudhary, Sagar Chaudhary, Sangam Chaudhary and Saroj Kumar Dangaura
- **Documentation :** Sangam Chaudhary and Saroj Kumar Dangaura

3.8 Challenges Encountered

Resource Constraints

- **Description :** Due to their problem, team members were unavailable during critical phases of the project, leading delay in the project.

- **Impact :** This result in 10 day delay in the project timeline, affecting subsequent tasks and deadlines.
- **Resolution :**
    - **Steps Taken**: Reallocated tasks among available team members, brought in temporary support where needed, and adjusted the project schedule to accommodate the delays.

CHAPTER 4
CONCLUSION AND RECOMMENDATION

4.1 Conclusion

The Mobile Application development is never completed. There is always a need for the modifications. There could have been other approaches to implement the app. We have tried to our level best to make the App an interactive as possible. The App has been developed with much care and free of errors and at the same time it is efficient and less time consuming. The purpose of this project was to develop a Mobile Application that is used for searching nearby vehicle and rent it. This app allows user to add their vehicle for rent and earn money for it.

4.2 Recommendation

This Project has given great satisfaction in having designed an application which can be implemented to any vehicle owner and offer vehicle to user for rent it and vehicle owner can earn money for rent it. This project can be more productive and efficient by doing the following function on the existing project. So we recommend to upgrade below mention features for making our Application more reliable and real time service.

- Enhance User Interface by adding more user interactive features.
- Provides Rewards to most rented, most rated vehicle and vehicle owner.
- Payment Options: Online like E-sewa, Ime-pay etc.
- Customize booking like featured vehicle.
- Making a websites is also a first step development.

BIBLIOGAPHY

[1] Turo (2024, May. 11). *User interface Guide for Renting services* [Online]. Available: https://turo.com/gb/en

[2] Self drive Nepal (2024, May. 13). *Guide for Vehicle Rental services* [Online]. Available: https://selfdrivenepal.com/about

[3] Sparkcar (2024, May. 14). *Navigation Guide for Rental System* [Online]. Available: https://sparkcar.org/

[4] Learn Code With Durgesh (2024, May. 20). *Backend code using spring Boot* [Online]. Available: https://www.youtube.com/watch?v=I7BTYi5augU&list=PL0zysOflRCen-GihOcm1hZfYAlwr63K_M

[5] GeeksForGeeks (2024, May. 25). *Create API using Spring Boot* [Online]. Available: https://www.geeksforgeeks.org/easiest-way-to-create-rest-api-using-spring-boot/

[6] Javatpoint (2024, May. 27). *crud operation using spring boot and mysql* [Online]. Available: https://www.javatpoint.com/spring-boot-crud-operations

[7]GeeksForGeeks (2024, May. 29). *crud operation using spring boot and mysql* [Online]. Available: https://www.geeksforgeeks.org/spring-boot-crud-operations-using-mysql-dat

[8] WsCube Tech (2024, June. 02). *Android App Development Course (Beginner to Advance)* [Online]. Available: https://www.youtube.com/watch?v=HyU4vkZ2NB8&list=PLjVLYmrlmjGdDps6HAwOOVoAtBPAgIOXL

[9] GeeksForGeeks (2024, June. 03). *Android App Development and android Layout* [Online]. Available: https://www.geeksforgeeks.org/android-tutorial/

[10]Tutorialspoint (2024, June. 08). *Android user interface design* [Online]. Available: https://www.tutorialspoint.com/android/android_relative_layout.htm

[11] Zipcar (2024, June. 12). *Navigation, Working and User interface design for Vehicle renting system* [Online]. Available: https://www.zipcar.com/

[12] Bird (2024, June. 20). *Working of Vehicle renting services* [Online]. Available: https://www.bird.co/

[13] Md Jamal (2024, July. 02). *Android Volley Library Tutorial for Android Backend* [Online]. Available: https://www.youtube.com/watch?v=Dp7woFp_GXo&list=PLirRGafa75rTh2JevXAqOIEmsZyq0rWJi

[14] GeeksForGeeks (2024, July. 03). *Android Backend code and volley library* [Online]. Available: https://www.geeksforgeeks.org/volley-library-in-android/

