Politecnico di Milano

Software Engineering II

myTaxiService Requirements Analysis and Specification Document

Authors

Giovanni Beri	852984
Biagio Festa	841988

Thursday 3rd December, 2015

Contents

1 Introduction

1.1 Purpose

Our team will project myTaxiService, which is a platform that will be used by the administration of a large city to manage the system of the taxi drivers in the city area. In particular myTaxiService will offer to the users the possibility to effectuate requests of a taxi ride using a web application or a mobile app, and it will also manage the choice of the taxi to assign to every users' request using a system of queues. Advanced features will also allow users to make a reservation of a ride, specifying origin, destination and picking up time of the ride.

This document aims to describe the high-level functionalities that will be offered by myTaxiService. The RASD is intended to be viewed by:

- the team of developers, in order to clarify every existing ambiguity in the assignment.
- the technicians of the government, that are going to evaluate the correctness of every assumption and decision described in this document.

1.2 Scope and Problem Description

The government of a large city aims at optimizing its taxi service. In particular, it wants to: i) simplify the access of passengers to the service, and ii) guarantee a fair management of taxi queues.

Passengers can request a taxi either through a web application or a mobile app. The system answers to the request by informing the passenger about the code of the incoming taxi and the waiting time.

Taxi drivers use a mobile application to inform the system about their availability and to confirm that they are going to take care of a certain call.

The system guarantees a fair management of taxi queues. In particular, the city is divided in taxi zones (approximately $2km^2$ each). Each zone is associated to a queue of taxis. The system automatically computes the distribution of taxis in the various zones based on the GPS information it receives from each taxi. When a taxi is available, its identifier is stored in the queue of taxis in the corresponding zone.

When a request arrives from a certain zone, the system forwards it to the first taxi queuing in that zone. If the taxi confirms, then the system will send a confirmation to the passenger. If not, then the system will forward the request to the second in the queue and will, at the same time, move the first taxi in the last position in the queue.

Besides the specific user interfaces for passengers and taxi drivers, the system offers also programmatic interfaces to enable the development of additional services (e.g., taxi sharing) on top of the basic one.

A user can reserve a taxi by specifying the origin and the destination of the ride. The reservation has to occur at least two hours before the ride. In this case, the system confirms the reservation to the user and allocates a taxi to the request 10 minutes before the meeting time with the user.

1.3 Goals

We have selected a list of objectives that we think myTaxiService has to reach:

1. To allow passengers to easily access to the taxi service.
2. To guarantee a fair management of the taxi queues.
3. To create a system that provides a programmatic interface, in order to let potential external developers to create additional services.
4. To allow passengers to make a reservation for a taxi ride in advance.
5. To guarantee a better reliability on the taxi ride reservation than on the normal taxi ride.
6. To create a system that gives the possibility to be monitored and administrated by a limited group of authorized users.

1.4 Domain Assumptions

We suppose that the following conditions are true in the analysed world:

1. The geographical area of the city is included in the coverage area of most common mobile communication technologies (3g, 4g) offered by main telecommunications companies
2. The taxi drivers signal themselves as available if and only if they really are
3. Passengers can access to the functionalities provided by the system only if they register to it.
4. When a taxi driver confirms a ride to a user, he's expected to actually take on the responsibility of the ride.
5. When a registered user send a ride reservation, the manually inserted data are correct.
6. The number of myTaxiDriver in the city is high enough to guarantee that 10 minutes are sufficient for the selected taxi driver to reach the picking up place of a ride request/reservation from the place in which he find himself when he accept a ride.We do that because we are obliged by the specifications to allocate a taxi for a reservation only 10 minutes before the time indicated in the reservation; we assume that parameter has been calculated by some experts as the right amount of time that allows taxi drivers to reach a pickin up place that statistically is a few taxi zones far from him.
7. The number of fake requests sent to the system is a low percentage compared to the number of the real ones.

1.5 Glossary

Software *or* System
The collection and the parts of programmes that compose the project. Also we will indicate with these words the components finalized at the user (*documentation* and *user interface* application).

Actor
An entity that interacts with the system. It can be either a physical user or a external component.

Taxi zones
Geographical areas of approximately $2km^2$ in which the interested city is divided. Taxi zones are not overlapping, so each point of the city is assigned to exactly one taxi zone.

Ride request
A request that is sent by a registered user to signal that he needs to be picked up by a taxi in the immediate future. The request includes the GPS coordinates of the position in which the registered user desires to be picked up.

Ride reservation
A request that an user sends to book a taxi ride in the "future". For "future" we intend an instant of time that is almost 2 hours distant from the instant in which the request is sent. It must contain origin and destination of the ride and picking up time and date.

Fake request
A Ride reservation or ride request that turns out to be not a real one, because the passenger who sent it did not show up in the indicated place (and, in the case of a ride reservation, at the indicated time and date)

ETA
The *estimated time of arrival.* It can be used to generate estimated times of arrival depending on either a static timetable or through measurements on traffic intensity. [1]

API
Application Programming Interface: is a set of routines, protocols, and tools for building software applications. [2]

1.6 Further Developments

The software *myTaxiService* has some limitations that could be developed in future versions:

- An embedded electronic payment method, that allows users to pay in a faster and safer way.
- Allow *myTaxiUsers* to leave a feedback after every ride, in which they can evaluate different aspects of the offered service.
- Include an internal call system that allows a passenger to contact the *myTaixDriver* he is assigned to.
- Signalation of lost-and-found objects.

[1]Wikipedia definition: `https://en.wikipedia.org/wiki/Estimated_time_of_arrival`
[2]Wikipedia definition: `https://en.wikipedia.org/wiki/Application_programming_interface`

1.7 Used Tools

The tools used in the creation of this RASD are:

Latex (Live 2013)
to redact and to typeset this document.

StarUML (2.0.0-beta 10)
to create UML Diagrams (*Use Cases, Activity, . . .*).

Inkscape (0.91)
to support for the Scalable Vector Graphics (*SVG*).

Dropbox *and* Google Docs
to provide us a simple mechanism to share files and synchronize our work.

2 Specific Requirements

In according to 'Scope and Problem Description' subsection 1.2 (see page 3), we will specify in detail the requirements of the system.

The following are the requirements of the software.

2.1 Functional Requirements

The following are the *functional requirements* of the software, concerning each *actor* of the system.

Guest User

The system only provides him the possibility to become a registered user, so he's allowed to:

1. registrate to *myTaxiService* as *myTaxiUser*.
2. send request in order to become a *myTaxiDriver* of the system.

Registered User

He can either be a *myTaxiDriver* or a *myTaxiUser* or a *Administrator* or a *Third Party Developer*; in order to autenticate, the system provides him the functionalities concerning the account management; so the system allows him:

3. to login into his account.
4. to do a password recovery in case the user has forgotten it.
5. to update account data.

myTaxiUser

He's a registered user who has availability of all the functionalities that the service gives to the passengers in order to accomplish goals 1 and 4 (see subsection Goals at page 4). So, the system allows him:

6. to send a ride request; the system will provide the following ways to get the picking up position:
   - 6.1 automatically detecting the user's position through GPS technology, or another equivalent technology (if supported by his device).
   - 6.2 letting the user to manually insert the desired address, if GPS data are not reliable or not available.
   - 6.3 letting the user to select the picking up position from a list of remarkable locations placed nearby the coordinates got by GPS signal.
7. after a ride request has been accepted, to check the estimated time of arrival, calculated by the system in function of the traffic condition and the position of the arriving taxi.
8. to cancel a *ride request* or a *ride reservation*.
9. to send a ride reservation; to select the origin and destination of the ride, the user has availability of all the methods listed at points [6.1], [6.2], [6.3]. A ride reservation must be made at least 2 hours before the ride.

myTaxiDriver

He's a registered user for whom all the offered functionalities are needed to accomplish the goal 1 (see subsection Goals at page 4), because they all are complementary to the services provided to passengers. The system allows him:

10. to manage an incoming ride request, by:

    10.1 showing him the ride details:

    - 10.1.1 the GPS coordinates of the picking up place selected by the passenger
    - 10.1.2 estimated time of arrival in the place mentioned above

    10.2 letting him choose to accept or decline the incoming request

11. to cancel the confirmation of an already accepted ride. It should avoid the block of the system in situations like a broken engine, an accident or other technical problems that prevent the taxi driver from managing a ride that he has accepted.

12. to signal his state: available or busy.

13. to signal that a ride request he accepted was a *fake request*. The system provides this functionality to the taxi driver only after he has waited 5 minutes for the passenger, in order to avoid false negatives. This functionality is necessary because the system needs to determine which users made the *fake request* and to take measures according to the company policies.

14. to confirm a ride, i.e. signal that he has actually reached the passenger who sent the handled *ride request* or *ride reservation*.

15. to visualize a map showing the *CityZone* partition of the city, and the distribution of *myTaxiDrivers* in the zones.

Third Party Developer

16. The system must provide the access to some of its functionalities (basically the *myTaxiUser* ones) using a programmable interface, in order to enable the development of additional services on top of the basic one. The accessible functionalities will be clarified in the next sections.

Administrator User

The system has to provide to a small group of registered users the access to some advanced functionalities, concerning the control of the system itself. So the administrator users will be allowed:

17. to create new *Administrator* account.

18. to visualize informations about position and state of all *myTaxiDrivers*.

19. to activate the *myTaxiDrivers* account.

2.2 Non Functional Requirements

2.2.1 Ride Request Security

We assume that the number of fake requests will not be very high (in accordance with assumption [6], see page 4), because only registered user are allowed to use use *myTaxiService* and because no one would have advantage of doing this; so we decided to limit the strictness of some safety measures in order to improve *usability.*

However, we decided to take the following security measures to prevent an excessive amount of *fake requests* or cancellations:

1. A *myTaxiUser* is not allowed to send a *ride request* if it still exists another *ride request* that hasn't been assigned to a taxi driver or cancelled yet.

