Bhai Chara

Project ID: F19-46
Session: BSCS Fall 2016-20

Supervisor: Dr. Yasir Niaz Khan

Submitted By

Adeem Akram	70068809
Ali Raza	70068788
Ghulam Muhammad	70069090
Sani Buttar	

Department of Computer Science & IT
The University of Lahore
Lahore, Pakistan

Declaration

We have read the project guidelines and we understand the meaning of academic dishonesty, in particular plagiarism and collusion. We hereby declare that the work we submitted for our final year project, entitled **Bhai Chara** is original work and has not been printed; published or submitted before as final year project, research work, publication or any other documentation.

Group Member 1 Name: Adeem Akram
SAP No: 70068809

Signature: **……**

Group Member 2 Name: Ali Raza
SAP No: 70068788

Signature: **……**

Ghulam Muhammad Sani Buttar
SAP No: 70069090

Signature: **……**

Statement of Submission

This is to certify that **Adeem Akram 70068809, Ali Raza 70068788 and Ghulam Muhammad Sani Buttar 70069090** have successfully submitted the final year project titled: **Bhai Chara**, in fulfillment of the requirements for the degree of **Bachelors of Science in Computer Science**, **The University of Lahore**

Project Supervisor: Dr. Yasir Niaz Khan

Signature: ……
Date: ……

Dedication

This Project is dedicated to all the working class of our country who face many hardships, commuting to work being one of them. We hope to solve this problem by our project so they will have one less thing to worry about and concentrate on their work resulting in the betterment of themselves, society and country.
We are also grateful to our teacher's only beings in the universe, whose sole ideology is to make their students better than themselves and much more

Acknowledgement

We acknowledge the cooperation and help made by **Dr. Yasir Niaz Khan**, **Mr. Ihtisham Ul Haq** and **Mr. Fahad Ishaque**. They had been a constant source of guidance and motivation throughout the course of this project. Whenever we encountered an obstacle either general or technical, they were the beacon of light who led us and pushed us through. We will be eternally grateful to them.

Dated:

Feb 8th, 2020

Abstract

There are over a million vehicles in Lahore and there had been an increase of 3000 every month. According to Gallop Pakistan, two seats are empty out of every four at average. Considering the environmental concerns, congestion of roads, and rising fuel costs and inflation ride sharing has gained a lot of popularity when it comes to environment-friendly and cheap ways of travelling. Ride sharing is when two or more persons share a ride in one of each other's personal car. Ride Sharing reduces pollution since we have less cars on the road. It is also economic since the travel expenses are shared among the riders. Finding people to share a ride with, is the challenge of ride sharing as it is difficult to find a person going to the same place as you at a given time. The proposed application enables users to create and share their trip and find passengers. The purpose of this project is to develop an application that tries to overcome the disadvantages of the other available options e.g. Uber Car sharing. "Bhai Chara" is a real-time application: any person taking the trip can check the meeting and ending point, communicate with other commuters. The main objective of the project proposed is to develop an android application, with the prime goal of sharing a ride in real time.

The beauty of ride-sharing lies in the fact that it taps into an abundant but underutilized resource: the empty seats in cars. These empty seats represent a huge amount of waste in our transportation system but potentially a huge opportunity for improvement.

Area of the Project

- Mobile Programming
- RESTFull APIs
- Web Server

Technologies used

- Android Studio
- Spring MVC
- Sockets
- Chat Framework
- Servlets

List of Figures

Figure 1 Use case User Login 27
Figure 2 Use case Sign Up 28
Figure 3 Use case Search For A ride 29
Figure 4 Use case Profile Settings 30
Figure 5 Use case Transaction History 31
Figure 6 Use case Rides History 32
Figure 7 Use case Switch Mode 33
Figure 8 Use case Start A Ride 34
Figure 9 Use case Ends A Ride 35
Figure 10 Use case Collects the fare 36
Figure 11 Use case Admin Sign In 37
Figure 12 Use case Adds an organization 38
Figure 13 Use case Blocks a user 39
Figure 14 Use case Unblocks the user 40
Figure 15 Use case View the users 41
Figure 16 Use case Change roles 42
Figure 17 Architecture Diagram 44
Figure 18 ER Diagram 45
Figure 19 Data Flow Diagram level 0 46
Figure 20 Data Flow Diagram level 1 Admin 47
Figure 21 Data Flow Diagram level 1 Driver 48
Figure 22 Data Flow Diagram level 1 Rider 49
Figure 23 Class Diagram 50
Figure 24 Sign Up Activity diagram 51
Figure 25 login Activity diagram 52
Figure 26 Search for a Ride Activity diagram 53
Figure 27 Add a Ride Activity diagram 54
Figure 28 Start a Ride Activity diagram 55
Figure 29 End a Ride Activity diagram 56
Figure 30 Collect Fare Activity diagram 57
Figure 31 View Ride History Activity diagram 58
Figure 32 View Transaction History Activity diagram 58
Figure 33 Switch Mode Activity diagram 59
Figure 34 Switch Role Activity diagram 59
Figure 35 Add Organization Activity diagram 60
Figure 36 Block User Activity diagram 61
Figure 37 Unblock User Activity diagram 62
Figure 38 Remove User Activity diagram 63

Figure 39 Admin Block User Sequence Diagram 64
Figure 40 Admin Login Sequence Diagram 65
Figure 41 Add Organization Sequence Diagram 66
Figure 42 Admin Unblock the User Sequence Diagram 67
Figure 43 Admin Remove User Sequence Diagram 68
Figure 44 Driver Start The Ride Sequence Diagram 69
Figure 45 User Sign Up Sequence Diagram 70
Figure 46 User Search Ride Sequence Diagram 71
Figure 47 User Login Sequence Diagram 72
Figure 48 User Transaction History Sequence Diagram 73
Figure 49 User Ride History Sequence Diagram 74
Figure 50 User Profile Setting Sequence Diagram 75
Figure 51 Driver End Ride Sequence Diagram 76
Figure 52 Collaboration Diagram Rider 77
Figure 53 Collaboration Diagram driver 78
Figure 54 Collaboration Diagram Admin 79
Figure 55 State Transition Diagram User 80
Figure 56 State Transition Diagram Admin 81
Figure 57 Component Diagram 82
Figure 58 Deployment Diagram 83

List of Tables

Table 1 Definitions, acronyms and abbreviations 4,5
Table 2 Functional Requirement login 9
Table 3 Functional Requirement Sign Up 10
Table 4 Functional Requirement Search for a driver 10
Table 5 Functional Requirement Add a ride 11
Table 6 Functional Requirement Profile settings 11
Table 7 Functional Requirement View ride history 12
Table 8 Functional Requirement Transaction history 12
Table 9 Functional Requirement Driver Login 13
Table 10 Functional Requirement Driver Sign Up 13
Table 11 Functional Requirement Search for riders 14
Table 12 Functional Requirement Add a ride 14
Table 13 Functional Requirement Profile settings 15
Table 14 Functional Requirement View ride history 15
Table 15 Functional Requirement Transaction history 16
Table 16 Functional Requirement Set Mode 16
Table 17 Functional Requirement Accepting a ride 17
Table 18 Functional Requirement Starting the ride 17
Table 19 Functional Requirement Ending the ride 18
Table 20 Functional Requirement Collecting the fare 18
Table 21 Functional Requirement Sign-in 19
Table 22 Functional Requirement Adding an organization 19
Table 23 Functional Requirement View users 20
Table 24 Functional Requirement Removing a user 20
Table 25 Functional Requirement Blocking a user 21
Table 26 Functional Requirement Unblocking the user 21
Table 27 Use case Sign In 27
Table 28 Use case Sign Up 28
Table 29 Use case Search for A ride 29
Table 30 Use case Profile Settings 30
Table 31 Use case Transaction History 31
Table 32 Use case Rides History 32
Table 33 Use case Switch Mode 33
Table 34 Use case Start a Ride 34
Table 35 Use case Ends A Ride 35
Table 36 Use case Collects the fare 36
Table 37 Use case Sign In 37

Table 38 Use case Adds an organization 38
Table 39 Use case Blocks a user 39
Table 40 Use case Unblocks the user 40
Table 41 Use case View the users 41
Table 42 Positive Test case sign in 85
Table 43 Negative Test case sign in 86
Table 44 Positive Test case sign up 87
Table 45 Negative Test case sign up 88
Table 46 Positive Test case search a ride 89
Table 47 Negative Test case search a ride 90
Table 48 Positive Test case profile settings 91
Table 49 Negative Test case profile settings 92
Table 50 Positive Test case transaction history 93
Table 51 Negative Test case transaction history 94
Table 52 Positive Test case rides history 95
Table 53 Negative Test case rides history 95
Table 54 Positive Test case switch mode 96
Table 55 Negative Test case switch mode 96
Table 56 Positive Test start ride 97
Table 57 Negative Test case start ride 98
Table 58 Positive Test case end ride 99
Table 59 Negative Test case end ride 100
Table 60 Positive Test case collect fare 101
Table 61 Negative Test case collect fare 102
Table 62 Positive Test add an organization 103
Table 63 Negative Test case add an organization 104
Table 64 Positive Test case block a user 104
Table 65 Negative Test case block a user 105
Table 66 Positive Test case unblock the user 105
Table 67 Negative Test case unblock the user 106
Table 68 Positive Test case view users 106
Table 69 Negative Test case view users 107
Table 70 Positive Test case view users 107
Table 71 Negative Test case view users 108
Table 72 Positive Test case change roles 109
Table 73 Negative Test case change roles 110
Table 74 Equivalence Partitions 111
Table 75 BVA for user 111
Table 76 BVA for password 112
Table 77 Decision Table for Login 112
Table 78 EP for Login Authentication 112
Table 79 BVA for Username 113
Table 80 BVA for Password 113

Table 81 Decision Table for Login 113
Table 82 EP for Logout Authentication 114
Table 83 Decision Table for Log out 114
Table 84 EP for Add person information in the database 114
Table 85 BVA for Name 115
Table 86 BVA for Place ID 115
Table 87 BVA for Image 115
Table 88 Decision Table for adding new Place information 116
Table 89 EP for Add person information in the 116
Table 90 BVA for Name 116
Table 91 BVA for Place ID 117
Table 92 Decision Table for adding new Place information 117
Table 93 EP for Delete information in the database 117
Table 94 BVA for Deleting the Information 118
Table 95 Decision Table for Deleting the Information 118
Table 96 State Transition table for Login 120

Table of Content

Declaration i
Statement of Submission ii
Dedication iiii
Acknowledgement iv
Abstract v
List of Figures vii
List of Tables ix
Chapter 1: Introduction to the Problem xiv
1.1 Introduction 1
1.2 Purpose 1
1.3 Objective 2
1.4 Existing Solution 2
1.5 Proposed Solution 2
Chapter 2: Software Requirement Specification 3
2.1 Introduction 4
2.1.1 Purpose 4
2.1.2 Scope 4
2.1.3 Definitions, acronyms, and abbreviations 4
2.2 Overall description 5
2.2.1 Product perspective 5
2.2.2 Product functions 9
2.2.3 User characteristics 21
2.2.4 Constraints 21
2.2.5 Assumptions and dependencies 22
2.2.6 Apportioning of requirements 22
2.3 Specific requirements 23
2.3.1 Functional Requirement 23
2.3.2 Non-functional Requirements 24
Chapter 3: Use Case Analysis 26
Chapter 4: Design 43
4.1 Architecture Diagram 44

4.2 ERD with data dictionary 45
4.3 Data Flow diagram 46
4.3.1 The level 0 46
4.3.2 The level 1 47
4.4 Class Diagram 50
4.5 Activity Diagram 51
4.6 Sequence Diagram 64
4.7 Collaboration Diagram 77
4.8 State Transition Diagram 80
4.9 Component Diagram 82
4.10 Deployment Diagram 83
Chapter 5: Testing 84
5.1 Test Case Specifications 85
5.2 Black Box Test Cases 111
5.2.1 Equivalence Partitions (EP) 111
5.2.2 Boundary Value Analysis 115
5.2.3 Decision Table Testing 116
5.2.4 State transition Testing 119
5.3 White Box Test Cases 120
5.3.1 Cyclometric complexity 121
5.4 Performance testing 125
5.5 Stress Testing 125
5.6 System Testing 125
5.7 Regression Testing 126
Chapter 6: Tools and Techniques 127
Chapter 7: Summary and Conclusion 132
Chapter 8: User Manual 134
Chapter 9: Lessons Learnt and Future Work 147
Chapter 10: Learning Outcomes 150
References 182
Appendix 183

Chapter 1
INTRODUCTION

1.1 Introduction

A million vehicles are registered in Lahore with a monthly increase of three thousand, According to a survey conducted by Gallop Pakistan; two seats are vacant out of every four at average. Considering the environmental concerns, congestion of roads, reduced parking space and rising fuel costs and inflation, some of this could be blamed on our infrastructure but the major cause is the increased number of vehicles. The idea of Ride Sharing comes handy when we try to address these concerns.

Ride sharing has gained a lot of popularity when it comes to environment-friendly and cheap ways of travelling. Ride sharing is when two or more people share a ride in one of each other's vehicle. Ride Sharing reduces pollution since we have less cars on the road. It is also economic since the travel expenses are shared among the riders. This model is very useful especially when people are travelling out of station or a ride is shared among the employees of the same company heading for work.

The proposed application enables users to create and share their trip and find passengers. "Bhai Chara" is a real-time application: any person wishing to take the trip can check the source and the destination and can communicate with other commuters, both drivers and the riders i.e. drivers searching for passengers and vice versa. The main objective of the project proposed is to develop an android application, with the prime goal of sharing a ride in real time.

Every idea comes with its own pros and cons. Finding people in real time to share a ride with, is one of the challenges of ride sharing. As it is not a dedicated taxi, service there might be an instance when no commuters are available (both riders and the drivers). Another concern is the security of our users.

1.2 Purpose

We are living in a digital world in which internet has become a basic commodity instead of luxury. People spend much of their time online for either work or recreation so an online application providing a relatively cheaper way of transportation will appease many. The purpose of this project is to develop an android application that tries to overcome the disadvantages of the other available applications e.g. Uber Car sharing, local transportation system. I.e. cheaper than Uber car sharing faster than local transportation. This will encourage a sharing attitude among people that could be better for society.

1.3 Objective

1. To optimize the traffic flow, reducing pollution.
2. To reduce congestion in crowded areas by promoting ride sharing.
3. To decrease the use of private cars, addressing the problem of less parking spaces.
4. To provide timely efficient, comfortable, cost effective way of commuting.
5. Implementing "quid pro que" principle. Drivers getting a share in fuel cost, while riders getting a low cost mode of transportation

1.4 Existing Solution

1. **Careem, Uber,** both are app-based vehicle booking Service Companies based in Dubai and USA respectively both connects drivers and passengers and have many vehicle categories.
2. **Swyl, Airlift,** both are app-based bus booking Service Companies, let users select a pick up and drop off, on fixed routes and times.

The available ride booking applications require drivers to dedicate certain time, follow the pick-up, and drop off the passengers. They are quite expensive too, buses have fewer route options and they are more prone to traffic jams.
Bhai Chara app is a low cost substitute that allows the driver to pick up passengers of same destination at same time to share a ride e.g. employees or students of university of Lahore coming to university

1.5 Proposed Solution

People always crave for what they think is best for them, when it comes to transportation same principle applies they may want a service that is fast, comfortable, secure and cost efficient. Bhai Chara is timely efficient as the drivers and the passengers both have same source and destination, it will also have a group code (a unique code for a particular organization or destination) using it only drivers and the passengers having same group code will be connected. It will be cost efficient for both drivers and the passenger's driver getting a share in the cost of fuel and passenger getting a cheaper way of transportation. For addressing the security concerns our application has two modes public and private, private mode allows the driver to share a ride only with the people he wants and in public mode, a ride could be shared with any available passenger.

Chapter 2:
Software Requirement Specification

2.1 Introduction

A software requirements specification (SRS) is a comprehensive description of the intended purpose and environment for software under development. The SRS fully describes what the software will do and how it will be expected to perform.

The following subsection of the software requirements specification document provides an overview of the entire SRS, which is being followed by the "Bhai Chara".

2.1.1 Purpose

An SRS minimizes the time and effort required by developers to achieve desired goals and minimizes the development cost. A good SRS defines how an application will interact with system hardware, other programs and human users in a wide variety of real-world situations.

SRS will provide a detailed description of requirements of "Bhai Chara". It will allow for a complete understanding of what is to be expected from the App to be developed. This will provide the foundation for the project more specifically what will be the design, what will be developed and how it will be tested.

2.1.2 Scope

The "Bhai Chara" is an android app, which will provide real time communication environment for users (Drivers and passengers). Every user will have to create their profile by using their mobile number, after profile is created user can access his/her profile by using the registered number. The drivers can define their source and destination. In addition, passengers can communicate with the driver via chat framework and pick their path. After mutual agreement with each other, the ride could be started from source to destination; once a ride is finished, passenger will pay a specific amount to the driver based on distance travelled and duration. Moreover, drivers can add a ride in advance whereas users can also search for a ride in future. The system will bring many advantages. For instance, the drivers and passengers spend less money on transportation. Moreover, there will be a decrease in air pollution and congestion of roads. The system will be developed using android studio for user App, RESTful APIs written in java using spring boot and servlets.

2.1.3 Definitions, acronyms, and abbreviations

Term	Definition
App	Android application
Servlets	Servlet technology is used to create a web application (resides at server side and generates a dynamic web page).
RESTFul API	A RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data.

API	An application programming interface is an interface or communication protocol between different parts of a computer program intended to simplify the implementation and maintenance of software
SRS	Software Requirement Specification
Chat Framework	A messaging platform is a unique tool that enables Internet users to exchange messages for the purpose of human communications
Spring boot	Spring Boot is an open source Java-based framework used to create a micro Service. It is developed by Pivotal Team and is used to build stand-alone and production ready spring applications.

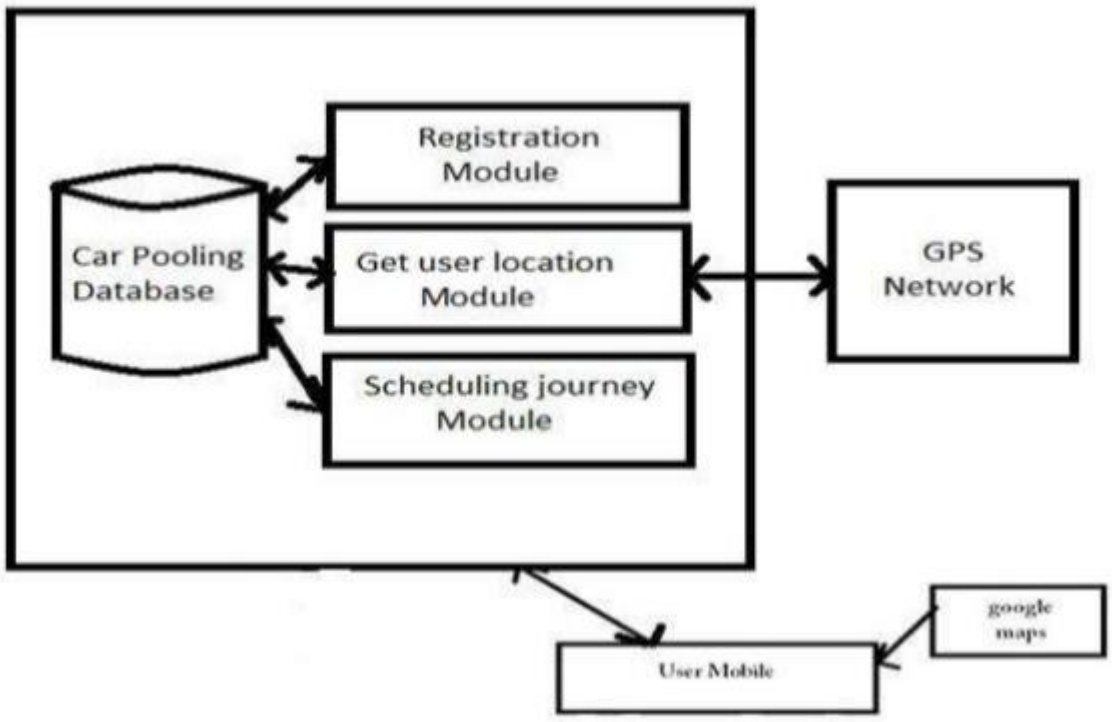
Table 1 Definitions, acronyms and abbreviations

2.2 Overall description

This section gives background information about specific requirements of the web based carpool environment to be developed in brief. Although we will not describe every requirement in detail, this section will describe the factors that affect the final product.

2.2.1 Product perspective

This project is independent and self-contained. The constraints, which describe how the software operates, are listed below:



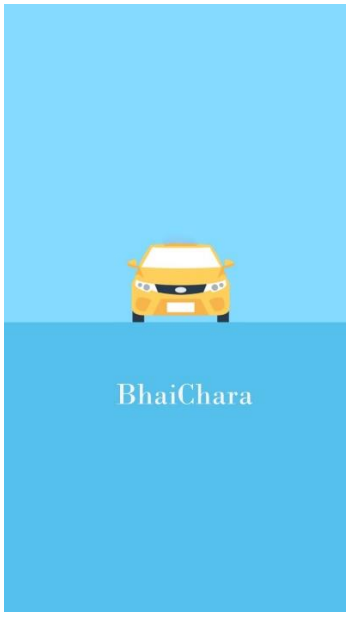
System interfaces

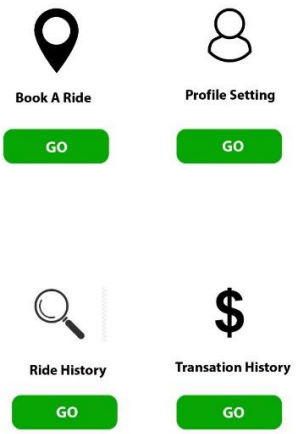
The system has Google Map API as a subsystem. Google Map subsystem has their own web-based interface, which is a map consists of roads and locations in a desired area and user can easily interact with this system.

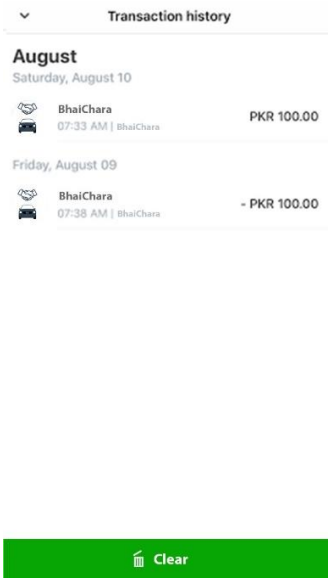
User interfaces

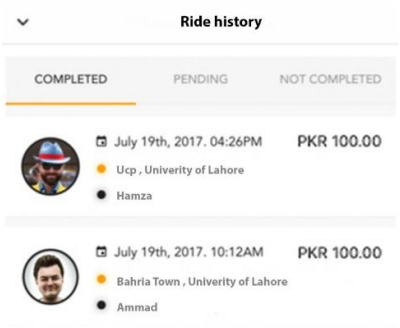
UI design usually refers to the design, look and feel of the application. It can also refer to other ways of interaction like natural and voice user interfaces.

The user interface for this application will be very simple and creative and will be designed keeping in mind the design principles of human computer interaction. The UI will be created using XML in android studio. The proposed UI will be responsive adapting to different screen sizes of mobiles









Hardware interfaces

There are no specific hardware requirements for the application; one only needs an android phone with a GPS or Location sensor.

Software interfaces

The application requires a minimum Software Development Kit (SDK) 17, Jelly Bean. Although newer versions may have a better support

Communications interfaces

A server is required for the application to be online. The server comes with an interface of its own we do not specially need to design it e.g. tomcat 9.0 and Amazon Web Services (AWS). The application will communicate via internet using sockets

Memory

Bhai Chara will approximately occupy general amount of disk space i.e. 30 to 35 MB and it will take up around 100 to 300 MB memory.

Operations

The system will allow users to search for both riders and drivers in the desired radius. It will connect both riders and drivers so those having same route may share a ride, once there is a mutual agreement driver picks up the rider starts the ride and after reaching the destination collects the ride fare. Users can add rides in advance and can also check their ride and transaction histories.

Site adaptation requirements

The device in which the application is installed should have an active internet connection and working GPS

2.2.2 Product functions

Al the functional requirements are divided into three modules namely:

- Rider
- Driver
- Admin

Rider

ID:	RC_01			
Name:	Login			
Description	Input	Output	Requirements	Basic Work Flow
Process by which user can access our application by authenticating themselves	Phone number	User logged in successfully	Device should have internet connection User should be registered	Open the verification page enter the phone number. If a user exists against this phone number the system will direct you to home screen else it'll direct user to the sign up screen

Table 2 Functional Requirement login

ID:	RC_02			
Name:	Sign Up			
Description	Input	Output	Requirements	Basic Work Flow
The process by which the user registers him/herself into the system	First name Last name Email-id Group code Click the sign up button	User Successfully Registered	Active internet connection User should not be registered before	Enter All the correct information and click the sign up button

Table 3 Functional Requirement Sign Up

ID:	RC_03			
Name:	Search for a driver			
Description	Input	Output	Requirements	Basic Work Flow
Riders can search for drivers	Source Destination Pick-up point Car type Number of seats required	Available drivers will be displayed	Active internet connection Rider should be logged in	Rider specifies his/her source and destination available drivers will be displayed accordingly

Table 4 Functional Requirement Search for a driver

ID:	RC_04			
Name:	Add a ride			
Description	Input	Output	Requirements	Basic Work Flow
Rider adds his ride in advance	Source Destination Pick-up point Car type Number of seats required	Available drivers will be displayed	Active internet connection Rider should be logged in	Rider specifies his/her source and destination available drivers will be displayed accordingly

Table 5 Functional Requirement Add a ride

ID:	RC_05			
Name:	Profile settings			
Description	Input	Output	Requirements	Basic Work Flow
This process will allow the user to change or update his/her details in user profile	First name Last name Email-id Group code Anything from the above mentioned that needs to be updated Click the update button	Updated Profile	Active internet connection Rider should be logged in	After logging in user opens the profile settings screen, updates the required details

Table 6 Functional Requirement Profile settings

ID:	RC_06			
Name:	View ride history			
Description	Input	Output	Requirements	Basic Work Flow
The process by which the user can view the history of rides taken	Click on the ride history button	Ride history displayed	Active internet connection Rider should be logged in	After logging in user clicks on the ride history button and views the ride history

Table 7 Functional Requirement View ride history

ID:	RC_07			
Name:	Transaction history			
Description	Input	Output	Requirements	Basic Work Flow
The process by which the user can view the transaction history of the rides taken	Click on the transaction history button	Transaction history displayed	Active internet connection Rider should be logged in	After logging in user clicks on the transaction history button and views the transaction history

Table 8 Functional Requirement Transaction history

Driver

ID:	DR_01			
Name:	Login			
Description	Input	Output	Requirements	Basic Work Flow
Process by which user can access our application by authenticating themselves	Phone number	User logged in successfully	Device should have internet connection User should be registered	Open the verification page enter the phone number. If a user exists against this phone number the system will direct you to home screen else it'll direct user to the sign up screen

Table 9 Functional Requirement Login

ID:	DR_02			
Name:	Sign Up			
Description	Input	Output	Requirements	Basic Work Flow
The process by which the user registers him/herself into the system	First name Last name Email-id Group code Driver's license number Click the sign up button	User Successfully Registered	Active internet connection User should not be registered before	Enter All the correct information and click the sign up button

Table 10 Functional Requirement Sign Up

ID:	DR_03			
Name:	Search for riders			
Description	Input	Output	Requirements	Basic Work Flow
Driver can search for riders	Source Destination Pick-up point Car type Number of seats available	Available riders will be displayed	Active internet connection Driver should be logged in	Driver specifies his/her source and destination available riders will be displayed accordingly

Table 11 Functional Requirement Search for riders

ID:	DR_04			
Name:	Add a ride			
Description	Input	Output	Requirements	Basic Work Flow
Driver adds his ride in advance	Source Destination route Car type Car number Number of seats available	Available riders will be displayed	Active internet connection Driver should be logged in	Driver specifies his/her source and destination available riders will be displayed accordingly

Table 12 Functional Requirement Add a ride

ID:	DR_05			
Name:	Profile settings			
Description	Input	Output	Requirements	Basic Work Flow
This process will allow the user to change or update his/her details in user profile	First name Last name Email-id Group code License number Anything from the above mentioned that needs to be updated Click the update button	Updated Profile	Active internet connection Driver should be logged in	After logging in user opens the profile settings screen, updates the required details

Table 13 Functional Requirement Profile settings

ID:	DR_06			
Name:	View ride history			
Description	Input	Output	Requirements	Basic Work Flow
The process by which the user can view the history of rides taken	Click on the ride history button	Ride history displayed	Active internet connection Rider should be logged in	After logging in user clicks on the ride history button and views the ride history

Table 14 Functional Requirement View ride history

ID:	DR_07			
Name:	Transaction history			
Description	Input	Output	Requirements	Basic Work Flow
The process by which the user can view the transaction history of the rides taken	Click on the transaction history button	Transaction history displayed	Active internet connection Rider should be logged in	After logging in user clicks on the transaction history button and views the transaction history

Table 15 Functional Requirement Transaction history

ID:	DR_08			
Name:	Set Mode			
Description	Input	Output	Requirements	Basic Work Flow
The process by which the driver can switch his mode public or private	Switch between public or private	Selected mode enabled	Active internet connection Driver should be logged in Driver should be active	After logging in driver clicks on the rides button and can set his mode public or private

Table 16 Functional Requirement Set Mode

ID:	DR_09			
Name:	Accepting a ride			
Description	Input	Output	Requirements	Basic Work Flow
The process by which the driver Can accept a ride	Clicking on the Accept button	Ride accepted	Active internet connection Driver should be logged in Driver should be active	After logging in driver clicks on the Accept button and accepts the ride

Table 17 Functional Requirement Accepting a ride

ID:	DR_10			
Name:	Starting the ride			
Description	Input	Output	Requirements	Basic Work Flow
The process by which the driver can start the ride by arriving at the agreed pick-up point	Clicking on the start ride button	Ride started	Active internet connection Driver should be logged in Driver should be active Driver should accept a ride	After logging in driver clicks on the Accept button and starts the ride after reaching at the pick-up point

Table 18 Functional Requirement Starting the ride

ID:	DR_11			
Name:	Ending the ride			
Description	Input	Output	Requirements	Basic Work Flow
The process by which the driver can end a ride after completing it	Clicking on the end ride button	Ride ended	Active internet connection Driver should be logged in Driver should be active Driver should start a ride	After logging in driver clicks on the Accept button and starts the ride after reaching at the pick-up point, finishes the ride reaching at the agreed drop-off.

Table 19 Functional Requirement Ending the ride

ID:	DR_12			
Name:	Collecting the fare			
Description	Input	Output	Requirements	Basic Work Flow
The process by which the driver can collect the fare after ending a ride.	Clicking on the collect fare button	Fare calculated	Active internet connection Driver should be logged in Driver should be active Driver should accept a ride Driver should end a ride	After logging in driver clicks on the Accept button and starts the ride after reaching at the pick-up point, finishes the ride reaching at the agreed drop-off and collects fare.

Table 20 Functional Requirement Collecting the fare

Admin

ID:	AD_01			
Name:	Sign-in			
Description	Input	Output	Requirements	Basic Work Flow
The process by which the admin can sign-in to our backend databases and server side	Id Password	Admin signed in successfully	Admin must be registered before	Open the verification page enter id and password. If credentials are validated the user logs in

Table 21 Functional Requirement Sign-in

ID:	AD_02			
Name:	Adding an organization			
Description	Input	Output	Requirements	Basic Work Flow
The process by which the admin can add an organization to the database and assigns a unique group code	Adding the organization and its group code into the database	Organization added	Admin must be logged in	After logging in the admin adds the organization and group code into the organizations table in database

Table 22 Functional Requirement Adding an organization

ID:	AD_03			
Name:	View users			
Description	Input	Output	Requirements	Basic Work Flow
The process by which the admin can view the registered users	Admin opens the admin panel	Users viewed	Admin should be logged in	After logging in admin views the users

Table 23 Functional Requirement View users

ID:	AD_04			
Name:	Removing a user			
Description	Input	Output	Requirements	Basic Work Flow
The process by which the driver can remove a user or an organization from the system	Selecting a user and removing the selected user	User/organization Removed	Admin should be logged in	Admin selects the specific user/organization and removes from the database

Table 24 Functional Requirement Removing a user

ID:	AD_05			
Name:	Blocking a user			
Description	Input	Output	Requirements	Basic Work Flow
The process by which the admin can block a user	Identify the user Block the user	User blocked	Admin should be logged in	Admin selects the specific user and blocks him/her

Table 25 Functional Requirement Blocking a user

ID:	AD_06			
Name:	Unblocking the user			
Description	Input	Output	Requirements	Basic Work Flow
The process by which the admin can unblock a user	Identify the user unblock the user	User unblocked	Admin must be logged in and have access to the block list	Selects the user from the block list and unblock him/her

Table 26 Functional Requirement Unblocking the user

2.2.3 User characteristics

- No experience required
- User should be literate if not he may memorize the intended functions of specific controls
- User should have know-how of android applications

2.2.4 Constraints

The system is developed under these limitations:

- **Regulatory policies**
  Considering that this is a commercial application intended for business use, the vehicles on this application have to follow a checklist that would ensure that cars registered do not bypass any legal or technical loopholes. The vehicles would be forced to display custom stickers on front and back windshields, and will be required to get a fitness certificate, which will be issued by the Motor Vehicle Examiner

(MVE). In the near future, the vehicles could also be required to convert their vehicle status from private to commercial

- **Hardware limitations**
  The application will not run on android version older than Jelly Bean/SDK 17

- **Interfaces to other applications**
  The proposed system will not support the interfaces of other applications

- **Parallel operation**
  The user can be either a rider or a driver at a time.
  Multiples rides cannot be requested at once.

