Faculty of Computer
& Artificial Intelligence

Tawseela

Carpooling Mobile App

Content List:

Chapter one: Introduction

- In this chapter, we will discuss a Brief Introduction about our problem, and what motivates us for choosing it, why this problem is important. Then will mention the Scope that informs about challenges that this project aims to solve, also showed Similar System Information, and finally the Overview of the rest of the document.

1.1 Overview:

- The Project “Carpooling” now becomes more important for both the individuals and communities.
- Many countries suffer from economic crises, lack of recourses and even environmental issues.
- At this project our focus on the best usage of our time and resources.
- This Carpooling application main idea is when someone who has a car going the somewhere and he/she has free seats in his/her car he/she can get another person who shares the same road and rider can benefits from the trip with money, company and even build relations.
- On the other side when someone who don’t have car, he/she can take a ride which paying less than taxi and more comfortable and faster than public transports.

1.2 OBJECTIVES:

- Using well-known maps app like “google maps”.
- Using stable and well-supported framework to ensure the efficiency of the application.
- Design the code using “SOLID principles” to ease the future updates.
- Optimize the root of each trip to achieve best performance.
- Authenticate each user joining the system and keep his/her date for any ethical behavior.
- Analyze the other transport systems behavior like “Uber, Careem,…” to improve the service.

1.3 PURPOSE:

- The purpose of this application is helping people making the best usage of their resources and saving (time, money, efforts) and helping communities in saving energy, solving traffic problems and cleaning the environment.

1.4 Scope:

- The scopes for this project are identified to make the transport process easier, the project is developed as a mobile application.
- The planning and documentation collecting of the project is done following ‘SDLC standers’, the design of the project is done following ‘SOLID principles’, and the implementation and testing are going to follow ‘quality assurance’.

1.5 GENERAL CONSTRAINTS:

- The life updates affect the analyses which has been made.
- Pay to get services for maps.
- Inaccurate paths due to expansions.
- Difficult to auditing everyone’s behaves.

Chapter two: Planning and analysis

2.1 Project planning:

2.1.1 FEASIBILITY STUDY:

- The first step of my project was to review the different carpool applications available to find requirements and improvements. Since the Play Store is the official source for Android applications, I used their search engine to find the carpooling application by typing on the Keyword "carpooling".
- A number of similar applications containing the same icon appeared in the results.
- All these applications are from the same publisher, but the difference is that they are all for a different country (carpooling.fr, carpooling.co.uk, …). After installing one of these applications and exploring its various features, I found that it offered long distance travel as well as frequent trips. The only downside is that the application only works in France and to access other countries you need to download a separate application. The other apps were similar to the ones mentioned above. Another app called Carma offered to make payments between passengers and the dryer via the application.
- Multiple app collections offer different types of travel. Parents or teachers can take their children to school, clubs or sports often and alternate. These apps are only available through the web and are not available as mobile apps. An example of this kind of applications Hop Ways.
- After exploring the different applications, I came up with essential features that are feasible and also some improvements that should be considered. Single trips and frequent should be implemented in order to have an application that answers the need of the market.

FOR THE SCOPE OF THIS PROJECT, THE PLAUSIBLE FEATURES TO IMPLEMENT IN ORDER TO IMPROVE WHAT ISAVAILABLE ON THE MARKET ARE:

- Location independent application: the same application (no need to download a country specific app) should work everywhere in the world.
- Socially enabled: Add some features of social apps in the application to improve the usability and get user interest.

- Map picker: for picking the meeting points.
- Personal profile
- Rate method: The rate is on the user not for the trip.

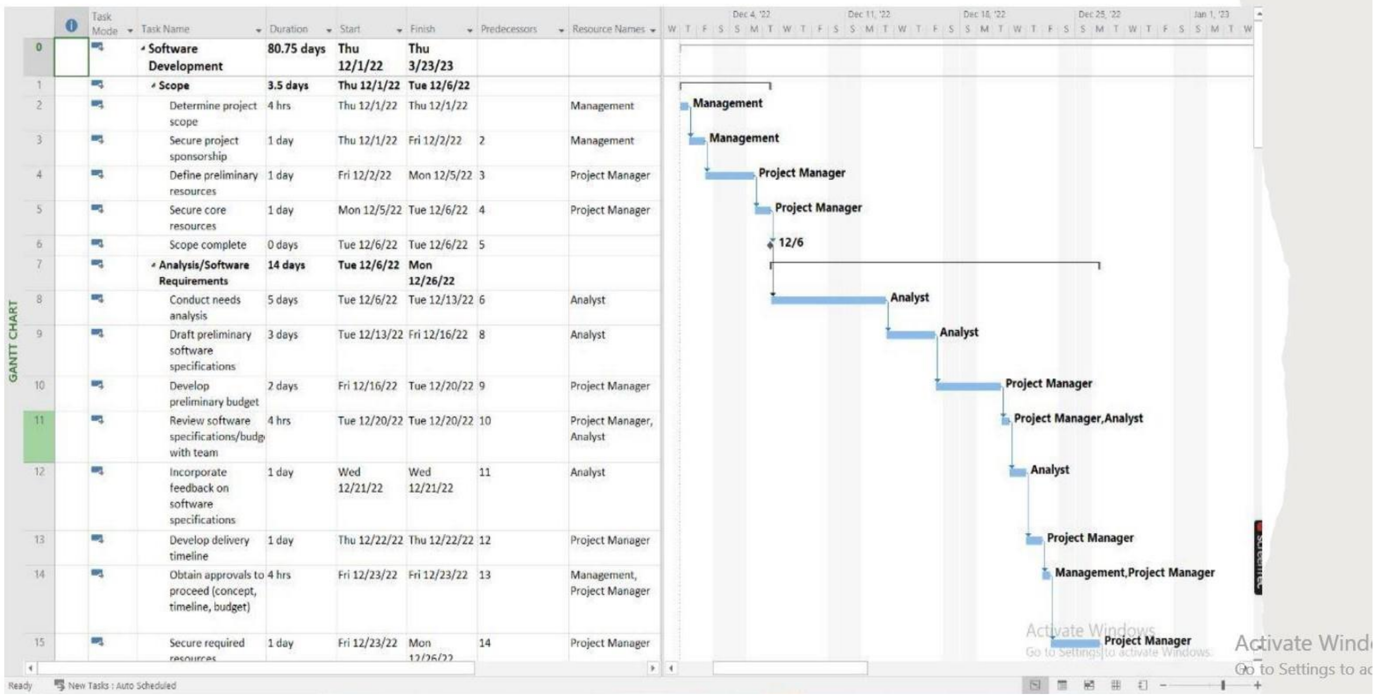
2.1.2 ESTIMATED COST:

- It's more complicated than you think, with many powerhouses already dominating the industry, such as Uber, Lyft, Grab, and BlaBlaCar.
- There are many opportunities to create great, high-performance ride-sharing apps. You should consider the basic steps. This includes conducting proper market research, selecting key features, estimating costs, and executing a strategic plan.
- The key element apps needed to build the ride-sharing service, as you need two builder apps: a passenger app and a driver app. We divide them into two categories.
- The estimated cost of developing these apps in the US ranges from $150,000 to $250,000. It's based on the features you need to create a working Minimum Viable Product (MVP).
- Costs vary depending on where you hire the on-demand app development company or freelancer

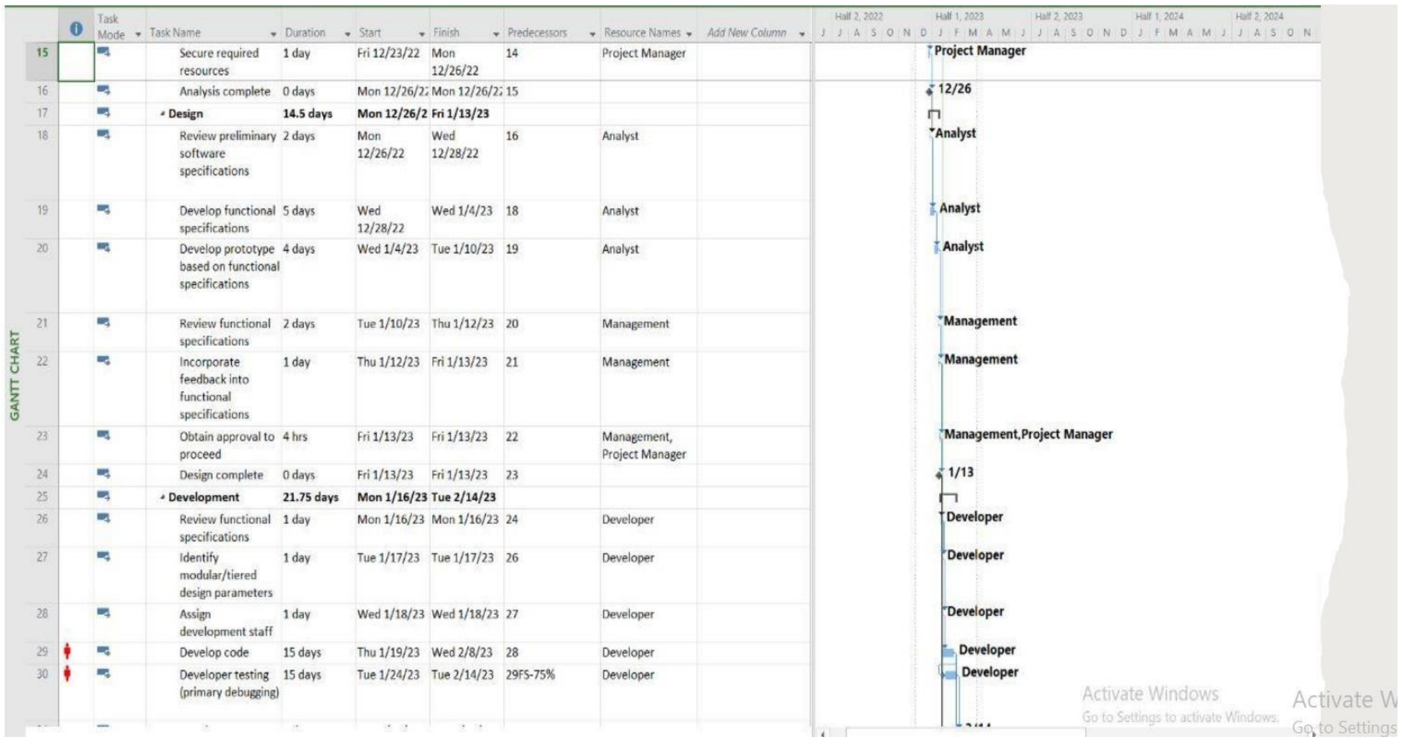
BASIC FEATURES:

- Geolocation and routing.
- Texting the driver right from the app.
- Ride states.
- Push notifications.
- Registration and personal data management.
- Personal profile.
- Browsing users profile.
- Rate.

2.1.3 GANTT CHART

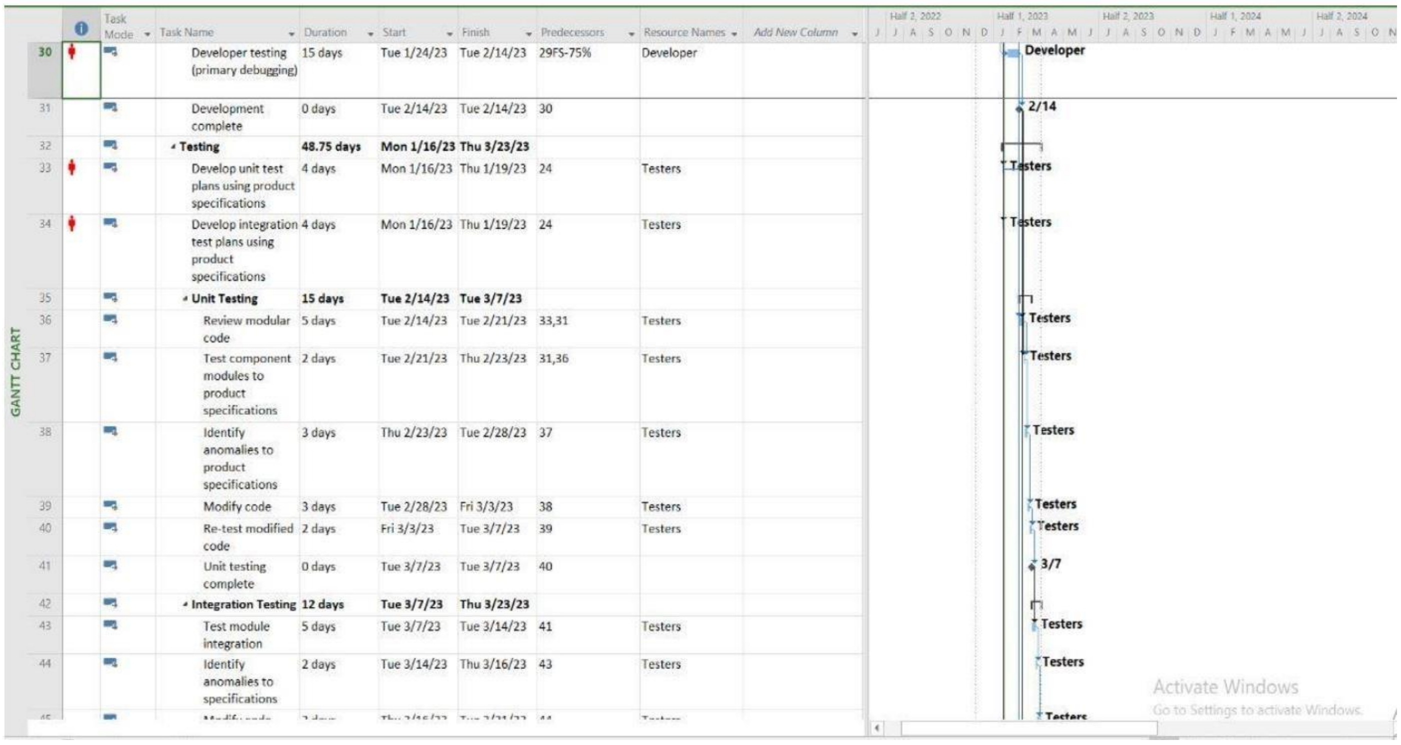


	Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Names
0		**Software Development**	**80.75 days**	**Thu 12/1/22**	**Thu 3/23/23**		
1		**Scope**	**3.5 days**	**Thu 12/1/22**	**Tue 12/6/22**		
2		Determine project scope	4 hrs	Thu 12/1/22	Thu 12/1/22		Management
3		Secure project sponsorship	1 day	Thu 12/1/22	Fri 12/2/22	2	Management
4		Define preliminary resources	1 day	Fri 12/2/22	Mon 12/5/22	3	Project Manager
5		Secure core resources	1 day	Mon 12/5/22	Tue 12/6/22	4	Project Manager
6		Scope complete	0 days	Tue 12/6/22	Tue 12/6/22	5	
7		**Analysis/Software Requirements**	**14 days**	**Tue 12/6/22**	**Mon 12/26/22**		
8		Conduct needs analysis	5 days	Tue 12/6/22	Tue 12/13/22	6	Analyst
9		Draft preliminary software specifications	3 days	Tue 12/13/22	Fri 12/16/22	8	Analyst
10		Develop preliminary budget	2 days	Fri 12/16/22	Tue 12/20/22	9	Project Manager
11		Review software specifications/budg with team	4 hrs	Tue 12/20/22	Tue 12/20/22	10	Project Manager, Analyst
12		Incorporate feedback on software specifications	1 day	Wed 12/21/22	Wed 12/21/22	11	Analyst
13		Develop delivery timeline	1 day	Thu 12/22/22	Thu 12/22/22	12	Project Manager
14		Obtain approvals to proceed (concept, timeline, budget)	4 hrs	Fri 12/23/22	Fri 12/23/22	13	Management, Project Manager
15		Secure required resources	1 day	Fri 12/23/22	Mon 12/26/22	14	Project Manager



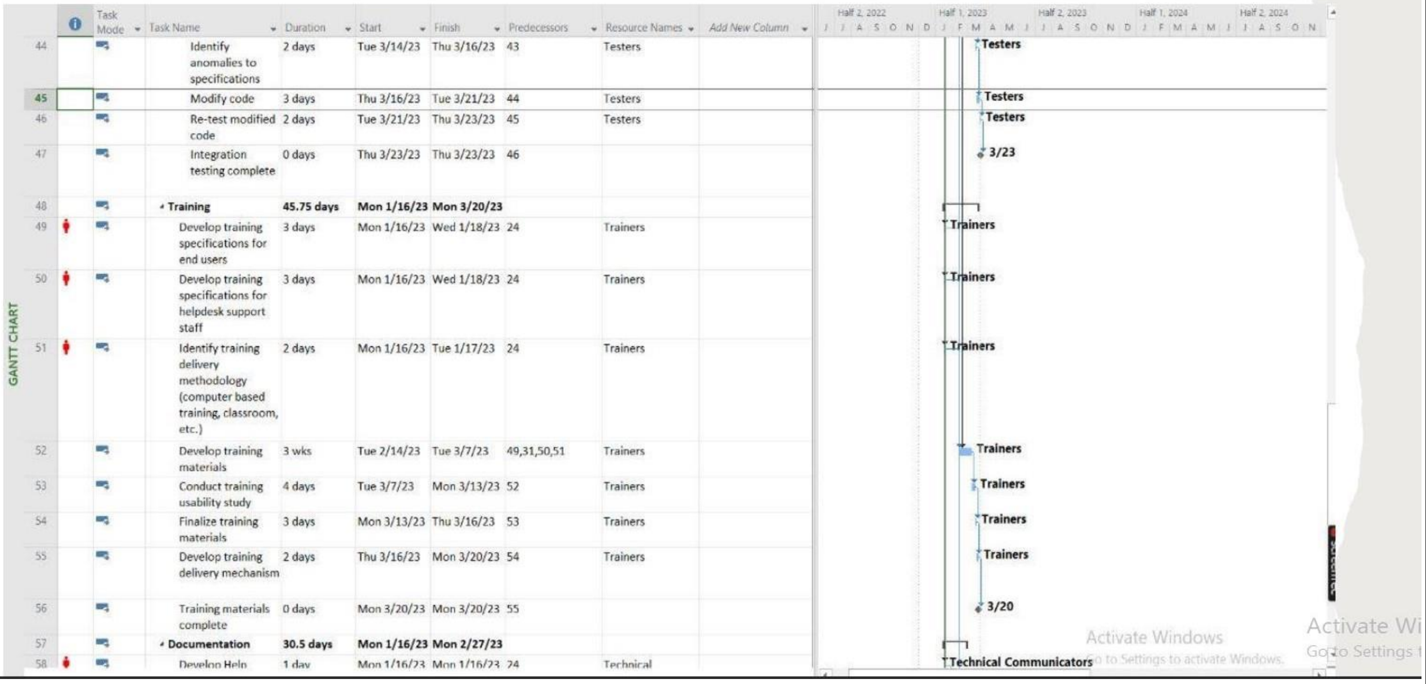
	Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Names	Add New Column
15		Secure required resources	1 day	Fri 12/23/22	Mon 12/26/22	14	Project Manager	
16		Analysis complete	0 days	Mon 12/26/2:	Mon 12/26/2:	15		
17		**Design**	**14.5 days**	**Mon 12/26/2**	**Fri 1/13/23**			
18		Review preliminary software specifications	2 days	Mon 12/26/22	Wed 12/28/22	16	Analyst	
19		Develop functional specifications	5 days	Wed 12/28/22	Wed 1/4/23	18	Analyst	
20		Develop prototype based on functional specifications	4 days	Wed 1/4/23	Tue 1/10/23	19	Analyst	
21		Review functional specifications	2 days	Tue 1/10/23	Thu 1/12/23	20	Management	
22		Incorporate feedback into functional specifications	1 day	Thu 1/12/23	Fri 1/13/23	21	Management	
23		Obtain approval to proceed	4 hrs	Fri 1/13/23	Fri 1/13/23	22	Management, Project Manager	
24		Design complete	0 days	Fri 1/13/23	Fri 1/13/23	23		
25		**Development**	**21.75 days**	**Mon 1/16/23**	**Tue 2/14/23**			
26		Review functional specifications	1 day	Mon 1/16/23	Mon 1/16/23	24	Developer	
27		Identify modular/tiered design parameters	1 day	Tue 1/17/23	Tue 1/17/23	26	Developer	
28		Assign development staff	1 day	Wed 1/18/23	Wed 1/18/23	27	Developer	
29		Develop code	15 days	Thu 1/19/23	Wed 2/8/23	28	Developer	
30		Developer testing (primary debugging)	15 days	Tue 1/24/23	Tue 2/14/23	29FS-75%	Developer	

