

POLITECNICO DI MILANO

Software Engineering 2 Project

myTaxiService

Requirements Analysis and Specification Document

Simone Guidi

Matteo Imberti

November 6, 2015

Contents

1	Introduction	4
1.1	Purpose	4
1.2	Scope	4
1.3	Definitions	4
1.4	Reference Documents	5
1.5	Overview	5
2	General description	5
2.1	Product Perspective	5
2.2	Goals	5
2.3	User Characteristics	6
2.3.1	General User	6
2.3.2	Taxi driver	6
2.4	Actors Definition	7
2.5	Domain Assumptions	7
2.6	General Constraints	7
2.6.1	Regulatory Policies	7
2.6.2	Hardware Limitations	7
2.6.3	Interface to other applications	8
2.6.4	Criticality of the application	8
2.6.5	Safety and security considerations	8
3	Specific Requirements	8
3.1	External Interface Requirements	8
3.1.1	User Interfaces	8
3.1.2	Software interfaces	14
3.2	Functional Requirements	14
3.3	Use Cases	16
3.3.1	Log in	16
3.3.2	Passenger Sign up	17
3.3.3	Administrative user/Taxi driver/Dispatcher registration	17
3.3.4	Logout	20
3.3.5	Taxi request	21
3.3.6	Service request answer	22
3.3.7	Dispatcher Taxi request	23
3.3.8	Change password	24
3.3.9	Reservation request	25
3.3.10	Incoming reservation request	25
3.3.11	Set unavailable status	26
3.3.12	Set available status	27
3.4	Non-Functional Requirements	27
3.4.1	Availability	27

4	Analysis Models	28
4.1	Use Case Diagram	28
4.2	Sequence Diagrams	29
4.2.1	Login	29
4.2.2	Logout	30
4.2.3	Passenger sign up	31
4.2.4	Taxi request	32
4.2.5	Service request answer	33
4.2.6	Administrative user/Taxi driver/Dispatcher registration .	34
4.2.7	Dispatcher service request	35
4.2.8	Change password	36
4.2.9	Set available status	37
4.3	Class Diagram	38
4.4	Alloy	39
4.4.1	Signatures	39
4.4.2	Facts	41
4.4.3	Assert	42
4.4.4	Predicates	43
4.4.5	Result	43
4.4.6	Generated world	44

1 Introduction

1.1 Purpose

This document represents the Requirement Analysis and Specification Document (RASD) for the myTaxiService. The purpose of this document is to describe the system in terms of functional, non-functional requirements and system interfaces, formalize the customer's needs, show the constraints under which it must operate and the limits of the software illustrating the typical use cases that will occur in its employment and how the system will behave. This document addresses to both the stakeholders and the developers.

1.2 Scope

The myTaxiService (from now on mTS for short) is a software ecosystem aiming to optimize a large city taxi service, on both passengers and taxi drivers' perspectives.

mTS is composed by a web application addressed to the passengers and administrative users, a mobile application addressed to taxi drivers and passengers but with different functionalities based on the kind of user. On the server side, the system assigns a zone to each taxi according to its GPS. Every zone has an associated queue in which the taxis of that zone are stored.

Passengers' requests are sent to the first taxi queuing in the request zone. Passengers can make a request for a taxi through either a web application or a mobile app, sending the position for the taxi to arrive and then retrieving the code for the incoming taxi and waiting time.

Taxi drivers can inform the system with how long they will be available to fulfill a request.

1.3 Definitions

- Def 1. User: any human being interacting with the system.
- Def 2. Taxi request: a request, made by a Passenger or a Dispatcher, containing the starting position for a ride.
- Def 3. Service request: a request directed to a taxi driver from the system, demanding to accept or decline a ride.
- Def 4. Waiting time: the estimated time for a taxi to reach the starting position of a ride.
- Def 5. Ride: the event of transporting a person from a starting position to a destination position along a path.
- Def 6. Reservation request: a request, made by a Passenger or a Dispatcher, containing a starting position, a destination position and the starting time in which the ride should occur

- Def 7. Reservation service request: a request directed to a taxi driver from the system, demanding to accept or decline a reservation for a ride.
- Def 8. Reservation: an arrangement to have a taxi held for one's use at a specified time.

1.4 Reference Documents

- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.
- Assignments 1 and 2 (RASD and DD).pdf

1.5 Overview

This RASD document is subdivided in four sections:

1. Introduction: gives basic information about the structure of this document and what the myTaxiService system is about
2. General Description: explains the goals of the myTaxiService system, the actors and the design constraints
3. Specific Requirements: illustrates the function the myTaxiService system will expose to users and its requirements
4. Analysis Models: contains diagrams for a more formal understanding of the system.

2 General description

2.1 Product Perspective

The software ecosystem myTaxiService is able to operate alongside existing phone based management systems.

2.2 Goals

- G1. A passenger should be able to request a taxi at a position and be provided with information about the incoming taxi
- G2. A passenger should be able to reserve a taxi with 2 hours in advance
- G3. A passenger requesting a reservation should be able to get a reservation confirmation
- G4. Taxi drivers should be able to receive service requests and accept or decline

- G5. Taxi drivers should be able to notify the system with an amount of hours in which they declare to be available
- G6. The system should be able to handle taxis using queue based priority mechanism
- G7. Registered users should be able to login to an existing profile
- G8. Guest users should be able to create Passenger profiles
- G9. Administrative users should be able to create Dispatcher/Taxi driver/Administrative user profiles
- G10. The system should be able to allocate a taxi 10 minutes before a scheduled reservation

2.3 User Characteristics

We have identified four potential classifications of users of our system:

- Taxi drivers: These users are workers with licensed taxi driver. They will use the system as a mean of getting work request and schedule their availability
- Passengers: These users are people requesting taxi. They will use the system to request a taxi to a specified position or to reserve a taxi
- Administrative users: These users are people managing the creation process of workers profile
- Dispatchers: These users are workers able to manually insert taxi requests in the system

2.3.1 General User

All users can be assumed to have the following characteristics:

- Ability to read and understand English.
- Ability to interface themselves with a web browser or a mobile application
- Beyond the above, no further facility with computer technology can be assumed.

2.3.2 Taxi driver

The taxi driver user can be assumed to have the following characteristics:

- Regular taxi license according to local laws

2.4 Actors Definition

- A1. Guest: A user that access to the system with a not logged in status.
- A2. Registered user: A user that has performed a login in the system with a generic profile.
- A3. Passenger: A user that logged in the system with a Passenger profile.
- A4. Administrative user: A user that logged in the system with an Administrative user profile.
- A5. Dispatcher: A user that logged in the system with a Dispatcher profile.
- A6. Taxi driver: A user that logged in the system with a Taxi driver profile.

2.5 Domain Assumptions

- D1. The city is divided in taxi zones
- D2. Taxi drivers are able to signal their position via GPS
- D3. Taxi drivers can freely move into any taxi zone, except they are assigned reservation
- D4. A call center is active 24/7 to handle requests via phone call for existing phone based taxi management
- D5. A mapping service is available offering real-time traffic conditions and route planning
- D6. Taxi drivers status can be modelled to be either be available, unavailable or reserved.
- D7. GPS tracking information are always available

2.6 Gernerall Constraints

2.6.1 Regulatory Policies

- myTaxiService has to undergo with city taxi regulations

2.6.2 Hardware Limitations

- 1. The mobile application of myTaxiService system requires OS among: iOS 8+, Android 4+
- 2. The web application of myTaxiService system requires a web browser with HTML4+ capability
- 3. The mobile application of myTaxiService requires access to the Internet

2.6.3 Interface to other applications

- Google maps API

2.6.4 Criticality of the application

- Taxi malfunctions may change waiting times in unpredictable ways
- Excessive service requests declining from the taxi driver may cause delay or aborting of the taxi request
- If no available taxis are queuing in the zone corresponding to a passengers request or every taxi in the zone decline the request, the system may forward the request to queue of the near taxi zones
- The time from the beginning of (3.3.5) use case to his completion cannot exceed 3 minutes