- **Reliability requirements**
  Users would certainly demand system reliability. They would demand a high likelihood of finding a match for any given trip. This requires the system to be able to generate a large number of potential matches, or a smaller number of good matches for each potential rideshare. This, in turn, would require, among other things, a significant or concentrated pool of users.

- **Criticality of the application**
  Connecting maximum number of related users is the critical point of our application as the core function is to share a ride.

- **Safety and security considerations**
  Drivers need to register themselves first, a driving license is required. Additionally they have public and private mode. Private gives them decision autonomy about whom they share a ride with. Both drivers and riders have group code, so they can share a ride with people of an organization of their own choosing e.g. co-workers.

2.2.5 Assumptions and dependencies

- The applications target platform is android.
- The application depends on google maps platform for navigating
- Networking calls are made by using ION and web sockets. Their availability will affect the SRS of our application.
- The android application will not work on IOS platform.

2.2.6 Apportioning of requirements

Some software requirements specifications that could be expected from future versions are as follows:

- Card or smart payment methods.

- Cross platform availability.
- Purpose Sharing.
- Website and phone number for rides available.

2.3 Specific requirements

Specific requirements include both functional and no-functional requirements which are as follow:

2.3.1 Functional Requirements

- **User**

  The following functional requirements are of both riders and the drivers

Sign-up

The process by which the user registers him/herself into the system.

Sign-in

Process by which users can access our application by authenticating themselves by using their mobile numbers.

Search for a ride

Users can search for users depending upon their source, destination and number of seats required.

Driver searching for riders and rider searching for drivers. Users can also search for a ride by using their group code (A unique group code assigned to a particular organization using that group code users can search for other users going to same destination)

Add a ride

User can add a ride specifying his source, destination, number of seats required/available and date and time for potential drivers.

Profile settings

A user should be able to update his profile information e.g. name, phone number etc.

Ride history

A user should be able to view his past rides.

Transaction history

The user should be able to check his monitory transactions of rides taken.

Initiating a ride

The driver after accepting a ride should go to the agreed pick-up point and start the ride.

Completing a rider

After reaching the destination, the driver should be able to end the ride he started and collect the fare.

Switching the mode
The driver should be able to switch between the modes, private and public.

Switching the roles
The user can switch roles between rider and the driver. It is to be noted that user can be either a driver or rider at a time.

- **Admin**

Sign-in
Admin should be able to sign-in to the system by authenticating themselves by using the id and password

View users
Admin should be able to see the users registered in the system.

Add an organization
Admin should be able to add an organization and a unique group code against it.

Block user
Admin should be able to block the selected users.

Unblock user
Admin should be able to unblock the selected users blocked previously.

Remove user
Admin should be able to remove a user, user could be a rider, driver or an organization.

2.3.2 Non-functional Requirements

Usability
Considering the usability perspective, the application should be easy to use, it should have a simple and clean user interface designed by following the design principles of the Human Computer Interaction, and no necessary technical training should be needed for users to use the application.

Reliability

For the users of a system it is the reliability of the system as a whole that is meaningful. The app should be designed keeping this principle in mind**;** the application should be duly tested to find out and remove any pressure or crash points improving the system making it more reliable and useful.

Performance

The system should have a fast response time. Should be able to handle large number of requests simultaneously, without wearing out the system or increasing the response time.

Design Constraints

The design should be robust and less prone to crashes. The current design do not have audio and visual aids limiting the users having retinal and auditory impairment

Maintainability

The system proposed should be easily maintainable requiring fewer individuals and less man-hours. The maintenance should also be cost effective.

Portability

As the proposed system is an android application, it is very easy to carry. It will work in every environment that has access to GPS and internet.

License agreement

To deploy the application on google we need a license and a developer account. We should also acquire a patent to protect our intellectual property from idea theft

Chapter 3:
Use Case Analysis

Use Case ID	UC_01	
Use Case Name	Sign In	
Description	Login in to Bhai Chara	
Primary Actor	User (Both Driver and Rider)	
+Secondary Actor	None	
Pre-Condition	User must be registered	
Post-Condition	User Logged In	
Basic Flow	Actor Action	System Action
	Enters Phone Number Clicks On Verify Button	Logs in the user
Alternate Flow	None	

Table 27 Use case Login

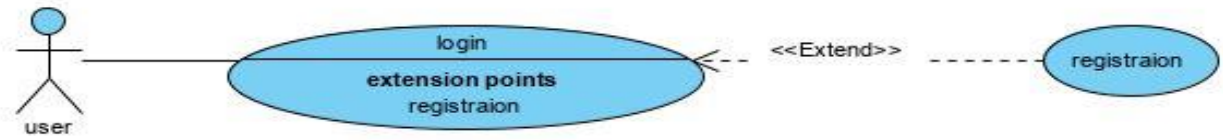


Figure 1 Use case Login

Use Case ID	UC_02	
Use Case Name	Sign Up	
Description	Creates User Account for Bhai Chara	
Primary Actor	Rider/Driver	
+Secondary Actor	None	
Pre-Condition	A user should not already have an account.	
Post-Condition	User account created	
Basic Flow	Actor Action	System Action
	Enters information required Name, Email, Group Code	Stores information about the user and creates user account
Alternate Flow	Another way to work with this function	

Table 18 Use case Sign Up

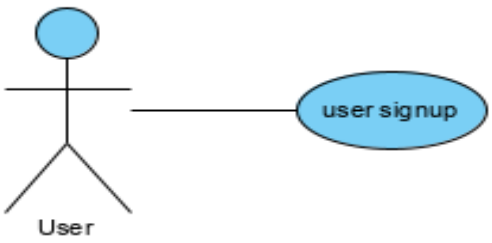


Figure 2 Use case Sign Up

Use Case ID	UC_03	
Use Case Name	Search For A ride	
Description	Rider should be able to search for drivers	
Primary Actor	Rider	
+Secondary Actor	None	
Pre-Condition	User Should be logged in Should provide source, destination and number of seats required	
Post-Condition	Suitable Drivers Are found	
Basic Flow	Actor Action	System Action
	Provides source, destination and the number of seats required Clicks the search button	System will search the drivers according to given criteria and provide the driver details
Alternate Flow	None	

Table 29 Use case Search for A ride

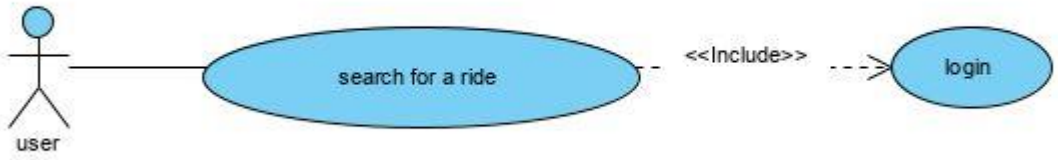


Figure 3 Use case Search for A ride

Use Case ID	UC_04	
Use Case Name	Profile Settings	
Description	User can update his or her profile settings	
Primary Actor	Rider/Driver	
+Secondary Actor	None	
Pre-Condition	User must have an account	
Post-Condition	Profile Updated	
Basic Flow	Actor Action	System Action
	User perform the required updates Clicks the update button	System should update the record according the specified details
Alternate Flow	None	

Table 30 Use case Profile Settings

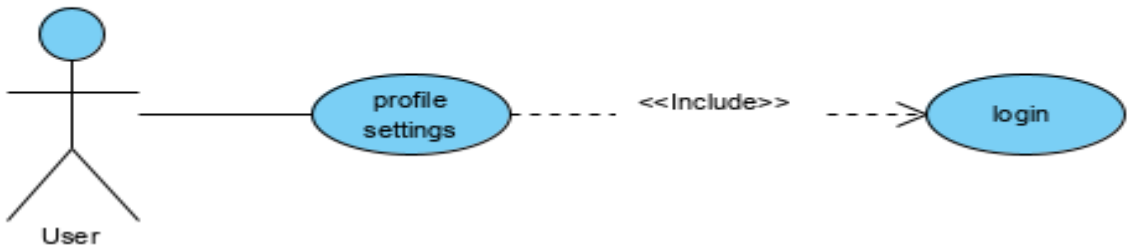


Figure 4 Use case Profile Settings

Use Case ID	UC_05	
Use Case Name	Transaction History	
Description	Users should be able to view the transactions of the rides taken	
Primary Actor	Riders/Drivers	
+Secondary Actor	None	
Pre-Condition	User Should be logged In	
Post-Condition	Transaction history displayed	
Basic Flow	Actor Action	System Action
	Clicks on the transaction History button	Provides the transaction details requested
Alternate Flow	None	

Table 31 Use case Transaction History

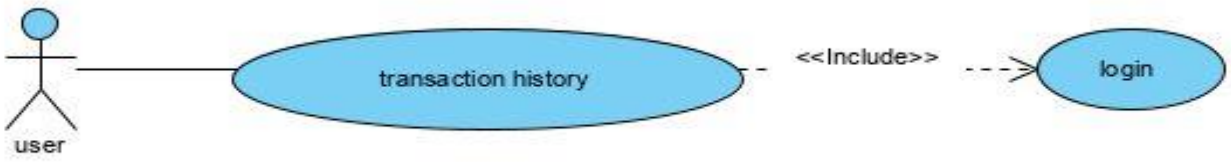


Figure 5 Use case Transaction History

Use Case ID	UC_06	
Use Case Name	Rides History	
Description	User Should be able to view the details of rides taken	
Primary Actor	Rider/Driver	
+Secondary Actor	None	
Pre-Condition	User Should be logged In	
Post-Condition	Rides History displayed	
Basic Flow	Actor Action	System Action
	Clicks on the Ride History button	Provides the rides history details requested
Alternate Flow	None	

Table 32 Use case Rides History

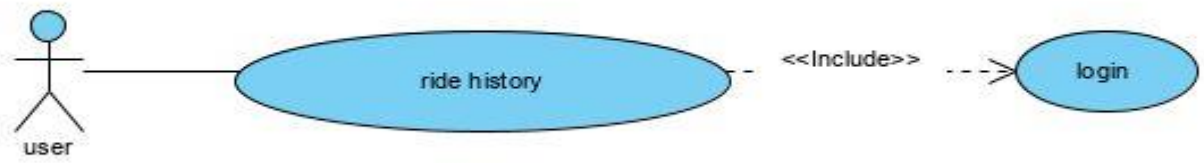


Figure 6 Use case Rides History

Use Case ID	UC_07	
Use Case Name	Switch Mode	
Description	Driver should be able to switch modes between private or public	
Primary Actor	Driver	
+Secondary Actor	None	
Pre-Condition	Driver Should be Active	
Post-Condition	Mode Switched	
Basic Flow	Actor Action	System Action
	Selects the mode public or private	Makes driver visible or invisible based on the mode selected
Alternate Flow	None	

Table 33 Use case Switch Mode

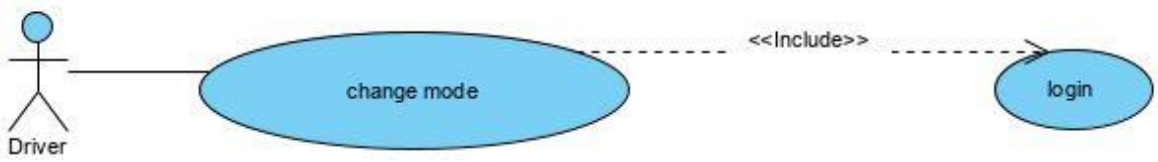


Figure 7 Use case Switch Mode

Use Case ID	UC_08	
Use Case Name	Start A Ride	
Description	Driver should be able to start a ride	
Primary Actor	Driver	
+Secondary Actor	None	
Pre-Condition	Reaches the pickup point and picks up the customer	
Post-Condition	Ride is started	
Basic Flow	Actor Action	System Action
	Driver after reaching at the agreed pick up point, picks up the customer and presses the start ride button	System stores the details of the rides i.e. time, distance travelled, the starting point.
Alternate Flow	None	

Table 34 Use case Start a Ride

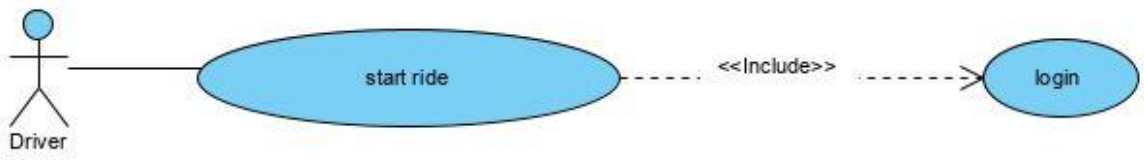


Figure 8 Use case Start a Ride

Use Case ID	UC_09	
Use Case Name	Ends A Ride	
Description	After completing the commitment driver should be able to end the ride	
Primary Actor	Driver	
+Secondary Actor	None	
Pre-Condition	Reaching the agreed destination Dropping off the rider	
Post-Condition	Ride ended	
Basic Flow	Actor Action	System Action
	After arriving at the destination, drops off the riders	System should record the ending point of the ride
Alternate Flow	None	

Table 35 Use case ends a Ride

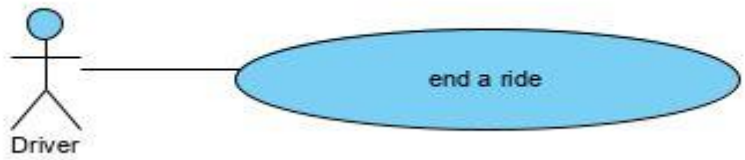


Figure 9 Use case ends a Ride

Use Case ID	UC_10	
Use Case Name	Collects the fare	
Description	Driver should be able to collect the fare of the ride	
Primary Actor	Driver	
+Secondary Actor	None	
Pre-Condition	Ride should be ended	
Post-Condition	Fare collected	
Basic Flow	Actor Action	System Action
	After completing a ride, driver collects the fare	System displays the fare based on ride done. Keeps the record of ride and fare.
Alternate Flow	None	

Table 36 Use case collects the fare

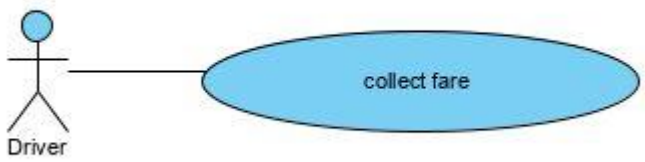


Figure 10 Use case collects the fare

Use Case ID	UC_11	
Use Case Name	Sign In	
Description	Should be able to log in to the system	
Primary Actor	Admin	
+Secondary Actor	None	
Pre-Condition	Admin must be registered	
Post-Condition	Admin logged in	
Basic Flow	Actor Action	System Action
	Admin provides id and password and clicks the log in button	System validates the id and password and allows access
Alternate Flow	None	

Table 37 Use case Sign In

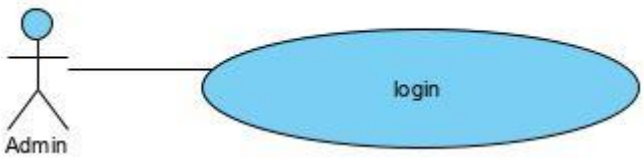


Figure 11 Use case Sign In

Use Case ID	UC_12	
Use Case Name	Adds an organization	
Description	Should be able to add an organization and assign an unique group code to that organization	
Primary Actor	Admin	
+Secondary Actor	None	
Pre-Condition	Admin must be logged in	
Post-Condition	Organization added	
Basic Flow	Actor Action	System Action
	Organization is assigned an unique group code and added to the system	System adds the given information into the database
Alternate Flow	None	

Table 38 Use case Adds an organization

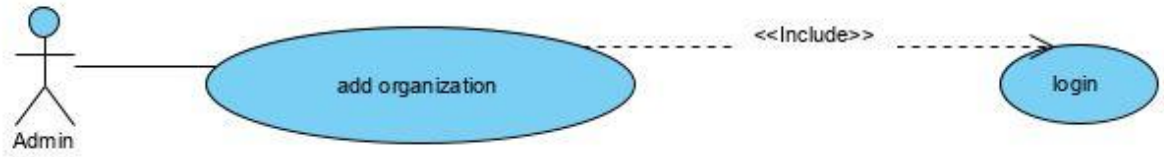


Figure 12 Use case Adds an organization

Use Case ID	UC_13	
Use Case Name	Blocks a user	
Description	Should be able to block a user	
Primary Actor	Admin	
+Secondary Actor	None	
Pre-Condition	Admin should be logged in	
Post-Condition	User blocked successfully	
Basic Flow	Actor Action	System Action
	Identifies a user and blocks him	System receives the command and blocks the user
Alternate Flow	None	

Table 39 Use case Blocks a user

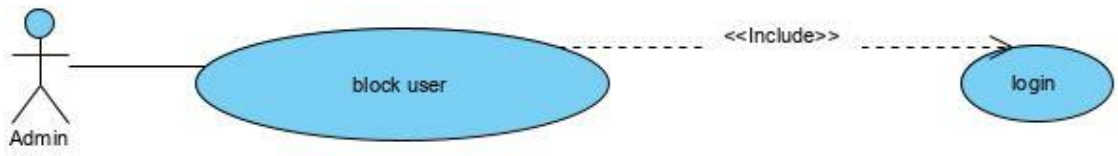


Figure 13 Use case Blocks a user

Use Case ID	UC_14	
Use Case Name	Unblocks the user	
Description	Should be able to unblock a user	
Primary Actor	Admin	
+Secondary Actor	None	
Pre-Condition	Admin must be logged in, Blocked user should be identified	
Post-Condition	User is unblocked	
Basic Flow	Actor Action	System Action
	Admin identifies the user to be unblocked and unblocks	System removes the user from the block list
Alternate Flow	None	

Table 40 Use case Unblocks the user

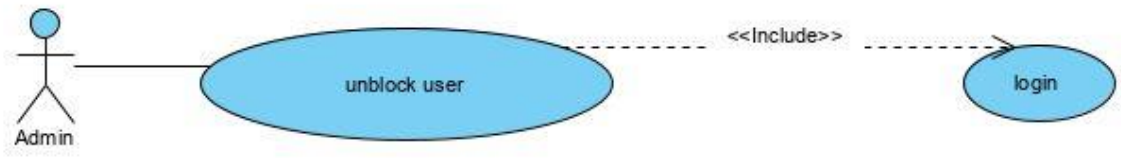


Figure 14 Use case Unblocks the user

Use Case ID	UC_15	
Use Case Name	View the users	
Description	Should be able to view the users using/registered in the application	
Primary Actor	Admin	
+Secondary Actor	None	
Pre-Condition	Admin must be logged in	
Post-Condition	Users viewed	
Basic Flow	Actor Action	System Action
	After logging in, viewing the users in the database	Provides the details requested
Alternate Flow	None	

Table 41 Use case View the users

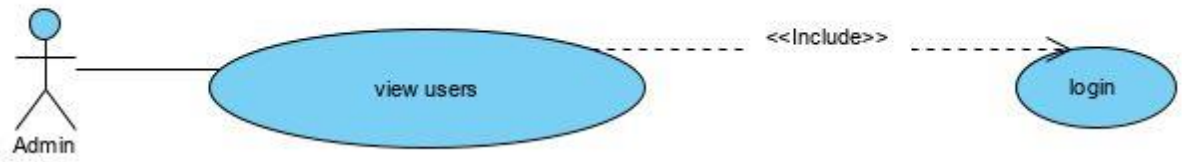


Figure 15 Use case View the users

Use Case ID	UC_16	
Use Case Name	Change roles	
Description	Should be able to switch between driver or rider	
Primary Actor	Driver/ Rider	
+Secondary Actor	None	
Pre-Condition	User must be logged in	
Post-Condition	Role Switched	
Basic Flow	Actor Action	System Action
	After logging in, switches the role between rider or driver	Rolls changed and provides system functionality accordingly
Alternate Flow	None	

