Requirements **A**nalysis and **S**pecification **D**ocument

Software Engineering 2: “myTaxiService”

Fioratto Raffaele, Longoni Nicolò
Politecnico di Milano

A.Y. 2015/2016

October 22, 2015

Contents

1 Introduction

1.1 Purpose

This document represents the Requirement Analysis and Specification Document (RASD). The main goal of this document is to describe completely the system in terms of functional and non-functional requirements, analyzing real needs of customers to model the system, showing constraints and limits of the software and simulating typical use cases that will occur after the development. This document is addressed to all developers and programmers who have to implement the requirements, to system analysts who want to integrate other systems with this one, and could be used as a contractual basis between the customer and the developer.

1.2 Actual System

The software house wants to offer a new service to optimize an existing taxi service of a large city. We suppose that until now, nothing has been created and we have to create the entire application without using or modifying a previous system.

1.3 Scope

The aim of the project is to create a new brand system that optimizes an existing taxi service. The system will be capable of automatizing and/or simplifying certain processes during requests or reservations of taxis. It will also guarantee a fair management of taxi queues. New passengers can sign up for service inserting some basic information in order to use service's features as soon as possible. A passenger can request a taxi using web service or through mobile application after registration. The system will be able to localize precisely the position of the passenger, determining the taxis that are available near him/her. The system will select a taxi and then will forward the request to its driver. Upon confirmation, the system will notify the customer about the successful completion of the operation and the ETA of the taxi. A passenger can also reserve a taxi by specifying the time and the date. Passengers who want to reserve a taxi should do it with at least two hours in advance, cancellation is also permitted. The system will actually process the request ten minutes before the time specified during the reservation in the same way as described previously.

1.4 Actors

- Visitor: a person that is not registered to the system or that is not already logged in. He/she can only sign in or visit the registration page.
- Registered Passenger: all customers that have a valid username and password and that are able to access all available features to customers, they can request or reserve a taxi, they can also cancel these operations with some constraints later explained.
- Taxi Driver: a person who is able to drive a taxi, owns a valid professional driving license. The driver informs the system about the availability of his/her taxi, logging in a specific mobile application installed in his/her smartphone, and do the same when the ride ends.

- System Administrator: the person in charge of updating the list of the actual cars and the list of qualified drivers.

1.5 Goals

List of goals for myTaxiService system:

- [G1] Allow visitors to sign up for the service
- [G2] Allow registered passengers to request a taxi
- [G3] Allow registered passengers to reserve a taxi
- [G4] Guarantee a fair management of the taxi queues
- [G5] Allow taxi drivers to manage their workday
- [G6] Allow the system administrator to update the information about drivers and taxis

1.6 Definitions, Acronyms, Abbreviations

1.6.1 Definitions

- System: it contains core functionalities of the software to be developed.
- Service: it is composed by the system, mobile application and web site.
- Location: it is a geographical point that univocally determine the position of something, it is defined by two coordinates in the two dimensional space generated by a map.
- Meeting Point: it is the location where one or more taxi drivers move to take the people requesting one o more taxis.

1.6.2 Acronyms

- RASD: Requirements Analysis and Specification Document
- DB: DataBase
- DBMS: DataBase Management System
- API: Application Programming Interface
- ETA: Estimated Time of Arrival
- OS: Operating System
- FIFO: First-In First-Out
- JVM: Java Virtual Machine
- RPC: Remote Procedure Calls
- JSON: JavaScript Object Notation

1.6.3 Abbreviations

- [Gn]: n-goal
- [Rn]: n-functional requirement
- [Dn]: n-domain assumption
- [An]: n-assumption

1.7 Identify Stakeholders

Our main stakeholders are government members of the city that are interested in optimizing the existing taxi service. Other stakeholders can be companies that want to enhance the service as well, but also the passengers and taxi drivers who want a more comfortable, intuitive and user-friendly service.

1.8 Reference Documents

- myTaxiService Specification Document
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.

1.9 Document overview

The document is essentially divided into four parts:

- Section 1: Introduction, it gives a description of this document and some basic information about software utilized.
- Section 2: Overall Description, gives general information about the software to project, focused on constraints and assumptions.
- Section 3: Specific Requirements, this part lists requirements, typical scenarios and use cases. To give an easy way to understand all functionality of this software. This section is filled with UML diagrams.
- Section 4: Appendix, this part contains information about the attached *.als files and some described screenshots of software used to generate it.

2 Overall Description

2.1 Product perspective

The final product will consist of a web service and a mobile application for customers and taxi drivers. The overall system will also have an internal interface for the system administrator. The system should also interact with existing external services, and will also provide additional APIs that allows the development of additional features in the future.

2.2 User characteristics

The final customer that wants to use this application is a person who wants to move around the city using taxis in an easier and faster way than an ordinary taxi service can assure. This user must have some kind of digital equipment that let him access the Web by means of a modern browser. The user can also use the mobile application to have a more enhanced user experience using this service. From the system administrator perspective, he expects a very intuitive interface that enables a fast way to manage the back-end information. We expect that an already hired employee will be trained to do such job.

2.3 Constraints

2.3.1 Regulatory policies

For the myTaxiService we have identified the following regulatory policies:

- Cookies: since it is mandatory to log in the web service to interact with the system, it is necessary to use cookies in order to keep track of each user's session, thus the system must comply with the current Cookie Law.
- Privacy: the system needs to manage some sensible data of customers and taxi drivers, for instance the name, the surname, an e-mail etc. Another crucial information is the current location detected by GPS of both. During the registration, customers must accept the Privacy Law (see login mockup), regarding taxi drivers, before being registered to the system by the administrator, drivers have to sign a document, accepting the processing of its personal data.
- Drivers during their work are subject to the Traffic code.
- The workday of a driver is subjected to the laws about Job safety, so the system must allow the driver to take some breaks during the day in order to prevent both illnesses, damage to people, tiredness problems and stress.

2.3.2 Hardware policies

For the myTaxiService we have identified the following regulatory policies:

- Smartphone: because of their limited power and battery, we must take care of application consumption and also web services (for instance, implementing a responsive design).

2.3.3 Interfaces to other applications

- Google Maps: it will be used to provide passengers an estimation of taxis arrival time.
- External Database: it contains multiple information about the drivers' license and data about cabs, the system will interrogate this database to check the correctness of the information supplied by taxi drivers during the registration in the system.

2.3.4 Parallel operation

myTaxiService must support parallel operations from different users when it works with database and with all operations done by users after connections.

2.3.5 Documents related

2.4 Assumptions and Dependencies

2.4.1 Assumptions

- [A1] Only registered users can use the service, because additional information such as name, surname and e-mail to confirm and to recap the reservation are needed, to avoid repetition, the user has to re-enter this information every time. Another reason is to prevent passengers to make requests to the system while another one is in processing.
- [A2] Passenger position is exactly known thanks to the usage of GPS available in smartphones or by using a geolocalization API from browser.
- [A3] The position of the taxi is exactly known by using the GPS available in the driver's smartphones.
- [A4] The city is divided into zones of 2 km^2 each and for each zone a taxi queue is allocated.
- [A5] The driver can take at most 2 breaks during his/her workday, during breaks the taxi belonging to him/her is not available.
- [A6] A taxi driver must own a valid professional driving license and a cab with all documents in order.
- [A7] Each taxi driver's license has a unique code that identifies it.
- [A8] Each taxi cab has a unique code that identifies it.
- [A9] An external database with all the correct and up to date information about the taxi drivers and their cars (valid driving license, cars allowed to circulate in streets, etc.)
- [A10] The existing taxi service is available 24 hours per day, 365 days per year.
- [A11] A taxi has a capacity of at most 3 passengers.

2.5 Future possible implementation

- Allow customers to pay using an electronic payment system. During registration the passenger will provide additional information about his/her credit card or PayPal e-mail, the system will charge a ride automatically.
- Extend the interface usable by customers to provide the ability to evaluate the service in general as a feedback to improve the overall system.
- Expand the system to allow the possibility to share a taxi with other customers using the existing API.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

3.1.1.1 Homepage The mockups below shows the home page and the initial screen showed when a user begins to use the service.