2.6.5 Safety and security considerations

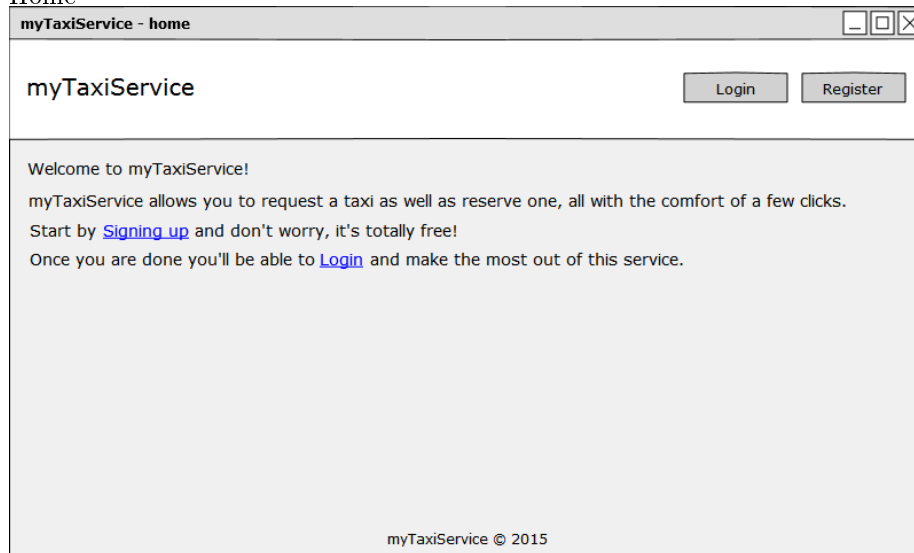
- Profile password will be crypted
- myTaxiService has to undergo privacy related laws

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

- Home



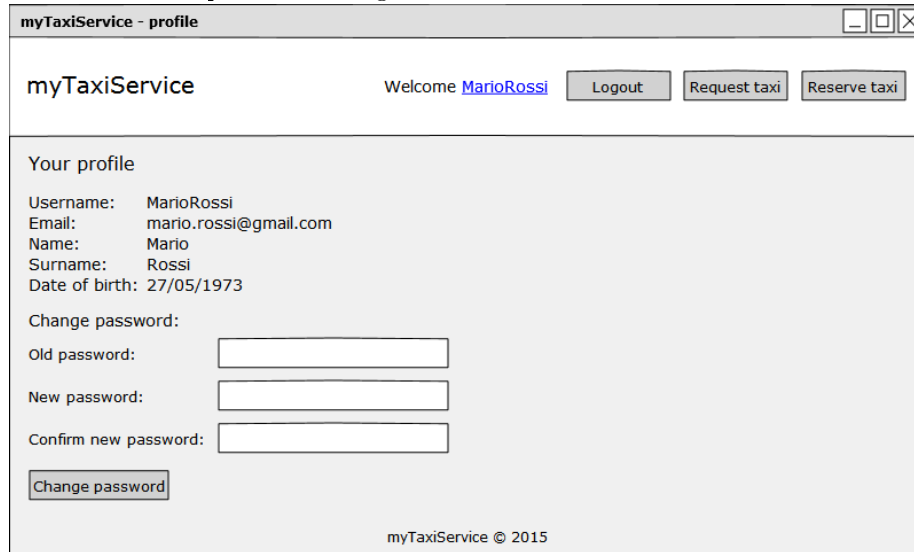
- Login ref. 3.3.1

The screenshot shows a web browser window titled "myTaxiService - login". The page has a header with the "myTaxiService" logo on the left and "Login" and "Register" buttons on the right. The main content area is titled "Login to your account" and contains a "Username:" label with a text input field, a "Password:" label with a text input field, and a "Login" button below them. Below the login fields, there is a link: "Not yet registered? [Sign up](#) now!". At the bottom right of the page, the text "myTaxiService © 2015" is displayed.

- Logout ref. 3.3.4

The screenshot shows a web browser window titled "myTaxiService - logout". The page has a header with the "myTaxiService" logo on the left and "Login" and "Register" buttons on the right. The main content area displays the message: "You were succesfully logged out.
Thank you for using myTaxiService!". At the bottom right of the page, the text "myTaxiService © 2015" is displayed.

- Profile view and password change ref. 3.3.8



The screenshot shows a web browser window titled "myTaxiService - profile". The page header includes the "myTaxiService" logo, a welcome message "Welcome MarioRossi", and three buttons: "Logout", "Request taxi", and "Reserve taxi". The main content area is titled "Your profile" and displays the following information:

- Username: MarioRossi
- Email: mario.rossi@gmail.com
- Name: Mario
- Surname: Rossi
- Date of birth: 27/05/1973

Below the profile information is a "Change password" section with three input fields: "Old password:", "New password:", and "Confirm new password:". A "Change password" button is located at the bottom of this section. The footer of the page reads "myTaxiService © 2015".

- Register ref. 3.3.2

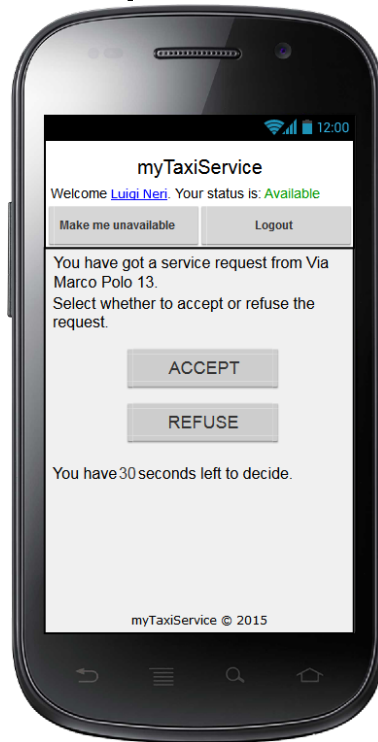


The screenshot shows a web browser window titled "myTaxiService - register". The page header includes the "myTaxiService" logo and two buttons: "Login" and "Register". The main content area is titled "Register a new Passenger profile" and contains the following registration form:

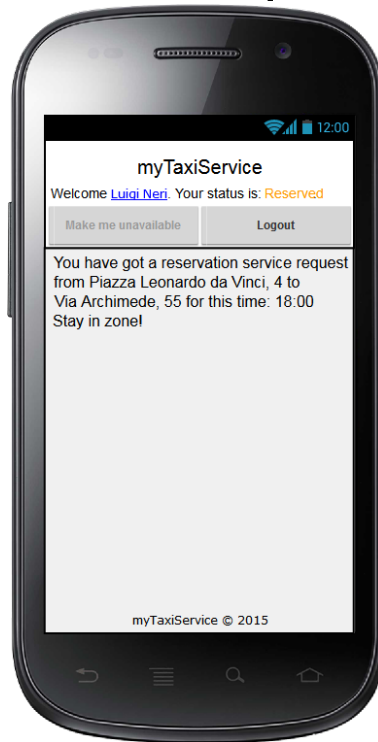
- Username:
- Name and surname:
- Date of birth: Day Month Year
- Email:
- Password:
- Confirm password:

A "Sign up" button is located at the bottom of the form. The footer of the page reads "myTaxiService © 2015".

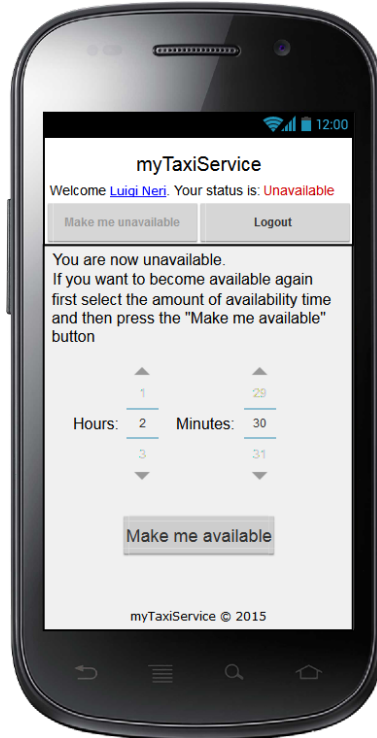
- Service request ref.3.3.6



- Reservation service request ref.3.3.10



- Set available state ref.3.3.12



- Taxi request ref.3.3.5

myTaxiService - taxi request

myTaxiService Welcome MarioRossi Logout Request taxi Reserve taxi

Insert the address of your location. Remember that you have to be already there as a taxi will arrive as soon as possible. If you need to reserve a taxi use the appropriate [section](#).