Table 42 Use case Change roles

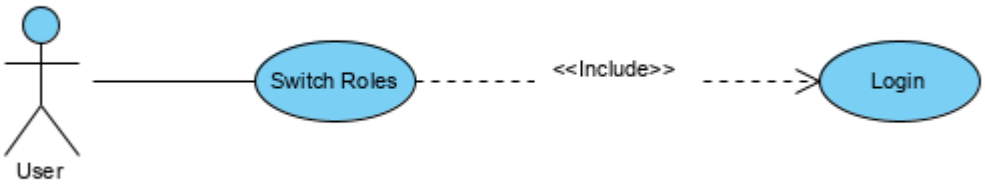


Figure 16 Use case Change roles

Chapter 4:
Design

4.1 Architecture Diagram

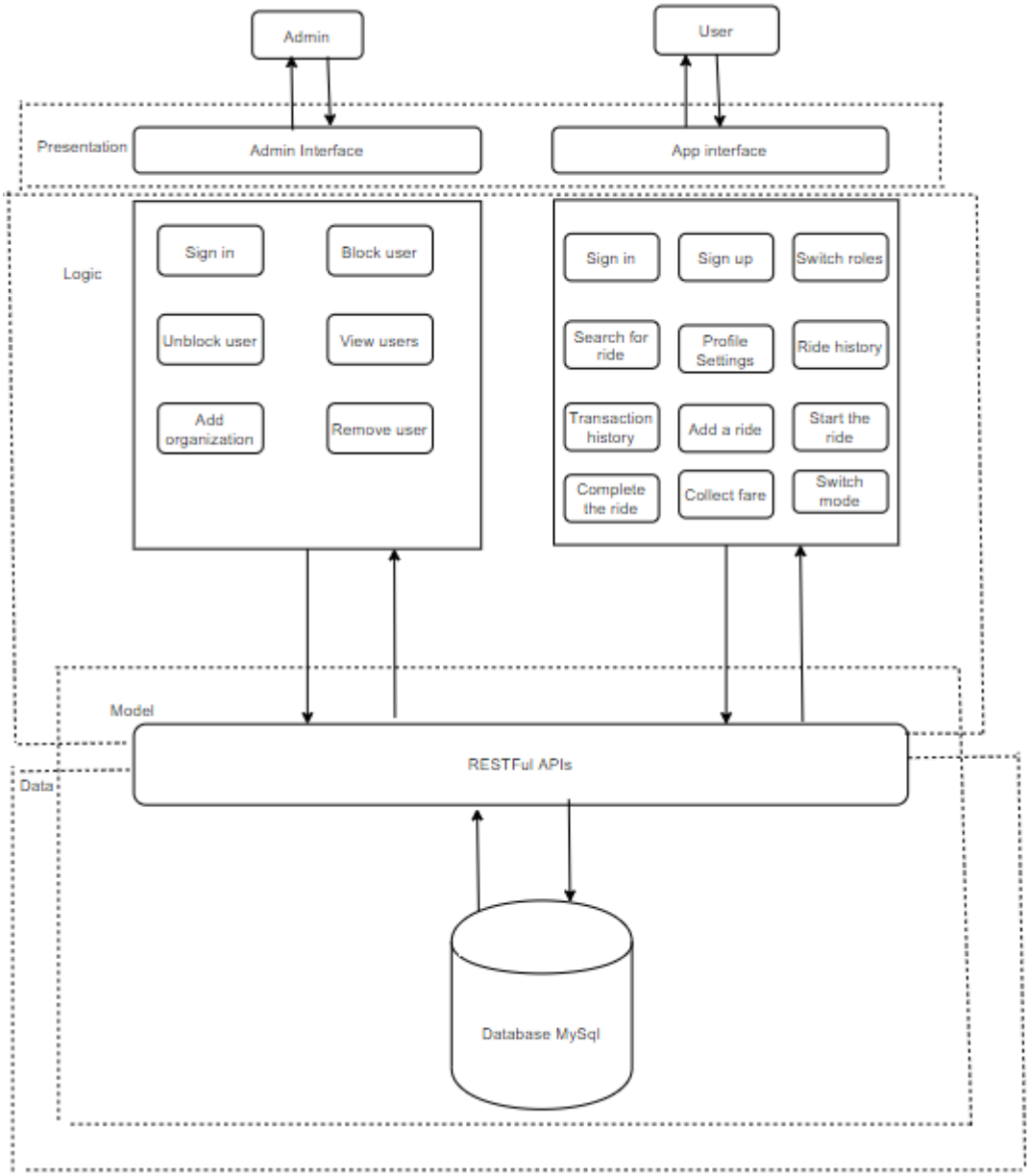


Figure 27 Architecture Diagram

4.2 ER Diagram

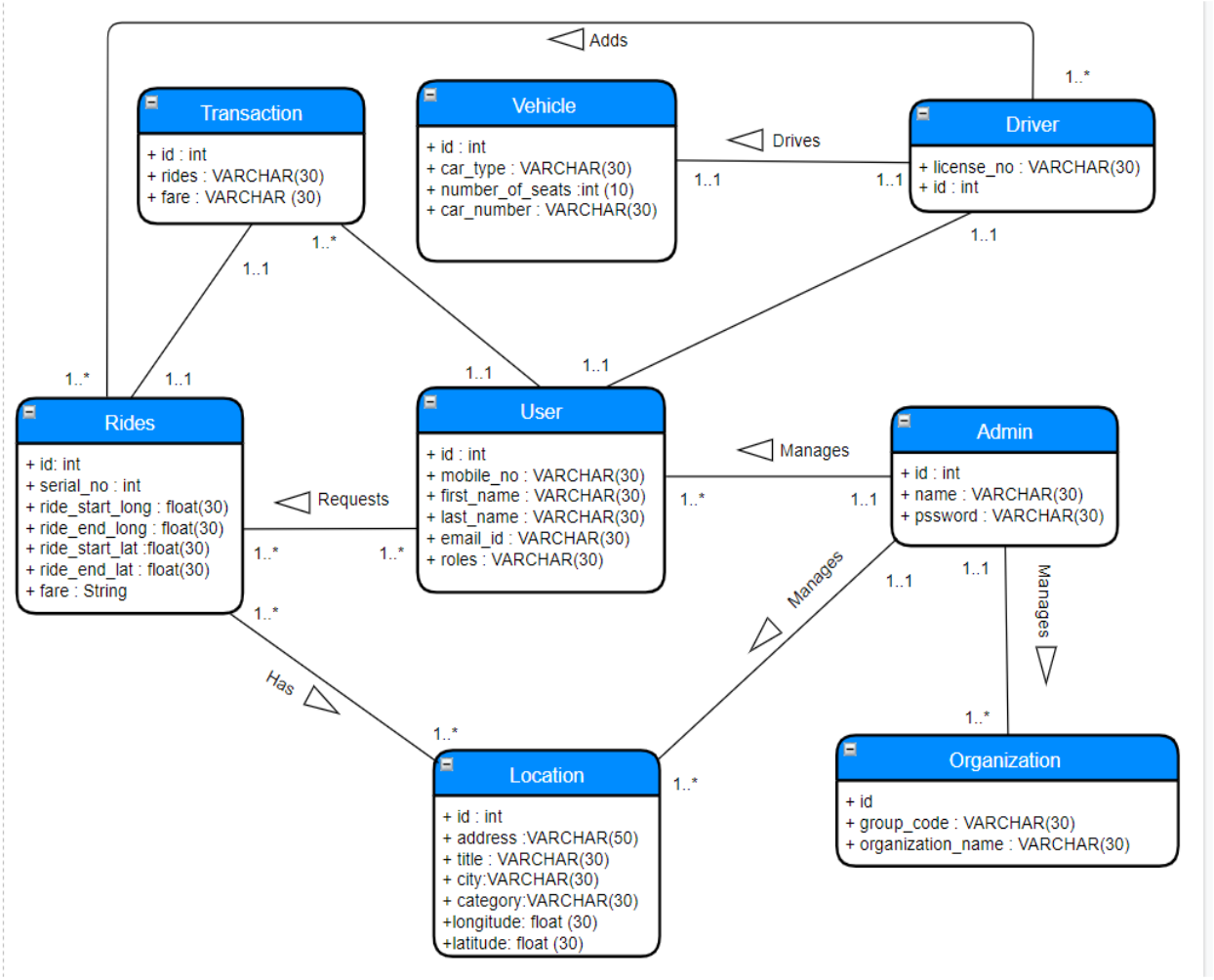


Figure 18 ER Diagram

4.3 Data Flow Diagram

4.3.1 The level 0

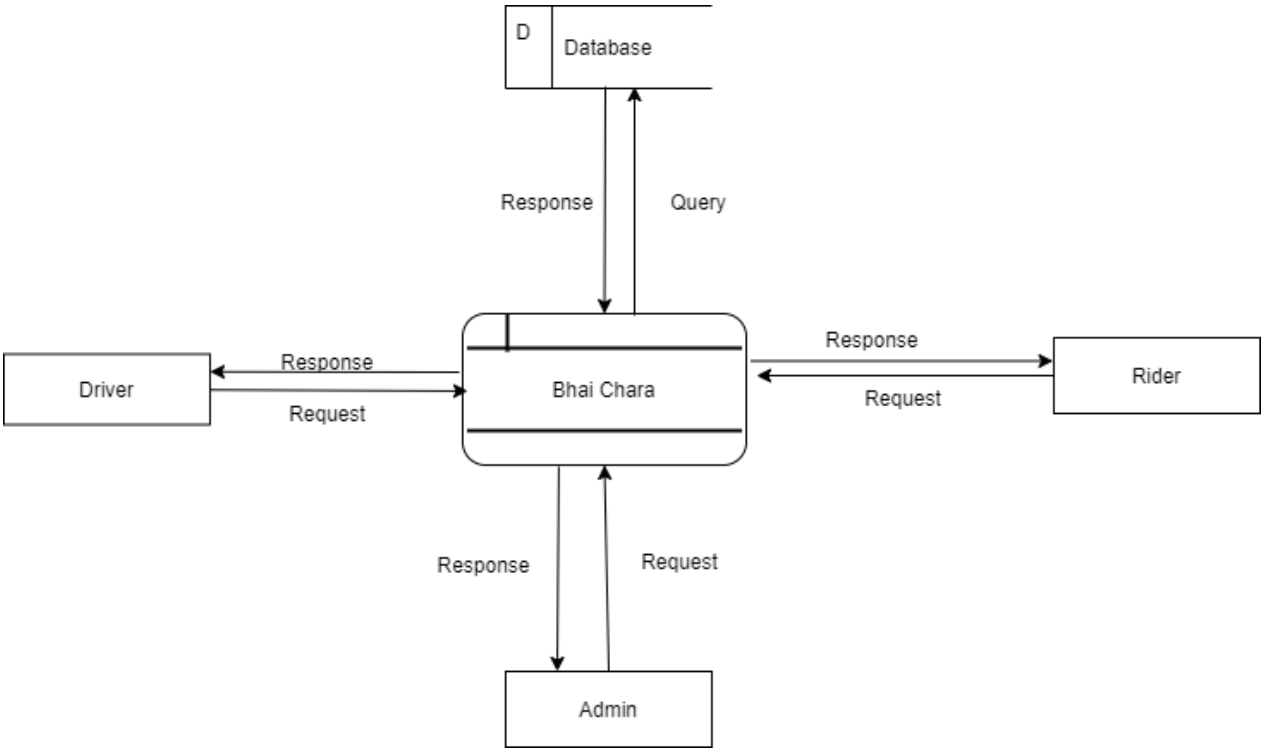


Figure 19 Data Flow Diagram level 0

4.3.2 The level 1

- Admin

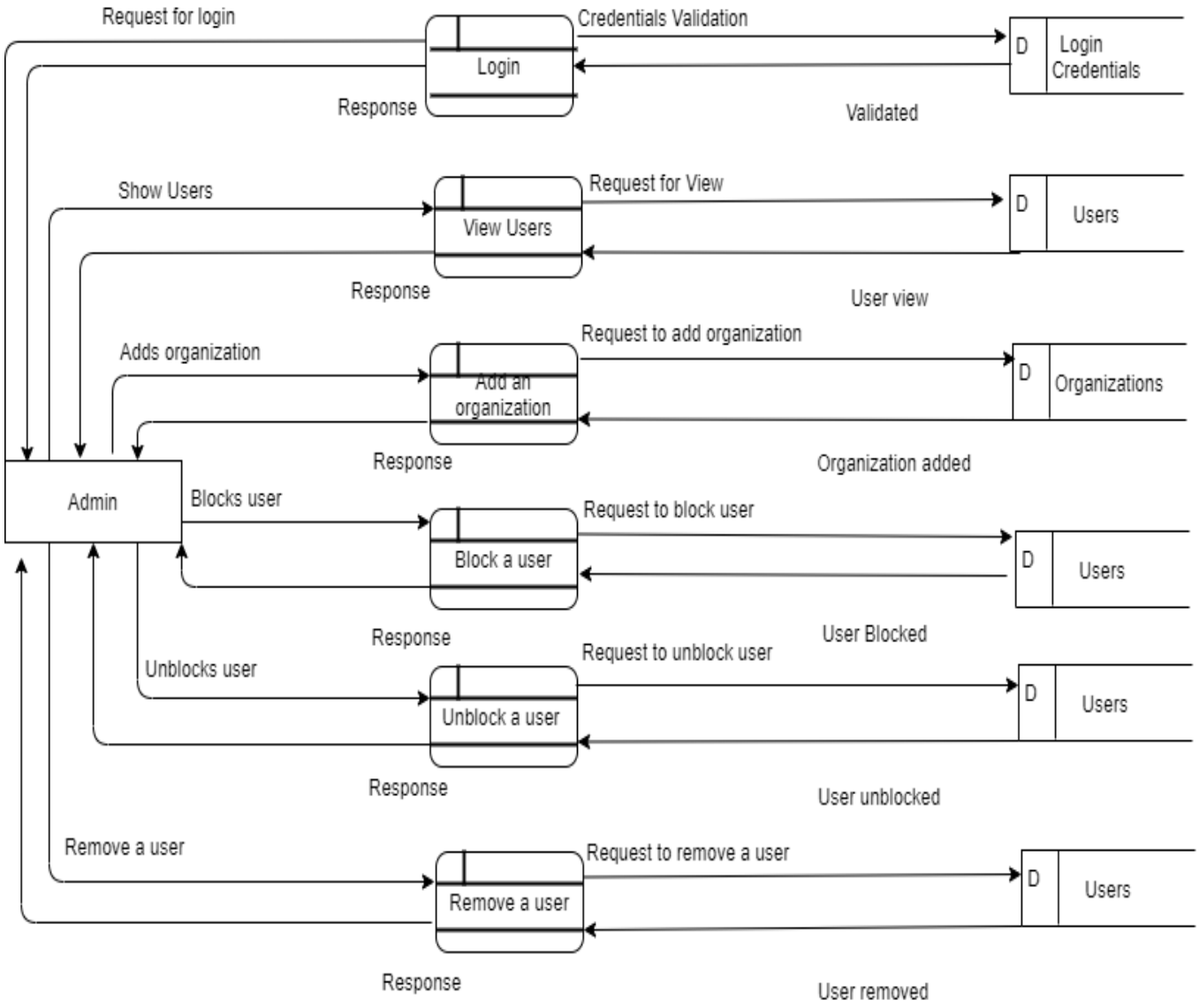


Figure 20 Data Flow Diagram level 1 Admin

- Driver

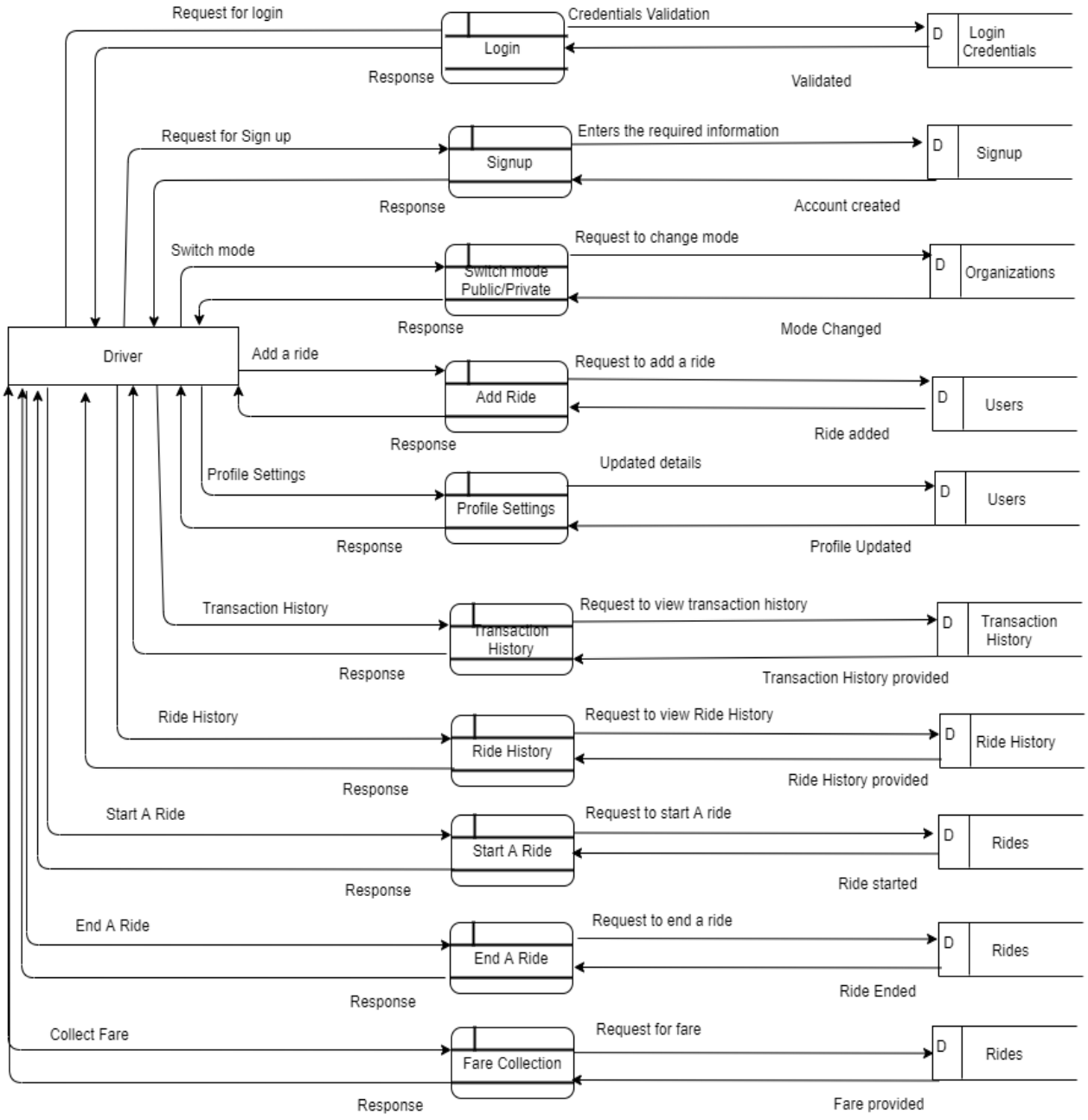


Figure 21 Data Flow Diagram level 1 Driver

- Rider

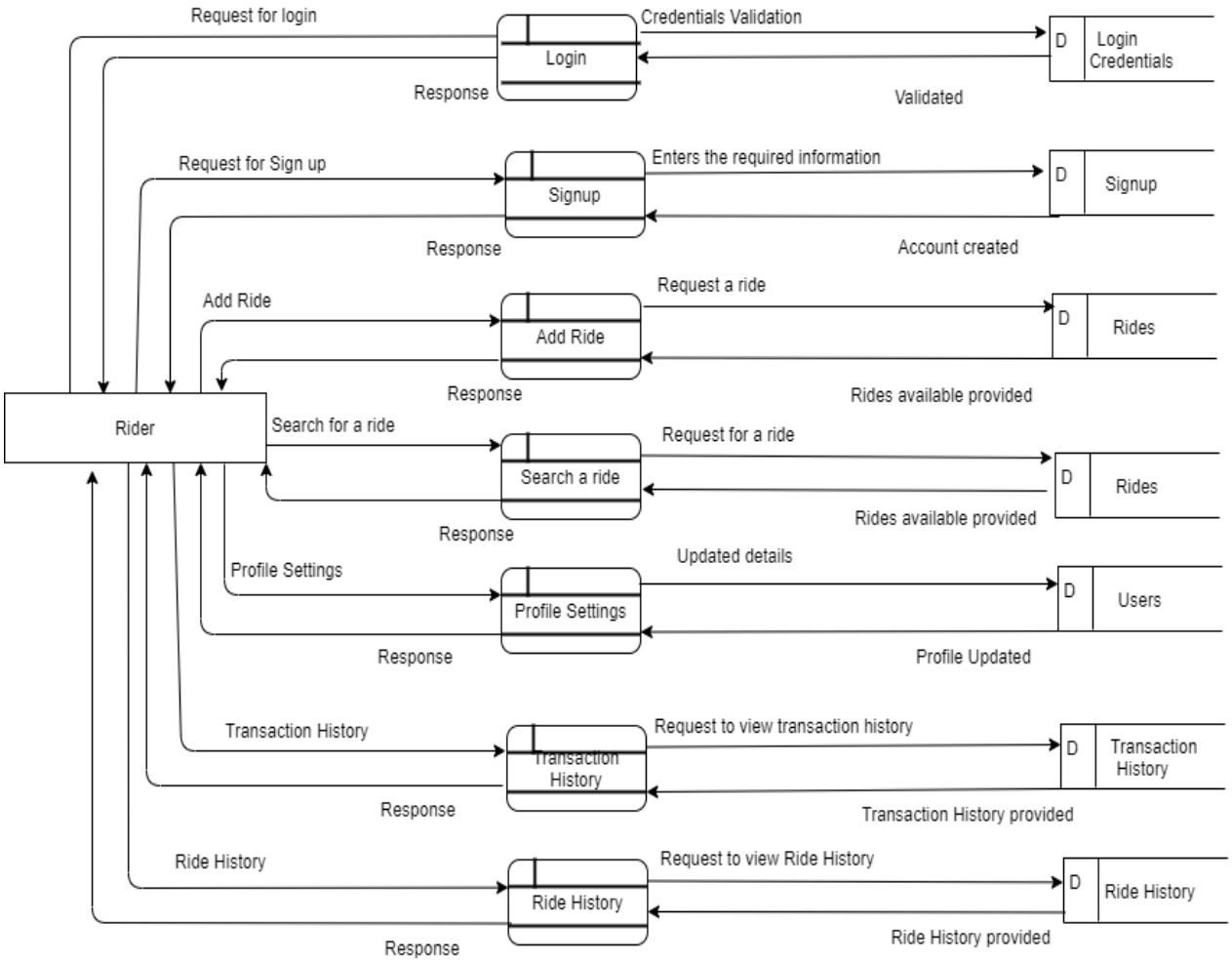


Figure 22 Data Flow Diagram level 1 Rider

4.4 Class Diagram

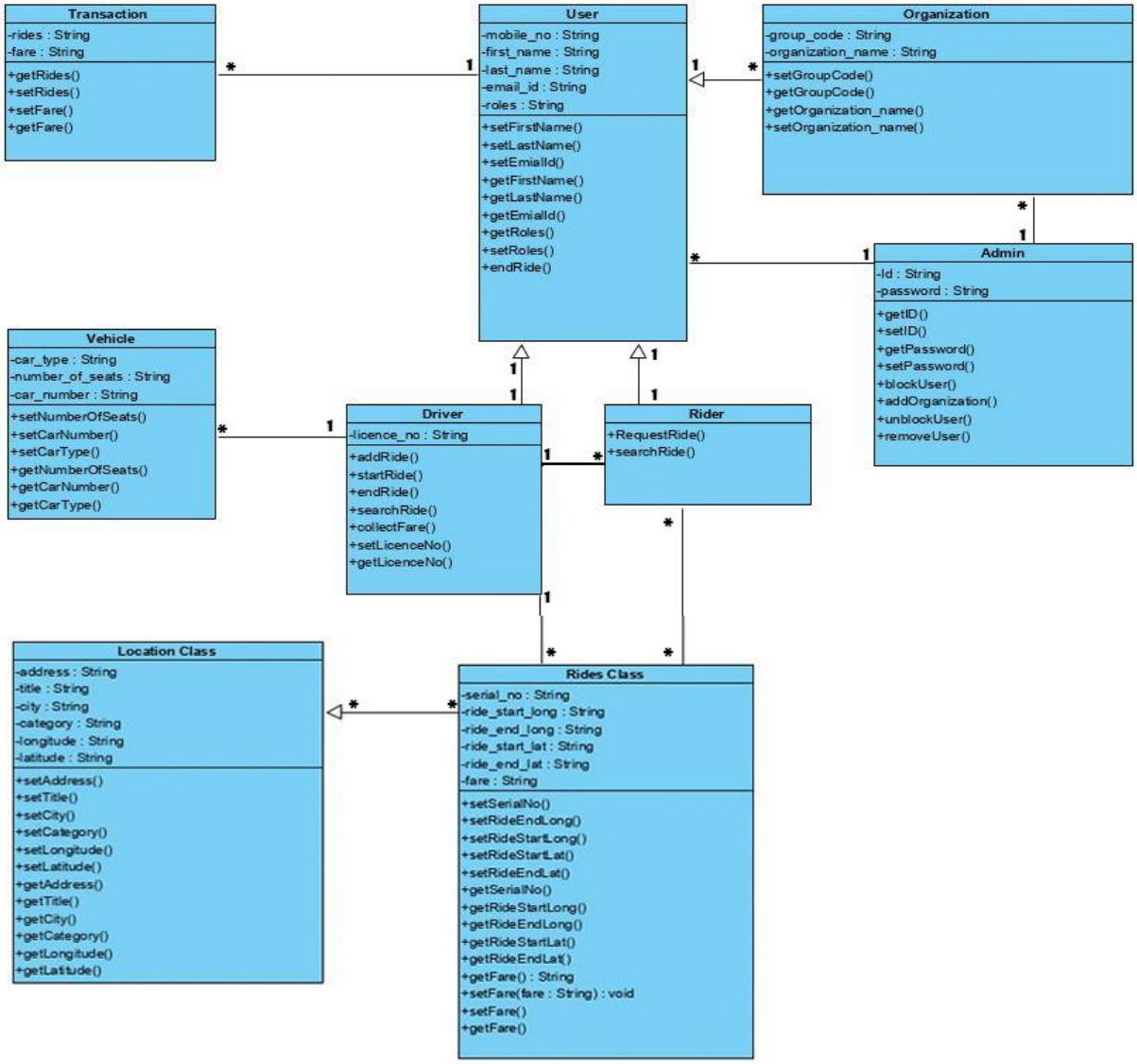


Figure 23 Class Diagram

4.5 Activity Diagram

Sign Up Activity

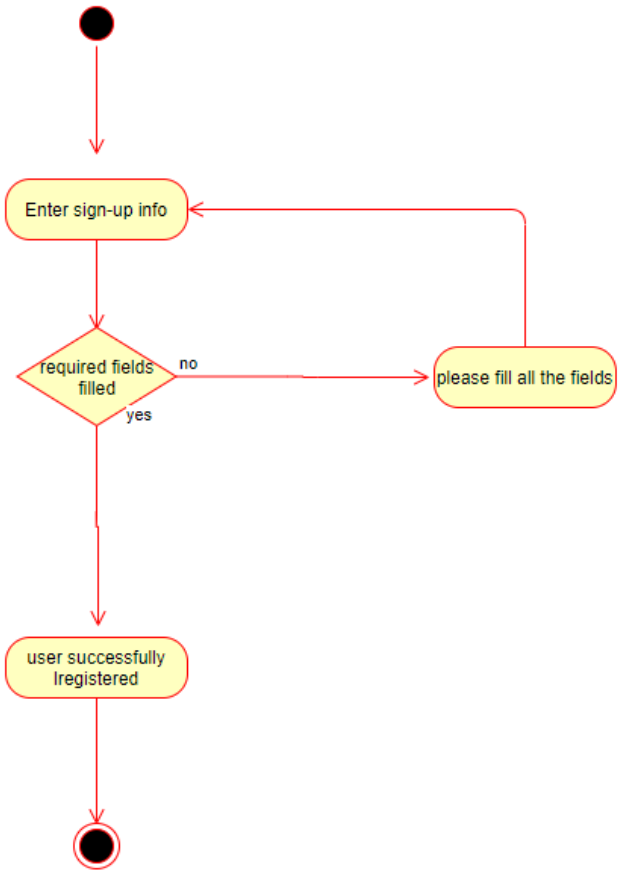


Figure 24 Sign up Activity diagram

Login activity
Admin

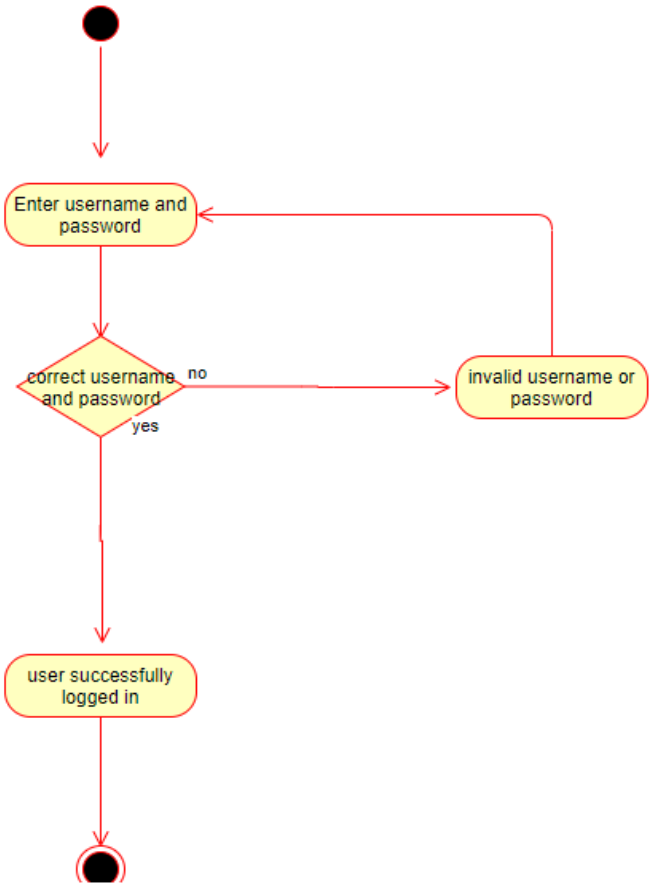


Figure 25 login Activity diagram

Search for a ride
Riders

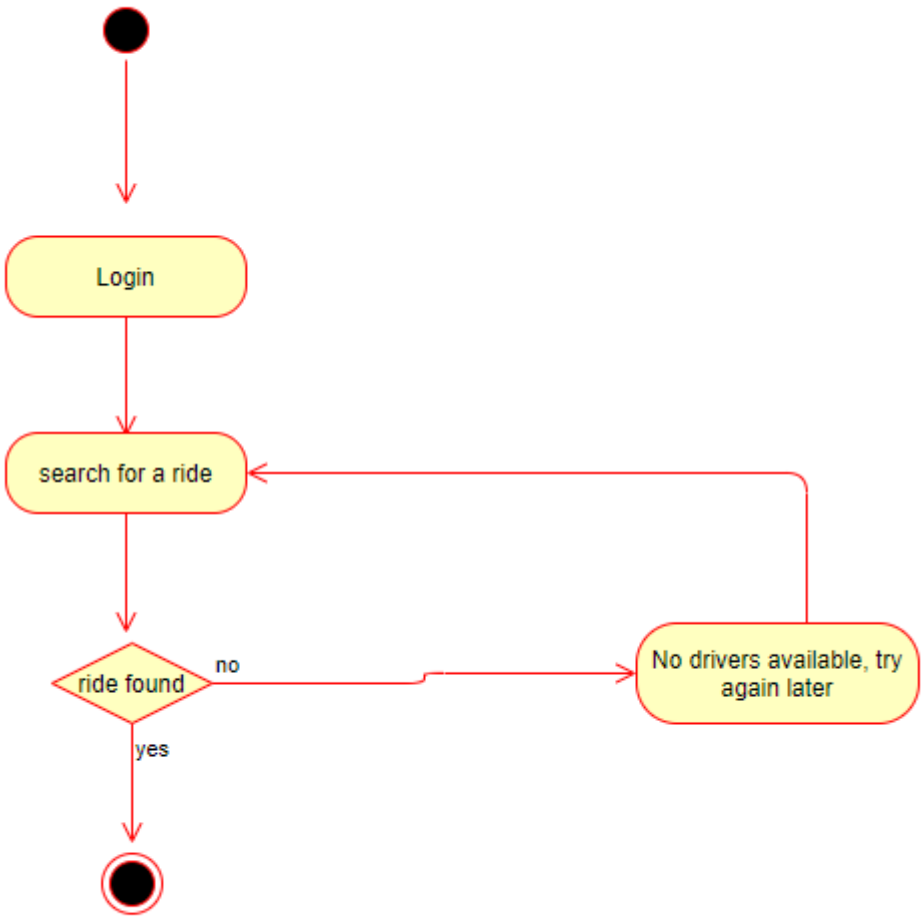


Figure 26 Search for a Ride Activity diagram

Add a ride
Drivers

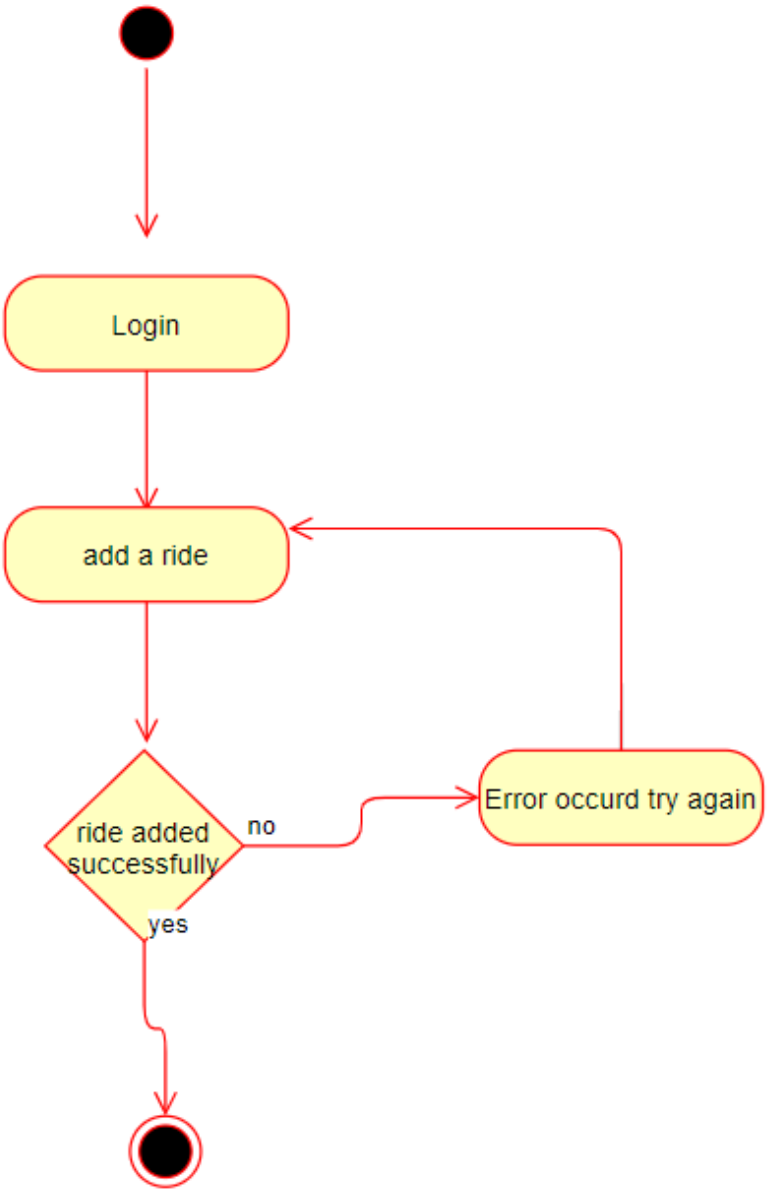


Figure 27 Add a Ride Activity diagram

Start a ride
Drivers

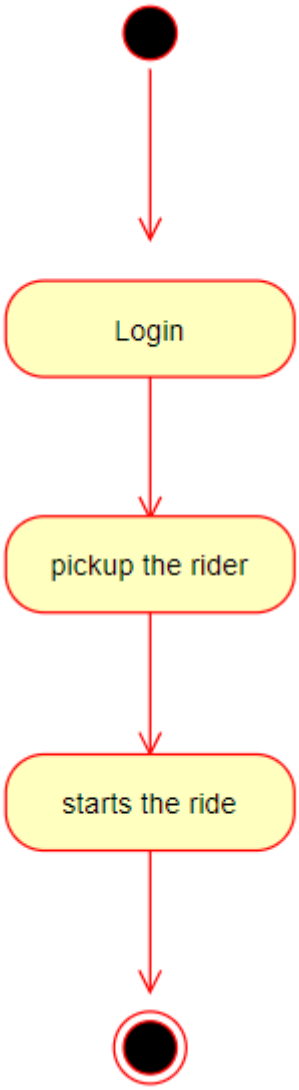


Figure 28 Start a Ride Activity diagram

End a ride
Driver

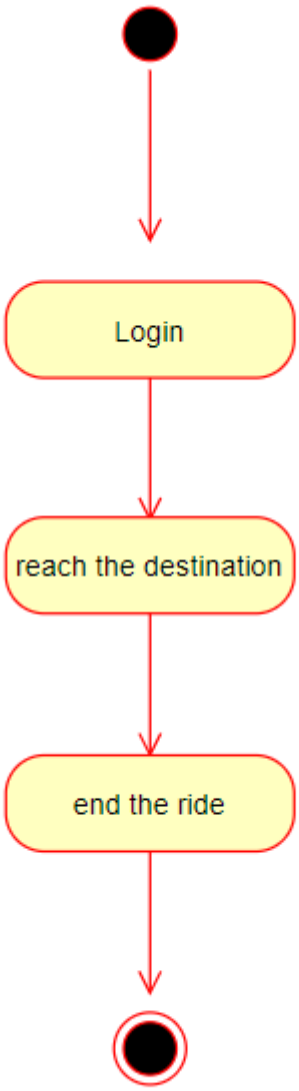


Figure 29 End a Ride Activity diagram

Collect fare
Driver

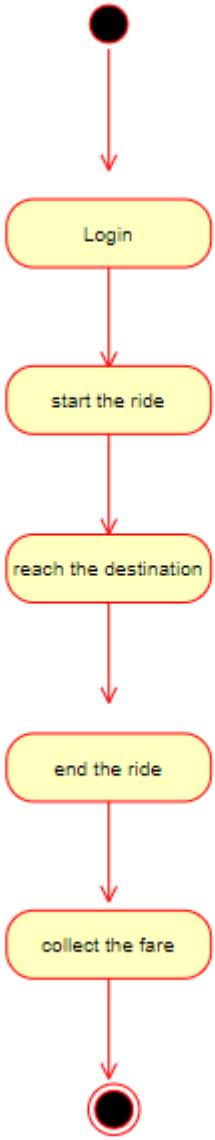


Figure 30 Collect Fare Activity diagram

View rides history

Driver/ Rider

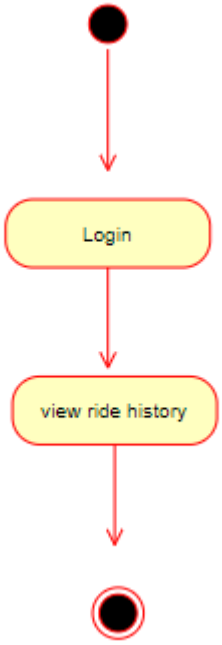


Figure 31 View Ride History Activity diagram

Rider/Driver

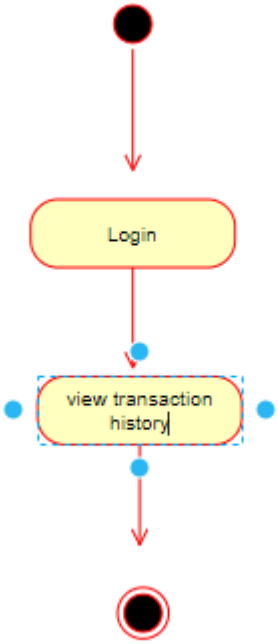


Figure 32 View Transaction History Activity diagram

Switch modes
Driver

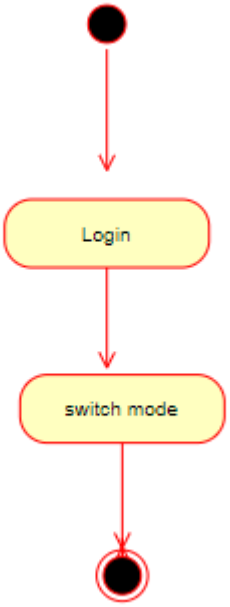


Figure 33 Switch Mode Activity diagram

Switch roles
Driver/Rider

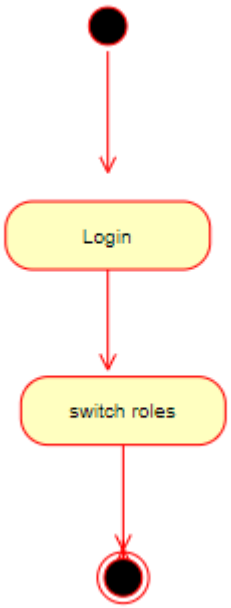


Figure 34 Switch Role Activity diagram

Add organization
Admin

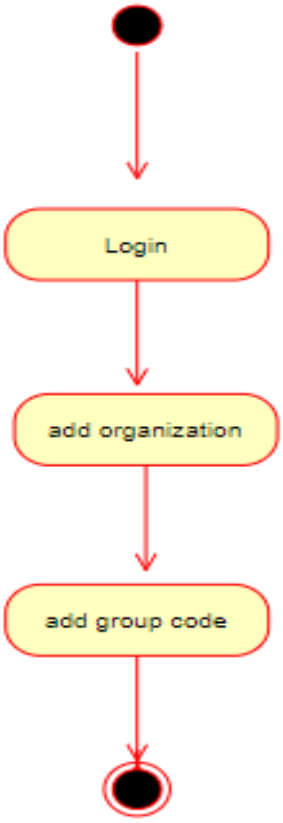


Figure 35 Add Organization Activity diagram

Block user
Admin

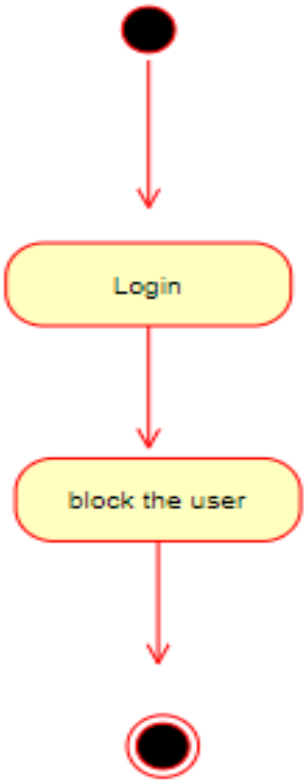


Figure 36 Block User Activity diagram

Unblock User
Admin

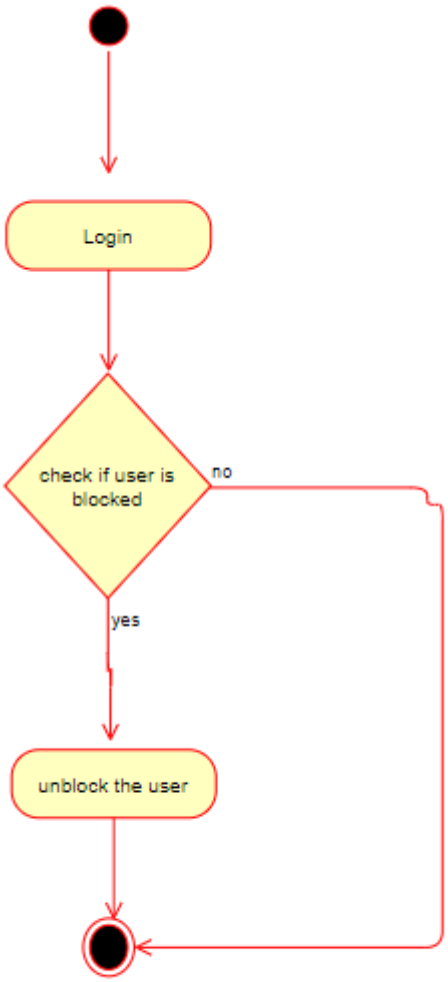


Figure 37 Unblock User Activity diagram

Remove user
Admin

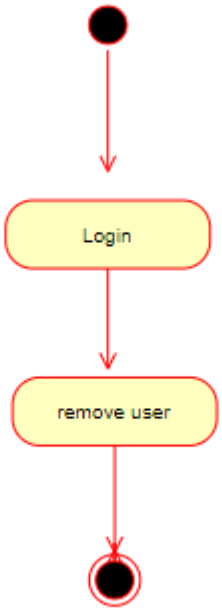


Figure 38 Remove User Activity diagram

4.6 Sequence Diagrams

Admin Block User

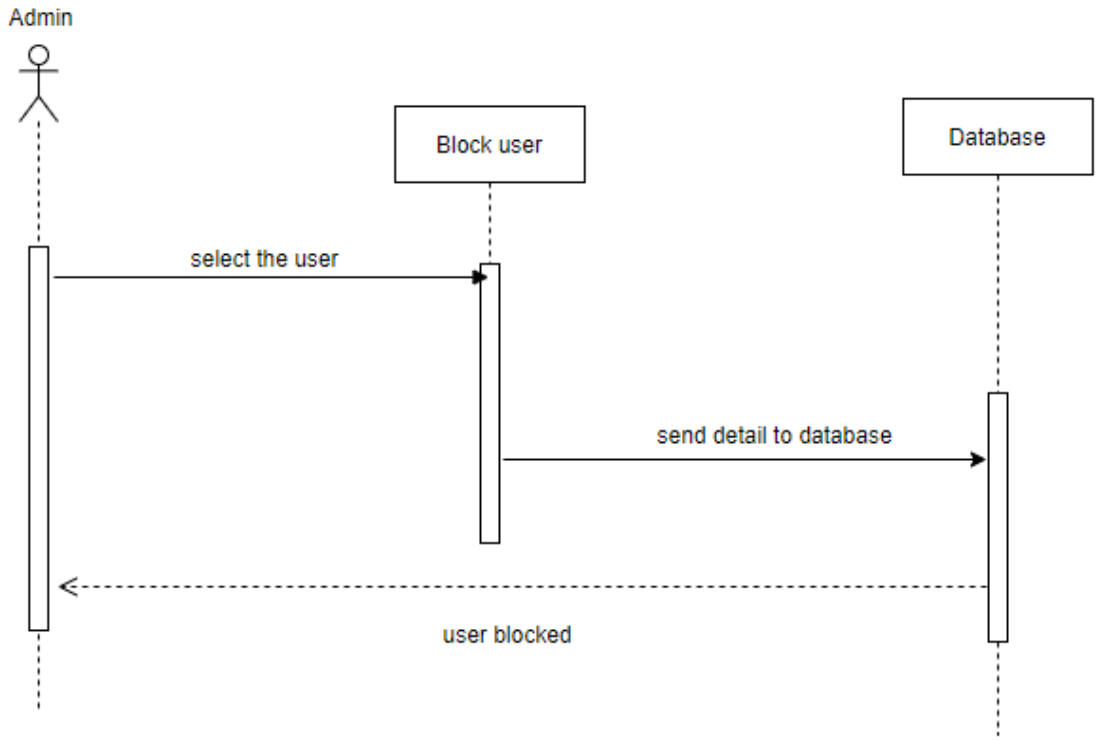


Figure 39 Admin Block User Sequence Diagram

Admin Login

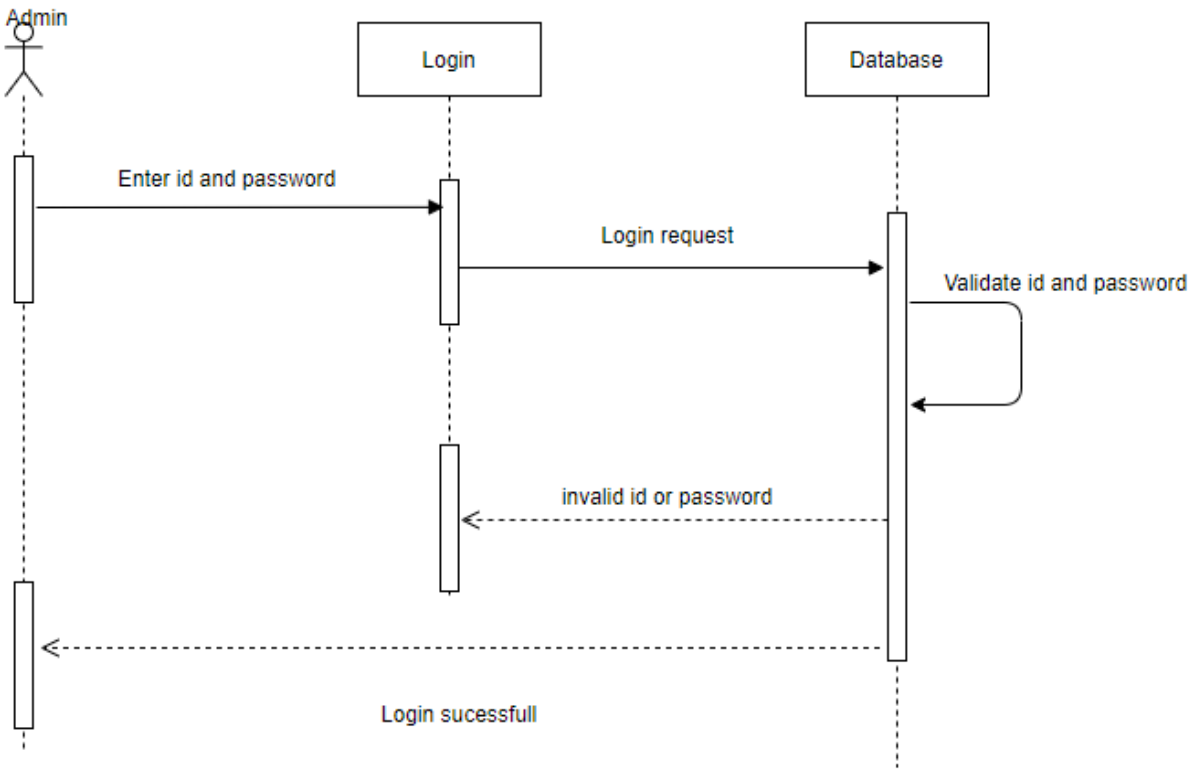


Figure 40 Admin Login Sequence Diagram

Add organization

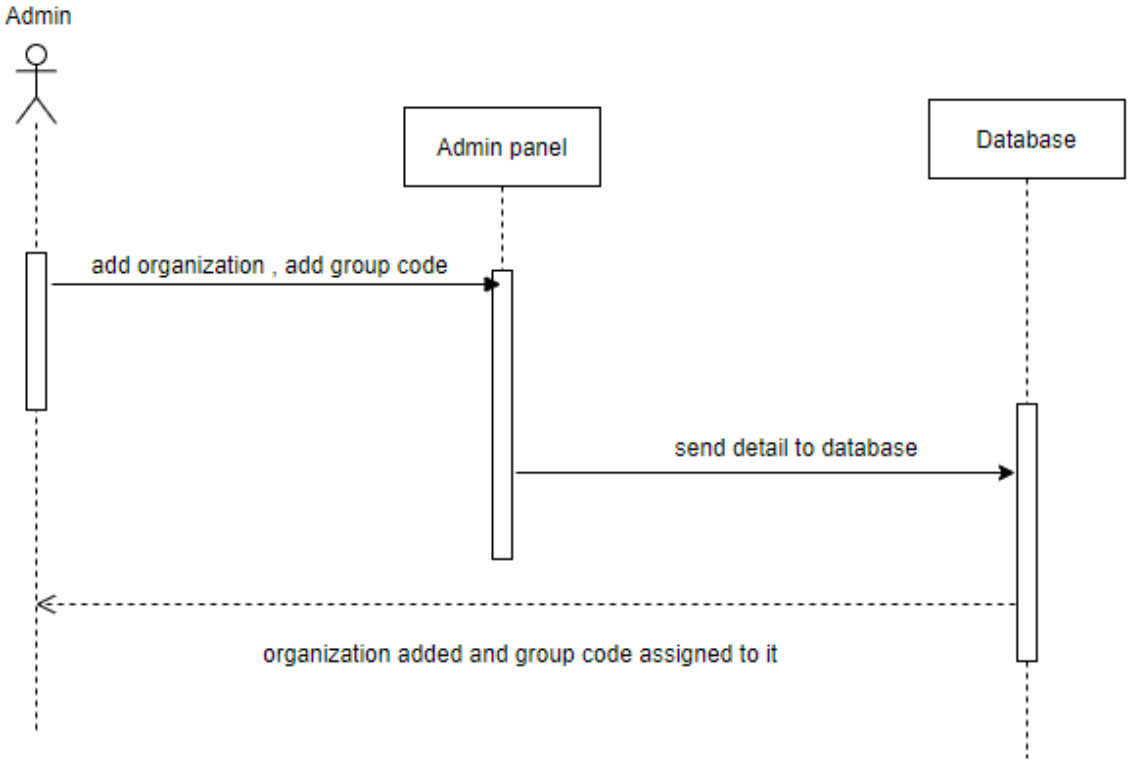


Figure 41 Add Organization Sequence Diagram

Admin Unblock the user

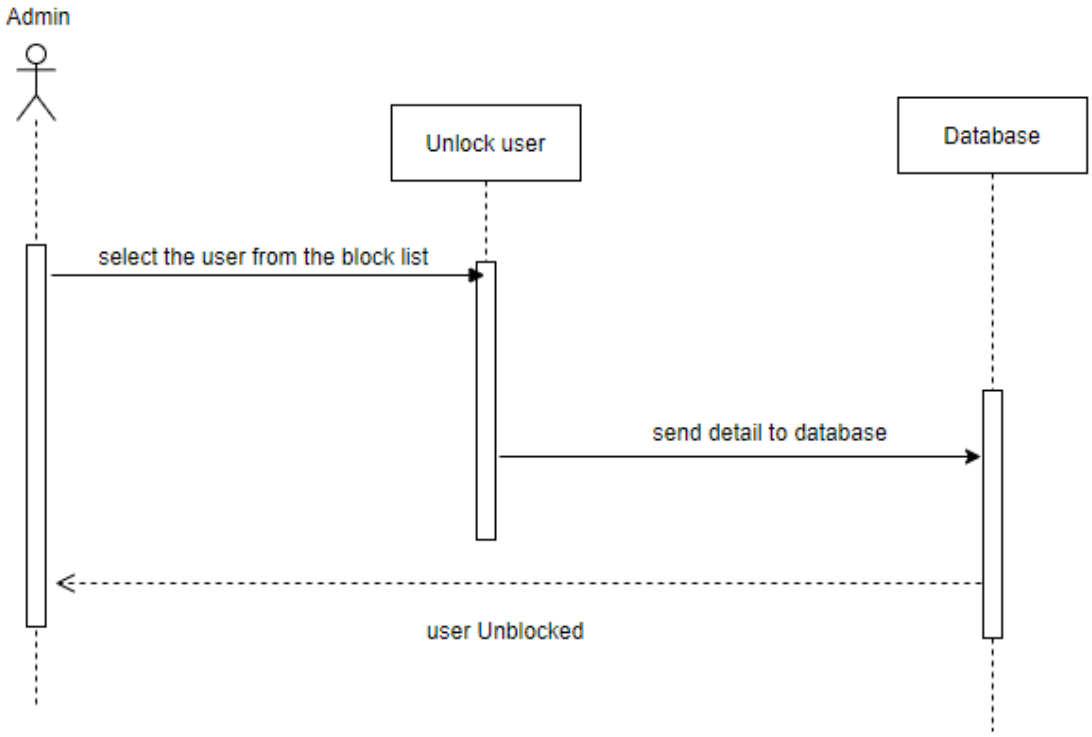


Figure 42 Admin Unblock the User Sequence Diagram

Admin Remove User

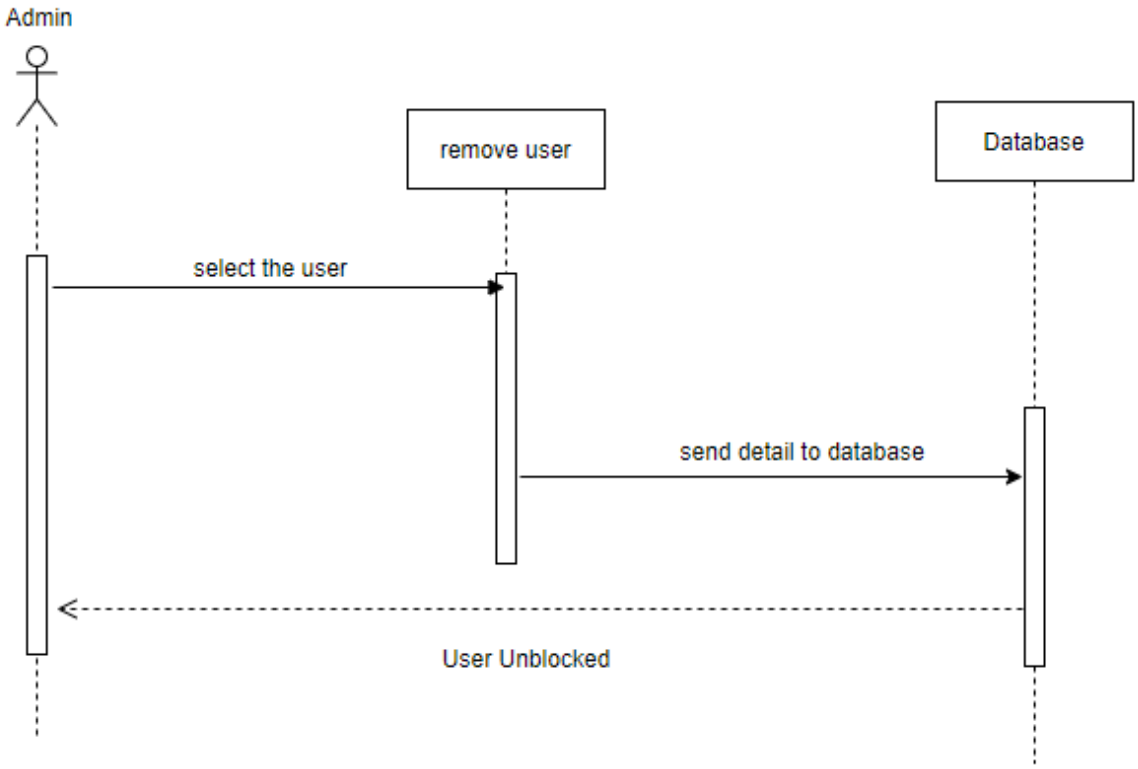


Figure 43 Admin Remove User Sequence Diagram

Driver starts the ride

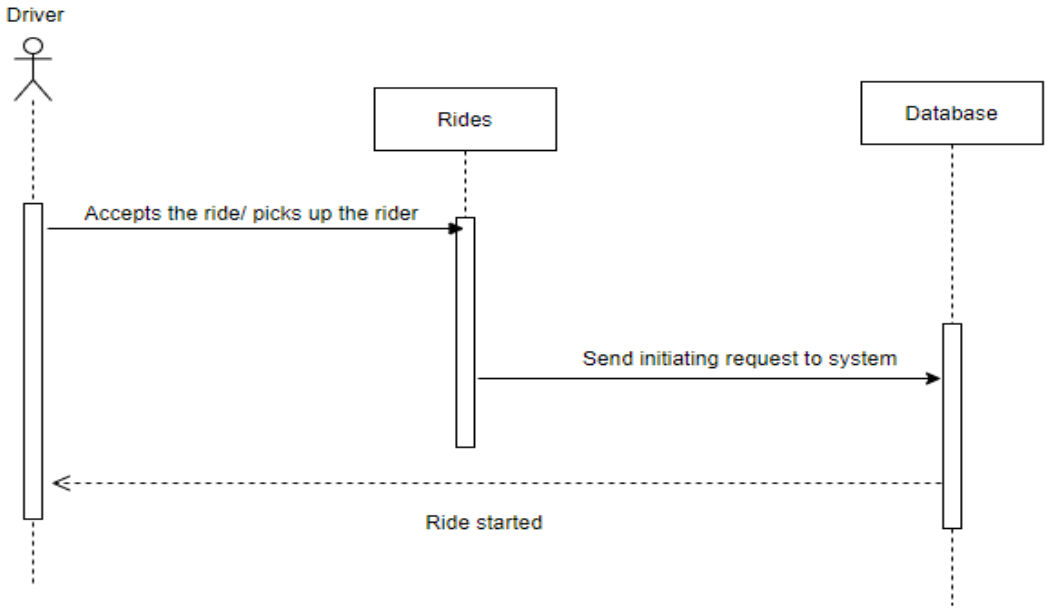


Figure 44 Driver Start the Ride Sequence Diagram

User SignUp
Driver/Rider

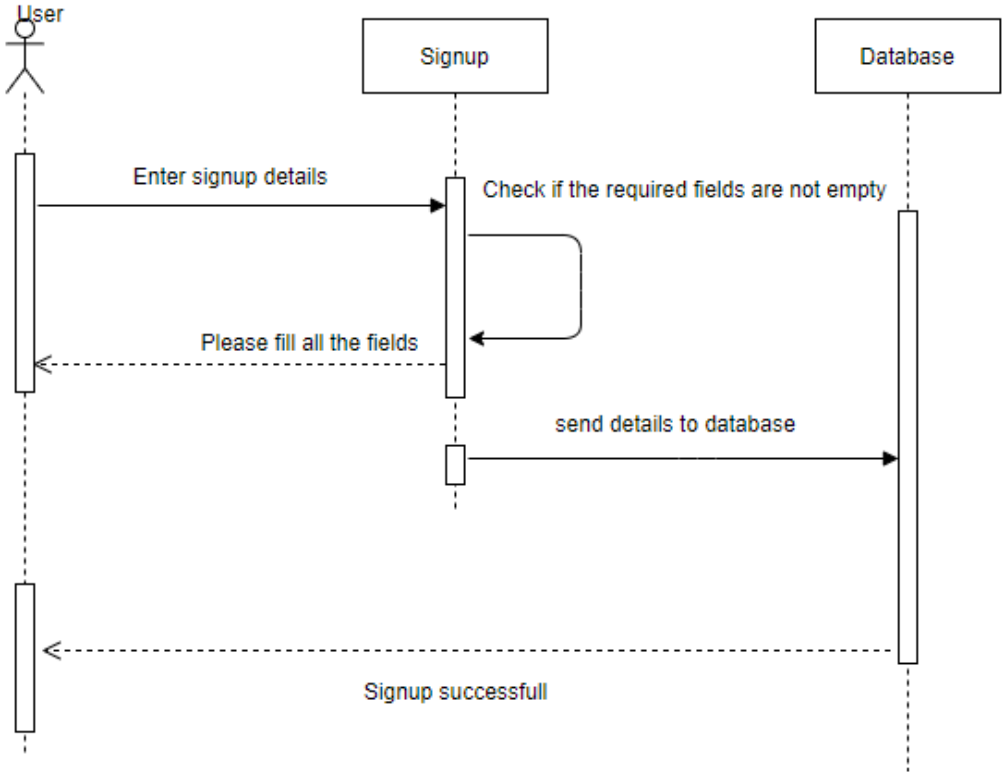


Figure 45 User Sign up Sequence Diagram

User Search Rides
Driver/Rider

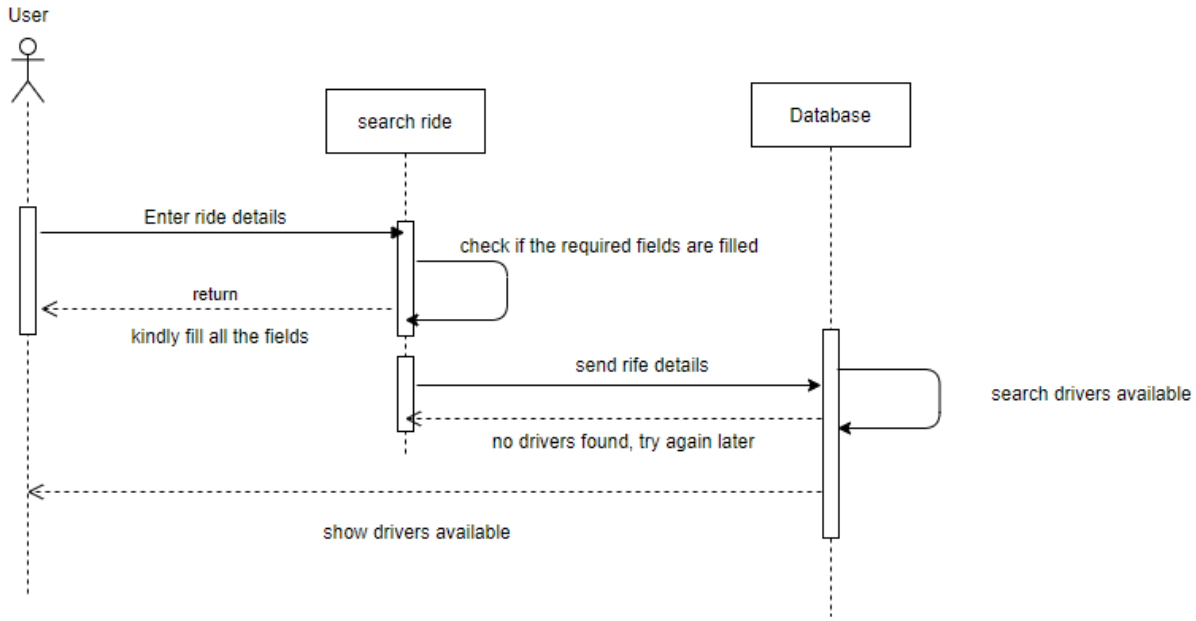


Figure 46 User Search Ride Sequence Diagram

User Login
Driver/Rider

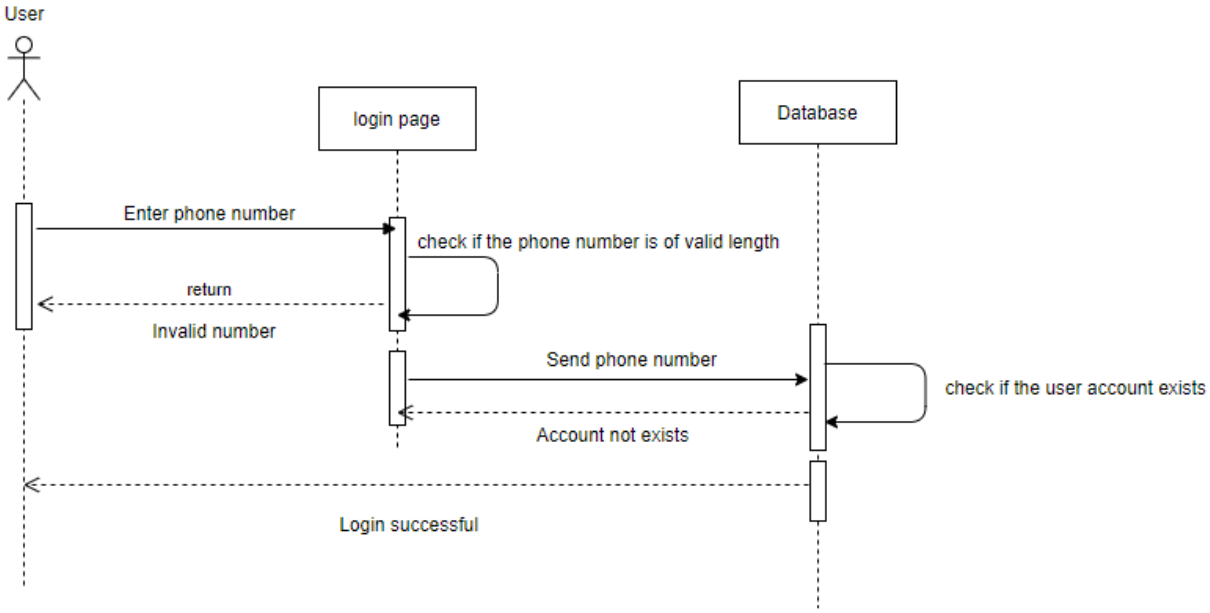


Figure 47 User Login Sequence Diagram

User Transaction History
Driver/Rider

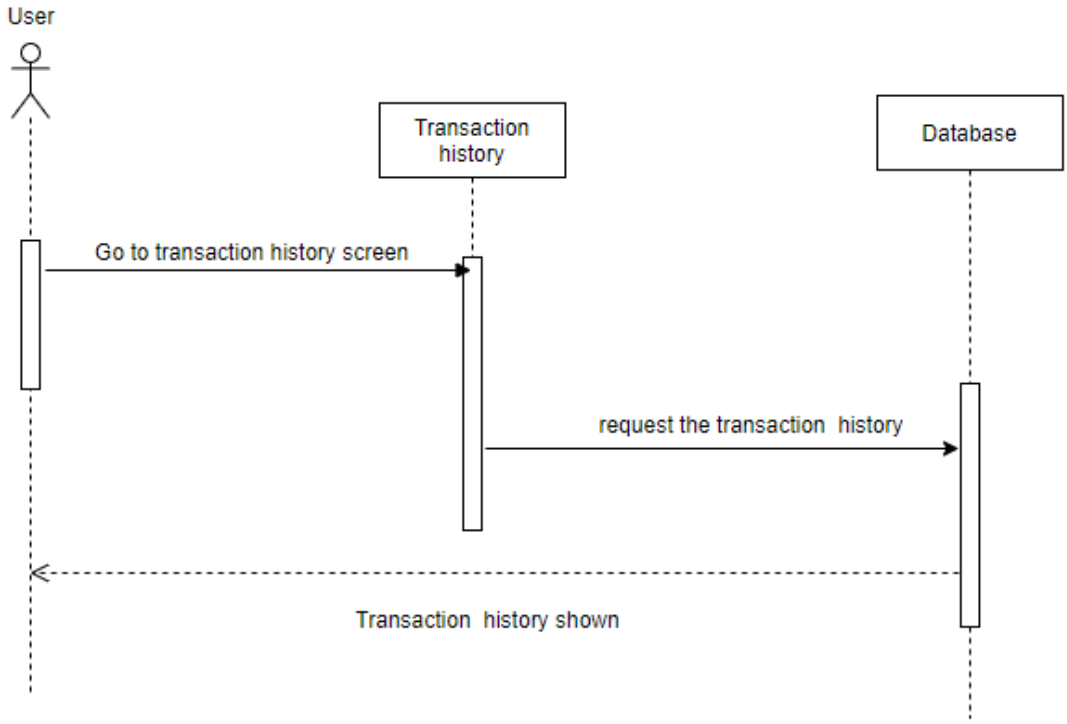


Figure 48 User Transaction History Sequence Diagram

User Ride History
Driver/Rider

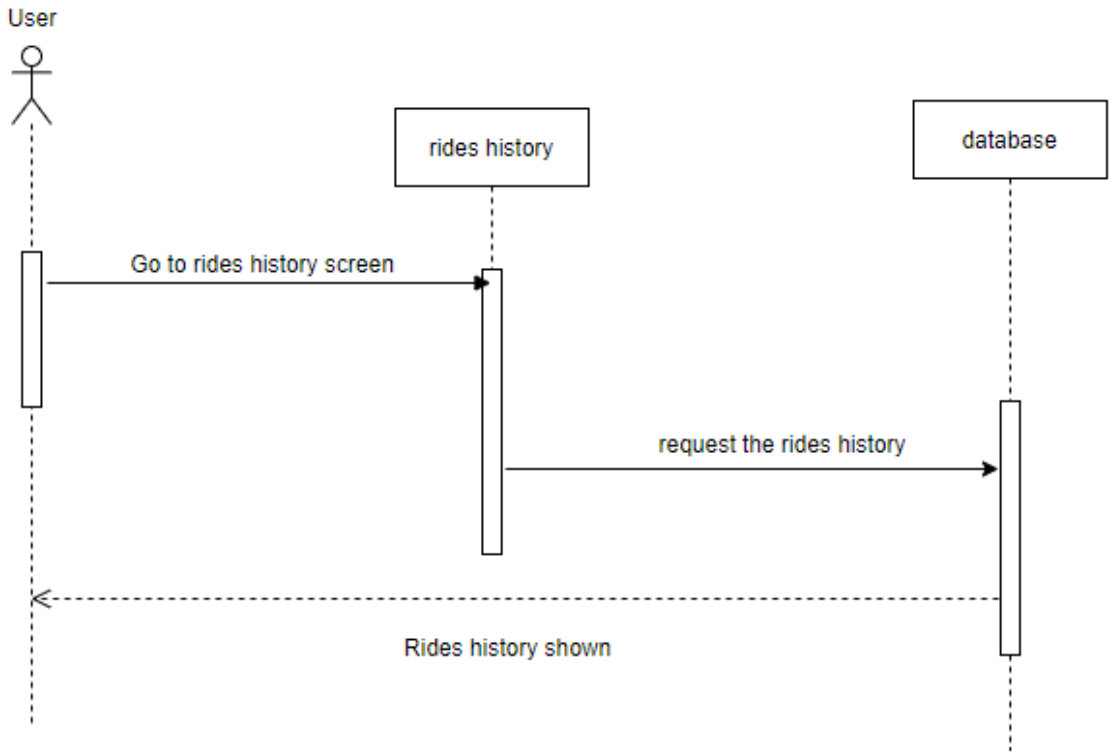


Figure 49 User Ride History Sequence Diagram

User Profile Settings
Driver/Rider

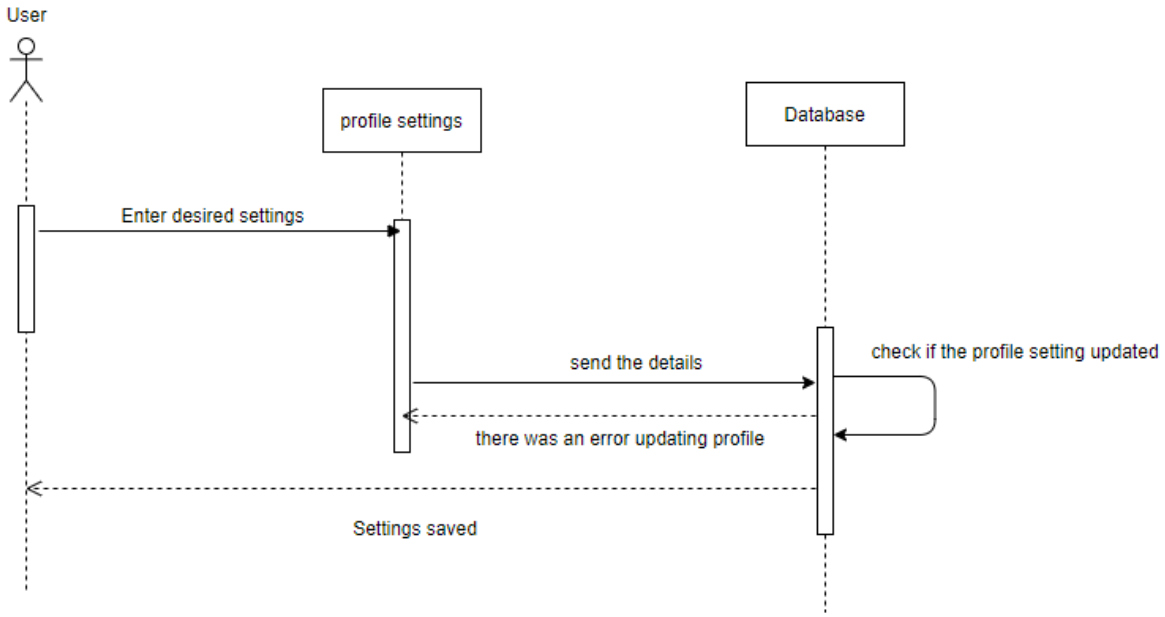


Figure 50 User Profile Setting Sequence Diagram

Driver End Ride

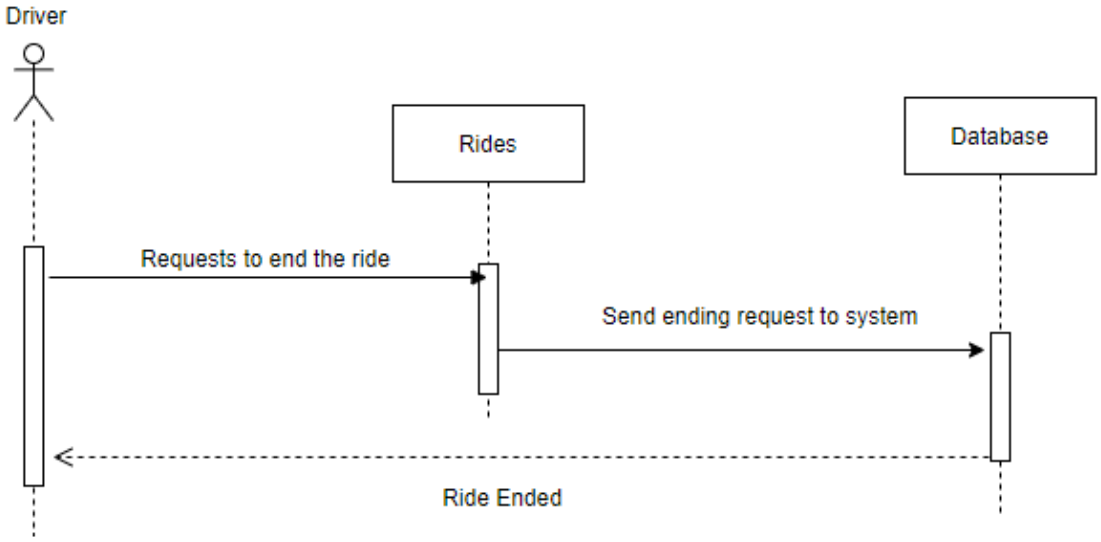


Figure 51 Driver End Ride Sequence Diagram

4.7 Collaboration Diagram

Rider

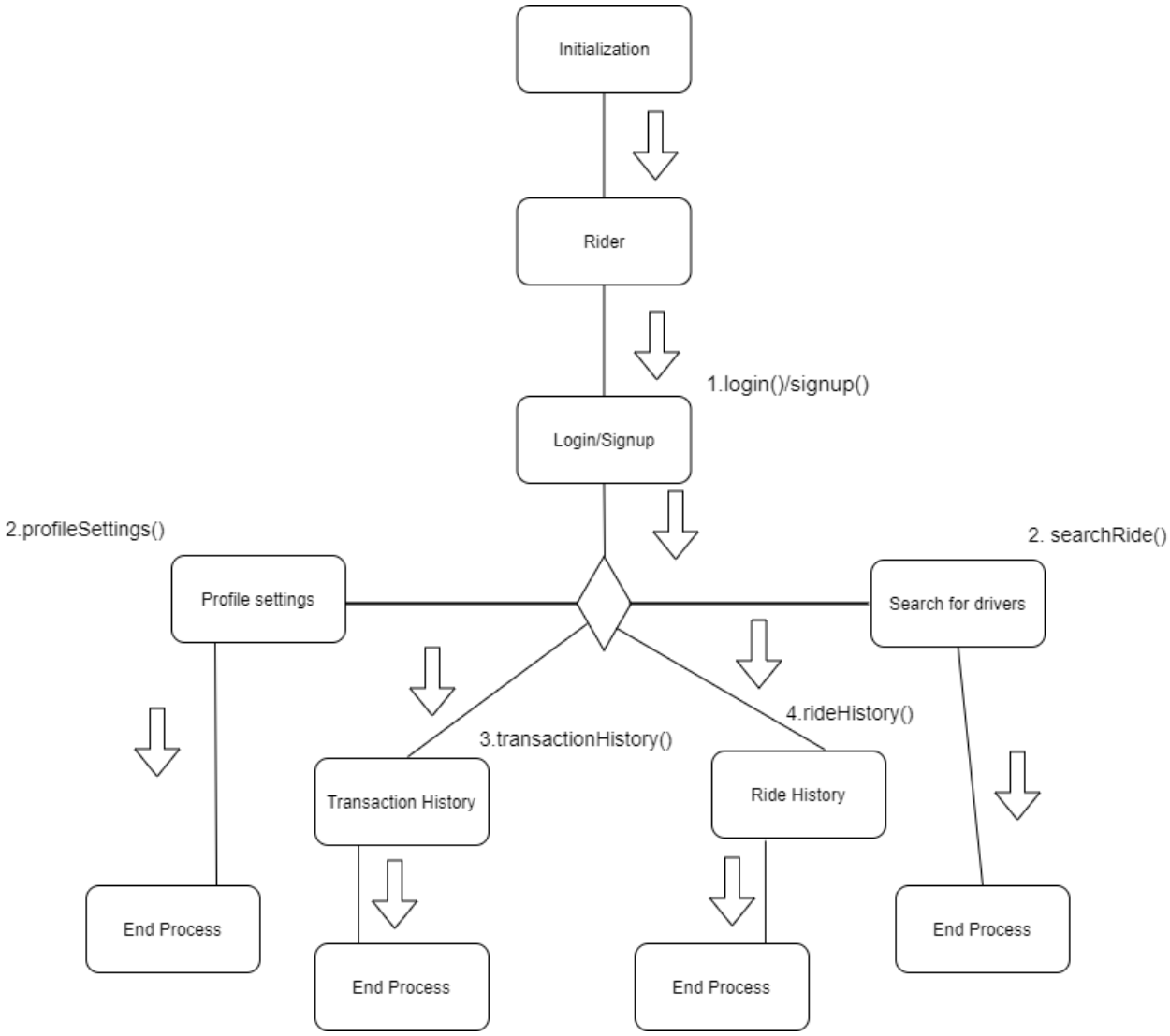


Figure 52 Collaboration Diagram Rider

Driver

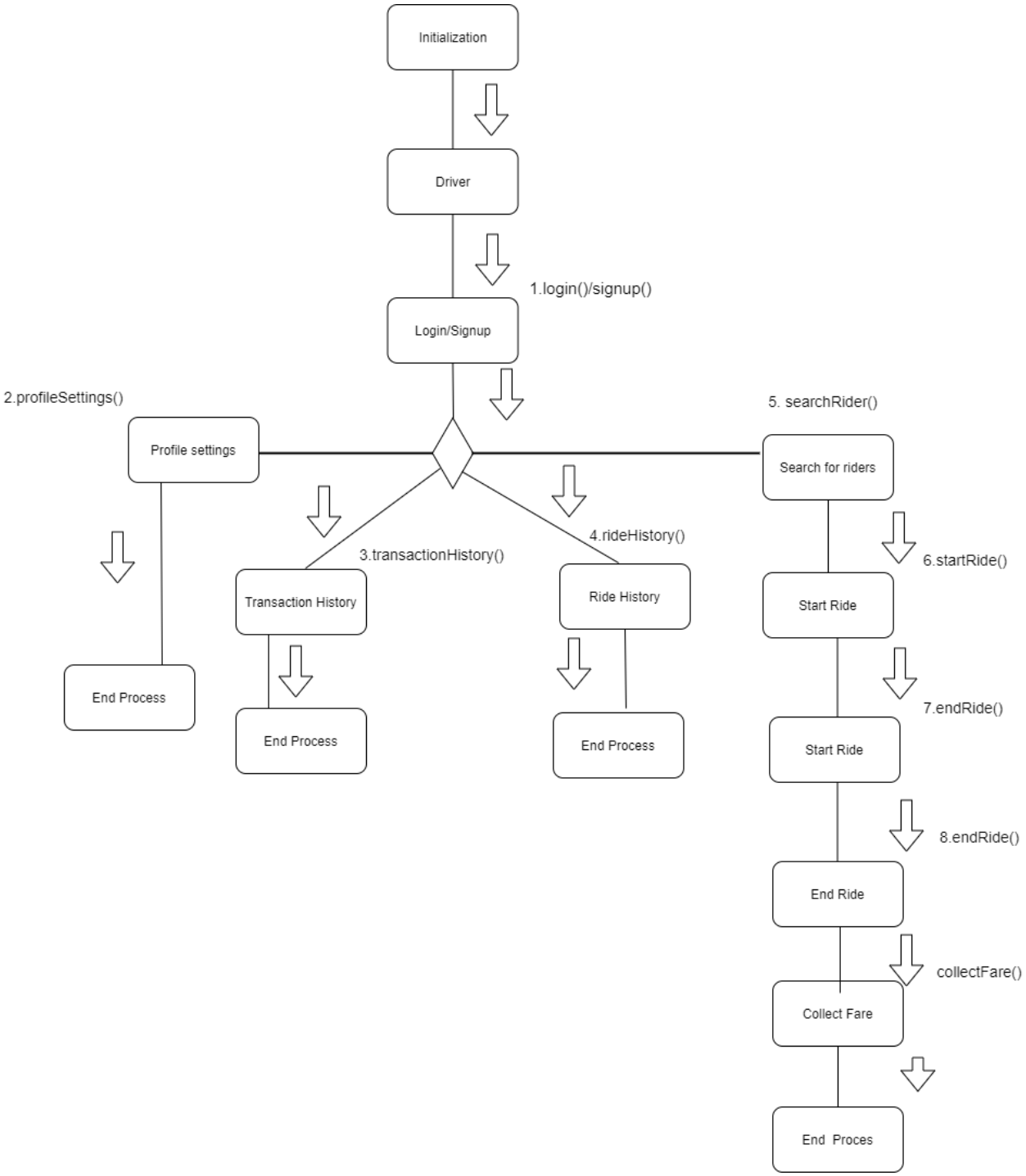


Figure 53 Collaboration Diagram driver

Admin

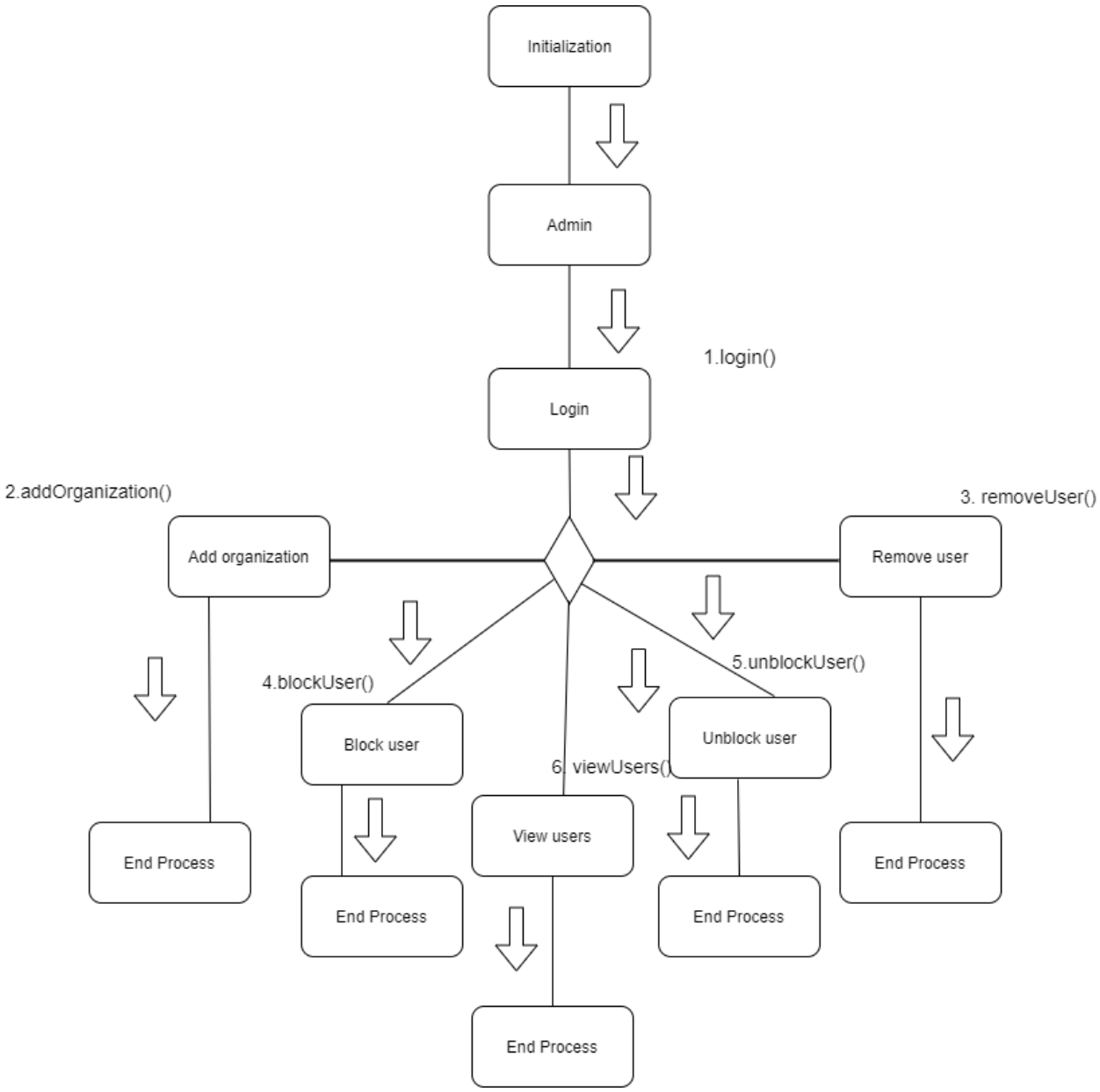


Figure 54 Collaboration Diagram Admin

4.8 State Transition Diagram
User (Driver and Rider)

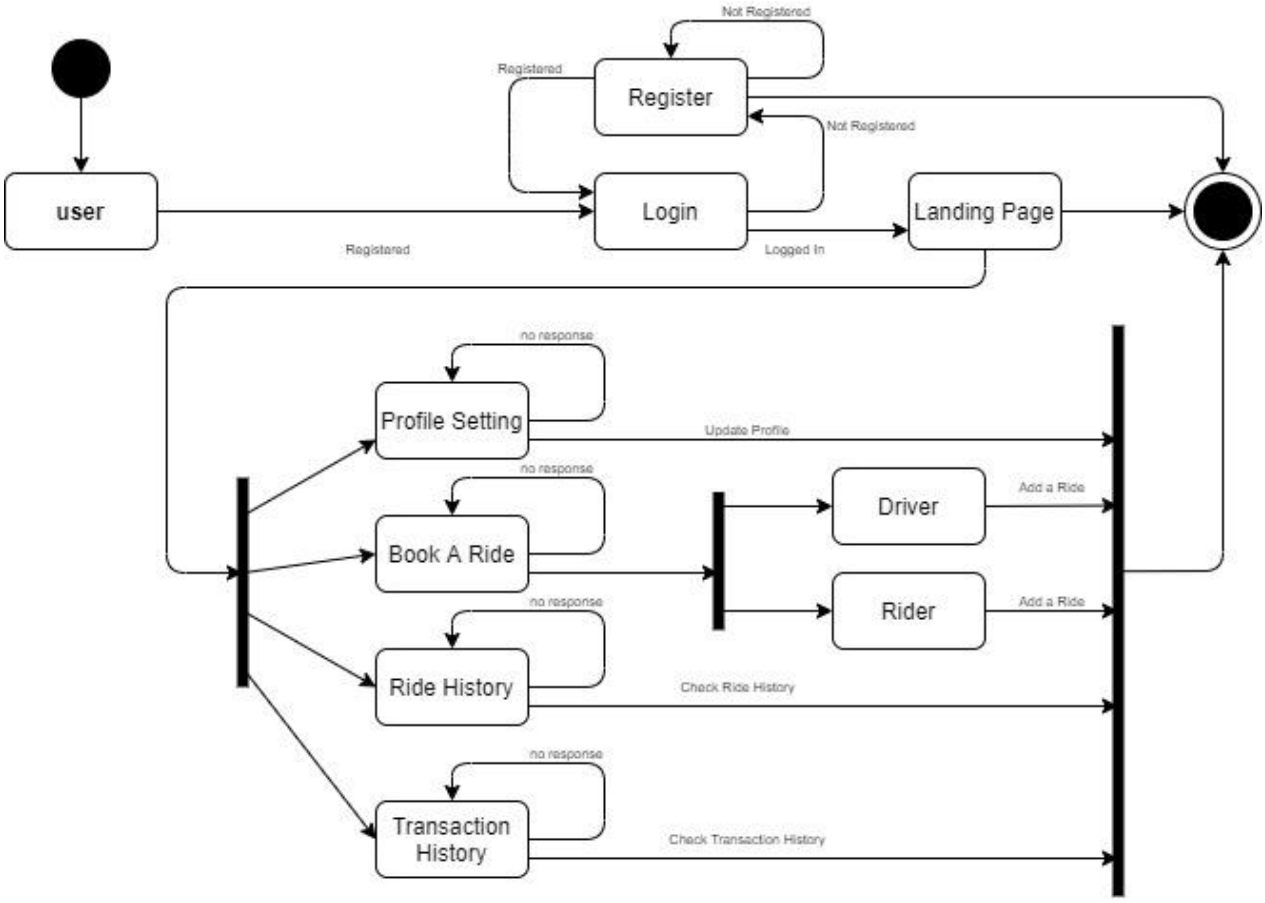