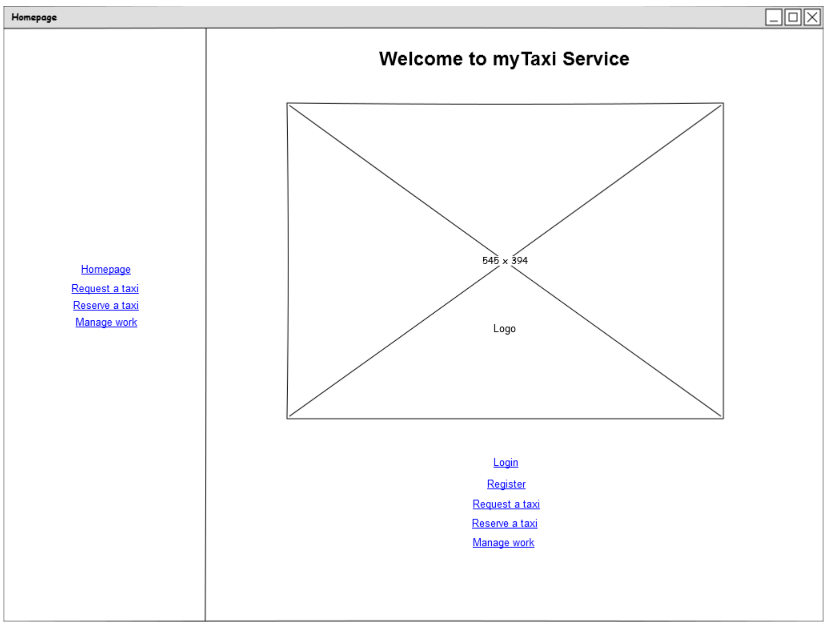


Figure 1: Homepage Web Version

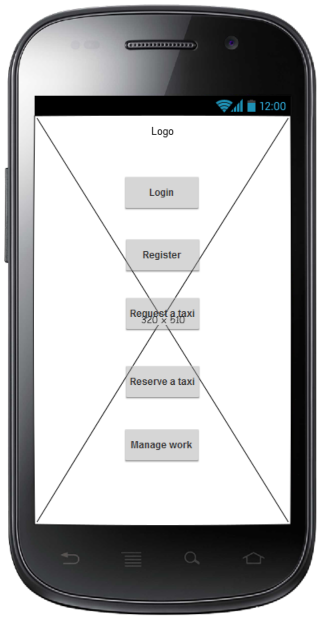


Figure 2: Homepage Mobile Version

3.1.1.2 Registration The mockups below shows the page in which a visitor can sign up.

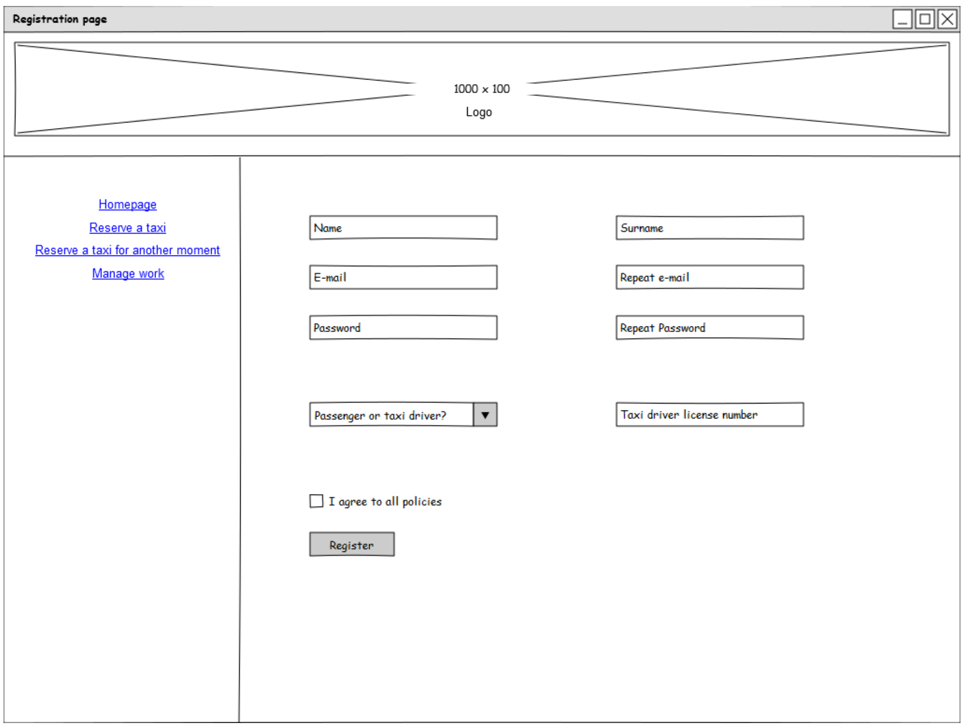


Figure 3: Registration Web Version

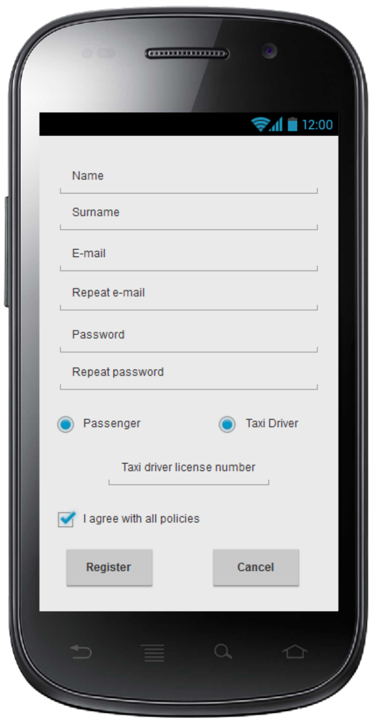


Figure 4: Registration Mobile Version

3.1.1.3 Login

The mockups below shows the page in which a visitor can log in.

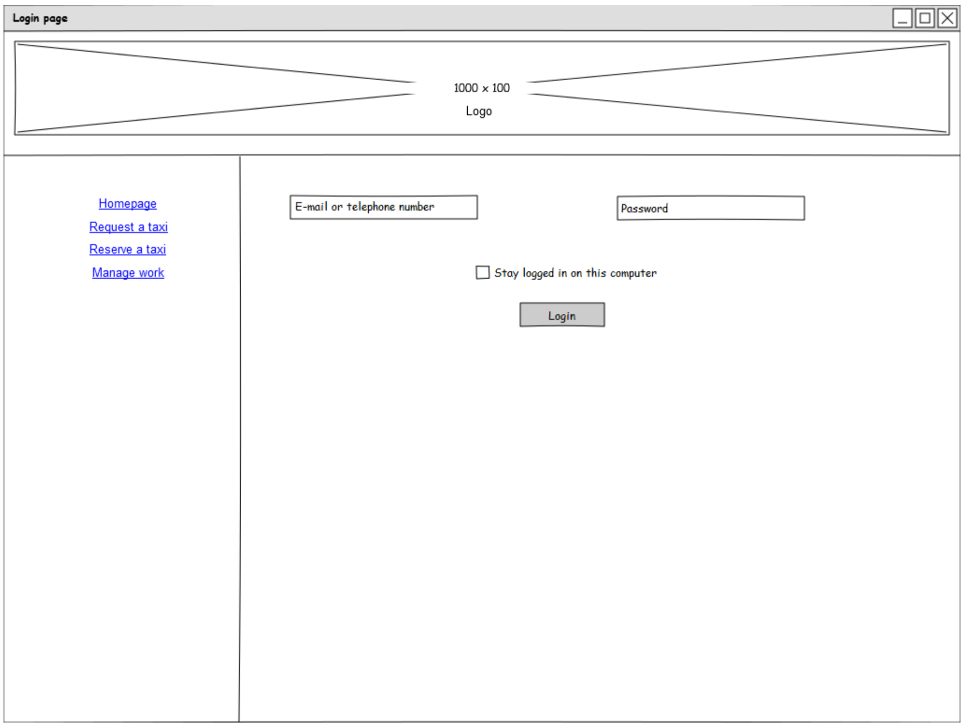


Figure 5: Login Web Version

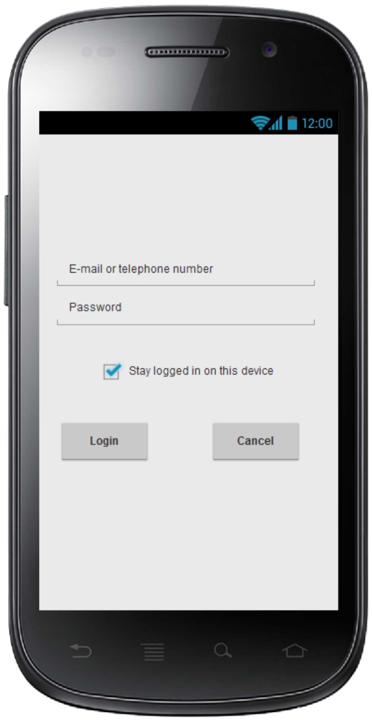


Figure 6: Login Mobile Version

3.1.1.4 Request

The mockups below shows the page in which a Registered Passenger can make a request

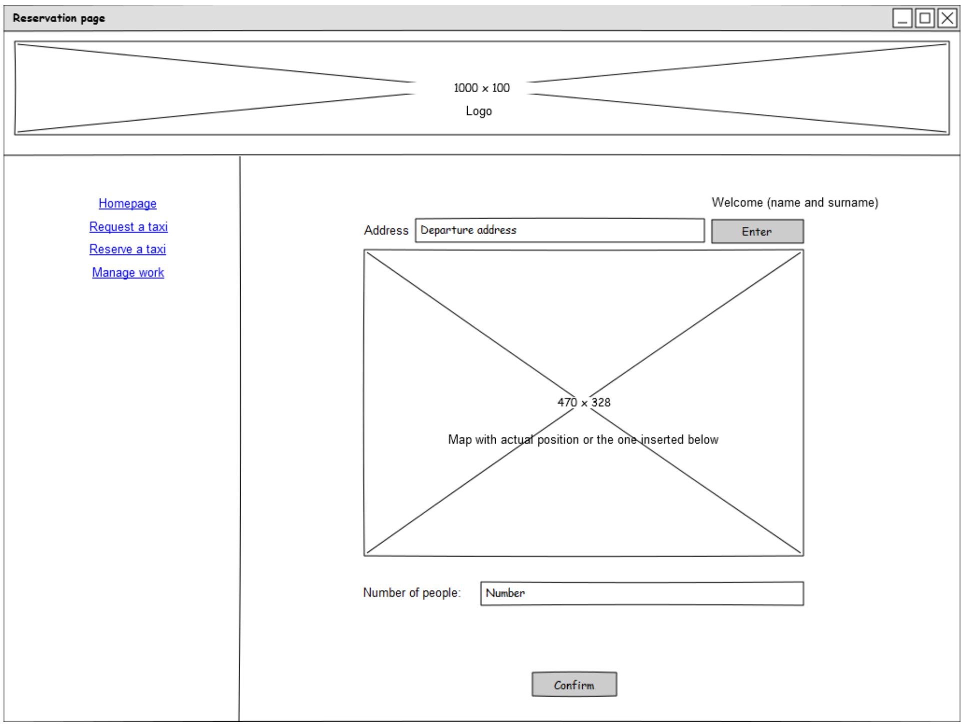


Figure 7: Request Web Version

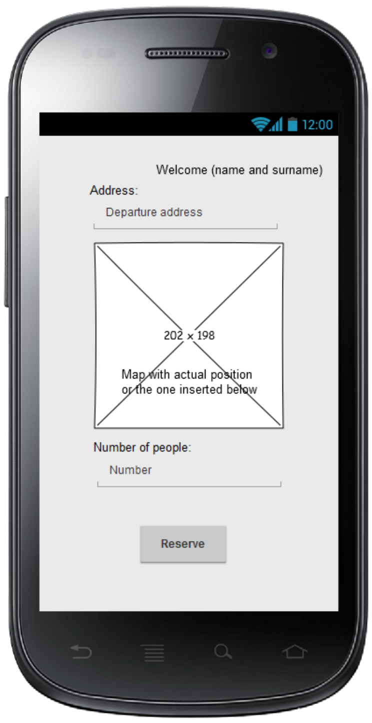

12:00
Welcome (name and surname)
Address:
Departure address
202 x 198
Map with actual position or the one inserted below
Number of people:
Number
Reserve