2. When a *fake request* is signaled by a *myTaxiDriver*, the *myTaxiUser* who sent it is notified with a warning and, if another fake request of the same user has already been notified in the previous 30 days, he is prevented from sending another *ride request* for the following 30 days.

3. When a *ride request* is cancelled, the *myTaxiUser* who sent it is notified with a warning and, if another cancellation from the same user has already verified in the previous 30 days, he is prevented from sending another *ride request* for the following 30 days. This safety measure is not applied in case of a *ride reservation* cancelled after the expected picking up time, because we assume that this is dued to a taxi driver lateness.

4. If the first ride (either a *ride request*) is signaled as a *fake request* or is cancelled by the user, it isn't necessary to another *fake request* or cancellation to happen before the banning measures are taken.

2.2.2 Ride Reservation Security

From the safety point of view, *ride reservations* are considered only as *ride requests* sent 10 minutes before the picking up time indicated in the reservation. So, every safety measure described in the previous paragraph is applied also to *ride reservations.*

2.2.3 Taxi Queues Policy

As written in the specifics document, the city is divided into different zones. Every city zone handles a unique queue in which taxis are registered. Each taxi is associated to one single queue that is the one linked to the city zone in which he find himself.

In order to guarantee a fair workload distribution among taxi drivers, each of them is associated to a priority order. Each taxi driver's priority is comparable with another one, even if two taxies belong to two different city zones. In that way if a taxi driver is moving and then changes its associated queue, in all cases the system can insert it into a new queues position coherently with other taxis priorities.

The maximum priority in a queue is assigned to the taxi drivers who has been waiting for longer.

Whenever a taxi driver changes the own state making himself no available, then he will be placed out from the queue.

Whenever a taxi driver who was no available becomes available, he will be inserted in the queue associated to the taxi zone in which he find himself, with the lowest priority.

Whenever a taxi driver notifies a fake request, he will be reinserted in the queue with available state and with the same priority that had before accepting the request.

Whenever a taxi driver manages *in time* a ride reservation he will get a bonus, consisting in the restoring of his priority at the same level he had before accepting the reservation management. We say that a ride reservation is managed *in time* if the taxi driver reaches the picking up point within the hour indicated in the reservation. This requirement aims to accomplish goal number [5].

If a *ride request* has a picking up point that belongs to a *city zone* which *taxi queue* is not able to handle it (*i.e.* the city zone doesn't contain any available *myTaxiDriver*), the system will forward the *ride request* to the nearest *city zone* which can handle that request.

2.2.4 Programmable Interface (API)

API Introduction

In order to extend its functionalities the system shall provide an programmable interface. That interface, from now on it called API (see page 5 for glossary), represents a way for programmers to create own application which can interact with the system's data and functionalities.

In general, the myTaxiService API uses HTTP/s POST requests with JSON arguments and JSON responses. In the figure 1 we can see how an external application can interface with the system.

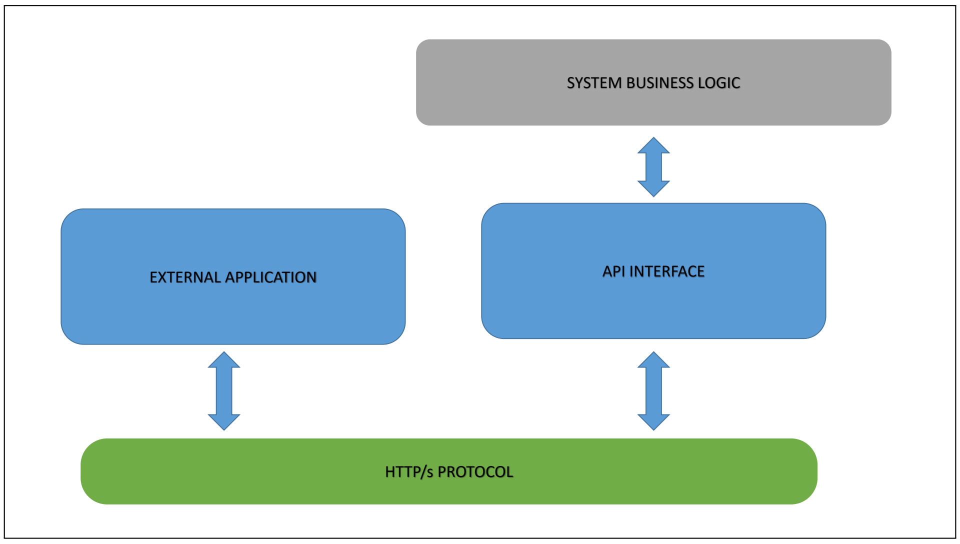


Figure 1: API conceptual structure

Security
A fundamental and critic point is about the security. The system shall provide a mechanism to identify each external application which interfaces with the system. In order to do that it's necessary that each developer has to register and use a kind of *passphrase* to identify his application. More details are demanded to the design document.

Another point about the security is the mechanism concerning user's sensitive data. The system must protect user's critical information such as the password. For this reason is mandatory that every external application must not handle user's information directly. Indeed the authentication mechanism is demanded to the myTaxiService system itself even if the user is using an external application.

The Figure 9 (see page 29) shows how this mechanism should work.

When an external application has to authenticate an user, it invokes an appropriate API's function: the user will be redirected on a myTaxiService web page, note that it is a page of the system itself. On that page the user can enter his data without any third part system can read them. At that point the user is authenticated and the system notifies the external app that the user has been recognized. The external application can access user's functionalities now.

3 Scenarios Identifying

We report below a list of possible scenarios, involving all the existing types of user.

3.1 Scenario 1

Alice has gone to a concert held in a place called Circolo Magnolia in the suburbs of the city, and it has finished late. The public transport doesn't cover that area, so she decides to call a taxi. She has already downloaded the myTaxiService app and sheìs already a *myTaxiUser*, so she only picks up her smartphone and selects the app. The position identified by the GPS seems not not to be very accurate, but she notices that Circolo Magnolia appears in the list of remarkable places around her, so she selects it as the picking up place and sends a *ride request*. After about 20 seconds, the data of the taxi she has been assigned to and an ETA of 5 minutes appear on the screen. After 6 minutes, the taxi arrives at the Circolo Magnolia and she gets in the car.

3.2 Scenario 2

Conan lives about 20 kilometers far from the city, but he's going there by train because he will have a job interview at 5 o'clock. He doesn't want to arrive tired at the interview so he decides he will take a taxi; with his PC he searches informations online about the taxi system in the city and he discovers the web application of myTaxiService. He finds out that a reservation service is provided, so he decides to become a *myTaxiUser* to enjoy this service. He registers to the service and, after having logged in, he selects the "Make a ride reservation" button. The system proposes Conan's home as origin of the ride but Conan changes it and select the address of the train station of the city. He inserts also the address of the place of the job interview as destination of the ride, and selects 15 minutes after the expected time of arrival of his train as picking up time, on the date of tomorrow. After having checked all the inserted data, Conan confirms his reservation. The next day, the train is late so Conan doesn't want to cause inconvenience, so he downloads the myTaxiService app and cancel his reservation. It's too late to make another reservation because he's going to arrive at the station in less than 2 hours, so he waits that the train arrives at destination and then he makes a *ride request*, selecting the position detected from the GPS as picking up place.

3.3 Scenario 3

Franco is a myTaxiDriver, and he's waiting for incoming requests. At a certain point a *ride request* arrives, and he accepts it by touching the "Accept" button. He drives to the indicated position, but he isn't able to see anyone looking for a taxi. After a few minutes he concludes that it must have been a silly joke, so he signals the *ride request* as *fake request*. He stops the car in order to wait for another request, but immediately another *ride request* is sent to him.

3.4 Scenario 4

Goku is a taxi driver, and he has been informed by the government of the city that a new system of taxi rides management called *myTaxiService* has been set, so he decides to become a *myTaxiDriver* because he wants to verify the goodness of the new service. He downloads *myTaxiService* app on his smartphone and follows the procedure to become a *myTaxiDriver*. After he has compiled the form, the app informs him that his registration request will be analyzed in the next hours. When he's at work, the day after, Goku is notified that his registration request has been approved, and he has become a *myTaxiDriver*. He wants to immediately try the application, so he logs in and puts his state as "Available". After 10 minutes he is notified that a *ride reservation* with origin place 2 kilometers away from his position can be assigned to him, and he accepts to take care of it by touching the "Accept" button. He runs like hell and arrives at the right place before the expected time, so he's notified that he will enjoy of a bonus.

3.5 Scenario 5

Oswald is a government employee, and with the introduction of new service *myTaxiService*, he has been selected to be an administrator user; more specifically, his assignment consists of the activation of the *myTaxiDriver* accounts. Today he arrives at work, and accesses to *myTaxiService* with his credentials from his PC. He checks the registration requests, and confirms all of them except one, in which the indicated licence number didn't match with the government policies.

3.6 Scenario 6

Murdoch is a software developer, and he's now working with a team of other developers in order to create *asFastAsPossible*, a software that helps people to move inside the city as fast as possible. The specifications of the software include also taxi service in the list of considered ways to move inside the city, but the project deadlines are not really far so Murdoch is now searching online, looking for an external service to include in their application. He discovers that *myTaxiService*, the notorious app that revolutionized the concept of taxi service, offers an interface that can be used by external developers to create additional services. He' already a *myTaxiUser* and he knows the quality of that service so he decide to use it. The *myTaxiService* API will be used in *asFastAsPossible* to provide users the possibility to make a simple taxi call, or to share a taxi ride with another user that is looking for a taxi for approximately the same path in the same moment.

4 UML Models And Use Cases

4.1 Use Cases Diagram

The following is a general use case of the system.