Figure 55 State Transition Diagram User

Admin

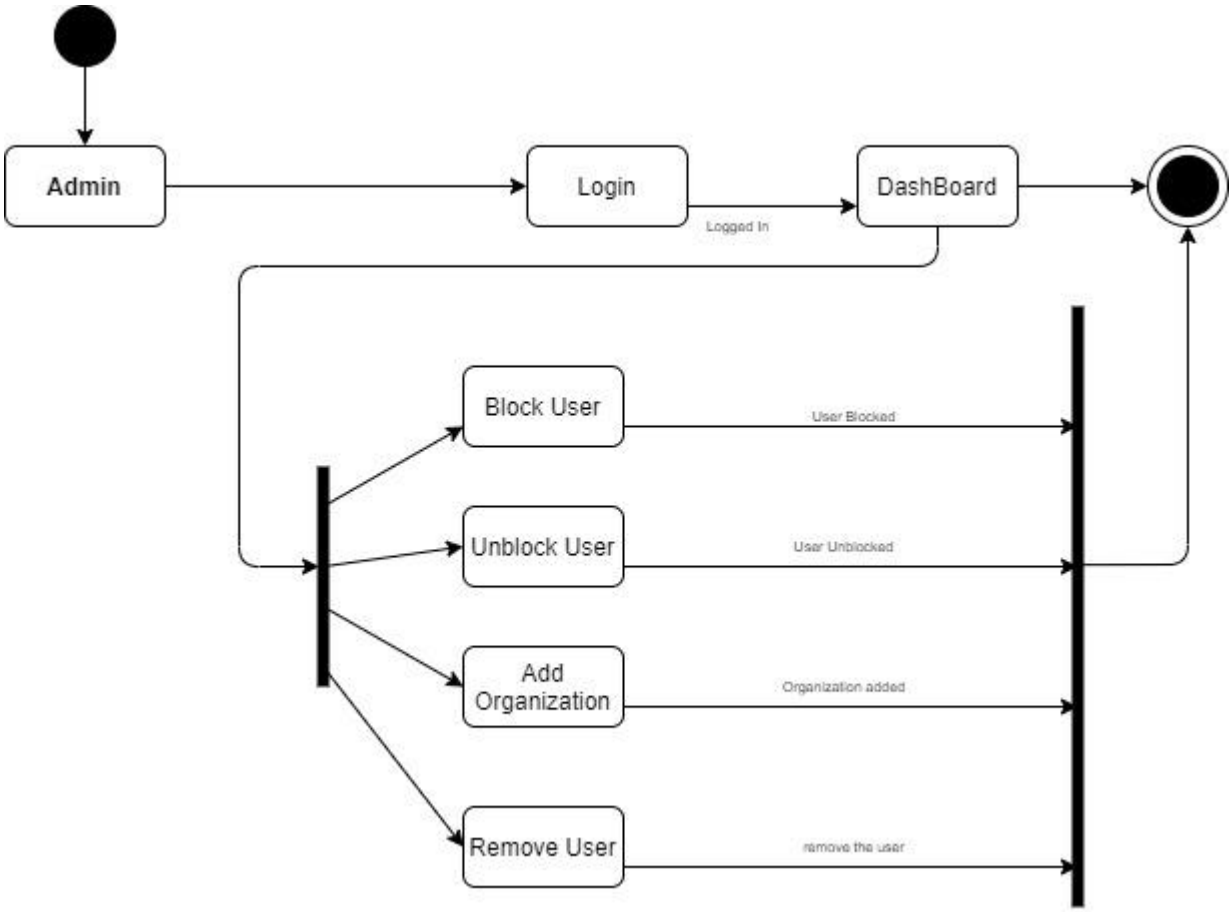
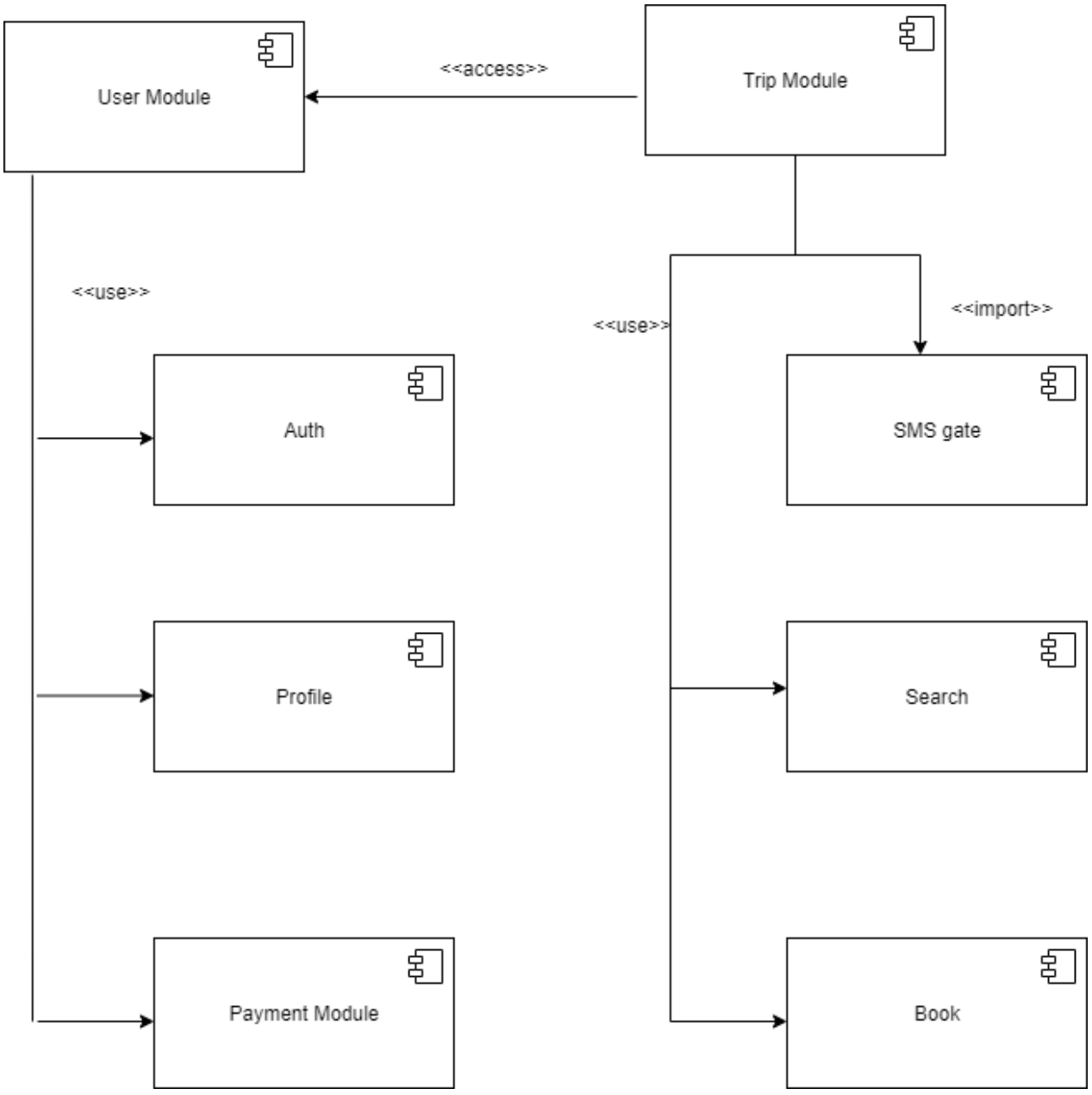


Figure 56 State Transition Diagram Admin

4.9 Component Diagram



. *Figure 57 Component Diagram*

4.10 Deployment Diagram

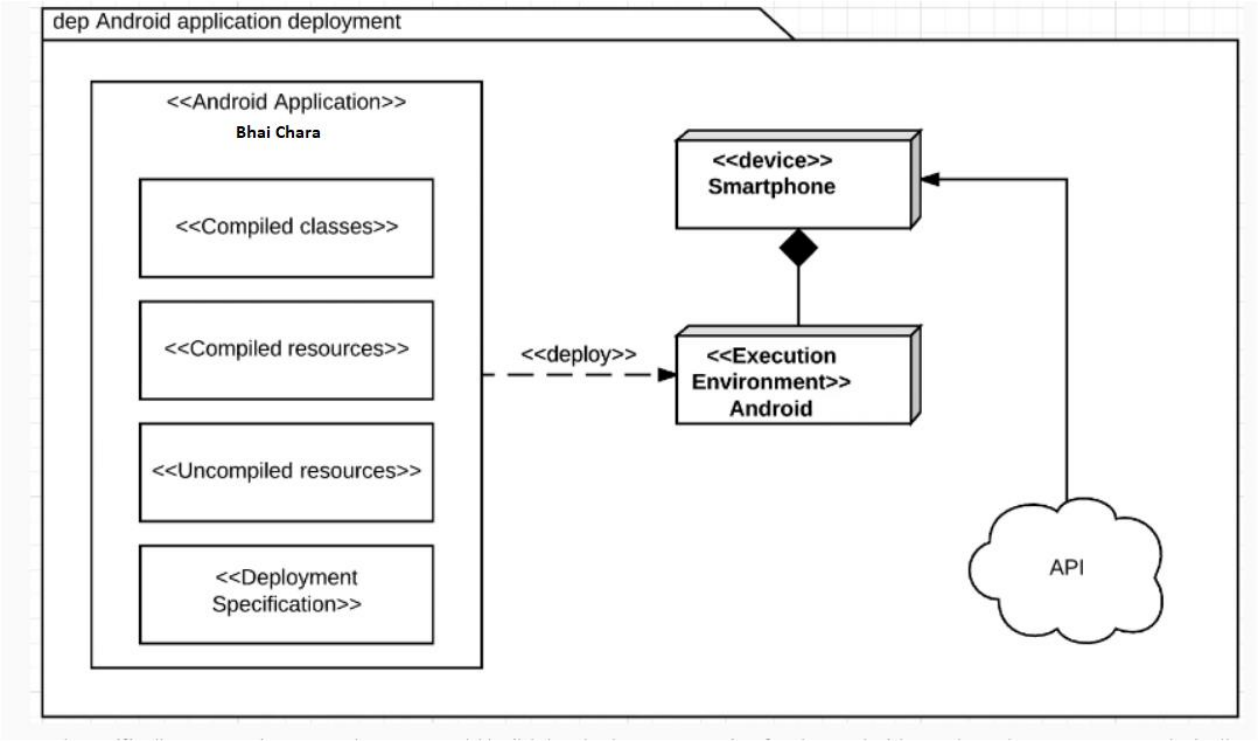


Figure 58 Deployment Diagram

Chapter 5:
Testing

5.1 Test case specifications

Test case for sign in

Positive Test Case	
ID	TC_LOGIN_SUCCESS
Priority	High
Description	To verify user authentication to system.
Reference	Functional Requirement reference
Users	Administrator/rider/driver
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Enter login id C Enter Password. D Press Login.
Input	Login id and password
Expected result	Successfully enters the system and main home page opens.
Status	Tested, passed.

Table 42 Positive Test case sign in

Negative Test Case	
ID	TC_LOGIN_FAILURE
Priority	High
Description	To verify user authentication to system.
Reference	Functional Requirement reference
Users	Admin/rider/driver
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application B Enter login id. C Enter Password. D Press Login.
Input	Incorrect Login id or password or deactivated credentials.
Expected result	Does not allows access to system features and notifies the error.
Status	Tested, passed.

Table 43 Negative Test case sign in

Test case for sign up

Positive Test Case	
ID	TC_SIGNUP_SUCCESS
Priority	High
Description	Test case to create account.
Reference	Functional Requirement reference
Users	Rider/driver
Pre-requisites	A System is online. B User has internet access
Steps	A Open the android application. B Provide the required information for signup C Press signup.
Input	The required information e.g. name , password, phone number and group code
Expected result	Account created successfully
Status	Tested, passed.

Table 44 Positive Test case sign up

Negative Test Case	
ID	TC_SIGNUP_FAILURE
Priority	High
Description	To sign up into the system
Reference	Functional Requirement reference
Users	Rider/driver
Pre-requisites	A System is online. B User must have internet connection
Steps	A Open the android application B Enter the required information for sign up C Press the sign up button.
Input	Invalid or incomplete information
Expected result	Does not allows the user to create the account and shows the error message
Status	Tested, passed.

Table 45 Negative Test case sign up

Test case to search a ride

Positive Test Case	
ID	TC_SEARCHRIDE_RIDER_SUCCESS
Priority	High
Description	To search for a ride
Reference	Functional Requirement reference
Users	Rider
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Provides source, destination and the number of seats required D Press Search ride.
Input	Provides source, destination and the number of seats required
Expected result	List of drivers matching the search is displayed on the screen
Status	Tested, passed.

Table 46 Positive Test case search a ride

Negative Test Case	
ID	TC_SEARCHRIDE_RIDER_FAILURE
Priority	High
Description	To search for a ride
Reference	Functional Requirement reference
Users	rider
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Provides source, destination and the number of seats required D Press Search ride.
Input	Provides the wrong information
Expected result	App does not allows to search and an error message is shown on the screen
Status	Tested, passed.

Table 47 Negative Test case search a ride

Test case for profile settings

Positive Test Case	
ID	TC_PROFILESETTINGS_SUCCESS
Priority	High
Description	To go into the profile settings
Reference	Functional Requirement reference
Users	Rider/driver
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C User perform the required updates D Clicks the update button
Input	User perform the required updates
Expected result	System should update the record according the specified details
Status	Tested, passed.

Table 48 Positive Test case profile settings

Negative Test Case	
ID	TC_PROFILESETTINGS_FAILURE
Priority	High
Description	To go into the profile settings
Reference	Functional Requirement reference
Users	Rider/driver
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C User perform the required updates D Clicks the update button
Input	User does not performs the updates correctly
Expected result	System should not update the record according the specified details and show an error message