Address:

Request taxi

myTaxiService © 2015

- Taxi reservation 3.3.9

The screenshot shows a web browser window titled "myTaxiService - taxi reservation". The page has a header with the "myTaxiService" logo, a welcome message "Welcome MarioRossi", and three buttons: "Logout", "Request taxi", and "Reserve taxi". The main content area contains a message: "You can reserve a taxi with at least 2 hours in advance and at most 12. Need a taxi now? Use our request taxi [section](#)." Below this is a prompt: "Insert the starting address, the destination address and the time of your reservation:". There are three input fields: "Starting address:", "Destination address:", and "Time:". The "Time:" field has three dropdown menus labeled "Day", "Hour", and "Minute". A "Reserve taxi" button is located below the input fields. The footer of the page says "myTaxiService © 2015".

3.1.2 Software interfaces

The system links with Google Maps Directions API in order to compute travel times of taxis and Google Maps Geolocalization API and Google Maps Geocoding API in order to get taxis position.

3.2 Functional Requirements

- R1. A passenger should be able to request a taxi at a position and be provided with information about the incoming taxi G1
 - (a) The system provides a passenger request form
 - (b) The system shows the taxi ID of an incoming taxi and the waiting time (obtained by mean of D5)
- R2. A passenger should be able to reserve a taxi with 2 hours in advance G2
 - (a) The system provides a reservation request form
- R3. A passenger requesting a reservation should be able to get a reservation confirmation G3
 - (a) The system sends a confirmation message
- R4. Taxi drivers should be able to receive service requests and accept or decline them G4
 - (a) The system provides a service request acceptance form

- (b) The Taxi driver uses the provided form to communicate his decision to the system
 - (c) A taxi driver is notified for an incoming service request if and only if he is first in his queue
- R5. Taxi drivers should be able to notify the system with an amount of hours in which they declare to be available G5
 - (a) The system provides a form for declaring an amount of hours in which they declare to be available if and only if a taxi driver status is unavailable
- R6. The system should be able to handle taxis using queue based priority mechanism G6
 - (a) A taxi driver id is stored in a queue if and only if his status is either available or reserved
 - (b) A taxi driver declaring himself available enters the queue in the last position.
 - (c) If the amount of hours in which the taxi driver has declared to be available expires, the taxi driver id is removed from the queue and his status is changed to unavailable
 - (d) If a taxi driver declines the service request, he is shifted to the last position of his queue
 - (e) If a taxi driver accepts, he is removed from his queue and his status is changed to unavailable
 - (f) If a taxi driver moves toward a different zone, he is removed from his previous queue and inserted in the last position of the queue associated to his new taxi zone
- R7. Registered users should be able to login to an existing prole G7
 - (a) The system provides a login form
 - (b) The system authenticates credentials
- R8. Guest users should be able to create Passenger proles G8
 - (a) The system provides a sign up form
 - (b) The system validates credentials
 - (c) The system stores credentials
- R9. Administrative users should be able to create Dispatcher/Taxi driver/Administrative user proles G9
 - (a) The system provides a profile creation form

- (b) The profile creation form is accessible if and only if the user requesting the form is an Administrative user.

R10. The system should be able to allocate a taxi 10 minutes before a scheduled reservation G10

- (a) When a reservation occurs, the status of the first taxi driver set to available is changed to reserved, his taxi id is associated to a reservation and the taxi driver is notified
- (b) The system keeps a taxi identification associated to each reservation request at any time

3.3 Use Cases

3.3.1 Log in

Name	Log in
Description	This use case describes how a user logs into the myTaxiService.
Actors	Guest, Registered user
Entry condition	None
Flow of events	<ul style="list-style-type: none"> • The system requests that the actor enter his/her username and password. • The actor enters his/her username and password. • The system validates the entered username and password and logs the actor into the system.
Exit conditions	If the use case was successful, the actor is now logged into the system as Registered user; if not, the system state is unchanged.
Exceptions	<ul style="list-style-type: none"> • If in the Basic Flow the actor enters credentials not in the system, an error message is displayed and the basic flow is restarted.

3.3.2 Passenger Sign up

Name	Passenger sign up
Description	This use case allows a Guest to register a Passenger profile in the myTaxiService.
Actors	Guest
Entry condition	None
Flow of events	<ul style="list-style-type: none">• The system requests the actor to enter username, password, email, name, surname, phone number and date of birth.• The actor enters username, password, email, name, surname, phone number and date of birth.• The system validates the entered credentials and provides the actor with an activation URL through a message to the actor's email.• As the user reach the activation URL the Passenger profile is created
Exit conditions	If the case was successful, the Passenger profile is created; if not, the system state is unchanged.
Exceptions	<ul style="list-style-type: none">• If in the basic flow the actor inserts invalid credentials, an error message is displayed and the basic flow is restarted.• If a timeout expires after sending the activation URL, the use case fails.

3.3.3 Administrative user/Taxi driver/Dispatcher registration

Name	Administrative user/Taxi driver/Dispatcher registration
Description	This use case allows an Administrative user to register one profile among Taxi driver, Dispatcher and Administrative user in the myTaxiService.
Actors	Administrative user
Entry condition	None

Flow of events	<ul style="list-style-type: none"> • The system requests the actor to specify what kind of user is going to be created. • The actor enters the kind of user is going to be created. • If the kind of user is “Taxi driver”, the “Taxi driver registration” subflow is executed. • If the kind of user is “Dispatcher”, the “Dispatcher registration” subflow is executed. • If the kind of user is “Administrative User”, the “Administrative user registration” subflow is executed.
Taxi driver registration subflow	<ul style="list-style-type: none"> • The system requests the actor to enter username, password, name, surname, date of birth, taxi ID and phone number. • The actor inserts username, password, name, surname, date of birth, taxi ID and phone number. • The system validates the inserted credentials. • The Taxi driver profile is created
Dispatcher registration subflow	<ul style="list-style-type: none"> • The system requests the actor to enter username, password, name, surname and date of birth. • The actor inserts username, password, name, surname and date of birth. • The system validates the inserted credentials. • The Dispatcher profile is created.

Administrative user registration subflow	<ul style="list-style-type: none"> • The system requests the actor to enter username, password, name, surname and date of birth. • The actor inserts username, password, name, surname and date of birth. • The system validates the inserted credentials. • The Administrative user profile is created
Exit conditions	If the case was successful, the Taxi Driver/Dispatcher/Administrative user profile is create; if not, the system state is unchanged.
Exceptions	<ul style="list-style-type: none"> • If in the “Taxi driver registration” subflow the actor inserts invalid credentials, an error message is displayed and the “Taxi driver registration” subflow is restarted. • If in the “Administrative user registration” subflow the actor inserts invalid credentials, an error message is displayed and the “Administrative user registration” subflow is restarted. • If in the “Dispatcher registration” subflow the actor inserts invalid credentials, an error message is displayed and the “Dispatcher registration” subflow is restarted.