2.1.3 GANTT CHART



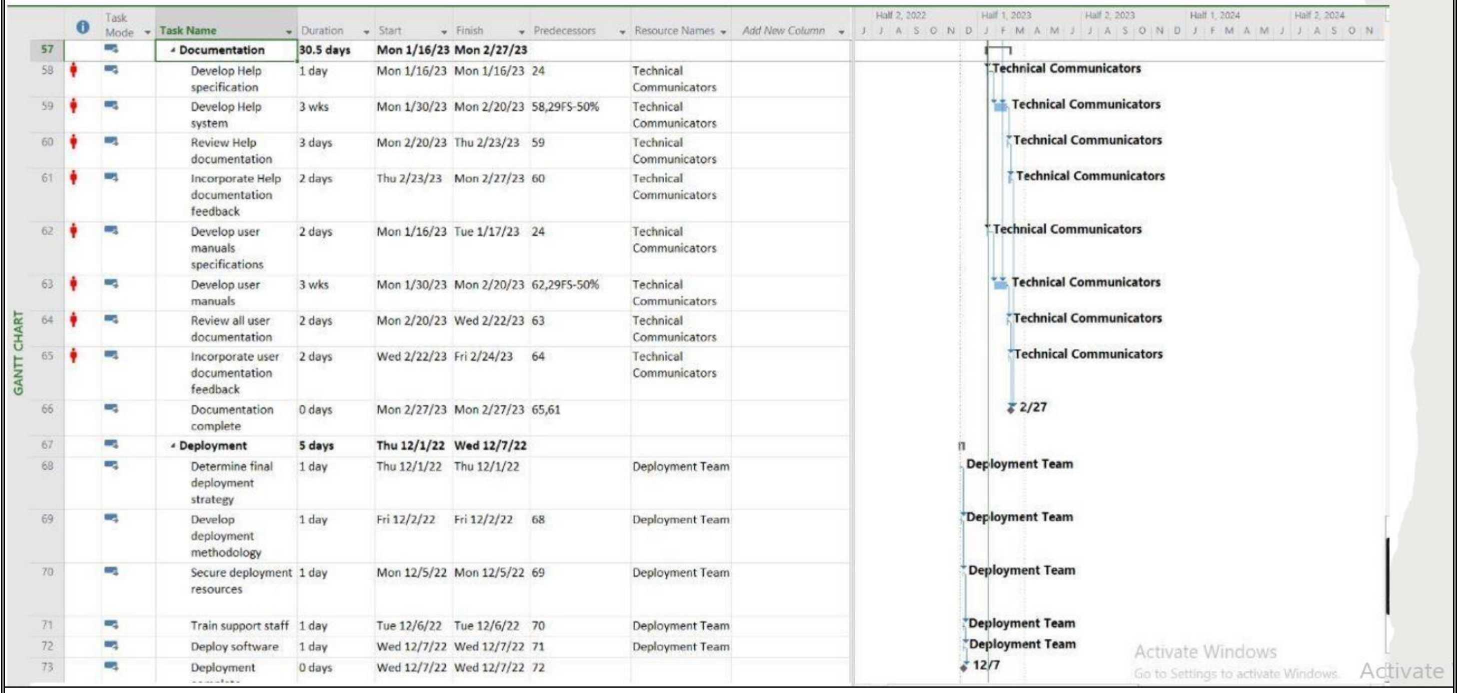
	Task Name	Duration	Start	Finish	Predecessors	Resource Names	Add New Column
30	Developer testing (primary debugging)	15 days	Tue 1/24/23	Tue 2/14/23	29FS-75%	Developer	
31	Development complete	0 days	Tue 2/14/23	Tue 2/14/23	30		
32	**Testing**	**48.75 days**	**Mon 1/16/23**	**Thu 3/23/23**			
33	Develop unit test plans using product specifications	4 days	Mon 1/16/23	Thu 1/19/23	24	Testers	
34	Develop integration test plans using product specifications	4 days	Mon 1/16/23	Thu 1/19/23	24	Testers	
35	**Unit Testing**	**15 days**	**Tue 2/14/23**	**Tue 3/7/23**			
36	Review modular code	5 days	Tue 2/14/23	Tue 2/21/23	33,31	Testers	
37	Test component modules to product specifications	2 days	Tue 2/21/23	Thu 2/23/23	31,36	Testers	
38	Identify anomalies to product specifications	3 days	Thu 2/23/23	Tue 2/28/23	37	Testers	
39	Modify code	3 days	Tue 2/28/23	Fri 3/3/23	38	Testers	
40	Re-test modified code	2 days	Fri 3/3/23	Tue 3/7/23	39	Testers	
41	Unit testing complete	0 days	Tue 3/7/23	Tue 3/7/23	40		
42	**Integration Testing**	**12 days**	**Tue 3/7/23**	**Thu 3/23/23**			
43	Test module integration	5 days	Tue 3/7/23	Tue 3/14/23	41	Testers	
44	Identify anomalies to specifications	2 days	Tue 3/14/23	Thu 3/16/23	43	Testers	

2.1.3 GANTT CHART



	Task Name	Duration	Start	Finish	Predecessors	Resource Names	Add New Column
44	Identify anomalies to specifications	2 days	Tue 3/14/23	Thu 3/16/23	43	Testers	
45	Modify code	3 days	Thu 3/16/23	Tue 3/21/23	44	Testers	
46	Re-test modified code	2 days	Tue 3/21/23	Thu 3/23/23	45	Testers	
47	Integration testing complete	0 days	Thu 3/23/23	Thu 3/23/23	46		
48	**Training**	**45.75 days**	**Mon 1/16/23**	**Mon 3/20/23**			
49	Develop training specifications for end users	3 days	Mon 1/16/23	Wed 1/18/23	24	Trainers	
50	Develop training specifications for helpdesk support staff	3 days	Mon 1/16/23	Wed 1/18/23	24	Trainers	
51	Identify training delivery methodology (computer based training, classroom, etc.)	2 days	Mon 1/16/23	Tue 1/17/23	24	Trainers	
52	Develop training materials	3 wks	Tue 2/14/23	Tue 3/7/23	49,31,50,51	Trainers	
53	Conduct training usability study	4 days	Tue 3/7/23	Mon 3/13/23	52	Trainers	
54	Finalize training materials	3 days	Mon 3/13/23	Thu 3/16/23	53	Trainers	
55	Develop training delivery mechanism	2 days	Thu 3/16/23	Mon 3/20/23	54	Trainers	
56	Training materials complete	0 days	Mon 3/20/23	Mon 3/20/23	55		
57	**Documentation**	**30.5 days**	**Mon 1/16/23**	**Mon 2/27/23**			
58	Develop Help	1 day	Mon 1/16/23	Mon 1/16/23	24	Technical	

2.1.3 GANTT CHART



		Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Names	Add New Column
57			**Documentation**	**30.5 days**	**Mon 1/16/23**	**Mon 2/27/23**			
58			Develop Help specification	1 day	Mon 1/16/23	Mon 1/16/23	24	Technical Communicators	
59			Develop Help system	3 wks	Mon 1/30/23	Mon 2/20/23	58,29FS-50%	Technical Communicators	
60			Review Help documentation	3 days	Mon 2/20/23	Thu 2/23/23	59	Technical Communicators	
61			Incorporate Help documentation feedback	2 days	Thu 2/23/23	Mon 2/27/23	60	Technical Communicators	
62			Develop user manuals specifications	2 days	Mon 1/16/23	Tue 1/17/23	24	Technical Communicators	
63			Develop user manuals	3 wks	Mon 1/30/23	Mon 2/20/23	62,29FS-50%	Technical Communicators	
64			Review all user documentation	2 days	Mon 2/20/23	Wed 2/22/23	63	Technical Communicators	
65			Incorporate user documentation feedback	2 days	Wed 2/22/23	Fri 2/24/23	64	Technical Communicators	
66			Documentation complete	0 days	Mon 2/27/23	Mon 2/27/23	65,61		
67			**Deployment**	**5 days**	**Thu 12/1/22**	**Wed 12/7/22**			
68			Determine final deployment strategy	1 day	Thu 12/1/22	Thu 12/1/22		Deployment Team	
69			Develop deployment methodology	1 day	Fri 12/2/22	Fri 12/2/22	68	Deployment Team	
70			Secure deployment resources	1 day	Mon 12/5/22	Mon 12/5/22	69	Deployment Team	
71			Train support staff	1 day	Tue 12/6/22	Tue 12/6/22	70	Deployment Team	
72			Deploy software	1 day	Wed 12/7/22	Wed 12/7/22	71	Deployment Team	
73			Deployment	0 days	Wed 12/7/22	Wed 12/7/22	72		

2.2 ANALYSIS AND LIMITATION OF EXISTING SYSTEM:

- In the current time the common idea of Carpooling is like taxi applications like “Careem” and “Uber”, etc. and these apps will encourage a lot of people to work as drivers so the number of cars in streets will increase so the crowd will increase too. And this usage of all these cars will cause a lot of consumption of resources and these cars will increase pollution.

2.3 NEED FOR NEW SYSTEM:

- The system helps our environment to be better as we decrease the crowd as the number of cars used in the streets will be less; so the consuming of the resources like gasoline or petrol will be decreased. This system will help people to make more relationships as users will share the same car so they can make conversations and build relationships. This system does not have the attitude of a person who works as a driver and a rider but both of them are users so this will equalize these people in treating each other. Also from the benefits of the system will make the cars used less so the cars will live longer with better performance. The last point is that both of users will have the win-win situation as both of them will reach there destination and both will get benefit specially the driver.

2.4 ANALYSIS OF THE NEW SYSTEM:

2.4.1 USER REQUIREMENTS:

- The system makes a carpooling function as person has a car can take another person with him if the destination of both of them is the same or the route of the car owner intersect with the destination of the person that he takes him with.

2.4.2 SYSTEM REQUIREMENTS:

- The system connects two persons with same route to share one car.
- The system works as way of communication by sending request for a trip or with another person already makes the trip.
- The system also makes the trip maker and the requester start chat with each other after accepting the request for the trip.
- The system also provides a map that shows the route of both users.
- The system also provides kind of rating for each side the trip maker and the person who requests the trip.

2.4.3 DOMAIN REQUIREMENTS:

- Multiple users must be able to use the application simultaneously without corrupting the database (whatever form it may be).

- A server must be set up to host the database, and the server must be accessible by all the systems running the inventory tracking software.

- The database should be backed up every once in a while, in case the original does become corrupt.

- Application must verify all values before making a change in the database.

2.4.4 FUNCTIONAL REQUIREMENTS:

- Admin can add user to the system as a user.
- Admin can also update the information of each user.
- Admin can make the user account on hold or blocked.
- Admin gets dashboard of all trips (active , finished).
- User can register to the system.
- User can login.
- User can update his information in the system.
- User chat with another user.
- User add car.
- User update car.
- User can make the trip but he must have a car.
- User while making a trip he puts his starting point and destination point and the system puts the route.
- User can ask by a request if there is a person who made trip and this user trip subset the route that the trip maker made.
- The trip maker can accept or reject the request.
- If the trip maker accepts the trip he will see the profile of the user who sent the request for the trip.
- Each user will put a rate for the person that was with him on the trip.

2.4.5 NON – FUNCTIONAL REQUIREMENTS:

- Security: To protect sensitive data, you may consider developing non-functional security features.
    - Account creation: Systems may require users to create accounts to access applications that store information and display profiles. A security system typically grants access to accounts when users enter the correct username and password.
    - Password generation: An application may not grant access until the user creates a strong password. For example, a strong password might contain a certain number of characters and a capital letter.
    - Security question answering: A security system for a product may ask questions that only the user knows the answer to. This can help verify a user's identity when they log into an account. Examples of security question topics include the color of your first car or your mother's maiden name.
    - Account locking: After a certain number of login attempts, a security system may lock an account to protect a user's information from potential hackers. To unlock their account, a

user can typically call the company to verify their identity and set a new password.

- Portability: It’s an application so it’s not going to be moved from place or a device to another several times, but also if the application is downloaded on a new phone it will work on it with the same efficiency as it was on the old phone.

- Compatibility: The system is flexible in working on different operating systems as it can work on android devices and IOS devices.

- Localization:
    - Languages: The system supports two languages English and Arabic.
    - Time zone: Each user in registration process will add his country and city to make the system discover the time and the area that this user will cover or use.

- Availability: The system is available for the both sides the trip maker and the trip requester 24 hours and 7 days a week, so both sides can make a trip anytime they want and there is no specific time that the system work at.

- Maintainability: The system will be made in solid principles so it will be modified easily by adding new features or removing features.

- Usability:
    - Navigation: Users can easily navigate through the interface, as the (UI) is simple with no messy shapes or forms and there is no crowd in each interface with buttons and words.
    - Purpose of features: The interface looks simple and some buttons will be predicted easily if it has a picture on it or if not the words used will be simple and will note make confusion.

2.5 ADVANTAGE OF THE NEW SYSTEM:

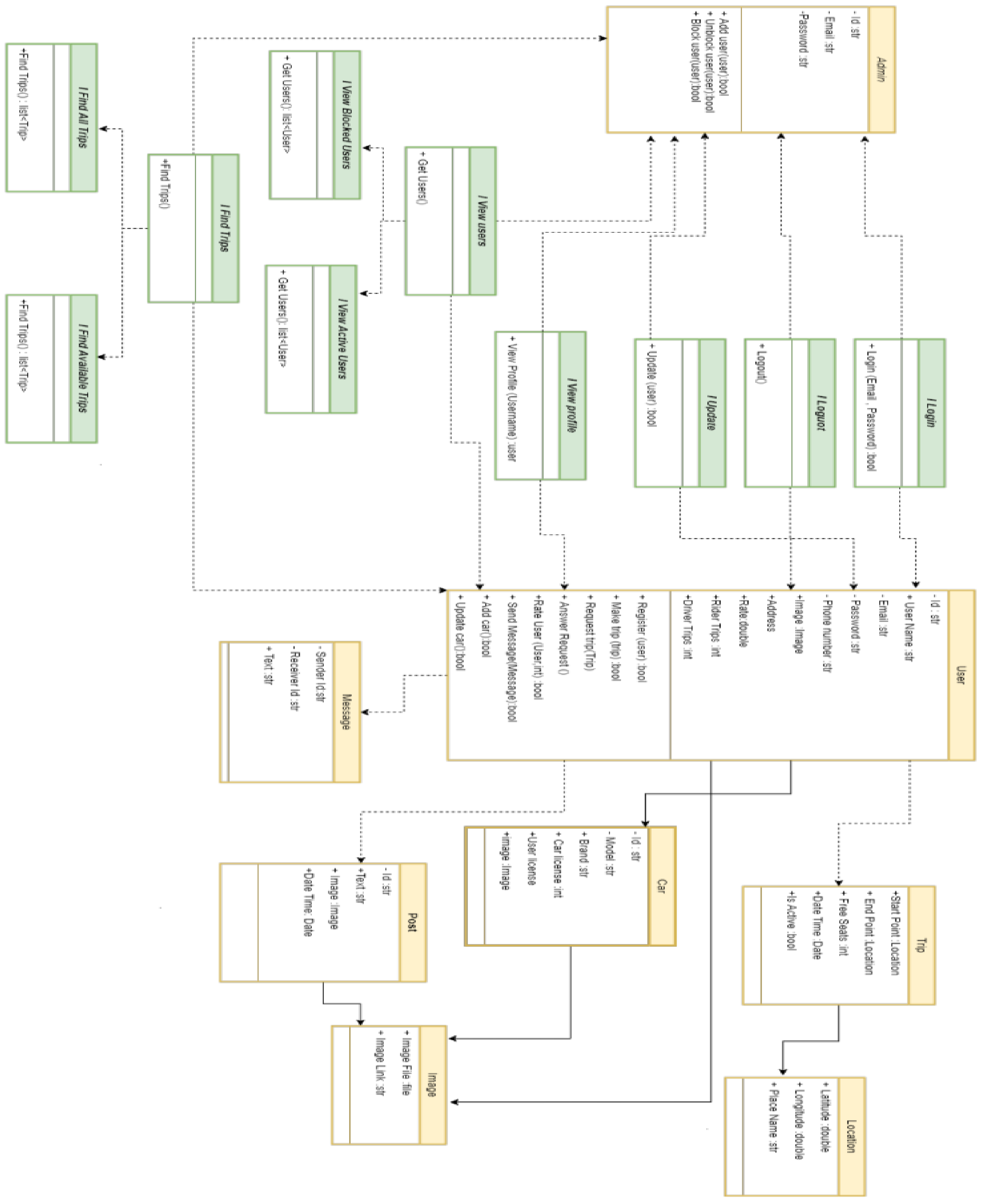
- Take ride as private transport and pay much less.
- Benefit from your empty car on your rides.
- Reduce traffic problems.
- Reduce energy usage.
- Reduce Cost of transportation.
- Encourage to Collaboration.
- Encourage to Acquaintance.
- Save car performance.
- Lighten the load of the public transport.

2.6 RISK AND RISK MANAGEMENTS:

- Fake Users
  - Handling Fake Users:
    - Make Sure of User Data.
    - Link User with Phone Number.
- System Failure
  - Handling System Failure:
    - Good Server Structure
    - Regular Maintenance
    - Regular Unit Testing
    - Regular System Testing
- Wrong Path For Drive
  - Handling Wrong Path for Drive:
    - Use One Map Service
    - Create Drawing for the Path
    - User Automatic Access Location
- Users Communication loss
  - Handling Users Communication loss:
    - Make the Communication inside the app.
- Users Multiple Device Access
  - Handling Users Multiple Device Access:
    - Force Only one device active
    - New Device login blocked and notify the user.
- Losing Data
  - Handling Users Losing Data:
    - Make Backup of data.
    - Make Recovery Methods for user.

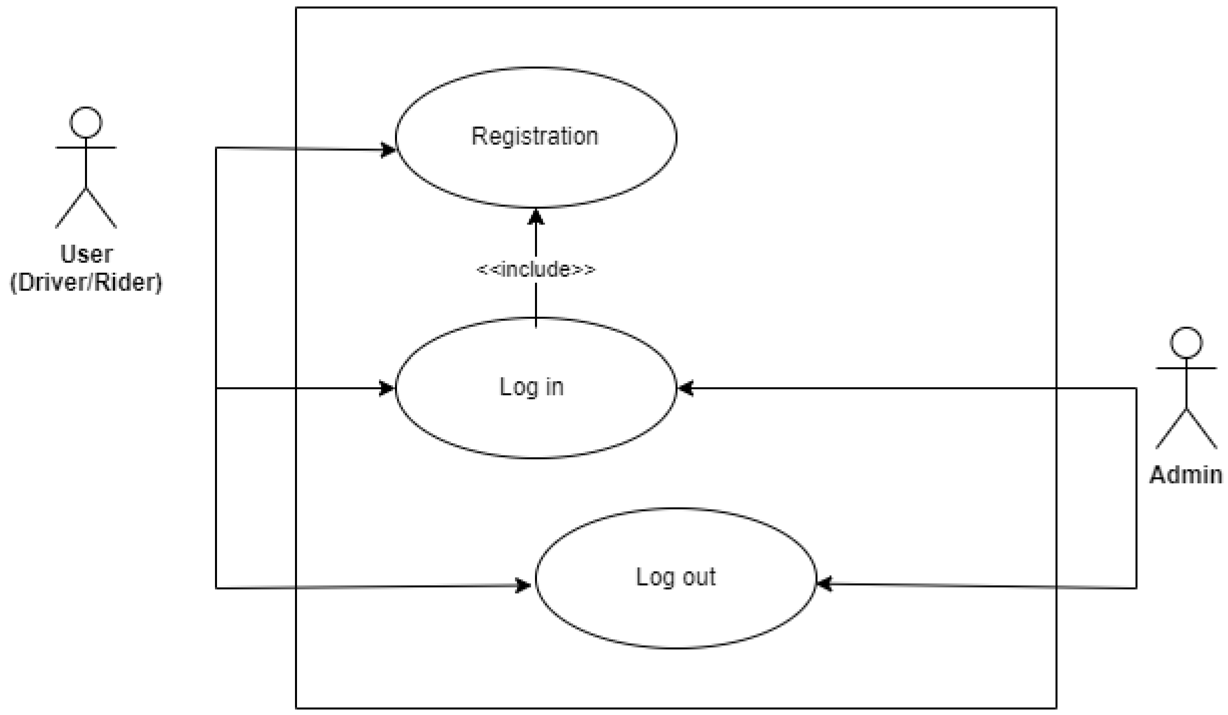
Chapter three: Project “Software Design”

3.1 Class diagram:



3.2 Use Case Diagrams:

- Sign up and log in:



- Registration use case:

Use case name:	Registration
Description:	Create an account on the application.
Actor:	User (Driver and rider).
Goals:	Creating an account on the application.
Pre-Conditions:	Download the application.
Post-Conditions:	Have an account on the application and start to use it (User Name, Email Address, Password, Phone, Address).
Basic flow:	While User Opened the app the login page will appear then he will press on Register now then add his information in the fields.
Alternate flow:	--------

- Log in use case:

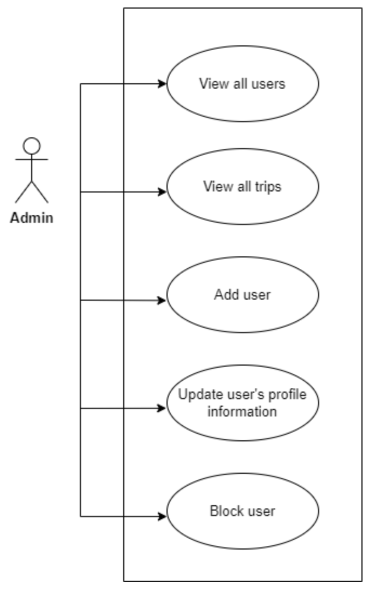
Use case name:	Log in
Description:	Opening the account on the application.

Actor:	User (Driver and rider) / Admin
Goals:	Login to the application
Pre-Conditions:	Having an account on the application
Post-Conditions:	Entering the application successfully (Login to the application) and The user entered the home page of the app.
Basic flow:	Opening the application then the login page will open then you enter the email and password and if the user exist and the email and password are correct so you will enter to the home page of your profile (Logged in) if not so the user will stay at the login page.
Alternate flow:	Opening the application on the registration page then press on the button (login) opening the application then the login page will open then you enter the email and password and if the user exist and the email and password are correct so you will enter to the home page of your profile (Logged in) if not so the user will stay at the login page.