Figure 8: Request Mobile Version

3.1.1.5 Reservation The mockups below shows the page in which a Registered Passenger can make a reservation

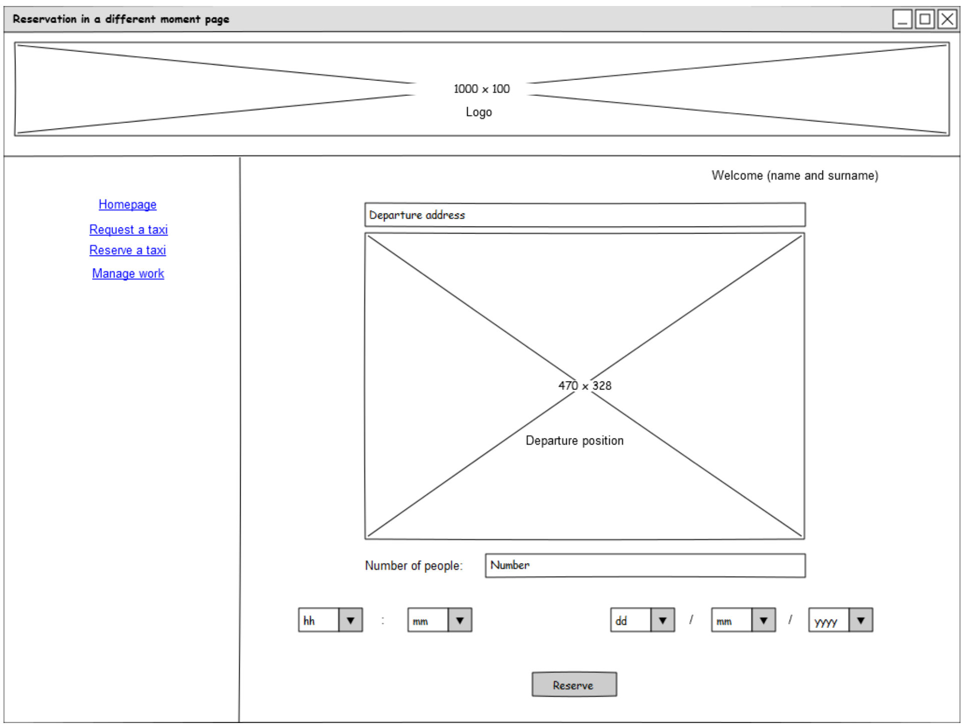


Figure 9: Reservation Web Version

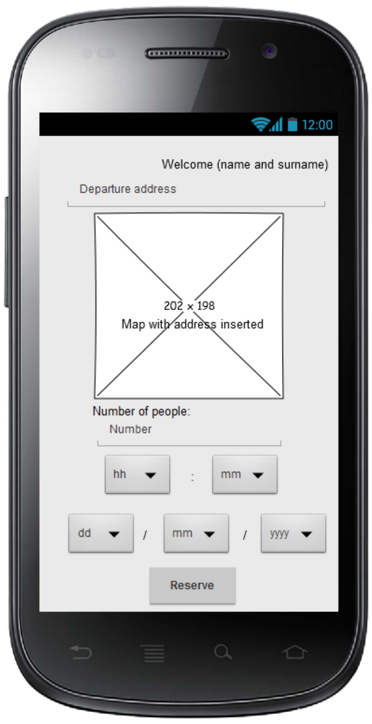


Figure 10: Reservation Mobile Version

3.1.1.6 Driver Interface The mockups below shows the page in which a Driver can manage an incoming request

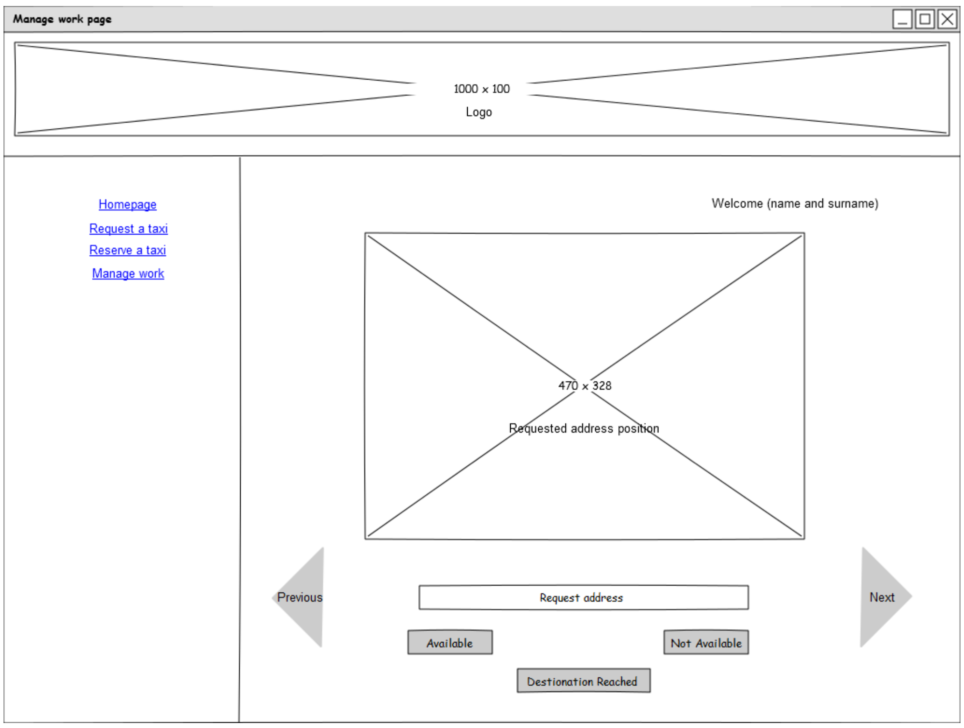


Figure 11: Driver Interface Web Version

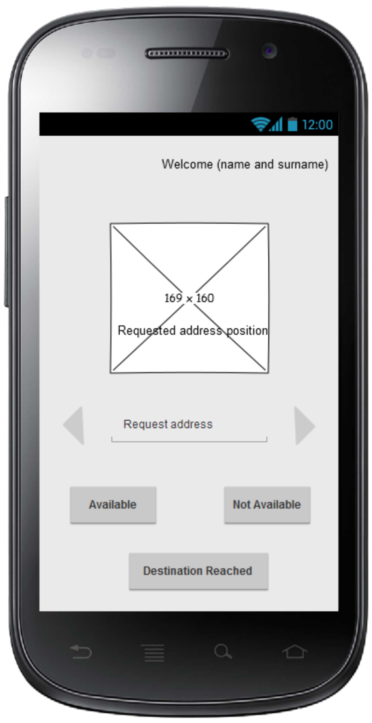


Figure 12: Driver Interface Mobile Version

3.1.2 API Interfaces

The application on the server side will provide some APIs that are used by the mobile application to interact with the system itself. These APIs are essentially RPCs with the data encapsulated using JSON exposed using some endpoints (i.e. specific URLs). The mobile application will connect to the central application by means of these endpoints and will perform necessary operations.

3.1.3 Hardware Interfaces

The system doesn't support any Hardware Interfaces.

3.1.4 Software Interfaces

- DBMS:
  - Name: MariaDB
  - Version: 10.0
  - Source: https://mariadb.org/
- JVM:
  - Name: Oracle JEE
  - Version: 8
  - Source: http://www.oracle.com/technetwork/java/javaee/overview/index.html
- Application Server:
  - Name: Glassfish
  - Version: 4.1.1
  - Source: https://glassfish.java.net/
- OS:
  - Name: Any operating system that can run host a DBMS application and the JVM
  - Version: N/A
  - Source: N/A
- Mobile OSes:
  - Name: Android
  - Version: $\geq$ 4.0
  - Source: https://www.android.com/
- Mobile OSes:
  - Name: iOS
  - Version: $\geq$ 7.0

  - Source: http://www.apple.com/ios/

- Mobile OSes:
  - Name: Windows Phone
  - Version: ≥ 8.0
  - Source: https://www.microsoft.com/en-us/windows/phones

3.1.5 Communication Interfaces

Protocol	Application	Port
TCP	HTTPS	443
TCP	HTTP	80
TCP	DBMS	3306 (default)

3.1.6 Memory

The minimum memory requirements are:

- Primary memory: 2GB+
- Secondary memory: 20GB+

For the mobile application at least 512MB of primary memory is required in order to have a satisfactory user experience.

3.2 Functional Requirements

3.2.1 [G1] Allow visitors to sign up for the service

- [R1] The system requests an e-mail or a telephone number during the registration
- [R2] The system rejects an e-mail or a telephone number already used by another registered passenger
- [R3] After the completion of registration process, the system must send a confirmation to newly registered passengers
- [R4] The system checks if e-mails or telephone numbers are well formed
- [R5] The system considers passengers officially registered if and only if passengers either open an activation link in a confirmation e-mail, or insert a code received through a SMS in mobile application or in web service.

3.2.2 [G2] Allow registered passengers to request a taxi

- [R1] The system must disallow visitors to perform such operation
- [R2] The system has to locate passengers using GPS in his/her smartphone or through browser.
- [R3] After selecting the most appropriate taxi (explained in goal [G4]), the system must inform drivers that a request is incoming.

