Software Requirements Specification

Of

CAB SHARING

Mohammad Hashemi
Nachiket Bhagwat
Praveen Kumar Devaraj

PART 2- REQUIREMENT SPECIFICATION

University of Colorado, Boulder

Table of Contents

1. Project Summary

TEAM : Mohammad Hashemi
Nachiket Bhagawat
Praveen Kumar Devaraj

TITLE : Cab Sharing

Cab-sharing means sharing your cab or taxi with other passengers (mainly strangers) in the same locality who are heading in same direction thus reducing the individual expenditure. There have been systems for carpooling, but we are looking for sharing cabs as next step in transportation. We came up with this sharing model from big companies like AirBnB (Sharing House), ZipCar (Sharing Cars), TaskRabbit (Sharing Services) etc.
We will be building an Android application for sharing cab where multiple users can book cabs and the grouping of users interested in sharing based on their preferences is done by the server. This application can be used by the cab companies to offer the sharing feature to the users.

2. System Requirements

Business Requirements			
ID	**Requirements**	**Topic Area**	**Priority**
BR-001	UserIds must be Valid and email must be verified by user	Authentication	Critical
BR-002	Pickup Location should be valid and in region where system provides the service	Validation	Critical

User Requirements				
ID	**Requirements**	**Topic Area**	**User**	**Priority**
UR-001	As a passenger, I should be able to request for a cab immediately and at a specific time	Booking	Passenger	Critical
UR-002	As a passenger, I should be able to specify filter criteria based on drivers ratings (like someone with rating greater than 4)	Booking	Passenger	Medium
UR-003	As a user, I should be able to cancel an active trip	Cancellation	Driver and Passenger	High
UR-004	As a passenger, I should be able to provide feedback on driver	Feedback	Passenger	Medium
UR-005	As a passenger, I should be able to select the number of passengers accompanying me	Booking	Passengers	Medium
UR-006	As a driver, I should be able select a ride	Booking	Driver	Critical
UR-007	As a driver, I should be able to start and end trip	Booking	Driver	Critical
UR-008	As a user, I should be able to signup, login and logout from the system	Authentication	Passenger and Driver	Critical
UR-009	As a user, I should be able to see all active bookings	Booking	Passenger and Driver	High
UR-010	As a driver, I should able to see new group bookings	Booking	Driver	Critical

Functional Requirements			
ID	**Requirements**	**Topic Area**	**Priority**
FR-001	System should verify the payment profile of the user	Payment	Critical
FR-002	System should be able to access user location for communicating to the drivers	Booking	Critical
FR-003	If sharing the cab, system should be able to find user availability in close locality and travelling in same direction.	Booking	High
FR-004	System should be able to group passengers travelling in the same direction by matching user preferences	Booking	High
FR-005	System should be able to compute peak hour charges	Booking	High
FR-006	System should be able to process payment	Payment	Critical
FR-007	System should be able to handle cancellation from user	Cancellation	Critical
FR-008	System should be able to suggest shortest path to driver for picking up all passengers	Booking	Optional
FR-009	System should be able to estimate total booking fare using pickup and dropoff locations	Payment	High
FR-010	System should be able to calculate individual fare from travel summary	Payment	Critical

Non-Functional Requirements			
ID	**Requirements**	**Topic Area**	**Priority**
NF-001	The User Interface should be intuitive	Usability	Medium
NF-002	Background Check of Drivers	Legal	Optional
NF-003	Drivers should be able to set a start and end point (For example, driver should be able to say that I'm going from my home to work, every day, if someone is interested, then they can book this driver)	Extensibility	Optional
NF-004	System should able to predict patterns based on booking history and give suggestions to passenger	Extensibility	Optional
NF-005	Reduce the number of notifications to users	Usability	Low

NF-006	Server should be able to handle multiple requests simultaneously	Scalability	Low
NF-007	Integration with other services like Uber	Extensibility	Optional
NF-008	Secure Storage of Payment profiles	Security	Medium

3. Use Case View

Use Case Diagrams

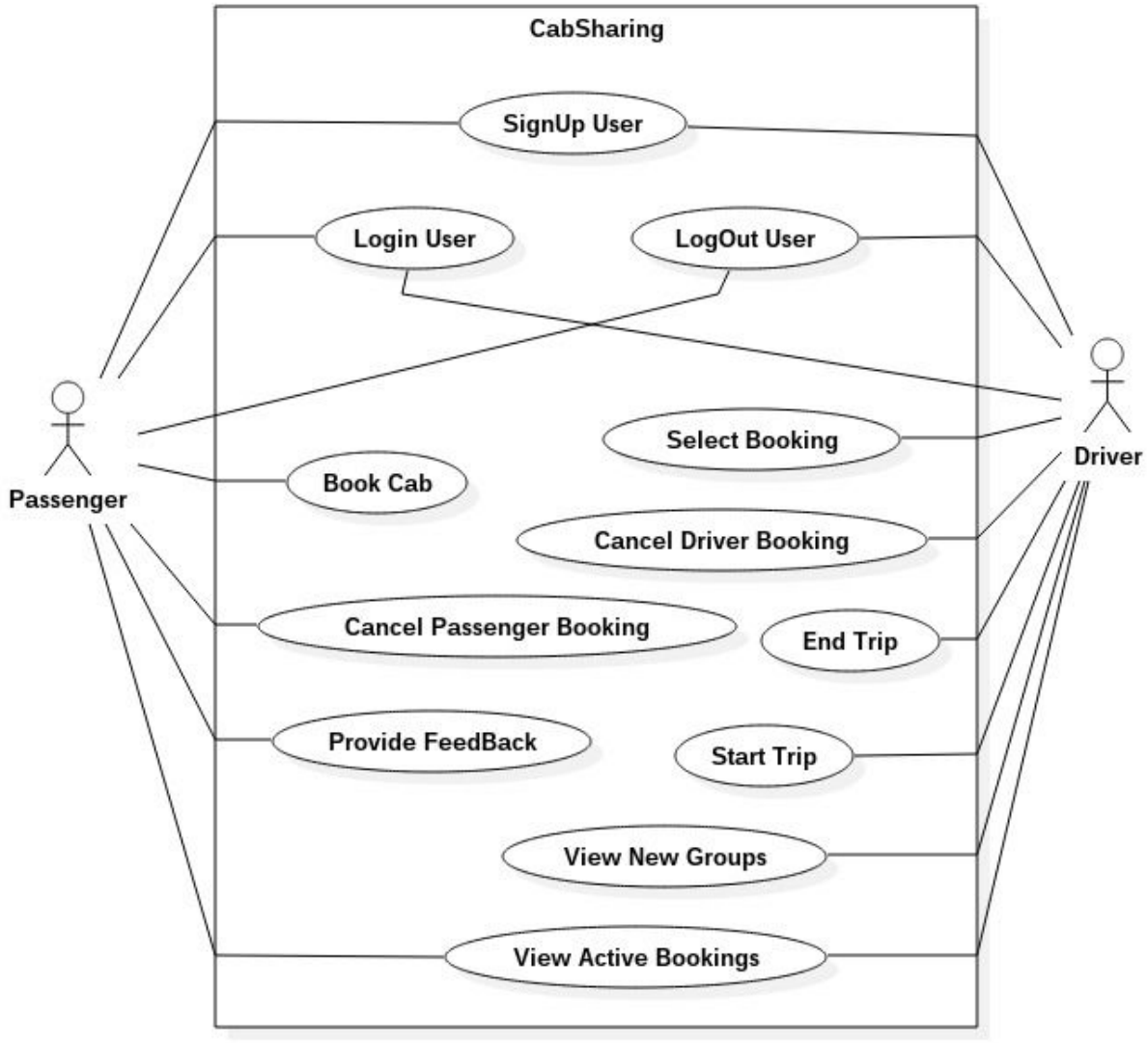


Fig 1: A high level Use case of the system.