Status	Tested, passed.

Table 49 Negative Test case profile settings

Test case transaction history

Positive Test Case	
ID	TC_TRANSACTIONHISTORY_SUCCESS
Priority	High
Description	To check the transaction history
Reference	Functional Requirement reference
Users	Rider/driver
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Clicks on the transaction History button
Input	Clicks on the transaction History button
Expected result	System should show the transaction history to the user
Status	Tested, passed.

Table 50 Positive Test case transaction history

Negative Test Case	
ID	TC_TRANSACTIONHISTORY_SUCCESS
Priority	High
Description	To check the transaction history
Reference	Functional Requirement reference
Users	Rider/driver
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Clicks on the transaction History button
Input	Does not Clicks on the transaction History button
Expected result	System should not take the user to the transaction history screen
Status	Tested, passed.

Table 51 Negative Test case transaction history

Test case rides history

Positive Test Case	
ID	TC_RIDEHISTORY_SUCCESS
Priority	High
Description	To check the ride history
Reference	Functional Requirement reference
Users	Rider/driver
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Clicks on the ride History button
Input	Clicks on the ride History button
Expected result	System should show the transaction history to the user
Status	Tested, passed.

Table 52 Positive Test case rides history

Positive Test Case	
ID	TC_RIDEHISTORY_FAILURE
Priority	High
Description	To check the ride history
Reference	Functional Requirement reference
Users	Rider/driver
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Clicks on the ride History button
Input	User performs an invalid operation
Expected result	System should not show the transaction history to the user
Status	Tested, passed.

Table 53 Negative Test case rides history

Test case switch mode

Positive Test Case	
ID	TC_SWITCHMODE_SUCCESS
Priority	High
Description	To switch between rider and driver modes
Reference	Functional Requirement reference
Users	Rider/driver
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Clicks on the switch mode button
Input	Clicks on the switch mode button
Expected result	System should switch between the modes
Status	Tested, passed.

Table 54 Positive Test case switch mode

Negative Test Case	
ID	TC_SWITCHMODE_FAILURE
Priority	High
Description	To switch between rider and driver modes
Reference	Functional Requirement reference
Users	Rider/driver
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Clicks on the switch mode button
Input	Performs an invalid button
Expected result	System should not allow to switch the modes
Status	Tested, passed.

Table 55 Negative Test case switch mode

Test case start ride

Positive Test Case	
ID	TC_STARTRIDE_SUCCESS
Priority	High
Description	To start the ride
Reference	Functional Requirement reference
Users	Driver
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Driver after reaching at the agreed pick up point, picks up the customer and presses the start ride button
Input	Driver clicks the start ride button
Expected result	System should start the ride
Status	Tested, passed.

Table 56 Positive Test start ride

Negative Test Case	
ID	TC_STARTRIDE_FAILURE
Priority	High
Description	To start the ride
Reference	Functional Requirement reference
Users	Driver
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Driver after reaching at the agreed pick up point, picks up the customer and presses the start ride button
Input	Driver doesn't clicks the start ride button or clicks the start ride button before reaching the pickup point
Expected result	System should not allow to start the ride
Status	Tested, passed.

Table 57 Negative Test start ride

Test case end ride

Positive Test Case	
ID	TC_ENDRIDE_SUCCESS
Priority	High
Description	To end the ride
Reference	Functional Requirement reference
Users	Driver
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access. D Ride should be started
Steps	A Open the android application. B Login to the system C Driver after reaching at the agreed destination drops off the customer and presses the end ride button
Input	Driver clicks the end ride button
Expected result	System should end the ride
Status	Tested, passed.

