Software Engineering 2: myTaxiService

Requirements Analysis and Specification Document

Chitti Eleonora, De Nicolao Pietro, Delbono Alex
Politecnico di Milano

November 3, 2015

Contents

Chapter 1

Introduction

1.1 Purpose

This document is the Requirement Analysis and Specification Document for the myTaxiService application. Its aim is to completely describe the system, its components, functional and non-functional requirements, constraints, and relationships with the external world, and to provide typical use cases and scenarios for all the users involved. Further, this document will provide formal specification of some features of the applications, by means of the Alloy language [8].

This document is written for project managers, developers, testers and Quality Assurance. It may be useful also to users. It may be used in a contractual requirement.

1.2 Scope

The system is a taxi reservation and dispatching system for large cities. Its main goal is to simplify the access of passengers to the service and to guarantee a fair management of taxi queues.

The system consists in a back-end server application (*myTaxi Server*), a web application front-end (*myTaxi Web*) and in a mobile application (*myTaxi Mobile*).

The system has 2 types of users: passengers and taxi drivers; it should allow the users to sign up and login with their credentials. The system has to know the location of both the passengers and the taxi drivers.

The system allows any passenger to request a taxi, informing him or her about the incoming taxi code and the estimated waiting time.

The system knows about the available taxi drivers and, when a request is incoming, informs one of them about the location of the available passenger; the taxi driver can either accept or deny the ride. If the taxi driver accepts the ride, the system sends a confirmation to the passenger, together with the estimated waiting time. If the taxi driver rejects the ride, the system looks for another taxi driver in the same area of the city.

The system offers programmatic interfaces (APIs) to enable the development of additional services on top of the basic one.

The system is provided with two optional modules:

Taxi reservation allows the passenger to reserve a taxi by specifying the origin, the destination and the hour of the ride.

Taxi sharing allows the passengers to share a ride together, dividing the costs.

1.3 Goals

The goals of the myTaxiService software are the following:

1. simplify the taxi call and reservation system of a large city;
2. let the passengers call a taxi;
3. let the passengers reserve a taxi for later;
4. let the taxi drivers to answer the calls in a fast and convenient way;
5. improve the efficiency of the service by tracking the positions of passengers and taxi drivers;
6. let the users sign up and login to the service;
7. let the users manage their profile.
8. let the users share a taxi with other people, dividing the costs.

1.4 Definitions, acronyms, and abbreviations

RASD: Requirements Analysis and Specification Document (this document).

System: the whole software system to be developed, comprehensive of all its parts and modules.

Core or back-end: the basic software part that offers application logic and APIs and communication interfaces to work with them. The core does not provide a graphical user interface, client code nor modules.

Module: an optional software component which uses the core system APIs to provide additional features.

Client: the web application component or the mobile app.

Passenger: the registered user who uses the service for a taxi ride. Only passengers can call or reserve a taxi.

Traveler: any person, registered or not, that gets on a taxi. A passenger is himself a traveler and can bring with himself more travelers. For example, a man can call a taxi for 2 people, himself and his wife, using his account. The man is the only passenger; both he and his wife are travelers.

Taxi driver: any taxi driver subscribed to the service.

User: any user (passenger or taxi driver) subscribed to the service.

RDBMS: Relational Data Base Management System.

JVM: Java Virtual Machine.

API: Application Programming Interface.

UI: User Interface.

FIFO: First In First Out (policy for queue management)

1.5 References

This document refers to the project rules of the Software Engineering 2 project [5] and to the RASD assignment [6].

This document follows the IEEE Standard 830-1998 [7] for the format of Software Requirements specifications.

1.6 Overview

This document is structured in three parts:

Chapter 1: Introduction. It provides an overall description of the system scope and purpose, together with some information on this document.

Chapter 2: Overall description. Provides a broad perspective over the principal system features, constraints, and assumptions about the users and the environment.

Chapter 3: Specific requirements. Goes into detail about functional and nonfunctional requirements. This chapter is arranged by feature.

Chapter 2

Overall description

2.1 Product perspective

2.1.1 User interfaces

The clients have to deal with the dichotomy between web user interaction and mobile interfaces. It is necessary to provide a common and uniform look and feel among the different hardware interfaces.

All the interfaces shall be intuitive and user friendly. They should not require the reading of detailed documentation to be used.

2.1.2 Hardware interfaces

The main hardware interface of the system consists in the access to the GPS data in the mobile application.

2.1.3 Software interfaces

The mobile application must support Android and iOS. The web application works on any web server that supports Java.

The back-end stores its data in a RDBMS and can run on every platform that supports the JVM.

The back-end must offer programmatic interfaces (APIs) for user interfaces and external modules, like:

- taxi reservation
- ride sharing
- online payments

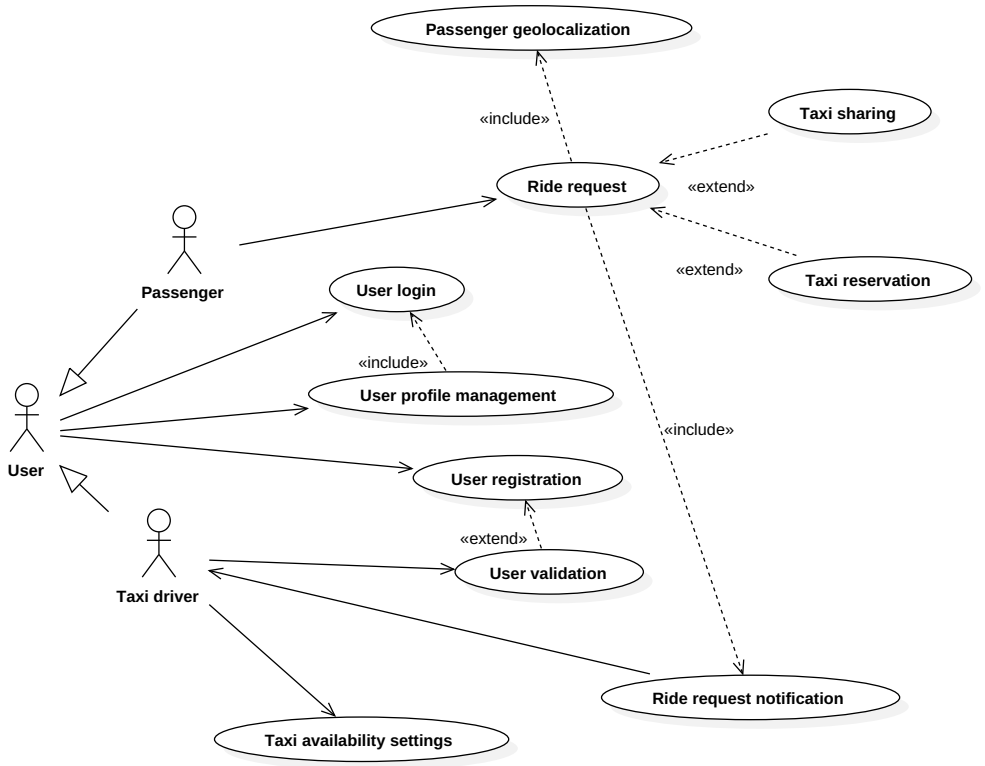

Figure 2.1: The comprehensive use-case diagram of all the functionalities implemented by the system.

- web interface

2.2 Product functions

The system allows passengers to book a taxi, and taxi drivers to take care of the request.

This is a list of what the users of the service can do.

- All users can:
  - create an account
  - login
  - edit profile data
  - delete their account

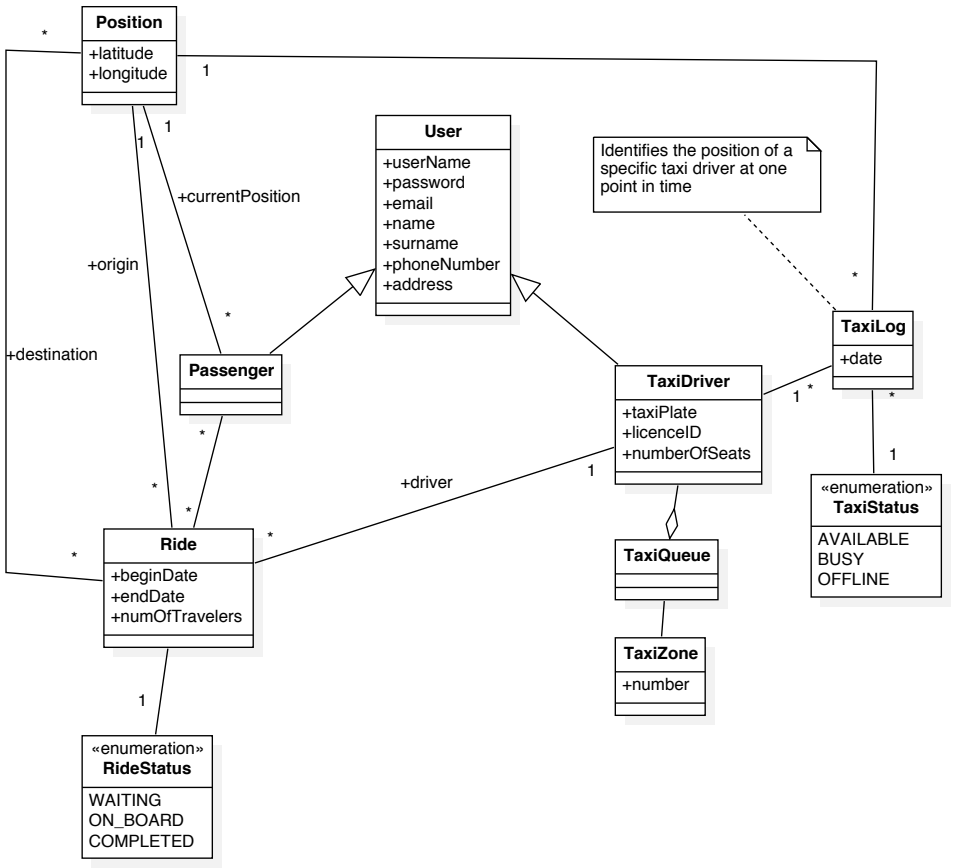

Figure 2.2: The comprehensive class diagram of the system.

- Passengers can:
  - request a taxi
  - share a taxi with other passengers
  - reserve a taxi in advance for a specific time
- Taxi drivers can:
  - mark themselves as available or busy
  - accept or deny a lift request
  - get the exact position of the passenger, after having accepted a ride request

2.3 User characteristics

The two kinds of users are *passengers* and *taxi drivers*. We give for granted that both kinds of users have access to Internet.

Taxi drivers must be able to install and use the mobile application on their cellphone to answer the ride requests.

Passengers have to use the browser application or the mobile app. They move alone or with other people ("travelers").

2.4 Constraints

2.4.1 Regulatory policies

It's user responsibility to ensure that the use of the system complies with the local laws and policies.

The system must ask the user for the permission to acquire, store and process personal data and web cookies. The system must offer to the user the possibility to delete all the personal data.

2.4.2 Hardware limitations

The system has to run under the following worst-case conditions:

- App:
  - 3G connection, at 2 Mb/s
  - 100 MB of free space

  - 2 GB of RAM

- Web application:
  - 2 Mb/s Internet connection
  - 800x600 resolution

2.4.3 Reliability requirements

The system must have a minimum availability of 98%.

2.4.4 Criticality of the application

The system is not employed in life-critical applications.

2.4.5 Safety and security considerations

The locations of the passengers and their destinations must be kept private unless the passengers choose to share rides.