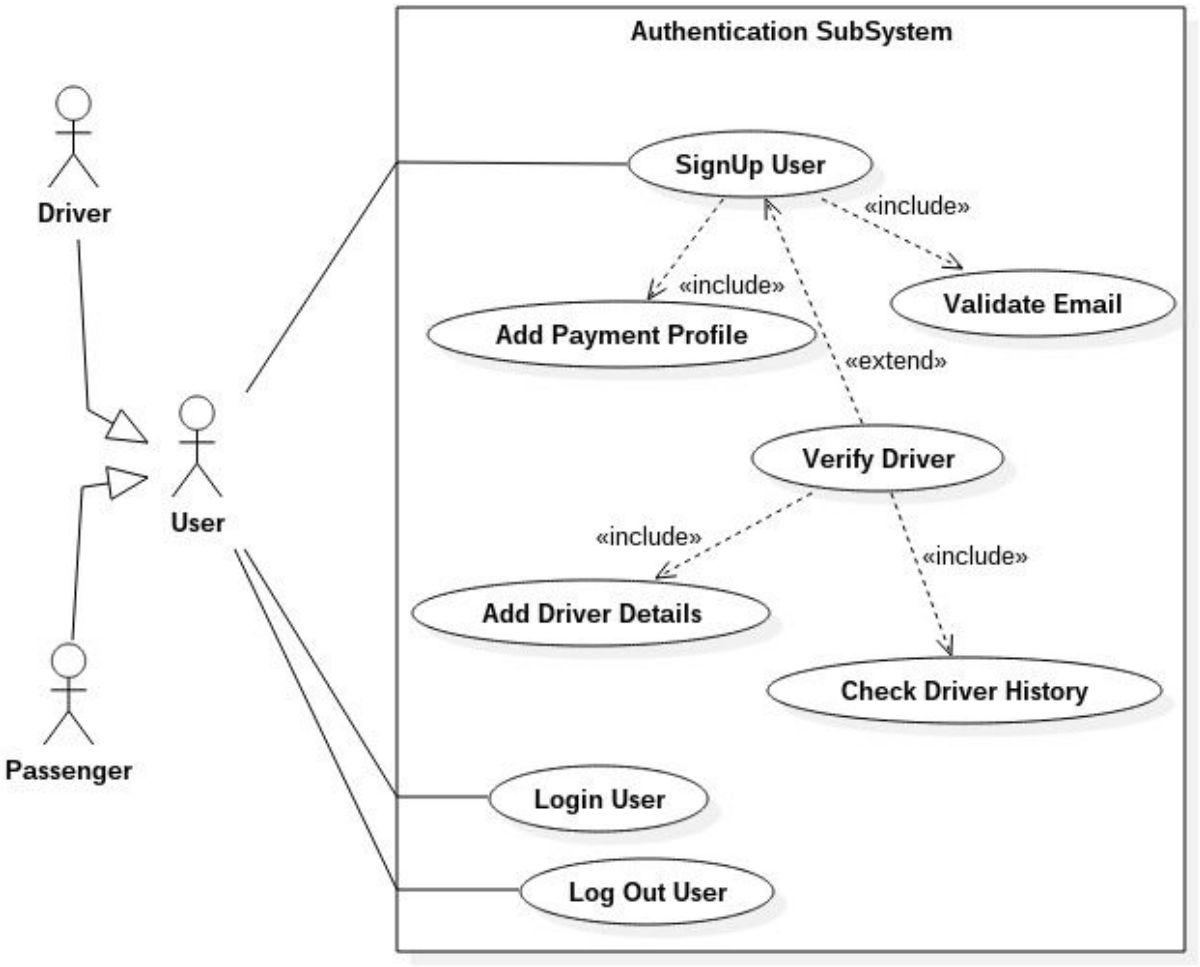


Fig 2 : Authentication SubSystem Use case

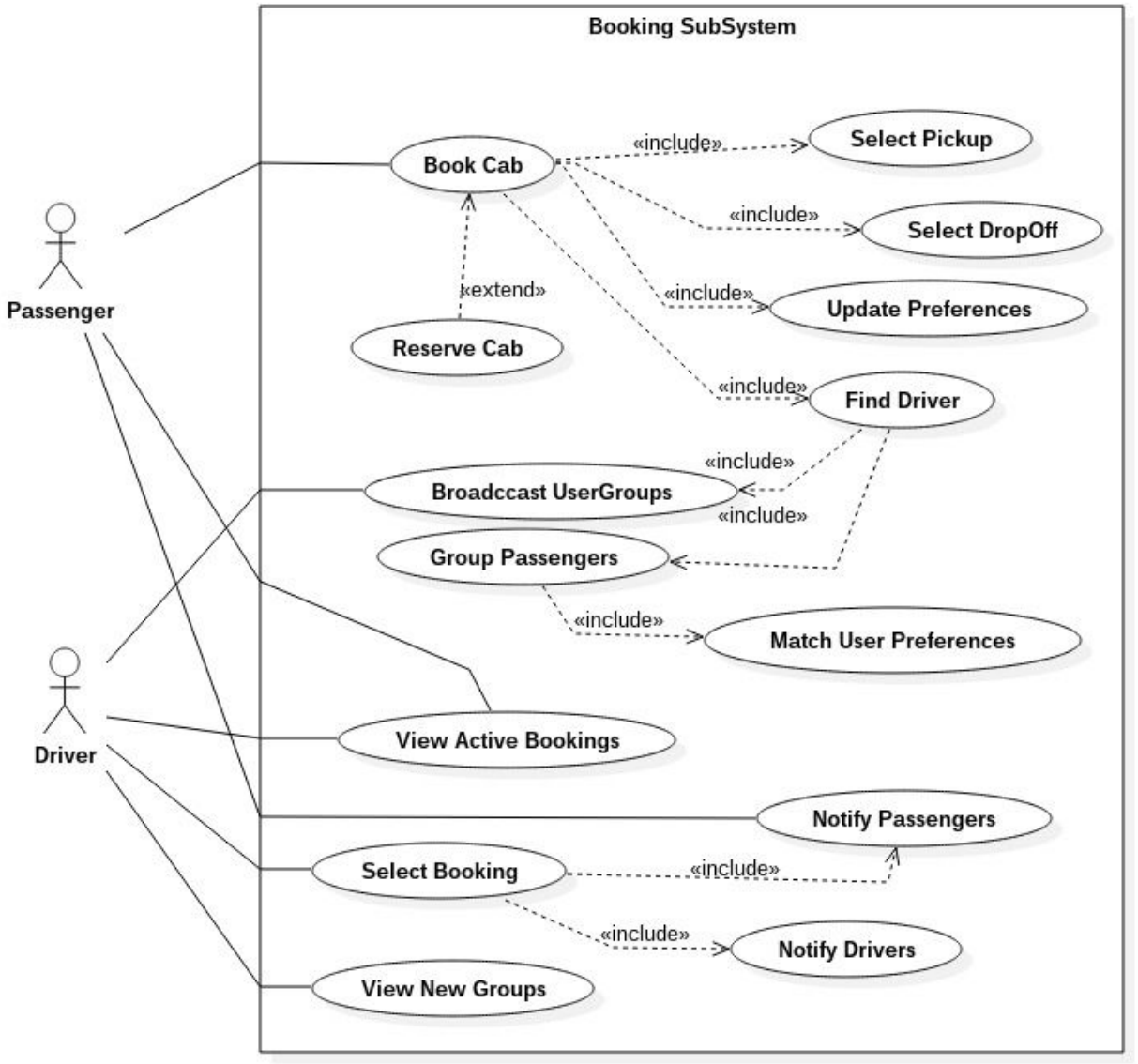


Fig 3 : Booking SubSystem Use case

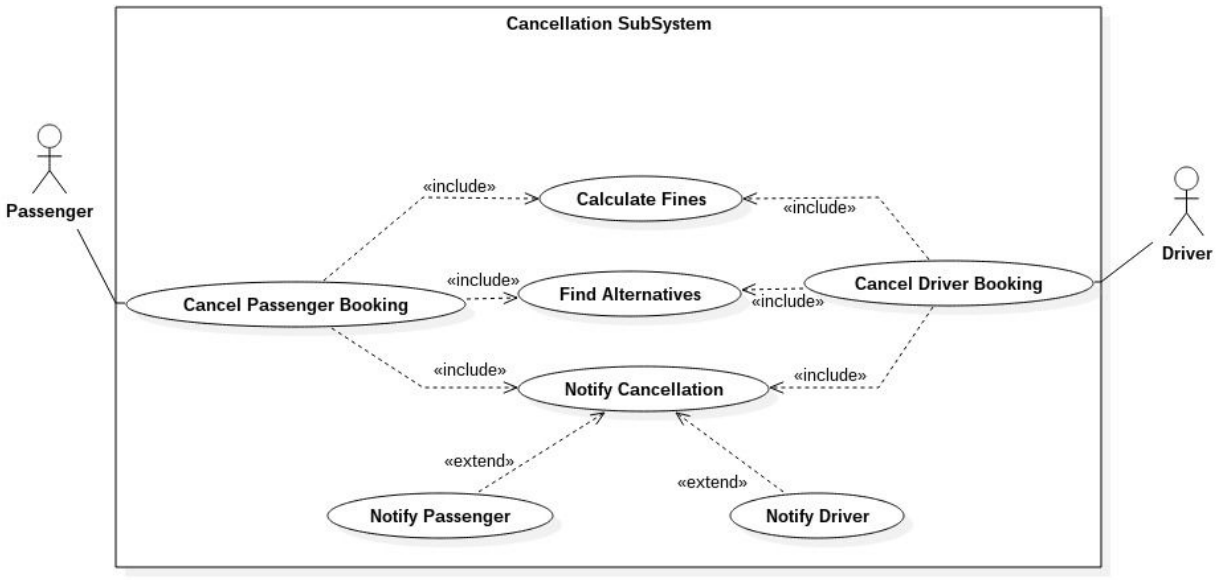


Fig 4: Cancellation SubSystem Use case

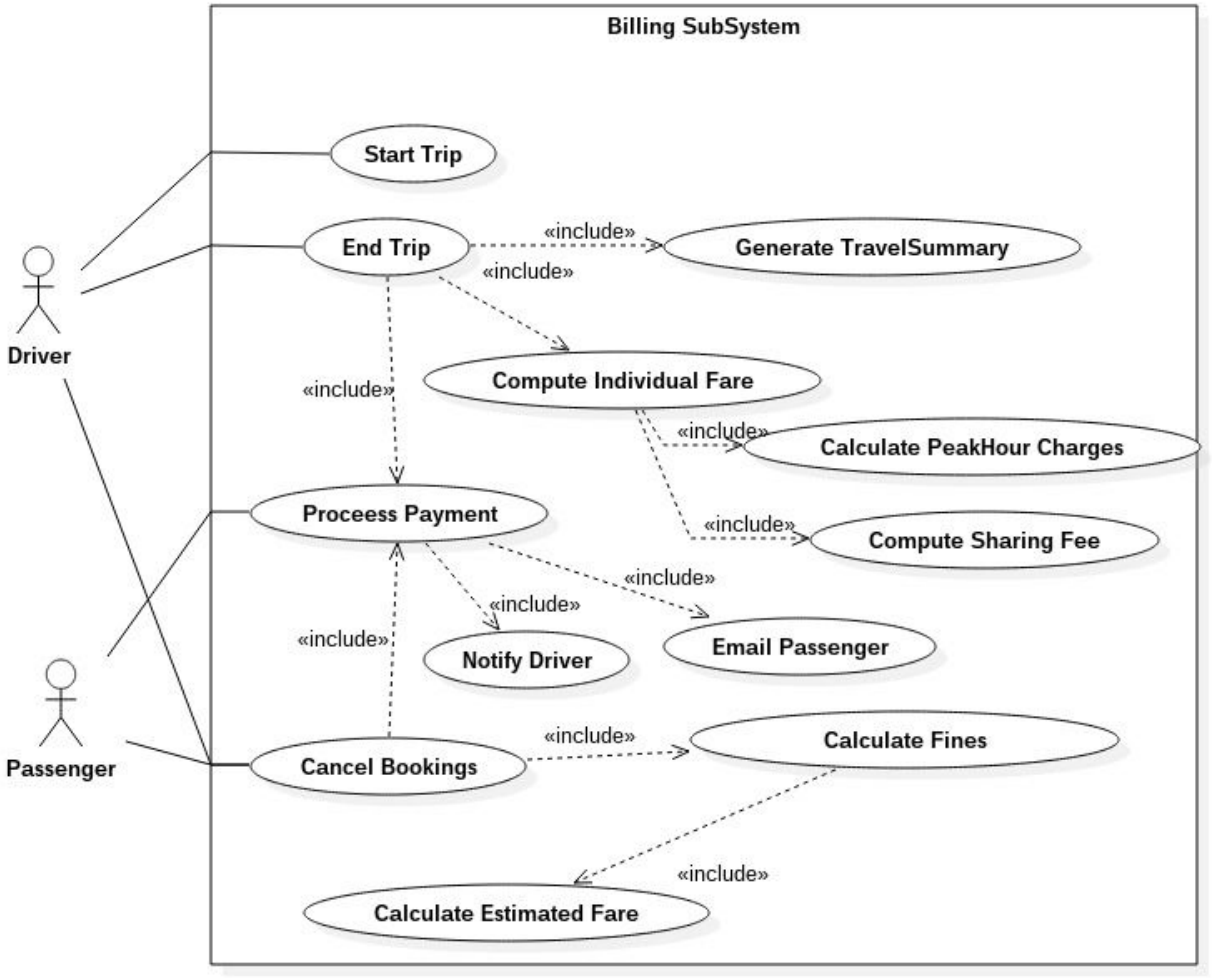


Fig 5: Billing SubSystem Use case

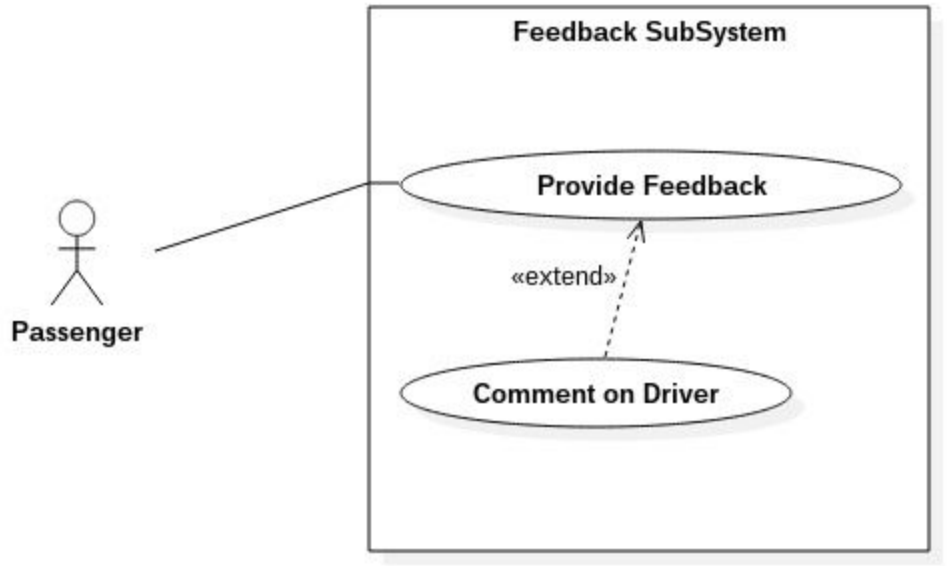


Fig 6 : FeedBack SubSystem Use case

Full size Use case diagrams can be found here,

Use Case Documents:

Use Case ID:	UC-001
Use Case Name:	Book Cab
Description:	Passenger can book a cab immediately or for future

Actors:	Passenger, Driver		
Pre-conditions	Passenger should be logged in and his payment profile should be verified.		
Post conditions	Passenger has active booking in the system and booking information in their mobile device		
Frequency of Use:	Frequently throughout the day by passengers.		
Flow of Events:		Actor Action	System Response
	1	Search pickup location on the map	Show that location on the map
	2	Put pin on the desired pickup location	
	3	Search dropoff location on the map	Show that location on the map
	4	Put pin on the desired drop off location	
	5	Update preferences for booking (driver rating, interval time for pickup, number of people with you, number of people to share a cab with them)	
	6	Click on “Book” button to Send booking details to the server	Create a booking based on booking details received from passenger. Add booking to the booking grid Add an event by time limits set by passenger into the bookingList. Notify booking reference to the passenger. Find a group for this passenger by matching the preferences and add this booking to the group If the group is full broadcast its information to the drivers in the same locality of passengers.

Variations:	2. User wants to book a cab immediately without sharing with anyone else. 3. After a group is found for the passenger, if the group is not full and passenger wants to book immediately, send fail notification to the passenger. 4. If system can't find a group for current booking it should create a new group. 5. If the found group is a "cancelled" group, system should set the substituted flag to "true"
Notes and Issues:	Groups which are not full should be deleted after the passenger time limit expires and system should send a failed notification to the passenger. A "cancelled" group is a group that one of the passenger from this group has cancelled his booking.
Developer Notes:	There should be a new thread to check time limits set by passenger and handle groups which are not full.