- [R4] Upon confirmation by the driver about the request of availability, the system has to inform the passenger about a positive outcome,if it occurs, and provide additional information, such as ETA
- [R5] In case of unavailability of a contacted driver, the system has to pick another taxi from the most appropriate queue and to repeat what described in requirement [R3]
- [R6] The system must proceed to the cancellation of the request if and only if the user requested to do it and the condition in [D1] has not met yet.
- [D1] The maximum number of cancellation requests is three.
- [D2] A passenger can request at most 3 taxis thus the amount of people involved in a single request is at most 9 [A11].

3.2.3 [G3] Allow registered passengers to reserve a taxi

- [R1] The system must disallow visitors to perform such operation.
- [R2] The system must check if time and the date specified is at least past two hours with the respect of present time.
- [R3] The system must disallow the reservation of a taxi with an invalid meeting point.
- [R4] The system must disallow the reservation of a taxi with a meeting point outside the area served by the service.
- [R5] The system shall allocate a timer that will timeouts ten minutes before the time and date specified, triggering the event of allocating a taxi for this reservation.

3.2.4 [G4] Guarantee a fair management of the taxi queues

- [R1] The system must select the first free taxi in the queue corresponding to the zone in which the meeting point is.
- [R2] The system has to manage the queue using a predefined policy.
- [R3] If there are no available taxis in a queue, the system scans queues of adjacent zones.
- [R4] The system must remove a taxi from the queue if and only if a passenger has confirmed the intention of taking that taxi.
- [R5] Queues contain only free taxis.
- [R6] The system must not select taxis that belong to drivers who are spending their breaks.
- [D1] The policy for managing the queue is FIFO.

3.2.5 [G5] Allow taxi drivers to manage their workday

- [R1] The system must disallow non-taxi drivers to perform such operations.
- [R2] The system must not allow to start a workday twice or more in the same day.
- [R3] The system must not allow to end a workday before a workday is started.
- [R4] The system must enforce the driver to take some breaks during the day.
- [D1] The workday cannot last more than 8 hours.

3.2.6 [G6] Allow the system administrator to update the information about drivers and taxis

- [R1] The system must disallow non-system administrator to perform such operations.
- [R2] The system must disallow the insertion of inconsistent data (for example drivers with invalid license or without a car).
- [R3] The system must disallow the insertion of already present data.

3.3 Scenarios

3.3.1 Scenario 1

Ilaria is late for work and unfortunately there is no available public transportation to get quickly to the office. She launches myTaxiService mobile application from her smartphone and she logs in. She wants to request a taxi so she taps the button "Request a Taxi". Her smartphone is able to locate her quickly and, though the Internet, the application sends to the system her coordinates and Ilaria's intention to request a taxi. The system looks for taxi queues near Ilaria's position and sends a notification to first driver associated with an available taxi in that queue. The mobile application of Daniele, the taxi driver, receives a notification but unfortunately due to a heavy traffic in that specific street he can't confirm his availability to carry out the request, and he sends a negative reply to the system by tapping the button "Not Available". The system acknowledges this rejection and moves the taxi driver from the top of his queue to the end, and selects the second taxi driver (which now is the first one) from the same queue. This time Marco confirms his availability tapping the button "Available" in the application. The system receives the message and thus allocates selected taxi to Ilaria's request, while a confirmation message is sent to her providing additional information such as the taxi code and its ETA. Now she waits for the taxi to come to her position. Meanwhile Marco has received Ilaria's position from the system and it is displayed in his smartphone, for this reason he moves from that location. When Marco arrives he picks up Ilaria, she provides the destination to Marco and the ride starts. When the destination is reached, Marco taps the button "Destination reached" and sends the corresponding message to the system. Ilaria pays the amount due and she leaves the taxi and reaches the office. When the system receives a "Destination reached" input from a taxi driver, the system calculates his/her new position, eventually the system adds the taxi to the appropriate queue according to the location received.

3.3.2 Scenario 2

Davide has just arrived to the city, a friend suggested him to install myTaxiService application, so he does it and tries to register. He provides to the application his e-mail, his name, his surname and his date of birth, he accepts the privacy and cookie policy and taps the button “Register”. After he sent his registration request, the system discovers that the e-mail was already used and so Davide receives a notification on the application about the problem and inserts another e-mail. This time the e-mail he inserted is accepted and the system stores his personal details in the database, while the system sends a confirmation e-mail to previously provided e-mail, containing a temporary password. To conclude the registration process Davide must log in at least one time, using the e-mail provided during the registration and the password generated by the system. As soon as he logs in, the system considers Davide a valid registered passenger and he can now use other system’s functionalities, and also change his password with another one.

3.3.3 Scenario 3

Bob has booked a flight so he decided to reserve a taxi driver that will take him to the airport. From his home, he browses myTaxiService website, he logs in and presses the button “Reserve a Taxi”, a new page loads up with a map and a textual box. The website asks him the permission to determine his position by the browser, but he declines because he wants to set a starting position that is different than the current one. He inserts the address in the textual box and presses “Enter” on the keyboard. The map will update moving the region centering the specific position inserted previously, placing also a marker to emphasize the position detected by the system. Then Bob inserts time and date he needs to reserve and presses “Confirm”. The system stores this data in the database since all the information provided are adequate. Bob receives an e-mail about either a successful or a negative outcome of the operation. When the date approaches, specifically 10 minutes before the time and date specified, the system processes the reservation, managing it in the same way as requests already described in Scenario 1.

3.3.4 Scenario 4

Jack is a taxi driver, after he completed the registration to myTaxiService application, he can start to use it. According to taxi driver job laws, the system enables the application to receive requests during working time as described previously. The system has break times registered in its records for each taxi driver. When Jack’s break time occurs, he can only terminate his last work, after that he can’t accept other works until the break ends. At the end of his working time, the application is disabled by the system until the next working day. Jack can insert unavailabilities if unexpected events occur, letting him not to work for the day or hours necessary to solve his problem.

3.3.5 Scenario 5

Frank has just requested a taxi to take him home, without thinking he tries to reserve another taxi, but the system recognises that he has already made a reservation, so it sends a message to Frank saying his last request has been rejected because of the impossibility to make a double reservation and the second request is ignored.

3.3.6 Scenario 6

Tom wants to eliminate his myTaxiService account, so he enters his account's settings and presses the button “Eliminate account”, after he confirmed his desire not to use more the application, the system removes Tom's personal details and his previous reservations from databases.

3.3.7 Scenario 7

Alice is a taxi driver and she has just been hired by the company. She has already provided her information about her driving license ID and her taxi car ID. Bill, the system administrator logs in the system and inserts data supplied by Alice in the DB. Upon the click of the “Confirm” button, the system connects itself to the external database and checks if data are present and valid. As expected all information are good and the system writes data on its DB and displays a confirmation message to Bill, now Alice is able to login in the system and start to work.

3.4 UML models

3.4.1 Use cases

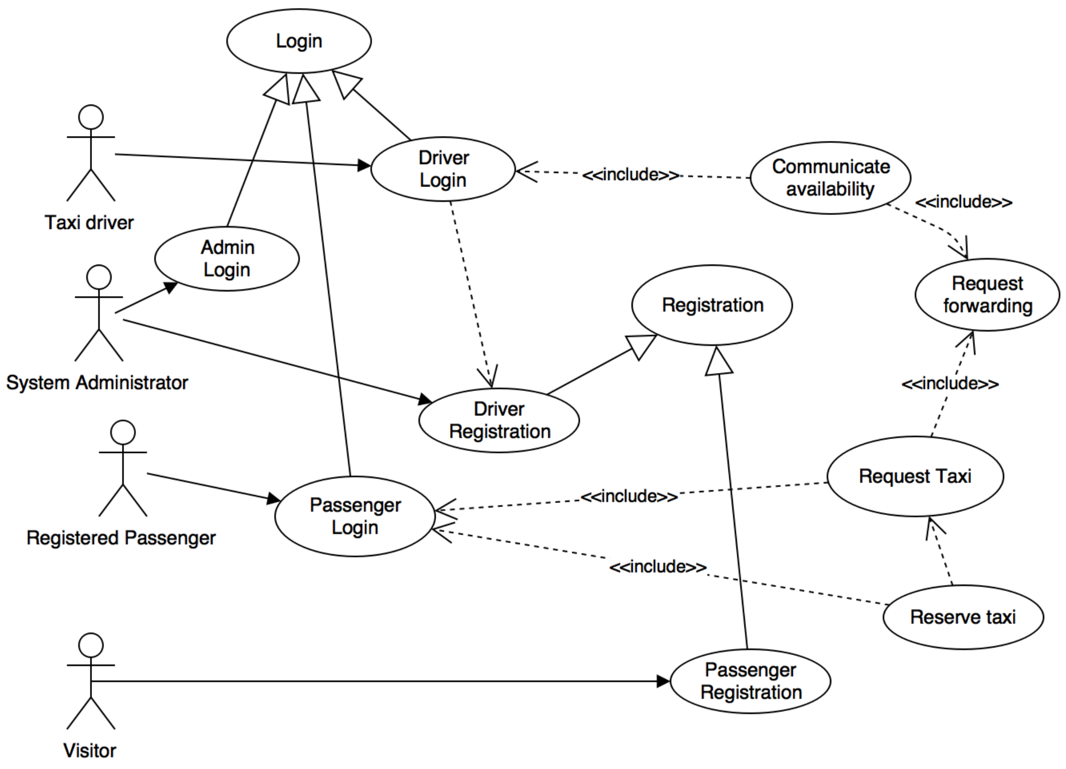


Figure 13: Diagram for the myTaxiService application

3.4.1.1 Visitors sign up for myTaxiService

Actor	Visitor
Goal	[G1] Allow visitors to sign up for the service
Input Condition	None
Event Flow	1. Visitor clicks on the registration button. 2. Visitor fills the registration form in with mandatory fields. 3. Visitor User clicks on the “Confirm” button to become a registered passenger. 4. Visitor User activates the account either • by opening the link received in the confirmation e-mail • or by inserting the confirmation code received by SMS. 5. The system will save information on the DB.
Output Condition	Visitor completes the registration and becomes a Registered Passenger. Now the Registered Passenger can log in and start using myTaxiService.
Exception	1. Visitor is already a Registered Passenger. 2. One of the mandatory fields contains invalid data 3. Email or phone number are already used by another Registered Passenger.