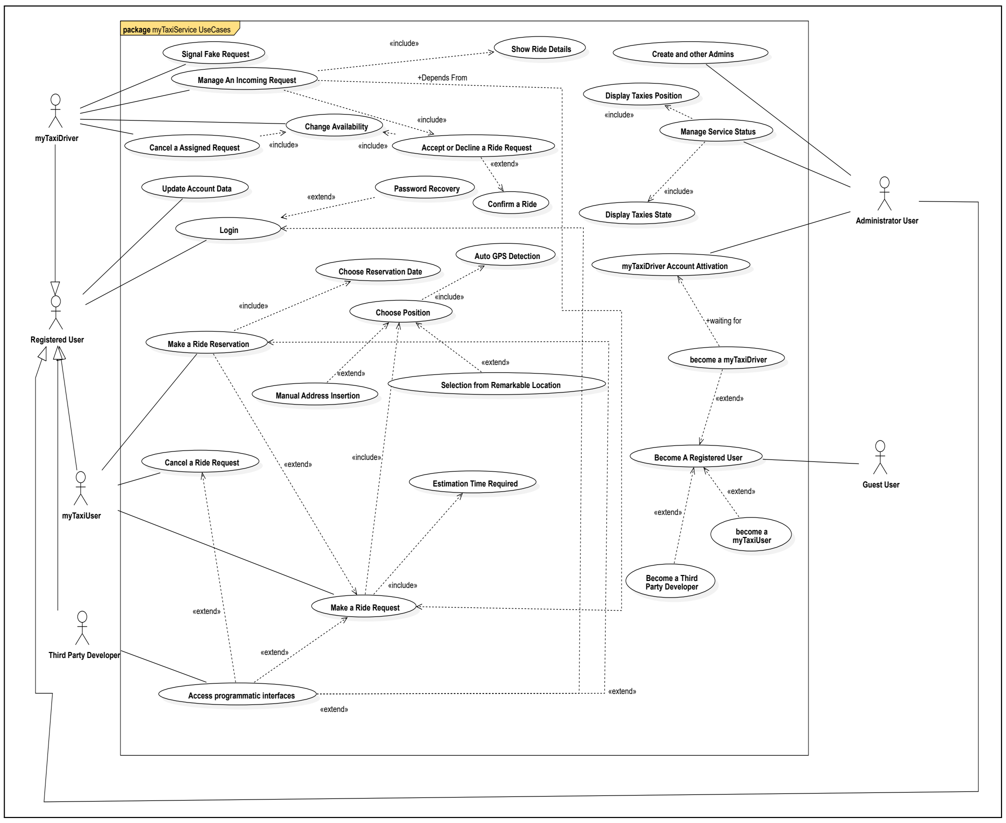


Figure 2: Use Case: General usage

4.2 Actors Identifying

Guest user
: A person who is using the system for the first time, or however he is not become a registered user (*myTaxiDriver* or *myTaxiUser*) yet.

myTaxiDriver
: A person who is registered to *myTaxiService* as a taxi driver; he needs to be already a taxi driver in the considered city.

myTaxiUser
: A person who has become a registered user of the service, in order to access to all the functionalities offered by the system to the potential passenger.

Administrator
: A person that has been selected by the application owners to perform administration tasks.

Third Party Developer

A person involved in an external project that has become a *Third Party Developer* in order to be authorized to access the programmatic interfaces offerd by *myTaxiService*.

4.3 Use Cases

4.3.1 Login

Name	Login
Actors	Registered User
Assumptions	• The user has successfully signed up to the system. • The user is not logged to the system yet.
Flow of events	• The user opens the '*welcome page*' of the system. • The user declares his identity. • The system recognizes the identity and ensures that the user who is logging it is who he claims to be. • The user can visualize his personal page and access to system's functionalities provided to him.
Exit conditions	The user is logged into the system now.
Exceptions	The informations inserted are wrong, an error message is shown. The user is not logged.

References: *User requirements* n. 3 (see page 7).

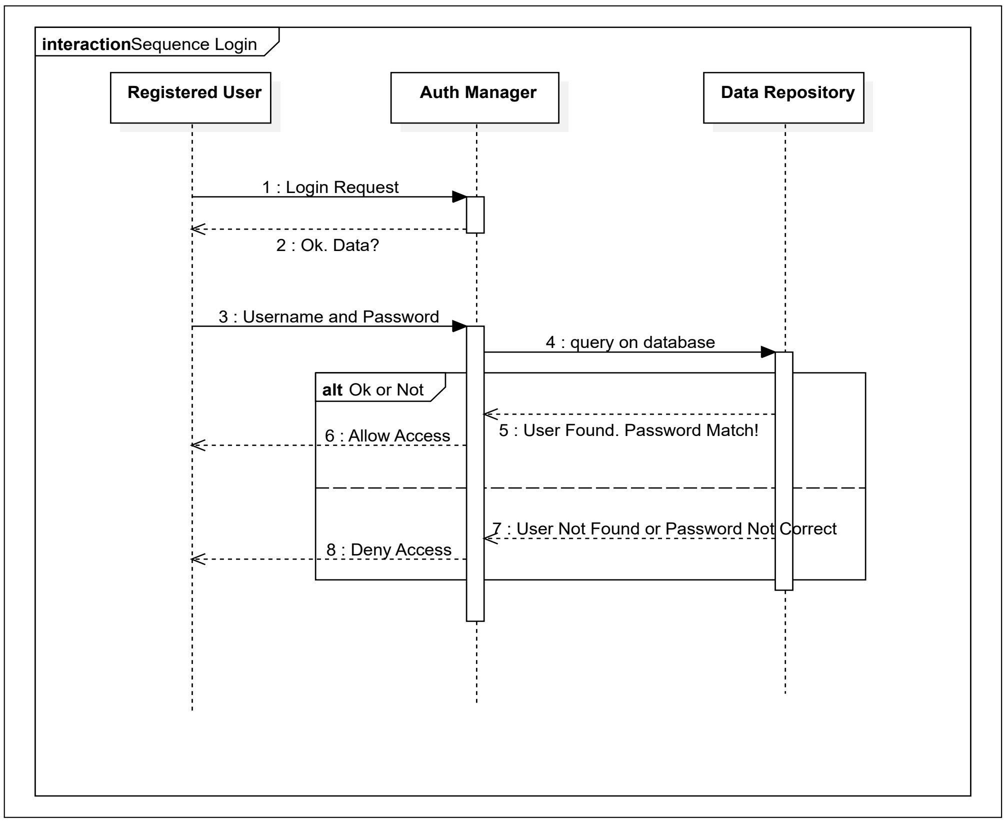


Figure 3: Sequence Diagram: Ride Request

4.3.2 Make a Ride Request

Name	Make a Ride Request
Actors	myTaxiUser
Assumptions	• The user really need a taxi.
Pre-Conditions	• The user has successfully signed to the system. • The user hasn't made another request before.
Flow of events	• The user uses the system's functionality and notifies his intention for requesting a taxi. • The system uses GPS technology to locate user's position. • The system shows to the user a map of the zone and a pin which indicates the detected position. • The user can confirm the position's correctness or improve it either with inserting address or with selection of a landmark from a nearest-place list. • The system forwards the request and once it has been assigned, shows to the user an approximate time for the taxi arrival.
Post-Conditions	• A taxi is allocated and he is coming to collect the user who made the request.
Exceptions	• An exceptional case can happen when there are no available taxies. In this case the system notifies the user the unavailability of the service, suggesting to try later again.

References: *User requirements* n. 6 (see page 7).