Use Case ID:	UC-002
Use Case Name:	Login User
Description:	Driver and Passenger can log in to the system whenever they want if they are not logged in.

Actors:	Passenger, Driver		
Pre conditions	User should be signed up before logging in. User should not be logged in.		
Post conditions	User is logged in to the system and they have a communication link with the server.		
Frequency of Use:	Frequently throughout the day by users.		
Flow of Events:		Actor Action	System Response
	1	Enter username	
	2	Enter password	
	3	Click on login button to send information to the server	Calculate the hash of password and query the database to see if this user exists and the password is correct. If username and password are correct, send back success and user type to the user, otherwise send a failure.
Variations:			
Notes and Issues:			
Developer Notes:	After login was successful, we should check the usertype. If it is a driver, he should see grouplist and if it is a passenger he should see the map.		

Use Case ID:	UC-003
Use Case Name:	View Active Bookings
Description:	Passengers and Drivers can see all the active bookings.

Actors:	Passenger, Driver		
Pre conditions	User should be logged in. User should have active bookings.		
Post conditions	User sees all the active bookings.		
Frequency of Use:	Frequently by user.		
Flow of Events:		Actor Action	System Response
	1	Select “View Active Bookings” button from the left panel.	Find all active bookings of a user, send back their information to the client app and display all the active bookings.
Variations:			
Notes and Issues:			
Developer Notes:	In the list of all active bookings, we need two buttons, one for seeing the details and other for canceling this ride.		

Use Case ID:	UC-004
Use Case Name:	View New Groups
Description:	Drivers can see all the new groups of passengers in the same locality.

Actors:	Driver		
Pre conditions	Driver should be logged in. There should be some requests from passengers in the same locality and their group is full.		
Post conditions	Driver sees all the new groups in the same locality.		
Frequency of Use:	Frequently throughout the day by driver based on frequency of bookings.		
Flow of Events:		Actor Action	System Response

	1	Select “View New Groups” from left panel.	Server will send groups for which no driver is assigned after they become full.
	2	Select “View Group Details”.	Show the map with pickup and dropoff locations of all the passengers and total miles and cost estimated for this trip.
Variations:			
Notes and Issues:	A group is full when it matches the number of passengers preferred by all members.		
Developer Notes:			

Use Case ID:	UC-005
Use Case Name:	End Trip
Description:	Driver must end the trip after reaching final destination.

Actors:	Driver		
Pre conditions	A trip is in progress and the passengers picked up and the last passenger is dropped at the destination.		
Post conditions	Travel summary is generated and sent to the server for billing and payment is processed in the server.		
Frequency of Use:	Once every ride.		
Flow of Events:		Actor Action	System Response
	1	Driver clicks on the “End Trip” button after dropping the last passenger at his destination.	Client app creates a travel summary with the total distance covered and the time taken for the ride. Travel summary is sent to the server. Server on receiving the information finds corresponding groups, and calculates sharing fee and peak hour charges for the booking. Calculate total fare, individual fare, process payment for passenger. Notify passenger about the payment and credit driver payment.
Variations:			

Notes and Issues:	
Developer Notes:	

Use Case ID:	UC-006
Use Case Name:	Select Booking
Description:	Driver can select a booking from the new group list.

Actors:	Driver		
Pre conditions	There is a booking from the passengers and the server has broadcasted the user groups to the all drivers in the same locality.		
Post conditions	The booking is assigned to a driver.		
Frequency of Use:	Once for every Booking.		
Flow of Events:		Actor Action	System Response
	1	"Select Booking" button is clicked by the driver.	The user group is assigned to a driver Corresponding event should be removed from booking event list. The details of the driver is notified to the passengers. Driver is notified with the information of all the passengers.
Variations:			
Notes and Issues:	A booking for a driver is a group of bookings made by different passengers.		
Developer Notes:	If a driver selects a booking, other drivers in the broadcast group cannot select that booking and it is removed from the new groups list.		

Use Case ID:	UC-007
Use Case Name:	Start trip
Description:	Driver can select start trip before starting at first pickup location.

Actors:	Driver		
Pre conditions	There is a booking from the passengers and a driver has accepted the ride.		
Post conditions	Exact pickup location and start time is stored and it's used to generate the travel summary.		
Frequency of Use:	Once for every Booking.		
Flow of Events:		Actor Action	System Response
	1	Start Trip button is clicked by the driver	Store the initial pickup location and pickup time of first passenger.
Variations:			
Notes and Issues:			
Developer Notes:			

Use Case ID:	UC-008
Use Case Name:	SignUp Passenger
Description:	A passenger can sign up using his email id.

Actors:	Passenger		
Pre conditions	Passenger has a valid email id and payment information.		
Post conditions	Passenger has a valid profile in the system.		
Frequency of Use:	Once for every passenger.		
Flow of Events:		Actor Action	System Response
	1	Passenger enters a valid email and password and clicks on "Sign Up" button.	Validate email and check for duplicate registration.

	2	Passenger enters the valid payment information and clicks on “Add Payment Profile” button.	Validate payment information and check for the status of the account(active or inactive).
Variations:	2. If the email is invalid, passenger is not allowed to sign up. 3. If the payment information is not valid, passenger is notified and requested to enter valid information.		
Notes and Issues:			
Developer Notes:			

Use Case ID:	UC-009
Use Case Name:	SignUp Driver
Description:	Driver can sign up with valid email id and driver details.

Actors:	Driver		
Pre conditions	Driver must have a valid email id, driver details and payment information.		
Post conditions	Driver has a valid profile in the system.		
Frequency of Use:	Once for every driver.		
Flow of Events:		Actor Action	System Response
	1	Driver enters a valid email and password, selects “Driver” checkbox and clicks on “Sign Up” button.	Validate Email and check for duplicate registration.
	2	Driver enters License Number and Vehicle Number and Model and clicks “Add Driver Details”	Cross check user data and check driving history of the driver from DMV records.
	3	Driver enters the valid payment information and clicks on “Add Payment Profile” button.	Validate payment information and check for the status of the account(active or inactive).
Variations:	2. If the email is invalid, driver is not allowed to sign up. 3. If the driving history reflects criminal behaviour, driver is not allowed to signup and proceed further. 4. If the payment information is not valid, driver is notified and requested to enter valid information.		
Notes and Issues:	.		
Developer Notes:			

Use Case ID:	UC-010
Use Case Name:	Cancel Passenger Booking
Description:	Passenger can cancel a booked cab from the app between the time the cab is booked and a driver is dispatched.

Actors:	Passenger		
Pre conditions	Passenger must be logged in. Passenger must have an active booking that can be cancelled.		
Post conditions	System will fine the passenger and add add money back to fellow cab sharing passengers.		
Frequency of Use:	Not very frequently by a passenger as there is fine involved.		
Flow of Events:		Actor Action	System Response
	1	Passenger clicks on "Cancel Booking" button with the booking reference.	Send cancellation details to server. Update group by removing booking from corresponding group. Notify driver of cancellation by passenger. Add booking to cancelled event list. After time limit set by passenger exceeded, if substitute passenger flag not set, fine should be calculated for the passenger. Process payment for cancelled passenger.
Variations:	2. If substitute found, passenger is replaced by substitute and driver is notified of the change. No fines are calculated for the cancelled passenger. 3. If passenger could not pay fine, block the passenger from further use. 4. No other passengers were travelling with cancelled passenger, notify driver of the cancellation of trip.		
Notes and Issues:	Pay back fellow cab sharing passengers from the fine.		
Developer Notes:			

Use Case ID:	UC-011
Use Case Name:	Cancel Driver Booking

Description:	Driver can cancel a selected booking from the app between the time the cab is booked and he is dispatched.

Actors:	Driver		
Pre conditions	Driver must be logged in. Driver must have an active selected booking that can be cancelled.		
Post conditions	System will fine the driver and add the money back to his passengers.		
Frequency of Use:	Not very frequently by a driver as there is fine involved.		
Flow of Events:		Actor Action	System Response
	1	Driver clicks on "Cancel Booking" button with the booking reference.	If substitute driver flag not set, calculate fine for driver. Notify passengers of the cancellation by the driver. Charge the driver.
Variations:	2. If substitute found, driver is replaced by substitute and passengers are notified of the change. No fines are calculated for the driver. 3. If driver could not pay fine, block the driver from further use and pay his passengers from system's profits.		
Notes and Issues:	Passengers should be paid back from the fine.		
Developer Notes:			

Use Case ID:	UC-012
Use Case Name:	Provide FeedBack
Description:	Passenger can provide the feedback from the app after journey is over.