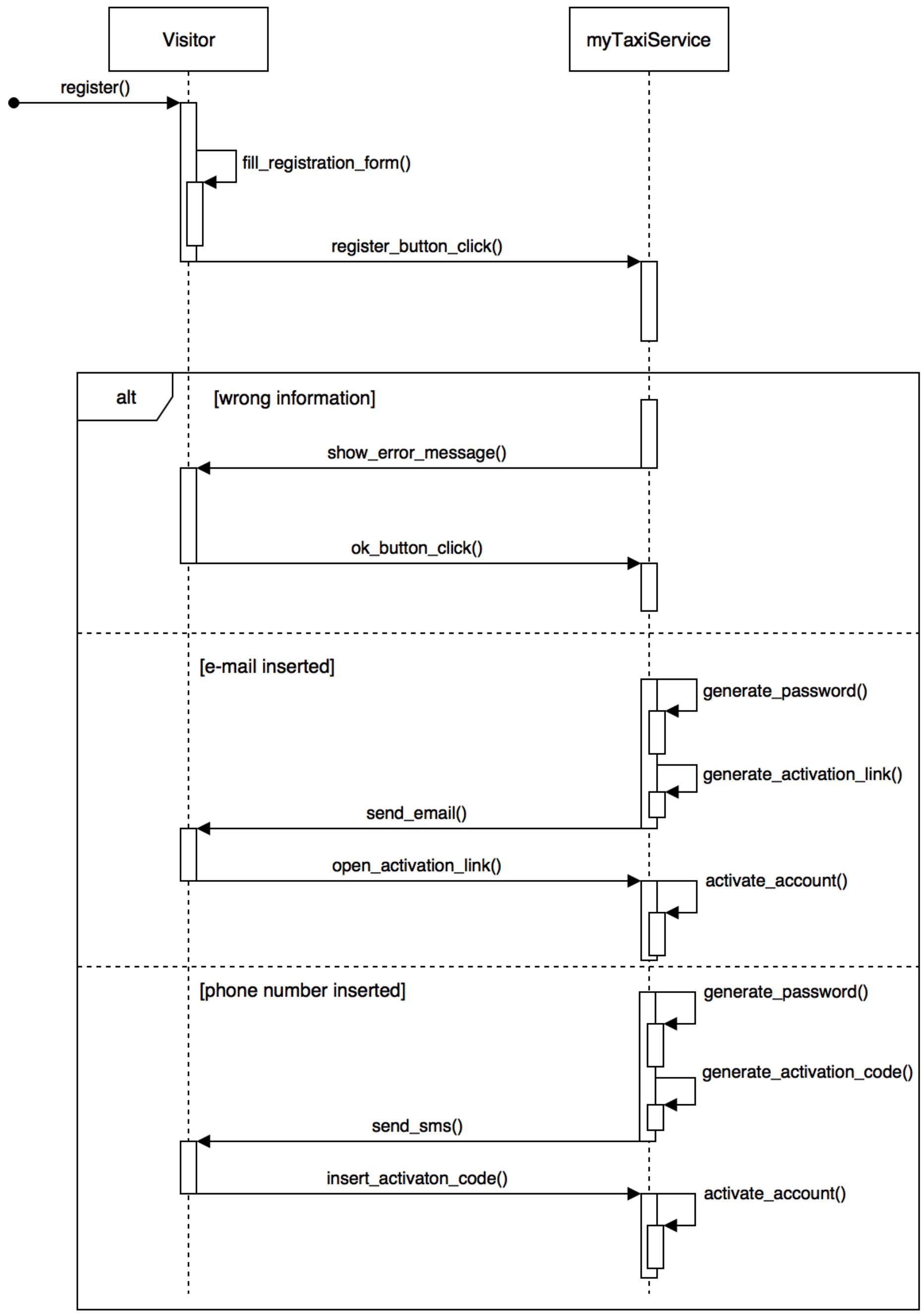


Figure 14: Sequence diagram for the use case

3.4.1.2 Visitor signs in

Actor	Visitor, Registered Passenger
Goal	A Visitor wants to sign in
Input Condition	Visitor is already a Registered Passenger
Event Flow	1. myTaxiService shows the login page 2. Visitor inserts his/her credentials chosen during registration and presses the "Login" button. 3. myTaxiService validates the data.
Output Condition	Visitor is promoted to Registered Passenger, now he/she can use all myTaxiService functions.
Exception	Incorrect e-mail, phone number or password inserted. myTaxiService will notify the visitor about this.

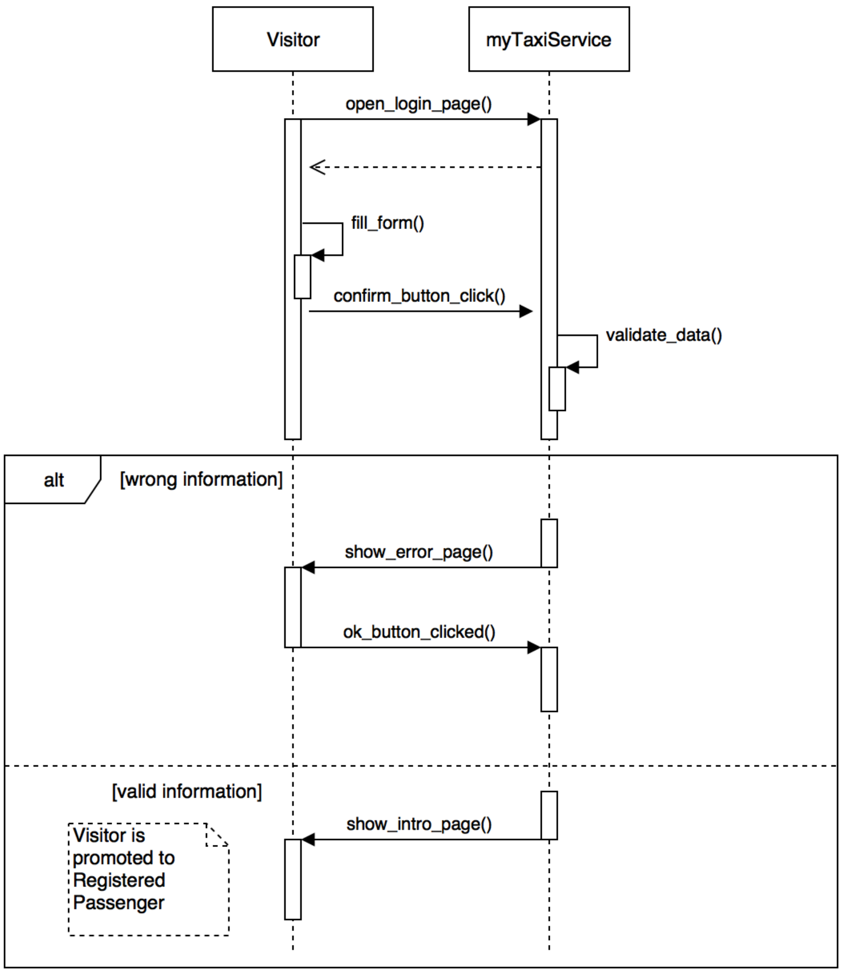


Figure 15: Sequence diagram for the use case

3.4.1.3 Registered Passenger makes a request

Actor	Registered Passenger
Goal	[G2] Allow registered passengers to request a taxi
Input Condition	Registered Passenger is authenticated into the myTaxiService.
Event Flow	1. Passenger chooses to request a taxi. 2. Passenger's location is determined by the GPS. 3. Passenger specifies the amount of people involved in the request. 4. Passenger presses the button "Confirm"
Output Condition	The Registered Passenger has successfully sent the request.
Exception	1. The application fails to locate the passenger. 2. The location detected is not served by the service. 3. The amount of people exceeds the maximum allowed. In both cases the system notifies the Passenger with a message on the screen.

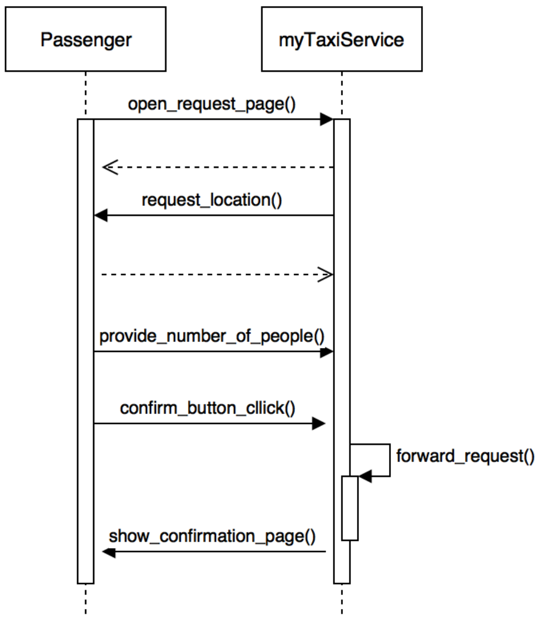


Figure 16: Sequence diagram for the use case

3.4.1.4 Passenger makes a reservation

Actor	Registered Passenger
Goal	[G3] Allow registered passengers to reserve a taxi
Input Condition	Registered Passenger is authenticated into the myTaxiService.
Event Flow	1. Passenger clicks on the button "Reserve a taxi". 2. Passenger specifies all mandatory fields. 3. Passenger presses the button "Confirm".
Output Condition	The Registered Passenger has successfully submitted the reservation. Information are stored in a DB, the system schedules a timer that will time out 10 minutes before the time and date specified by the Passenger.
Exception	1. The meeting point is not found on the map 2. The meeting point is not served by the system. 3. The time and date specified is too close to the present time.