3.3.4 Logout

Name	Logout
Description	This use case allows a Registered user to logout from the myTaxiService.
Actors	Registered user
Entry condition	None
Flow of events	<ul style="list-style-type: none">• The actor request the system to logout• The system acknowledges the actor logout• The actor is logged out from the system
Exit conditions	If the case was successful, the actor is now recognized as Guest; if the case fails, the system state is unchanged
Exceptions	<ul style="list-style-type: none">• None

3.3.5 Taxi request

Name	Taxi request
Description	This use case allows a Passenger to request a Taxi at a desired position.
Actors	Passenger
Entry condition	None
Flow of events	<ul style="list-style-type: none">• The system requests the actor to enter a position.• The actor enters the position.• The system validates the entered position.• The system handles the Taxi request.• The actor is provided with incoming “taxi id” and estimated waiting time.
Exit conditions	If the case was successful, a service request, and a ride associated to the actor are created; if not, the system state is unchanged.
Exceptions	<ul style="list-style-type: none">• If in the basic flow the actor insert an invalid position, an error message is displayed and the basic flow is restarted.• If the system is unable to handle the request, the flow fails.

3.3.6 Service request answer

Name	Service request answer
Description	This use case allows a Taxi driver to receive a service request and accept or decline the service request.
Actors	Taxi driver
Entry condition	<ul style="list-style-type: none">• The system receives a Taxi request and send a service request to the actor.• The actor has the highest priority in his taxi zone.
Flow of events	<ul style="list-style-type: none">• The actor receives the service request.• The system request the actor to accept or decline the service request.• The actor accept or decline the service request.• The system acknowledges actor's input.• The system signals the actor with a confirmation message.• The actor receive the confirmation message.
Exit conditions	If the use case was successful and the actor accepted the service request, the actor is removed from his queue and the actor status becomes unavailable; If the use case was successful and the actor declined the service request, the actor is moved to the last position of his queue; if the use case fails, the actor is moved to the last position of his queue.
Exceptions	<ul style="list-style-type: none">• If the use case exceed a fixed timeout, the use case fails, and a message is displayed to the actor and the flow fails.

3.3.7 Dispatcher Taxi request

Name	Dispatcher Taxi request
Description	This use case allow a Dispatcher to create a service request.
Actors	Dispatcher
Entry condition	None
Flow of events	<ul style="list-style-type: none">• The system request the actor to insert a Taxi request data.• The actor inserts Taxi request data.• The system validates the inserted data.• The system handles the Taxi request
Exit conditions	If the use case was successful, a service request is created; If not, the system state is unchanged
Exceptions	<ul style="list-style-type: none">• If the system is unable to handle the Taxi request, the flow fails.• If the Taxi request data inserted by the actor are invalid, an error message is displayed and the basic flow is restarted.

3.3.8 Change password

Name	Change password
Description	This use case allow a registered user to change the password of his profile.
Actors	Registered user
Entry condition	None
Flow of events	<ul style="list-style-type: none">• The system request the actor to insert the current password of his profile and the new password.• The actor inserts the current password of his profile and the new password.• The system validates the fields.
Exit conditions	If the basic flow was successful, the new password inserted from the user replace the previous password of the actor's profile; If the basic flow fails, the system is unchanged.
Exceptions	<ul style="list-style-type: none">• If the current password inserted by the actor in the basic flow does not actually match the current password, an error message is displayed and the basic flow is restarted.

3.3.9 Reservation request

Name	Reservation request
Description	This use case allow a Passenger to make a Reservation request
Actors	Passenger
Entry condition	None
Flow of events	<ul style="list-style-type: none">• The system request the actor to insert the starting position, destination position and the starting time for a ride.• The actor inserts the starting position, destination position and starting time for the ride.• The system validates the information inserted by the actor.• The system handle the reservation request.• The system sends a confirmation message to the actor.• The actor receive the confirmation message.
Exit conditions	If the basic flow was successful, a reservation service request is created in the system; If not, the system is unchanged.
Exceptions	<ul style="list-style-type: none">• If the starting time inserted by the actor in the basic flow is invalid, an error message is displayed and the basic flow is restarted.• If the starting position inserted by the actor in the basic flow is invalid, an error

3.3.10 Incoming reservation request

Name	Incoming reservation request
Description	This use case allow a taxi driver to receive a notification for his temporary assignment to a reservation.
Actors	Taxi driver
Entry condition	<ul style="list-style-type: none">• The actor state is available

Flow of events	<ul style="list-style-type: none"> • The system send a notification message to the actor. • The actor receives the notification.
Exit conditions	If the basic flow is successful, a reservation service request is created in the system; If it fails, the system is unchanged.
Exceptions	None

3.3.11 Set unavailable status

Name	Set unavailable status
Description	This use case allows a Taxi driver to set his status to unavailable
Actors	Taxi driver
Entry condition	<ul style="list-style-type: none"> • The actor status is available
Flow of events	<ul style="list-style-type: none"> • The system request the actor to confirm his status change. • The actor confirms his status change. • The system processes the status change. • The system send a confirmation to the actor. • The actor receives confirmation.
Exit conditions	If the use case was successful, the actor status is changed to unavailable and he is removed from any queue; if not, the system is unchanged
Exceptions	None

3.3.12 Set available status

Name	Set available status
Description	This use case allows a Taxi driver to change his status to available for a specified time amount.
Actors	Taxi driver
Entry condition	<ul style="list-style-type: none">• The actor state is unavailable
Flow of events	<ul style="list-style-type: none">• The system requests the actor to insert the time amount of his availability• The actor inserts the time amount of his availability.• The system processes the status change.• The system sends a confirmation to the actor.• The actor receives the confirmation.
Exit conditions	If the use case was successful, the actor status is changed to available and he is inserted in the last position of the appropriate queue; if not, the system is unchanged
Exceptions	None