- Log out use case:

Use case name:	Log out
Description:	Logging out from the application and go to the login page
Actor:	User (Driver and rider) / Admin
Goals:	The user log out from the application.
Pre-Conditions:	The user already logged in to the application.
Post-Conditions:	The user will be out of the home page of his own account and navigated to the login page.
Basic flow:	The user is already logged in to the application the press on the profile button then press on the button logout. which navigates the user to the login page.
Alternate flow:	--------

- Admin Use cases:



- Displaying all users use case:

Use case name:	View all users
Description:	Displaying all users of the application in a list.
Actor:	Admin
Goals:	Displaying the users of the application to the admin.
Pre-Conditions:	Previously logged in to the application.
Post-Conditions:	Displaying all the users of the application .
Basic flow:	After logging in to the application the admin opens the users page and in this page all users are appeared to him.
Alternate flow:	--------

- Displaying all trips use case:

Use case name:	View al trips
Description:	Displaying all the trips that has done to the application.
Actor:	Admin
Goals:	Displaying all the trips on the application the active trips and the trips that are ended.
Pre-Conditions:	Admin is logged in to the application.
Post-Conditions:	All the trips in the application have been displayed to the admin.

Basic flow:	After logging in to the application the admin opens the trips page and in this page all trips are appeared to him.
Alternate flow:	--------

- Adding user use case:

Use case name:	Add user
Description:	create a user's account manually
Actor:	Admin
Goals:	Creating an account to the user manually.
Pre-Conditions:	Admin is already logged in to the application and took the permission from the user to make him Account in the application.
Post-Conditions:	The account has been created by the admin to the user.
Basic flow:	After logging in to the application by the admin enters add user page then writes the information of the user as registration form then press on add user button which creates the account to the user with his information.
Alternate flow:	--------

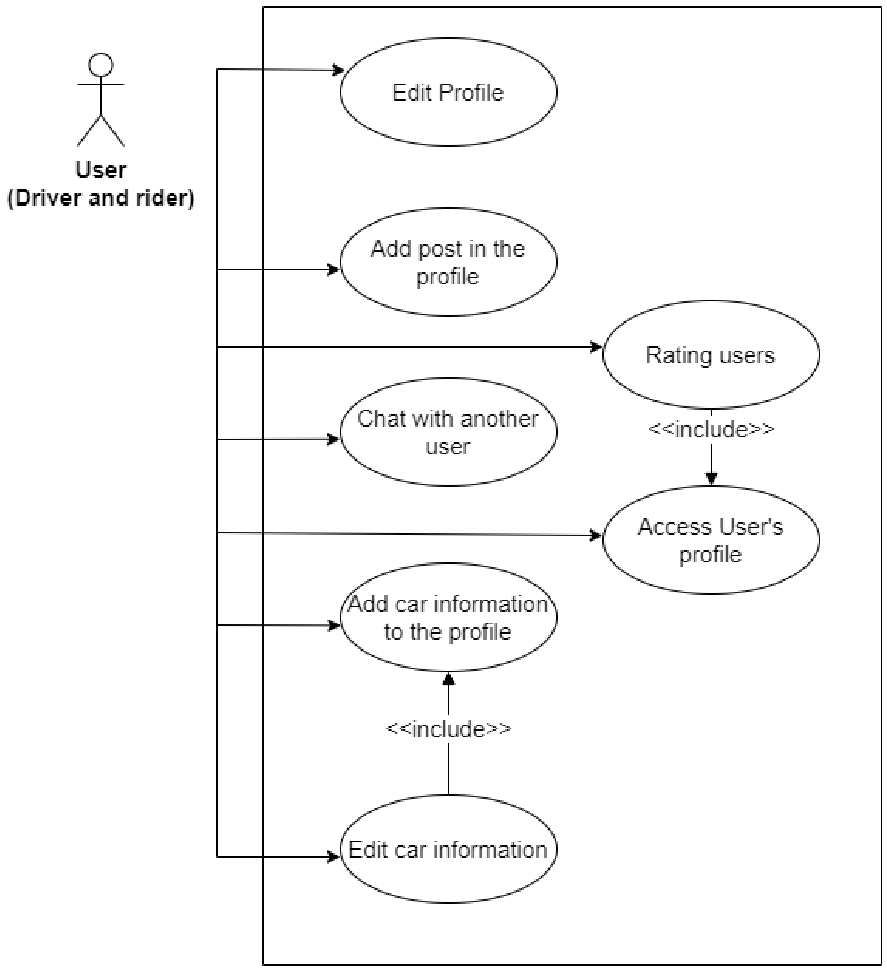
- Updating user’s profile information use case:

Use case name:	Update user’s profile information
Description:	Editing the account information of specific user.
Actor:	Admin
Goals:	Changing some information about specific user
Pre-Conditions:	Admin is already logged in to the application and took the permission from the user to updates his/her information of the account of the application.
Post-Conditions:	The account has been Edited by the admin to the user.
Basic flow:	After logging in to the application by the admin enters the users page choose the user that needed to update his/her information then press on edit button that allows the admin to change information about the user.
Alternate flow:	--------

- Blocking user use case:

Use case name:	Block user
Description:	Making a specific user not allowed to use the application.
Actor:	Admin
Goals:	Blocking a specific user.
Pre-Conditions:	Admin is already logged in to the application.
Post-Conditions:	The User is blocked or on hold from using the application
Basic flow:	After logging in to the application by the admin enters the users page choose the user that needed to be blocked and press on the block button which makes the user unable to use the application.
Alternate flow:	--------

- Common use cases between the two types of users (Rider and driver)



- Editing profile use case:

Use case name:	Edit profile
Description:	Changing some information about the user.
Actor:	User (Drive/Rider)
Goals:	Editing some information about the user
Pre-Conditions:	User is already logged in to the application.
Post-Conditions:	Changing some information about the user’s profile
Basic flow:	Opening the application then enter the profile page and then press on edit button then change the user’s information.
Alternate flow:	--------

- Adding post use case:

Use case name:	Add post to user’s profile
Description:	Adding posts to the user’s profile
Actor:	User (Drive/Rider)
Goals:	Adding posts to the user’s profile
Pre-Conditions:	User is already logged in to the application.
Post-Conditions:	A new post has been added to the user’s profile
Basic flow:	Opening the application then enter the profile page and then press on add post button then enter the text or the picture or picture with text to user’s timeline.
Alternate flow:	--------

- Chatting use case:

Use case name:	Chat with another user
Description:	Sending and receiving messages from another user
Actor:	User (Drive/Rider)

Goals:	Communication between two users
Pre-Conditions:	User is already logged in to the application.
Post-Conditions:	A message sent/received
Basic flow:	Opening the application then press on the chat button so all the chats is appeared in front of the user so once he entered the chat with the other user he can chat with him.
Alternate flow:	Opening the application then press on users page so all the users appeared in front of the user if he wants to chat with any of them so the user need to press on the user he need then press on message button which opens the chat with him.

- Adding car use case:

Use case name:	Add car information to the profile
Description:	Adding the car information to the user's profile (Car model, car licence ,etc…)
Actor:	User (Drive/Rider)
Goals:	Add the car information
Pre-Conditions:	User is already logged in to the application.
Post-Conditions:	The car information has been added to the profile
Basic flow:	Opening the application then enter the car page the fill the fields with the car details.
Alternate flow:	--------

- Editing car information use case:

Use case name:	Edit car information
Description:	Changing some information about the car of the owner
Actor:	User (Rider)
Goals:	Editing the car information
Pre-Conditions:	User is already logged in to the application and added a car to his profile.
Post-Conditions:	The car information is edited successfully.
Basic flow:	Opening the application then enter the car page then the user modify the car information that the user want to change.

Alternate flow:	--------

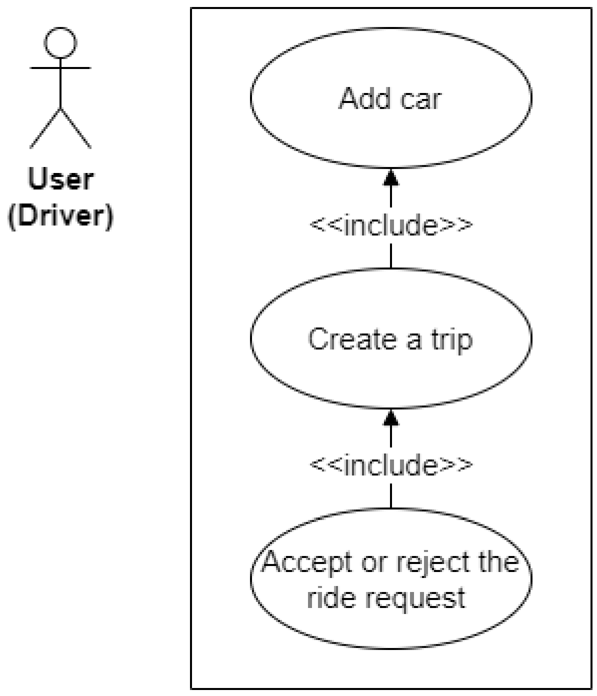
- Accessing profile use case:

Use case name:	Access user’s profile
Description:	Entering another user profile to see his rate and posts to know who is that person you want to take the ride with.
Actor:	User (Drive/Rider)
Goals:	Enter the user’s profile and give a rate about according to the ride with him.
Pre-Conditions:	User is already logged in to the application and accessed the user’s profile.
Post-Conditions:	Gives the other user rate.
Basic flow:	Opening the application then enter the users page then press on the rate button then choose the appropriate rate for this user.
Alternate flow:	--------

- Rating users use case:

Use case name:	Rating users
Description:	It is giving a rate to the user depends on the ride and the experience with him/her.
Actor:	User (Drive/Rider)
Goals:	To know if that person is a suitable to take a ride with or not.
Pre-Conditions:	User is already logged in to the application and entered or accessed the other users profile.
Post-Conditions:	The user has given a rate which is calculated in average.
Basic flow:	Opening the application then enter the profile page and then press on add post button then enter the text or the picture or picture with text to user’s timeline.
Alternate flow:	--------

- Use cases for the User (Driver):



- Adding the car has been discussed in the use case before this.
- Creating trip use case:

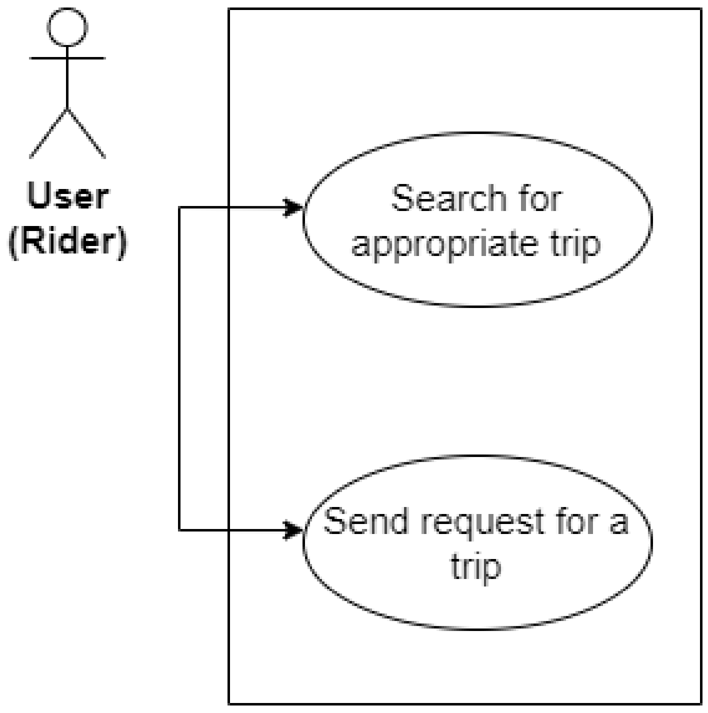
Use case name:	Create a trip
Description:	Making a trip with a source and destination.
Actor:	User (Driver)
Goals:	Making a trip with source and destination to connect the rider and the driver together
Pre-Conditions:	User is already logged in to the application and already added a car details.
Post-Conditions:	The trip is added to the list of trips with the source and destination.
Basic flow:	Opening the application then press on the plus sign then add the destination of the trip.
Alternate flow:	--------

- Accept or reject the ride request use cases:

Use case name:	Accept or reject the trip request.
Description:	Accepting or rejecting the trip request to take the user with him on the trip or not.
Actor:	User (Driver)

Goals:	Connecting the users with each other.
Pre-Conditions:	User is already logged in to the application and created a trip.
Post-Conditions:	The trip request will be rejected or accepted.
Basic flow:	Opening the application and creating the trip another user will send a request to join the trip with the driver and the driver can accept or reject this request.
Alternate flow:	--------

- Rider use cases:



- Search for appropriate trip use case:

Use case name:	Search for appropriate trip
Description:	Searching in the trips list for appropriate trip to the rider's source and destination.
Actor:	User (Rider)
Goals:	Connecting the user with the suitable trip for his/her destination.
Pre-Conditions:	User is already logged in to the application.
Post-Conditions:	The user found his suitable trip and should send a request.
Basic flow:	Opening the application then enter the trips page that displays al the trips in the application or specially the available trips till he found a suitable one.

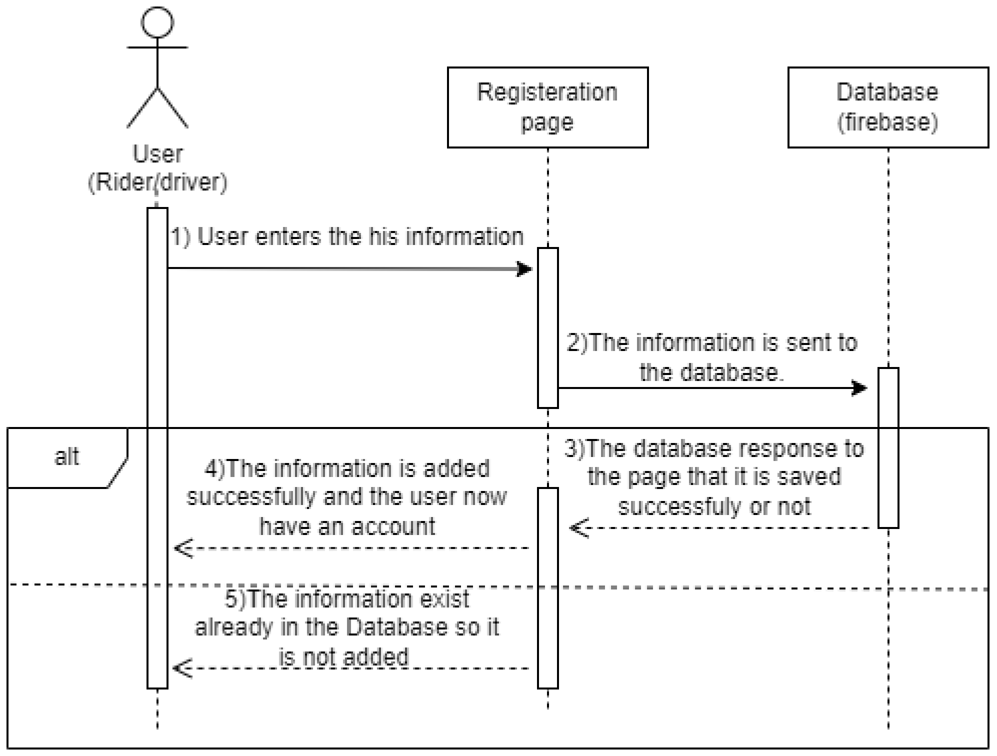
Alternate flow:	--------

- Send request for a trip use case:

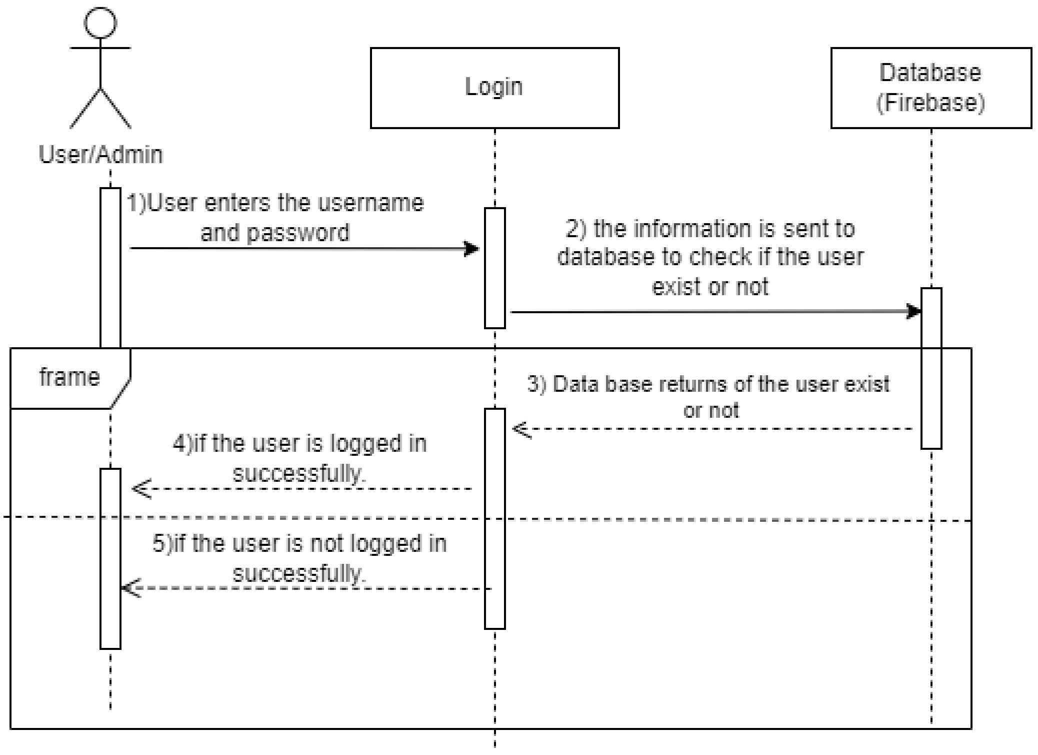
Use case name:	Send request for a trip
Description:	Sending the trip request to the driver
Actor:	User (Rider)
Goals:	Connecting to the driver by sending the trip request
Pre-Conditions:	User is already logged in to the application and found an appropriate trip to the rider's source and destination.
Post-Conditions:	The trip request is sent to the trip maker.
Basic flow:	Opening the application then choose the appropriate trip then send a request to the trip maker.
Alternate flow:	--------

3.3 Sequence diagram:

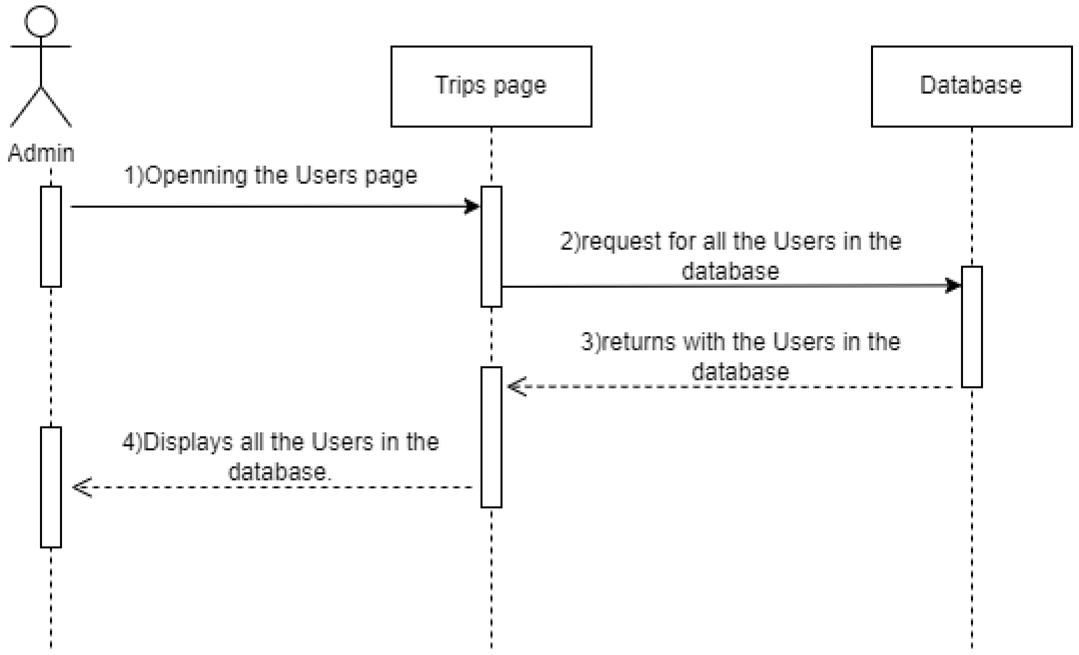
- Registration sequence:


User (Rider/driver)
Registeration page
Database (firebase)
1) User enters the his information
2)The information is sent to the database.
alt
3)The database response to the page that it is saved successfuly or not
4)The information is added successfully and the user now have an account
5)The information exist already in the Database so it is not added

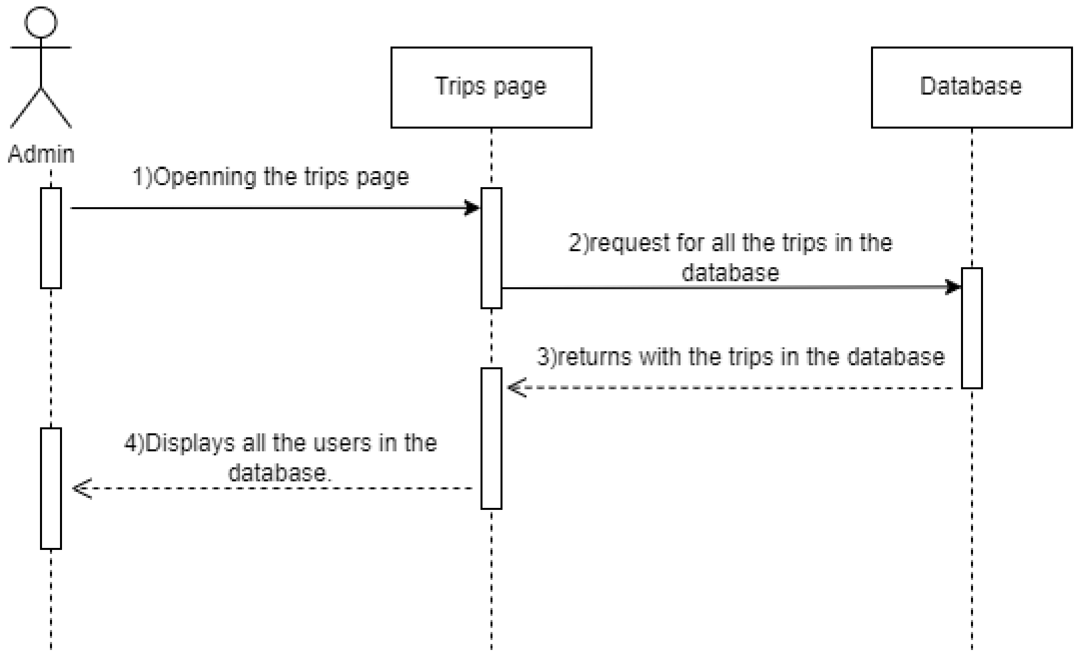
- Log in sequence:


User/Admin
Login
Database (Firebase)
1)User enters the username and password
2) the information is sent to database to check if the user exist or not
frame
3) Data base returns of the user exist or not
4)if the user is logged in successfully.
5)if the user is not logged in successfully.