For security reasons, only taxi drivers with a valid license must be able to use the service.

2.5 Assumptions and dependencies

We assume that:

- All users have access to a stable Internet connection.
- The taxi driver's cellphone is provided with a GPS navigator.
- The GPS on the mobile phone of the taxi driver is available.
- The taxi is in a taxi zone queue.
- The taxi driver is able to reach the meeting point within 10 minutes from the agreed time 90% of the times.
- The taxi driver is able to reach the meeting point within 20 minutes from the agreed time 100% of the times.
- The passenger waits in the same place until the taxi arrives.
- The taxi driver picks up the correct passenger.

- The taxi driver correctly updates his status (off shift, available, busy).
- The passenger specifies the correct location, if its GPS is not available.
- The passenger specifies the correct number of travelers.
- The taxi driver is able to see notifications of new passengers during shared rides.
- The passenger specifies the correct destination when ride sharing is enabled.
- The passenger is ready to modify his route, though not greatly, in order to allow ride sharing.
- Each taxi driver owns and uses only one taxi.
- The number of seats in the taxi is fixed and never changes.
- The taxi drivers charge the passengers directly. The system has no role in the payments.
- The system does not know the taxi costs.
- The number of taxis is sufficient to satisfy the demand in each area.

2.6 Future extensions

The system will be implemented foreseeing the possibility of further extensions, for example:

1. Provide secure and reliable methods for ride payments. This functionality would allow the passengers to pay for the rides using credit card information stored securely.
2. Offer a taxi rating system which lets the passengers evaluate taxi drivers.
3. Monitor passenger reliability, recording when they do not show up at the meeting point or how many minutes later they arrive, in order to limit their access to the service when their reliability is bad.
4. Create a fidelity score for every passenger, which gives them the opportunity to benefit from special offers. The score increases every time the passenger uses the taxi service, proportionally with the distance covered.

Chapter 3

Specific requirements

3.1 External interface requirements

3.1.1 User interfaces

The user interfaces must satisfy the following UI constraints:

- Web application
    1. The web pages must adhere to the W3C standards. In particular, the software shall conform to the HTML 5 [1] and CSS [2] standards.
    2. The web pages must be accessible also by text-only browsers.
- Mobile app
    1. The iOS version must adhere to the iOS Human Interface Guidelines [3].
    2. The Android version must follow Android design guidelines [4].
- Server back-end
    1. The server back-end must be configurable by means of a configuration text file.
- Common to all client interfaces:
    1. The clients must have an UI that is accessible to disabled people.
    2. The interface must offer the possibility to choose the language used at all times.

3. The first screen must ask the user to login in order to begin operations.
4. A dashboard with links to every function shall be displayed in the home page in order to show the user the capabilities of the system and allow him to save time.
5. The dashboard must be linked in every screen.
6. The top bar must show the last taxi service called, with a link to a screen which displays the reserved taxi history.
7. The compilation of data fields has to be made with suitable controls (multiple choice, date picker, text field, ...) in order to simplify the user's experience of the app.
8. UI controls and views must be suitable for the input interface and the screen size.

3.1.2 Hardware interfaces

The client app must be able to access the GPS data of the user's phone. It has to deal with security dialogs, authorization and platform-specific standards.

3.1.3 Software interfaces

The system is built in a modular fashion: the core part of the system exposes public APIs allowing the building of new interfaces and modules.

For a detailed specification of the programmatic interfaces, see subsection 3.2.9.

The required software products used by the back-end are:

- MySQL 5.7[1]
- Java SE 8[2]

3.1.4 Communications interfaces

The clients communicate with the server via HTTPS requests (port 443).

[1] `http://dev.mysql.com`
[2] `http://java.com`

3.2 System features

3.2.1 User registration

Purpose

Any user can subscribe through the web application or the mobile app.

In both cases the user has to fill a registration form and must agree to the personal data policy according to his/her country privacy laws, otherwise the registration request shall be aborted.

As soon as the user has submitted all the data, the system verifies the consistency of the information and a confirmation mail is sent to check the availability of the email-address. After this last check the registration ends successfully.

The system has two kind of registration forms (for the two kinds of registered users): one for passengers and one for taxi drivers.

The taxi driver registration is more restrictive and requires more user information. The system registers a user that claims to be a taxi driver only if he/she is able to prove it with a currently valid taxi license.

Scenario 1

Alice, a normal citizen without a car, has just discovered the existence of myTaxiService web application and she wants to use it. She opens the homepage of myTaxiService on the website and clicks on "passenger registration".

She gives all the information required and authorises the personal data treatment.

The system verifies the submitted information and sends a confirmation mail. Alice checks the mailbox, opens the mail and clicks on the "confirm e-mail" link.

The system informs Alice that the registration succeeded.

Scenario 2

Bob is a taxi driver that wants to subscribe to myTaxiService application. He downloads the mobile application from his phone app-store and once he opens it, he selects "taxi driver registration". He fills the form, enters his license ID and authorises the personal data treatment.

However he forgets to write his phone number on the form so the system warns him about the forgetfulness. Only after the complete and correct filling of the form, and the personal data treatment authorisation, the registration is one step near the successfully end.

The system verifies the information submitted and sends a confirmation mail. Bob checks the mailbox, opens the mail and clicks on "confirm e-mail".

The system informs Bob that the registration has ended successfully.

Use case

The use case for user registration is shown in Table 3.1.

Statechart

The statechart of the registration process is illustrated in Figure 3.2.

Associated functional requirements

1. On registration, the user can choose to register as a passenger or as a taxi driver.
2. Passengers must provide the following information:
   - e-mail address
   - username
   - password
   - name
   - surname
   - address (optional filling)
   - phone number (optional filling)
3. Taxi drivers must provide the following information:
   - e-mail address
   - username
   - password
   - name
   - surname
   - address
   - phone number
   - taxi license ID
   - taxi number-plate

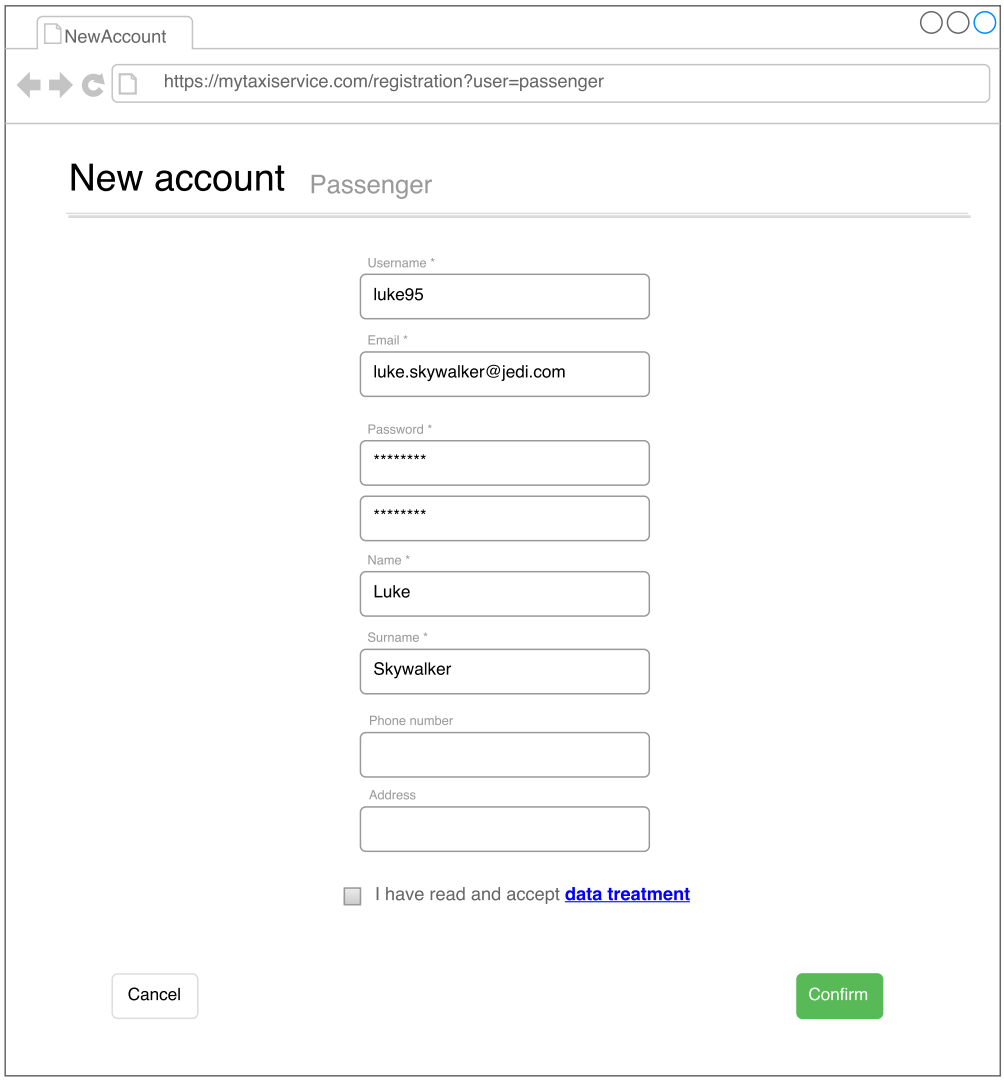

Figure 3.1: Concept of the registration webpage.

Actor	Unregistered user, Registered passenger or taxi driver
Goal	Goal 6
Input condition	The user chooses to create a new user account.
Event Flow	1. The user selects taxi driver or passenger registration. 2. The registration form is loaded and the user compiles it. 3. The user authorizes the personal data treatment. 4. The user reads the e-mail received by myTaxiService and clicks on the link to confirm the registration.
Output condition	The system tells the user that he/she has been successfully registered.
Exception	• Some exceptions are handled notifying the user of the problem and reloading the registration form (step 2 of Event Flow). The requirements that generate these kind of exceptions are: 4, 5, 6, 7, 9. • Some exceptions are handled aborting the registration (all user's data is deleted). The requirements that generate these kind of exceptions are: 10, 12c.

Table 3.1: Use case for user registration.

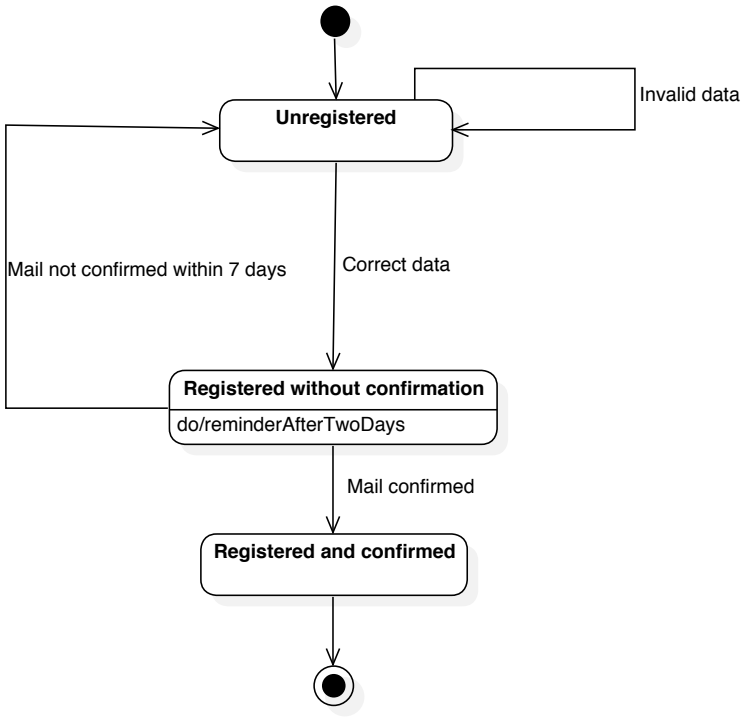

Figure 3.2: Statechart of the registration process.