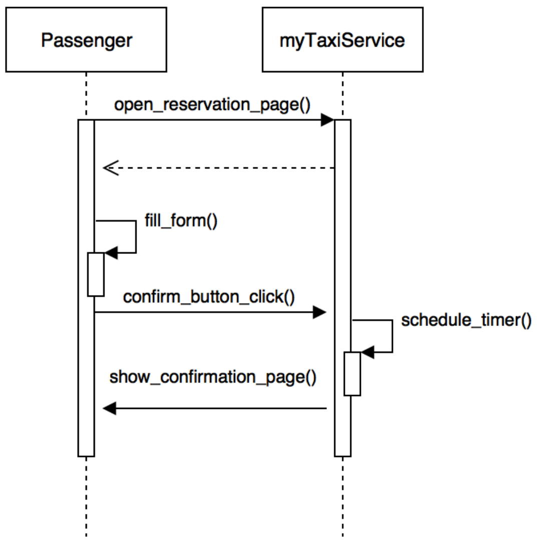


Figure 17: Sequence diagram for the use case

3.4.1.5 Request forwarded to a taxi

Actor	Taxi driver, Registered Passenger
Goal	[G2] Allow registered passengers to request a taxi, [G3] Allow registered passengers to reserve a taxi
Input Condition	The passenger has just sent a request, or the time remaining to a scheduled reservation is equal to ten minutes
Event Flow	1. The system looks for the queue in the zone of the meeting point and selects the first available taxi. 2. The system sends a notification to the driver of the selected taxi, the driver has 30 seconds to answer: • The taxi driver presses the button "Confirm", a notification to the Passenger is sent indicating the ID of the taxi and the ETA to the meeting point, now the Passenger can: – Accepts the request by pressing the button "Confirm", the taxi is removed from the queue and the system informs the user about the succeed of the operation. – Cancel the request by pressing the button "Decline", at this point the Event flow jumps to 1. • The taxi driver presses the button "Decline", the matching taxi is moved to the bottom of the queue, the Event flow jumps to 1.
Output Condition	The request is fulfilled by the selected driver and he/she begins to move to the meeting location.
Exception	1. There are neither available taxis in the zone of the meeting point nor in the adjacent zones. 2. The Passenger cancels the request for three times (see [D2] of [G2]).

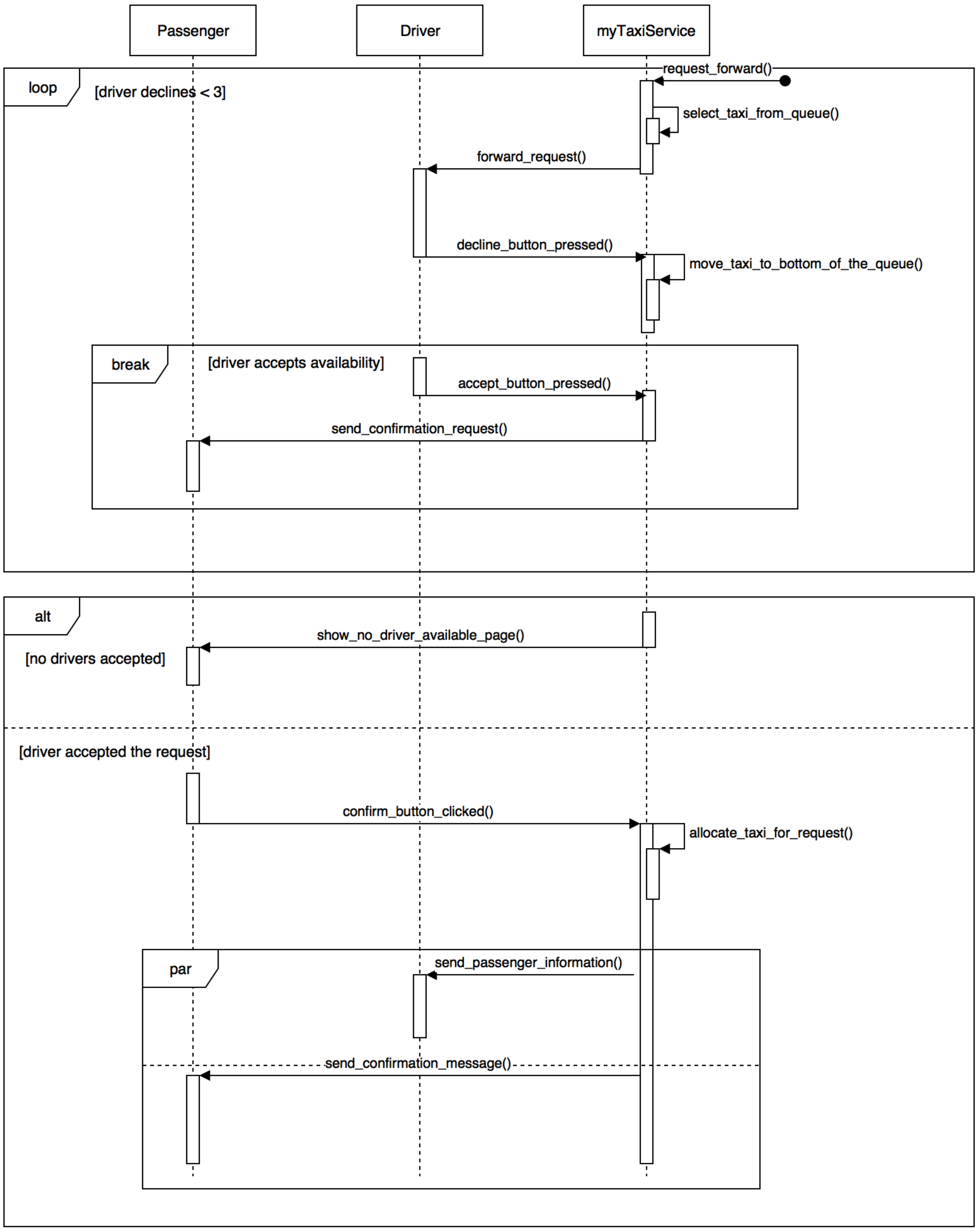

Passenger
Driver
myTaxiService
loop
[driver declines < 3]
request_forward()
select_taxi_from_queue()
forward_request()
decline_button_pressed()
move_taxi_to_bottom_of_the_queue()
break
[driver accepts availability]
accept_button_pressed()
send_confirmation_request()
alt
[no drivers accepted]
show_no_driver_available_page()
[driver accepted the request]
confirm_button_clicked()
allocate_taxi_for_request()
par
send_passenger_information()
send_confirmation_message()

Figure 18: Sequence diagram for the use case

3.4.1.6 Driver communicates availability

Actor	Taxi driver
Goal	[G5] Allow taxi drivers to manage their workday
Input Condition	The driver is logged in the system
Event Flow	1. Taxi driver presses the button “Available”. 2. The system determines the position of the driver and inserts his/her taxi to the appropriate queue. 3. The system sends confirmation to the waiting driver.
Output Condition	The driver is now available and his/her taxi is in a queue.
Exception	1. The driver is already available. 2. The position of the driver is invalid. 3. The position of the driver is outside the area served by the service.

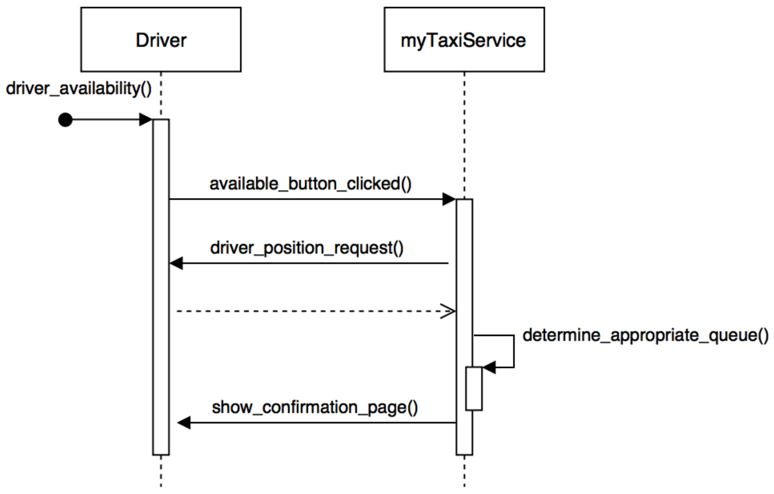

Driver
myTaxiService
driver_availability()
available_button_clicked()
driver_position_request()
determine_appropriate_queue()
show_confirmation_page()

Figure 19: Sequence diagram for the use case

3.4.1.7 Driver login

Actor	Taxi driver
Goal	[G5] Allow taxi drivers to manage their workday
Input Condition	The driver is registered in the system
Event Flow	1. The driver launches the application on the smartphone 2. The driver insert his/her credentials.
Output Condition	The driver is now capable of accepting incoming requests and has access to the dedicated features
Exception	1. Invalid credentials