3.4 Non-Functional Requirements

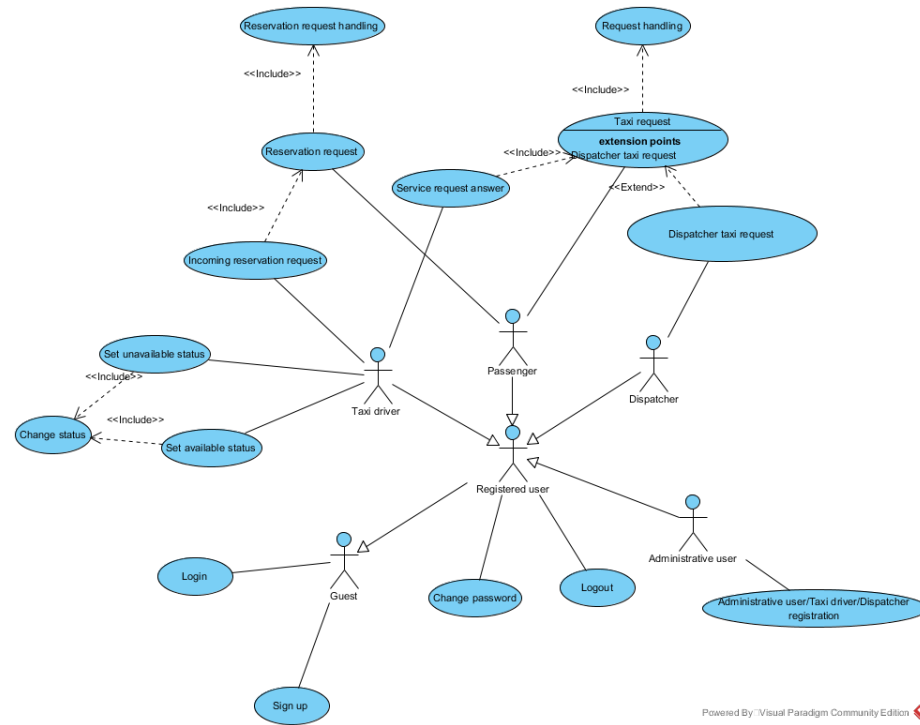
3.4.1 Availability

- The system should be able to accept taxi requests and reservation requests

24/7

4 Analysis Models

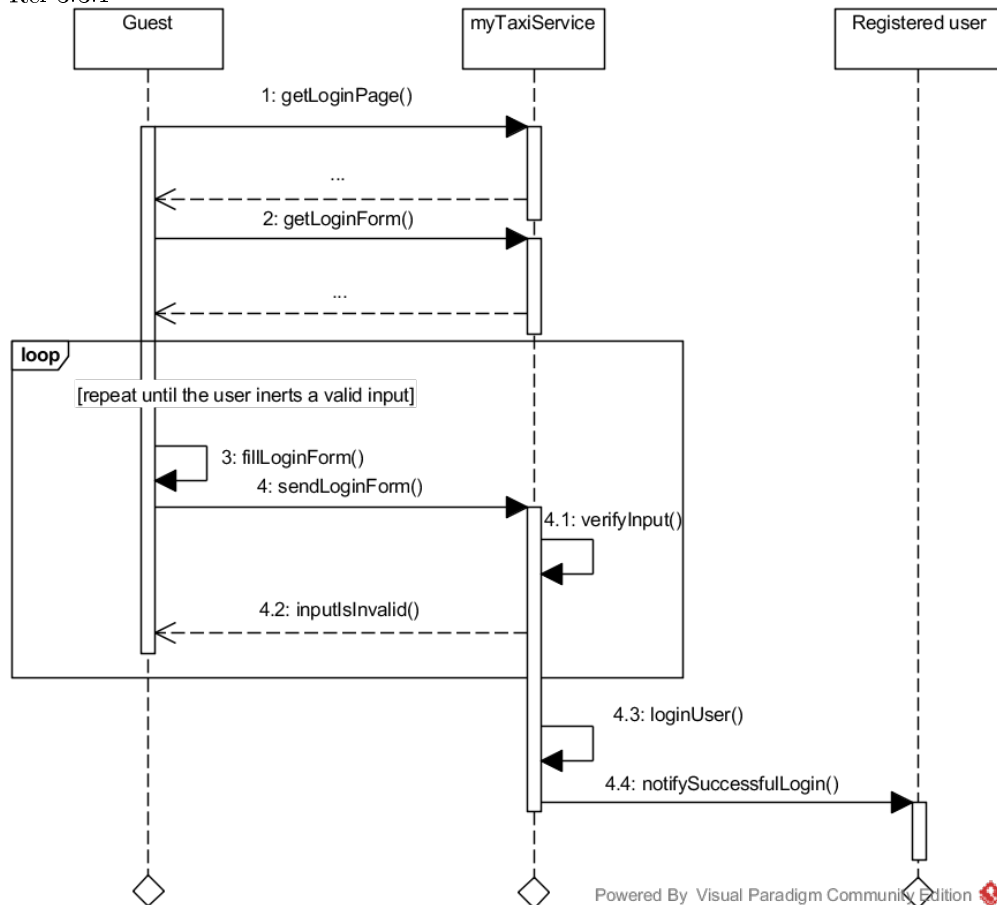
4.1 Use Case Diagram



4.2 Sequence Diagrams

4.2.1 Login

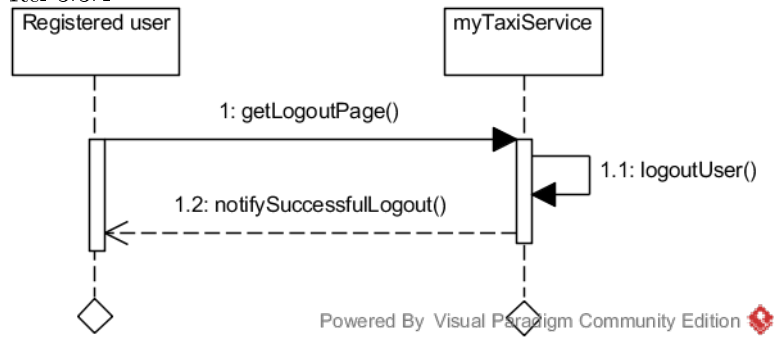