ANNEX I

Figure: Annex 1. 1 Splash Screen

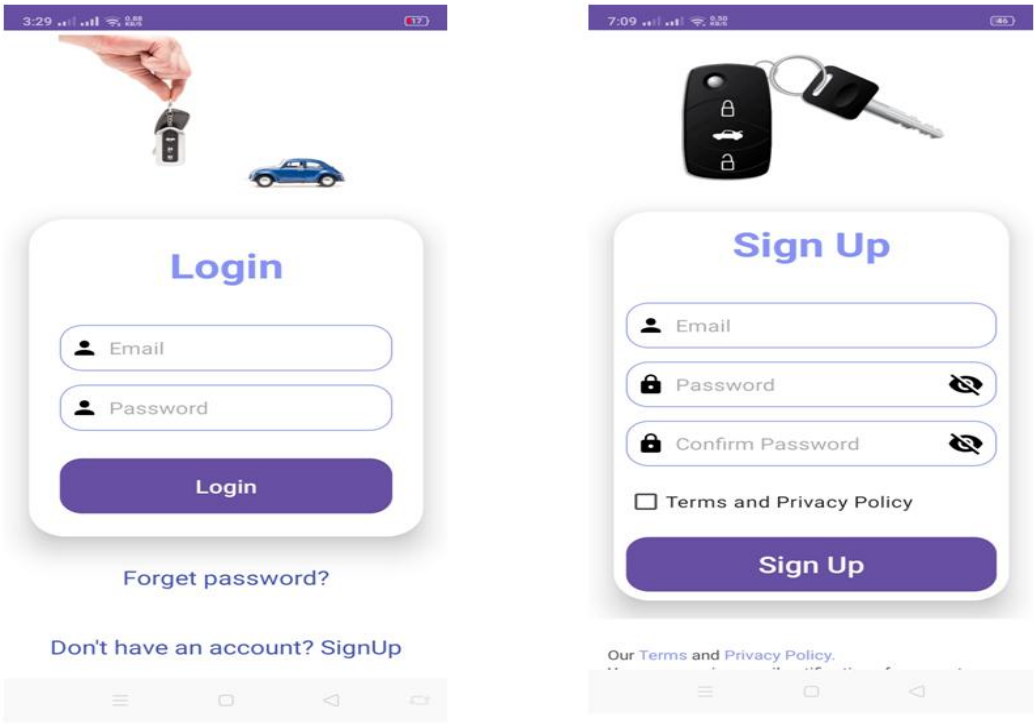


Figure: Annex 1. 2 Login and Signup

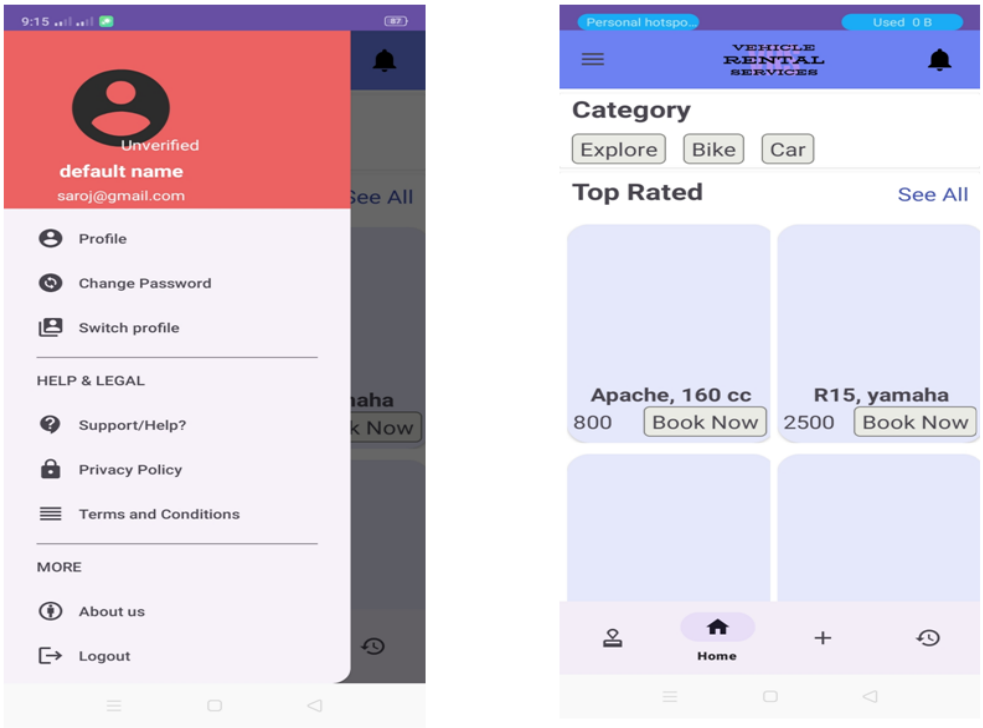


Figure: Annex 1. 3 Navigation Drawer and Home

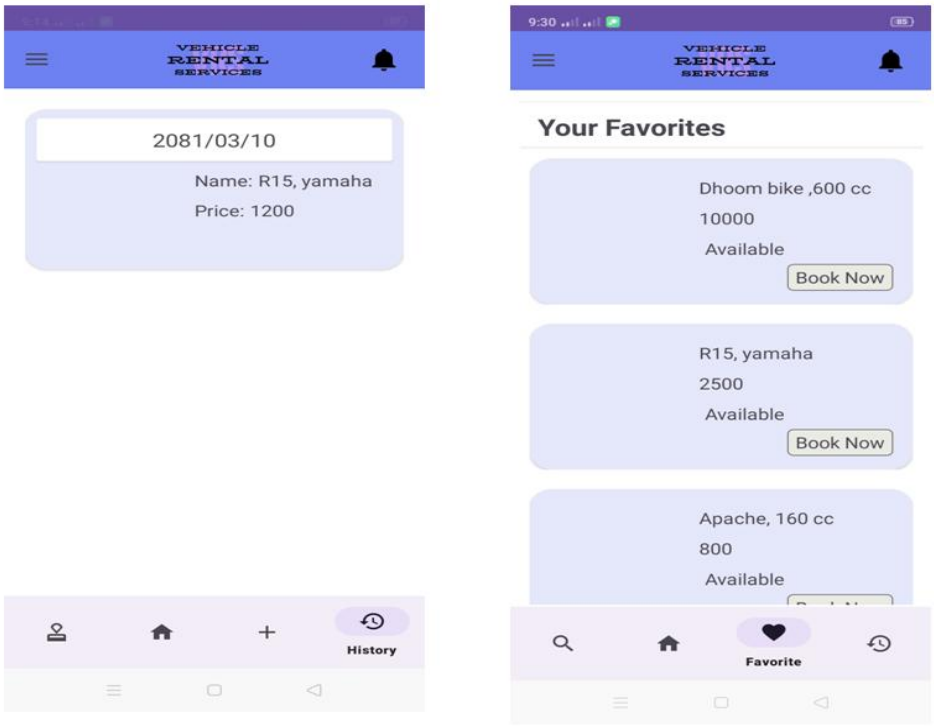