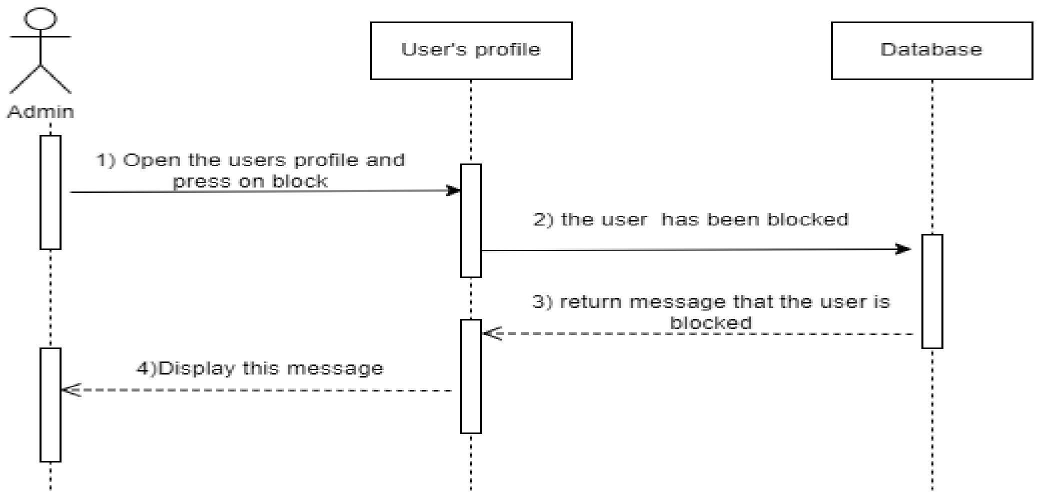
- Display all users for Admin:


Admin
Trips page
Database
1)Openning the Users page
2)request for all the Users in the database
3)returns with the Users in the database
4)Displays all the Users in the database.

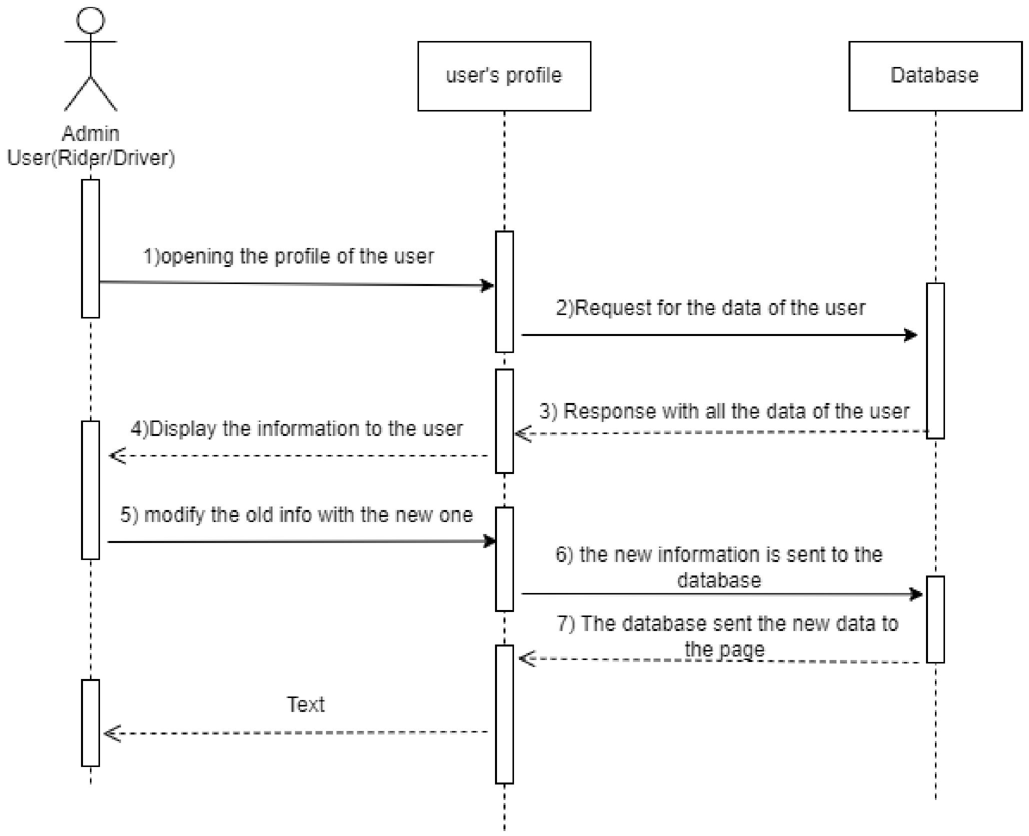
- Display all trips to the admin:


Admin
Trips page
Database
1)Openning the trips page
2)request for all the trips in the database
3)returns with the trips in the database
4)Displays all the users in the database.

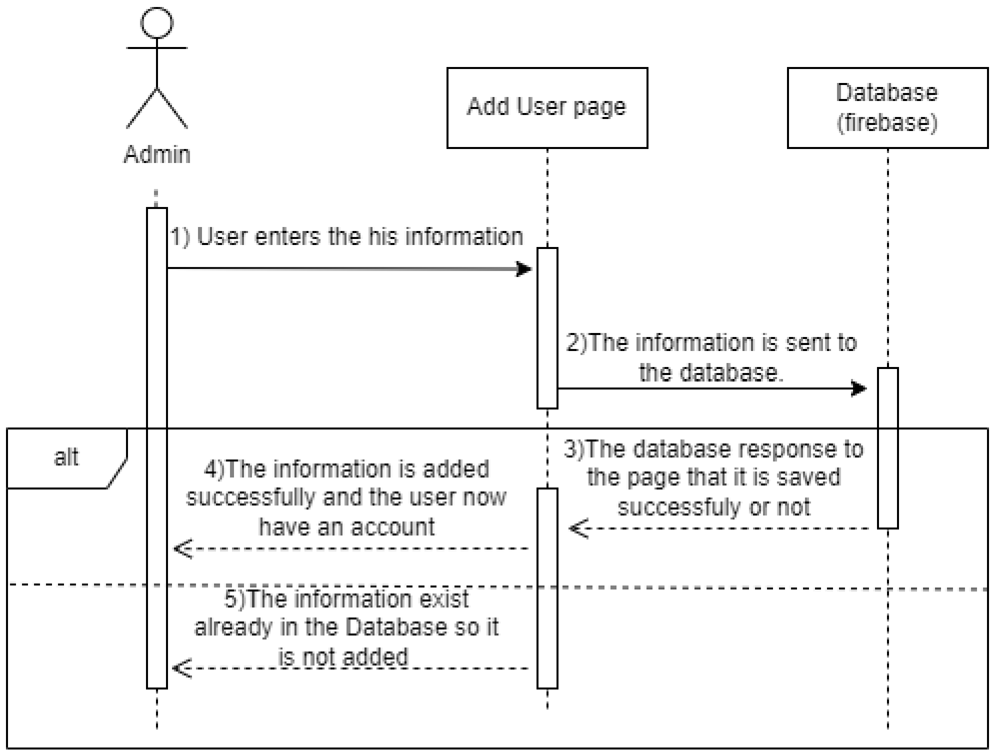
- Block users:


Admin
User's profile
Database
1) Open the users profile and press on block
2) the user has been blocked
3) return message that the user is blocked
4)Display this message

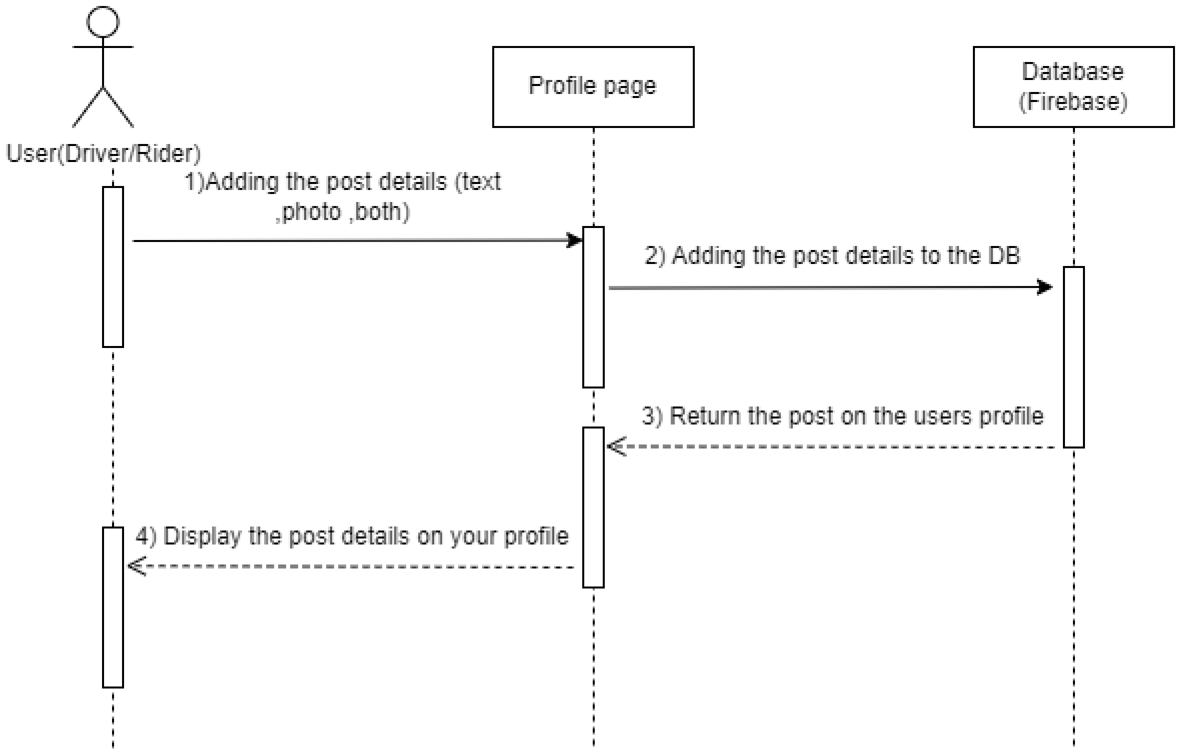
- Edit user's profile information:


Admin
User(Rider/Driver)
user's profile
Database
1)opening the profile of the user
2)Request for the data of the user
3) Response with all the data of the user
4)Display the information to the user
5) modify the old info with the new one
6) the new information is sent to the database
7) The database sent the new data to the page
Text

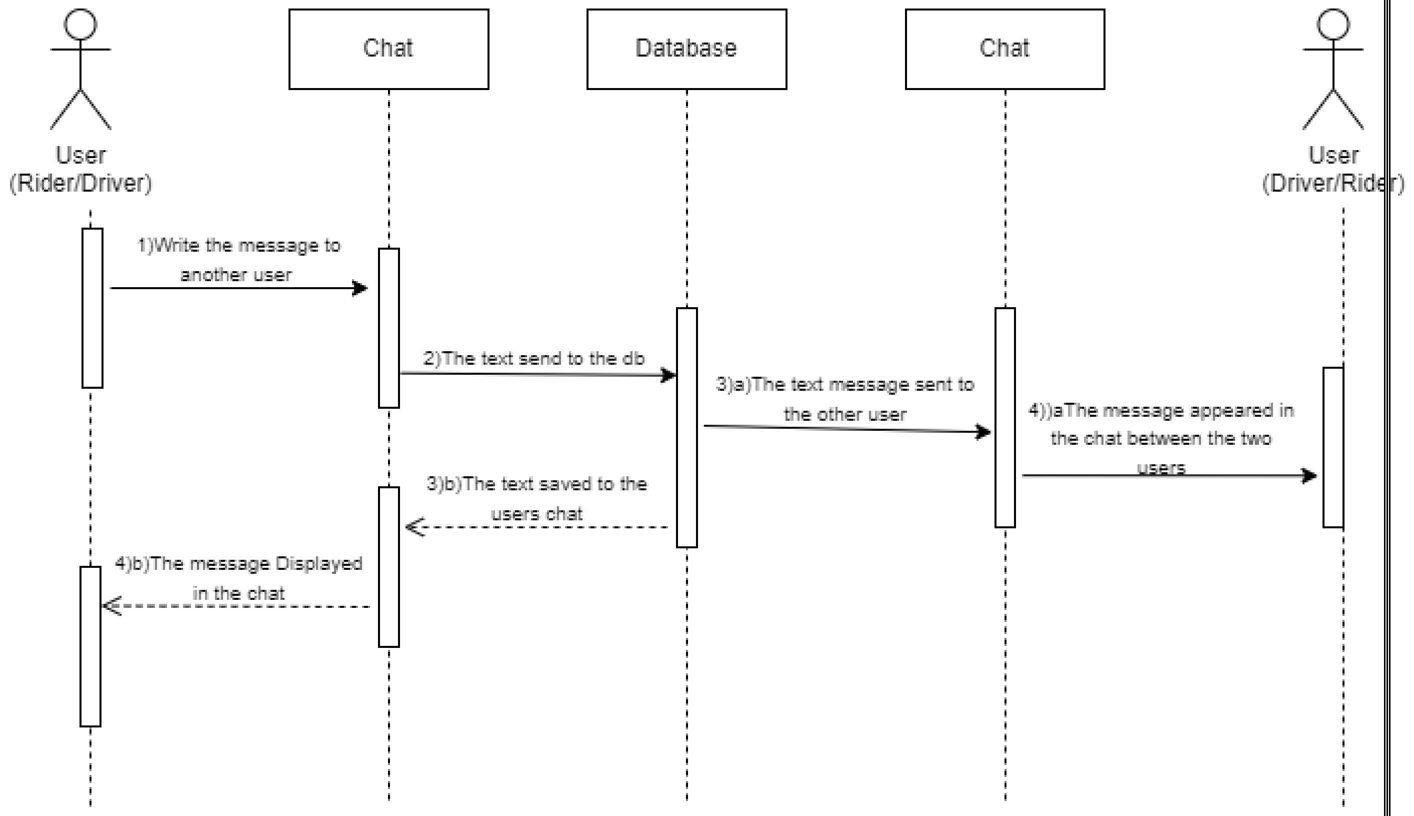
- Adding user:


Admin
Add User page
Database (firebase)
1) User enters the his information
2)The information is sent to the database.
alt
3)The database response to the page that it is saved successfuly or not
4)The information is added successfully and the user now have an account
5)The information exist already in the Database so it is not added

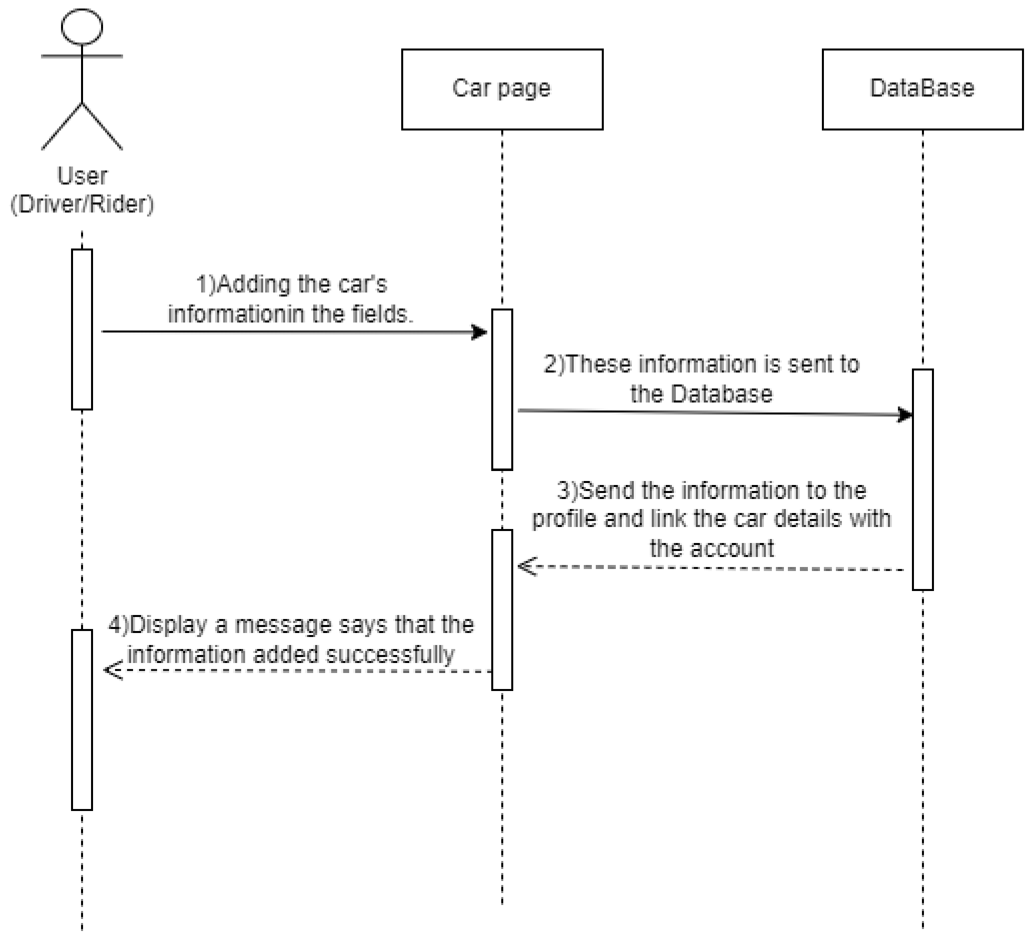
- Add post to the profile:


User(Driver/Rider)
Profile page
Database (Firebase)
1)Adding the post details (text ,photo ,both)
2) Adding the post details to the DB
3) Return the post on the users profile
4) Display the post details on your profile

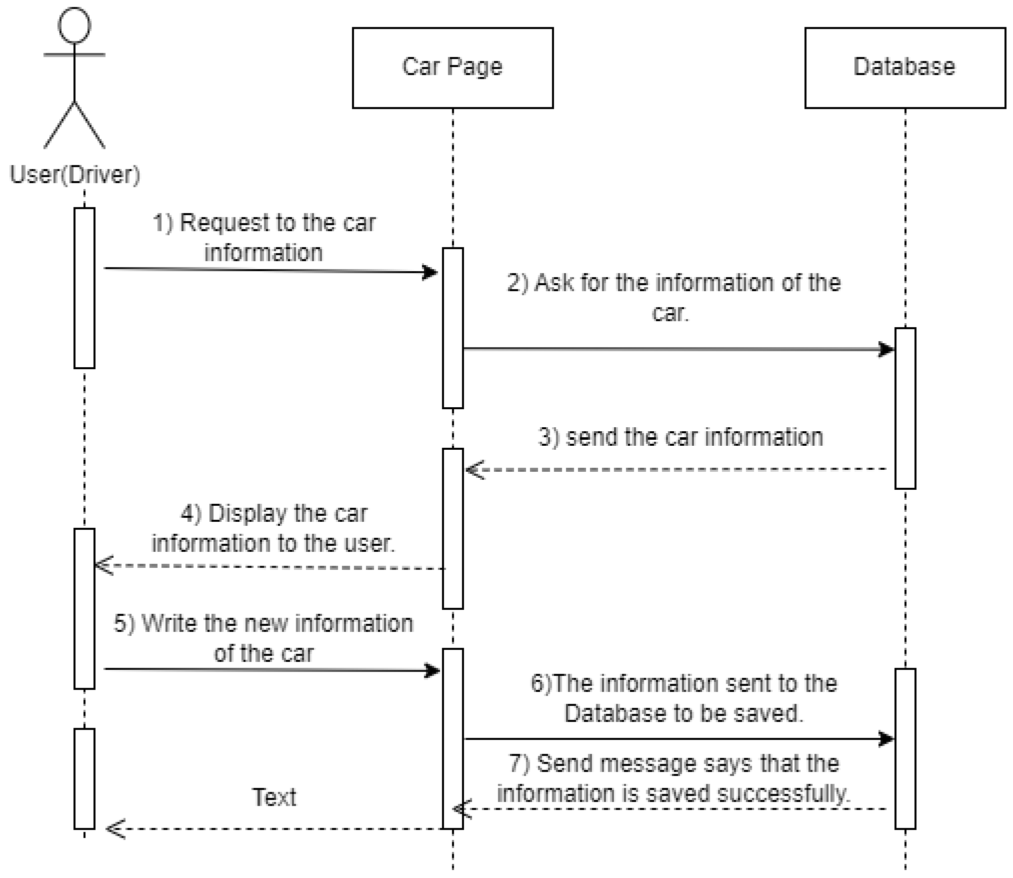
- Chatting:


User (Rider/Driver)
Chat
Database
Chat
User (Driver/Rider)
1)Write the message to another user
2)The text send to the db
3)a)The text message sent to the other user
4))aThe message appeared in the chat between the two users
3)b)The text saved to the users chat
4)b)The message Displayed in the chat

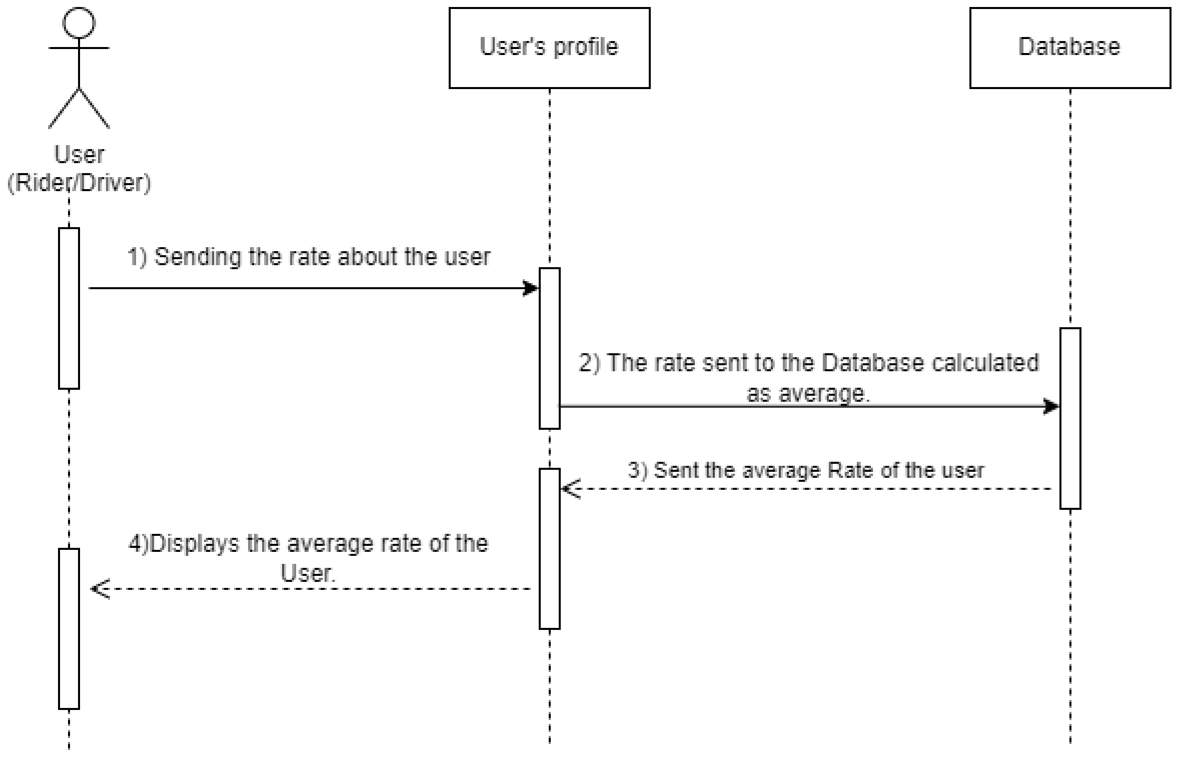
- Add Car Information:


User (Driver/Rider)
Car page
DataBase
1)Adding the car's informationin the fields.
2)These information is sent to the Database
3)Send the information to the profile and link the car details with the account
4)Display a message says that the information added successfully

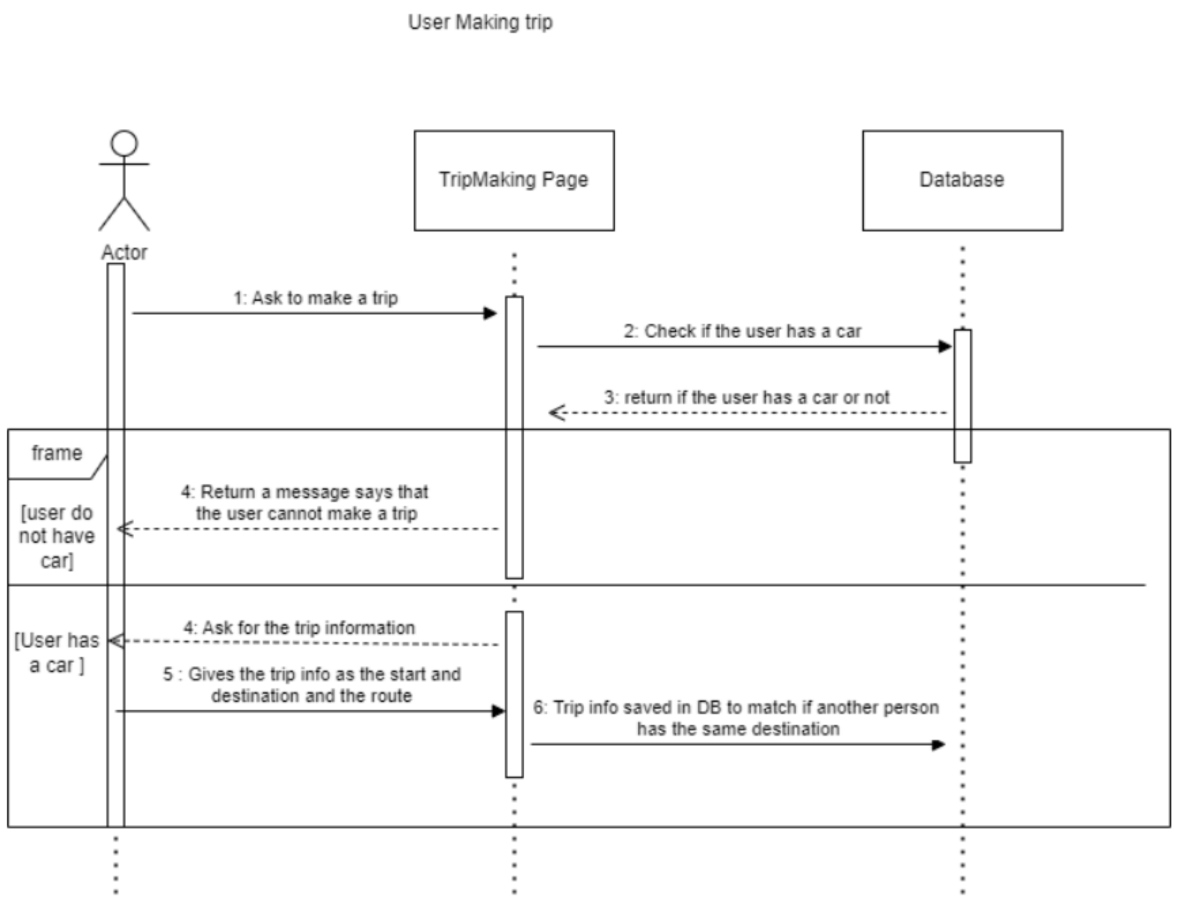
- Edit car information:


User(Driver)
Car Page
Database
1) Request to the car information
2) Ask for the information of the car.
3) send the car information
4) Display the car information to the user.
5) Write the new information of the car
6)The information sent to the Database to be saved.
7) Send message says that the information is saved successfully.
Text

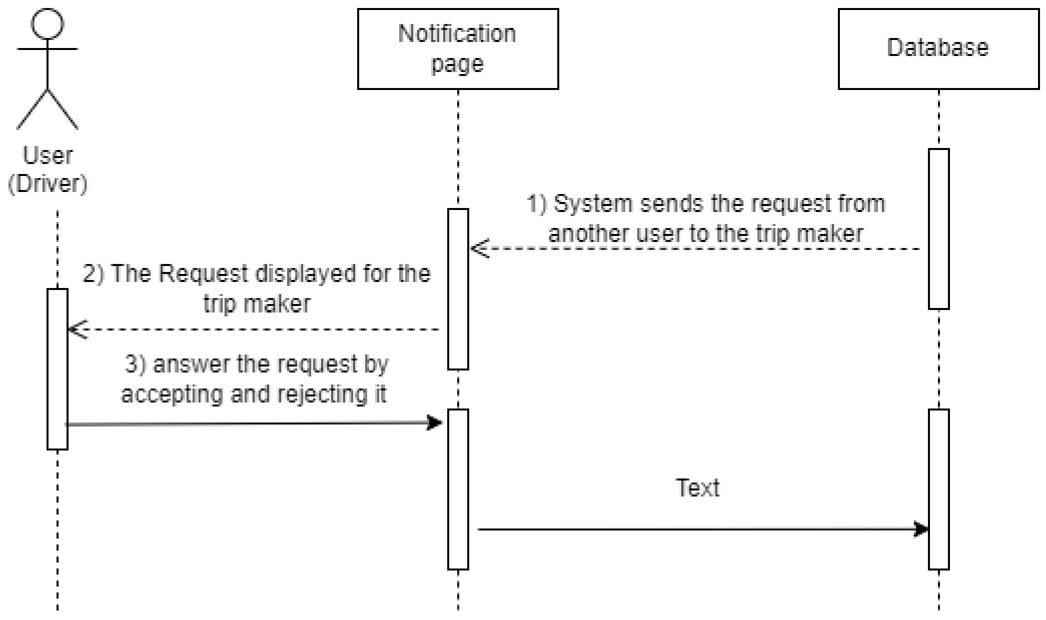
- Rating user:


User (Rider/Driver)
User's profile
Database
1) Sending the rate about the user
2) The rate sent to the Database calculated as average.
3) Sent the average Rate of the user
4)Displays the average rate of the User.

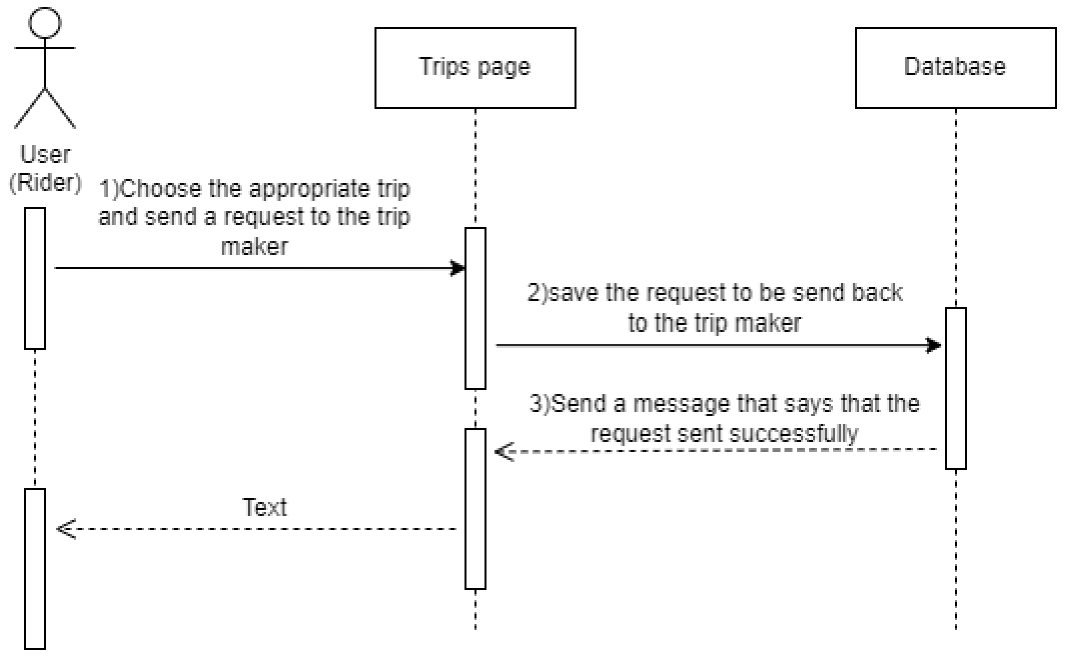
- Creating a trip:


User Making trip
Actor
TripMaking Page
Database
1: Ask to make a trip
2: Check if the user has a car
3: return if the user has a car or not
frame
[user do not have car]
4: Return a message says that the user cannot make a trip
[User has a car]
4: Ask for the trip information
5 : Gives the trip info as the start and destination and the route
6: Trip info saved in DB to match if another person has the same destination

- Accepting or rejecting the request:

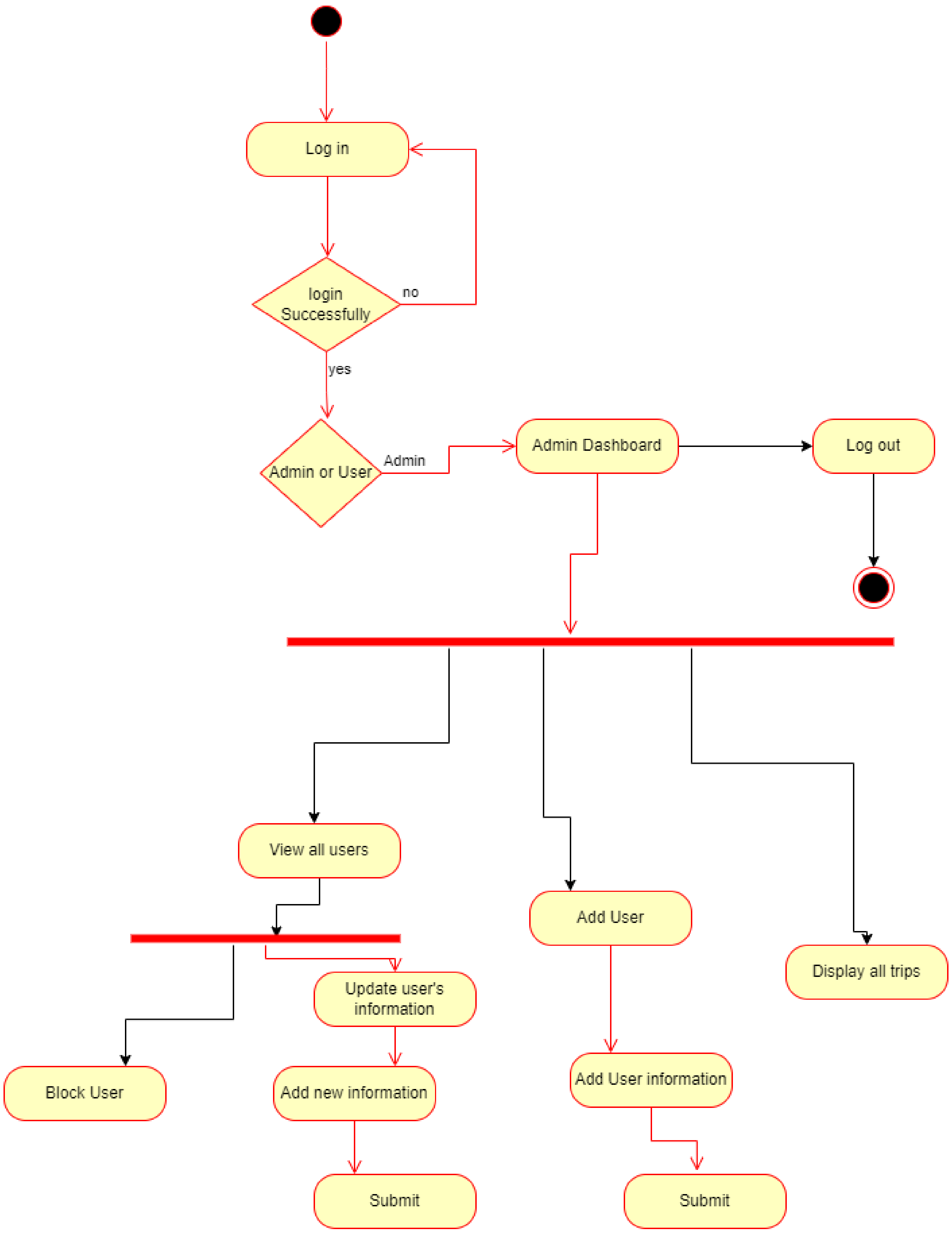

User (Driver)
Notification page
Database
1) System sends the request from another user to the trip maker
2) The Request displayed for the trip maker
3) answer the request by accepting and rejecting it
Text

- Sending the trip request:

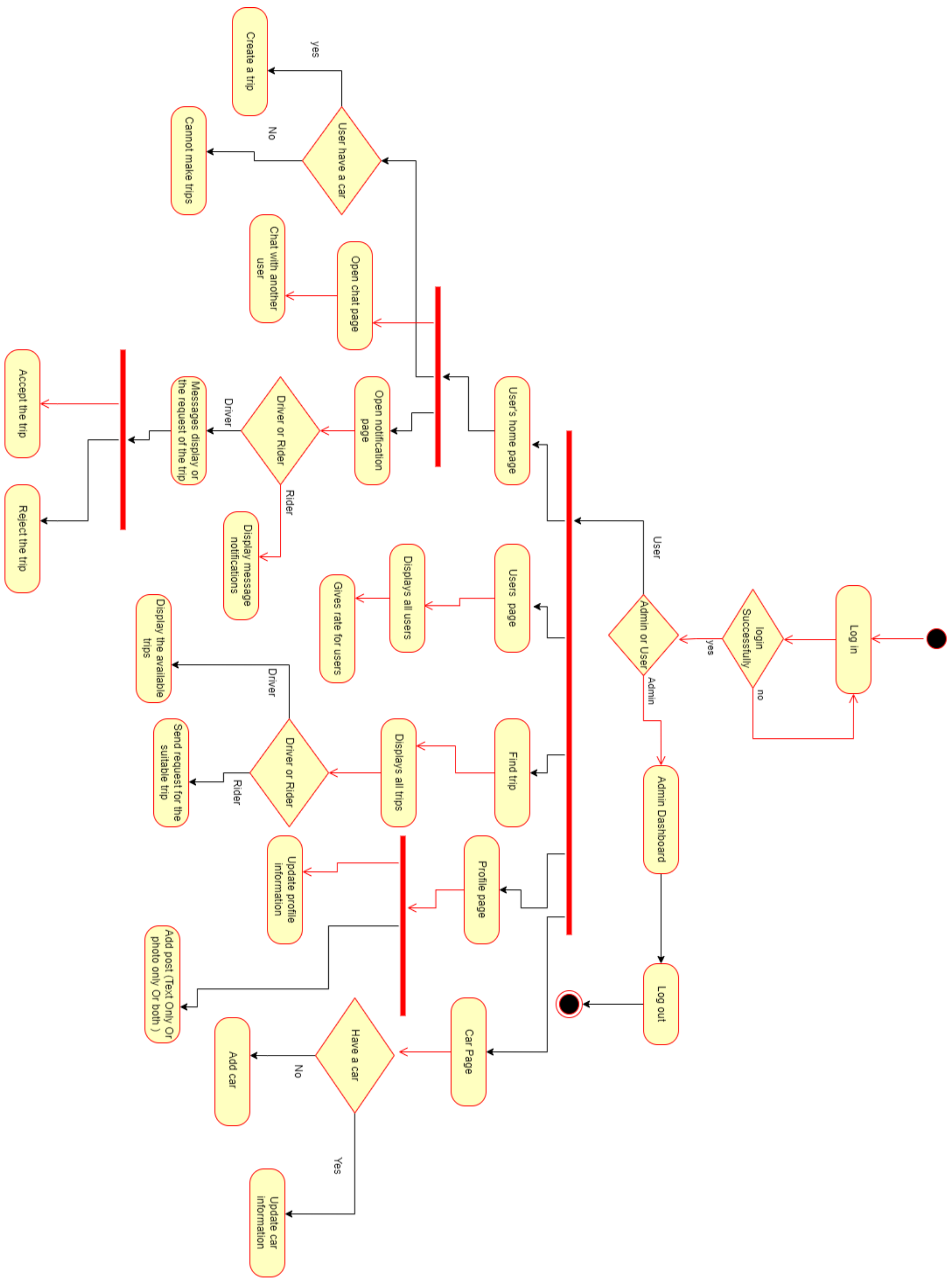

User (Rider)
Trips page
Database
1)Choose the appropriate trip and send a request to the trip maker
2)save the request to be send back to the trip maker
3)Send a message that says that the request sent successfully
Text

3.4 Activity Diagram:

Admin Activity Diagram:



- User Activity Diagram:

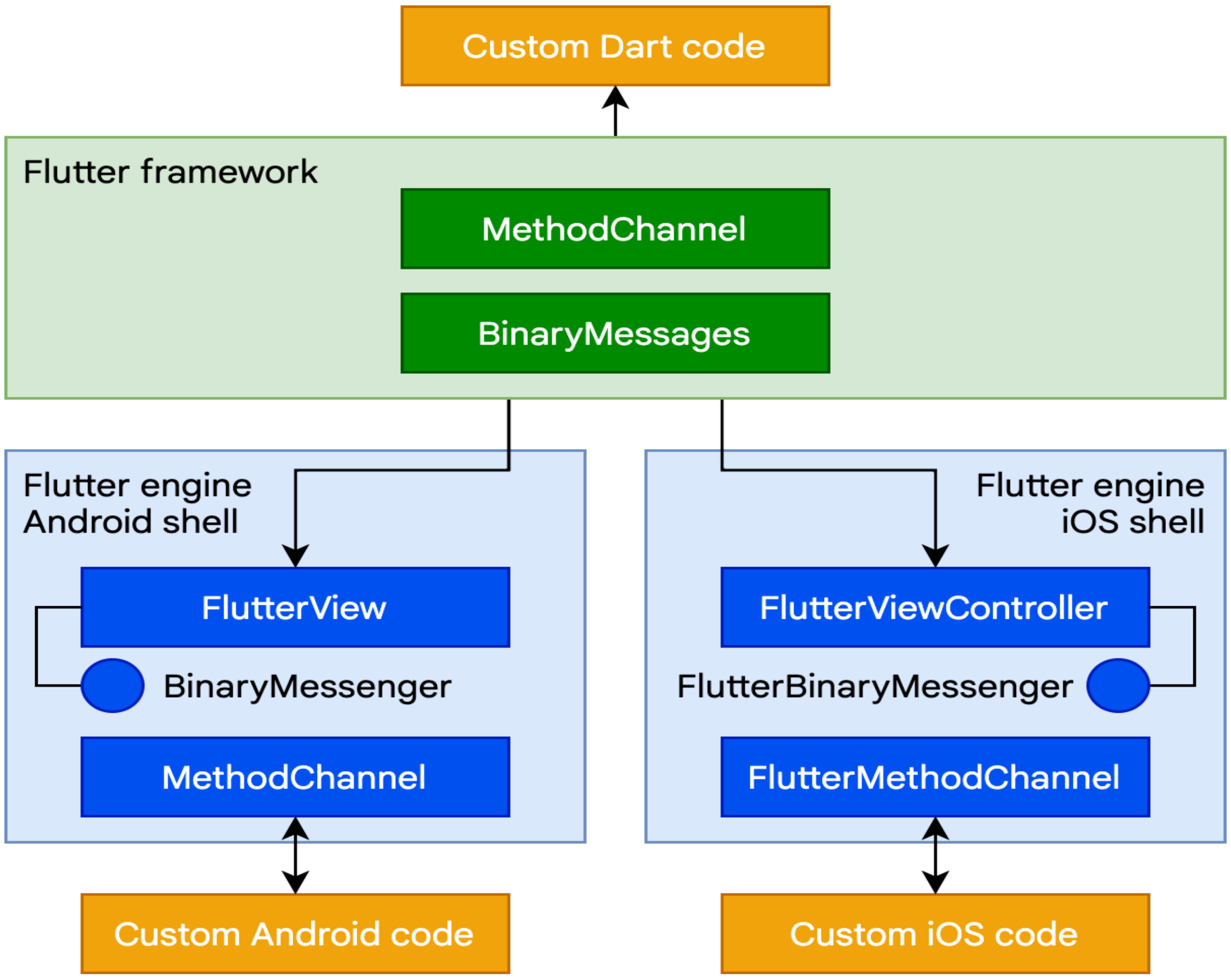


Chapter 4: Implementation

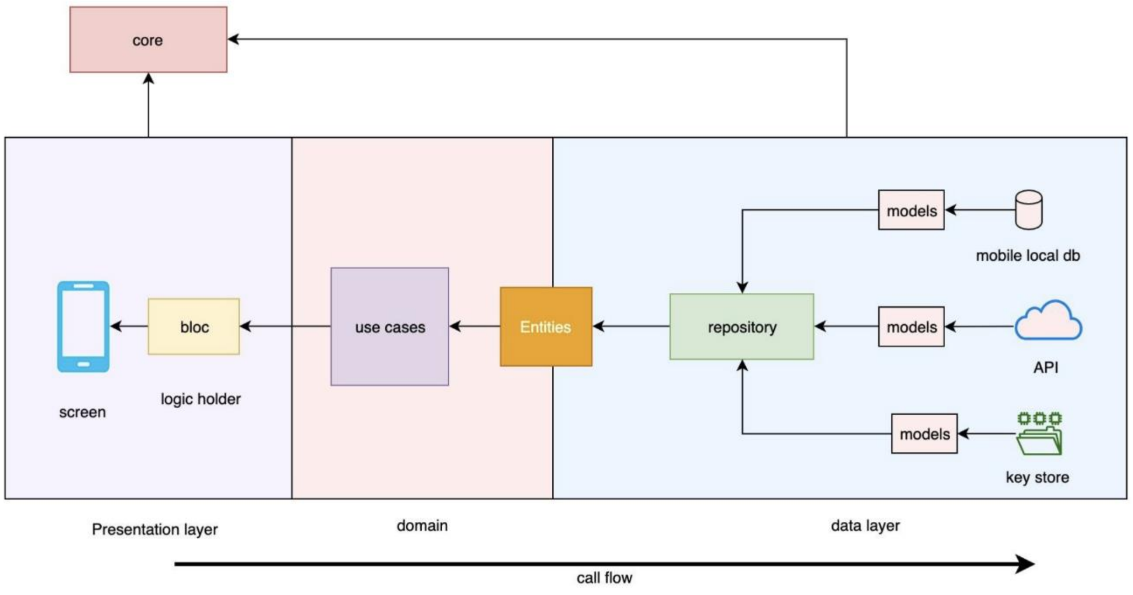
4.1 Software Architecture

Flutter is designed as an extensible, layered system. It exists as a series of independent libraries that each depend on the underlying layer. No layer has privileged access to the layer below, and every part of the framework level is designed to be optional and replaceable.

For mobile and desktop apps, Flutter allows you to call into custom code through a *platform channel*, which is a mechanism for communicating between your Dart code and the platform-specific code of your host app. By creating a common channel (encapsulating a name and a codec), you can send and receive messages between Dart and a platform component written in a language like Kotlin or Swift. Data is serialized from a Dart type like Map into a standard format, and then deserialized into an equivalent representation in Kotlin (such as HashMap) or Swift (such as Dictionary).



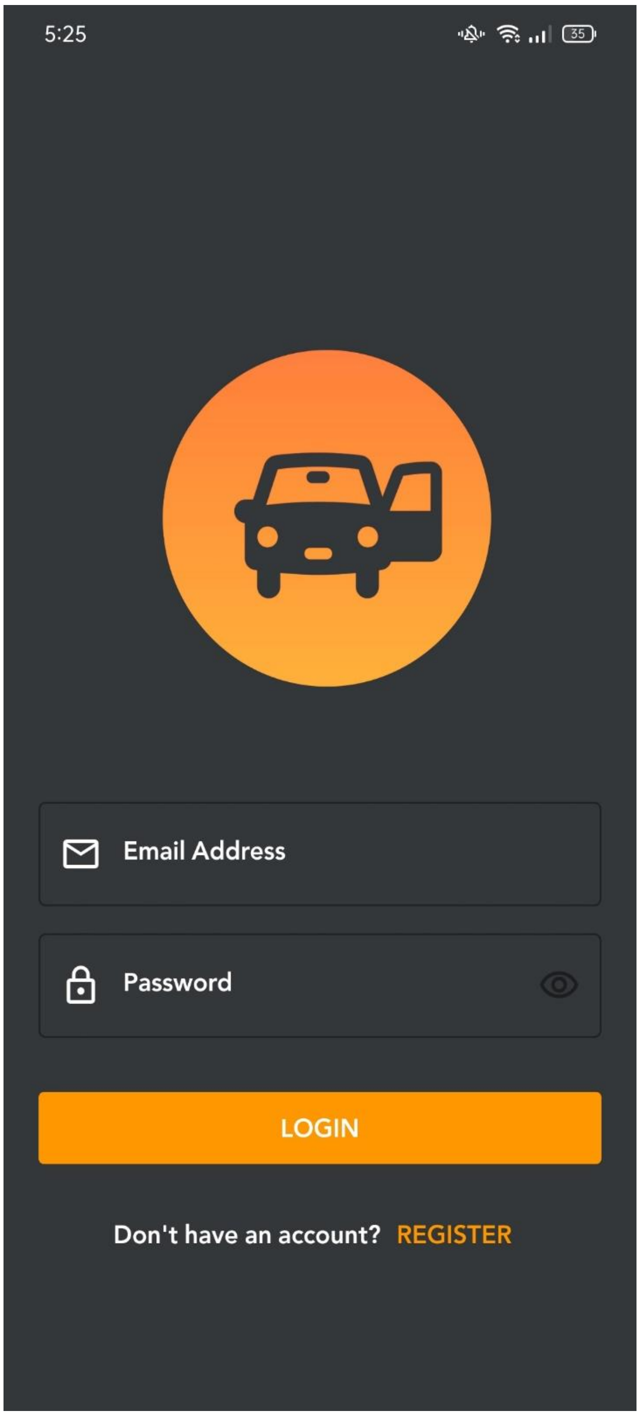
The integration of other codes and channeling

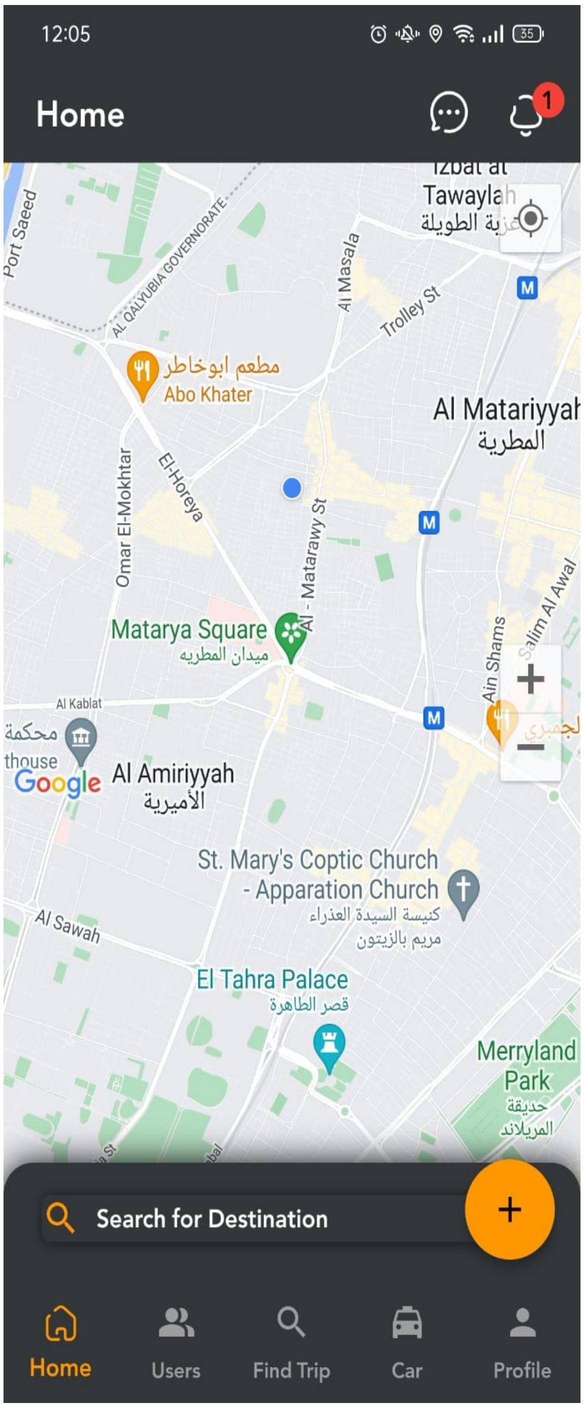


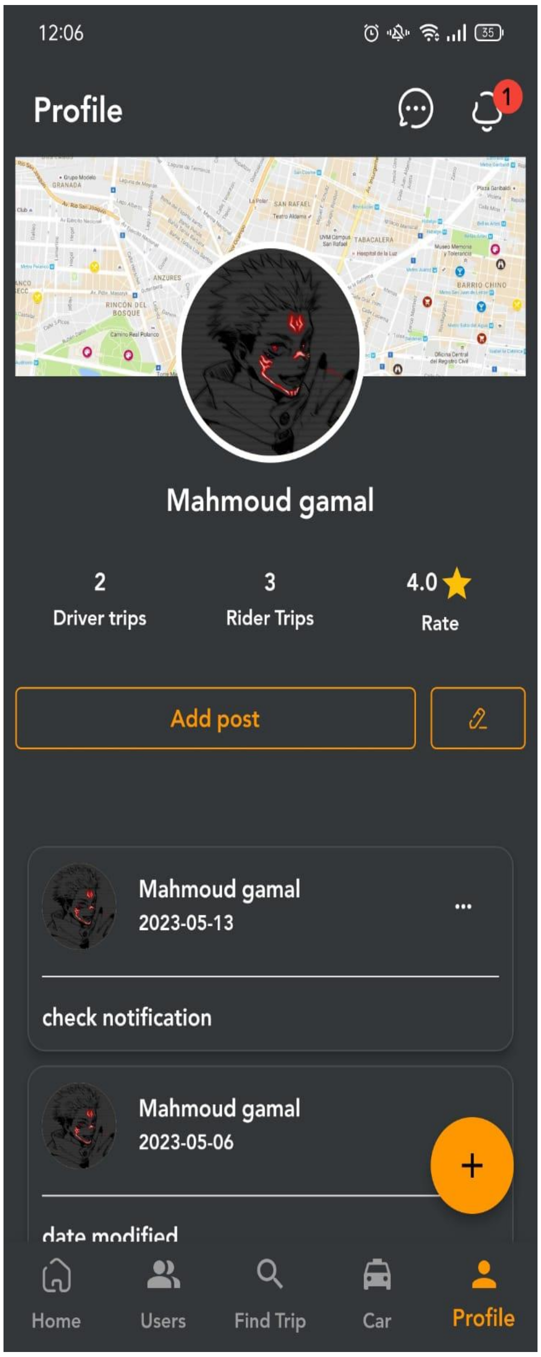
The software architecture and repository

4.2 Main Functionality

- Screens from the app







12:06

Profile

Mahmoud gamal

2 Driver trips

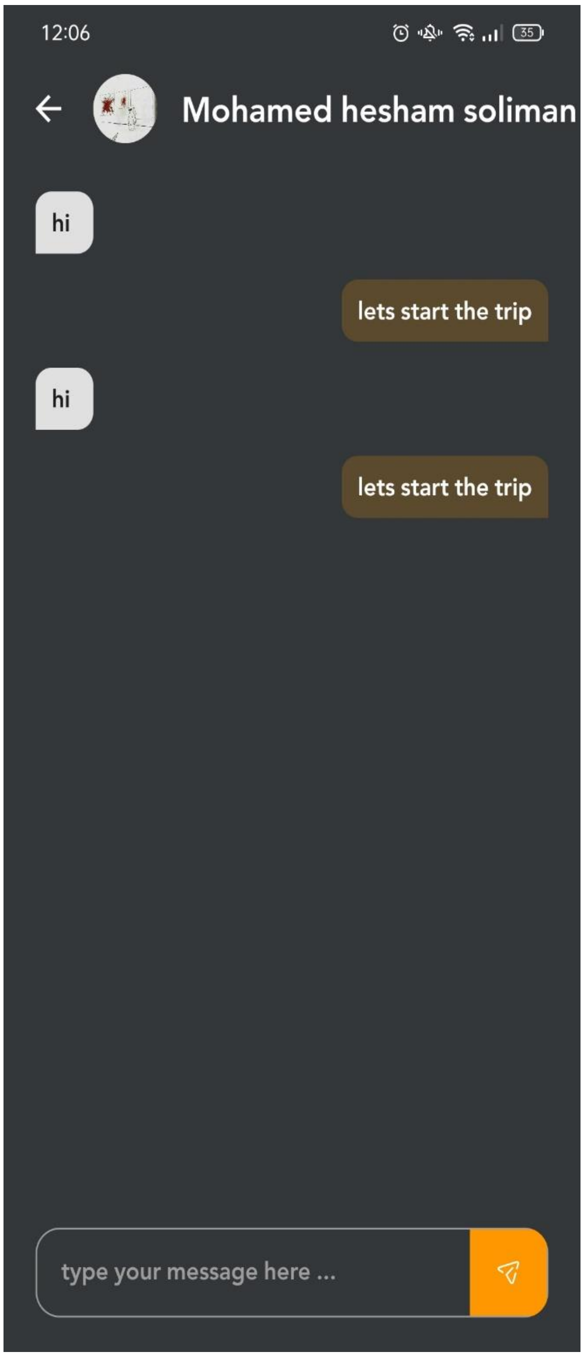
3 Rider Trips

4.0 Rate

Add post

Mahmoud gamal
2023-05-13
check notification

Mahmoud gamal
2023-05-06
date modified



12:06

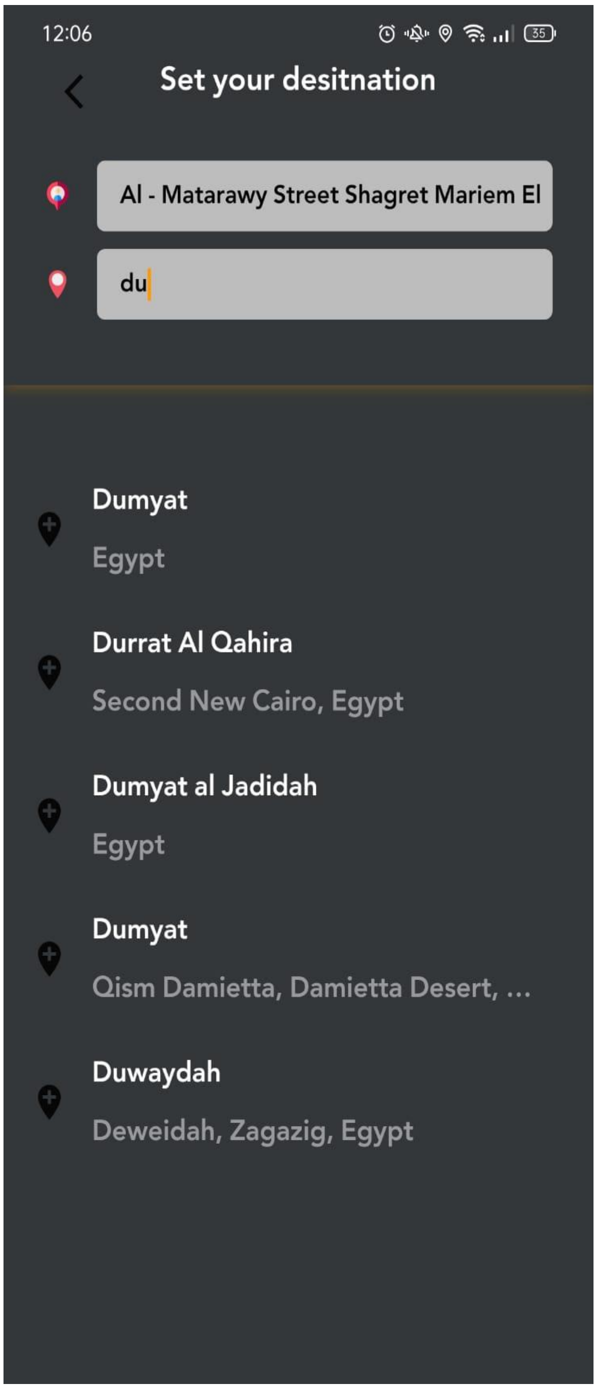
Mohamed hesham soliman

hi

lets start the trip

hi

lets start the trip



12:06

Set your desitnation

Al - Matarawy Street Shagret Mariem El

du

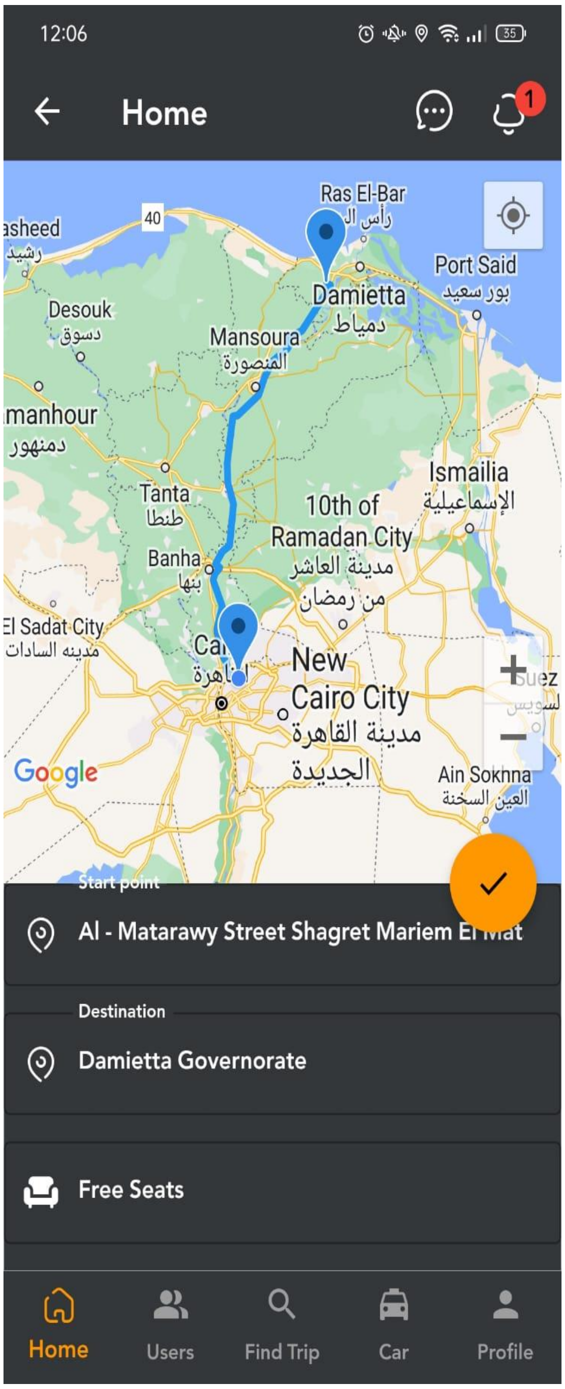
Dumyat
Egypt

Durrat Al Qahira
Second New Cairo, Egypt

Dumyat al Jadidah
Egypt

Dumyat
Qism Damietta, Damietta Desert, ...