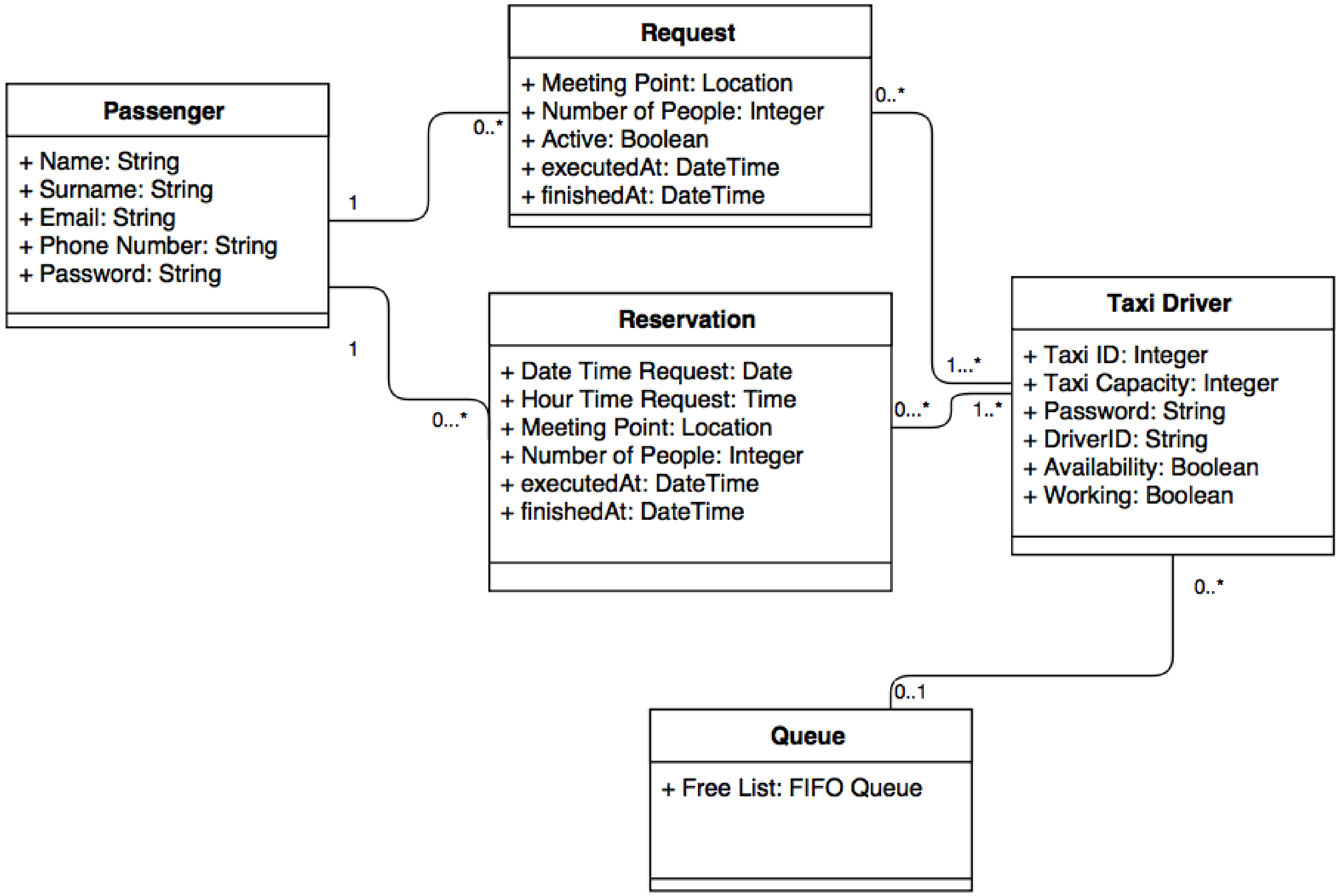
3.4.1.8 Admin login

Actor	System Administrator
Goal	[G5] Allow taxi drivers to manage their workday
Input Condition	The system administrator is registered in the system
Event Flow	1. The system administrator opens the administration page 2. The system administrator inserts his/her credentials.
Output Condition	The system administrator is now capable of managing the taxi driver and their car information.
Exception	1. Invalid credentials

3.4.1.9 Driver registration

Actor	Taxi Driver, System Administrator
Goal	[G5] Allow taxi drivers to manage their workday
Input Condition	The system administrator is registered in the system, the taxi driver has been hired by the company.
Event Flow	1. The system administrator logs in the system. 2. The system administrator presses button “Insert new Driver” 3. The system administrator inserts the taxi driver license ID and car ID. 4. The system checks the validity of the data, it can happen: 5. The system connects to the external database and checks the consistency of the data 6. The system stores the data in the DB.
Output Condition	The taxi driver is now able to log in the system and thus he/she can start to use it.
Exception	1. Invalid credentials of the system administrator 2. Invalid data provided by the taxi driver 3. Connection error to the external database

3.4.2 Class Diagram



3.4.3 State chart diagram

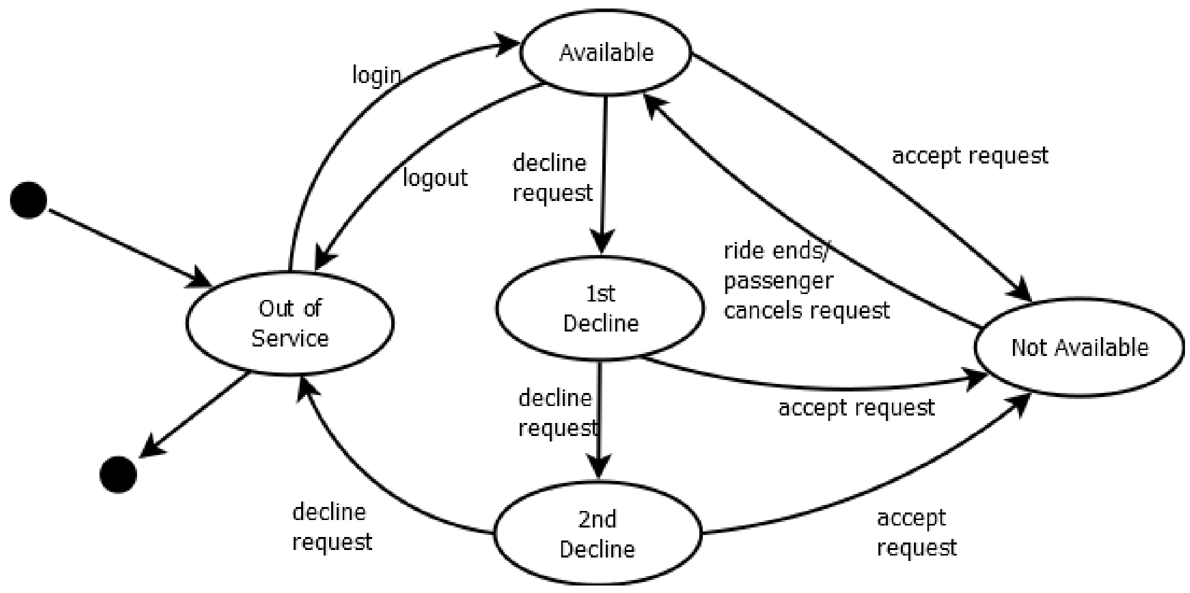


Figure 20: State chart diagram for driver

3.5 Non Functional Requirements

3.5.1 Performance Requirements

The overall performance of the system must be acceptable so that a good level of usability is achieved. The response time with the respect to the system must be near to zero. The response time of mobile application is tricky to evaluate since there are lots of different devices with different power processing, but in general the application should perform well on the majority of the devices available in today market. Some very low level or entry level devices might show some delays rendering the application on the screen. Nevertheless the major bound is customers and drivers internet connection.

3.5.2 Design Constraints

3.5.3 Software System Attributes

3.5.3.1 Availability The overall system must be available anytime according to [A10]. Dedicated servers must be used to reach this goal. In order to achieve even more reliability, some fault tolerant mechanisms must be implemented, also to reduce risks of high loads that could eventually result in denials of service. Servers must be replicated to work in parallel by multiple instances of the same application.

3.5.3.2 Maintainability The application provides specific APIs that will be documented, as the whole application.

3.5.3.3 Portability The mobile application will be developed on all the major mobile OSes available in the market, also web application is portable since nowadays the majority of widespread OSes has a browser. The web application will be designed with a responsive layout to be also accessible from mobile devices as well. The business logic in the servers will be implemented using the Java language.

3.5.4 Security

3.5.4.1 External Interface Side Every user's profile is protected by an authentication system, in fact users have to insert registration e-mail or their mobile phone number and profile password. To protect this classified information the system uses a hash cryptographic function. Every 6 month the system requires users to change their password, demanding to respect password's constraints, such as password length, minimum presence of numbers and characters, special and not.

3.5.4.2 Application Side Data traded between client and server are protected by HTTPS protocol. To enhance security, TLS protocol is used to protect socket connections. All this methods prevent attacks like man in the middle or phishing.

3.5.4.3 Server side Firewalls and DMZ zones protect server layers by external attacks, to make these last ones more difficult, business logic and data are separated. Physical servers are secured by UPSes (uninterruptible power supply), to prevent problems causes by electrical interruptions. Furthermore servers are placed in locations without damp and not exposed to the sun.