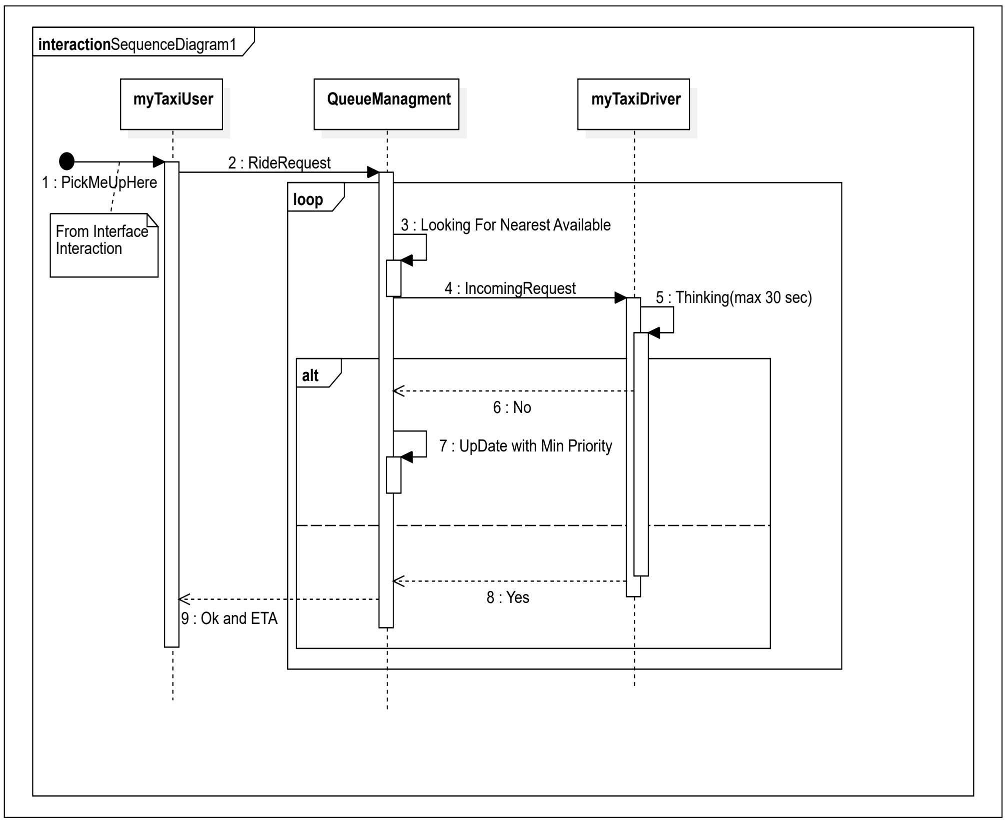


Figure 4: Sequence Diagram: Ride Request

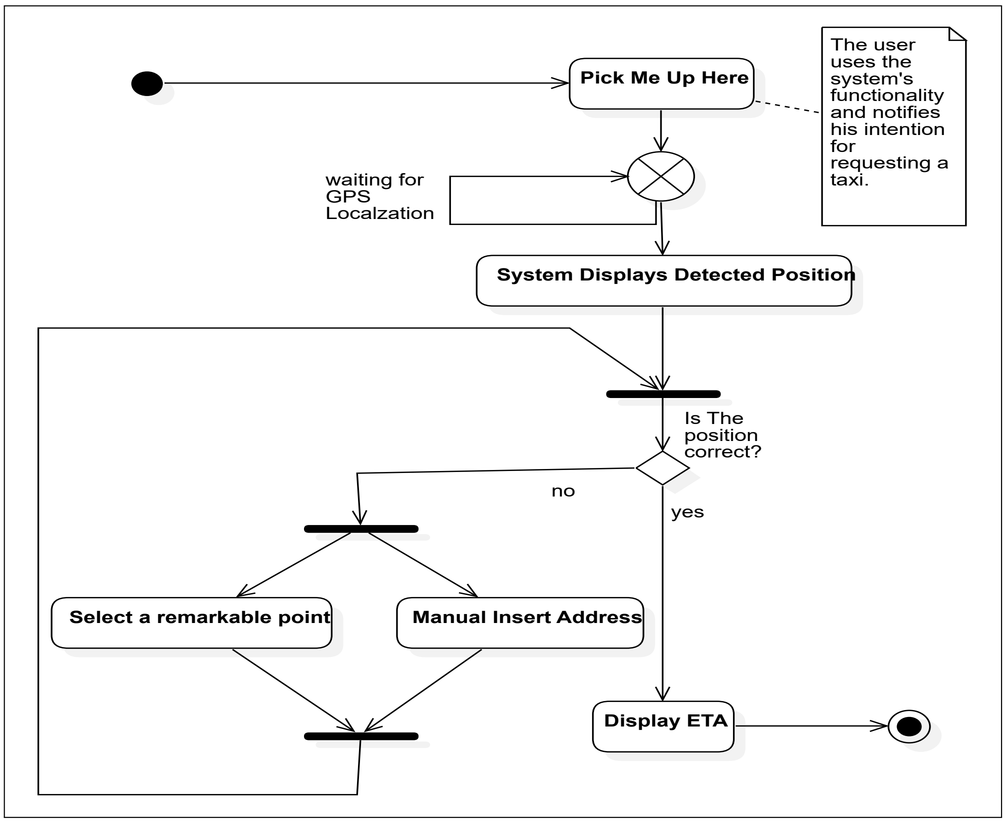

Pick Me Up Here
The user uses the system's functionality and notifies his intention for requesting a taxi.
waiting for GPS Localzation
System Displays Detected Position
Is The position correct?
no
yes
Select a remarkable point
Manual Insert Address
Display ETA

Figure 5: Activity Diagram: Ride Request

4.3.3 Make a Ride Reservation

Name	Make a Ride Reservation
Actors	myTaxiUser
Assumptions	• The user really need a taxi.
Pre-Conditions	• The user has successfully signed to the system. • It doesn't exists any not concluded ride reservation made by the same user.
Flow of events	• The user uses the system's functionality and notifies his intention for reserving a taxi • The system uses GPS technology to locate user's position. • The system shows to the user a map of the zone and a pin which indicates the detected position, and proposes it as the origin and destination of the future ride. • The user can confirm the positions' correctness or change one of them, or both, either with inserting address or with selection of a remarkable place from a nearest-place list. • The system shows to the user a calendar from which he can select the day and hour of the ride starting from 2 hours after the current time • The user chooses a date and an hour from the calendar • The system save the reservation data and notifies the user of the stored reservation
Post-Conditions	• The reservation of the user has been stored in the system, waiting to be processed
Exceptions	• A connection fault or an internal system error makes impossible to store the reservation data. The user is notified of the error

References: *User requirements* n. 9 (see page 7).

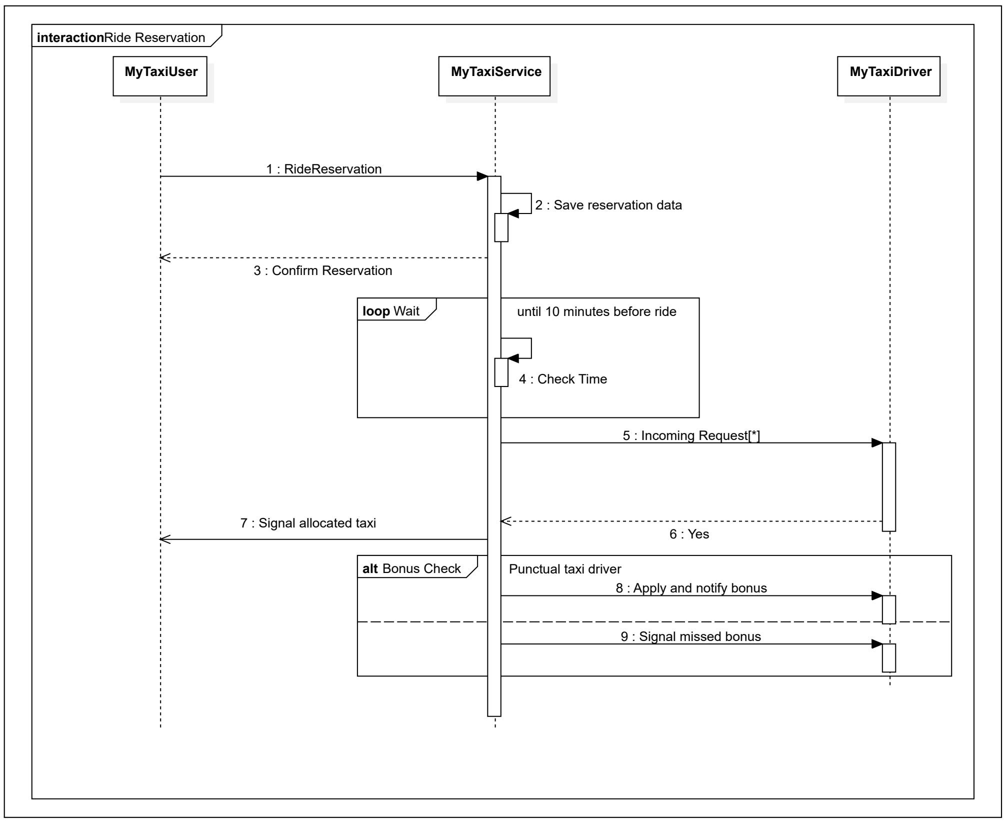


Figure 6: Sequence Diagram: Ride Reservation

4.3.4 Manage an Incoming Request

Name	Manage an incoming ride request
Actors	myTaxiDriver
Assumptions	• The taxi driver is really available and ready to manage an incoming request.
Pre-Conditions	• The taxi driver is been successfuly authenticated in the system. • The taxi driver is signed as available.
Flow of events	• The taxi driver displays a incoming request through his mobile application running on his phone. • The system provides the taxi driver all information about the incoming request: information about the user who performed the request, and the eventual meeting point. • The system shows to the taxi driver the precise position of the request's starting point, displaying to him a map and an estimation time in order to reach it. • The taxi driver has a limited amount of time to make a decision: the taxi driver can either accept or decline the request. • If the taxi driver accepts the request, he signals if the request is a fake or confirms the ride.
Post-Conditions	• Whether the request has been accepted the taxi becomes not available anymore and the request is solved: a taxi is allocated and he is going to collect the user who made the request. • Whether the request is declined the taxi driver's application will set the taxi driver's state as "*Busy*".
Exceptions	• An exceptional case can happen during the taxi driver's decision time, the mobile application cannot communicate anymore for any reasons. In this unlucky case the answer of the taxi driver has to be interpreted as declined.

References: *User requirements* n. 10 (see page 8).

4.3.5 Signal a *fake request*

Name	Signal a *fake request*
Actors	myTaxiDriver
Assumptions	• When a taxi driver confirms a ride to a user, he's expected to actually take on the responsibility of the ride.
Pre-Conditions	• The taxi driver has accepted a ride request before and he was supposed to going to collect the passenger.
Flow of events	• The taxi driver reaches the place indicated as the origin of the ride. • The passenger is not in the picking up place at the moment of the taxi driver arrival. • The passenger doesn't show up for the next 5 minutes. • The taxi driver signals the ride as a *fake request.*
Post-Conditions	• The taxi driver is not associated to that request anymore. • The taxi driver's priority is restored to the level reached before the ride acceptance. • The *fake request* signal is processed by the system, in order to take measures according to the company policies.
Exceptions	...

References: *User requirements* n. 13 (see page 8). Figure 7.