Duwaydah
Deweidah, Zagazig, Egypt



12:06

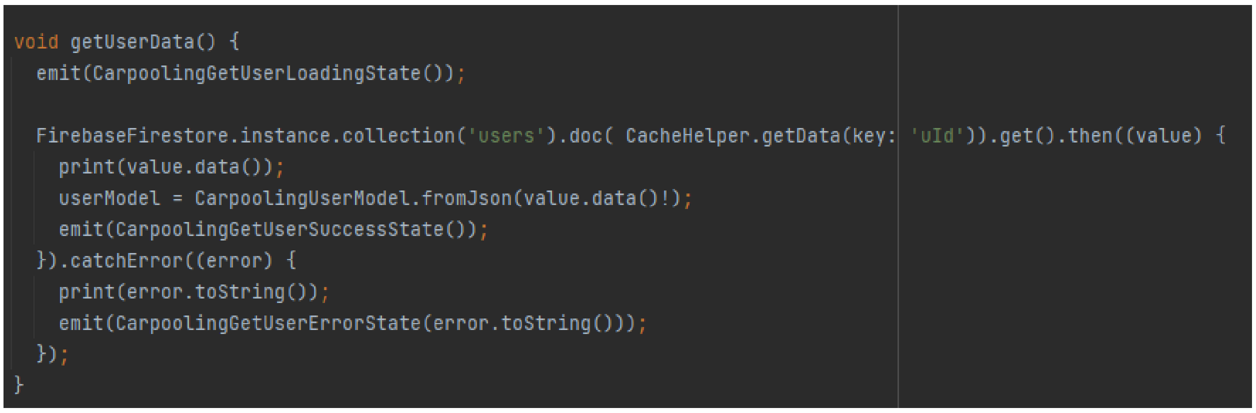
Home

Start point
Al - Matarawy Street Shagret Mariem El Mat

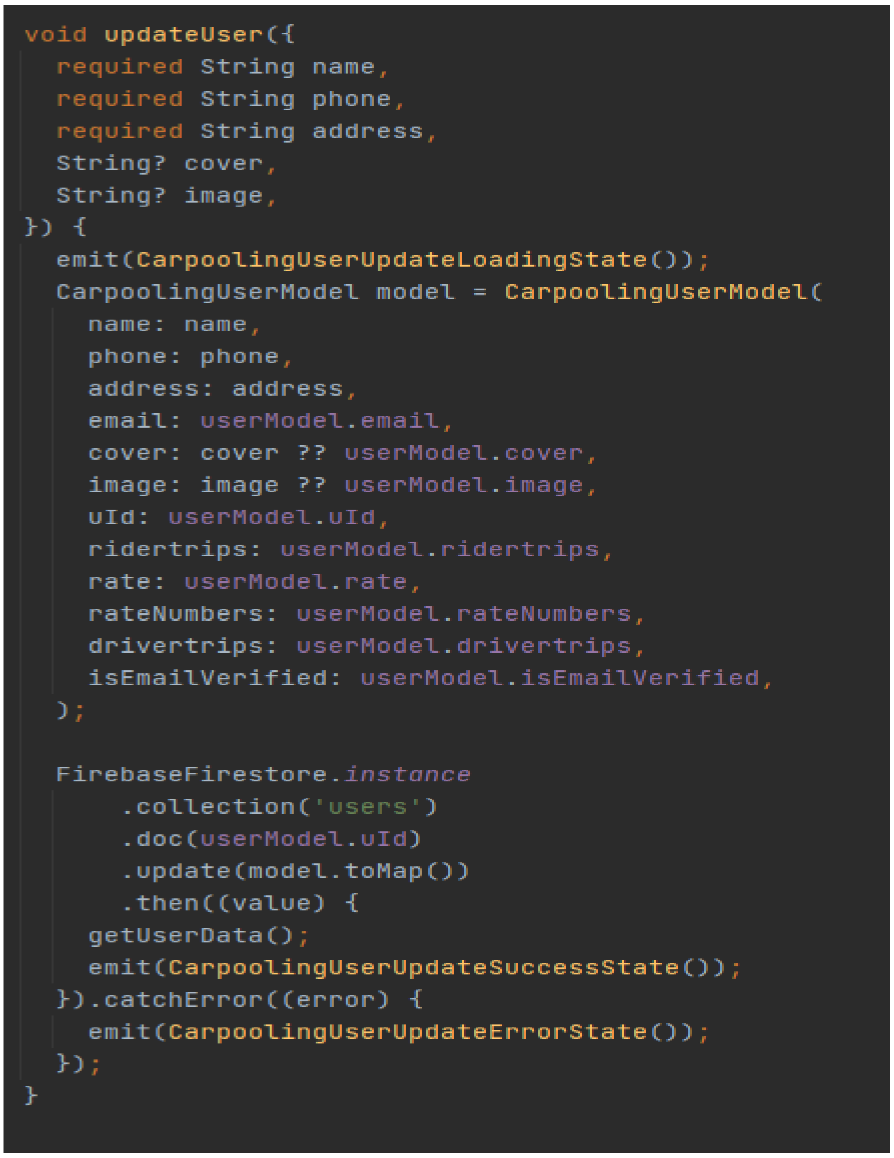
Destination
Damietta Governorate

Free Seats

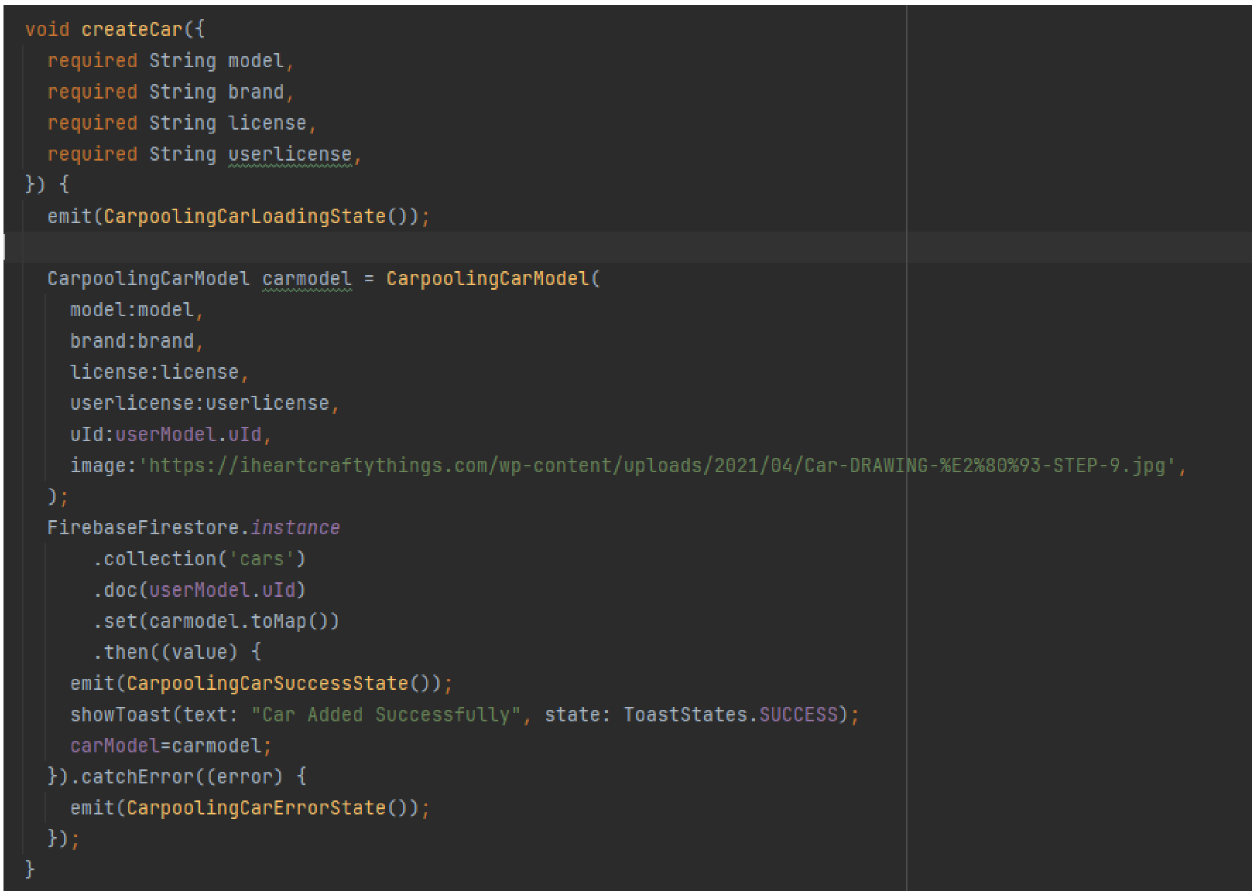
- Screens from the code



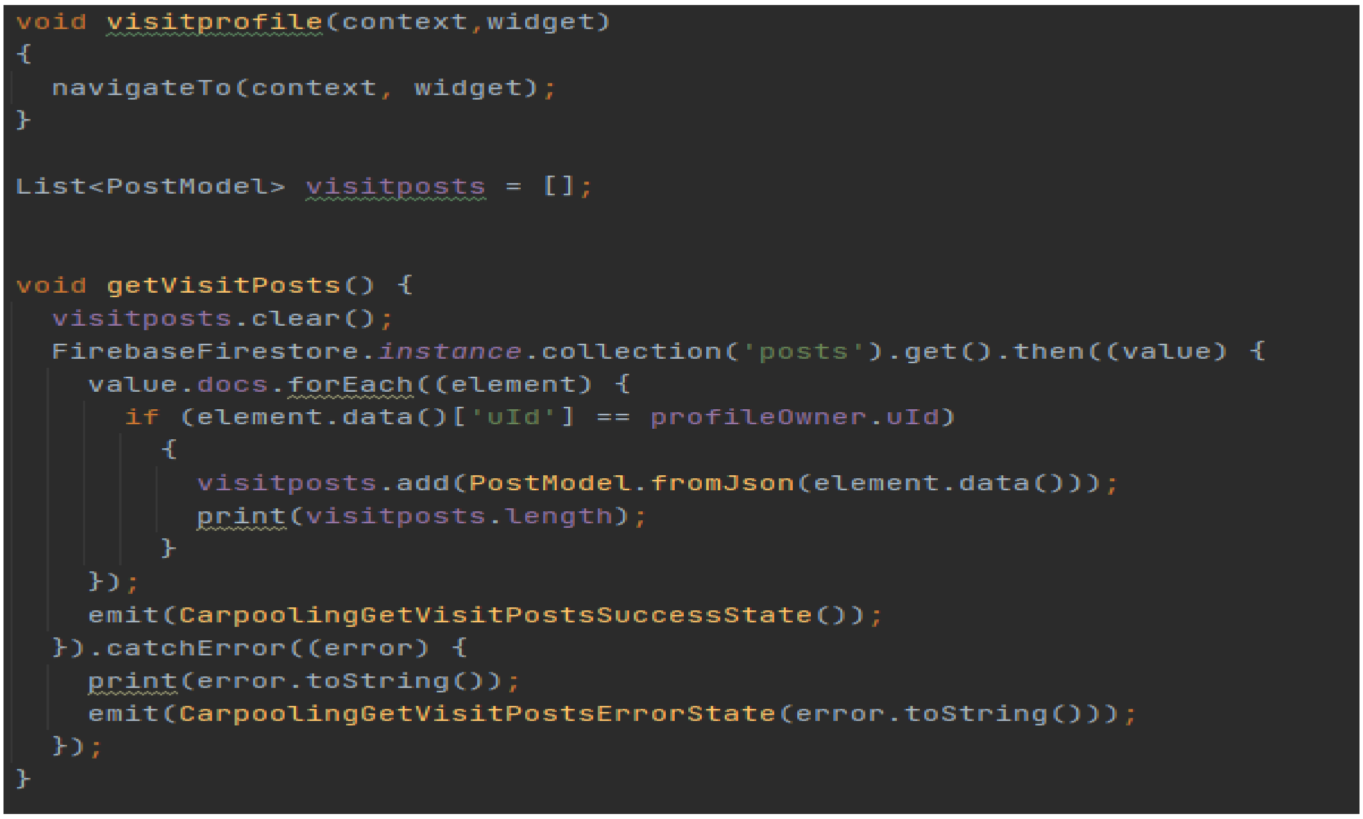
```
void getUserData() {
  emit(CarpoolingGetUserLoadingState());

  FirebaseFirestore.instance.collection('users').doc( CacheHelper.getData(key: 'uId')).get().then((value) {
    print(value.data());
    userModel = CarpoolingUserModel.fromJson(value.data()!);
    emit(CarpoolingGetUserSuccessState());
  }).catchError((error) {
    print(error.toString());
    emit(CarpoolingGetUserErrorState(error.toString()));
  });
}
```



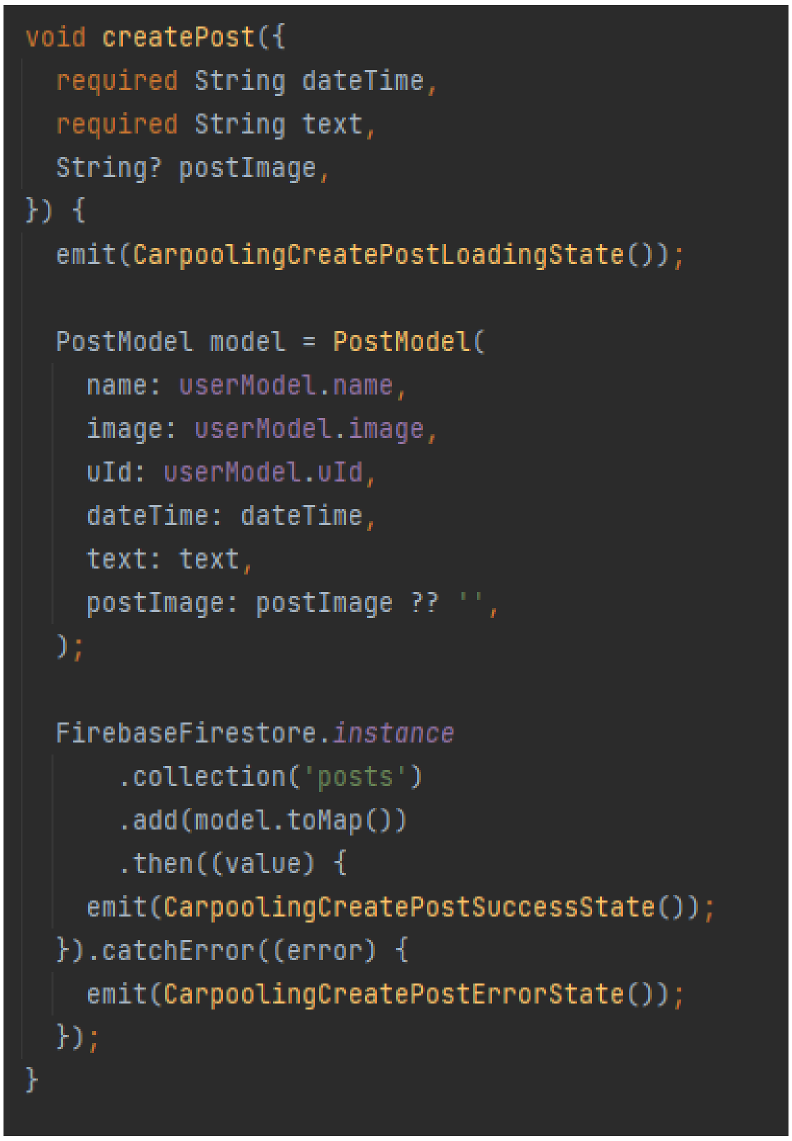
```
void updateUser({
  required String name,
  required String phone,
  required String address,
  String? cover,
  String? image,
}) {
  emit(CarpoolingUserUpdateLoadingState());
  CarpoolingUserModel model = CarpoolingUserModel(
    name: name,
    phone: phone,
    address: address,
    email: userModel.email,
    cover: cover ?? userModel.cover,
    image: image ?? userModel.image,
    uId: userModel.uId,
    ridertrips: userModel.ridertrips,
    rate: userModel.rate,
    rateNumbers: userModel.rateNumbers,
    drivertrips: userModel.drivertrips,
    isEmailVerified: userModel.isEmailVerified,
  );

  FirebaseFirestore.instance
      .collection('users')
      .doc(userModel.uId)
      .update(model.toMap())
      .then((value) {
    getUserData();
    emit(CarpoolingUserUpdateSuccessState());
  }).catchError((error) {
    emit(CarpoolingUserUpdateErrorState());
  });
}
```



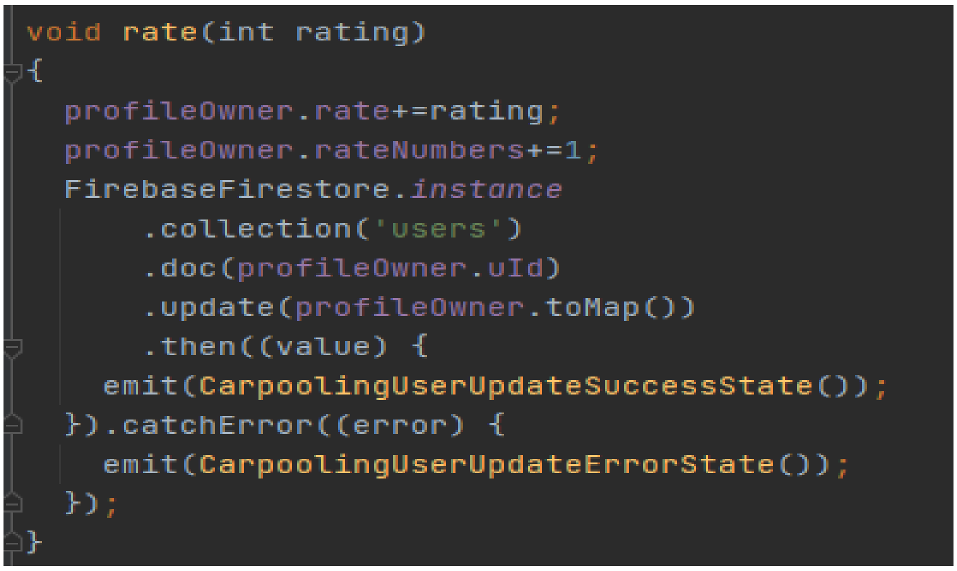
```
void createCar({
  required String model,
  required String brand,
  required String license,
  required String userlicense,
}) {
  emit(CarpoolingCarLoadingState());

  CarpoolingCarModel carmodel = CarpoolingCarModel(
    model:model,
    brand:brand,
    license:license,
    userlicense:userlicense,
    uId:userModel.uId,
    image:'https://iheartcraftythings.com/wp-content/uploads/2021/04/Car-DRAWING-%E2%80%93-STEP-9.jpg',
  );
  FirebaseFirestore.instance
      .collection('cars')
      .doc(userModel.uId)
      .set(carmodel.toMap())
      .then((value) {
    emit(CarpoolingCarSuccessState());
    showToast(text: "Car Added Successfully", state: ToastStates.SUCCESS);
    carModel=carmodel;
  }).catchError((error) {
    emit(CarpoolingCarErrorState());
  });
}
```



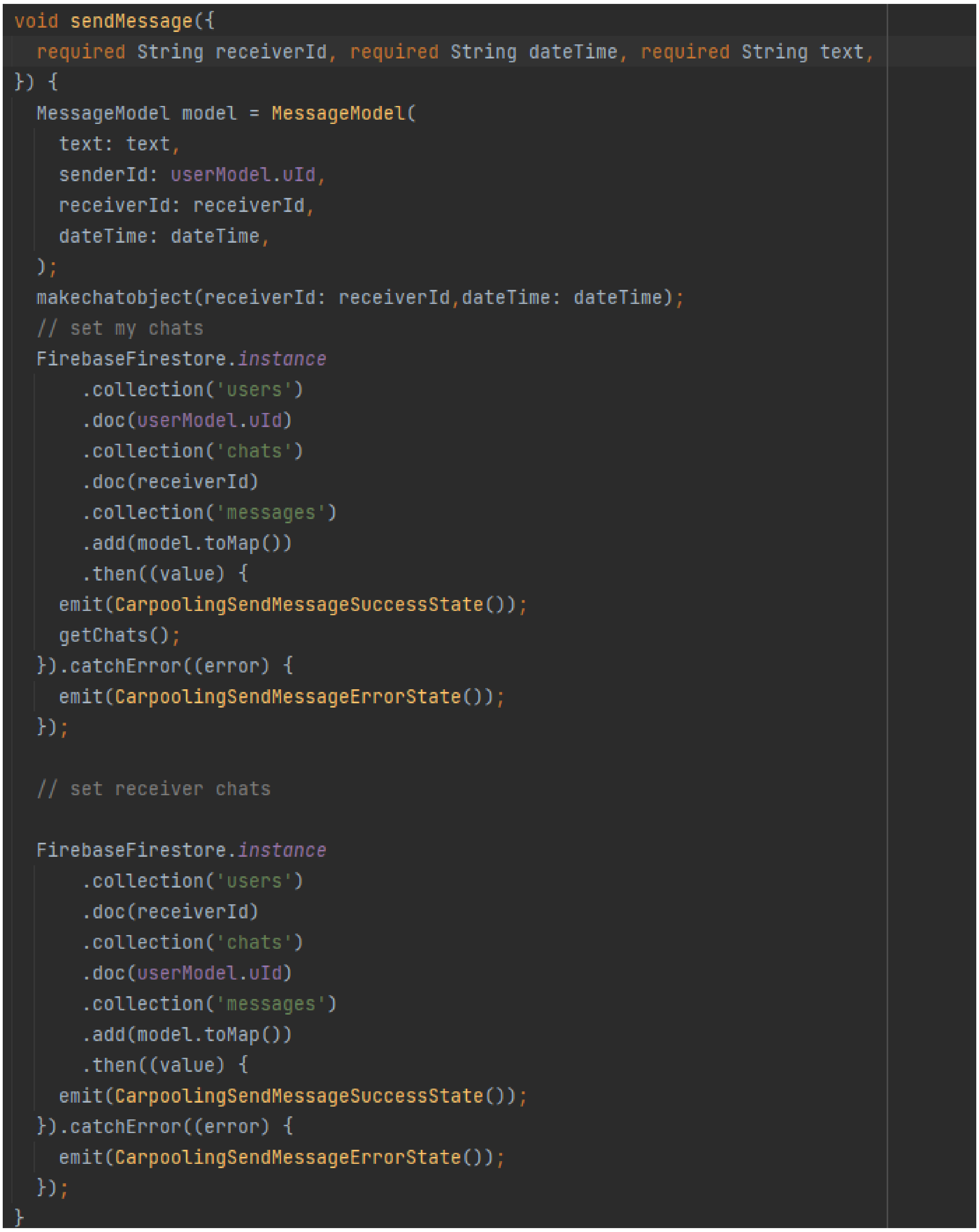
```
void visitprofile(context,widget)
{
  navigateTo(context, widget);
}

List<PostModel> visitposts = [];


void getVisitPosts() {
  visitposts.clear();
  FirebaseFirestore.instance.collection('posts').get().then((value) {
    value.docs.forEach((element) {
      if (element.data()['uId'] == profileOwner.uId)
        {
          visitposts.add(PostModel.fromJson(element.data()));
          print(visitposts.length);
        }
    });
    emit(CarpoolingGetVisitPostsSuccessState());
  }).catchError((error) {
    print(error.toString());
    emit(CarpoolingGetVisitPostsErrorState(error.toString()));
  });
}
```