Ref 3.3.1



Powered By Visual Paradigm Community Edition

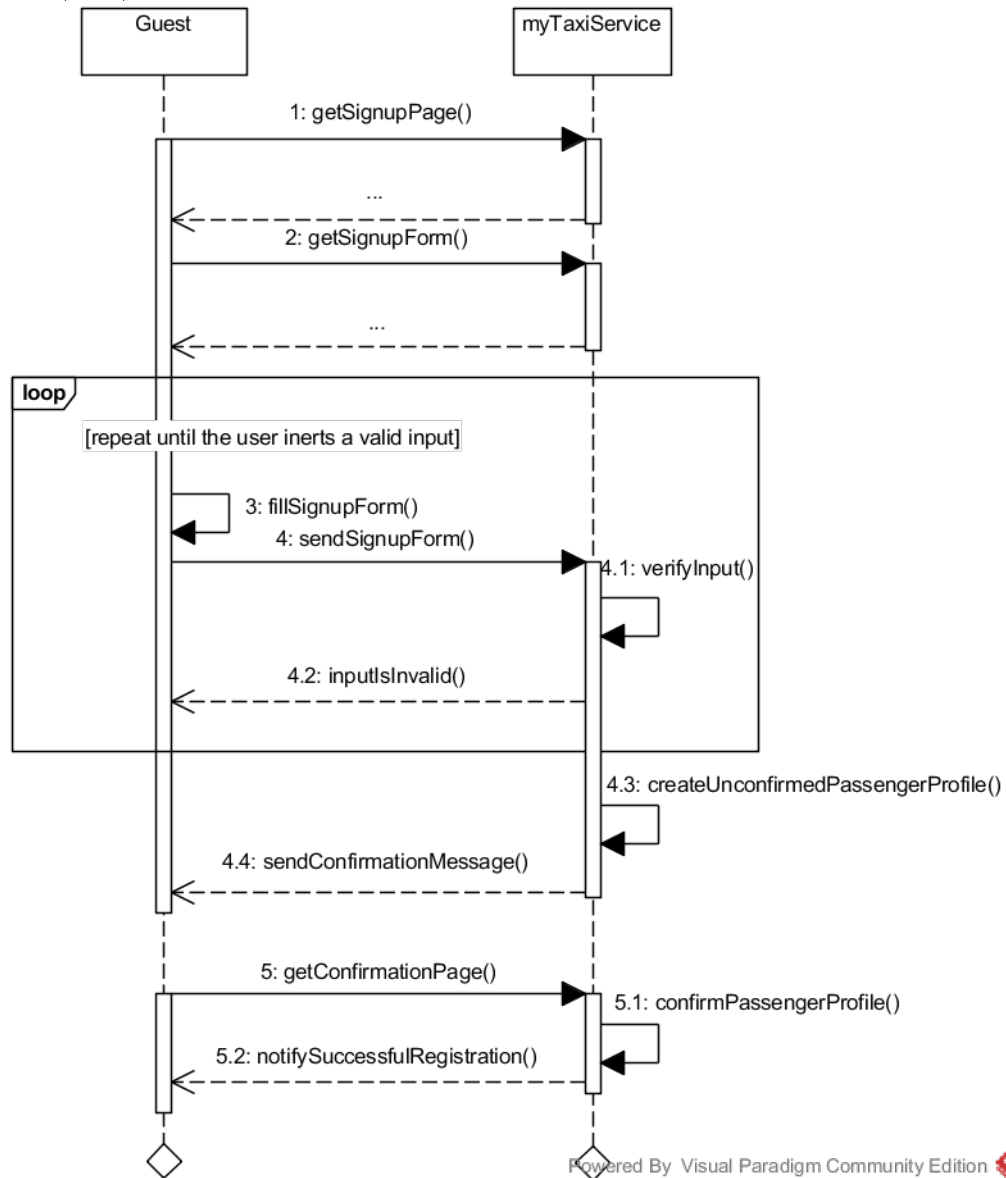
4.2.2 Logout

Ref 3.3.4



4.2.3 Passenger sign up

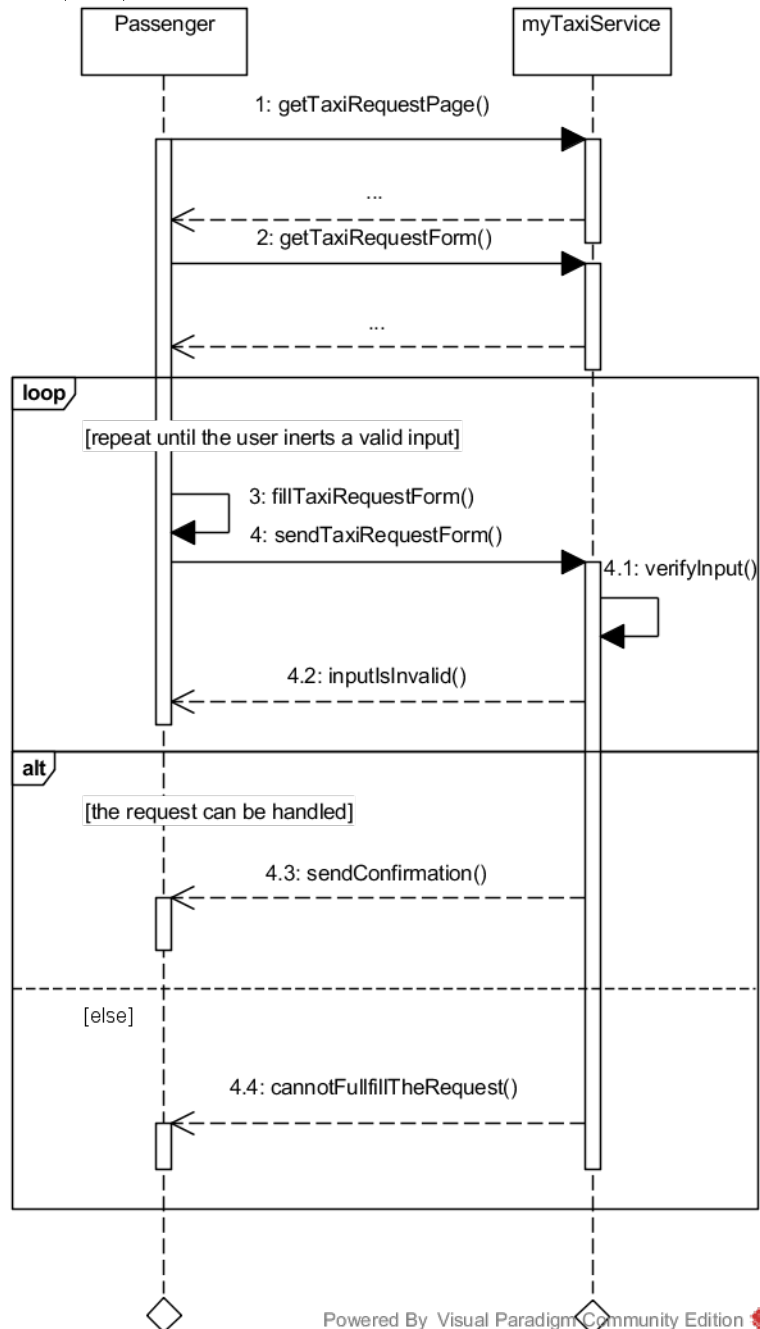
Ref (3.3.2)



Powered By Visual Paradigm Community Edition

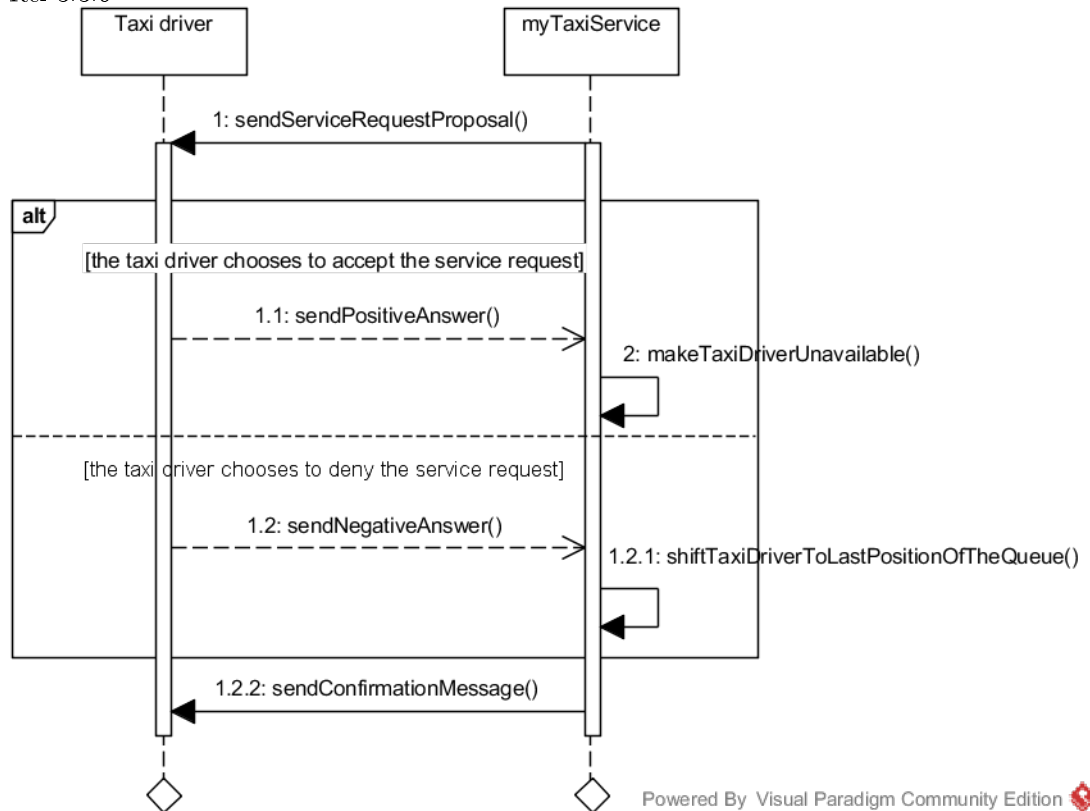
4.2.4 Taxi request

Ref (3.3.5)