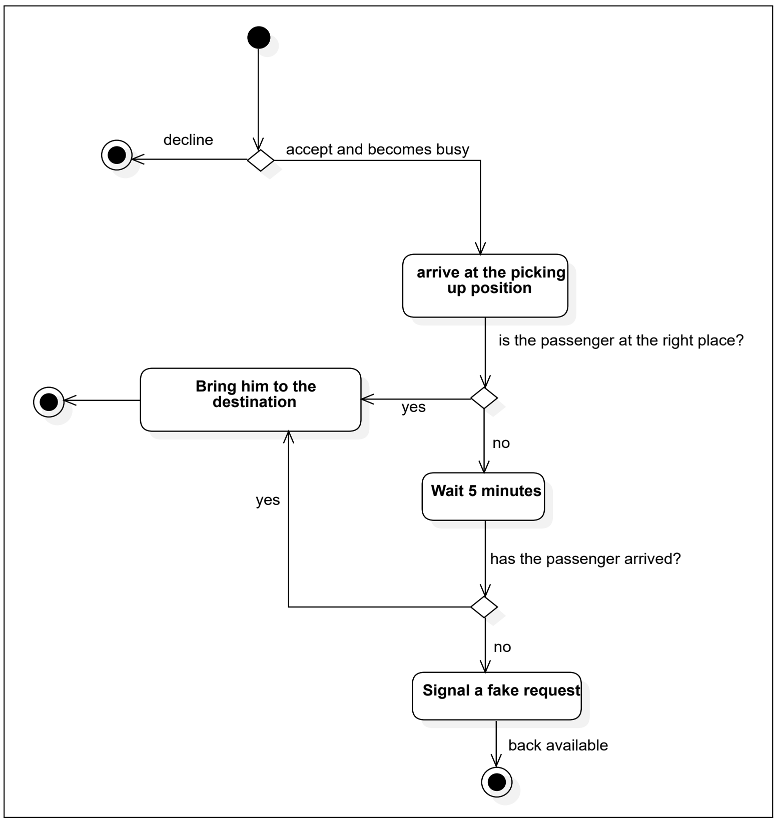

decline
accept and becomes busy
arrive at the picking up position
is the passenger at the right place?
Bring him to the destination
yes
no
Wait 5 minutes
yes
has the passenger arrived?
no
Signal a fake request
back available

Figure 7: Sequence Diagram: Signal Fake Request

4.3.6 Cancel a Made Ride Request

Name	Cancel a Ride Request
Actors	myTaxiUser
Assumptions	• There are few fake requests: that means the user really need to cancel the request due a sort of emergency.
Pre-Conditions	• The user has already made a ride request or reservation before.
Flow of events	• The user goes into his account management page. • The user displays the request made before and expresses his intention to cancel that request using the relative functionality provided by the system. • The user displays a confirmation message and accepts it.
Post-Conditions	• The ride request or reservation is dismissed. The taxi driver who was assigned to that request is notified by the system and he will get back available. • The user interface is updated: the user don't display the request annymore and he is allowed to make another request since now.
Exceptions	...

References: *User requirements* n. 8 (see page 7).

4.3.7 Cancel an Assigned Request

Name	Cancel an Assigned Ride Request
Actors	myTaxiDriver
Assumptions	• The taxi driver takes care about the passenger, we assume that functionality will be used only in emergency case.
Pre-Conditions	• The taxi driver has already accepted earlier a ride request and he was supposed to go to collect the passenger.
Flow of events	• The taxi driver's application is showing the ongoing request. • The taxi driver expresses his intention of cancel the previously accepted request.
Post-Conditions	• The taxi driver is not associated to that request anymore. • The taxi driver's application gets back in a standby state and it will be ready to accept the next incoming request. • The system will handle the cancellation in order to assign the request to another taxi driver.
Exceptions	...

References: *User requirements* n. 11 (see page 8).

4.3.8 Evaluate a *myTaxiDriver* account activation requests

Name	Evaluate *myTaxiDriver* account activation requests
Actors	Administrator
Pre-Conditions	• Some pending myTaxiDriver account activation requests exist. • The *Administrator* user has already logged in.
Flow of events	• The *Administrator* user visualize the list of all pending requests. • The *Administrator* choose a request to evaluate. • The request data (e.g. licence number) are verified and evaluated, according to the company policy. • If all the required data are evaluated positively, the request is approved; otherwise, the request is rejected. • The operations are repeated until the list is empty.
Post-Conditions	• Every taxi driver who sent an activation request is notified with the evaluation outcome. • All and only the *myTaxiDriver* accounts related to the positively evaluated requests are activated.

References: *User requirements* n. 19 (see page 8). Figure 19.

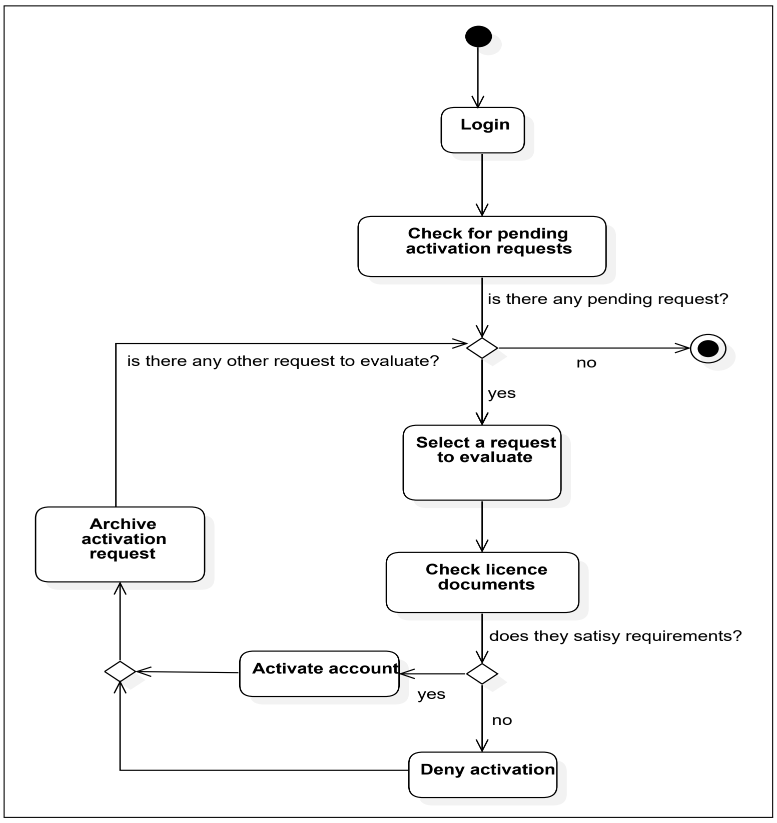

Login
Check for pending activation requests
is there any pending request?
is there any other request to evaluate?
no
yes
Select a request to evaluate
Archive activation request
Check licence documents
does they satisy requirements?
Activate account
yes
no
Deny activation

Figure 8: Activity Diagram: Activate Taxi Drivers

4.3.9 Login Through API

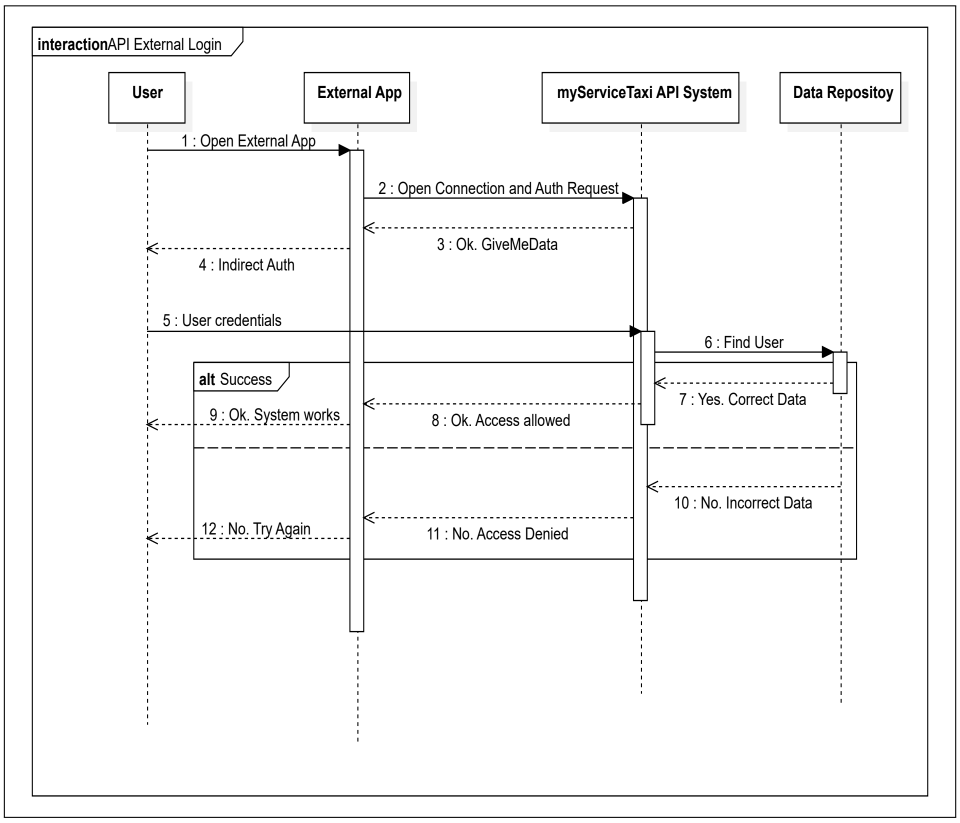


Figure 9: Sequence Diagram: Login through API system

4.4 Class Diagram

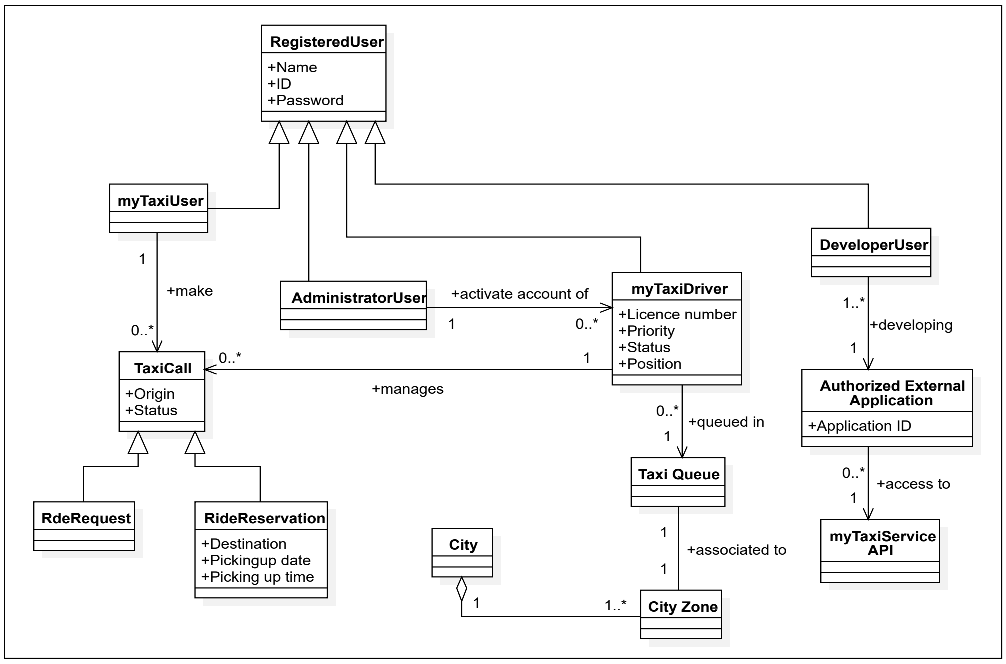


Figure 10: System Class Diagram

5 External Interfaces

5.1 Hardware Interface

The mobile application must be authorized to access the device locationing services, either GPS technology and network-based.