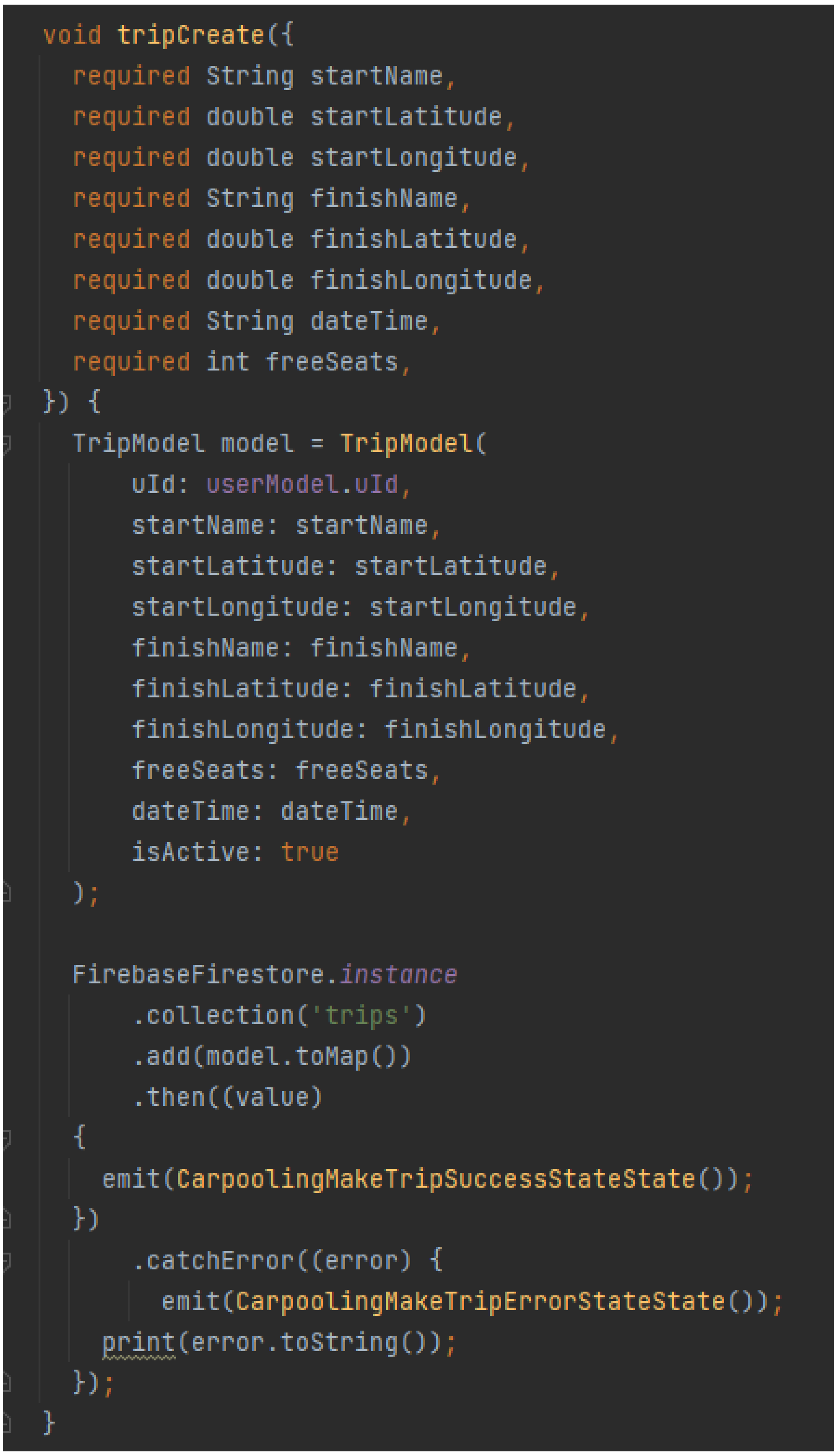
```
void createPost({
  required String dateTime,
  required String text,
  String? postImage,
}) {
  emit(CarpoolingCreatePostLoadingState());

  PostModel model = PostModel(
    name: userModel.name,
    image: userModel.image,
    uId: userModel.uId,
    dateTime: dateTime,
    text: text,
    postImage: postImage ?? '',
  );

  FirebaseFirestore.instance
      .collection('posts')
      .add(model.toMap())
      .then((value) {
    emit(CarpoolingCreatePostSuccessState());
  }).catchError((error) {
    emit(CarpoolingCreatePostErrorState());
  });
}
```



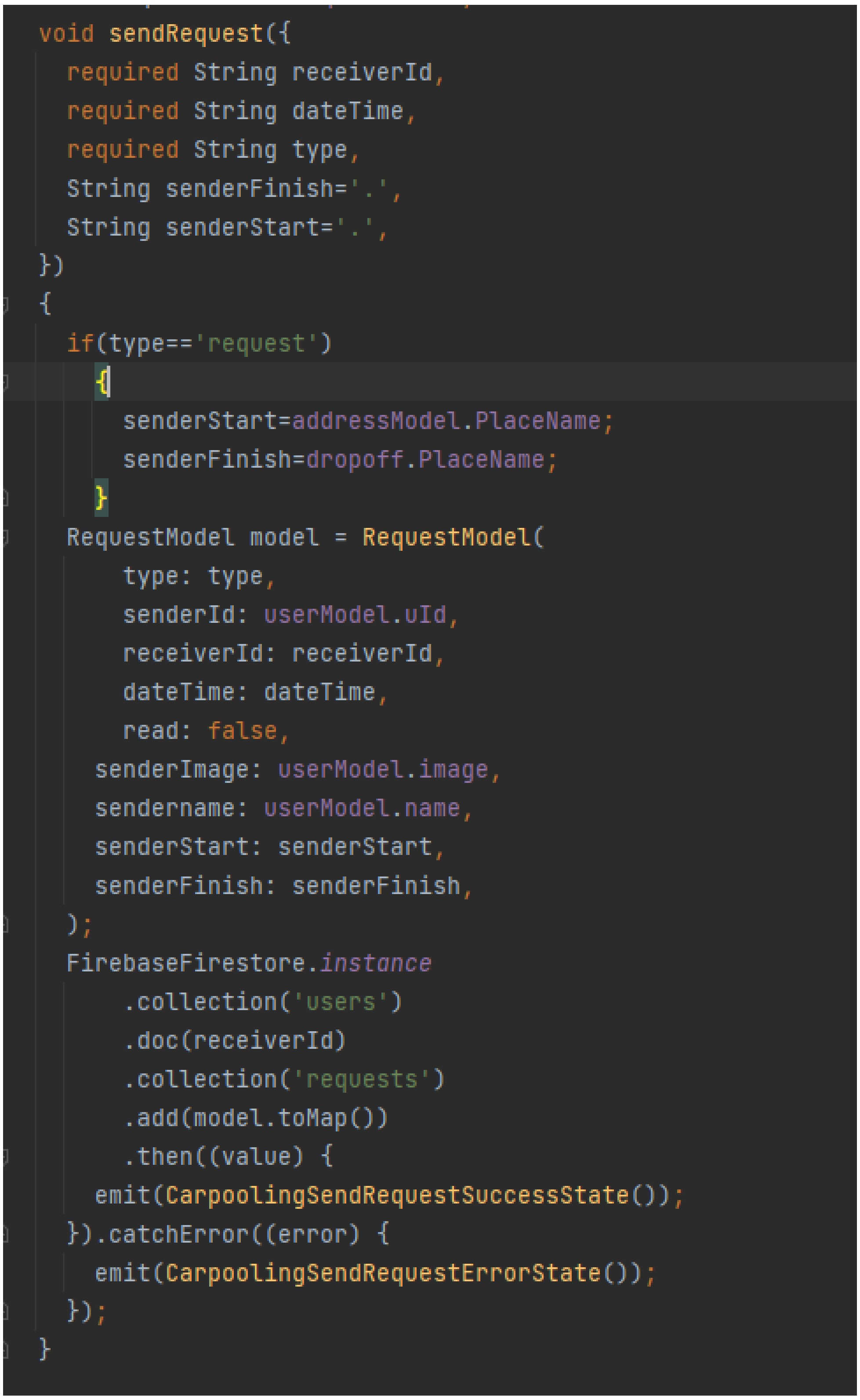
```
void rate(int rating)
{
  profileOwner.rate+=rating;
  profileOwner.rateNumbers+=1;
  FirebaseFirestore.instance
      .collection('users')
      .doc(profileOwner.uId)
      .update(profileOwner.toMap())
      .then((value) {
    emit(CarpoolingUserUpdateSuccessState());
  }).catchError((error) {
    emit(CarpoolingUserUpdateErrorState());
  });
}
```



```dart
void sendMessage({
  required String receiverId, required String dateTime, required String text,
}) {
  MessageModel model = MessageModel(
    text: text,
    senderId: userModel.uId,
    receiverId: receiverId,
    dateTime: dateTime,
  );
  makechatobject(receiverId: receiverId,dateTime: dateTime);
  // set my chats
  FirebaseFirestore.instance
      .collection('users')
      .doc(userModel.uId)
      .collection('chats')
      .doc(receiverId)
      .collection('messages')
      .add(model.toMap())
      .then((value) {
    emit(CarpoolingSendMessageSuccessState());
    getChats();
  }).catchError((error) {
    emit(CarpoolingSendMessageErrorState());
  });

  // set receiver chats

  FirebaseFirestore.instance
      .collection('users')
      .doc(receiverId)
      .collection('chats')
      .doc(userModel.uId)
      .collection('messages')
      .add(model.toMap())
      .then((value) {
    emit(CarpoolingSendMessageSuccessState());
  }).catchError((error) {
    emit(CarpoolingSendMessageErrorState());
  });
}
```



```
void tripCreate({
  required String startName,
  required double startLatitude,
  required double startLongitude,
  required String finishName,
  required double finishLatitude,
  required double finishLongitude,
  required String dateTime,
  required int freeSeats,
}) {
  TripModel model = TripModel(
      uId: userModel.uId,
      startName: startName,
      startLatitude: startLatitude,
      startLongitude: startLongitude,
      finishName: finishName,
      finishLatitude: finishLatitude,
      finishLongitude: finishLongitude,
      freeSeats: freeSeats,
      dateTime: dateTime,
      isActive: true
  );

  FirebaseFirestore.instance
      .collection('trips')
      .add(model.toMap())
      .then((value)
  {
    emit(CarpoolingMakeTripSuccessStateState());
  })
      .catchError((error) {
        emit(CarpoolingMakeTripErrorStateState());
    print(error.toString());
  });
}
```



```
void sendRequest({
  required String receiverId,
  required String dateTime,
  required String type,
  String senderFinish='.',
  String senderStart='.',
})
{
  if(type=='request')
    {
      senderStart=addressModel.PlaceName;
      senderFinish=dropoff.PlaceName;
    }
  RequestModel model = RequestModel(
      type: type,
      senderId: userModel.uId,
      receiverId: receiverId,
      dateTime: dateTime,
      read: false,
    senderImage: userModel.image,
    sendername: userModel.name,
    senderStart: senderStart,
    senderFinish: senderFinish,
  );
  FirebaseFirestore.instance
      .collection('users')
      .doc(receiverId)
      .collection('requests')
      .add(model.toMap())
      .then((value) {
    emit(CarpoolingSendRequestSuccessState());
  }).catchError((error) {
    emit(CarpoolingSendRequestErrorState());
  });
}
```

Chapter 5: Testing

5.1 Unit testing:

Testing of individual items (e.g., modules, programs, objects, classes, etc.) Usually as part of the coding phase, in isolation from other development items and the system as a whole.

5.2 Integration testing:

Testing the interfaces between major (e.g., systems level application modules) and minor (e.g., individual programs or components) items within an application which must interact with each other.

5.3 additional testing:

-System testing

Testing a system behavior as a whole when development is finished and the system can be tested as a complete entity.

-Regression Testing

To check older functionality after integrating new functionality.

-Acceptance Testing

Testing to ensure that a development is ready to be deployed into the business, operational or production environment.

-Performance Testing

Accomplished a designated function regarding processing time and throughput rate.

-Load Testing

Measuring the behavior of within creasing load which can be handled by the component or system.

-Stress Testing

Evaluate a system or component at or beyond the limits of its specified requirements.

-Security Testing

Testing how well the system protects against unauthorized internal or external access.

- **Test Cases**
- User Register Test Case

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass or fail
01	Check user register with valid data	1. Open The app 2. Chooses Sign Up 3. enter data 4. click on register button	username:Mohamed hesham email:mohamed201900764@gmail.com password:123456eE@ address:Egypt,cairo phoneNumber:01153442908	User will register successfully	As expected	pass
02	Check user register with invalid data	1. Open The Site 2. Chooses register 3. enter data	username:Mohamed hesham email:mohamed201900764@gmail.com password:123456 address: phoneNumber:0112	Alert user with errors,	As expected	pass
03	Check user register with valid data and used mail	1. Open The Site 2. Chooses Sign Up 3. enter data 4. click on sign up button	username:userx email:mohamed201900764@gmail.com password:123456eE@ address:Egypt,cairo phoneNumber:01125432404	Give error message	As expected	pass

User login

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass or fail
05	log in with valid data	1 : open the app 2 : enter valid username and password 3 : click log in	email:mohamed201900764@gmail.com Password : 123456eE@	The system enter the user page with all his privileges	As expected	pass
06	log in with invalid data	1 : open the app 2 : enter invalid username or password 3 : click log in	email:mohamed201900764@gmail.com Password : 123eloi@	The system return error message	As expected	pass

User logout

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass or fail
07	log out from the app	1 : open edit profile page 2 : click logout	-	The system clears cache and send user to login page	As expected	Pass

User add post

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass or fail
08	Create post with text and image	1 : open profile page 2 : click add post 3: write post text 4: add post image 5:click post	Text :xxxxxxxxxxxxxxxxxxxxxxxx Image :image file	The post is added to user profile	As expected	Pass
09	Create post with text only	1 : open profile page 2 : click add post 3: write post text 4:click post	Text :xxxxxxxxxxxxxxxxxxxxxxxx	The post is added to user profile	As expected	Pass
10	Create post with image only	1 : open profile page 2 : click add post 3: add post image 4:click post	Image :image file	The post is added to user profile	As expected	pass
11	Create post without text or image	1 : open profile page 4:click post	-	The post doesn't success	As expected	pass

User edit profile

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass or fail
12	Edit profile data with valid data	1 : open profile page 2 : click edit profile 3 change the data 4: click update	User name:Mohamed Address: giza phoneNumber:01125432404	The profile data is updated	As expected	Pass
13	Edit profile data with some empty data	1 : open profile page 2 : click edit profile 3 change the data 4: click update	User name:Mohamed Address: giza phoneNumber:	The profile isn't updated and send error	As expected	Pass

User rate

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass or fail
14	Rate user for first time	1 : open user profile 2 : click star 3 add rate 4: click ok	Rate: 3.0	The rated user's rate changed	As expected	Pass
15	Rate user rated before	1 : open user profile 2 : click star 3 add rate 4: click ok	Old rate:3.0 New rate :4.0	The rate updated and the value override	Put new rate As first rate without update	fail

User add and update car

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass or fail
16	User don't have car :add car	1 : open car page 2: add car data 3: click submit	License: 123xaaa66 User license: 111mmpa55 Car model: fiat 2016 Car brand :fiat	The user have car	As expected	Pass
17	User has a car :update car with valid data	1 : open car page 2: change car data 3: click submit	License: 123xaaa66 User license: 111mmpa55 Car model: lada 2002 Car brand :lada	The car data is updated	As expected	Pass
18	User has a car :update car with some empty data	1 : open car page 2: change car data 3: click submit	License: 123xaaa66 User license: Car model: fiat 2016 Car brand :fiat	The car data isn't updated and send error	As expected	pass

User request trip

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass or fail
19	User request to join trip	1 : open find tripe page 2 : click on the trip 3 add your route 4: click request	Start point : Maadi End point : Helwan	Request is sent to trip owner and user wait for notification	As expected	Pass

User make trip

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass or fail
20	User who has a car make trip	1 : click search for destination 2 : add end point 3 : click + button 4: add free seats 5: click ok	Start point : Maadi End point : Helwan Free seats :3	Trip created and wait for requests	As expected	Pass
21	User who don't have a car make trip	1 : click search for destination 2 : add end point 3 : click + button	-	Error message : should have a car to make trip	As expected	Pass
22	User make trip while he/she is in trip	1 : click search for destination 2 : add end point 3 : click + button	-	Open the existing trip	As expected	pass

User accept request

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass or fail
23	User accept request	1 : open notifications 2:check requests 2 : click accept for request	-	Request is accepted and notification,message are sent to request sender for being accepted	As expected	Pass

User reject request

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass or fail
24	User reject request	1 : open notifications 2:check requests 2 : click reject for request	-	Request is rejected and notification in sent to request sender for being rejected	As expected	Pass

User send message

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass or fail
25	User send message for first time	1 : open user profile 2: click send message 3: type the message 4: click arrow button	Hello	Message is sent to user in the same time and appear for sender and receiver	As expected	Pass
26	User send message for chat	1: open chats 2: click on the user chat 3: type the message 4: click arrow button	hello	Message is sent to user in the same time and appear for sender and receiver and added after the old messages	As expected	Pass

Chapter 6: Results and discussion

6.1 Results

During the analysis and planning phase, we gathered all the information we needed about the project's requirements and the features we'd use to finish it 100%. We planned to create an application that meets the needs of anyone who wants to use carpooling application.

6.1.1 Expected result

- User can register.
- User can login with Facebook or Gmail account
- User can log in and log out.
- User can add car.
- User can update car.
- User can update profile.
- User can add post.
- User can access another user profile.
- User can rate another user.
- User can send message.
- User can make trip.
- User can request trip.
- User can answer trip.
- User can view notifications.
- User location automatically declare.
- Admin can log in and log out.
- Admin can block user.
- Admin can update user profile.
- Admin can unblock user.

6.1.2 Actual result

- User can register.
- User can log in and log out.
- User can add car.
- User can update car.
- User can update profile.
- User can add post.
- User can access another user profile.
- User can rate another user.
- User can send message.
- User can make trip.
- User can request trip.
- User can answer trip.
- User can view notifications.
- User location automatically declare.
- Admin can log in and log out.
- Admin can block user.
- Admin can update user profile.
- Admin can unblock user.

6.2 Discussion

- We have managed to get most of our expected results except the matching of routes because of the lack of resources and technology which we consider as future work, login with Facebook or Gmail.

Chapter 7: Conclusion

In general, in the presence of technology, it makes human life simpler and easier, and because science and progress are the reason for the existence of technology and the development in which we live now. We believe that it is important to provide all humans everywhere an application that helps them to find better transports easily and nor wasting resources for cars owners. Our application solves the problem of balancing between private and public transports for both rider and driver.

Since the project purpose does not end with the final discussion, we are planning to develop more items to make the applications work the best afterwards.

- Add payment recommendation for trips.
- Match the routes by graph.
- Add more communication methods.
- Monitor every user behavior to recommend users with similar preferences.
- Make trip recommendation based on direction.
- Add more security levels.
- Make the app more socially.

Chapter 8: Future work

Eventually, in this project we tried to make the best we can to help car owners to benefit from their car while going into trip, and help people who don’t have car to find cheaper trip than private transport. Consider it as a way of expressing our duty towards community and environment as we felt responsible to participate in making transports easier and more comfortable for them.

We tried to do that with our limited fund, and resources and we are sure we are going to expand this application and enhance its performance and fame if we get the right resources and fund.

As we said before in the previous chapter there are some functions can be added to improve the project like: matching the routes by graph, monitor every user behavior to recommend users with similar preferences, add payment recommendation for trips.