Figure: Annex 1. 4 History and Favorite

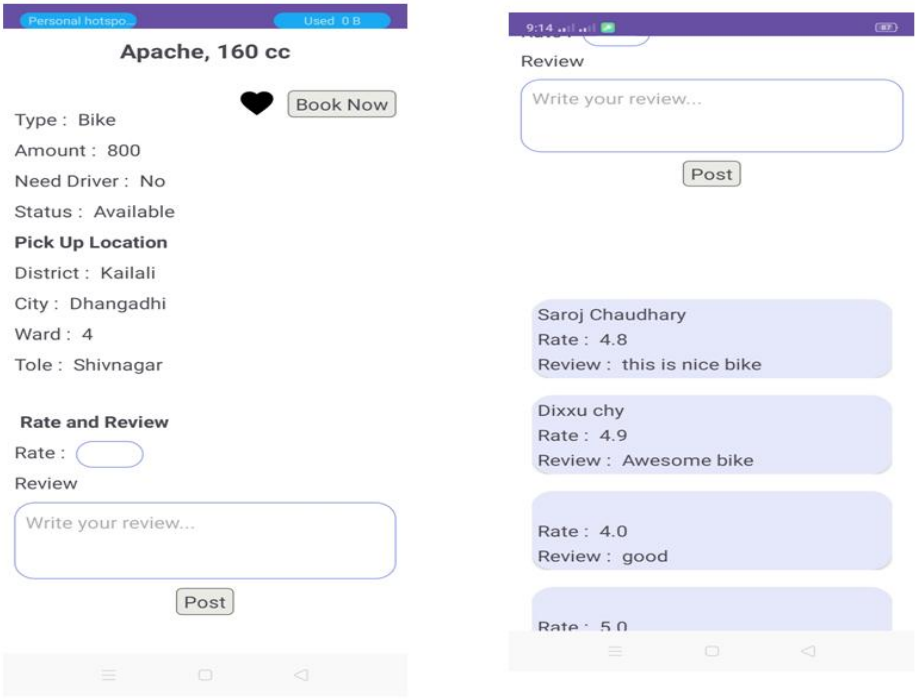


Figure: Annex 1. 5 Vehicle details

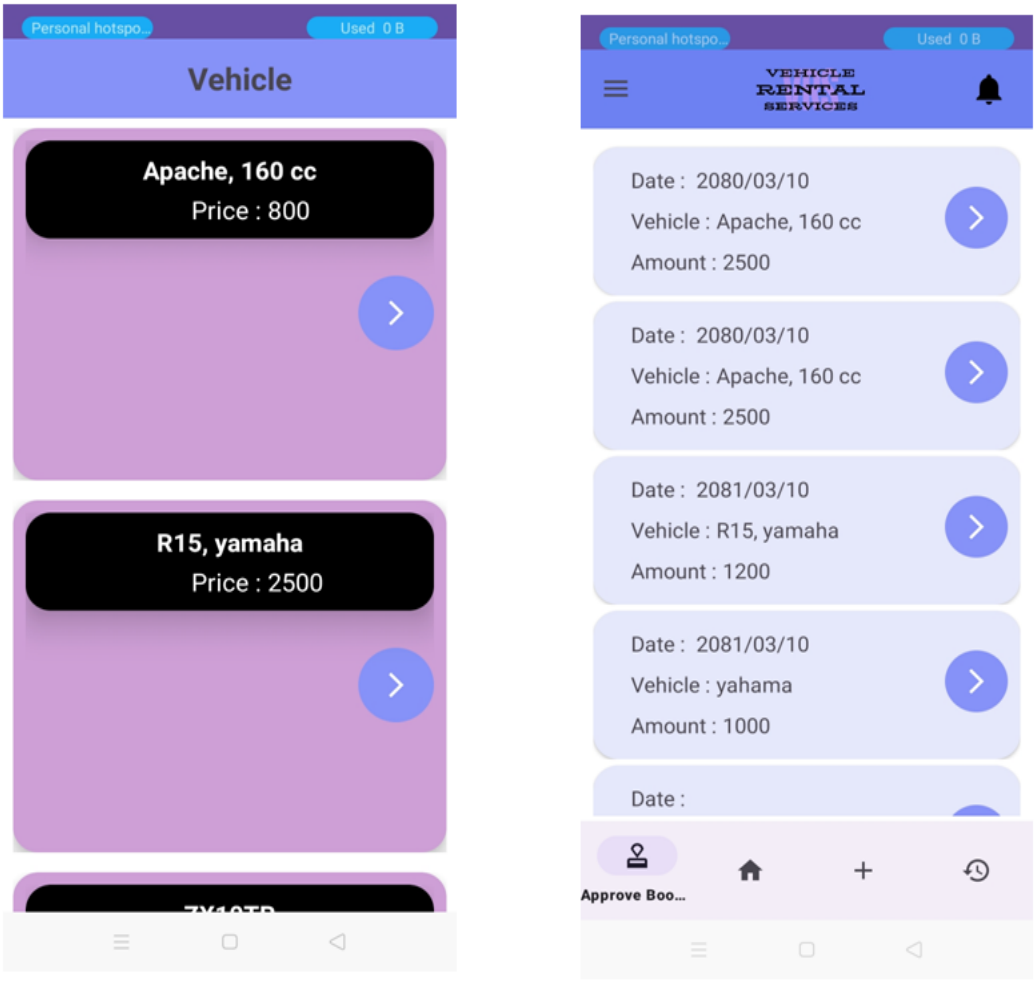
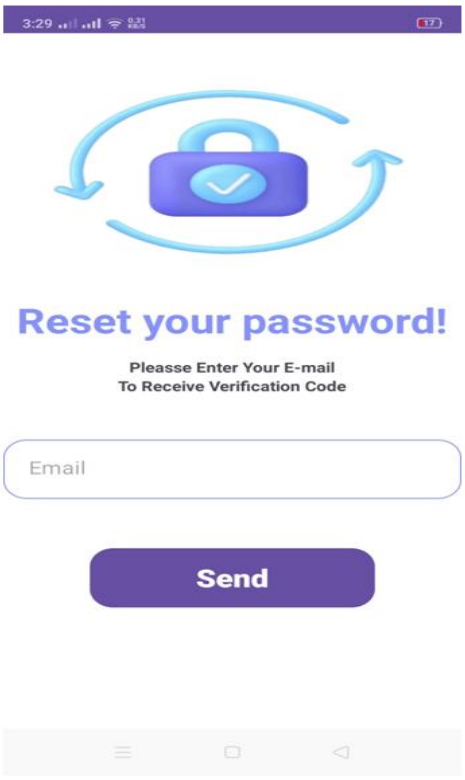
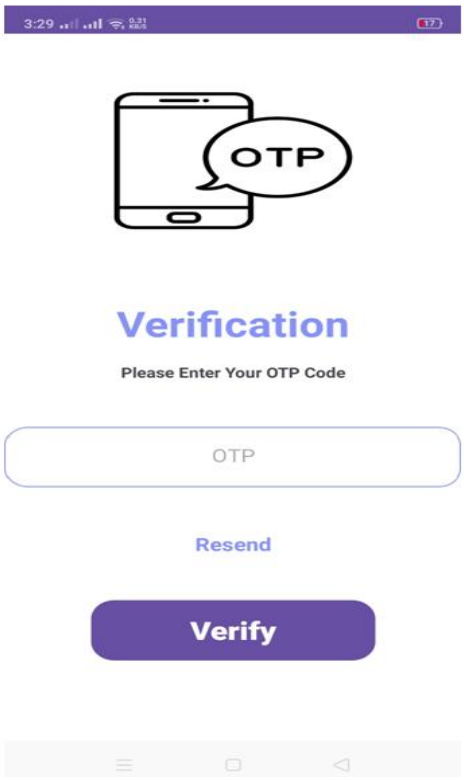