- maximum number of seats available for travelers in the taxi

4. There mustn't be another user already subscribed with the same username or e-mail.
5. There mustn't be another taxi driver already subscribed with the same taxi license or taxi number plate.
6. The username must match the regular expression "`[a-zA-Z][a-zA-Z0-9]{2,20}`"
7. The system accepts a password that contains at least one number and one capital letter and that has a minimum length of eight characters.
8. The system must ask for the password twice.
9. The system accepts the password only if it is entered identically both times.
10. If the personal data treatment is not authorised, the subscription is canceled.
11. The system must allow the user to abort the registration process at any time.
12. Email confirmation process:
    (a) The subscription ends successfully when the user clicks on the link in the confirmation e-mail.
    (b) After two days without an answer, the systems sends another confirmation e-mail.
    (c) After seven days, the user's registration info are deleted and the user may re-try the registration process.

3.2.2 User login

Purpose

Any subscribed user can login to use myTaxiService services. The system requires the user to provide nickname and password, or e-mail and password, to log in correctly.

If the user doesn't remember his password, a new one is sent to the user's e-mail address. If the user clicks on the link in the e-mail, the password is reset and, as soon as he logs in, the system asks him to choose a new one.

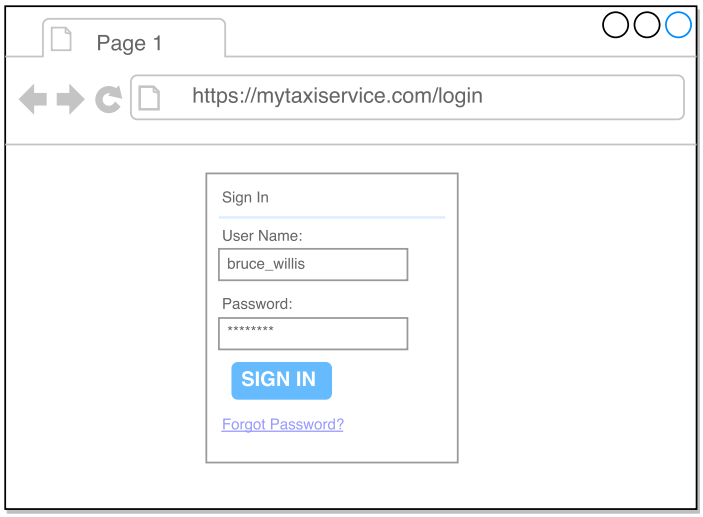

Figure 3.3: Concept of the login webpage.

Scenario 1

Bob opens the home page of myTaxiService on the web and clicks on "login". he's asked to enter the nickname or the e-mail and the password. He doesn't recall his nickname, so he enters the registration e-mail and the password. He clicks on "enter". Everything is correct so he can access to the services as a logged user.

Scenario 2

Alice opens the home of myTaxiService page on web and clicks on "login". She is asked to enter the nickname or e-mail and the password. She enters nickname and password and selects "enter". However, the entered password isn't correct so the system asks her to re-enter it. After some failed attempts she selects "password forgot". The system sends her a new password via e-mail. She enters that one and the access is permitted, but before successfully ending, the login the system requires her to change the password immediately. Alice changes her password and finally logs in to the services.

Use case

The use case for user login is shown in Table 3.2.

Actor	Registered passenger or taxi driver
Goal	Goal 6
Input condition	The user, already registered, chooses to login.
Event Flow	1. The user selects "login". 2. The login page is shown. 3. The user enters his/her credentials.
Output condition	The system tells the user that he/she has been successfully logged in. It is loaded the main page where the user can choose one of the myTaxiService options.
Exception	If the username/e-mail or password entered are wrong, the system notifies the user and the login page is reloaded (step 2 of Event Flow): the login has failed.

Table 3.2: Use case for user login.

Response sequence

The response sequence is illustrated in Figure 3.4.

Associated functional requirements

1. In order to log in, the user must insert either a nickname, or the registration e-mail, but not both.
2. On login, the system must grant the user access to his/her account if and only if the following conditions are met:
   (a) The inserted nickname corresponds to a username of an existing user, or the inserted e-mail corresponds to the registration e-mail of an existing user.
   (b) The inserted password is the same of that of the user identified above.
3. The system sends an e-mail after the "password forgot" button is selected.
4. The system resets the user's password only after the link on e-mail is clicked.
5. If the entered password is wrong a new attempt can be made only 10 seconds later.

3.2.3 Standard taxi call

Purpose

Any subscribed passenger shall be able to request a taxi either through the web application or the mobile app. After the request, the passenger is informed by the system about the waiting time and the code of the incoming taxi.

Requests shall be forwarded to available and active taxi drivers in the same zone of the passengers. Taxi drivers shall be able to accept or reject an incoming request.

Scenario 1

Alice needs a taxi. She opens the myTaxiService mobile app on her phone, and selects "Call a taxi". She authorizes the application to access her GPS

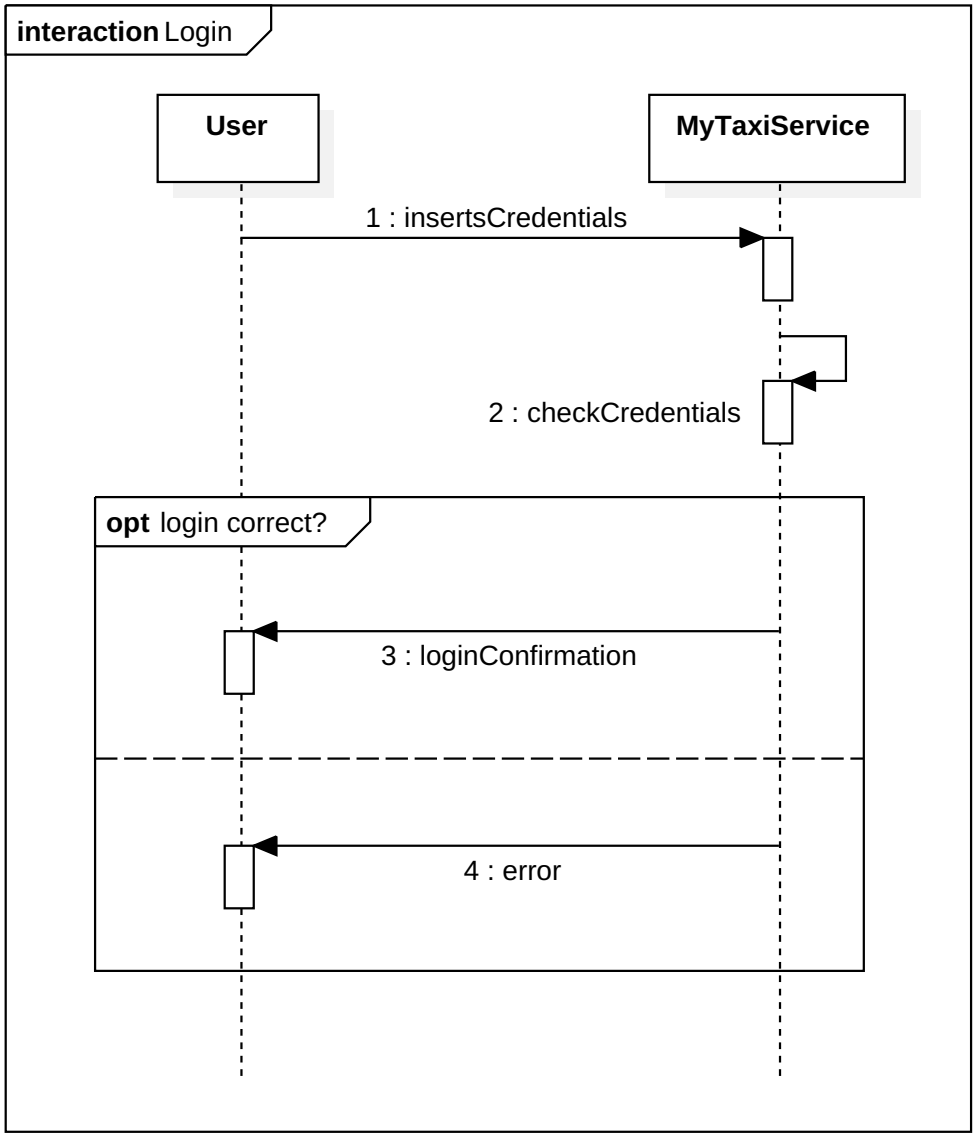

Figure 3.4: Sequence diagram of a user login.

data, checks on the map that her position is correct, enters "1 person" as the number of travelers and confirms the request.

The system forwards the request to Bob, the first taxi driver in Alice's taxi zone. Bob decides to accept the call: a map of Alice's position gets displayed on Bob's phone and the navigator starts.

Bob's position is transmitted from his phone to the system, which computes the ETA for the incoming taxi and shows it to Alice. Bob arrives and picks Alice up. Then he confirms that the passenger is on board.

When the ride is over, Bob taps on "Finish ride" so that the system knows that Bob is ready for another ride.

Scenario 2

Luke Skywalker needs a taxi to his favourite pub, "The Death Star". Luke requests a taxi for 1 person and the request is forwarded to the first taxi driver in queue, Chewbacca. Chewbacca decides to reject the incoming request, so the system puts him at the bottom of the queue.

Luke's request gets forwarded to the new first taxi driver in queue, Han Solo. Han accepts the request and comes to pick up Luke.

Scenario 3

Luke, Leia and Obi-Wan need a taxi. Luke has the myTaxiService app on his phone so he opens it and selects "Call a taxi". He checks that the position is correct, he enters three as the number of seats required and confirms the request.

Luke's request gets forwarded to the first taxi driver in queue, Han Solo, that accepts the request and comes to pick up the group of friends.

Use case

The use case for a taxi call is shown in Table 3.3.

Response sequence

The response sequence is illustrated in figures 3.6 and 3.7.

Associated functional requirements

1. The system must localize the passenger before he or she makes a taxi request.

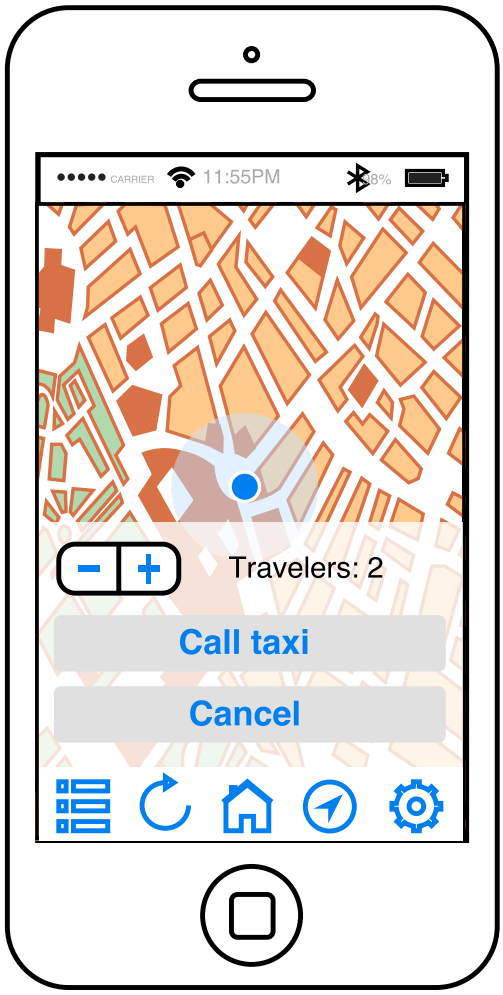

Figure 3.5: Concept of the taxi call interface.