5.2 Software Interface

myTaxiService mobile application will use the Google Maps API to provide the map visualization and position insertion services.

5.3 Communication Interface

The mobile application must be authorized to access the internet connection of the device, either Wi-Fi connection or mobile data connection.

5.4 UserInterface

There are few points that are not very clear in the specification document, so we will assume that:

- In the specification document is not clear how the taxi-driver gets the mobile application. For this preliminary document we suggest two different approaches:
  - The system is composed only by *common* applications.

    Both user and taxi-driver mobile applications are stored and maintained in a *digital distribution platform* (such as *Google Play* or *App Store*). It means that everybody can download and use them.

    Although this is not a problem for passenger application (because our purpose is to reach as many people as we can), it could be for the taxi-driver application. Indeed someone could download the mobile application and spoofs others customers passing himself off as licenced taxi-driver.

    In order to avoid this problem, the system shall provide a kind of authentication mechanism. That could be solve by requirement [19] (see 8).

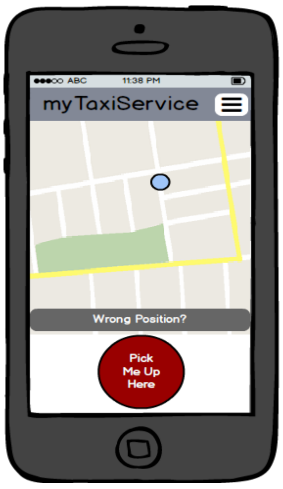
    Moreover this solution could be take into account for a possible selling of the entire system, but a study of this is beyond the scope of this document.
  - The passenger mobile application is stored as the previous point (via common digital market). Instead the taxi-drivers application is distributed through an internal system. *i.e. the city provides every taxi driver a mobile device with the pre-installed application.*
- From the passenger point of view, *myTaxiService* will be accessible from two different interfaces (see figures 11 and 12 for mockups):
  - Mobile Application: the user can access to all functionalities listed in the section 2.1 (see page 7).

  - Web Application: not all functionalities are at disposal of the users, because we assume that users access to web applications interfaces using a personal computer, so all informations related to the position detection are not useful. For that reason we decided to allow the user to make only *ride reservation* and deny to make a *simple ride request*.

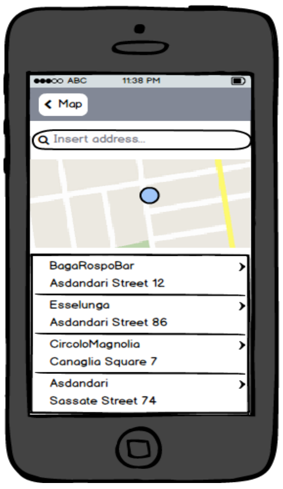
- The system will be accessible only via web browser interface. Indeed the administrator's work should be made using a personal computer and the design of another mobile application would have been an overkill.

5.4.1 Mockup User Interface

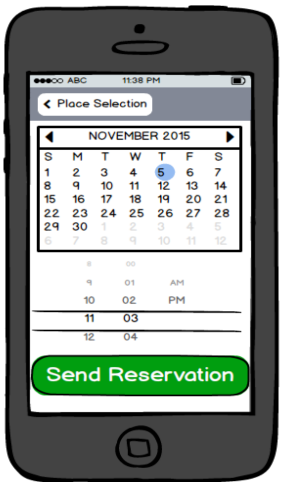
myTaxiUser Graphic Interface



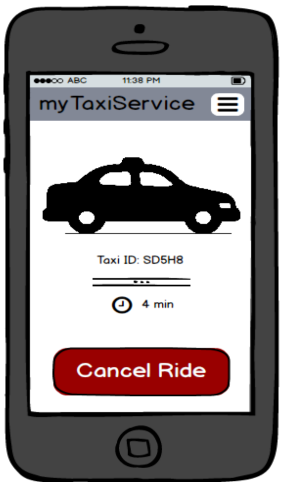
(a) *Pick Me Up Function.*



(b) *Manual Starting Point Address Insertion.*



(c) *Make a Reservation.*



(d) *Waiting For Taxi Arrival.*

Figure 11: Some mockups user interface for passenger user.

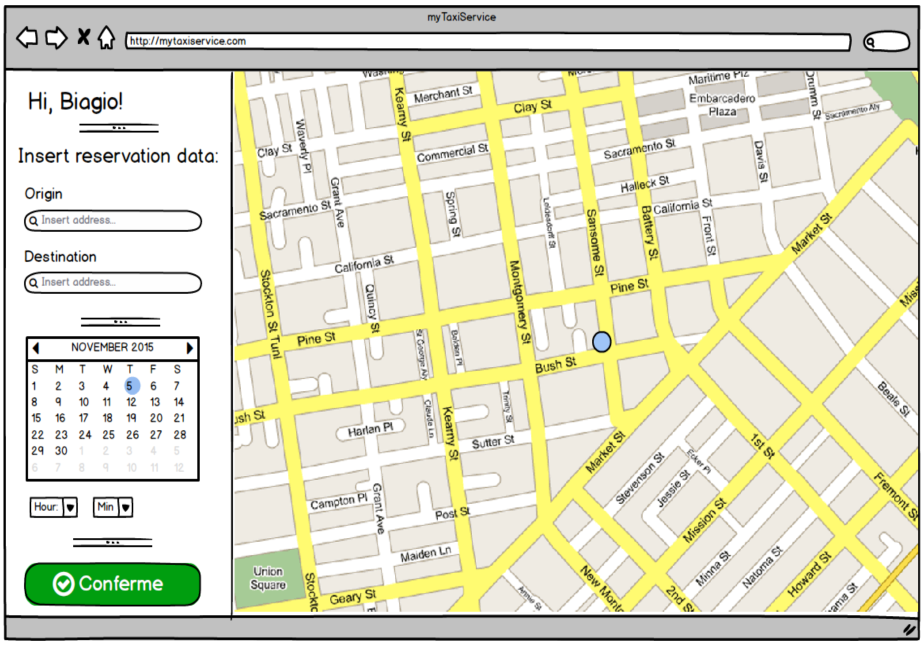
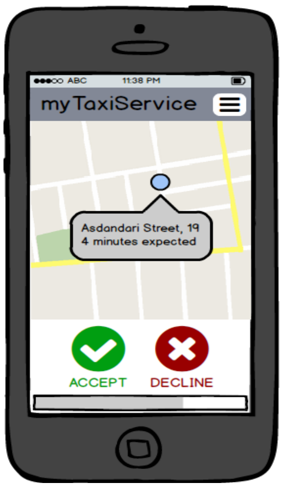
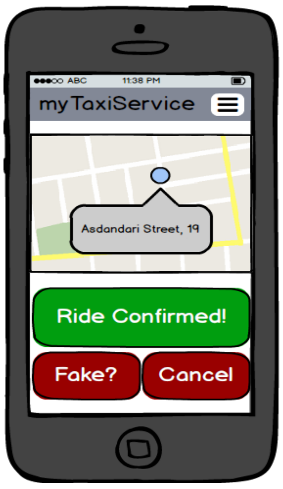


Figure 12: User web interface mockup

myTaxiDriver Graphic Interface



(a) *Manage Incoming Request*



(b) *Collecting The Passenger*

Figure 13: Some mockups user interface for taxi driver.

6 Alloy Model

In order to verify the consistency of our model and relative constraints, we will use the software *Alloy Analyzer*[3].

6.1 Alloy Source Model

The following is reported the complete model of the system in *alloy code*:

```
/*         myTaxiService
           ----Alloy Model-----

               Biagio Festa
               Giovanni Beri
*/

module myTaxiService
open util/boolean          //Module for boolean entities.
open util/ordering[Priority]     //Module for ordering priority. NOTE: 0
     highest priority

//    CITY ZONE: each of them has one TaxiQUeue.
sig CityZone
{
  has: one TaxiQueue,
  near: set CityZone
}

/*      TAXI QUEUE
    Each TaxiQueue manage a set of taxies.
    Each TaxiQueue manage RideRequest.
*/
sig TaxiQueue
{
  contains: set Taxi,
  assignedTo: RideRequest lone-> Taxi
}
{
  // Exsists one and only one zone assigned to this TaxiQueue
  one z:CityZone | z.has = this

  //All tuple (request,taxi) have to guarantee some proprieties:
    all t:Taxi, r:RideRequest | r->t in assignedTo
        implies
      t in contains   //it can assign request only for taxi which are in its
     zones.

  //Each request cannot be assigned to more than one taxi.
  all r:RideRequest, t1:Taxi | r->t1 in assignedTo
        implies
    no t2:Taxi | r->t2 in assignedTo and t1!=t2
}

//STATUS REQUEST.
abstract sig StatusRequest {}
one sig Pending extends StatusRequest { }
one sig Assigned extends StatusRequest { }

//  TAXI REQUEST: start from a specific zone and with its state.
sig RideRequest
{
  startFrom: one CityZone,
  status: one StatusRequest
}
{     //RideRequest properties:
  //If there is no queue handles this request then this is Pending (not
     allocate) else is Assigned
  (no q:TaxiQueue | q.assignedTo[this]!=none) implies (status = Pending)
     else (status = Assigned)
```