Figure: Annex 1. 6 All Vehicle and Booking Approve





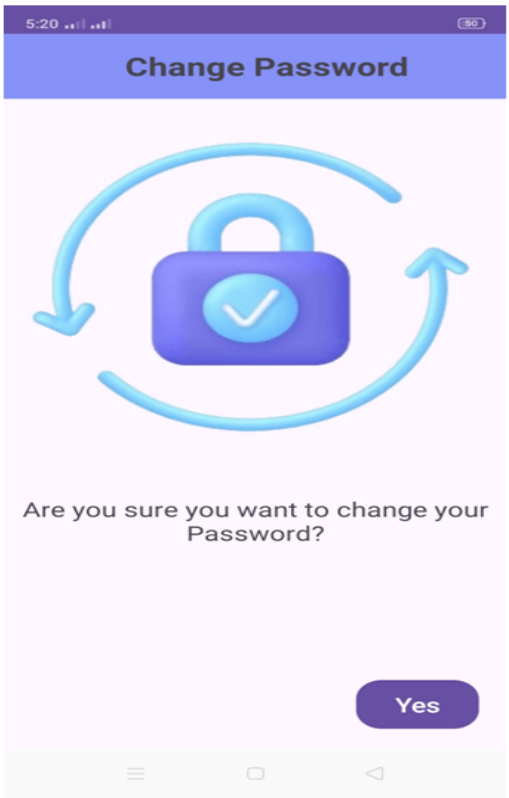


Figure: Annex 1. 7 Change Password

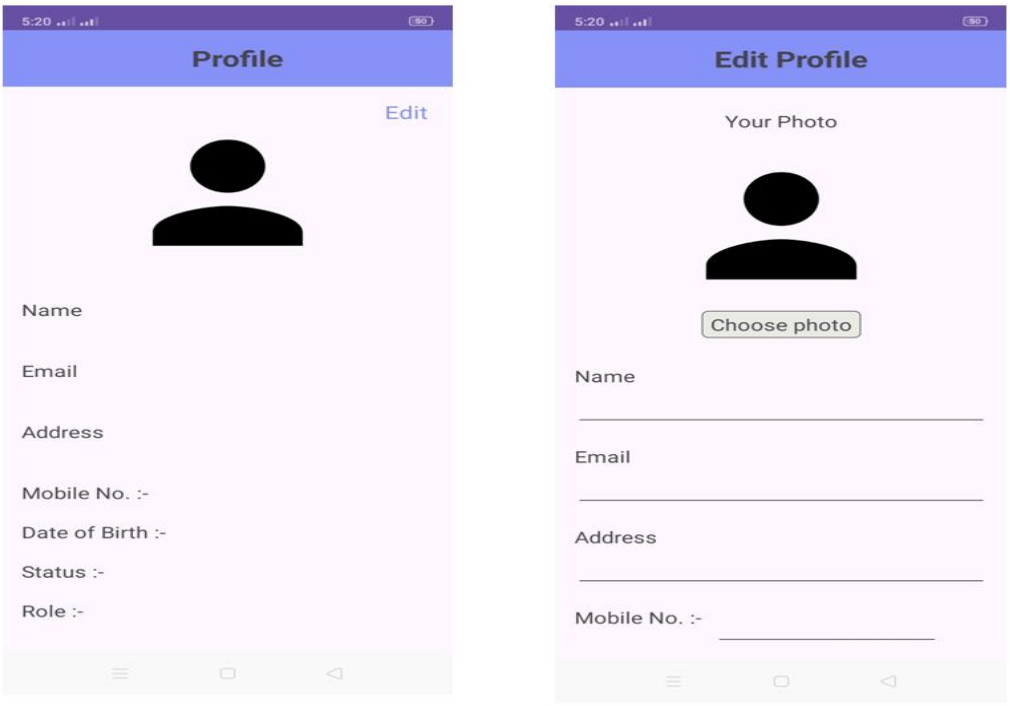


Figure: Annex 1. 8 Profile and Edit Profile

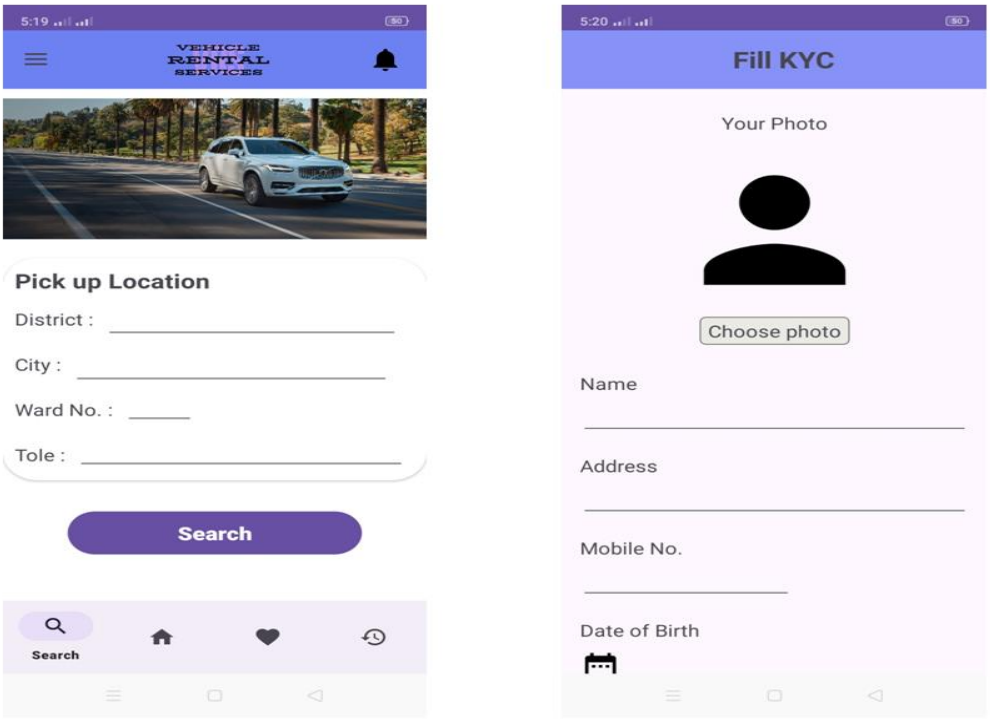