Actor	Passenger
Goal	Goal 2
Input condition	Passenger is already logged in and requests a taxi.
Event Flow	1. The passenger requests a taxi through the client, specifying the number of travelers and the position. 2. The request gets forwarded to the first taxi in the queue of the same taxi zone of the user. 3. The taxi driver can either accept or deny the request; if he denies it, the request is forwarded to the next taxi driver in the queue, and so on. 4. Finally the request gets accepted by a taxi driver. 5. The passenger gets notified that a taxi driver accepted his request and is given the ETA of the incoming taxi. 6. When the taxi driver meets the passenger, he/she signals that the passenger is on board.
Output condition	The driver confirms that the passenger is aboard.
Exception	No taxi is available in the passenger's zone.

Table 3.3: Use case for a standard taxi call.

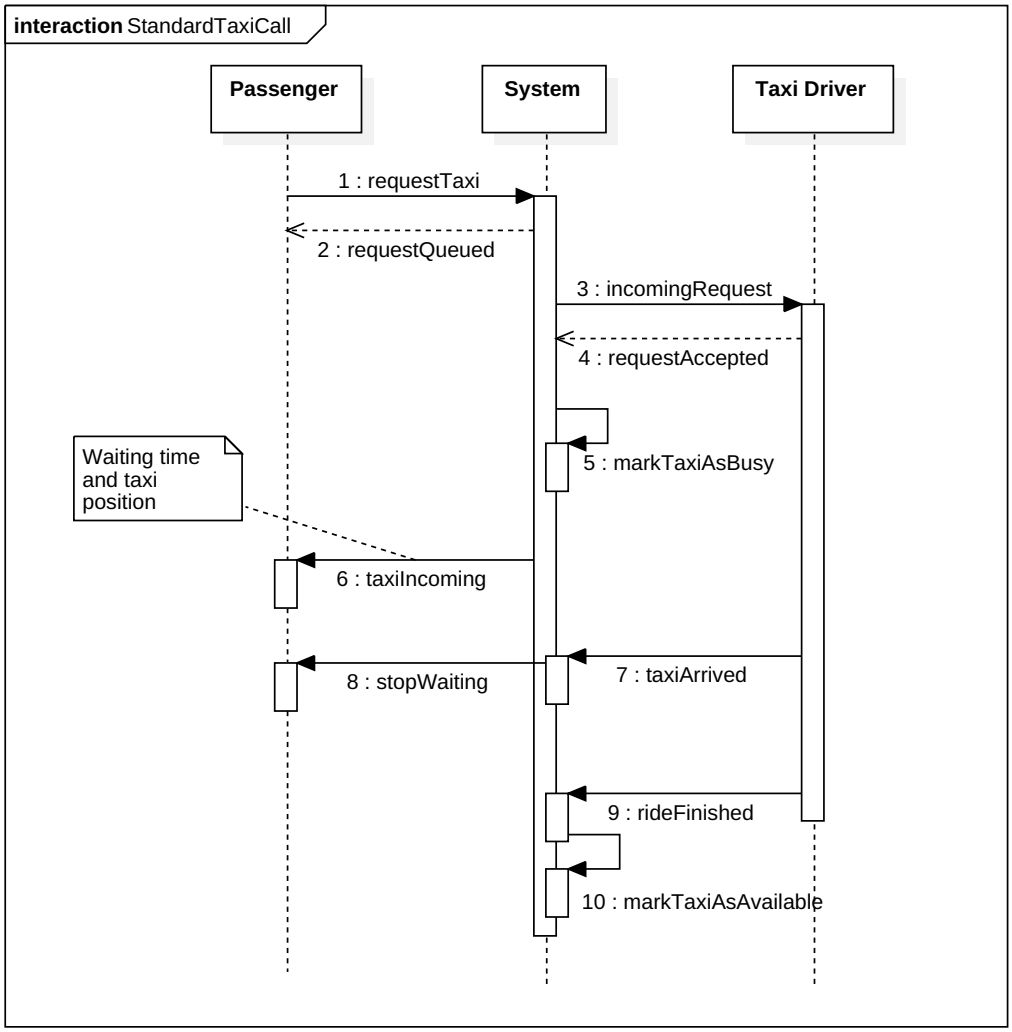

Figure 3.6: Sequence diagram of a successful taxi call, picked up by the first taxi driver called by the system.

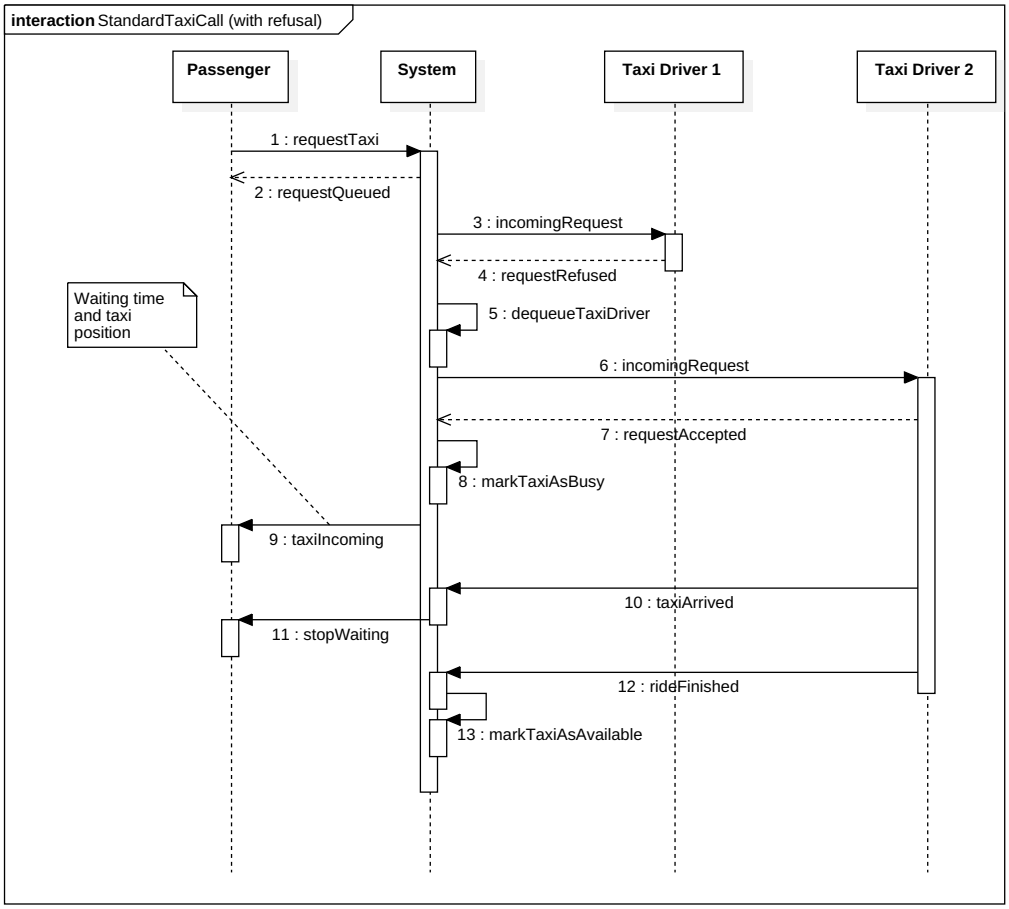

Figure 3.7: Sequence diagram of a taxi call when the first taxi driver to receive the request refuses the call and the second one accepts it.

(a) (App) If GPS info is available and the passenger can be tracked within a radius of 50 m, then the passenger is presented with the option of using the current GPS position.

(b) (App) If GPS info is not available or the precision is less than 50 m, then the app requests the passenger to insert a valid address inside the city. Then the address is shown on a map and the passenger can confirm its position.

(c) (Web) In the web application the user is always requested to insert a valid address inside the city. Then the address is shown on a map and the user can confirm its position.

(d) The passenger needs to be inside a taxi zone in order to use the system. If the passenger is not in any taxi zone, the system does not allow any taxi call.

2. When the system knows the passenger's position, it asks the number of travelers and then it presents the passenger the option to request a taxi.

3. The system must ask the passenger for confirmation before delivering the taxi request.

4. After the passenger's confirmation of a taxi request, the request is delivered to the first taxi with a sufficient number of seats in the queue for the taxi zone in which the user is located.

5. Taxi requests are forwarded only to active taxi drivers which are located in the same taxi zone, who are not currently busy.

6. Taxi requests are processed in order of arrival.

7. If there's no available taxi driver in the passenger's taxi zone, or if all of available taxi drivers have refused the request, an error message is displayed to the passenger and the request is aborted.

8. When a taxi driver receives a request, the mobile application shows a notification and emits a sound.

9. When receiving a request, the taxi driver is presented with the possibility of accepting or denying it.

10. After having accepted a request, the taxi driver is automatically marked as "busy".

11. After having accepted a request, the mobile application shows to the taxi driver a map with the location of the passenger, and automatically starts navigation instructions on the app preferred by the taxi driver (Google Maps, Waze, TomTom, etc.).

12. After the taxi request is accepted, the app shows the passenger a map with the current position of the incoming taxi.

13. After the taxi request is accepted, the passenger is informed about the ETA for the incoming taxi.

14. The ETA for the incoming taxi is fetched from the server every 90 s.

15. The ETA for the incoming taxi is computed by the system considering the distance from the taxi to the passenger and the current traffic conditions.

16. When the taxi is within 20 m from the passenger, the app asks the taxi driver to confirm that the passenger is aboard.

17. When the taxi driver confirms that the passenger is aboard, the taxi request is marked as fulfilled and the system stops showing the waiting time to the passenger.

18. The system must prevent the passenger to call a taxi while he/she is already waiting for one.

19. The system must prevent the passenger to call a taxi while he/she is already traveling in a taxi.

3.2.4 Ride request notification to the taxi driver

Purpose

After a taxi driver has informed the system about his/her availability (section 3.2.5) he/she will be able to receive ride request notifications. The taxi driver receives the notification on his/her cell phone and has one minute to accept or reject the request. If after one minute the request is not answered, the system will consider the request as refused.

Scenario 1

Travis, who suffers from chronic insomnia, every night waits in his taxi for an incoming request. When a user requests a taxi, the system processes the request and decides which taxi driver to send.

For a new incoming request, the system chooses Travis, who is on the top of the queue and in the same taxi zone of the passenger. The system sends him a notification on his mobile app. Travis reads the message which reports location and time of the meeting and decides to accept the request pushing the appropriate button on the display.

The system acknowledges the decision and waits for Travis to notify that the passenger is on board (last part of 3.2.3).

Scenario 2

The system receives a new incoming request from a user and selects Frank Martin as designated taxi driver (section 3.2.5). The system sends him a notification and waits for the reply.

Frank has just finished a very demanding ride, so he decides to refuse the request using the interface of the app. The system receives Frank's decision and looks for another available taxi on the same taxi area.

Use case

The use case for a driver notification is shown in Table 3.4.

Response sequence

The use case associated to the response sequence is shown in Figure 3.7.