Actors:	Passenger		
Pre conditions	Passenger must be logged in. Passenger must have completed a trip from the booking made by him earlier.		
Post conditions	System will add the feedback in database for future use.		
Frequency of Use:	Once by passenger after every trip.		
Flow of Events:		Actor Action	System Response
	1	After trip, passenger chooses rating to driver from specified scale, fills comments about driver and clicks on "Provide Feedback" button.	Store comments about driver in the database. Calculate new rating for the driver for further use.

Variations:	
Notes and Issues:	If driver rating is below a threshold, system may block driver from further use.
Developer Notes:	

Use Case ID:	UC-013
Use Case Name:	LogOut User
Description:	Driver and Passenger can log out of the system after logging in.

Actors:	Passenger, Driver		
Pre conditions	User should be logged in before logging out.		
Post conditions	User is logged out of the system and can't communicate with server without logging in again.		
Frequency of Use:	Once for every login.		
Flow of Events:		Actor Action	System Response
	1	User clicks on "Logout" button.	Tear down connection.
Variations:			
Notes and Issues:			
Developer Notes:			

4. Logical View

Class Diagrams

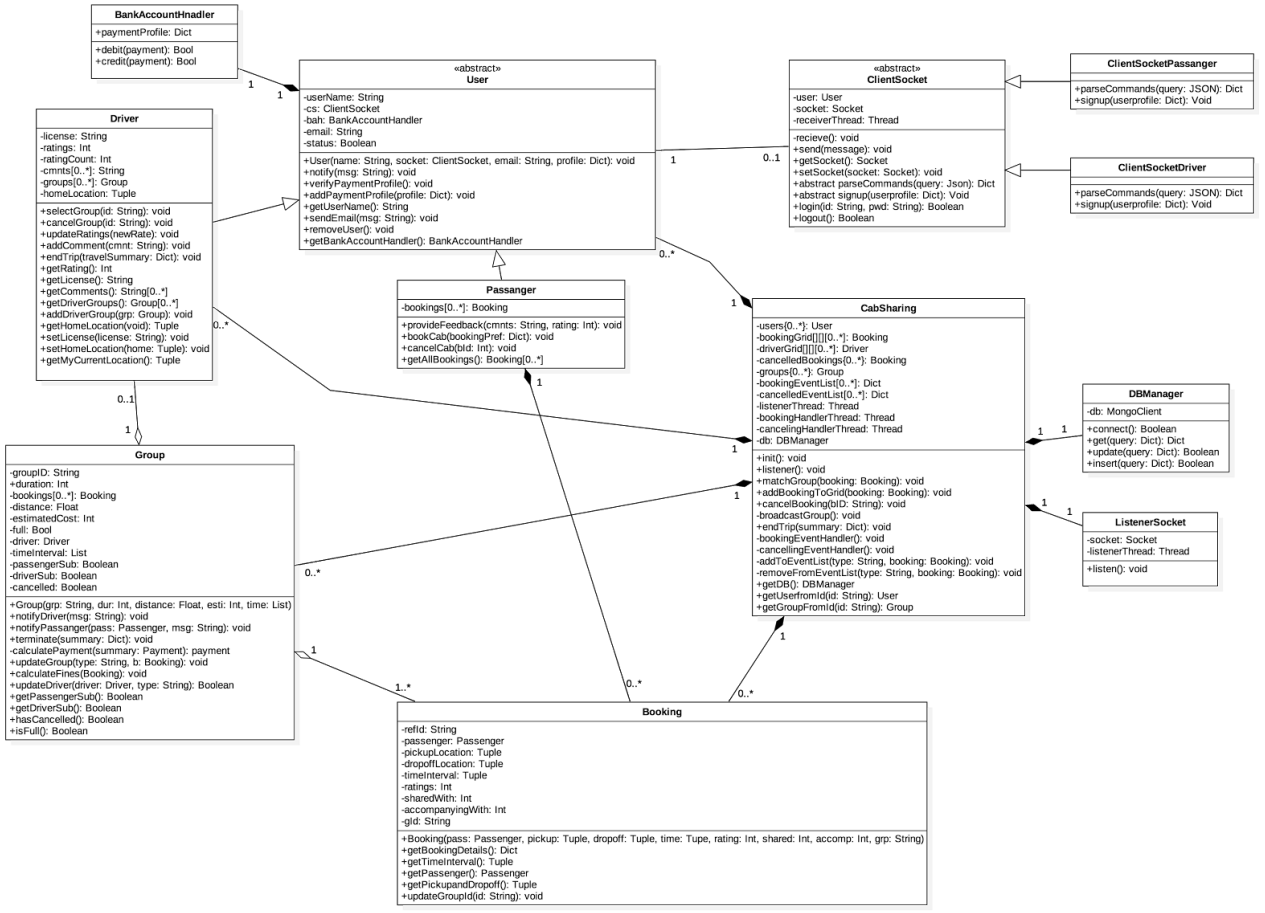


Fig 1: Class Diagram of the Server

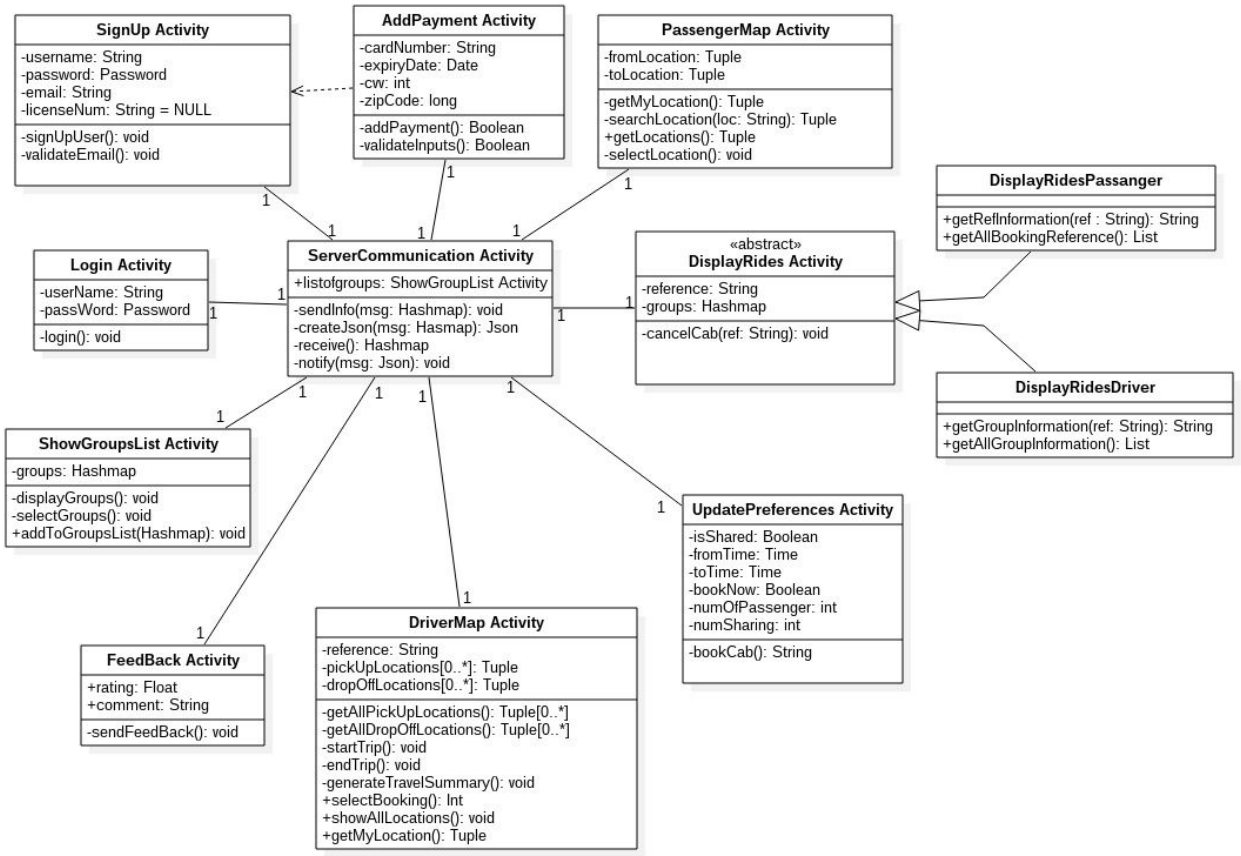


Fig 2: Class diagram of Client

Full size class diagrams can be found here.