Figure: Annex 1. 9 Search and KYC

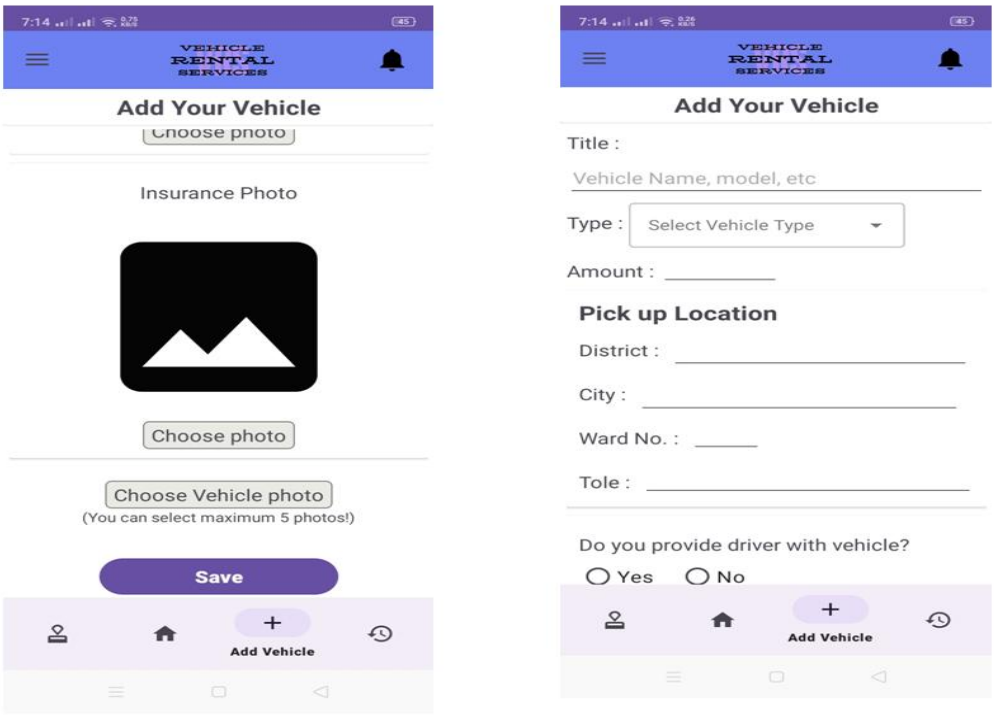


Figure: Annex 1. 10 Add Vehicle

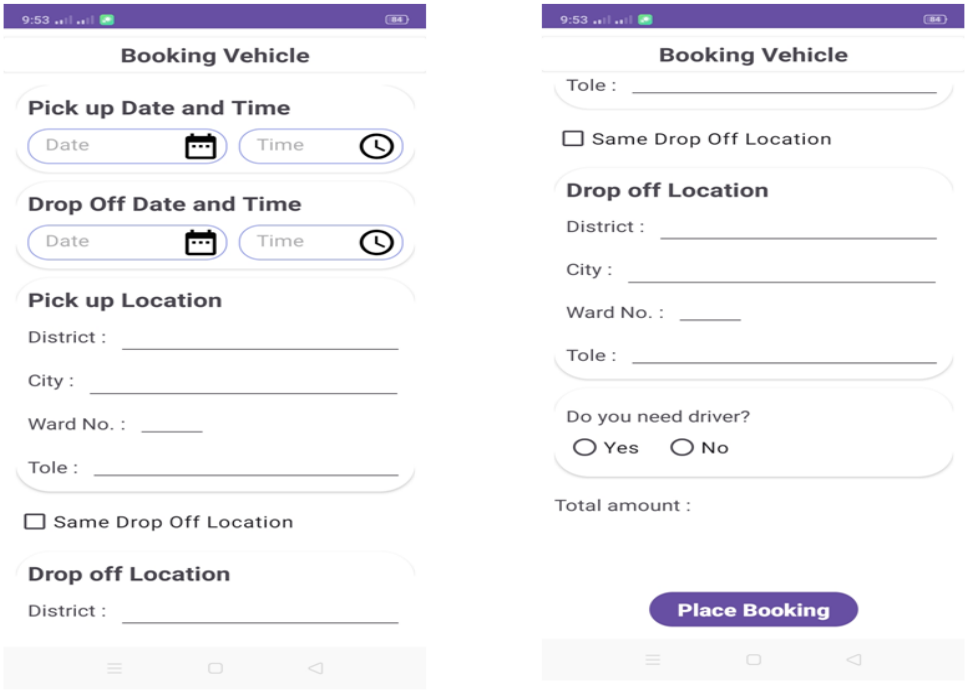


Figure: Annex 1. 11 Booking

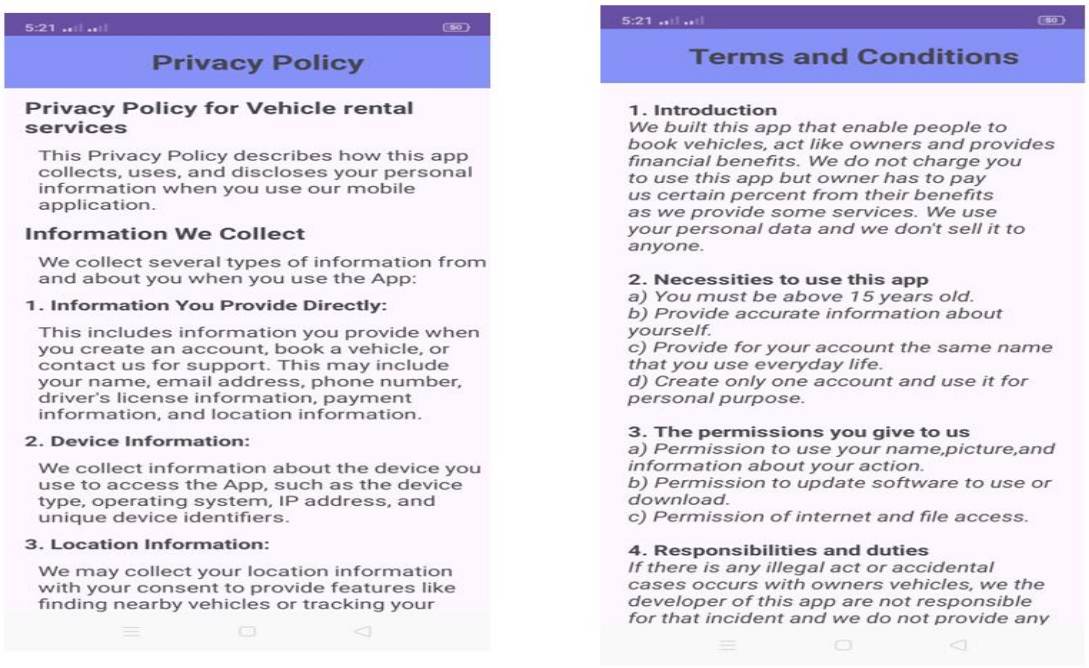


Figure: Annex 1. 12 Privacy Policy and Terms and Conditions