Associated functional requirements

1. The app must show the request notification to the taxi driver.
2. The system allows the taxi driver to accept or decline the request using the app.
3. The app must produce sounds or visual notifications as the taxi driver has chosen.
4. The system must use timeouts in order to prevent infinite pending requests.

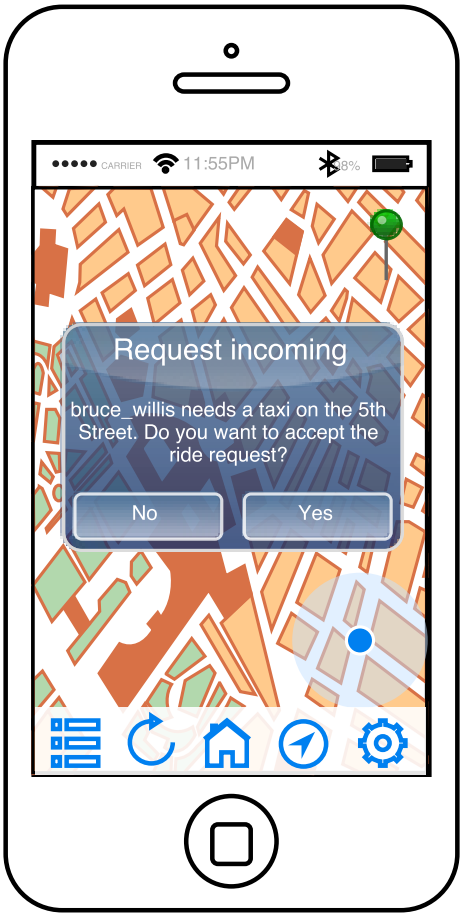

Figure 3.8: Concept of the ride request notification.

Actor	Taxi driver
Goal	Goal 4
Input condition	The system informs the taxi driver about a new request.
Event Flow	1. The system informs the taxi driver about a new request, specifying time and location of the ride. 2. The taxi driver sees the information on his cell phone and chooses whether to accept or to refuse the request. 3. If after a minute the taxi driver has not sent the response, the system considers the request refused. 4. If the request is accepted, the system dequeues the taxi and marks it as busy, otherwise it sends the request to the next taxi. 5. When someone has accepted, the system notifies it to the passenger specifying the arrival time of the taxi.
Output condition	The system informs the passenger about the estimated time arrival of the taxi.
Exception	No taxi replies the request.

Table 3.4: Use case for driver notification.

5. The system has to provide the taxi driver with information about the time and place of the meeting with the passenger.

6. The system gives to the taxi driver one minute to reply. Else, the request is automatically refused.

7. The request notification shall contain the following information:

   (a) number of travelers;

   (b) location of the passenger;

   (c) username of the passenger;

   (d) estimated time of the meeting with the .

3.2.5 Taxi availability handling

Purpose

When a taxi driver is ready for a ride, he/she shall be able to notify the system using his mobile phone. With the app he/she can change his status from "not in service" to "in service".

The city is divided in "taxi zones". Taxis can start and stop only inside taxi zones: origins and destinations located outside taxi zones are not allowed by the system. Each taxi zone is associated with a taxi queue.

When the system receives a status update from the taxi driver, it inserts him/her in the right queue for his/her taxi zone using the information from the taxi GPS if the new status is "in service", or it removes him/her from the queue if the new status is "not in service".

After a ride, a taxi driver has to notify, using the dedicated section of the app, that the ride is over. The system shall reinsert the taxi into the right queue.

The queues are FIFO and when a taxi driver refuses a ride, the system moves the taxi to the bottom of the same queue.

When a taxi driver accepts a ride, the system marks the taxi as busy and removes it from the top of the queue.

Scenario 1

Ernie gets in his taxi, ready to start his working day. He takes out his phone from his pocket and after logging in he changes his status in "in service".

The system acquires the changing and retrieves the GPS position of Ernie's taxi. It analyses the data and puts the taxi in the right queue.

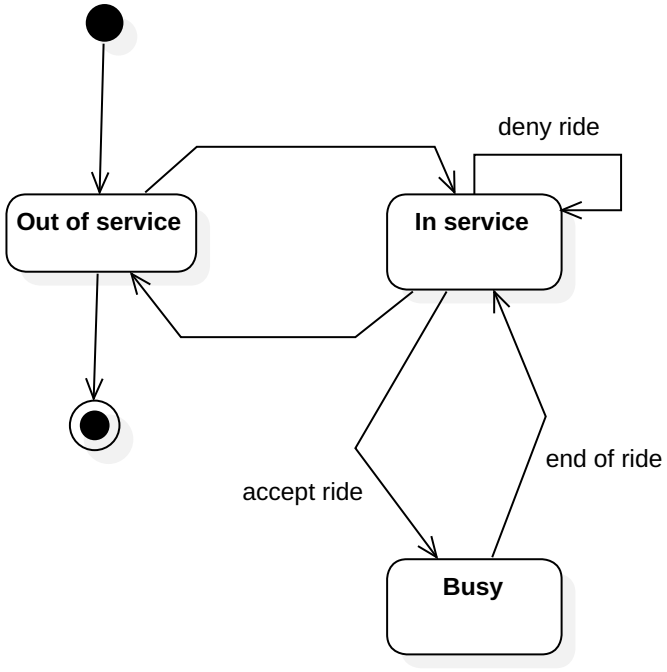

Figure 3.9: Statechart of the possible statuses of a taxi driver.

After thirty minutes the system has an incoming request from Ernie's zone. The first taxi on the top of the queue is Ernie's. The system sends a notification to Ernie and waits for a reply.

Ernie receives the request and accepts the meeting. The system pops Ernie from the top of the queue and marks it as busy.

When Ernie accomplishes the ride, he notifies the system, which retrieves the new GPS position and inserts Ernie's taxi in the queue.

Use case

The use case for a taxi availability handling is shown in Table 3.5.

Response sequence

The statechart representing the taxi driver's possible statuses is represented in Figure 3.9.

The response sequence associated with this functionality is shown in Figure 3.10.

Actor	Taxi driver
Goal	Goal 4
Input condition	Taxi drivers are logged in and change their status to "in service".
Event Flow	1. The taxi driver changes his status to "in service" 2. The system enqueues the taxi in the right queue using GPS information 3. A new request incomes 4. The system chooses the first taxi in the queue and sends the request for a ride 5. The taxi driver can either accept or deny the request; if he denies it, the request is forwarded to the next taxi driver in the queue, and so on. 6. When a taxi driver accepts the request, the system dequeues it and marks it as busy 7. The taxi driver drives to the location 8. The taxi driver notifies that the passenger is on board 9. When the ride is completed, the taxi driver notifies it to the system 10. The system enqueues the taxi in the right queue using GPS information 11. The taxi driver changes his status to "not in service" 12. The system dequeues the taxi
Output condition	The driver changes his status to "not in service".
Exception	No taxi is available in the passenger's zone.

Table 3.5: Use case for taxi availability handling.

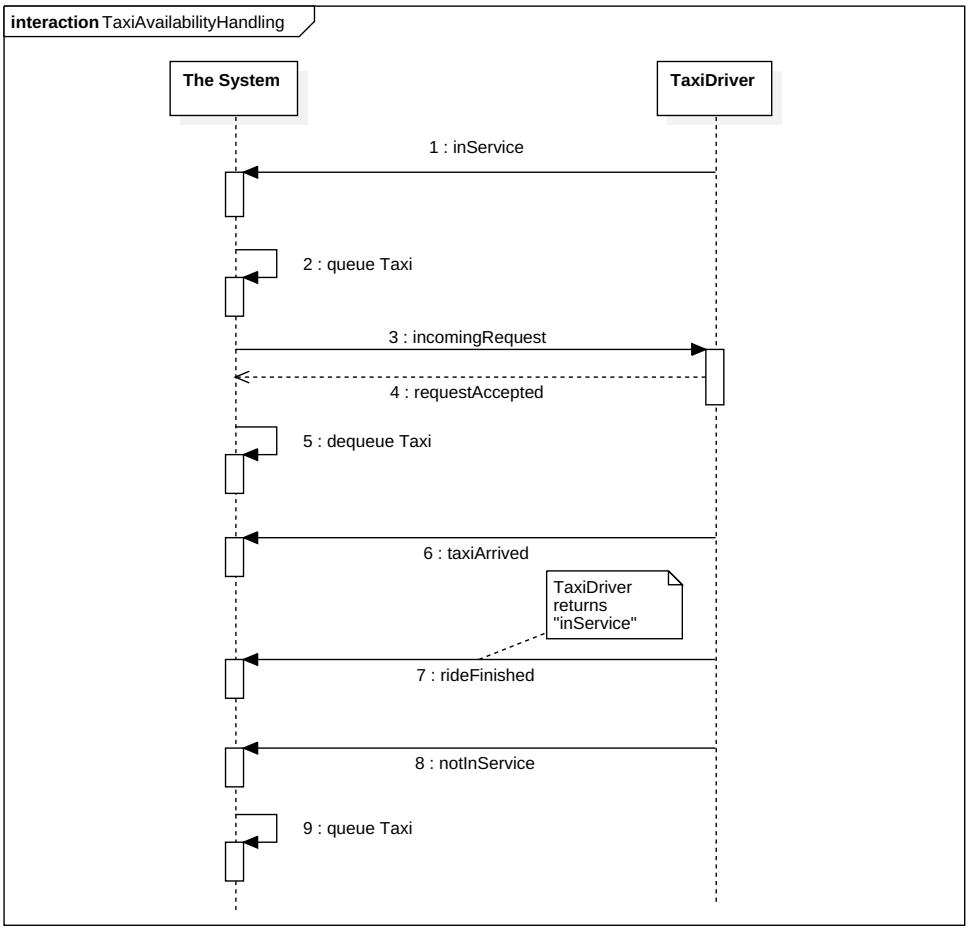

Figure 3.10: Sequence diagram of an accepted taxi call.

Associated functional requirements

1. The system knows the information about the GPS installed on the taxi.
2. The mobile app has to offer to the taxi driver the "change status" function and the "ride complete" function.
3. The system must notify the taxi when a request is incoming.
4. The system inserts the taxi in the right queue and changes it if the taxi changes area.
5. For every new request, the system chooses the taxi driver on the top of the queue.
6. The taxi driver has to reply in one minute to the request, otherwise the ride is automatically refused.
7. The system manages the queues using the FIFO policy.
8. When busy, the taxi driver is presented by the mobile application with the option to end the ride.
9. When the taxi driver ends the ride, he or she is marked as available and gets re-inserted in the taxi queue of the taxi zone where he or she is located.
10. If the taxi driver refuses a ride, he or she is automatically reinserted at the bottom of the queue for his zone.
11. If a taxi driver in "in service" state goes out of a taxi zone and enters into another taxi zone, his status is preserved. He is removed from the old taxi zone queue and inserted at the bottom of the new taxi zone queue.
12. If a taxi driver in "in service" state goes out of any taxi zone, the system marks him automatically as "out of service". When the taxi driver re-enters, he or she may manually change back his status.