Activity Diagrams

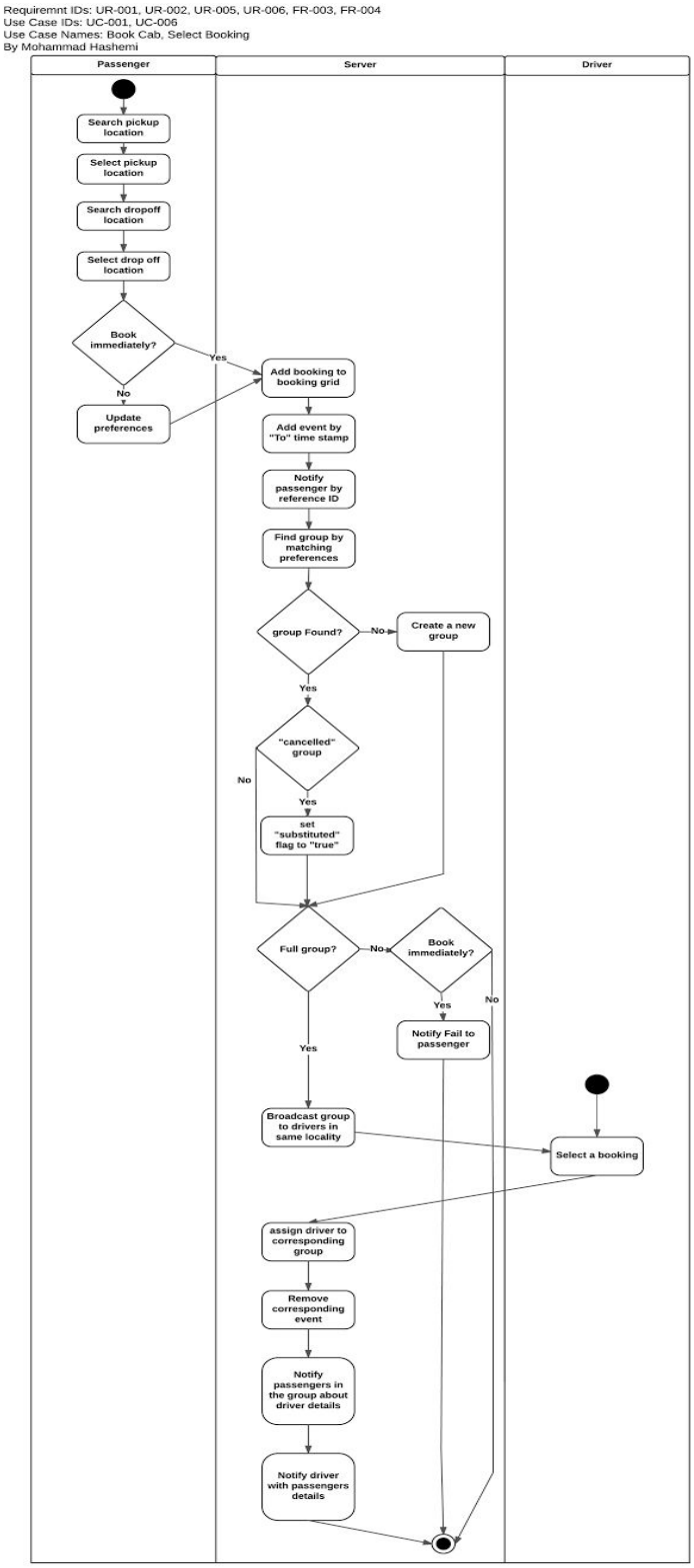


Fig1 : Activity diagram for Book Cab

Requirement Id - UR-003, FR-007, FR-009, FR-010
Use Case Id - UC-010
Use case names - Cancel Passenger Booking

Nachiket Bhagwat
nabh1518@colorado.edu

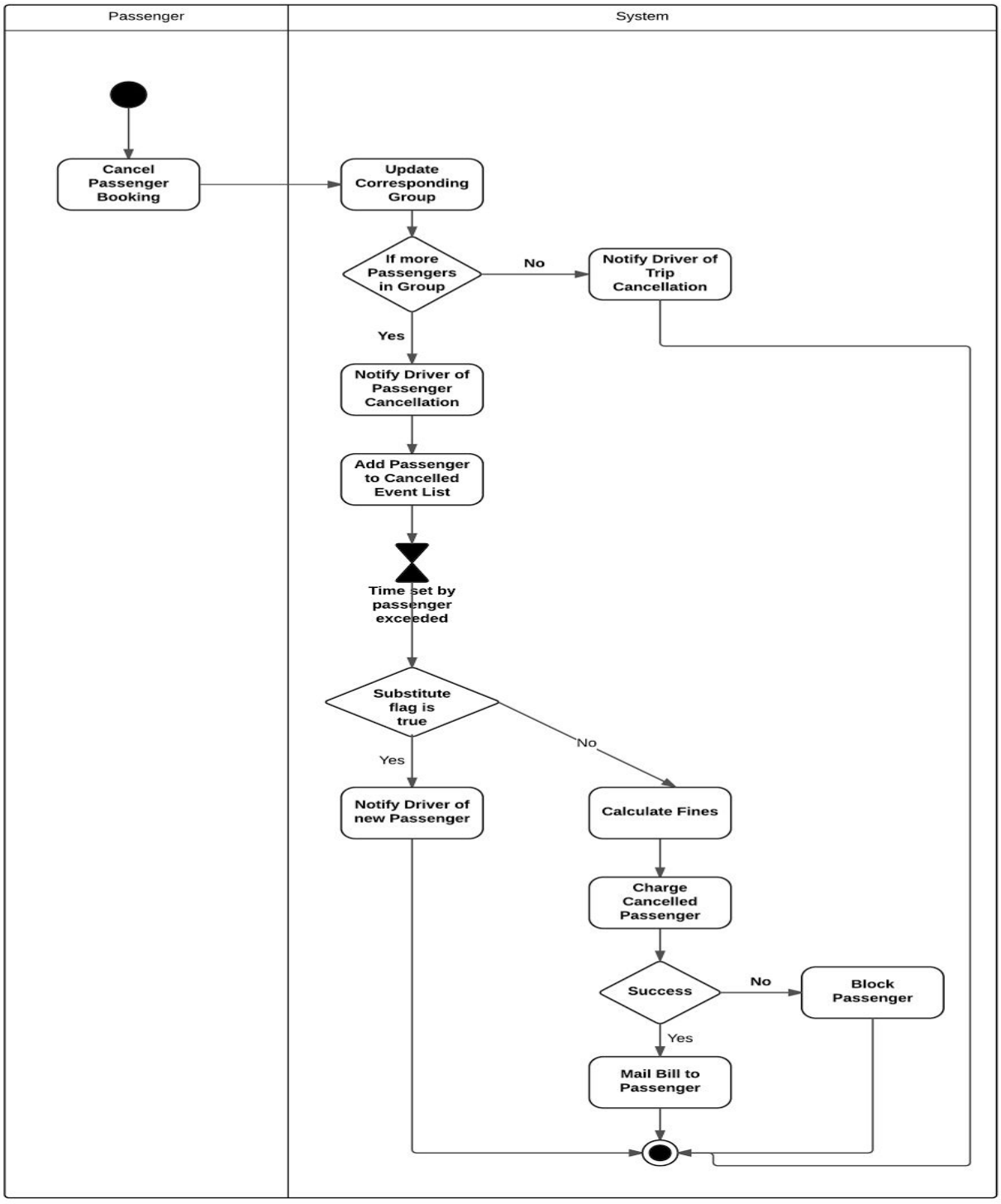


Fig 2: Activity diagram for Cancel Cab

Requirement Ids: UR-007, FR-005,FR-006,FR-010
UseCase Ids: UC-005
UseCase Naames: End Trip