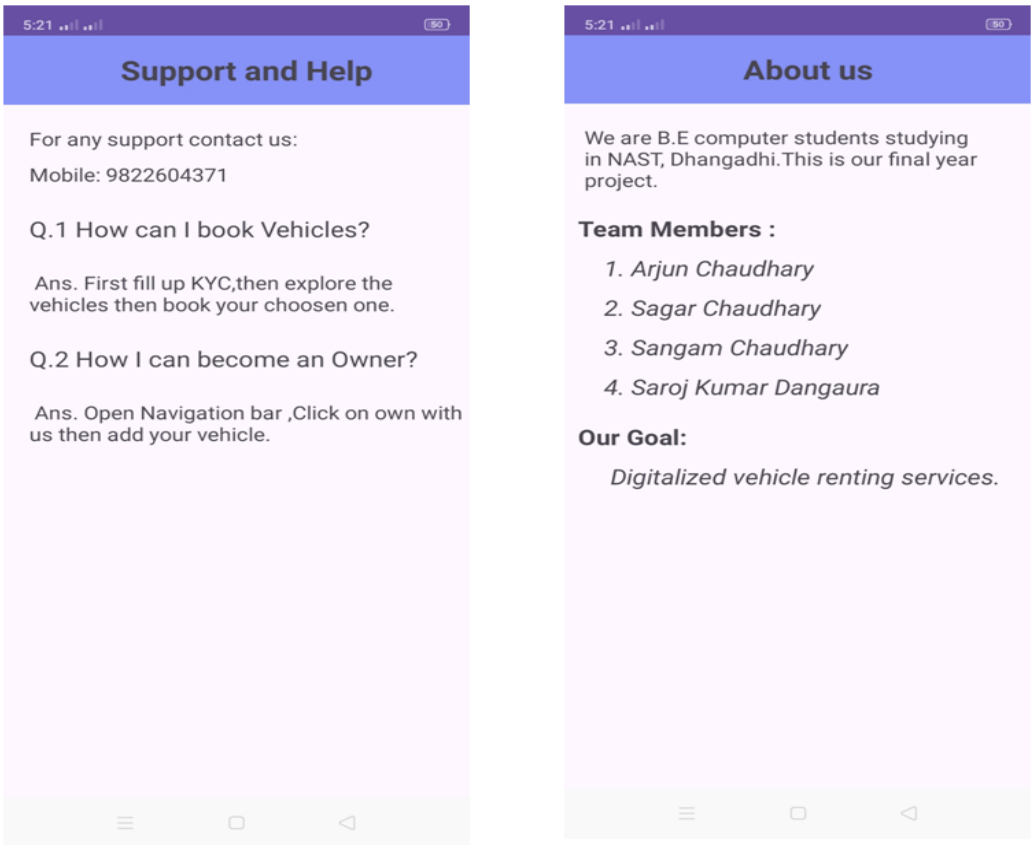


Figure: Annex 1. 13 Support and About us

ANNEX II

Source Code

Login Request

```
@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

public class JwtRequest {

        private String username;

        private String password;

}
```