3.2.6 Taxi reservation

Purpose

Any subscribed passenger shall be able to reserve a taxi for a ride at a predefined time. The passenger has to specify in advance the origin and the

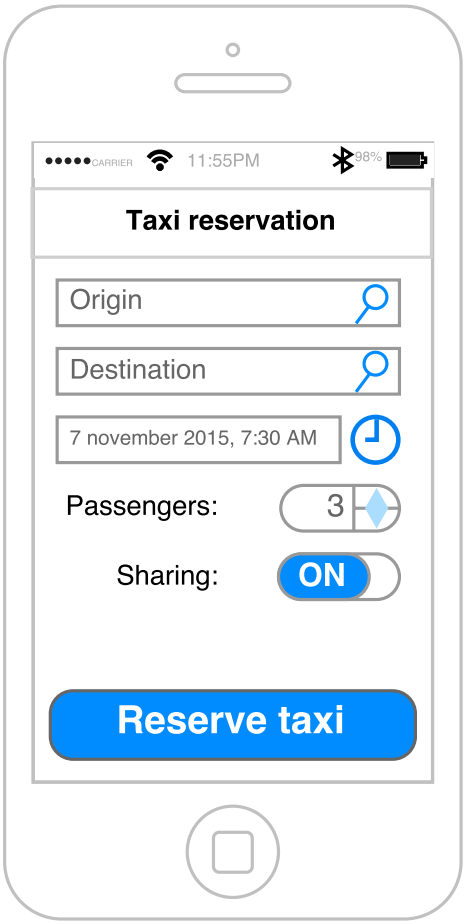

Figure 3.11: Concept of the taxi reservation screen on the mobile application.

destination of the ride, along with the starting date and time and the number of travelers.

The passenger can also enable the sharing mode for the ride. In that case, also the requirements in subsection 3.2.7 apply.

Scenario 1

John McClane will need a taxi to get to the airport tomorrow morning. He opens the web application of myTaxiService and decides to book a taxi for 6:00 AM for a ride from his home to the airport, for one person. He confirms the request.

The morning after, at 5:50 AM, the first taxi driver in the queue gets McClane's request and accepts it. He comes to pick up McClane and brings him to the airport.

Scenario 2

Cordell Walker needs to get to the Texas Rangers police station tomorrow morning. His pick-up truck needs to be repaired, so he reserves a taxi for 7:30 AM. He enables the sharing option. No one wants to go to the Texas Rangers in the same zone at the same hour for now, so Walker becomes the owner of the ride.

Trivette, who doesn't have a car, also needs to go to the Texas Rangers station, and tries to reserve a taxi with the sharing option. The systems informs him that a shared ride with his colleague Walker is available. He accepts.

In the morning, the taxi driver gets called at 7:20 AM and goes to Walker's place. Trivette reaches the starting point, where the taxi driver picks up Walker and Trivette and leaves them by the Texas Rangers.

Use case

The use case for a taxi reservation is shown in Table 3.6.

Response sequence

The response sequence for a simple taxi reservation is illustrated in Figure 3.12. The statechart of a taxi reservation with ride sharing is shown in Figure 3.13

Associated functional requirements

1. The system presents the passenger with the option to reserve a taxi.
2. The system asks the passenger the origin and the destination of the ride.
3. The system asks the passenger the total number of travelers.
4. The system asks the passenger if he wants to share the ride.
5. Origin and destination must be valid addresses inside the city.
6. If GPS info is present and accurate within 50 m, the passenger can specify "current position" as the destination of the ride.

Actor	Passenger
Goal	Goal 3
Input condition	Passenger is already logged in and chooses to reserve a taxi.
Event Flow	1. The passenger chooses to reserve a taxi using his client. 2. The passenger chooses the origin and destination addresses, the number of travelers, and the time of pick-up. 3. The passenger confirms the reservation. 4. 10 minutes prior to the ride time, the request gets forwarded to the first taxi in the queue of the same taxi zone of the user. 5. The taxi driver can either accept or deny the request; if he denies it, the request is forwarded to the next taxi driver in the queue, and so on. 6. Finally the request gets accepted by a taxi driver. 7. The passenger gets notified that a taxi driver accepted his request and is given the ETA of the incoming taxi. 8. When the taxi driver meets the passenger, he/she signals that the passenger is on board.
Output condition	The driver confirms that the passenger is aboard.
Exception	No taxi is available in the passenger's zone.

Table 3.6: Use case for taxi reservation.

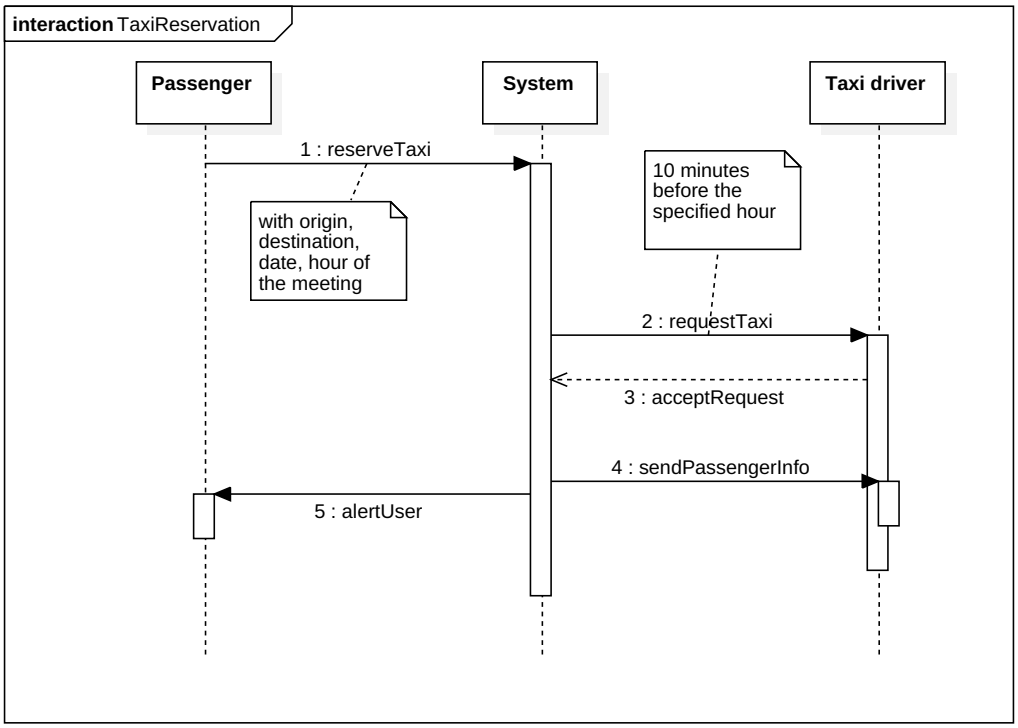

Figure 3.12: Sequence diagram of a successful taxi reservation.

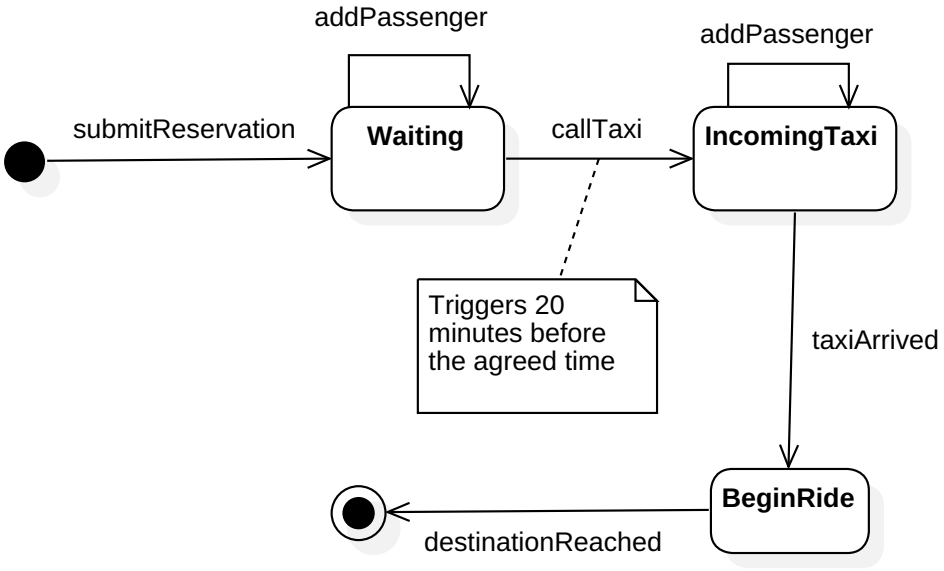

Figure 3.13: Statechart of a taxi reservation with ride sharing.

7. The system asks the passenger the date and time of the ride.
8. The system lets the passenger enter only valid dates and times.
9. The system lets the passenger reserve a taxi from 48 hours to 2 hours before the actual ride time.
10. 10 minutes before the specified arriving time, the system allocates a taxi for the passenger by putting a request in the queue as described in subsection 3.2.3.
11. If no taxi is available at that time, the system keeps retrying until a taxi is available.
12. After the request is accepted, the passenger gets notified with the ETA of the incoming taxi along with its position.
13. The system must prevent the passenger from reserving more than one taxi per hour.
14. A reserved ride can be shared.

15. If the ride is shared, other passengers (with their travelers) can join the ride since it is submitted to the system.

16. If the ride if shared, requirements in subsection 3.2.7 apply.

3.2.7 Ride sharing

Purpose

Every subscribed passenger shall be able to activate the ride sharing function in the mobile app or in the web interface. When this mode is enabled, the passenger has to provide the system also with a destination for the ride.

The system, before allocating a taxi, inserts the pending ride in the set of shared rides and tries to identify possible sharing solutions. With an adequate algorithm it can propose every feasible sharing solution to the user, also considering the number of seats available.

The first user who reserves a taxi is called the "owner" of the ride. The passenger can choose to join one of the pending rides or to refuse the proposal, becoming the owner of a new ride.

In the first case the system informs the owner of the ride. The new passenger and eventually his crowd, i.e. the travelers, have to go to the meeting point with the owner of the ride in time. If they are not able to do that the taxi does not wait for them.

In the second case the system allocates the taxi (3.2.5) and applies the standard procedure (3.2.3). The user becomes the owner of the ride. If there is another passenger willing to share the same ride, the system allows it and redirects this last user to the meeting point with the owner, who is informed as well as the taxi driver.

At the end of the ride, the system splits the taxi fee among all the passenger proportionally with the distance traveled.

Scenario 1

Batman needs a taxi and decides to enable the sharing mode on his cell phone. The app also asks Batman for the destination of his travel and the number of seats (one).

When Batman submits the form, the system matches the path of Batman's ride with every pending ride started from Batman's area. It finds only one compatible ride and sends it to Batman.

Batman finds that it is Joker's pending ride and refuses to share the ride. So, the system allocates a new taxi.

A new user, Robin, is in Batman's area and needs a taxi. He decides to enable the sharing function and the system proposes him Batman's and Joker's rides. He chooses Batman's: the system, which has already communicated location and time to Robin, notifies Batman and the taxi driver that there is an additional passenger.

When the taxi arrives, the taxi driver confirms from his mobile app how many passengers he has picked up.

Use case

The use case for a ride sharing is shown in Table 3.7.

Response sequence

The response sequence is illustrated in Figure 3.14

Associated functional requirements

1. The system has to know the starting point of the new passenger in order to provide feasible sharing solutions. It has to compute the estimated walking time to reach the meeting location and compare it with the estimated taxi arrival time.