Student : Praveen Kumar Devaraj

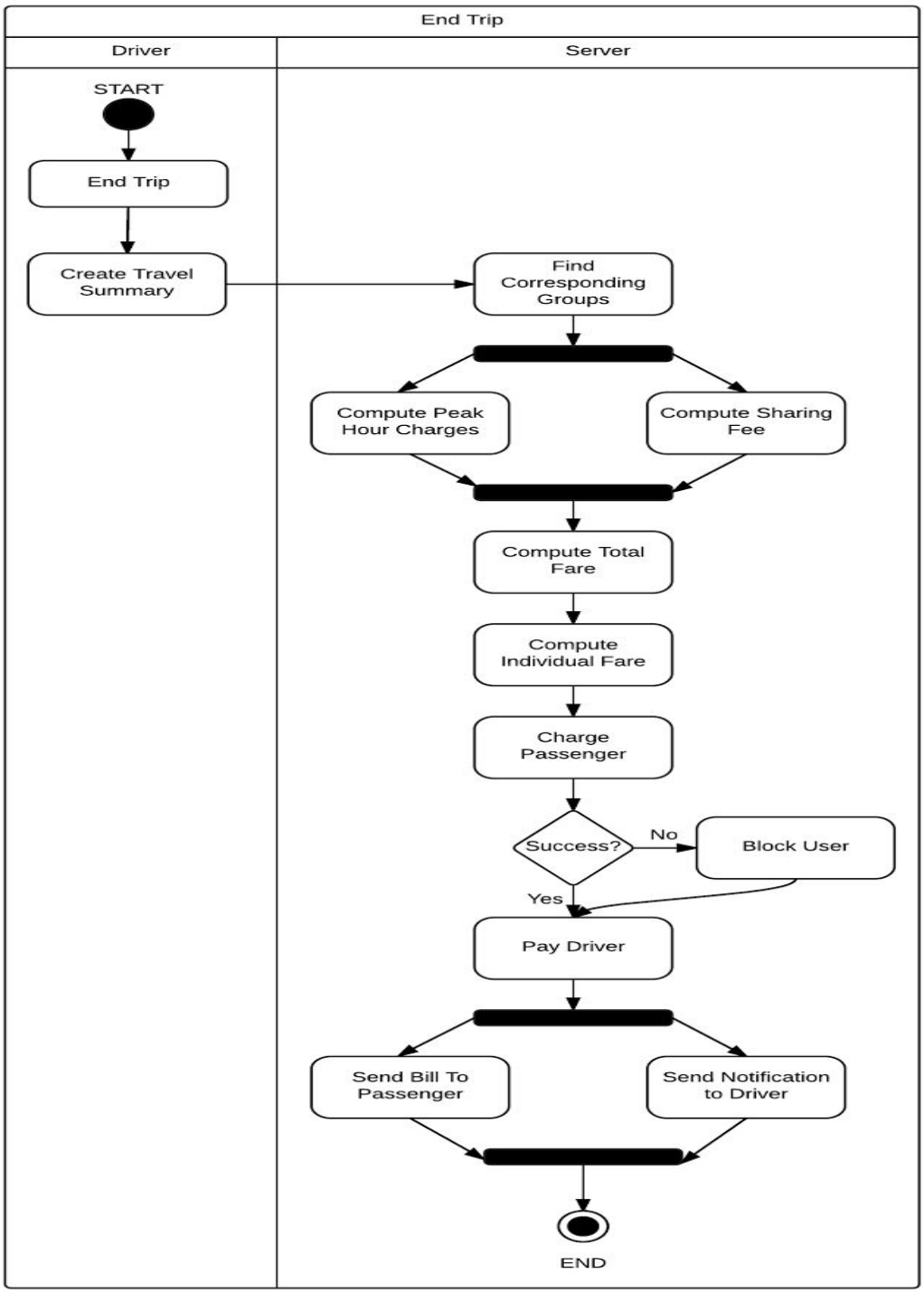


Fig 3 : Activity diagram for End Trip

Full size Activity diagrams can be found here.