Login Controller

```
@RestController

@AllArgsConstructor

@RequestMapping("api/vehicleRentalServices")

public class JwtController {

            @Autowired

        private UserDetailsService userDetailsService;

            @Autowired

         private AuthenticationManager manager;

            @Autowired

         private JwtTokenHelper helper;
```

```
@PostMapping("/login")
public ResponseEntity<JwtResponse> login(@RequestBody JwtRequest request,Authentication authenticattion ){
System.out.println(request);
    this.doAuthenticate(request.getUsername(), request.getPassword());
    UserDetails userDetails = userDetailsService.loadUserByUsername(request.getUsername());
    String token = this.helper.generateToken(userDetails);
    com.vehiclerentalservices.app.security.JwtResponse response = JwtResponse.builder()
            .jwtToken(token)
            .build();
    return new ResponseEntity<>(response, HttpStatus.OK);
}
private void doAuthenticate(String email, String password){
    UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(email, password);
    try {
       manager.authenticate(authentication);
    } catch (BadCredentialsException e) {
      throw new PasswordNotFoundException();}}
}
```

Login in Android Studio

```
Button loginbtn;

loginbtn = (Button) findViewById(R.id.lgnbtn);

loginbtn.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {


      String username = uname.getText().toString();
      String password = pass.getText().toString();
      if (username.isEmpty()) {
         if (password.isEmpty()) {
            Toast.makeText(LoginActivity.this, "Insert Username and Password !!",
Toast.LENGTH_SHORT).show();
         } else {
            Toast.makeText(LoginActivity.this, "Insert Username !!",
Toast.LENGTH_SHORT).show();
         }
      } else if (password.isEmpty()) {
         Toast.makeText(LoginActivity.this, "Insert password !!",
Toast.LENGTH_SHORT).show();
      } else {
         loginData(username, password);
         saveForRole();
      }}});

private void loginData(String username, String password) {
   String url = "http://192.168.1.74:8080/api/vehicleRentalServices/login";


   JSONObject login = new JSONObject();
   try {
      login.put("username", username);
      login.put("password", password);
```

```
    } catch (Exception e) {
        e.printStackTrace();}
    JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(
            Request.Method.POST, url, login,
            new Response.Listener<JSONObject>() {
                @Override
                public void onResponse(JSONObject response) {
                    try {
                        SharedPreferences pref = getSharedPreferences("LoginInfo",
MODE_PRIVATE);
                        SharedPreferences.Editor edit = pref.edit();
                        edit.putBoolean("Logs", true);
                        edit.apply();
                        String token = response.getString("jwtToken");
                        long expirationTime = System.currentTimeMillis() + (5 * 24 * 60 * 60
* 1000L); // 5 days in milliseconds
                        SharedPreferencesUtil.saveToken(LoginActivity.this, token,
expirationTime);
                        Toast.makeText(LoginActivity.this, "Login Successful !!",
Toast.LENGTH_LONG).show();
                        Intent intent = new Intent(LoginActivity.this, MainActivity.class);
                        startActivity(intent);
                        finish();
                    } catch (Exception e) {
                        e.printStackTrace();    }          }
            },
            new Response.ErrorListener() {
                @Override
                public void onErrorResponse(VolleyError error) {
                    if (error instanceof AuthFailureError) {
                        Toast.makeText(LoginActivity.this, "Login failed !!",
Toast.LENGTH_SHORT).show();
                    } else {
                        Toast.makeText(LoginActivity.this, "failed!!",
```

```
Toast.LENGTH_SHORT).show();
                }
            }
        }) {
    @Override
    public Map<String, String> getHeaders() throws AuthFailureError {
        Map<String, String> headers = new HashMap<>();
        headers.put("Content-Type", "application/json");
        return headers;
    }

};
requestQueue.add(jsonObjectRequest);
}
```

Sign Up API

Service Implementation

```
@Service
public class UserServiceImpl implements UserService {
        @Autowired
        private UserRepositary userRepo;
        @Autowired
        private PasswordEncoder passwordEncoder;
        @Autowired
        private ModelMapper mapper;
        @Override
        public UserDto createUser(UserDto custdto) {
```

```
        User user = this.dtoToUser(custdto);
        user.setPassword(passwordEncoder.encode(custdto.getPassword()));
        user.setName("default name");
        user.setAddress("default address");
        user.setMobile("1234567890");
        user.setDob("2080/10/10");
        user.setPhoto("photo.jpg");
        user.setLicensephoto("lisence.jpg");
        user.setStatus(0);
        user.setActivestatus(0);
        User saveduser = this.userRepo.save(user);
        return this.userToDto(saveduser);
    }
```

User Controller

```
@RestController
@RequestMapping("api/vehicleRentalServices")
public class UserController {
    @Autowired
    private UserService userService;
    @Autowired
    private FileService fileService;
    // add user
    @PostMapping("/user/signup")
```

```
        public ResponseEntity<?> createUser(@Valid @RequestBody UserDto
custdto) {

                String email = custdto.getEmail();

                List<UserDto> uEmail = this.userService.getAllByEmail(email);

                 if(!uEmail.isEmpty()) {

                        return new ResponseEntity<ApiResponse>(new
ApiResponse("Exist", true), HttpStatus.OK);

                        }

                UserDto dtos = this.userService.createUser(custdto);

                return new ResponseEntity<>(dtos, HttpStatus.CREATED);

        }
```