2. The passenger can enable the sharing function both in the mobile app and in the web interface.

3. The passenger can choose between possible sharing solutions or a new taxi.

4. The system has to communicate to the taxi driver the presence of a new passenger.

5. The system must be able to know if the taxi driver has already picked up the travelers or not.

6. The system must be able to know how many travelers are in the taxi and what is the next scheduled destination.

7. The total number of travelers must not exceed the number of seats available in each taxi.

8. The taxi driver can insert the number of passengers who have been picked up.

Actor	Taxi driver and Multiple Passengers
Goal	Goal 8
Input condition	The user enables the sharing option from his mobile phone or from the web interface.
Event Flow	1. The passenger enables the sharing option. 2. The passenger enters a destination, and the number of required seats. 3. The system computes the feasible shared rides and proposes them to the passenger. 4. The passenger can accept one of the sharing options or refuse all of them. 5. If the passenger refuses, the system executes the standard taxi call 3.2.3, otherwise it notifies the owner of the ride and the taxi driver. 6. The taxi driver picks up all the passenger at the starting point and starts the ride. 7. When the ride is finished, the system reports the percentages of the taxi fee that each passenger has to pay.
Output condition	The ride ends.
Exception	No taxi available in the area.

Table 3.7: Use case for ride sharing.

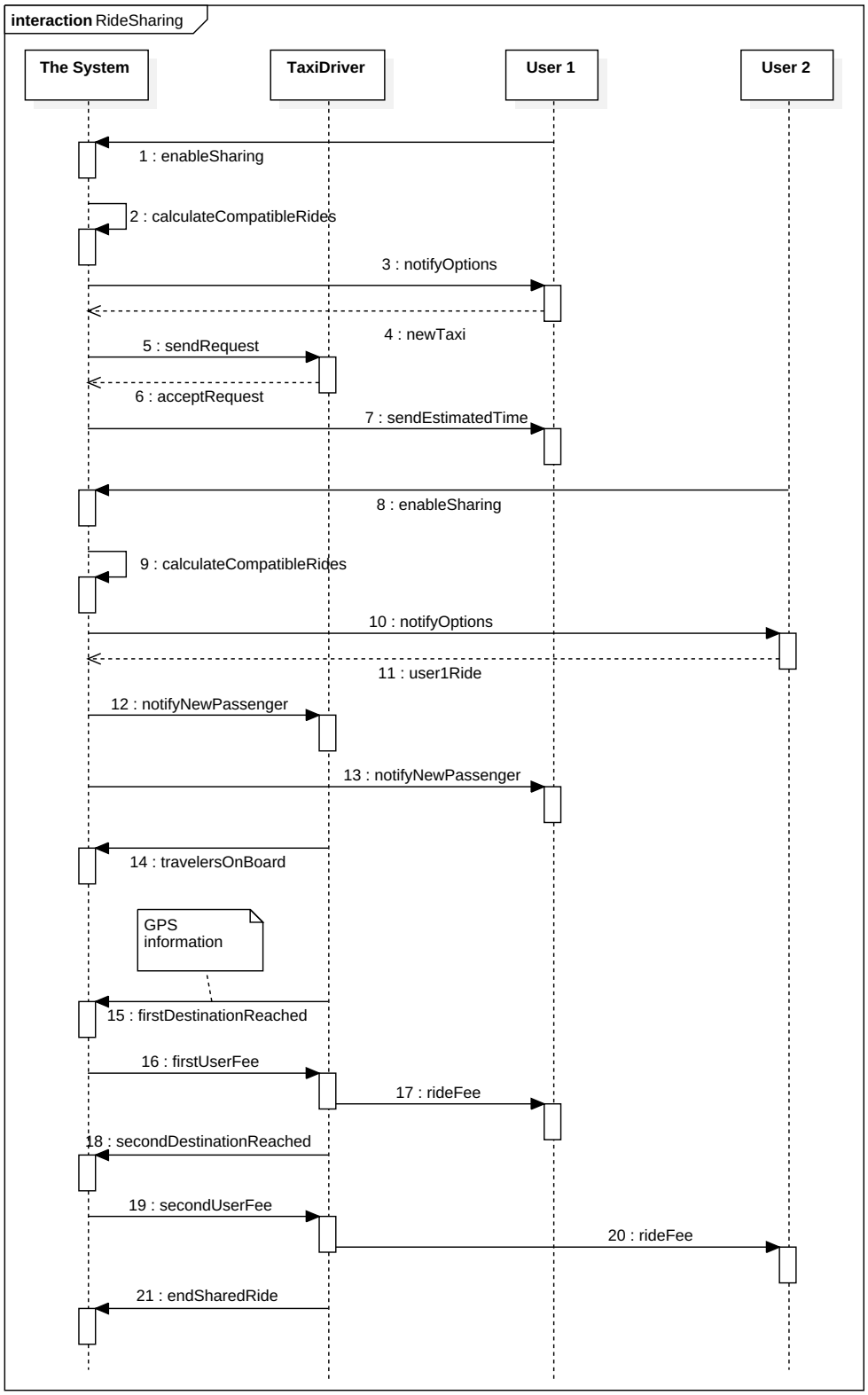

Figure 3.14: Sequence diagram of a ride sharing.

9. After the user chooses to enable the sharing mode, the system has to specify, for every sharing ride, the location and the time.

10. The system splits the ride costs among the passengers, considering the number of travelers linked to a passenger.

    (a) The fee for each passenger is reported by the system as a percentage, because the system knows nothing about the taxi fees in each city.

    (b) The fee for each traveler is computed as follows: for each traveler the system computes the total distance travelled, from the starting point to his or her destination. Then the system divides the distance traveled by each traveler by the sum of all the distances.

    (c) Each passenger has to pay for himself/herself and for all his/her travelers.

11. The system reports all the percentages of the fees to the taxi driver and to all the passengers.

3.2.8 User profile management

Purpose

Any subscribed user can view or update his profile information.

The system allows taxi drivers to use the service only if they load a valid license. A reminder e-mail is sent to the taxi driver three months before the expiration.

Scenario 1

Alice, a myTaxiService passenger without a car, wants to count how many times she has used a taxi in the last month. She opens the home page of myTaxiService page on the web site and clicks on "login". After she has logged in correctly, she clicks on "load profile" and all the information about her account appears on screen, including the taxi request list.

Scenario 2

Gabriele's girlfriend has discovered his password but he doesn't want her to know where he has been. So he decides to change his password immediately. He opens the app on her cell phone, he selects "load profile" and after he selects "modify password". The system asks him to enter the old password

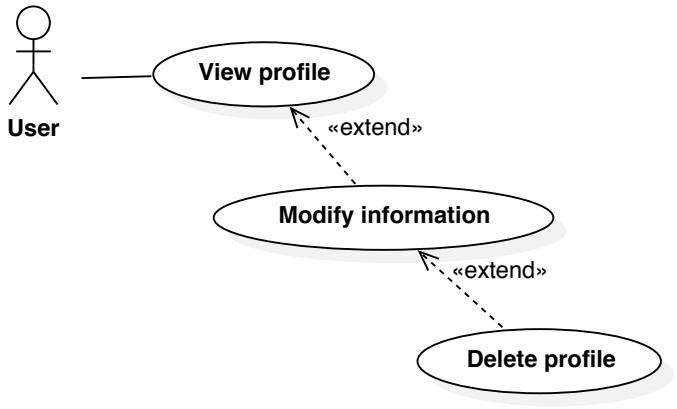

Figure 3.15: Use case diagram of user profile management.

and the new one two times to avoid errors. Once he has verified that everything's ok, he selects "done", and the system confirms that the password has been successfully set.

Scenario 3

A recall mail is sent to Bob, a myTaxiService user that has registered as taxi-driver, to notice him that he has to update his profile with a valid license. A week later he opens the app on his cell phone, he selects "load profile", then "modify". He enters the new license, then confirms selecting "done". The system confirms that the license has been successfully inserted.

Use case

The use case of profile management is illustrated in Figure 3.15.

The use case for viewing user profile is shown in Table 3.8.

The use case for modifying the user profile is shown in Table 3.9.

The use case for deleting the user profile is shown in Table 3.10.

Associated functional requirements

1. Logged passengers can:
    - view user profile
    - modify an information

Actor	Registered user
Goal	Goal 7
Input condition	Passenger is already logged in and wants to check his profile.
Event Flow	1. The user selects load profile. 2. The user's profile page is loaded.
Output condition	The profile page is shown and the user can check his information.
Exception	None.

Table 3.8: Use case for user profile visualization.

Actor	Registered user
Goal	Goal 7
Input condition	Passenger is already logged in and wants to modify his profile.
Event Flow	1. The user selects load profile. 2. The user's profile page is loaded. 3. The user selects edit profile. 4. The edit profile page is loaded. 5. The user enters the new info. 6. The user confirms
Output condition	The information in the profile are updated.
Exception	If one of these requirements is not respected an exception occur: 5b, 5c, 5d, 5f, 5g, 5h, 5j. The exception is handled ignoring the modified info, noticing the user and reloading the edit profile page.

Table 3.9: Use case for user profile modification.

Actor	Registered user
Goal	Goal 7
Input condition	Passenger is already logged in and wants to delete his profile.
Event Flow	1. The user selects load profile. 2. The user selects edit profile. 3. The user selects delete profile. 4. The systems asks the password. 5. The user enters the password and confirms.
Output condition	The user account is removed from the system database.
Exception	If the password entered is wrong, the user gets redirected to the "edit profile" page.

Table 3.10: Use case for user profile deletion.

- delete account
- view the latest taxi request
- view taxi requests list

2. Logged taxi drivers can:
   - view user profile
   - modify an information
   - delete account
   - view the latest taxi request accepted
   - view requests accepted list
3. The system sends a recall mail three months before the expiration of the license submitted.
4. Delete the account:
   (a) The system allows to delete the user's account with an option on screen.
   (b) After "delete account" is selected, the user has to ensure his decision selecting on screen "proceed": the system deletes the account only after the user's confirmation.
   (c) When an account is deleted, all the relative information is destroyed.
5. Modify the profile:
   (a) The system allows to change the information on the profile: any information can be changed, even the username or e-mail.
   (b) The system must verify that the new username entered is unique, i.e. there mustn't be another user that has already entered the same username.
   (c) The system accepts the new username only if it matches the regular expression
   "`[a-zA-Z][a-zA-Z0-9]{2,20}`"
   (d) The system must verify that the new e-mail entered is unique, i.e. there mustn't be another user that has already entered the same e-mail.

(e) If the e-mail is changed the system sends a confirmation mail on the new address.

(f) The system saves the new mail only when the user clicks the link on the e-mail sent.

(g) The system accepts a new password that contains al least one number and one capital letter and that has a minimum length of eight characters.

(h) The system accept a new password only if the old one has been submitted correctly before.

(i) The system asks for the new password two times, when a user wants to change it.

(j) The system accepts the new password only if the same one was entered twice.

(k) The system must allow the user to abort the modification at any time.

(l) The old information isn't replaced until the "done" choice isn't selected by the user.

3.2.9 Programmatic interface

Purpose

The back-end software component shall expose programmatic interfaces to ease the creation of modules that implement new functions (e.g. taxi reservation, taxi sharing).

The programmatic interface will consist in Java public interfaces, classes and methods. Modules are required to use this public interface to carry out their functions, and will be limited in what they can do by this library.

The public interface shall provide safe and controlled access to the data, throwing suitable exceptions when an illegal action is attempted by a module.

Associated functional requirements

1. Creation of a new user (taxi driver or passenger)

   (a) The system shall have a UserManager class that allows to create new users of both passenger and taxi driver types.

   (b) The class shall raise reasonable exceptions if the requirements in subsection 3.2.1 trigger exceptions.