4 Alloy

4.1 Model source file

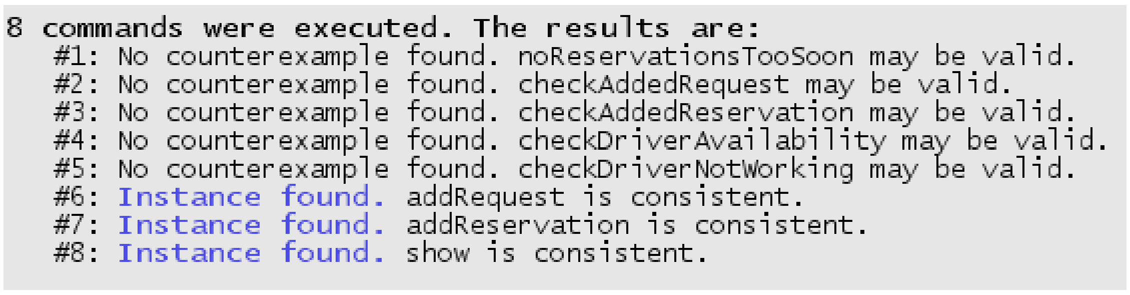
```
open util/boolean
open util/integer

// *** SIGNATURES ***
sig Str{}
sig SInt{v: one Int}
{
v≥0
}
sig Integer{}

sig Location{
        latitude: one Integer,
        longitude: one Integer
}

sig Passenger {
        email: one Str,
        phone: one Str,
        requests: some Request,
        reservations: some Reservation
}

sig Request{
        ID: one Integer,
        meetingPoint: one Location,
        people: one Int,
        passenger: one Passenger,
        active: one Bool,
        driver: some TaxiDriver,
        executedAt: one SInt,
        finishedAt: one SInt
}
{
        people>0
        #driver = div[people,3]
}


sig Reservation {
        ID: one Integer,
        meetingPoint: one Location,
        people: one Int,
        passenger: one Passenger,
        active: one Bool,
        driver: some TaxiDriver,
        executedAt: one SInt,
        finishedAt: lone SInt,
        requestTime: one SInt,
}
{
        requestTime.v≤minus[executedAt.v,2]
        people>0
        #driver = div[people,3]
}

sig TaxiDriver {
        taxiID: one Integer,
        taxiCapacity: one Integer,
        driverID: one Str,
        availability: one Bool,
        working: one Bool,
        requests: set Request,
        reservations: set Reservation
}
{
        (availability=True implies working=True) and (working=False implies availability
            ↪ =False)
```

```
}


sig Queue {
        ID: one Integer,
        drivers: set TaxiDriver
}

sig System {
        reservations: set Reservation,
        requests: set Request,
        queues: set Queue,
        drivers: set TaxiDriver
}

// *** FACTS ***

// only one System containing all the data
fact oneSystem {
        #System=1
        all r: Request | r in System.requests
        all q: Queue | q in System.queues
        all t: TaxiDriver | t in System.drivers
}

fact noDuplicateLocations {
        no disj l1, l2: Location | (l1.latitude=l2.latitude and l1.longitude=l2.
            ↪ longitude)
}

fact uniqueEmailAndPhones {
        no disj p1, p2: Passenger | (p1.email=p2.email)
        no disj p1, p2: Passenger | (p1.phone=p2.phone)
}

fact uniqueIDsRequest {
        no disj r1, r2: Request | (r1.ID=r2.ID)
}

fact uniqueIDsReservation {
        no disj r1, r2: Reservation | (r1.ID=r2.ID)
}


fact uniqueQueues {
        no disj q1, q2: Queue | (q1.ID=q2.ID)
}

fact uniqueTaxiDriver {
        no disj t1, t2: TaxiDriver | (t1.taxiID=t2.taxiID) and (t1.driverID=t2.driverID)
}


fact driverOnlyInOneQueue {
        all disj q1, q2: Queue | (q1.drivers & q2.drivers) = none
}

fact noRequestsFinishedBeforeBeingExecuted {
        all r: Request | (r.finishedAt.v > r.executedAt.v)
        all r: Reservation | (r.finishedAt.v > r.executedAt.v)
}

fact driverAvailableInOneQueue{
        all q: Queue | (all t: TaxiDriver | t in q.drivers iff t.availability=True)
}

fact noSameDriversTwoOverlappingRequests {
        all disj r1, r2: Request | (r1.driver & r2.driver)≠none
                        iff r2.finishedAt.v =< r1.executedAt.v or r2.executedAt.v ≥ r1.
                            ↪ finishedAt.v
        all disj r1, r2: Reservation | (r1.driver & r2.driver)≠none
                        iff r2.finishedAt.v =< r1.executedAt.v or r2.executedAt.v ≥ r1.
                            ↪ finishedAt.v
        all r1: Request, r2: Reservation | (r1.driver & r2.driver)≠none
                        iff r2.finishedAt.v =< r1.executedAt.v or r2.executedAt.v ≥ r1.
                            ↪ finishedAt.v
}

fact driverRequestAndDriverReservationRelation {
        all d: TaxiDriver | (all r: Request | r in d.requests implies d in r.driver)
        all d: TaxiDriver | (all r: Reservation | r in d.reservations implies d in r.
            ↪ driver)
}

fact driversUnavailableDuringActiveRequestsOrReservations {
        all r: Request | (all d: TaxiDriver | r.active=True and d in r.driver implies (d
            ↪ .availability=False and d.working=True))
        all r: Reservation | (all d: TaxiDriver | r.active=True and d in r.driver
            ↪ implies (d.availability=False and d.working=True))
}

fact availableDriverMeansAlreadyFinishedAllHisRides {
        all d: TaxiDriver | (all r: Request | d.availability=True and r in d.requests
            ↪ implies r.active=False)
        all d: TaxiDriver | (all r: Reservation | d.availability=True and r in d.
            ↪ reservations implies r.active=False)
}

fact noSameDriverInDifferentActiveRequests {
        some disj r1, r2: Request | (r1.active=True and r2.active=True) implies ((r1.
            ↪ driver & r2.driver)=none)
        some disj r1, r2: Reservation | (r1.active=True and r2.active=True) implies ((r1
            ↪ .driver & r2.driver)=none)
        some r1: Request, r2: Reservation | (r1.active=True and r2.active=True) implies
            ↪ ((r1.driver & r2.driver)=none)

}

// *** ASSERTIONS ***

assert noReservationsTooSoon {
        no r: Reservation | (r.requestTime.v > minus[r.executedAt.v,2])
}

check noReservationsTooSoon

assert checkAddedRequest {
        all s, s': System, r: Request | addRequest[s, s', r] implies (r in s'.requests)
}

check checkAddedRequest

assert checkAddedReservation {
        all s, s': System, r: Reservation | addReservation[s, s', r, 1] implies (r in s
            ↪ '.reservations)
}

check checkAddedReservation

assert checkDriverAvailability {
        no r: Request | all d: TaxiDriver | (r.active=True and d.availability=True)
        no r: Reservation | all d: TaxiDriver | (r.active=True and d.availability=True)
}

check checkDriverAvailability

assert checkDriverNotWorking {
        no r: Request | (some d: TaxiDriver | d in r.driver and r.active=True and d.
            ↪ working=False)
        no r: Reservation | (some d: TaxiDriver | d in r.driver and r.active=True and d.
            ↪ working=False)
}

check checkDriverNotWorking

// *** PREDICATES ***
```

```
pred addRequest[s, s': System, r:Request] {
        s'.requests = s.requests + r
}

pred addReservation[s, s': System, r:Reservation, time: Int] {
        r.requestTime.v = time and
        s'.reservations = s.reservations + r
}

pred show {
        #Reservation>1
        #Request>1
        #{x: Reservation | x.active=False}>1
        #{x: Reservation | x.active=True}>1
        #{x: Request | x.active=True}>1
        #{x: Request | x.active=False}>1
}

// *** RUN COMMANDS ***

run addRequest
run addReservation
run show for 4
```

4.1.1 Result

Here are the the results from Alloy Analyzer that proves the consistency of the model.



```
8 commands were executed. The results are:
   #1: No counterexample found. noReservationsTooSoon may be valid.
   #2: No counterexample found. checkAddedRequest may be valid.
   #3: No counterexample found. checkAddedReservation may be valid.
   #4: No counterexample found. checkDriverAvailability may be valid.
   #5: No counterexample found. checkDriverNotWorking may be valid.
   #6: Instance found. addRequest is consistent.
   #7: Instance found. addReservation is consistent.
   #8: Instance found. show is consistent.
```

Figure 21: Results

4.2 Generated world

The following figure shows the generated world for the predicate show

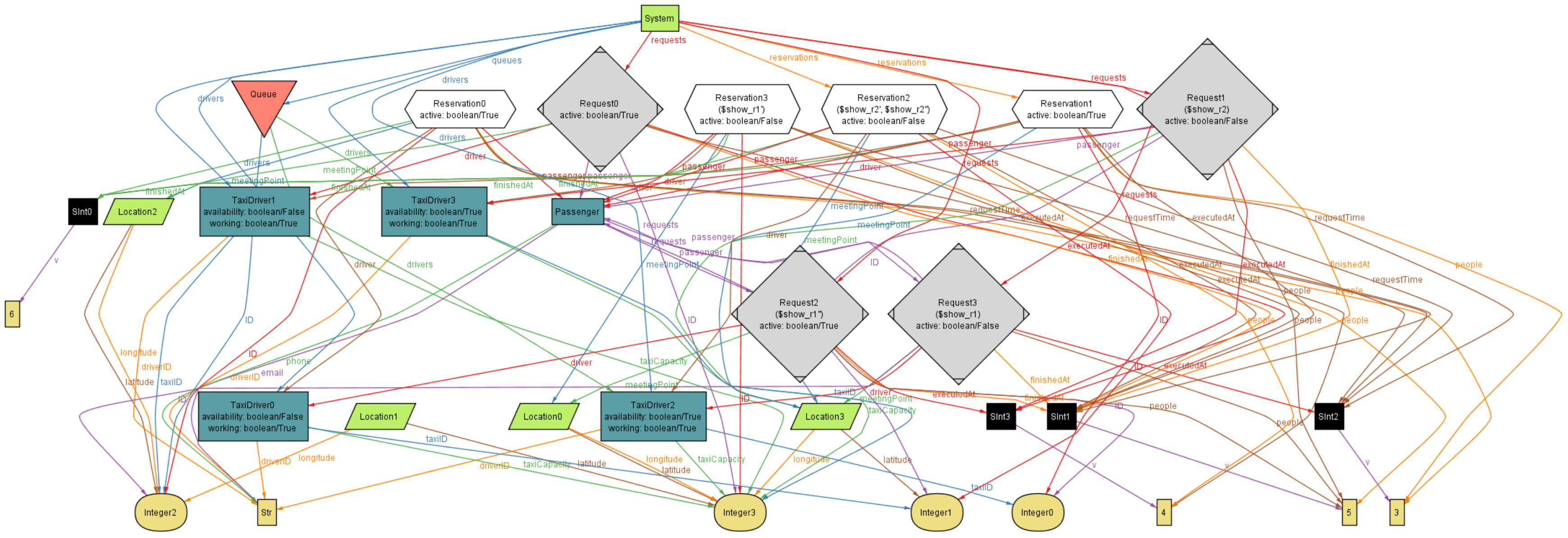


Figure 22: Results

Tools Used

- Alloy 4.2 - `http://alloy.mit.edu/alloy/`
- Basic MiKTeX 2.9.5721 64-bit - `http://miktex.org/`
- Texmaker 4.5 - `http://www.xm1math.net/texmaker/index.html`
- draw.io - `https://www.draw.io/`
- Pencil 2.0.5 - `http://pencil.evolus.vn`

Hours spent to write this document

- Raffaele Fioratto: 35 hours
- Nicolò Longoni: 33 hours