Sign Up in Android Studio

```
Button signUpBtn;

signUpBtn = findViewById(R.id.signUpbtn);

signUpBtn.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      String Username = uname.getText().toString();
      String password = pass.getText().toString();
      String confirmPassword = conformPass.getText().toString();
      if (acceptTerm.isChecked()) {
         checkBox = true;
      }
      if (Username.isEmpty() && password.isEmpty() &&
confirmPassword.isEmpty() && !checkBox) {
         Toast.makeText(SignUpActivity.this, "Insert all field !!",
Toast.LENGTH_SHORT).show();
      } else {
```

```
        if (Username.isEmpty()) {
            Toast.makeText(SignUpActivity.this, "Insert your email !!",
Toast.LENGTH_SHORT).show();
        } else {
            if (password.isEmpty()) {
                Toast.makeText(SignUpActivity.this, "Insert password !!",
Toast.LENGTH_SHORT).show();
            } else {
                if (confirmPassword.isEmpty()) {
                    Toast.makeText(SignUpActivity.this, "Insert confirm password !!",
Toast.LENGTH_SHORT).show();
                } else {
                    if (!checkBox) {
                        Toast.makeText(SignUpActivity.this, "Please accept our terms and
privacy policy !!", Toast.LENGTH_SHORT).show();
                    } else {
                        if (password.equals(confirmPassword)) {
                            processData(name, address, mobile, Username, password,
licensephoto, photo, dob, status, activestatus, role);
                        } else {
                            Toast.makeText(SignUpActivity.this, "Insert same password and
conform password !!", Toast.LENGTH_SHORT).show();
}}}}}}});

private void processData(String name, String address, String mobile, String username,
String password, String licensephoto, String photo, String dob, int status, int
activestatus, String role) {
   String url = "http://192.168.1.75:8080/api/vehicleRentalServices/user/signup";
   JSONObject sign = new JSONObject();
   try {
      sign.put("name", name);
      sign.put("address", address);
      sign.put("mobile", mobile);
      sign.put("email", username);
```

```
        sign.put("password", password);
        sign.put("licensephoto", licensephoto);
        sign.put("photo", photo);
        sign.put("status", status);
        sign.put("activestatus", activestatus);
        sign.put("role", role);
    } catch (Exception e) {
        e.printStackTrace();
    }
    JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(
            Request.Method.POST, url, sign,
            new Response.Listener<JSONObject>() {
                @Override
                public void onResponse(JSONObject response) {
                    try {
                        Toast.makeText(SignUpActivity.this, "Sign Up Successful !!",
Toast.LENGTH_SHORT).show();

                        Intent intent = new Intent(SignUpActivity.this, LoginActivity.class);
                        startActivity(intent);
                        finish();
                    } catch (Exception e) {
                        e.printStackTrace();}}
            },
            new Response.ErrorListener() {
                @Override
                public void onErrorResponse(VolleyError error) {
                    if (error instanceof AuthFailureError) {
                        Toast.makeText(SignUpActivity.this, "failed !!",
Toast.LENGTH_SHORT).show();
                    } else {
                        Toast.makeText(SignUpActivity.this, "failed!!",
Toast.LENGTH_SHORT).show();
                }}
```

```
        }) {
      @Override
      public Map<String, String> getHeaders() throws AuthFailureError {
         Map<String, String> headers = new HashMap<>();
         headers.put("Content-Type", "application/json");
         return headers;
      }};
   requestQueue.add(jsonObjectRequest);}
```

Booking Vehicle

```
@Override
        public BookingDto createBooking(BookingDto bookDto, Integer uId, Integer
vId) {
                User user = this.userRepo.findById(uId)
                                .orElseThrow(()-> new
ResourceNotFoundException("User", "id", uId));
                Vehicle vehicle = this.vehicleRepo.findById(vId)
                                .orElseThrow(()-> new
ResourceNotFoundException("Vehicle", "id", vId));
                Booking book = this.dtoToBooking(bookDto);
                book.setUser(user);
                book.setVehicle(vehicle);
                Booking books = this.bookingRepo.save(book);
                return this.bookingToDto(books);
        }

        @Override
        public BookingDto updateBooking(BookingDto bookDto, Integer id) {
                Booking book = this.bookingRepo.findById(id)
                                .orElseThrow(()-> new
ResourceNotFoundException("Booking", "id", id));

                book.setFlag(bookDto.getFlag());
```

```
                Booking books = this.bookingRepo.save(book);
                BookingDto updateBook = this.bookingToDto(books);
                return updateBook;
}
```

Controller

```
@PostMapping("/user/{uid}/vehicle/{vid}/booking")
        public ResponseEntity<BookingDto> createBookVechile(@Valid
@RequestBody BookingDto book, @PathVariable int uid, @PathVariable int vid) {
                BookingDto books = this.bookingService.createBooking(book, uid,
vid);
                return new ResponseEntity<>(books, HttpStatus.CREATED);}
@PutMapping("/booking/{id}")
        public ResponseEntity<BookingDto> updateBookVehicle(@Valid
@RequestBody BookingDto book, @PathVariable int id) {

                BookingDto books = this.bookingService.updateBooking(book, id);
                return ResponseEntity.ok(books);
        }
```

Add Vehicle

```
@Override
        public VehicleDto createVehicle(VehicleDto vehicleDto) {
                Vehicle vehicle = this.dtoToVehicle(vehicleDto);
                Vehicle saveVehicle = this.vehicleRepo.save(vehicle);
                return this.vehicleToDto(saveVehicle);}

        @Override
        public VehicleDto updateVehicle(VehicleDto vehicleDto, Integer id) {
                Vehicle vehicle = this.vehicleRepo.findById(id)
                                .orElseThrow(()-> new
ResourceNotFoundException("Vehicle", "id", id));
                vehicle.setName(vehicleDto.getName());
                vehicle.setType(vehicleDto.getType());
```

```
            vehicle.setAmount(vehicleDto.getAmount());
            vehicle.setStatus(vehicleDto.getStatus());
            vehicle.setBluebookphoto(vehicleDto.getBluebookphoto());
            vehicle.setInsurancephoto(vehicleDto.getInsurancephoto());
            vehicle.setDriverstatus(vehicleDto.getDriverstatus());
            vehicle.setLocation(vehicleDto.getLocation());

            Vehicle vehicles = this.vehicleRepo.save(vehicle);
            VehicleDto vehicles1 = this.vehicleToDto(vehicles);
            return vehicles1;
    }
```

Controller

```
@PostMapping("/create/vehicle")
    public ResponseEntity<VehicleDto> addVehicle(@Valid @RequestBody
VehicleDto vehicleDto){
            VehicleDto vehicle = this.vehicleService.createVehicle(vehicleDto);
            return new ResponseEntity<>(vehicle, HttpStatus.CREATED);
    }
    @PutMapping("vehicle/{id}")
    public ResponseEntity<VehicleDto> updateVehicle(@Valid @RequestBody
VehicleDto vehicleDto, @PathVariable int id){
            VehicleDto dtos = this.vehicleService.updateVehicle(vehicleDto, id);
            return ResponseEntity.ok(dtos);
    }
```