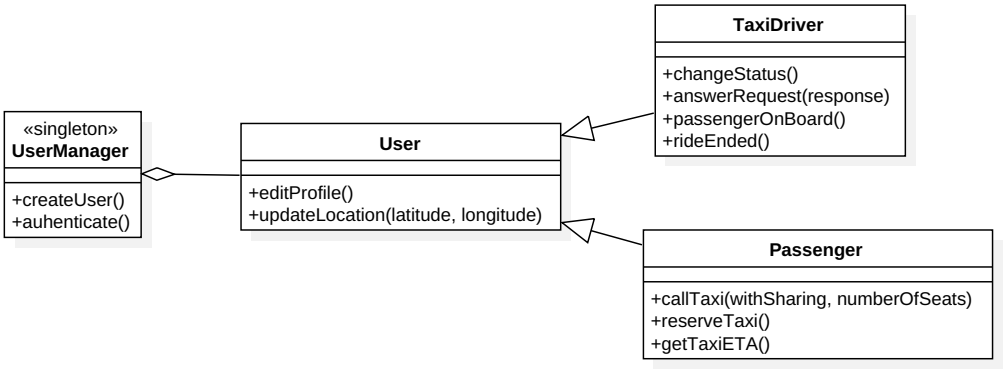

Figure 3.16: The class diagram of the public APIs.

2. User authentication and login
   (a) The system must provide an interface to authenticate a user.
   (b) The system must return reasonable exceptions if the login fails.
   (c) If the login succeeds, the system returns to the caller a token in order for the requests to be stateful.
3. Modification of user profile
   (a) The system shall allow any logged user to change data in his/her profile, according to the requirements in subsection 3.2.8.
4. Updating the location of a user
   (a) The system shall allow external clients to update the location of the logged user.
   (b) The user location must be given as latitude and longitude (GPS coordinates).
5. Taxi availability and status handling
   (a) The system shall provide methods that allow the taxi driver to get and change his/her status (as in subsection 3.2.5).
   (b) This interface shall also accept the "passenger on board" and "ride ended" messages.
6. Taxi call

(a) The system must provide a method that allows any logged in passenger to call a taxi from the current position, according to requirements in subsection 3.2.3.

(b) The method blocks until the request is accepted.

(c) When the request is accepted, the taxi ETA is returned.

(d) The interface must offer a method to get the updated ETA for the incoming taxi.

7. Reservation of a taxi for a specific destination in a specific time

   (a) The system must offer a method to let any logged in user reserve a taxi, according to requirements in subsection 3.2.6.

   (b) The user must provide the following data: initial position, destination, time of meeting.

   (c) If the data is not valid (according to the same requirements), the method should raise suitable exceptions.

8. Taxi sharing

   (a) The system must provide a method that allows any passenger to enable sharing mode, according to requirements in subsection 3.2.7

   (b) The system must allow user to provide a destination and the number of seats required when sharing mode is enabled

   (c) The system must provide the user with a possibility to choose among compatible rides or a new ride

9. Taxi driver notification

   (a) The system must provide the taxi driver with the possibility to choose to accept or refuse an incoming request, according to requirements in subsection 3.2.4

3.3 Performance requirements

The system must support at least 1000 connected passengers at once, and at least 500 simultaneously active taxi drivers at any given time. 95% of requests shall be processed in less than 5 s; 100% of requests shall be processed in less than 10 s.

There is no limit on the total number of registered users.

3.4 Software system attributes

3.4.1 Reliability

The system is not currently designed to run in a distributed environment, but it may be the case in future versions. The reliability of the system is strictly related to the reliability of the server it runs on.

3.4.2 Availability

1. The system must guarantee an availability of 98%.

3.4.3 Security

1. All the communications between server and clients must be protected by strong encryption using the SSL protocol.
2. All attempts of establishing an unsecure communication channel (e.g. plain HTTP) with the server must be refused.
3. Users' passwords must not be stored in plain text in the database: instead, they must be hashed and salted.
4. The system must log all login attempts with IP addresses for at least 7 days.
5. The modules must state clearly which data they will need to read and write.

3.4.4 Maintainability

The code should be well documented using JavaDoc in order to be understood and fixed by developers later. The system must provide a configurable logging function for debugging purposes.

The development of the software will follow the object-oriented Model-View-Controller pattern and the separation of concerns principle.

The build from source code in the Version Control System will be completely automated, as well as unit testing.

The system will support modular extensions in order to have a stable core and divide the core development from the external modules, debugging them separately.

3.4.5 Portability

The back-end server software will be written in Java. It must run on every platform that supports the JVM, and extensive testing will be carried out on every OS to make sure that the system's really portable.

There will not be host-dependent dependencies.

The web application must support the current versions of IE, Firefox, Chrome, Safari, Opera as of 2013. The mobile application must be supported by the last 2 major versions of Android and iOS.

3.5 Alloy

```
// Alloy model for myTaxiService system.

// Defines Bool, True, False
open util/boolean

// Dates are expressed as the number of seconds from 1970-01-01

// ----- SIGNATURES -----

abstract sig User {
    username: one Stringa,
    email: one Stringa,
    password: one Stringa,
    name: one Stringa,
    surname: one Stringa,
    address: lone Stringa,
    phoneNumber: lone Stringa,
    emailConfirmed: one Bool,
}

sig Stringa {}

sig Passenger extends User {
    currentPosition: lone Position,
}

sig TaxiDriver extends User {
    licenseID: one Int,
    taxiNumberPlate: one Stringa,
    logs: set TaxiLog,
```

```
    currentLog: lone TaxiLog,
    numberOfSeats: one Int,
}{
    currentLog in logs
    no log: logs | log.date > currentLog.date

    #address = 1
    #phoneNumber = 1

    licenseID > 0
    numberOfSeats > 0
}

sig Float {}

// GPS position
sig Position {
    latitude: one Float,
    longitude: one Float,
}

// Signature representing a generic ride
// (from when the taxi driver accepts the call
// until he leaves the passenger on destination).
sig Ride {
    origin: lone Position,
    destination: some Position,
    beginDate: lone Int,
    endDate: lone Int,
    taxiDriver: one TaxiDriver,

    // passengers who booked a taxi
    registeredPassengers: some Passenger,

    // number of people in the taxi
    numOfTravelers: one Int,

    status: one RideStatus,
    isShared: one Bool,
} {
    beginDate > 0
    beginDate < endDate
    numOfTravelers <= taxiDriver.numberOfSeats
```

```
74      #registeredPassengers <= numOfTravelers
        (#registeredPassengers > 1) implies (isShared = True)
76      (#destination > 1) implies (isShared = True)
        #destination <= #registeredPassengers
78  }

80  sig TaxiLog {
        date: one Int,
82      position: one Position,
        status: one TaxiStatus,
84  } {
        date > 0
86  }

88  abstract sig TaxiStatus {}
    sig AVAILABLE extends TaxiStatus {}
90  sig BUSY extends TaxiStatus {}
    sig OFFLINE extends TaxiStatus {}
92
    abstract sig RideStatus {}
94  sig WAITING extends RideStatus {}
    sig ON_BOARD extends RideStatus {}
96  sig COMPLETED extends RideStatus {}

98  sig TaxiZone {
        number: one Int,
100     queue: one TaxiQueue
    } {
102     number > 0
    }
104
    sig TaxiQueue {
106     zone: one TaxiZone,
        drivers: set TaxiDriver,
108 }

110 // ----- FACTS -----

112 fact UniqueTaxiDrivers {
        no u1, u2: TaxiDriver | (u1 != u2 and
114         (u1.licenseID = u2.licenseID
                or u1.taxiNumberPlate = u2.taxiNumberPlate))
116 }
```

```
fact UniqueTaxiLog {
    // There should not be two taxi logs
    // for the same taxi driver in the same date.
    no tl1, tl2: TaxiLog, td: TaxiDriver |
        tl1 in td.logs and tl2 in td.logs
        and tl1.date = tl2.date and tl1 != tl2
}

fact UniqueTaxiZone {
    no z1, z2: TaxiZone | z1 != z2 and z1.number = z2.number
    queue = ~zone
}

fact UniqueUsers {
    no u1, u2: User | (u1 != u2 and
        (u1.username = u2.username or u1.email = u2.email))
}

// If a taxi driver participates is a ride,
// he should be busy for the entire duration of the ride.
fact BusyDuringRide {
    all t: TaxiDriver, r: Ride, log: TaxiLog |
        (r.taxiDriver = t and log in t.logs
        and r.beginDate <= log.date and r.endDate >= log.date)
        implies
        (log.status = BUSY)
}

// Two rides for the same passenger must not overlap.
fact OneConcurrentRidePerPassenger {
    all p: Passenger, r1, r2: Ride | (p in r1.registeredPassengers
        and p in r2.registeredPassengers and r1 != r2)
        implies
        (r1.endDate < r2.beginDate or r2.endDate < r1.beginDate)
}

// Two rides for the same taxi driver must not overlap.
fact OneConcurrentRidePerDriver {
    all t: TaxiDriver, r1, r2: Ride | (t = r1.taxiDriver
        and t = r2.taxiDriver and r1 != r2)
        implies
        (r1.endDate < r2.beginDate or r2.endDate < r1.beginDate)
```

```
160 }

162 // If the taxi driver is available,
    // he must be inserted in at least a taxi queue.
164 fact AvailableDriverInSomeQueue {
        all t: TaxiDriver | (t.currentLog.status = AVAILABLE)
166         <=> (some q: TaxiQueue | t in q.drivers)
    }
168
    // Each taxi driver must be inserted in at most one taxi queue.
170 fact OneQueuePerDriver {
        all t: TaxiDriver | (lone q: TaxiQueue | t in q.drivers)
172 }

174 // ----- ASSERTIONS -----

176 assert A {
        all t: TaxiDriver | (lone q: TaxiQueue | t in q.drivers)
178 }
    //check A
180
    // ----- PREDICATES -----
182
    pred show(){
184     #Passenger > 1
        #Ride > 1
186     #TaxiDriver > 1
        #Position > 1
188     #{x: Ride | x.isShared = True} > 1
    }
190
    run show for 4
```

Appendix A

Appendix

A.1 Software and tools used

- LaTeX for typesetting this document.
- GitHub[1] for version control and distributed work.
- Draw.io[2] for mockups.
- StarUML[3] for UML diagrams.
- Alloy Analyzer[4] to check the consistence of our model.

A.2 Hours of work

The statistics about commits and code contribution are available on GitHub [5]. Please keep in mind that many commits are actually group work (it's reported in the commit message).

- Eleonora Chitti: ?? hours
- Alex Delbono: 22 hours
- Pietro De Nicolao: 25 hours

[1] `https://github.com`
[2] `https://www.draw.io/`
[3] `http://staruml.io/`
[4] `http://alloy.mit.edu/alloy/`
[5] `https://github.com/pietrodn/se2-mytaxiservice`

Bibliography

[1] W3C, *HTML5 - W3C Recommendation 28 October 2014*, http://www.w3.org/TR/html5/.

[2] W3C, *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*, http://www.w3.org/TR/CSS21/

[3] Apple, *iOS Human Interface Guidelines*, https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/

[4] Google, *Android Developers - Design* https://developer.android.com/design/index.html

[5] Software Engineering 2 Project, AA 2015/2016 - *Project goal, schedule and rules*

[6] Software Engineering 2 Project, AA 2015/2016 - *Assignments 1 and 2*

[7] IEEE Standard 830-1998: *IEEE Recommended Practice for Software Requirements Specifications*

[8] MIT, *alloy: a language & tool for relational models* http://alloy.mit.edu/alloy/