Table 58 Positive Test case end ride

Negative Test Case	
ID	TC_STARTRIDE_SUCCESS
Priority	High
Description	To end the ride
Reference	Functional Requirement reference
Users	Driver
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access. D Ride should be started
Steps	A Open the android application. B Login to the system C Driver after reaching at the agreed destination drops off the customer and presses the end ride button
Input	Driver doesn't clicks the end ride button or performs any other invalid operation
Expected result	System should not end the ride
Status	Tested, passed.

Table 59 Negative Test case end ride

Test case collect fare

Positive Test Case	
ID	TC_COLLECTFARE_SUCCESS
Priority	High
Description	To collect fare from the customer
Reference	Functional Requirement reference
Users	Driver
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access. D Ride should be ended
Steps	A Open the android application. B Login to the system C Driver after reaching at the agreed destination drops off the customer and clicks the collect cash button
Input	Driver clicks the collect cash button
Expected result	System displays the fare based on ride done. Keeps the record of ride and fare.
Status	Tested, passed.

Table 60 Positive Test case collect fare

Negative Test Case	
ID	TC_COLLECTFARE_FAILURE
Priority	High
Description	To collect fare from the customer
Reference	Functional Requirement reference
Users	Driver
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access. D Ride should be ended
Steps	A Open the android application. B Login to the system C Driver after reaching at the agreed destination drops off the customer and clicks the collect cash button
Input	Driver does not clicks the collect cash button
Expected result	System does not displays the fare based on ride done
Status	Tested, passed.

Table 61 Negative Test case collect fare

Test case to add an organization

Positive Test Case	
ID	TC_ADDORGANIZATION_SUCCESS
Priority	High
Description	To add an organization to the system
Reference	Functional Requirement reference
Users	Administrator
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Adds the required information and assigns the company a unique group code
Input	Adds the required information
Expected result	Organization should be added and should be given a unique code
Status	Tested, passed.

Table 62 Positive Test add an organization

Negative Test Case	
ID	TC_ADDORGANIZATION_SUCCESS
Priority	High
Description	To add an organization to the system
Reference	Functional Requirement reference
Users	Administrator
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Adds the required information and assigns the company a unique group code
Input	Does not adds the required information or adds the invalid information
Expected result	Organization should not be added
Status	Tested, passed.

Table 63 Negative Test add an organization

Test case to block a user

Positive Test Case	
ID	TC_BLOCK_USER_SUCCESS
Priority	High
Description	To block the user from the system
Reference	Functional Requirement reference
Users	Administrator
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Views the user and presses the block user button
Input	Finds the user and clicks on the block user button
Expected result	User should be blocked
Status	Tested, passed.

Table 64 Positive Test case block a user

Negative Test Case	
ID	TC_BLOCK_USER_FAILURE
Priority	High
Description	To block the user from the system
Reference	Functional Requirement reference
Users	Administrator
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Views the user and presses the block user button
Input	Does not finds the user or performs an invalid operation
Expected result	User should not be blocked
Status	Tested, passed.

Table 65 Negative Test case block a user

Test case to unblock the user

Positive Test Case	
ID	TC_UNBLOCK_USER_SUCCESS
Priority	High
Description	To unblock the user from the system
Reference	Functional Requirement reference
Users	Administrator
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Views the user and presses the unblock button
Input	Finds the user and clicks on the unblock button
Expected result	User should be unblocked
Status	Tested, passed.

Table 66 Positive Test case unblock the user

Negative Test Case	
ID	TC_UNBLOCK_USER_SUCCESS
Priority	High
Description	To unblock the user from the system
Reference	Functional Requirement reference
Users	Administrator
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Views the user and presses the unblock button
Input	Does not Finds the user or performs any other invalid operation
Expected result	User should not be unblocked
Status	Tested, passed.

Table 67 Negative Test case unblock the user

Test case view users

Positive Test Case	
ID	TC_UNBLOCK_USER_SUCCESS
Priority	High
Description	To unblock the user from the system
Reference	Functional Requirement reference
Users	Administrator
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Views the user and presses the unblock button
Input	Finds the user and clicks on the unblock button
Expected result	User should be unblocked
Status	Tested, passed.

Table 68 Positive Test case view users

Negative Test Case	
ID	TC_UNBLOCK_USER_SUCCESS
Priority	High
Description	To unblock the user from the system
Reference	Functional Requirement reference
Users	Administrator
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Views the user and presses the unblock button
Input	Finds the user and does the wrong operation
Expected result	User should be not be unblocked
Status	Tested, passed.

Table 69 Negative Test case view users

Test case view users

Positive Test Case	
ID	TC_VIEW_USERS_SUCCESS
Priority	High
Description	To view all the users
Reference	Functional Requirement reference
Users	Administrator
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Clicks on the view users button
Input	Clicks on the view users button
Expected result	List of the users should be sown on the main screen
Status	Tested, passed.

Table 70 Positive Test case view users

Negative Test Case	
ID	TC_VIEW_USERS_SUCCESS
Priority	High
Description	To view all the users
Reference	Functional Requirement reference
Users	Administrator
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Clicks on the view users button
Input	Does not Clicks on the view users button or performs another invalid operation
Expected result	List of the users should not be sown on the main screen
Status	Tested, passed.

Table 71 Negative Test case view users

Test case change roles

Positive Test Case	
ID	TC_CHANGE_ROLES_SUCCESS
Priority	High
Description	To change the roles between riders and drivers
Reference	Functional Requirement reference
Users	Administrator
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Performs the required functionality and Clicks on the change role button
Input	Clicks on the change role button
Expected result	The role of the user should be changed from driver to rider or vice versa
Status	Tested, passed.

Table 72 Positive Test case change roles

Negative Test Case	
ID	TC_CHANGE_ROLES_SUCCESS
Priority	High
Description	To change the roles between riders and drivers
Reference	Functional Requirement reference
Users	Administrator
Pre-requisites	A System is online. B User must have active login credentials provided by system administrator. C User has internet access.
Steps	A Open the android application. B Login to the system C Performs the required functionality and Clicks on the change role button
Input	Performs incorrect action
Expected result	The role of the user should not be changed from driver to rider or vice versa
Status	Tested, passed.

Table 73 Negative Test case change roles

5.2 Black Box testing

Black box testing also known as Behavioral Testing, is a software testing method in which the internal structure/ design/ implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional.

This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see. This method attempts to find errors in the following categories:

- Incorrect or missing functions
- Interface errors
- Errors in data structures or external database access
- Behavior or performance errors
- Initialization and termination errors

Black box testing contains 3 types of testing techniques

- **Equivalence Partitions (EP)**
- **Boundary Value Analysis (BVA)**
- **Decision Table Testing**

Signup Authentication

1.1 Equivalence Partitions

Variables	Valid Partition	Invalid Partition
Phone number	Numeric characters	Alphabets, symbols and Empty Field.
Password	Length should be greater than 5 characters. May contain symbols, alphabets [a-z A-Z] and digits [0-9].	Length less than 5 characters. Empty field.

Table 74: EP for Login Authentication

Boundary value analysis

MINIMUM VALUES			MAXIMUM VALUES		
Invalid Partition	Valid Partition	Valid Partition (min + 1)	Invalid Partition	Valid partition	Valid Partition (max + 1)
Less than 4 letters.	5 letters are valid input.	Greater than 6 letters.	Less than 99 letters.	100 letters.	Greater than 100 letters.

Table 75: BVA for Username

The given table is based on the criterion of Password:

MINIMUM VALUES			MAXIMUM VALUES		
Invalid Partition (min - 1)	Valid Partition	Valid Partition (min + 1)	Invalid Partition (max - 1)	Valid partition	Valid Partition (max + 1)
Less than 4 letters.	5 letters are valid input.	Greater than 6 letters.	Less than 24 letters.	25 letters.	Greater than 26 letters.

Table 76: BVA for Password

Decision Table

Condition 1: User should enter valid name.
Condition 2: User should set a valid password.
Result 1: Sign up successful
Result 2: Please enter invalid password or name.

CONDITIONS	VALUES	COMBINATIONS			
Condition 1	T, F	T	T	F	F
Condition 2	T, F	T	F	T	F
RESULTS					
Result 1		T	F	F	F
Result 2		F	T	T	T

Table 77: Decision Table for Login

Login Authentication

1.1 Equivalence Partitions

Variables	Valid Partition	Invalid Partition
Username	Only username “admin”. Case in-sensitive. Compulsory field.	Alphabets, digits and symbols other than “admin”. Empty Field.
Password	Length should be greater than 5 characters. May contain symbols, alphabets [a-z A-Z] and digits [0-9].	Length less than 5 characters. Empty field.

Table 78: EP for Login Authentication

Boundary value analysis

MINIMUM VALUES			MAXIMUM VALUES		
Invalid Partition	Valid Partition	Valid Partition (min + 1)	Invalid Partition	Valid partition	Valid Partition (max + 1)
Less than 4 letters.	5 letters are valid input.	Greater than 6 letters.	Less than 99 letters.	100 letters.	Greater than 100 letters.

Table 79: BVA for Username

The given table is based on the criterion of Password:

MINIMUM VALUES			MAXIMUM VALUES		
Invalid Partition (min - 1)	Valid Partition	Valid Partition (min + 1)	Invalid Partition (max - 1)	Valid partition	Valid Partition (max + 1)
Less than 4 letters.	5 letters are valid input.	Greater than 6 letters.	Less than 24 letters.	25 letters.	Greater than 26 letters.

Table 80: BVA for Password

Decision Table

Condition 1: Admin should enter valid name.
Condition 2: Admin should enter valid password.
Result 1: login successfully.
Result 2: Please enter invalid password or name.

CONDITIONS	VALUES	COMBINATIONS			
Condition 1	T, F	T	T	F	F
Condition 2	T, F	T	F	T	F
RESULTS					
Result 1		T	F	F	F
Result 2		F	T	T	T

Table 81: Decision Table for Login

Logout Authentication

Equivalence Partitions

Variables	Valid Partition	Invalid Partition
Logout	Admin should log-in. System is online. Logout button should work.	Admin is not logged-in. System is offline. Logout button is not working.

Table 82: EP for Logout Authentication

Decision Table

Condition 1: Requirement for the Logout field.
Condition 2: Requirement for the Admin Login.
Result: Admin Logged-out.

Conditions	Values	Test 1	Test 2	Test 3	Test 4
Logout Field is showing	T/F	T	T	F	F
Admin Logged in	T/F	T	F	T	F
Actions/Effects					
Account Logged out		T	F	F	F

Table 83: Decision Table for Log out

Search for riders

Equivalence Partitions

Variables	Valid Partition	Invalid Partition
Destination	1. Alphabets only [a-z A-Z], Numerical. 2. Compulsory	1. Symbols 2. Empty field.
Source	1. Alphabets only [a-z A-Z], Numerical. Compulsory	1. Symbols 2. Empty field.
Car type	2. Alphabets only [a-z A-Z], Numerical. Compulsory	1. Symbols 2. Empty field.
Number of seats	1. Digits only	1 Symbols 2. Empty field. 3. Alphabets

Table 84: EP for Add person information in the database

Boundary value analysis

The given table is based on the criterion of Name:

MINIMUM VALUES			MAXIMUM VALUES		
Invalid Partition	**Valid Partition**	**Valid Partition (min + 1)**	**Invalid Partition**	**Valid partition**	**Valid Partition (max + 1)**
Less than 1 letters.	1 letters are valid input.	Greater than 1 letters.	Less than 99 letters.	100 letters.	Greater than 100 letters.

Table 85: BVA for Name

The given table is based on the criterion of User Place ID

MINIMUM VALUES			MAXIMUM VALUES		
Invalid Partition	**Valid Partition**	**Valid Partition (min + 1)**	**Invalid Partition**	**Valid partition**	**Valid Partition (max + 1)**
0	1	2	9 Digit	10 Digit	11 Digit

Table 86: BVA for Place ID

The given table is based on the criterion of Image

MINIMUM VALUES			MAXIMUM VALUES		
Invalid Partition	**Valid Partition**	**Valid Partition (min + 1)**	**Invalid Partition**	**Valid partition**	**Valid Partition (max + 1)**
Less than 240 Pixels	240 Pixels	Greater than 240 Pixels	Less than 320 Pixels.	320 Pixels.	Greater than 320 Pixels.

Table 87: BVA for Image

Decision Table

Condition 1: User should add a valid source and destination

Condition 2: User should select a valid card type

Condition 3: User should add a valid number of seats.

Result 1: Ride search completed successfully

Result 2: Invalid information entered

Conditions	**Values**	**Combinations**							
Condition 1	T,F	T	T	T	T	F	F	F	F
Condition 2	T,F	T	T	F	F	T	T	F	F
Condition 3	T,F	T	F	T	F	T	F	T	F
Results									
Result 1		T	F	F	F	F	F	F	F
Result 2		F	T	T	T	T	T	T	T

Table 88: Decision Table for adding new Place information

\

Search for drivers

Variables	**Valid Partition**	**Invalid Partition**
Destination	1 Alphabets only [a-z A-Z], Numerical.	1 Symbols 2 Empty field.
Source	1 Alphabets only [a-z A-Z], Numerical. Compulsory	1 Symbols 2 Empty field.

Table 89: EP for Add person information in the database

Boundary value analysis

The given table is based on the criterion of destination:

MINIMUM VALUES			**MAXIMUM VALUES**		
Invalid Partition	**Valid Partition**	**Valid Partition (min + 1)**	**Invalid Partition**	**Valid partition**	**Valid Partition (max + 1)**
Less than 1 letters.	1 letters are valid input.	Greater than 1 letters.	Less than 99 letters.	100 letters.	Greater than 100 letters.

Table 90: BVA for Name

The given table is based on the criterion of source

MINIMUM VALUES			MAXIMUM VALUES		
Invalid Partition	**Valid Partition**	**Valid Partition (min + 1)**	**Invalid Partition**	**Valid partition**	**Valid Partition (max + 1)**
0	1	2	9 Digit	10 Digit	11 Digit

Table 91: BVA for Place ID

Decision Table

Condition 1: User should add a valid source

Condition 2: User should add a valid destination

Result 1: Ride search completed successfully

Result 2: Invalid information entered

CONDITIONS	VALUES	COMBINATIONS			
Condition 1	T, F	T	T	F	F
Condition 2	T, F	T	F	T	F
RESULTS					
Result 1		T	F	F	F
Result 2		F	T	T	T

Table 92: Decision Table for adding new Place information

Add organization

Equivalence Partitions

Variables	Valid Partition	Invalid Partition
Name	1. Alphabets only [a-z A-Z], Numerical.	3. Symbols 4. Empty field.
Group code	1. Numeric characters	5. Symbols 6. Empty field.

Table 93: EP for Delete information in the database

Boundary Value Analysis

The given table is based on the criterion of User ID

MINIMUM VALUES			MAXIMUM VALUES		
Invalid Partition	**Valid Partition**	**Valid Partition (min + 1)**	**Invalid Partition**	**Valid partition**	**Valid Partition (max + 1)**
0	1	2	9 Digit	10 Digit	11 Digit

Table 94: BVA for Deleting the Information

Decision Table

Condition 1: Admin entered a valid name

Condition 2: Admin given a valid group code

Result 1: Organization added to the system successfully

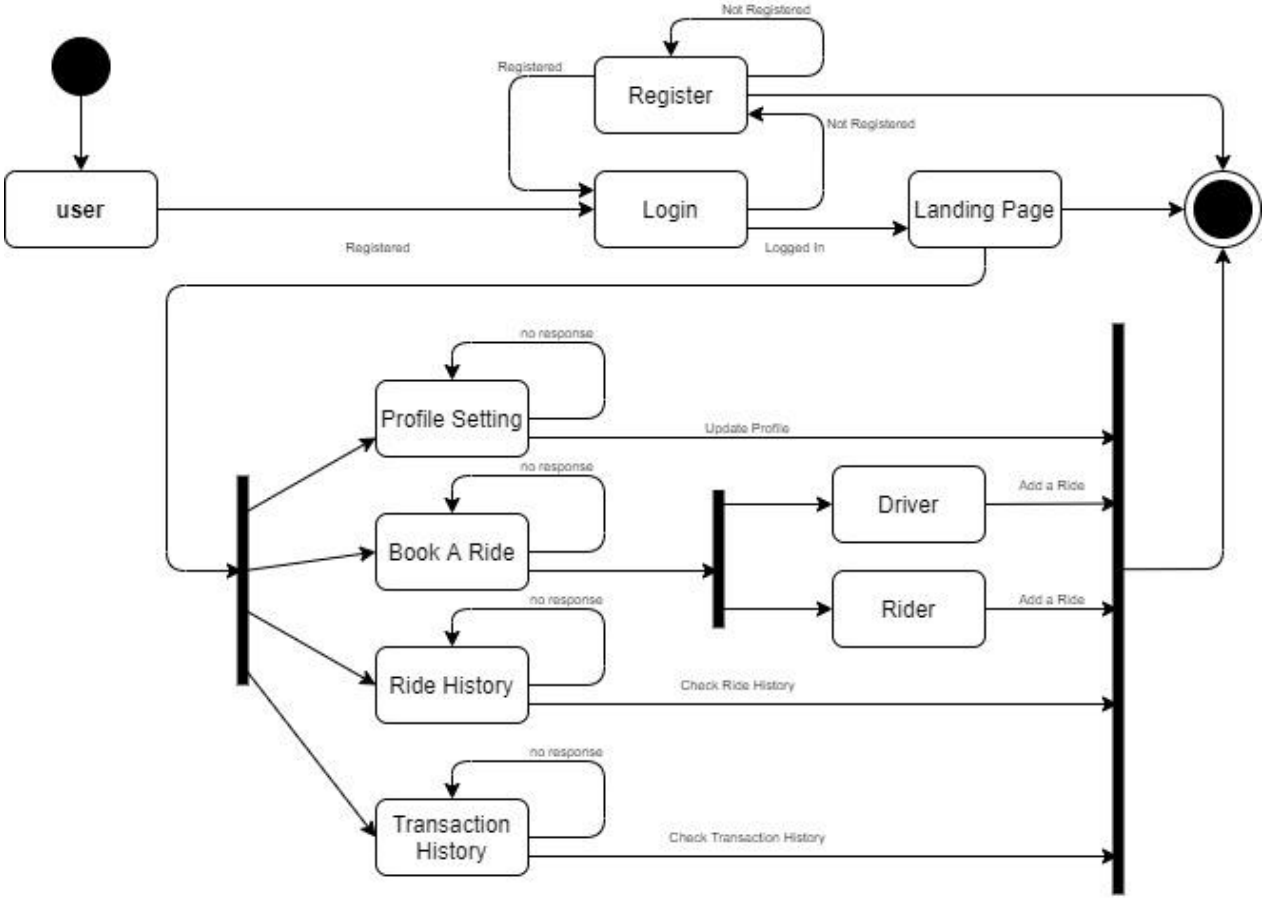
Result 2: organization not added

CONDITIONS	VALUES	COMBINATIONS			
Condition 1	T, F	T	T	F	F
Condition 2	T, F	T	F	T	F
RESULTS					
Result 1		T	F	F	F
Result 2		F	T	T	T

Table 95: Decision Table for Deleting the Information

State Transition Testing

State Transition testing is defined as the testing technique in which changes in input conditions cause's state changes in the Application under Test (AUT).



Login Diagram

Login Table

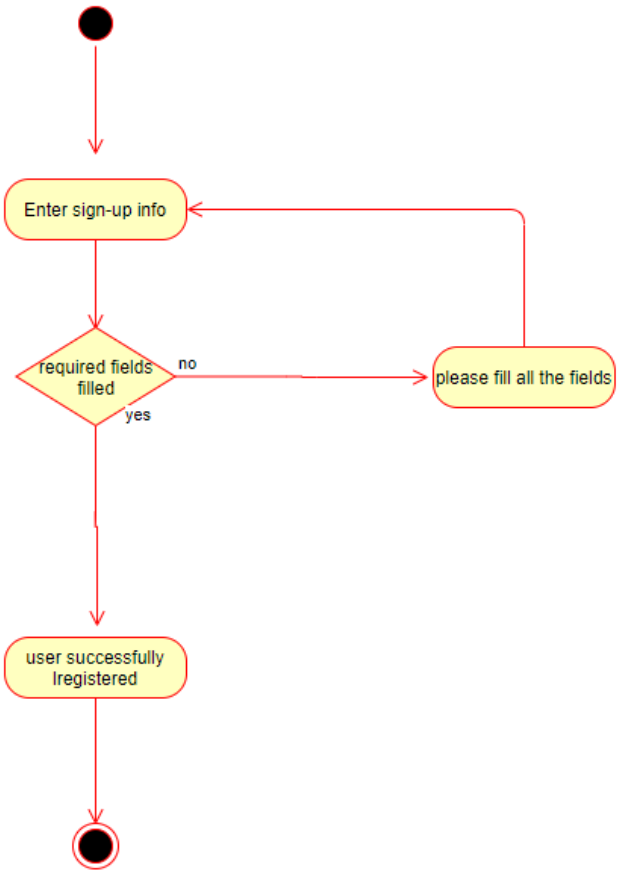
State graph	Correct password	Incorrect password
S1) Start	S6	S2
S2) 1st try	S6	S3
S3) 2nd try	S6	S4
S4) 3rd try	S6	S5
S5) 4th try	S6	S7
S6) access	-	-
S7) close application	-	-

Table 96: State Transition table for Login

White Box Testing

White box testing is a testing technique that examines the program structure and derives test data from the program logic/code. The other names of glass box testing are clear box testing, open box testing, logic driven testing or path driven testing or structural testing.

Sign up



Cyclometric Complexity

M = E – N + 2P

M = 8 – 6 + 2(1)

M= **4**

Statement Coverage

- Input correct username and password
  Covered statement = **5**
  Total statement = **6**
  Statement Coverage = **(5/6)*100 = 83.333%**
- Input incorrect username and password
  Covered statement = **3**
  Total statement = **6**
  Statement Coverage = **(3/6)*100 = 50%**

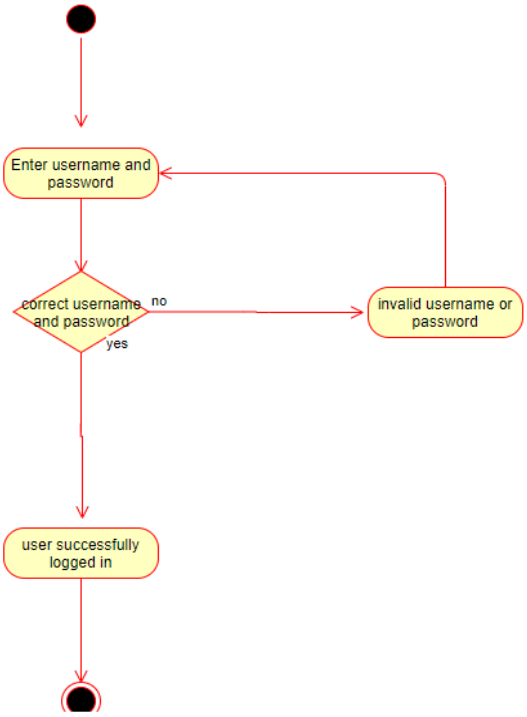
Path Coverage

Path coverage = **2**

Decision Coverage

Decision Coverage = **2**

Login



Cyclometric Complexity

M = E – N + 2P

M = 8 – 6 + 2(1)

M= **4**

Statement Coverage

- Input correct username and password
  Covered statement = **5**
  Total statement = **6**
  Statement Coverage = **(5/6)*100 = 83.333%**
- Input incorrect username and password
  Covered statement = **3**
  Total statement = **6**
  Statement Coverage = **(3/6)*100 = 50%**

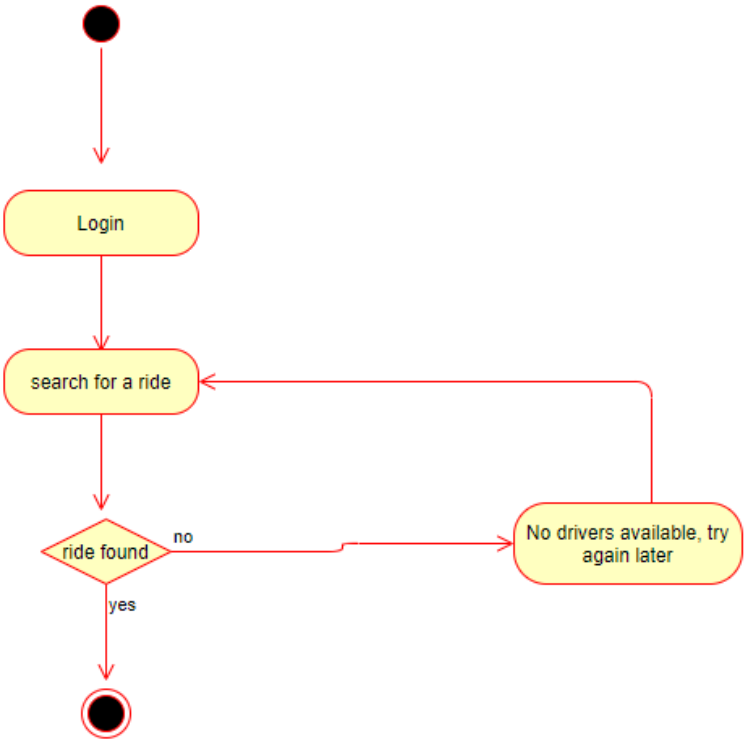
Path Coverage

Path coverage = **2**

Decision Coverage

Decision Coverage = **2**

Search for a ride

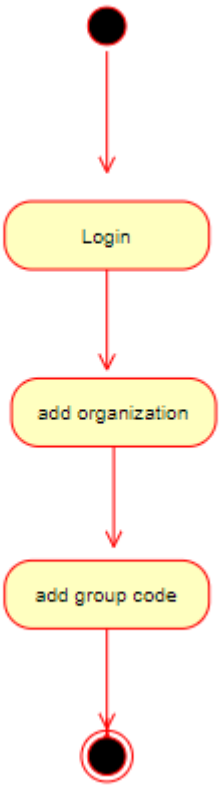


Cyclometric Complexity
M = E – N + 2P
M = 8 – 6 +2(1)
M= **4**
Statement Coverage

- If detail is valid
  Covered statement = **5**
  Total statement = **6**
  Statement Coverage = (**5/6)*100 = 83.33%**
- If detailed is not valid
  Covered statement = **4**
  Total statement = **7**
  Statement Coverage = **(4/7)*100 = 57.142%**

Path Coverage
Path coverage = **2**
Decision Coverage
Decision Coverage = **2**

Add organization



Cyclometric Complexity

M = E – N + 2P

M = 8 – 6 + 2(1)

M= **4**

Statement Coverage

- If detail is valid for update
  Covered statement = **5**
  Total statement = **6**
  Statement Coverage = **(5/6)*100 = 83.33%**
- If detailed is not valid for update
  Covered statement = **4**
  Total statement = **6**
  Statement Coverage = **(4/6)*100 =66.66%**

Path Coverage

Path coverage = **2**

Decision Coverage

Decision Coverage = **2**

Performance testing

Performance testing, a non-functional testing technique that should be carried out after implementation of the product. We shall test the speed, accuracy and effectiveness of functions as well as quantitative test e.g. response time of any function. We have tested the performance of our system by checking the working of every complex operation again and again multiple times. We have followed following steps.

- List of all complex functions especially database related function.
- Assess response time and accuracy of core functions.
- Risk assessment of error or exception code (if any).

Stress Testing

Stress testing a Non-Functional testing technique that should also be carried out after implementation of the product. We shall test the product's robustness, availability and reliability under extreme conditions. The goal of stress testing is to identify application issues that arise or become apparent only under extreme conditions. We have tested our system by simultaneously using our system as different users and performing their associated tasks in a continuous manner. Following steps have been followed:

- We have analyzed system behavior under high traffic.
- We have gathered statistics and reviews about crashing application, time taking operations and shall resolve those issues in later revisions.

System Testing

System Testing (ST) is a black box testing technique that carried out after implementation. It is performed to evaluate the complete system. In System testing, the functionalities of the system are tested from an end-to-end perspective. A team that is independent of the development team in order to measure the quality of the system usually carries out system Testing. It includes both functional and Non-Functional testing.

- List all functions, which fulfill our functional requirements.
- Ensure functionality of these functions.
- Statistical analysis and graphical representation for reference view in future.
- Ensure our product does not conflict with other work products or software (if any).

Regression Testing

Regression testing is a type of software testing that verifies that software previously developed and tested still performs correctly after it was changed or interfaced with other software. Changes may include software enhancements, configuration changes, etc.

Chapter 6
Tools and Techniques

6.1 Languages

6.1.1 Java

Java is a general-purpose programming language that is class-based, object-oriented, and designed to have as few implementation dependencies as possible, first released by Sun Microsystems in 1995. There are lots of applications and websites that will not work unless you have Java installed, and more are created every day. Java is fast, secure, and reliable. From laptops to datacenters, game consoles to scientific supercomputers, cell phones to the Internet, Java is everywhere!

6.1.2 XML

XML stands for **E**xtensible **M**arkup **L**anguage. It is a text-based markup language derived from Standard Generalized Markup Language (SGML).

XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data. XML is not going to replace HTML in the near future, but it introduces new possibilities by adopting many successful features of HTML and it documents in a format that is both human-readable and machine-readable.

6.1.3 HTML

Hypertext Markup Language, a standardized system for tagging text files to achieve font, color, graphic, and hyperlink effects on World Wide Web pages.

HTML is not a programming language, meaning it doesn't have the ability to create dynamic functionality. Instead, it makes it possible to organize and format documents, similarly to Microsoft Word.

When working with HTML, we use simple code structures (tags and attributes) to mark up a website page. For example, we can create a paragraph by placing the enclosed text within a starting <*p*> and closing </*p*> tag.

6.1.4 PHP

PHP (recursive acronym for PHP: Hypertext Preprocessor) is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML. The advantages of PHP are that it is open source, with a huge online community to support it and that it's compatible across multiple platforms. PHP is most often uses by websites with lower traffic demands.

6.1.5 CSS

CSS stands for Cascading Style Sheets. CSS describes how HTML elements are to be displayed on screen or in other media. CSS saves a lot of work. It can control the layout of multiple web pages all at once. External style sheets are stored in CSS files.

6.1.6 SQL

SQL is a domain-specific language used in programming and designed for managing data held in a relational database management system, or for stream processing in a relational data stream management system.

6.2 Applications and Tools

6.2.1 Android Studio

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA . On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps.

It is available for download on Windows, macOS and Linux based operating systems or as a subscription-based service in 2020. It is a replacement for the Eclipse Android Development Tools (E-ADT) as the primary IDE for native Android application development.

6.2.2 Spring Boot

Spring Boot is an open source Java-based framework used to create a micro Service. It is developed by Pivotal Team and is used to build stand-alone and production ready spring applications.

6.2.3 IntelliJ IDEA

IntelliJ IDEA is an **Integrated Development Environment (IDE)** for JVM languages designed to maximize developer productivity. It does the routine and repetitive tasks for you by providing clever code completion, static code analysis, and refactoring, and lets you focus on the bright side of software development, making it not only productive but also an enjoyable experience.

6.2.4 Eclipse JEE

Eclipse is an integrated development environment (IDE) used in computer programming. It contains a base workspace and an extensible plug-in system for customizing the environment. Eclipse is written mostly in Java and its primary use is for developing Java applications.

6.2.5 Visual Studio

Microsoft Visual Studio is an integrated development environment from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps.

6.2.6 Bootstrap

Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains CSS- and JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components.

6.2.7 Postman

Postman is an interactive and automatic **tool** for verifying the APIs of your project. **Postman** is a Google Chrome app for interacting with HTTP APIs. It presents you with a friendly GUI for constructing requests and reading responses. It works on the backend, and makes sure that each API is working as intended.

6.2.8 XAMPP

XAMPP is a free and open-source cross-platform web server solution stack package developed by Apache Friends, consisting mainly of the Apache HTTP Server, MariaDB database, and interpreters for scripts written in the PHP and Perl programming languages.

6.2.9 My SQL

MySQL is an open-source relational database management system. Its name is a combination of "My", the name of co-founder Michael Widenius's daughter, and "SQL", the abbreviation for Structured Query Language.

6.2.10 Github

GitHub, Inc. is a United States-based global company that provides hosting for software development and version control using Git. It has been a subsidiary of Microsoft since 2018. It offers the distributed version control and source code management functionality of Git, plus its own features.

6.2.11 Draw.IO

Draw.io is completely free online diagram editor built around Google Drive(TM) that enables you to create flowcharts, UML, entity relation, network diagrams, mockups and more. Your data is stored only in Google Drive, so no additional third-party to trust with your data.

6.2.12 Visual Paradigm

Visual Paradigm is a UML CASE Tool supporting UML 2, SysML and Business Process Modeling Notation from the Object Management Group. In addition to modeling support, it provides report generation and code engineering capabilities including code generation.