4.2.5 Service request answer

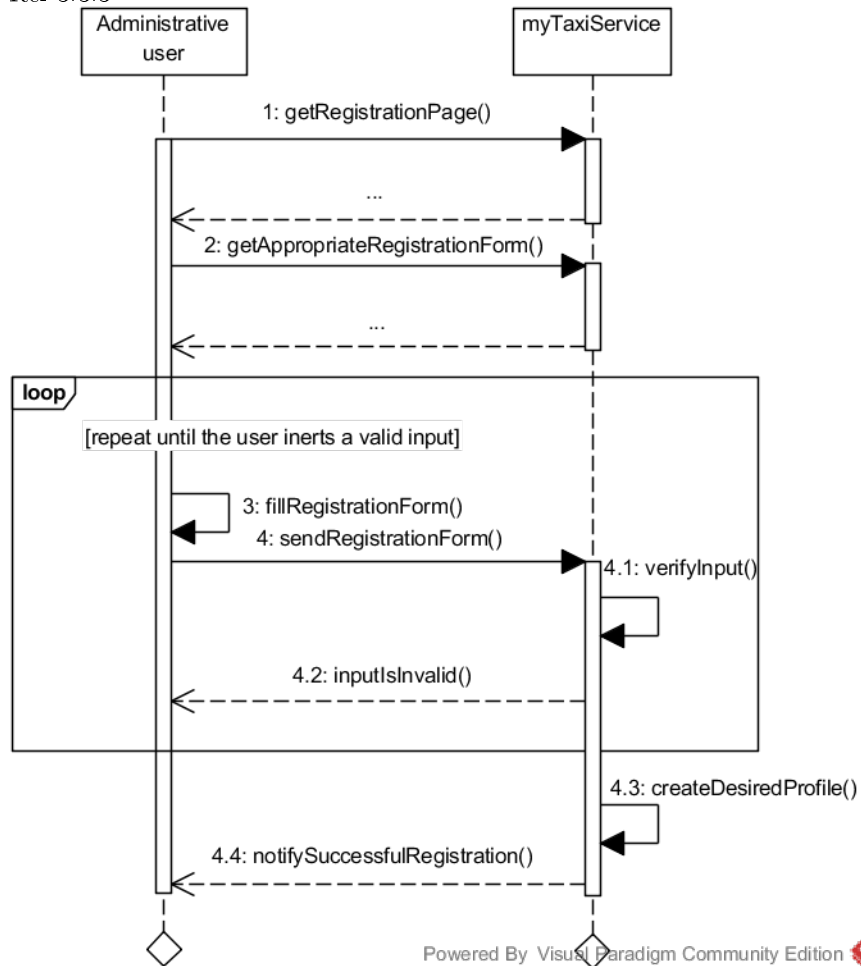
Ref 3.3.6



Powered By Visual Paradigm Community Edition

4.2.6 Administrative user/Taxi driver/Dispatcher registration

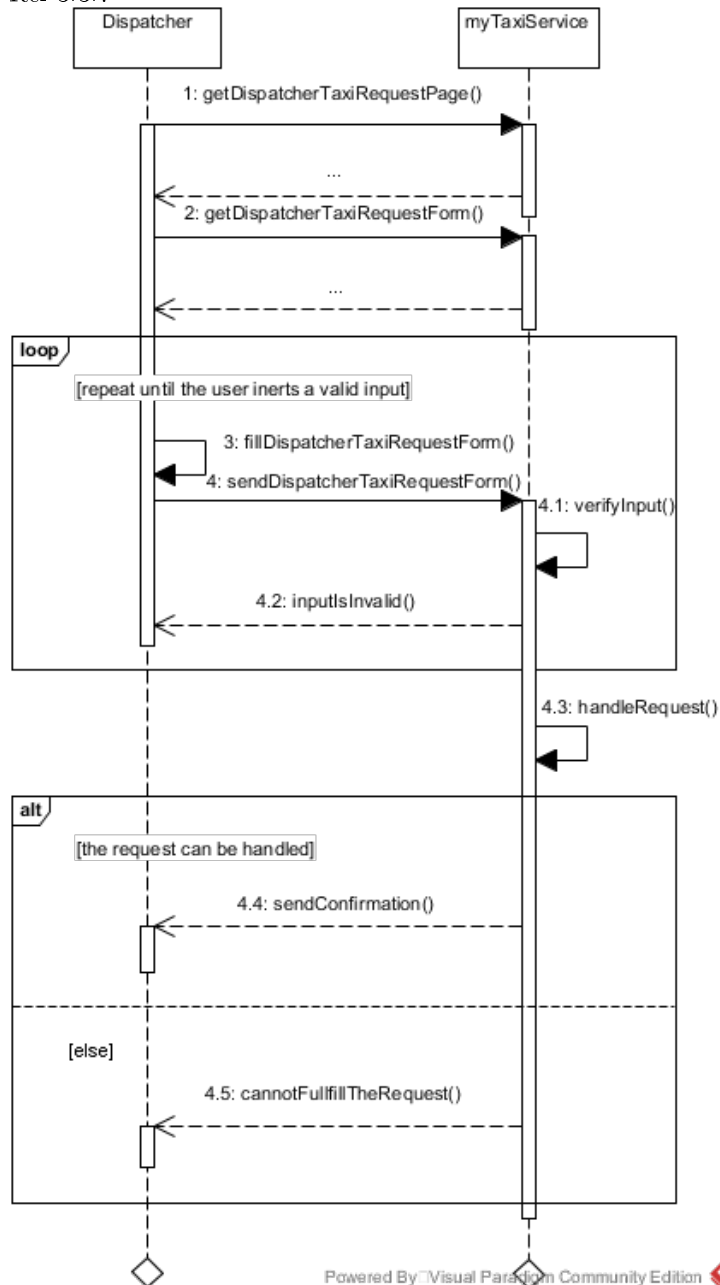
Ref 3.3.3



Powered By Visual Paradigm Community Edition

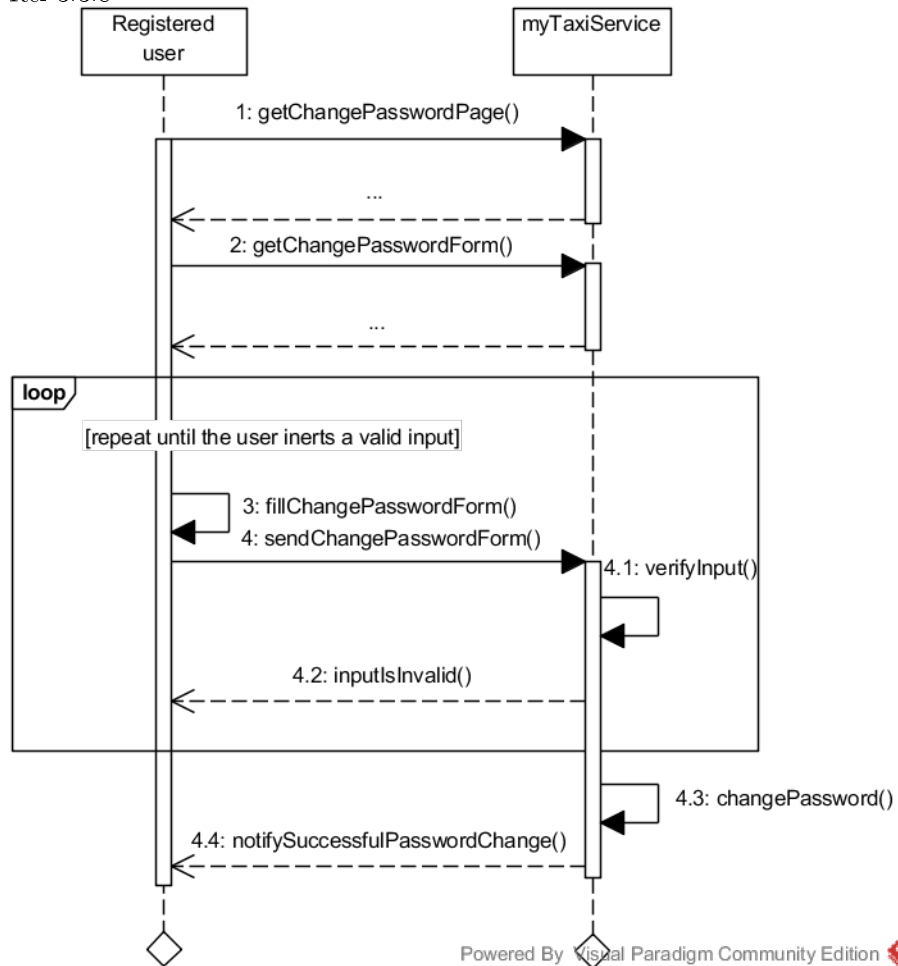
4.2.7 Dispatcher service request

Ref 3.3.7



4.2.8 Change password

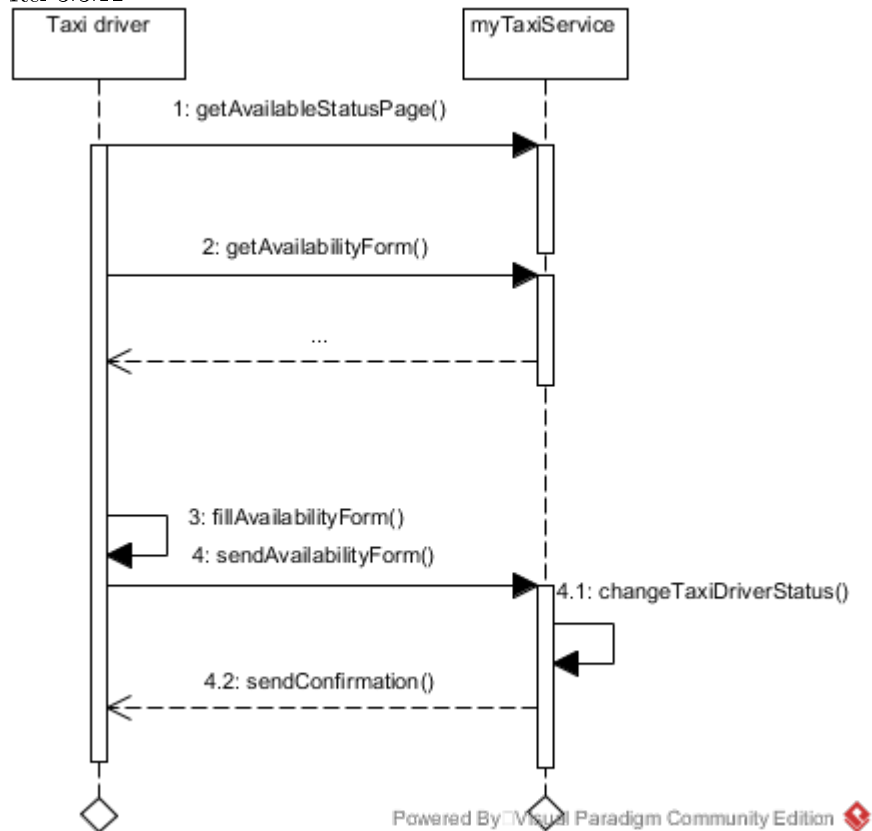
Ref 3.3.8



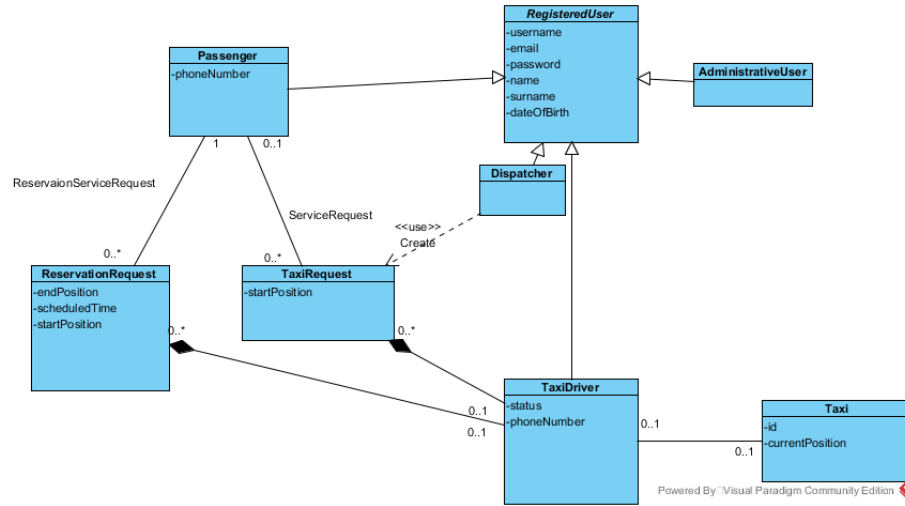
Powered By Visual Paradigm Community Edition

4.2.9 Set available status

Ref 3.3.12



4.3 Class Diagram



4.4 Alloy

4.4.1 Signatures

```
sig Integer {}
sig Text {}

sig Taxi
{
  id: one Integer
}

abstract sig RegisteredUser
{
  username: one Text,
  password: one Text
}

sig Passenger extends RegisteredUser {}

sig TaxiDriver extends RegisteredUser
{
  taxi: one Taxi
}

sig Dispatcher extends RegisteredUser {}

sig AdministrativeUser extends RegisteredUser {}

one sig UsersRecord
{
  users: set RegisteredUser
}
```

```
sig TaxiRequest
{
    passenger: one Passenger,
    taxiDriver: one TaxiDriver
}

one sig TaxiRequestsRecord
{
    requests: set TaxiRequest
}

sig ReservationRequest
{
    passenger: one Passenger,
    taxiDriver: one TaxiDriver
}

one sig ReservationRequestsRecord
{
    requests: set ReservationRequest
}
```


4.4.2 Facts

```
fact NoUnrecordedRequests
{
  no taxiRequest: TaxiRequest |
    some taxiRequestsRecord: TaxiRequestsRecord |
      taxiRequest not in taxiRequestsRecord.requests

  no reservationRequest: ReservationRequest |
    some reservationRequestsRecord: ReservationRequestsRecord |
      reservationRequest not in reservationRequestsRecord.requests
}

fact NoTaxiDriverWithMoreThanOneRequest
{
  all taxiRequest: TaxiRequest |
    no taxiRequest2: TaxiRequest |
      (taxiRequest != taxiRequest2 && taxiRequest.taxiDriver = taxiRequest2.taxiDriver)

  all taxiRequest: TaxiRequest |
    no reservationRequest: ReservationRequest |
      (taxiRequest.taxiDriver = reservationRequest.taxiDriver)

  all reservationRequest: ReservationRequest |
    no reservationRequest2: ReservationRequest |
      (reservationRequest != reservationRequest2 && reservationRequest.taxiDriver = reservationRequest2.taxiDriver)

  all reservationRequest: ReservationRequest |
    no taxiRequest: TaxiRequest |
      (reservationRequest.taxiDriver = taxiRequest.taxiDriver)
}
```

```

fact AtLeastOneAdministrativeUser
{
    #AdministrativeUser >= 1
}

fact NoUsersWithSameUsername
{
    no disj u1, u2: RegisteredUser | u1.username = u2.username
}

fact UsernameDiffersFromPassword