Sequence Diagrams

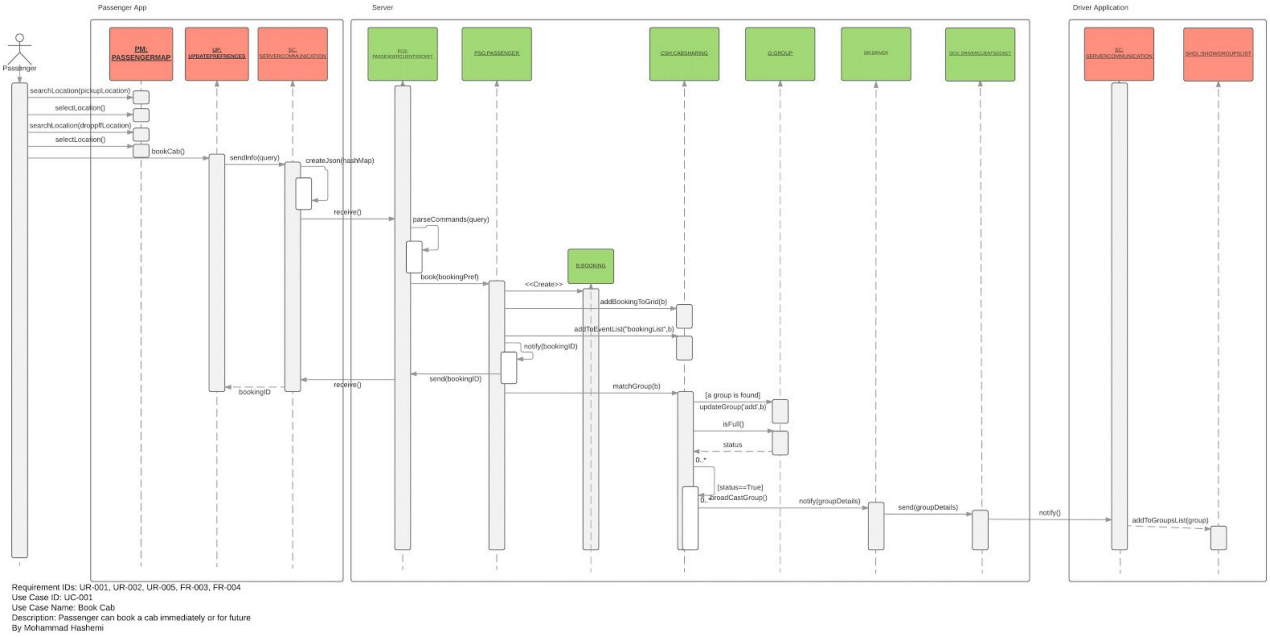


Fig 1 : Sequence diagram for Book Cab

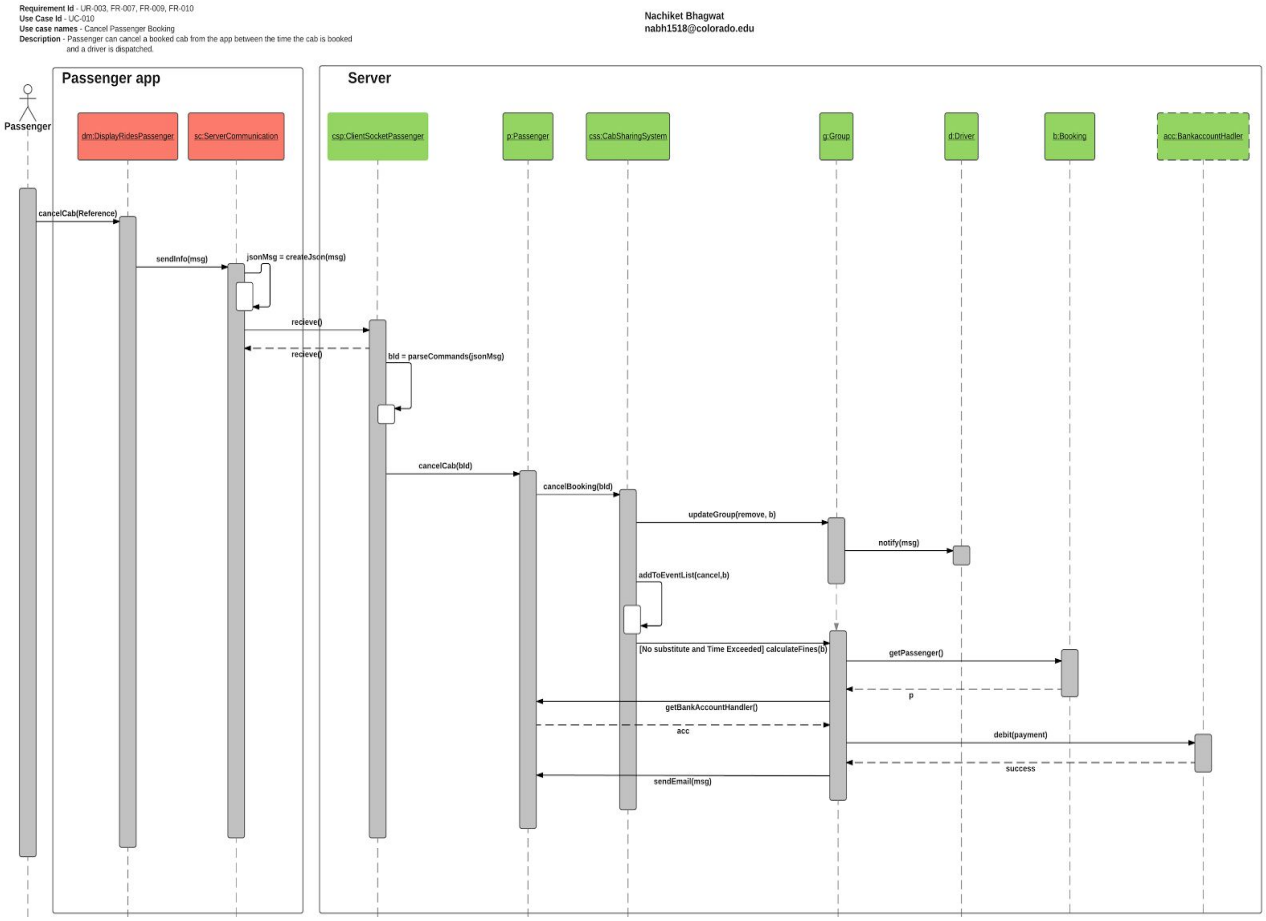
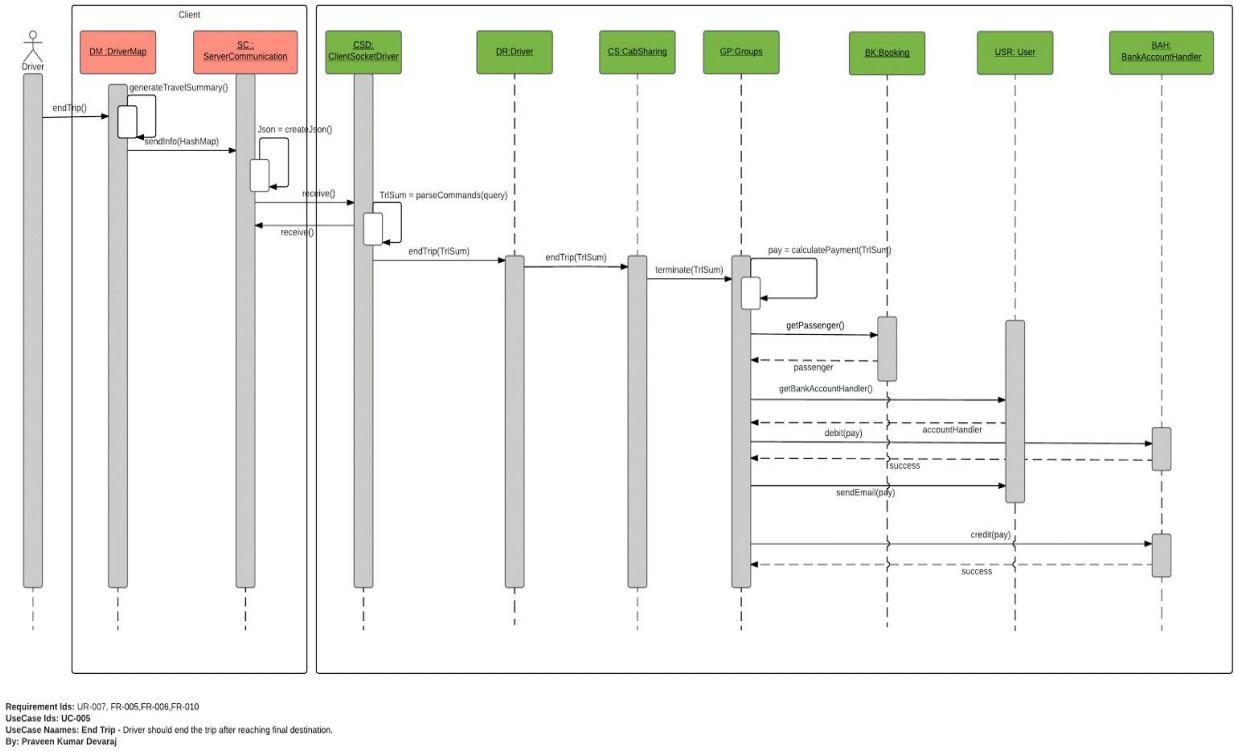


Fig 2: Sequence diagram for Cancel Cab



Requirement Ids: UR-007, FR-005,FR-006,FR-010
UseCase Ids: UC-005
UseCase Naames: End Trip - Driver should end the trip after reaching final destination.
By: Praveen Kumar Devaraj

Fig 3: Sequence diagram for End Trip

Full size Sequence diagrams can be found here.

5. UI MOCK-UPs

Full size mockups could be found here

6. Data Storage:

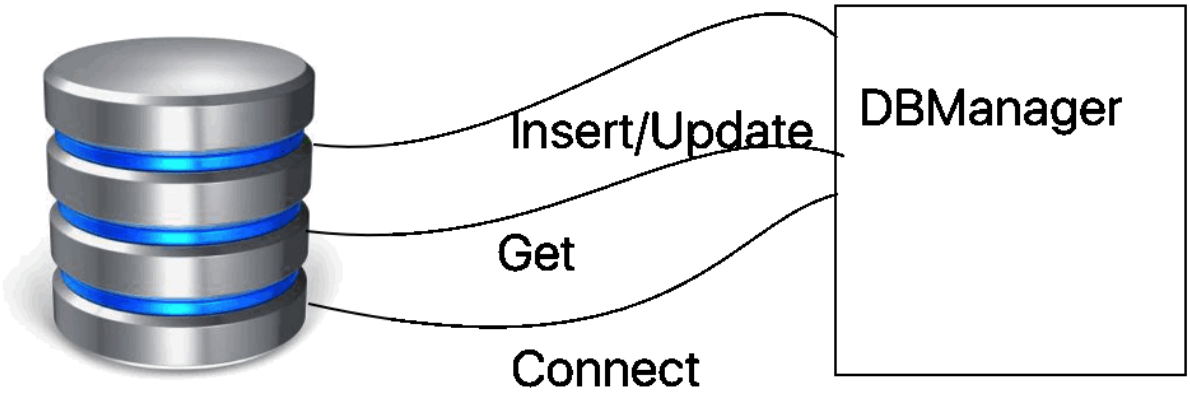
Data Storage: We will store most of the data in MongoDB in the server side on the same machine on which we run the server application. The only data that will be stored in the client machines are username and password that will be stored in Android framework. We have a DBManager class in server side that will be the only class that directly communicates with our database. All other classes should use this class to get data from the database or update some document in it. The MongoDB api for python lets us to create a dictionary and directly store it in a collection as well as get the documents as a dictionary which will make our implementation much more easier rather than using a SQL database. We decide to keep the current booking and group information in memory for performance concerns and to handle grouping of passengers as fast as possible.

We will store the User details such as username, hash of password, email, rating, driver's license during signup. We will also store the booking history after termination of any trip. We need these booking history for the extension of the system later.

Classes:

- ClientSocket class to store information about users- passengers' and drivers' details
- Group class to store information about successful trip history.
- ClientSocket class to get the information of every user when they want to log in to the system

We will use singleton pattern for DBManager class.



Contributors and Contributions:

1. User requirements, use case, class diagram - all three
2. Book Cab - Activity and Sequence diagram - Mohammad Hashemi
3. Cancel Cab - Activity and Sequence diagram - Nachiket Bhagwat
4. End Trip - Activity and Sequence diagram - Praveen Kumar Devaraj
5. Data Storage - Mohammad Hashemi and Nachiket
6. UI MockUps - Praveen Kumar Devaraj