6.2.13 MS Word 2013

Microsoft Word is a graphical word processing program that users can type with. Its purpose is to allow users to type and save documents. Similar to other word processors, it has helpful tools to make documents.

6.3 Libraries and Extensions

6.3.1 Google Maps API

The Google Maps Platform is a set of APIs and SDKs that allows developers to embed Google Maps into mobile apps and web pages, or to retrieve data from Google Maps.

6.3.2 Google Directions API

The **Directions API** is a service that calculates **directions** between locations. You can search for **directions** for several modes of transportation, including transit, driving, walking, or cycling.

6.3.3 ION

It is an API for Android Asynchronous Networking and Image Loading.

6.3.4 Naik

This library provide support for STOMP protocol https://stomp.github.io/ At now library works only as client for backend with support STOMP, such as NodeJS (stompjs or other) or Spring Boot (SockJS).

6.3.5 Spring Web Sockets

WebSockets is a bi-directional, full-duplex, persistent connection between a web browser and a server. Once a **WebSocket** connection is established the connection stays open until the client or server decides to close this connection.

Chapter 7
Summary and Conclusion

7.1 Summary

There are over a million vehicles in Lahore and there had been an increase of 3000 every month. According to Gallop Pakistan, two seats are empty out of every four at average. Considering the environmental concerns, congestion of roads, and rising fuel costs and inflation ride sharing has gained a lot of popularity when it comes to environment-friendly and cheap ways of travelling.

Bhai Chara would connect potential drivers and riders having same route and time by using a simple android application and internet. In addition to that, it provides intercity ride sharing, driver getting a monetary benefit and rider getting an easy and comfortable way of commuting. The competitive edge it has over its competition is that it is not a dedicated cab application i.e. the driver sharing the ride to a particular destination is headed there anyway and picking some riders along is just like a bonus. The scope of usage is very broad, it doesn't matter whether a driver is regular or shares a ride once a year, the principle stays same. It also features a group code, every destination has some group code. User using that group code will be connected with people from that organization only, it also helps with searching.

7.2 Conclusion

Today we are living in a digital world with more and more things becoming available on our phones or gadgets. This digitalization is taking over every sector and industry, transportation is no different. So an effective and simple to use ride sharing application fits right in. Allowing users to share their rides, the main target market is two kinds of people one who couldn't afford their own vehicles and the second who do afford to have personal vehicles but minimizes the use of them citing costs etc. Bhai Chara would connect both these, hence contributing much for betterment of society and the transportation industry, resulting in less number of cars on roads and no flooding of parkings.

Chapter 8
User Manual

8.1 Login Manual

Following is the login screen, from where user logs in to the system.

User must have a valid phone number for this

Steps:

1. Open application
2. Enter Phone Number
3. Click Verify Done

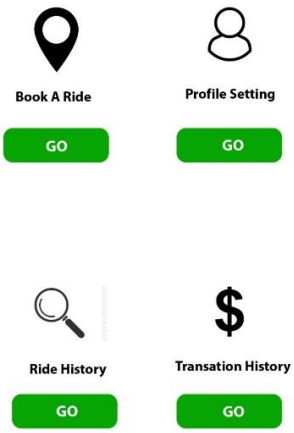
Steps:

1. Open login webpage
2. Enter login details.
3. Click Login.
4. Done.

8.2 Home Screen

Steps:

1. Click on whatever the function(Book a ride, Profile Settings, Transaction History and Ride History) you want to perform
2. Done.

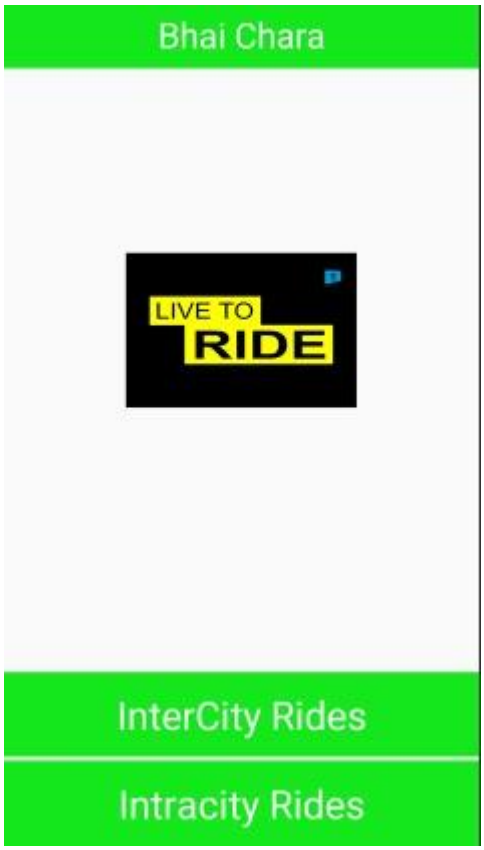


8.3 Driver Inter/Intra City ride

Steps:

1. Click on inter or intra city rides for whichever desired
2. Done

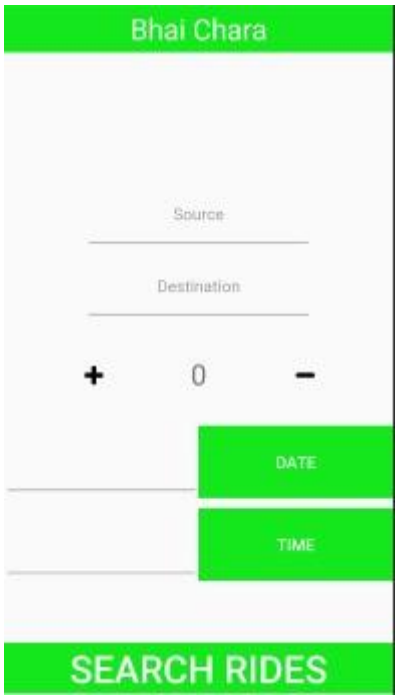
8.4 Rider Inter/Intra City rides:



Steps:

1. Click on inter or intra city rides for whichever desired
2. Done

8.5 Intercity rides



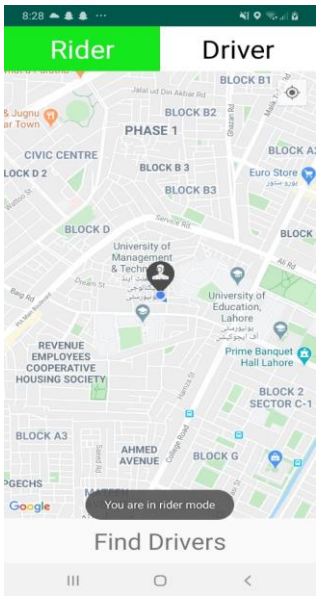
Steps:

1. Enter source
2. Enter destination
3. Click “plus” to add seats, click minus to subtract.
4. Select date and time
5. Press search rides
6. Done

8.6 Chose Roles

Steps:

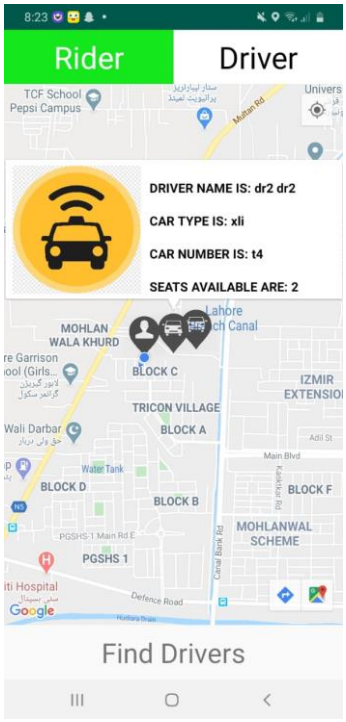
1. Click “rider” to be in rider mode
2. Click “Driver” to be in driver mode
3. Done



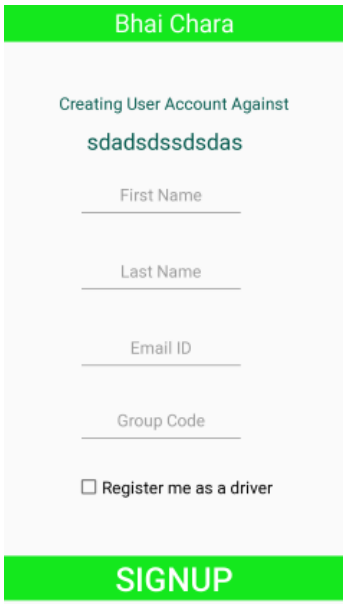
8.7 Request Ride

Steps:

1. Click find drivers
2. Click on the marker to request ride
3. Done



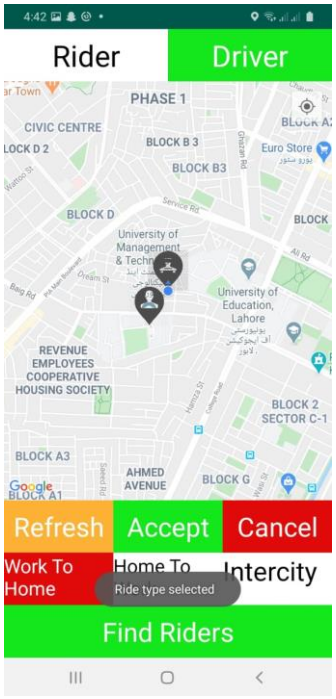
8.8 Sign Up



Steps:

1. Enter first name
2. Enter last name
3. Enter email address
4. Enter group code
5. Check the check box depending on the requirement
6. Done

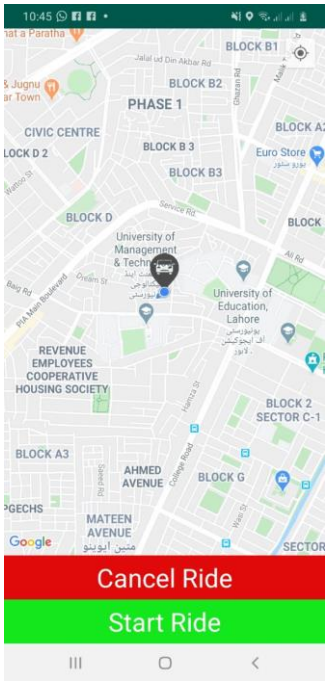
8.9 Driver Search riders



Steps:

1. Click Refresh to update
2. Click accept or cancel to accept or cancel a coming ride
3. Select a category from any three
4. Click find riders to see riders nearby
5. Done

8.10 Start Ride



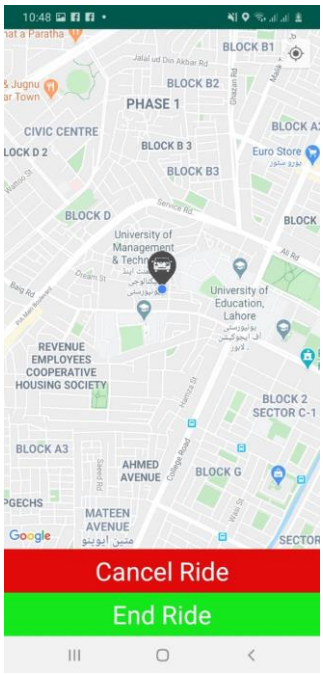
Steps:

1. Click on start ride, to start a particular ride
2. Click cancel to cancel the particular ride
3. Done

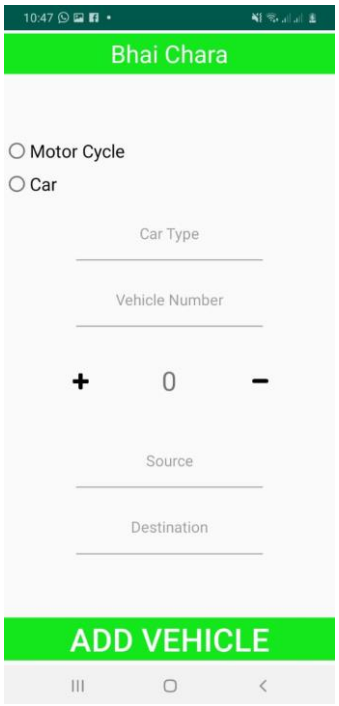
8.11 End Ride

Steps:

4. Click on end ride, to end/finish a particular ride
5. Click cancel to cancel the particular ride
6. Done



8.12 Add vehicle/ Ride



Steps:

1. Check the bike circle for bike, car circle for car
2. Select number of seats to offer
3. Put source
4. Put destination
5. Click on add vehicle
6. Done

Chapter 9
Lessons Learnt and Future Work

9.1 Lessons Learnt

9.1.1 The technical skillset

This was the first time we worked on a project of this magnitude, we had a dedicated team and a project supervisor to report to. It tested our technical skill set to its limits and provided us with an unparalleled learning experience.

9.1.2 Team Work

We all are aware of importance of team work, this provided us the first hand insight into the team working. The four of us spent a year together, working tirelessly, the weekly follow up meetings, new assignments, hitting the deadlines and all. It taught us to know other persons have empathy for them. Appreciate their strengths and help them with their weakness.

9.1.3 The field exposure

We worked in a scrum like manner, weekly meetings with presentations of increments that are developed and the about the ones that would be developed in the coming weeks. Our supervisor Dr. Yasir Niaz Khan was a boss like figure, appreciating us for the hard work, scolding us whenever necessary. Quite often we would stay late at campus during our meetings. So this gave us the first-hand experience of how the field of IT works.

9.2 Difficulties and challenges

9.2.1 Getting location updates with most accuracy and least cost

When working with google maps we needed location updates every 10 seconds, considering its an mobile ride sharing application, this magnitude of requests would drain the battery way too quick. So optimizing it was a challenge.

9.2.2 Reducing the number of requests for our database

Drivers and riders would simultaneously request rides, we couldn’t query that kind of requests to our database considering the resources we had at our disposal. For that we came up with a ride cache, going through it was one tough job.

9.2.3 Route Optimization

We optimized the routes available for our drivers considering the traffic, distance as well as the number of passengers it was quite challenging

9.2.4 GUI simplification

The goal was to keep the design as simple as possible without effecting the functionality.

9.3 Future Enhancements

9.3.1 Using artificial intelligence for route selection

Incorporating AI for optimizing the routes in our application.

9.3.2 Cloud Hosting

Currently the application runs on local server, we would like to host it on some cloud platform so it is available online.

9.3.3 Hybrid Application

Currently it is native android application, in future we would like to go for a hybrid platform.

9.3.4 Security and Payments

Talking about cyber security the application does not have a well-structured security package, we would do some future enhancements for that as well as include multiple payment options.

Chapter 10
Learning Outcomes

Learning Outcomes for Team BhaiChara

This is the first time I have been working on a project like this, this is a remarkable experience so far, full of challenges and learning opportunities. In this report I will mention what I learn during this period, I will try to mention every high and low.
After my idea was accepted, the first thing I studied was the planned technology stack, mentioned below:

- Android Studio
- Spring Boot
- Web Sockets
- RESTful API
- Google Map API
- Retrofit/ Ion/Volley
- SQL

After finalizing the technology stack, I started the research. The first thing I started working on was an algorithm that would match the two strings and calculates the similarity between them. Their intended use was to minimize the requests for google map API. I.e. The plan was to have a local database that will store locations, our algorithm would compare the location's name entered by the user with the names stored. The string having greatest similarity would be set as the location, hence reducing the requests for places API and decreasing the response time. It took me 3 weeks to come up with a working algorithm. I tested it and found out that it doesn't consider the distance among two strings being compared. I tried to remedy it but it was way out of my league, so a decision was made to go through existing algorithms to check whether I can come up with a solution. The algorithms considered were:

- Levenshtein Algorithm
- Longest Common Subsequence
- Knuth–Morris–Pratt algorithm

Levenshtein Algorithm:
Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other.

For example, the Levenshtein distance between "kitten" and "sitting" is 3, since the following three edits change one into the other, and there is no way to do it with fewer than three edits:

1. **k**itten → **s**itten (substitution of "s" for "k")
2. sitt**e**n → sitt**i**n (substitution of "i" for "e")
3. sittin → sittin**g** (insertion of "g" at the end).

Longest Common Subsequence:

The longest common subsequence (LCS) problem is the problem of finding the longest subsequence common to all sequences in a set of sequences (often just two sequences).

For example, consider the sequences (ABCD) and (ACBAD). They have 5 length-2 common subsequences: (AB), (AC), (AD), (BD), and (CD); 2 length-3 common subsequences: (ABD) and (ACD); and no longer common subsequences. So (ABD) and (ACD) are their longest common subsequences.

Wagner–Fischer algorithm

The **Wagner–Fischer algorithm** is a dynamic programming algorithm that computes the edit distance between two strings of characters.

The Levenshtein distance allows deletion, insertion and substitution. The Longest common subsequence (LCS) distance allows only insertion and deletion, not substitution. The Levenshtein distance is a string metric for measuring the difference between two sequences. The Wagner–Fischer algorithm is a dynamic programming algorithm that computes the edit distance between two strings of characters.

Each had their own pros and cons, but there was one problem common to all. Consider strings “Islamabad” and “Isl”. Now we all know that if we are talking about context both are same, but if we are talking about string similarity it will be 33% give or take.

The work on android app was started, android app development is done by using “XML” for front end and “JAVA” or “Kotlin” for back end. I opted a combination of “XML” and “JAVA”, android also supports drag and drop but it is not recommended, limits the functionality. In android development two things have paramount importance the interface design and the target sdk (Software development kit). I started with learning the structure of android and how to get the most out of it. I studied the rules for interface design, primarily **Ben Shneiderman’s eight golden principles** namely,

- Strive for consistency.
- Enable frequent users to use shortcuts.
- Offer informative feedback.
- Design dialog to yield closure.
- Offer simple error handling.
- Permit easy reversal of actions.
- Support internal locus of control.
- Reduce short-term memory load.

The entire interface was designed keeping in mind these principles. The next step was to choose an sdk, an sdk is a set of development tools used to develop applications for **Android** platform. Sdk gets updated every year or two. Device support depends on the target sdk.

Retrofit/Ion/Volley:

Volley is a networking library for **Android** that manages network requests. It bundles the most important features you'll need, such as accessing JSON APIs, loading images and String requests in an easier-to-**use** package. Features include:

- Google open source, focusing on processing small requests for high frequency data
- Internally, HttpURLConnection and HttpClient are still used for network requests, but the response is switched for different Android versions.
- Support cancellation request
- With network request and picture loading function

Retrofit is a REST Client for Java and Android. It makes it relatively easy to retrieve and upload JSON (or other structured data) via a REST based webservice. Features include:

- It is an open source type-safe HTTP client for Android
- The underlying layer is based on OkHttp and uses OkHttp for requests
- Convert the definition of java API to interface form
- Use annotations to describe http requests
- Support for configuring json parser

ION by Koushik Dutta is an Android Asynchronous Networking and Image Loading library. It uses Image View or Bitmaps, JSON (via Gson), Strings, Files and Java types using Gson for asynchronous download. Features include:

- It supports the dual functions of network request and image loading

- Has a chain api style (Fluent API)
- Support automatic cancellation when Activity ends
- Supports SPDY / HTTP2, caching, Gzip compression, HTTP connection reuse, etc.
- Based on AndroidAsync, it implements sockets and follows the http protocol.

Considering all this I opted for ION, instead of retrofit or volley due to the support of both sockets and http protocol.

Stateless vs Stateful:

Network Protocols for web browser and servers are categorized into two types: Stateless Protocol, and Stateful protocol.

Stateless Protocols are the type of network protocols in which Client send request to the server and server response back according to current state. It does not require the server to retain session information or a status about each communicating partner for multiple request. Stateful protocol is the one in which client connects to the server and stays, it is used for real time data transmission, which is very essential for our project i.e. locations of users communicating or online.

Locations Manager Vs Google Services:

Getting the user's location on Android is a little less straightforward. To start there are two totally different ways you can do it. The first is using Android APIs from `android.location. LocationListener`, and the second is using Google Play Services APIs `com.google.android.gms.location.LocationListener`.

The Android's location APIs use three different providers to get location -

- LocationManager.GPS_PROVIDER — This provider determines location using satellites. Depending on conditions, this provider may take a while to return a location fix.
- LocationManager.NETWORK_PROVIDER — This provider determines location based on availability of cell tower and WiFi access points. Results are retrieved by means of a network lookup.
- LocationManager.PASSIVE_PROVIDER — This provider will return locations generated by other providers. You passively receive location

updates when other applications or services request them without actually requesting the locations yourself.

Google's Location Services API is a part of the Google Play Services APK. They're built on top of Android's API. These APIs provide a "Fused Location Provider" instead of the providers mentioned above. This provider automatically chooses what underlying provider to use, based on accuracy, battery usage, etc. It is fast because you get location from a system-wide service that keeps updating it. And you can use more advanced features such as geo fencing.

Initially I was using Locations Manager to get location, but as I studied further I switched to google location services.

Using Shared Preferences:

Shared preferences allow you to store small amounts of primitive data as key/value pairs in a file on the device. To get a handle to a preference file, and to read, write, and manage preference data, use SharedPrefrences class. The Android framework manages the shared preferences file itself. The file is accessible to all the components of your app, but it is not accessible to other apps.

The data you save to shared preferences is different from the data in the saved activity state:

- Data in a saved activity instance state is retained across activity instances in the same user session.
- Shared preferences persist across user sessions. Shared preferences persist even if your app stops and restarts, or if the device reboots.

Use shared preferences only when you need to save a small amount data as simple key/value pairs. To manage larger amounts of persistent app data, use a storage method such as the Room library or an SQL database.

I have studied shared preferences in my android course as well as by using other resources available, what I failed to grasp was, when we have Server Site Database and SQLite for android why we need to work with the shared preferences as it stores rather small amount of data and it is not well organized. Just out of the blue a question came into my mind that how the different apps we use know that the user is already signed in and they don't ask the user to sign in whenever the user uses the app. I searched about it and I found that is the most

used way for shared preferences. Whenever a user signs up, the developer stores that userobj into shared preferences, he can use that data throughout the application. Secondly when an app is started, the first activity that opens is called as splash activity or screen. We usually write a check in our splash screen whether userobj is there in shared preferences or not. If it is available it means that user already exists and if not then it's a new user and we have to divert him to our sign up page.

Consider the following example:

```
SharedPrefrences prefs = this.getSharedPreferences (
    "MySharedPrefs", Context.MODE_PRIVATE);
prefs.put("UserObj", UserObject);
```

Note the two parameters, the first one is the key the other one is the value against the key for putting operation.

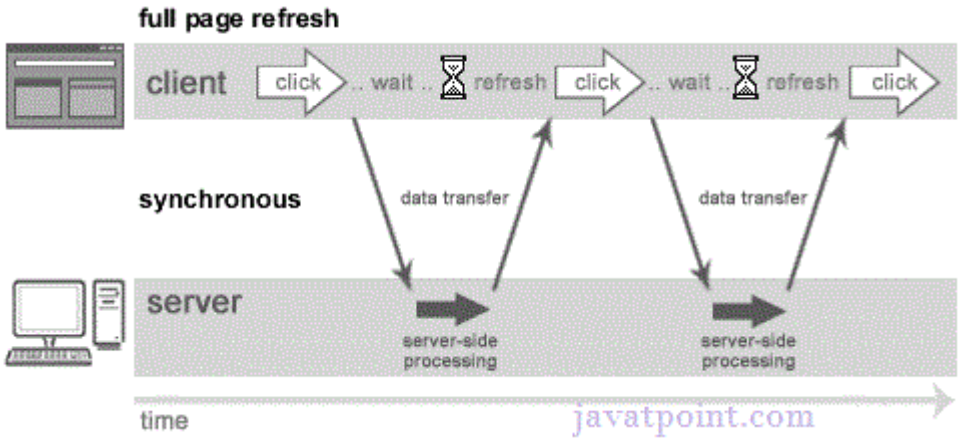
```
During splash screen:
If(getFromSharedPrefrences("UserObj", " ") == null)
{
Move to sign Up activity }
Else{
Move to home activity
}
```

When getting the values, the first one is the key and other one is the default value if userobj is not found.

Understanding Synchronous vs Asynchronous

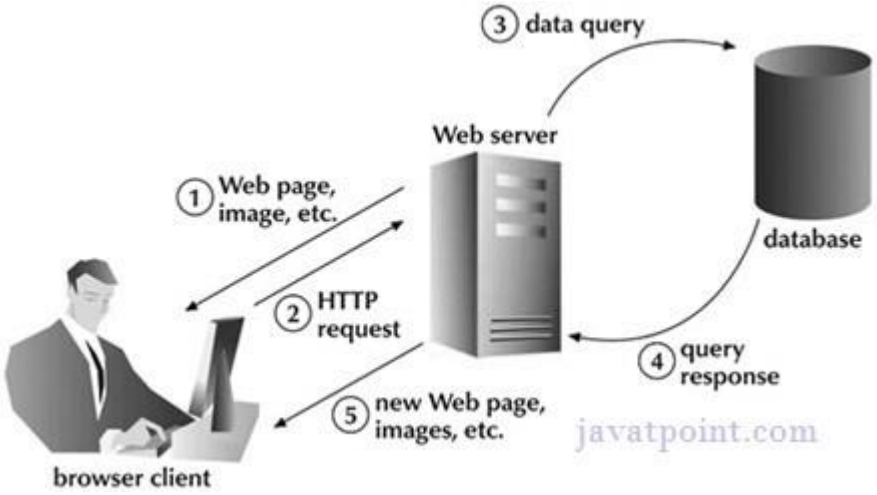
Synchronous (Classic Web-Application Model)

A synchronous request blocks the client until operation completes i.e. browser is unresponsive. In such case, javascript engine of the browser is blocked.



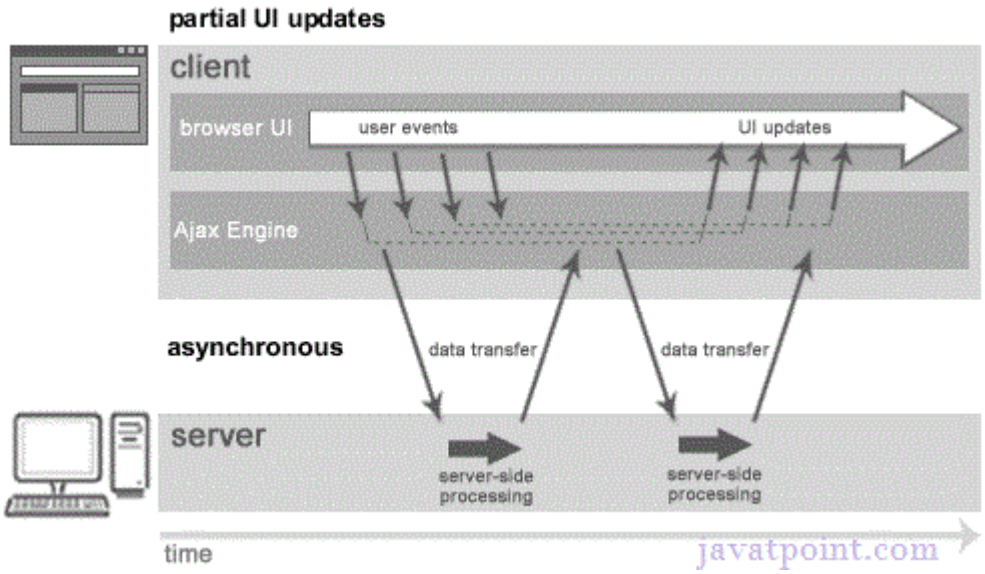
As you can see in the above image, full page is refreshed at request time and user is blocked until request completes.

Let's understand it another way.



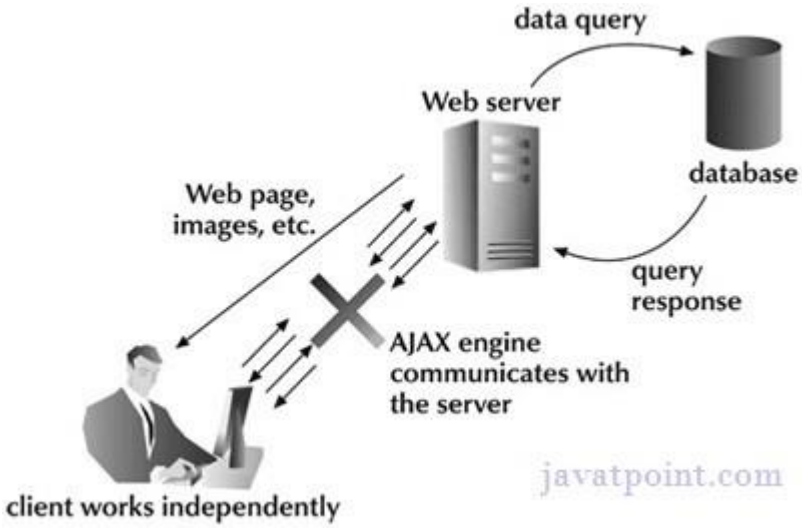
Asynchronous (AJAX Web-Application Model)

An asynchronous request doesn't block the client i.e. browser is responsive. At that time, user can perform another operations also. In such case, javascript engine of the browser is not blocked.



As you can see in the above image, full page is not refreshed at request time and user gets response from the Ajax engine.

Let's try to understand asynchronous communication by the image given below.



As my application was client server architecture, the requests are of paramount importance, so Sync and Async was literally my bread and butter when it came to requests.

Socket Programming:

This class implements client sockets (also called just "sockets"). A socket is an endpoint for communication between two machines. The actual work of the socket is performed by an instance of the Socketimpl class. An application, by

changing the socket factory that creates the socket implementation, can configure itself to create sockets appropriate to the local firewall.

As I am working in Java it is also worth mentioning that, **Java** is very simple and yet pretty powerful in handling TCP/IP and UDP sockets. No platform dependence (unlike Winsock or UNIX sockets) as it is pure TCP. Also, most network programming will need multi-threading and **Java's** multi-threading capabilities are first-class.

STOMP Over Web Socket?

STOMP, an acronym for Simple Text Oriented Messaging Protocol, is a simple HTTP-like protocol for interacting with any STOMP message broker. Any STOMP client can interact with the message broker and be interoperable among languages and platforms.

For this some of the libraries or Apis I went through are as follow:

NaikSoftware/StompProtocolAndroid:

This library provides support for STOMP protocol https://stomp.github.io/ . At now library works only as client for backend with support STOMP, such as NodeJs or Spring Boot.

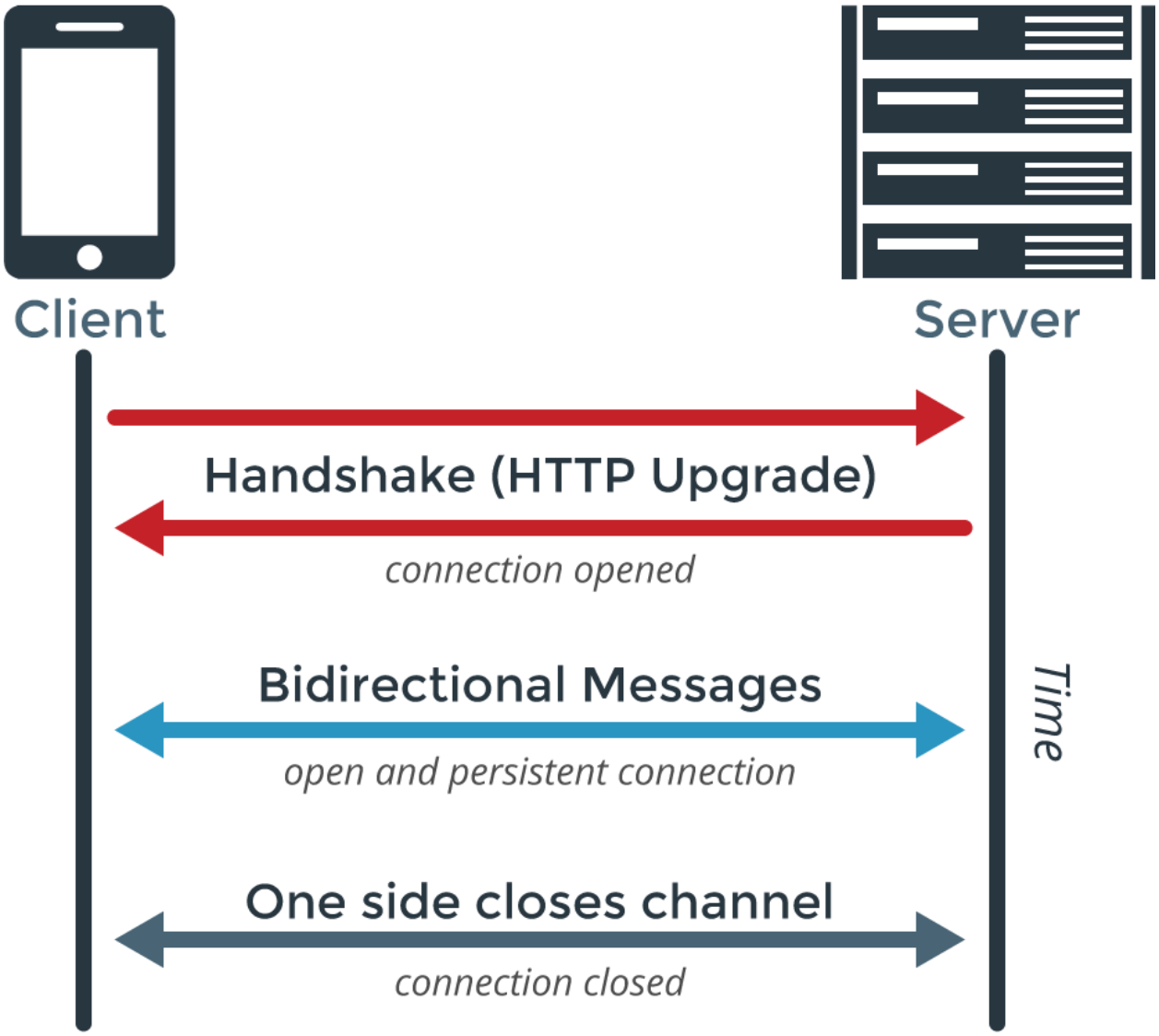
The github link is as follow:

https://github.com/NaikSoftware/StompProtocolAndroid

PubNub real time communication Api

https://www.pubnub.com/products/pubnub-platform/

Following image describes the sockets better:



Additionally, I have learned many new things like SQLite, GSON, JSON but they are not stand alone techs and therefore not necessary to mention in detail.