{
    no u: RegisteredUser | u.username = u.password
}

fact NoTaxisWithSameID
{
    no disj t1, t2: Taxi | t1.id = t2.id
}

fact NoTaxiDriversWithSameTaxi
{
    no disj t1, t2: TaxiDriver | t1.taxi = t2.taxi
}

fact NoUnrecordedUsers
{
    no user: RegisteredUser |
        some usersRecord: UsersRecord |
            user not in usersRecord.users
}

```

4.4.3 Assert

```

assert userActuallyAdded
{
  all user: RegisteredUser, usersRecord1, usersRecord2: UsersRecord |
    (user not in usersRecord1.users && signup[user, usersRecord1, usersRecord2]) implies
      user in usersRecord2.users
}

assert taxiRequestActuallyAdded
{
  all taxiRequest: TaxiRequest, taxiRequestsRecord1, taxiRequestsRecord2: TaxiRequestsRecord |
    (taxiRequest not in taxiRequestsRecord1.requests && newTaxiRequest[taxiRequest, taxiRequestsRecord1, taxiRequestsRecord2]) implies
      taxiRequest in taxiRequestsRecord2.requests
}

assert reservationRequestActuallyAdded
{
  all reservationRequest: ReservationRequest, reservationRequestsRecord1, reservationRequestsRecord2: ReservationRequestsRecord |
    (reservationRequest not in reservationRequestsRecord1.requests && newReservationRequest[reservationRequest, reservationRequestsRecord1,
      reservationRequest in reservationRequestsRecord2.requests
}

```

4.4.4 Predicates

```

pred show
{
  #TaxiRequest >= 1
  #ReservationRequest >= 1
}

pred signup[user: RegisteredUser, usersRecord1, usersRecord2: UsersRecord]
{
  user not in usersRecord1.users implies
    usersRecord2.users = usersRecord1.users + user
}

pred newTaxiRequest[taxiRequest: TaxiRequest, taxiRequestsRecord1, taxiRequestsRecord2: TaxiRequestsRecord]
{
  taxiRequest not in taxiRequestsRecord1.requests
  implies taxiRequestsRecord2.requests = taxiRequestsRecord1.requests + taxiRequest
}

pred newReservationRequest[reservationRequest: ReservationRequest, reservationRequestsRecord1, reservationRequestsRecord2: ReservationRequestsRecord]
{
  reservationRequest not in reservationRequestsRecord1.requests
  implies reservationRequestsRecord2.requests = reservationRequestsRecord1.requests + reservationRequest
}

```

4.4.5 Result

```

7 commands were executed. The results are:
#1: Instance found. show is consistent.
#2: Instance found. signup is consistent.
#3: Instance found. newTaxiRequest is consistent.
#4: Instance found. newReservationRequest is consistent.
#5: No counterexample found. userActuallyAdded may be valid.
#6: No counterexample found. taxiRequestActuallyAdded may be valid.
#7: No counterexample found. reservationRequestActuallyAdded may be valid.

```

4.4.6 Generated world