[3]See 1.7 (Used Tools)

```
}

//PRIORITY: concept for priority a queue.
sig Priority {}

// TAXI: can be avaiable or not and has own priority (note: the priority is
    global into the entire system)
sig Taxi
{
  availability: one Bool,
  priority: one Priority
}
{     //Taxi properties:
  //Every Taxi has to be in one (and only one) queue.
  one q:TaxiQueue | this in q.contains
}



//Returns all request handle by a queue.
fun ReqOfQ[q:TaxiQueue] : set RideRequest{
  Taxi.(~(q.assignedTo))
}

//Returns all taxi handle by a queue which are assigned for some request.
fun TaxiAssignedOfQ[q:TaxiQueue] : set Taxi{
  RideRequest.(q.assignedTo)
}

//Returns the taxi associated to a Ride Request.
fun TaxiOfReq[r:RideRequest] : set Taxi{
  r.(TaxiQueue.assignedTo)
}



/*    FACT
  If there is a taxi associated to one ride request then it will be
    unavailable.
*/
fact EachRequestAssignedMakesTaxiBusy{
  all q:TaxiQueue, r:RideRequest, t:Taxi | t = q.assignedTo[r] implies t.
    availability = False
}

/*    FACT
  Every Taxi is assigned to different priorities
  NOTE: This fact could be deleted, but it's useful to generate more
    particular cases.
*/
fact UniquePriority{
  all p:Priority, t:Taxi | t.priority = p implies (no t2:Taxi | t2!=t and t2.
    priority = p)
}

/*    FACT
  Describes the queue's priority.
  The Logic, briefly:
    For all taxi avaiable doesn't exsist another taxi in the same queue with
    lower
    priority which handle a ride request.
*/
fact PriorityPolicy{
  all t:Taxi, q:TaxiQueue | t in q.contains and t.availability = True
      implies
  (no t2:Taxi | some r:RideRequest |
      t2 in q.contains and r in ReqOfQ[q] and
      t2!=t and  r->t2 in q.assignedTo and
      t2.priority in t.priority.nexts)
}

/*    FACT
  No zones is near to itself.
*/
```

```
fact NoSelfNear{
  no c:CityZone | c in c.near
}

/*    FACT
  Whether a request is assigned into a different zone than starting
  then it must exist a path among those zones.
*/
fact ForwardOnlyIfNear{
  all r:RideRequest, q:TaxiQueue, c:CityZone | r in ReqOfQ[q] and c.has =q
     and r.startFrom != c
      implies
  c in r.startFrom.*near
}

/*      FACT
  A queue can handle a ride request which has been generated from the another
      city zone,
  only if the 'correct' queue has not taxies available.
*/
fact HandleForeignOnlyIfFull{
  all r:RideRequest | lone q:TaxiQueue | lone c:CityZone | (c.has = q and r
     in ReqOfQ[q] and r.startFrom != c)
      implies
  (no t:Taxi | t in r.startFrom.has.contains and t.availability = True)
}


/*    FACT
  Symmetric property for zones:   If z1 near z2 implies z2 near z1.
*/
fact SymmetricPropertyNearZones
{
  all disj c1,c2:CityZone | c2 in c1.near  implies c1 in c2.near
}

/*    FACT
  Cannot be two assignements for the same ride request.
*/
fact AnAssignedRequestCannotBeAssignedAgain{
  all disj q1,q2: TaxiQueue | ReqOfQ[q1] & ReqOfQ[q2] = none
}


//ASSERT:   Check if there is no taxi queue which assigns more request than
     taxi avaiability number.
assert QueueCanAssignMaxRequests
{
  no q:TaxiQueue | #ReqOfQ[q] > #q.contains
}
check QueueCanAssignMaxRequests for 5

//ASSERT:   One request is associated at most with one taxi.
assert EveryRequestIsAssiciatedAtMostOneTaxi{
  all r:RideRequest | #TaxiOfReq[r] = 1 or #TaxiOfReq[r] = 0
}
check EveryRequestIsAssiciatedAtMostOneTaxi for 10


/*Generate R as impossible request to serve
  Brief Description:
    R will be placed into a CityZone without taxi and without near zone.
*/
pred ImpossibleRequest[r:RideRequest]{
  #CityZone = 2
  some c:CityZone | #c.has.contains > 0
  #r.startFrom.has.contains = 0
  #r.startFrom.near = 0
}
SharingQueueLimitation :check{
  //Check if a RideRequest which is isolated and without taxi resource it
     will be never assigned.
  all r:RideRequest | ImpossibleRequest[r] implies r.status=Pending
}
```

```
/*    This predicate and assertion show that the priority of the queue is
      consistent.
     For all request and taxi assigned combination into a queue cannot exist
     another taxi assigned with lower priority.
*/
pred OneZoneOneAssignedRequest[r:RideRequest, t:Taxi]{
  #CityZone = 1
  #RideRequest = 1
  #Taxi > 10
  r.status=Assigned
  one q:TaxiQueue | r->t in q.assignedTo
}
PriorityPolicyCheck :check{
  all r:RideRequest, t:Taxi | OneZoneOneAssignedRequest[r,t] implies
    (no t2:Taxi | t2!=t and t2.availability=True and t2.priority in t.
     priority.prevs)
}


/*    This predicate shows the cosistency of the system with the QueuePolicy:
     when a queue has no available taxi then the request is handle by another
     queue which is near to the cityzone.
*/
pred CascadeSharingQueuePolicy[r:RideRequest]{
  #CityZone = 2
  some c:CityZone | #c.has.contains > 0
  #r.startFrom.has.contains = 0
  #r.startFrom.near = 1
  r.status = Assigned
}
run CascadeSharingQueuePolicy



//A general instance!
pred show{}
run show
```

Listing 1: Complete Alloy Code

6.2 Analyzer Result

And here the report of Alloy Analyzer:

```
6 commands were executed. The results are:
   #1: No counterexample found. QueueCanAssignMaxRequests may be valid.
   #2: No counterexample found. EveryRequestIsAssiciatedAtMostOneTaxi may be
     valid.
   #3: No counterexample found. SharingQueueLimitation may be valid.
   #4: No counterexample found. PriorityPolicyCheck may be valid.
   #5: Instance found.CascadeSharingQueuePolicy is consistent.
   #6: Instance found.show is consistent.
```

6.3 Model Representation

In this subsection we can visualize some instances generated by *Alloy* software.

As we can see in the figure 14, our world is composed by some ride request, city zone, relative queues, and taxies.

Let's indicate some properties:

- Every taxi queue is related with only one city zone.
- Every taxi queue manages a number of taxies.
- Every taxies, coherently with our specification document, has a state which can be available or not.
- Every taxies has one priority. Note that the priorities are entities which can be compared each other. The priority with lowest number is the first in the sequence and should be served before the other ones.
- Every ride request has a state which indicates whether the request itself is assigned or is waiting for an assignment.

6.3.1 Instance I: Overall World's Description

In this first representation we can see (in the figure 14) two request which have been sent by users but the system is processing them in order to find an appropriate taxi assignment.

As we can see there are three taxi managed by the queue, but only one of them is available. That means only one of the requests could be satisfied.

Despite of this both requests are waiting to be assigned, that why the model could represent a situation in which the taxi is thinking whether accept the request or not, moreover the system could still processing the request. Indeed the model is so coherent.

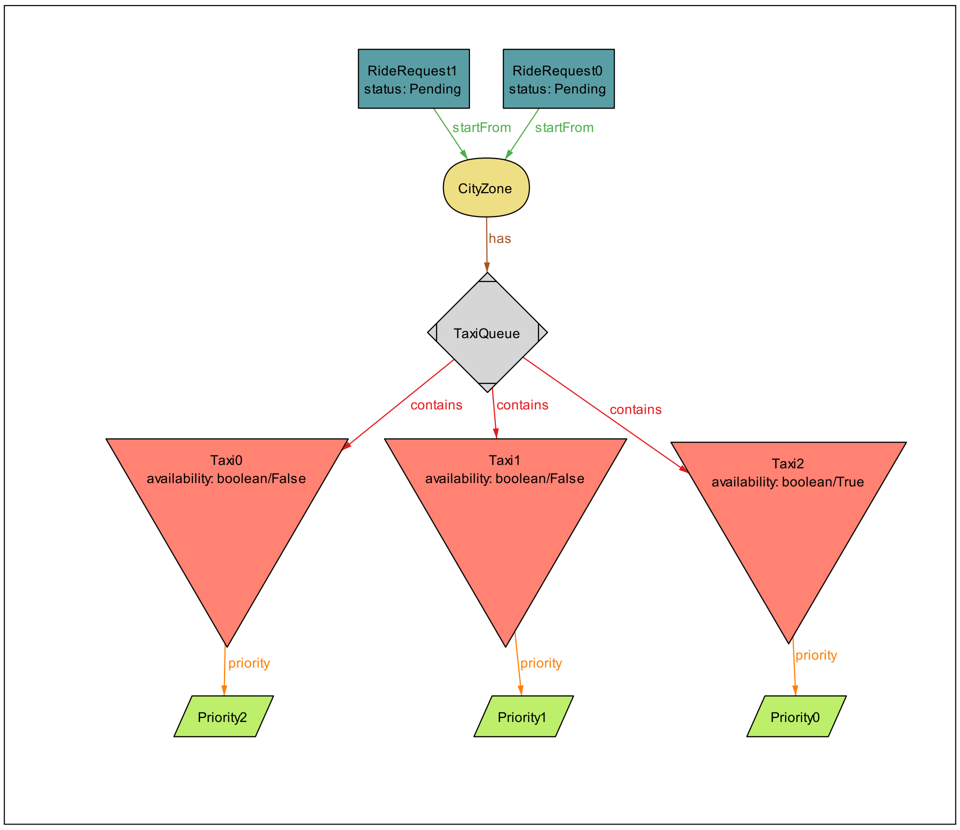


Figure 14: Alloy Result: Run World Instance 1

6.3.2 Instance II: Priority Assignment

In this scenario we can see better how the priority works. (See subsection "Taxi Queues Policy" [2.2.3]) In the figure 15 we still have two requests but they have been assigned. We can see how the "*Taxi Queue*" has assigned the "*Request 0*" to the "*Taxi 2*" and the "*Request 1*" to the "*Taxi 1*".