I have now spent almost a year with my team and my Supervisor Dr. Yasir Niaz Khan, It was one of the most joyous and challenging time, I spent throughout my university not only for itself but also how it shapes one for his career ahead. Dr. Yasir is a perfectionist and convincing him about something is a challenge in itself. Also my team, Ali Raza and Ghulam Muhammad Sani I am ever grateful to them. They worked hard and with faith and taught me the importance of efficient team work.

ADEEM AKRAM

August 4, 2020

adeemakram@outlook.com

A final Year Project compromises many things hard work, teamwork, and learning being some. I started working on this project from September. In my group we specifically don't work on anything particular, we all study everything but we emphasize one particular domain each, mine being server-side.
Starting from September initially, I worked on our project proposal, learned work breakdown structure, and ms project management during this. The first step towards our project was research, which tool to use? Which technologies to employ? Etc... These questions needed answers. We all started with basic android programming with an emphasis on design. We planned to keep our interface simple and flexible. After this my group seconded me towards research about vehicles in major cities of Pakistan, to check whether our idea was viable or not and I learned some interesting facts about these. There are over a million cars in Lahore with an increase of 3000 every month. For us, these were a million drivers we can tap into. After finalizing my research we planned the execution. Mine was the servlets and the filters on the server-side. These are used for filtering the requests and to make sure about the security of our APIs. To efficiently perform my task I learned APIs first. Here I will share some details about APIs which I learned and think are helpful

APIs:

An application programming interface is an interface or communication protocol between different parts of a computer program intended to simplify the implementation and maintenance of software. An API may be for a web-based system, operating system, database system, computer hardware, or software library. In other words APIs are the functions already written by someone else and we can just use them for our work.

My primary aim was to have a grip on RESTful API. Here I will share details which I found interesting and necessary.

A RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data.

A RESTful API -- also referred to as a RESTful web service or REST API -- is based on representational state transfer (REST) technology, an architectural style and approach to communications often used in web services development.

REST technology is generally preferred to the more robust Simple Object Access Protocol (SOAP) technology because REST leverages less bandwidth, making it more suitable for internet usage. An API for a website is code that allows two software programs to communicate with each another. The API spells out the proper way for a developer to write a program requesting services from an operating system or other application.

The framework I started working on was spring boot. **Spring Boot** is an open source Java-based framework used to create a micro Service, it can also be used to write an API. API can further be divided into:

1. JDBC
2. Model
3. Controller
4. DAO
5. DAO Implementation
6. Row Mapper
7. Service
8. Service Implementation

REST (Representational State Transfer) Vs. SOAP (Simple Object Access Protocol)

Difference	SOAP	REST
Style	Protocol	Architectural style
Function	Function-driven: transfer structured information	Data-driven: access a resoruce for data
Data format	Only uses XML	Permits many data formats, including plain text, HTML, XML, and JSON
Security	Supports WS-Security and SSL	Supports SSL and HTTPS
Bandwidth	Requires more resources and bandwidth	Requires fewer resources and is lightweight
Data cache	Can not be cached	Can be cached
Payload handling	Has a strict communication contract and needs knowledge of everything before any interaction	Needs no knowledge of the API
ACID compliance	Has built-in ACID compliance to reduce anomalies	Lacks ACID compliance

REST and SOAP offer different methods to invoke a web service. REST is an architectural style, while SOAP defines a standard communication protocol specification for XML-based message exchange. REST applications can use SOAP.

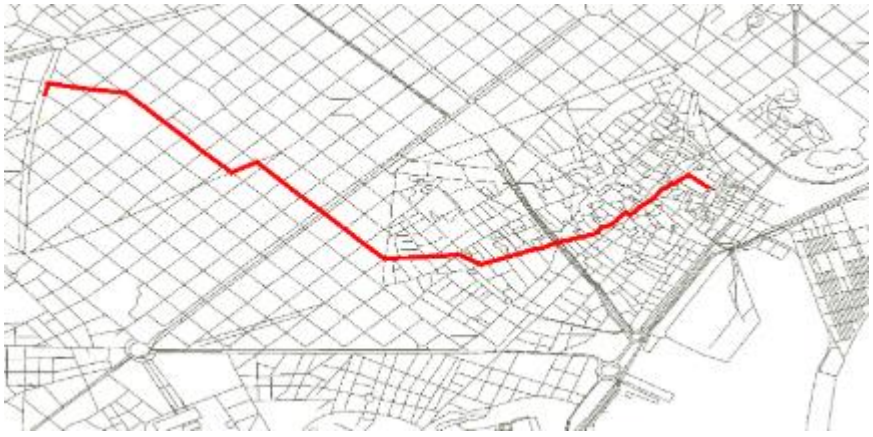
RESTful web services are stateless. A REST-based implementation is simple compared to SOAP, but users must understand the context and content being passed along, as there's no standard set of rules to describe the REST web services interface. REST services are useful for restricted profile devices, such as mobile, and are easy to integrate with existing websites. While I was learning how to write APIs I encountered a very interesting quote" If there were no APIs we still would have been living in digital stone age". As I made my grip stronger I moved to servlets and filters. They are essentially a barrier when someone tries to hit our urls. It is kind of interesting as well as challenging. A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers.

There are two types of servlets the ones I learned and worked on are; HTTP servlets. Extend javax.servlet.HttpServlet. Have built-in HTTP protocol support and are more useful in a Sun Java System Web Server environment. We could also have used generic servlets, but they are platform independent making it hard to code and use. Servlet Filters are pluggable java components that we can use to intercept and process requests before they are sent to servlets and response after servlet code is finished and before container sends the response back to the client. Well our app needs to transmit data in real time for that I started chat framework. An app cannot perform well if its database is not working efficiently. I needed to use joins and normalization techniques which was a very hard task indeed, it took me some time to excel it and to get our database ready.

Latently my project supervisor demanded that artificial intelligence must be included into the project, me having keen interest in the subject jumped right into it.The goal on which I am currently working is to mark the shortest routes available to the driver considering passengers, duration, and fuel cost. The algorithm that would be used is A star Algorithm. I studied it in depth, basics are as follow:

AI

A* (pronounced "A-star")
Is a graph traversal and path search algorithm, which is often used in computer science due to its completeness, optimality, and optimal efficiency. One major practical drawback is its space complexity, as it stores all generated nodes in memory. Thus, in practical travel-routing systems, it is generally outperformed by algorithms which can pre-process the graph to attain better performance, as well as memory-bounded approaches; however, A* is still the best solution in many cases.



A* is an informed search algorithm, or a best-first search, meaning that it is formulated in terms of weighted graphs: starting from a specific starting node of a graph, it aims to find a path to the given goal node having the smallest cost (least distance travelled, shortest time, etc.). It does this by maintaining a tree of paths originating at the start node and extending those paths one edge at a time until its termination criterion is satisfied.

At each iteration of its main loop, A* needs to determine which of its paths to extend. It does so based on the cost of the path and an estimate of the cost required to extend the path all the way to the goal. Specifically, A* selects the path that minimizes the cost.

Me and my group members are also going to start numerical computing and linear algebra to come up with a formula to calculate cost to use A* Algorithm.

All mentioned above it is also worth mentioning that I studied JSON and GSON, and how to parse data from one into another.

I still have some time left and I want to maximize my efforts to get most of my time and project.

Bootstrap

Bootstrap is used to quickly design and customize the Site, the world's most popular front-end open-source toolkit, responsive grid system, extensive prebuilt components Bootstrap employs a handful of important global styles and settings that you'll need to be aware of when using it,
Bootstrap is developed *mobile-first*, a strategy in which we optimize code for mobile devices first and then scale up components as necessary using CSS media queries. **Bootstrap's** responsive CSS adjusts to phones, tablets, and desktops. To ensure proper rendering and touch zooming for all devices.

I aimed to use Bootstrap in the Admin Panel in our Project to quickly design the Layout of it because there are so many templates already made by someone.
The template I used in the project:
SB Admin 2 (https://startbootstrap.com/themes/sb-admin-2/)

Admin Panel use for **CRUD** operation (Create, read, update and delete)
In the Admin panel, there are

- Login Page
- Dashboard
- Admin Page (CRUD)
- User Page (CRUD)
- User Group Page (CRUD)
- Organization Page (CRUD)
- Driver Page (CRUD)
- Location Page (CRUD)
- Vehicle Detail Page (CRUD)
- Ride Page (CRUD)
- InterCity Page (CRUD

CRUD operations Implemented using **PHP**

PHP is a server-side scripting language. That is used to develop Static websites or Dynamic websites or Web applications. **HTML** is used to develop static web pages whereas PHP is used to develop components which make website dynamic. The output of PHP script is HTML as they will be similar to HTML files which can include both HTML and PHP in script code but for the end user while opening PHP web page be able to access only HTML elements.

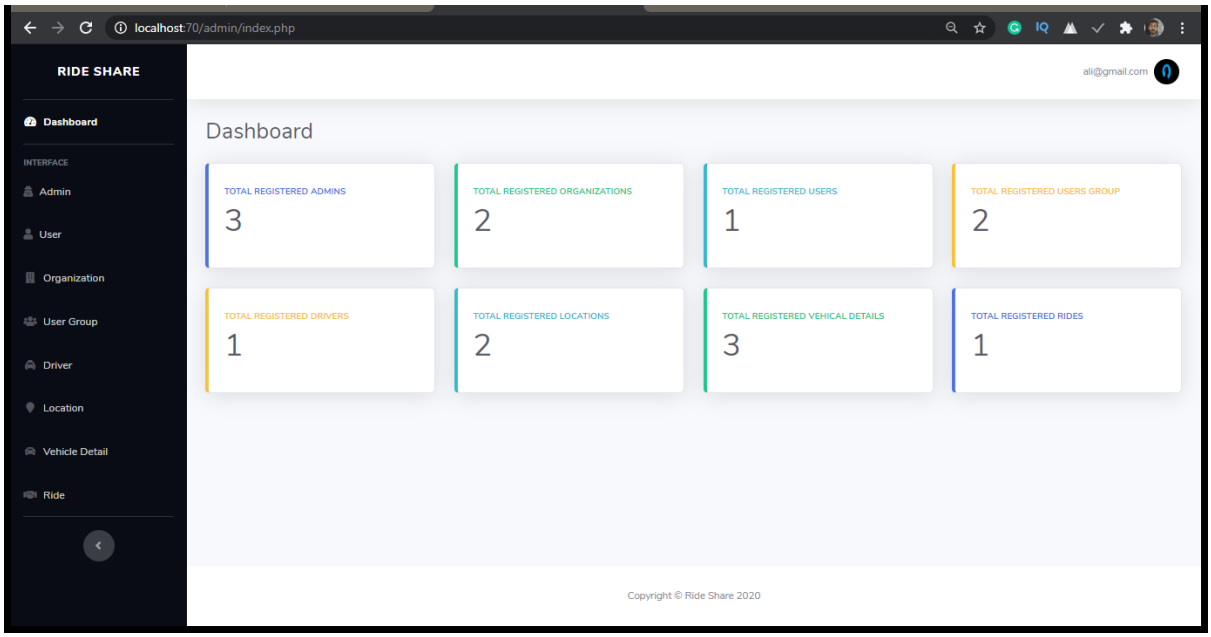
So I already know the basic knowledge of PHP because of the web Application course. Someway I know something about how I am going to implement the CRUD operation in the Bootstrap, so I continue revised PHP and implemented the PHP into bootstrap.

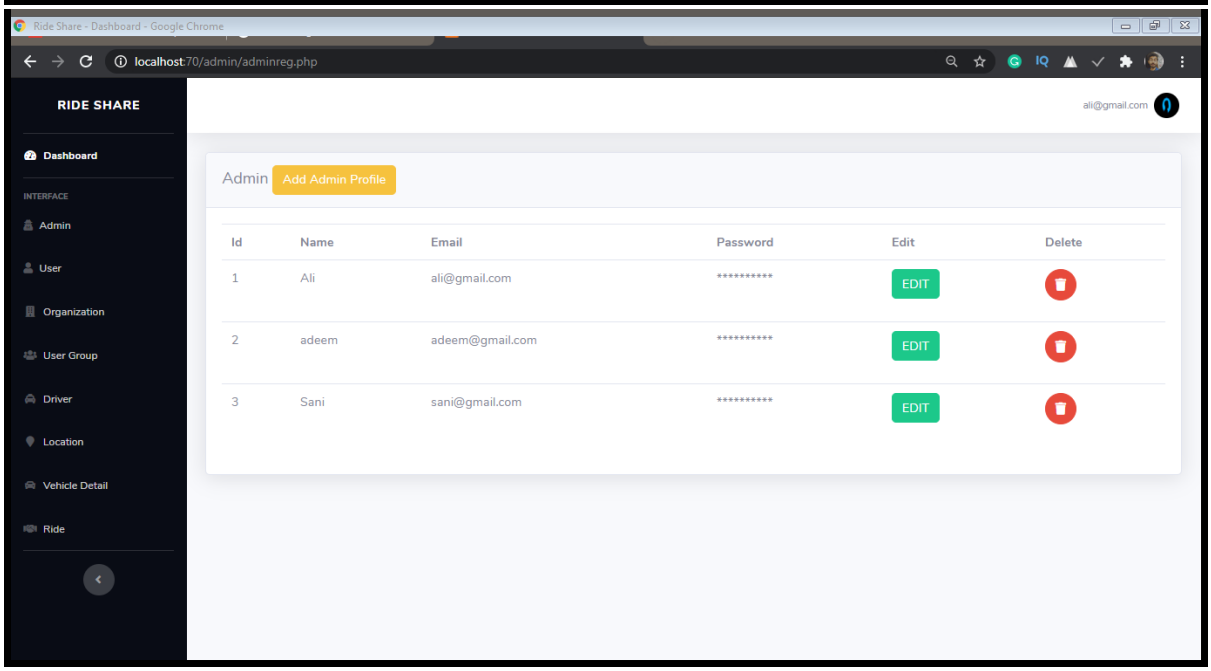
In the implementation state I occurred many problems.

First I have to set the layout suit to our project have to delete so many things that we don't need in it after that changes the HTML code to PHP, and then add more page in it, write queries.
Learn about session and how to use it in login process. A **session** is a way to store information (in variables) to be used across multiple pages. Unlike a cookie, the information is not stored on the user's computer.

The Admin panel continuously grows whenever we create a new table in the database to manage the data.

This is the outlook of our admin panel





JSON

It is primarily **used** to transmit data between a server and web applications. Web services and APIs use **JSON** format to provide public data.

A **JSON** object is a key-value data format Key-value pairs have a colon between them as in "key": "value". Each key-value pair is separated by a comma, so the middle of a **JSON looks like**


```
[
    {
        "name": "bank of pun",
        "address": "adfawef",
        "groupCode": "1133",
        "longitude": "14.2645998",
        "latitude": "14.1653996",
        "ml": "dont know"
    },
    {
        "name": "Uol",
        "address": "Raiwand road",
        "groupCode": "44558",
        "longitude": "120.5479965",
        "latitude": "254.5167999",
        "ml": "dont know"
    }
]
```


JSON is used to sending some data from the server to the client. It uses in our app parsing the JSON data to the data that can be used by the client. JSON Data is easy to understand by humans and machines. On the other hand, is used to create a **JSON** string out of an object or array. It's a lightweight data-interchange format that is quickly becoming the default format for data exchange on the internet today

An **APIs** that **return JSON** objects **are** RESTful **API** which **is** a type of Web **API**.

To test the API is returning the Correct JSON Data firstly we call the data from the postman and check.

Now my FYP is nearly done, it taught me much. It helped me to shape my future career, to improve my skills and flourish. To contribute to the betterment of the community and make a name for me in the IT world. I also want to thank our project supervisor Yasir Niaz Khan and my wonderful team Adeem Akram and Ghulam Muhammad Sani. A better combination doesn't exist.

Ali Raza

August 4, 2020

Aliraza90s@outlook.com

Since I have started working on this project along with my team members under the guidance of Dr. Yasir Niaz Khan, I learnt a lot of things regarding the project. In my team my role is to work on server side which is quite a difficult task to be done but I am doing it till now because of the continuous support of my group members. The work was started after when our project proposal was accepted. And the first challenge was to select a framework to work on.

Selecting a framework

First of all we had to choose a web framework for our server side, and it was my job so, I did research on it and came to a decision that we should use Spring Boot. Spring boot is a java framework, the main reason to use a java framework is that the application we are going to develop is an android application, and the other main reason to use spring boot is that spring is a powerful Java application framework, used in a wide range of Java applications. Spring provides a very clean division between controllers, JavaBean models, and views, Spring MVC is entirely based on interfaces. Furthermore, just about every part of the Spring MVC framework is configurable via plugging in your own interface. Of course we also provide convenience classes as an implementation option. Spring provides interceptors as well as controllers, making it easy to factor out behavior common to the handling of many requests.

Learning Thymeleaf

The first technical thing I learnt working on this project was thymeleaf. Thymeleaf is just like HTML and going to be used to develop a web interface for our application. Basically **Thymeleaf** is a modern server-side Java template engine for both web and standalone environments.

Thymeleaf's main goal is to bring elegant *natural templates* to your development workflow, HTML that can be correctly displayed in browsers and also work as static prototypes, allowing for stronger collaboration in development teams.

With modules for Spring Framework, a host of integrations with your favorite tools, and the ability to plug in your own functionality, Thymeleaf is ideal for modern-day HTML5 JVM web development, although there is much more it can do.

Learning Spring Boot

After selecting a web framework, now it was the time to understand the framework and start working on it. I had to write restful APIs to be deployed at the server side of our application. This is because REST is the most logical, efficient and widespread standard in the creation of APIs for Internet services. To

give a simple definition, REST is any interface between systems using HTTP to obtain data and generate operations on those data in all possible formats, such as XML and JSON.

At that time i had no idea about the restful APIs so, to learn how to write restful APIs and how to use restful services and it took me almost 3 weeks. As we are going to develop a ride sharing application so I had to write a multiple APIs regarding each and every feature and the functionality of our application. An API includes a model class in which the information is mapped that comes as a result of a request from the user side to the server side a model class only has a number of attributes and their respective getters and setters.

The other main component of an Application Programming Interface (API) is the RestController. The controller is a java class that handles the requests coming from the server. Spring RestController annotation is a convenience annotation that is itself annotated with @Controller . This annotation is applied to a class to mark it as a request handler. And there are a lot more annotations that are used in spring boot like @GetMapping and @PostMapping to make Get and Post requests to the server.

Spring RestController annotation is used to create RESTful web services using Spring MVC. Spring RestController takes care of mapping request data to the defined request handler method. Once response body is generated from the handler method, it converts it to JSON or XML response.

Spring has a complete flow of classes and interfaces that combine to complete a RestAPI. The next one is service and service implementation. Service Components are the class file which contains @Service annotation. These class files are used to write business logic in a different layer.

And the last and most important thing is the database connectivity it uses JDBC template for database connectivity. Spring JdbcTemplate is a powerful mechanism to connect to the database and execute SQL queries. It internally uses JDBC API, but eliminates a lot of problems of JDBC API. In spring boot database connection the basic functionality is performed by DAO interface and DAO implantation class that executes the SQL queries and then comes the row mapper class that maps the entries from the database to the model class.

Writing My First API

After I learnt the java Spring Boot I started writing my first API which was of adding user profile to the database and after writing it, now it was the time to test my first restful API so, when I completed writing the API I needed a tool to test the API and check if it works properly or not.

Testing My API

To get the best testing toot for the API I searched it over the internet and found "Postman". Postman is a great tool for prototyping APIs, and it also has some powerful testing features, a Google Chrome app for interacting with HTTP APIs. It presents you with a friendly GUI for constructing requests and reading responses. That's why I preferred to use postman as my testing tool.

So when I run the server and deployed the API as a spring boot application it worked perfectly, it added a user profile into the database. It was my first accomplishment throughout till that moment.

After that till now I have wrote a lot of APIs for our application using different business logics to perform multiple operations regarding the application requirements, like adding the source and destination locations, their coordinates, user profile settings, user details and performing basic CRUD operations on the information. And we also tested these APIs with our android application and the work is still going on.

Learning Web Sockets

Web Sockets is a **bi-directional, full-duplex, persistent connection** between a web browser and a server. Once a Web Socket connection is established the connection stays open until the client or server decides to close this connection. This is the best way to make a connection between the users and establish a continuous real time communication. So to perform basic chat functionality I learnt spring web sockets recently. It uses html and JavaScript and java language to perform the basic communication

Rest vs Soap

SOAP - SOAP is a protocol which was designed before REST and came into the picture. The main idea behind designing SOAP was to ensure that programs built on different platforms and programming languages could exchange data in an easy manner.

REST - This was designed specifically for working with components such as media components, files, or even objects on a particular hardware device. Any web service that is defined on the principles of REST can be called a RestFul web service. A Restful service would use the normal HTTP verbs of GET, POST, PUT and DELETE for working with the required components.

They key differences are as follow:

SOAP	**REST**
SOAP stands for Simple Object Access Protocol	REST stands for Representational State Transfer
SOAP is a protocol. SOAP was designed with a specification. It includes a WSDL file which has the required information on what the web service does in addition to the location of the web service.	REST is an Architectural style in which a web service can only be treated as a RESTful service if it follows the constraints of being 1. Client Server 2. Stateless 3. Cacheable 4. Layered System 5. Uniform Interface
SOAP cannot make use of REST since SOAP is a protocol and REST is an architectural pattern.	REST can make use of SOAP as the underlying protocol for web services, because in the end it is just an architectural pattern.
SOAP uses service interfaces to expose its functionality to client applications. In SOAP, the WSDL file provides the client with the necessary information which can be used to understand what services the web service can offer. SOAP requires more bandwidth for its usage. Since SOAP Messages contain a lot of information inside of it, the	REST use Uniform Service locators to access to the components on the hardware device. For example, if there is an object which represents the data of an employee hosted on a URL as http://demo.guru99 , the below are some of URI that can exist to access them REST does not need much

amount of data transfer using SOAP is generally a lot.	bandwidth when requests are sent to the server. REST messages mostly just consist of JSON messages. Below is an example of a JSON message passed to a web server. You can see that the size of the message is comparatively smaller to SOAP.
SOAP can only work with XML format. As seen from SOAP messages, all data passed is in XML format.	REST permits different data format such as Plain text, HTML, XML, JSON, etc. But the most preferred format for transferring data is JSON.

Considering both, the use of one depends on the project itself. I used rest for the following reasons.

- **Limited resources and bandwidth** – Since SOAP messages are heavier in content and consume a far greater bandwidth, REST should be used in instances where network bandwidth is a constraint.
- **Statelessness** – If there is no need to maintain a state of information from one request to another then REST should be used. If you need a proper information flow wherein some information from one request needs to flow into another then SOAP is more suited for that purpose. We can take the example of any online purchasing site. These sites normally need the user first to add items which need to be purchased to a cart. All of the cart items are then transferred to the payment page in order to complete the purchase. This is an example of an application which needs the state feature. The state of the cart items needs to be transferred to the payment page for further processing.
- **Caching** – If there is a need to cache a lot of requests then REST is the perfect solution. At times, clients could request for the same resource multiple times. This can increase the number of requests which are sent to the server. By implementing a cache, the most frequent queries results can be stored in an intermediate location. So whenever the client requests for a resource, it will first check the cache. If the resources exist then, it will not proceed to the server. So caching can help in minimizing the amount of trips which are made to the web server.

- **Ease of coding** – Coding REST Services and subsequent implementation is far easier than SOAP. So if a quick win solution is required for web services, then REST is the way to go.

Sending server response wrapping it in response object instead of raw:

In client server architecture, a server returns a response either in the form of json/ Xml or views. Initially I sent the response raw as server generated this led to many complications e.g. Server throwing an exception instead of response, the client will try to parse that response leading to app crashes. The approach we worked on was wrapping the response in the form of response object and sending that instead.

We created a Response Object Class, inside it was Response Code, Response description and payload.

Response code tells the client what kind of response was returned. Response codes are as follow:

200 (Successful)

400 (Bad Request)

401 (Unauthorized)

403 (Forbidden)

404 (Not Found)

405 (Method Not Allowed)

500 (Internal Server Error)

Before parsing the entire response, we imposed a check on response code if it is successful we parsed the data if not we showed the description to the user.

Along with response code, response description is important as well. It gives the description in English what kind of response is returned by the server so it can be showed to the client.

Finally the payload, in this we stored the actual response from the server to the client in the form of json.

Http methods

Post

The POST verb is most-often utilized to **create** new resources. In particular, it's used to create subordinate resources. That is, subordinate to some other (e.g. parent) resource. In other words, when creating a new resource, POST to the parent and the service takes care of associating the new resource with the parent, assigning an ID (new resource URI), etc.

On successful creation, return HTTP status 201, returning a Location header with a link to the newly-created resource with the 201 HTTP status.

POST is neither safe nor idempotent. It is therefore recommended for non-idempotent resource requests. Making two identical POST requests will most-likely result in two resources containing the same information.

Get

The HTTP GET method is used to **read** (or retrieve) a representation of a resource. In the “happy” (or non-error) path, GET returns a representation in XML or JSON and an HTTP response code of 200 (OK). In an error case, it most often returns a 404 (NOT FOUND) or 400 (BAD REQUEST).

According to the design of the HTTP specification, GET (along with HEAD) requests are used only to read data and not change it. Therefore, when used this way, they are considered safe. That is, they can be called without risk of data modification or corruption—calling it once has the same effect as calling it 10 times, or none at all. Additionally, GET (and HEAD) is idempotent, which means that making multiple identical requests ends up having the same result as a single request.

Do not expose unsafe operations via GET—it should never modify any resources on the server.

Put

PUT is most-often utilized for **update** capabilities, PUT-ing to a known resource URI with the request body containing the newly-updated representation of the original resource.

However, PUT can also be used to create a resource in the case where the resource ID is chosen by the client instead of by the server. In other words, if the PUT is to a URI that contains the value of a non-existent resource ID. Again, the request body contains a resource representation. Many feel this is convoluted and confusing. Consequently, this method of creation should be used sparingly, if at all.

Alternatively, use POST to create new resources and provide the client-defined ID in the body representation—presumably to a URI that doesn't include the ID of the resource (see POST below).

On successful update, return 200 (or 204 if not returning any content in the body) from a PUT. If using PUT for create, return HTTP status 201 on successful creation. A body in the response is optional—providing one consumes more bandwidth. It is not necessary to return a link via a Location header in the creation case since the client already set the resource ID.

PUT is not a safe operation, in that it modifies (or creates) state on the server, but it is idempotent. In other words, if you create or update a resource using PUT and then make that same call again, the resource is still there and still has the same state as it did with the first call.

If, for instance, calling PUT on a resource increments a counter within the resource, the call is no longer idempotent. Sometimes that happens and it may be enough to document that the call is not idempotent. However, it's recommended to keep PUT requests idempotent. It is strongly recommended to use POST for non-idempotent requests.

Delete

DELETE is pretty easy to understand. It is used to **delete** a resource identified by a URI.

On successful deletion, return HTTP status 200 (OK) along with a response body, perhaps the representation of the deleted item (often demands too much bandwidth), or a wrapped response (see Return Values below). Either that or return HTTP status 204 (NO CONTENT) with no response body. In other words, a 204 status with no body, or the JSEND-style response and HTTP status 200 are the recommended responses.

HTTP-spec-wise, DELETE operations are idempotent. If you DELETE a resource, it's removed. Repeatedly calling DELETE on that resource ends up the same: the resource is gone. If calling DELETE say, decrements a counter (within the resource), the DELETE call is no longer idempotent. As mentioned previously, usage statistics and measurements may be updated while still considering the service idempotent as long as no resource data is changed. Using POST for non-idempotent resource requests is recommended.

There is a caveat about DELETE idempotence, however. Calling DELETE on a resource a second time will often return a 404 (NOT FOUND) since it was already removed and therefore is no longer findable. This, by some opinions, makes DELETE operations no longer idempotent, however, the end-state of the resource

is the same. Returning a 404 is acceptable and communicates accurately the status of the call.

Above are the basic http methods that I studied and learned, I used post method most frequently.

Rest Controller vs Controller

In the Spring framework, A Controller is a class that is responsible for preparing a model Map with data to be displayed by the view as well as choosing the right view itself. It can also directly write into the response stream by using @ResponseBody annotation and complete the request.

The behavior of writing directly into response stream is very useful for responding calls to RESTful web services because their we just return data instead of returning a view as explained in my earlier post about how Spring MVC works internally.

If you have developed RESTful Web services before Spring 4 like in Spring 3 or Spring 3.1, you would have been familiar with using a combination of @Controller and @ResponseBody to create a RESTful response. Spring guys take cognizant of these issues and created @RestController.

Now, you don't need to use @Controller and @RestponseBody annotation, instead you can use @RestController to provide the same functionality. In short, it is a convenience controller that combines the behavior of @Controler and @Response body into one.

This was my overall learning and progress throughout my working on the project along with my group members. And now it is almost done I am looking forward to have a great time in my career, learn and grow further. I want to thank my team Adeem Akram and Ali Raza and my project supervisor Yasir Niaz Khan for their dedication and support. It would not have been possible without these three.

Ghulam Muhammad Sani Buttar

August 4, 2020

sanibuttar23@gmail.com

References

1. https://www.careem.com/en-ae/.
2. https://www.uber.com/pk/en/
3. https://swvl.com/
4. http://dinus.ac.id/repository/docs/ajar/Sommerville-Software-Engineering-10ed.pdf
5. https://www.geeksforgeeks.org/software-engineering-architectural-design/
6. https://www.marsdd.com/news/ride-sharing-the-rise-of-innovative-transportation-services/
7. https://www.dawn.com/news/137591
8. https://www.gallop.com/
9. https://www.sciencedirect.com/science/article/abs/pii/0191260781900157
10. https://medium.com/@tariqul.islam.rony/system-analysis-of-ride-sharing-application-api-and-practical-experience-2d2d0a626263
11. https://www.researchgate.net/publication/276512126_A_Priori_Approach_Of_Real-Time_Ridesharing_Problem_With_Intermediate_Meeting_Locations
12. http://www.cs.unc.edu/~stotts/145/CRC/state.html
13. https://www.geeksforgeeks.org/designing-use-cases-for-a-project/
14. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/
15. https://www.interaction-design.org/literature/topics/ui-design
16. https://medium.com/acing-ai/uber-case-study-the-state-of-autonomous-transport-42bf2da90fb3
17. https://medium.com/acing-ai/uber-case-study-the-state-of-autonomous-transport-42bf2da90fb3
18. https://www.online-sciences.com/robotics/artificial-intelligence-in-transportation-advantages-disadvantages-applications/
19. https://developers.google.com/maps/documentation
20. http://findnerd.com/list/view/Stateless-Protocol-Vs-Stateful-Protocol/662/

Appendix

There are now more than 2.5 billion active **Android devices**. **Android** is a ludicrously popular operating system, and Google let us know just how popular at its I/O keynote today. According to **Android** director Stephanie Cuthbertson, there are more than two and a half billion active **Android devices** around the world today.

MarketsandMarkets forecasts the **smart transportation** market to grow from **USD** 55.0 billion in 2017 to **USD** 149.2 billion by 2023, at a Compound Annual Growth Rate (CAGR) of 14.7% during the forecast period.

The main goal of **Spring Boot** Framework is to reduce Development, Unit Test and Integration Test time and to ease the development of Production ready web applications very easily compared to existing **Spring** Framework, which really takes more time. To provide some defaults to quick start new projects within no time.

The Maps API returns helpful data about places and locations. It is called by javascript. It does two major things:

It can cause maps to appear for the user.

It can return data about a latitude/longitude location, or return data about an address.

Google's documentation is strong for this API, so this guide will mostly link to the official documentation, and discuss how Refuge uses the different API features.