It's interesting note that despite of availability "*Taxi0*" has been not selected. That's why his priority is the lowest in the queue.

$$Priority0 > Priority1 > Priority2$$

Another notable case is showed in the figure 16. Indeed the single request has been assigned to a taxi with lowest priority. Despite of that the model is still valid because, as we can see, the taxi with higher priority is not available anyway. Indeed the request is forwarded to the "*Taxi0*".

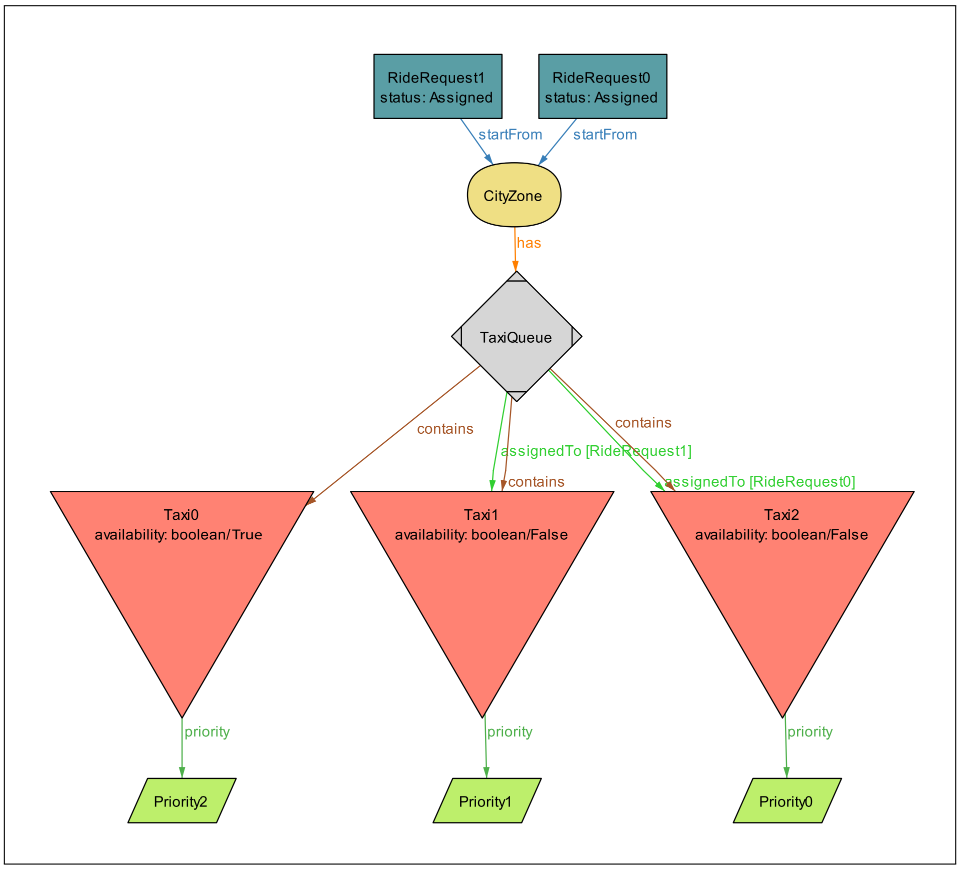


Figure 15: Alloy Result: Run World Instance 2

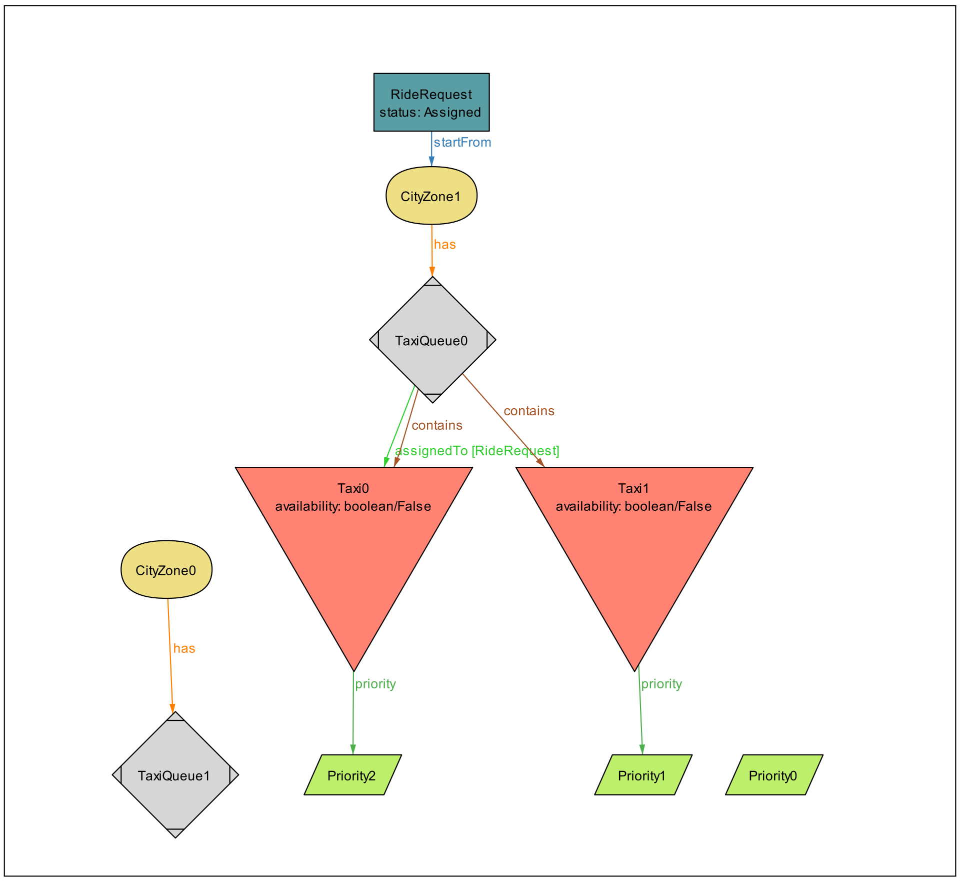


Figure 16: Alloy Result: Run World Instance 3

6.3.3 Instance III: Forwarding Policy

In accordance with the specifics, we have described in our alloy model what will happen if a certain queue cannot assign a request because of lack of resources (available taxies).

In the figure 17 we can see a particular instance of the "*CascadeSharingQueuePolicy*" predicate: when a queue has no available taxi then the request is handled by another queue referring another city zone which is near to the one from the request has been generated.

The "*Ride Request 1*" (*r*) has been generated from the "*City Zone 0*". The queue linked to that city zone has no available taxies then it cannot properly handle the request. Since "*City Zone 0*" is near to "*City Zone 1*" (*c*) the request is forwarded. In the figure 17 we can see how the request is indeed correctly assigned to "*Taxi 0*" by the queue.

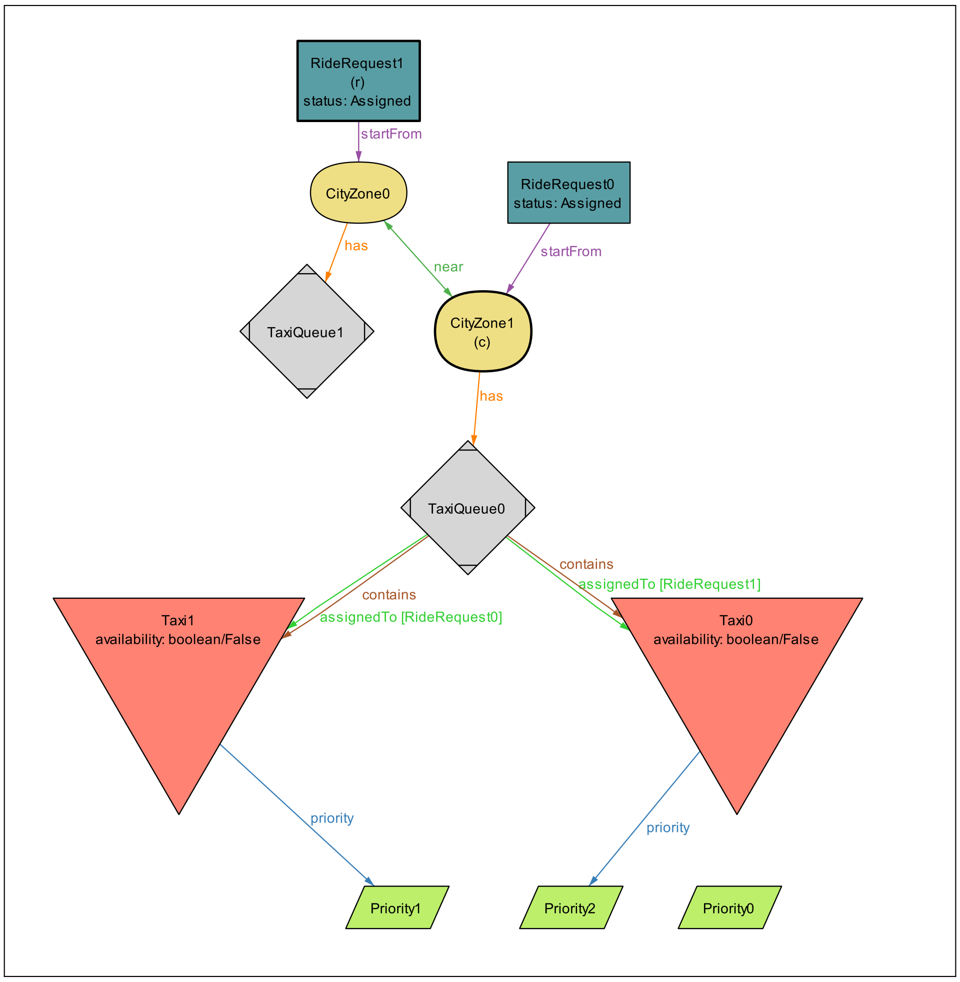


Figure 17: Alloy Result: Example of Forwarding Request

7 Hours of Work

- Beri Giovanni: ~ 30 hours
- Festa Biagio: ~ 30 hours