POLITECNICO DI MILANO

# Requirements Analysis and Specification Document (RASD)

Computer Science and Engineering (CSE)

Software Engineering 2 Project

Year 2015/16 – Version 2.0 (04/12/15)

**STUDENTS:**

*Martino Andrea (**788701**)*

*Marchesani Francesco (**852444**)*

**PROFESSOR:**

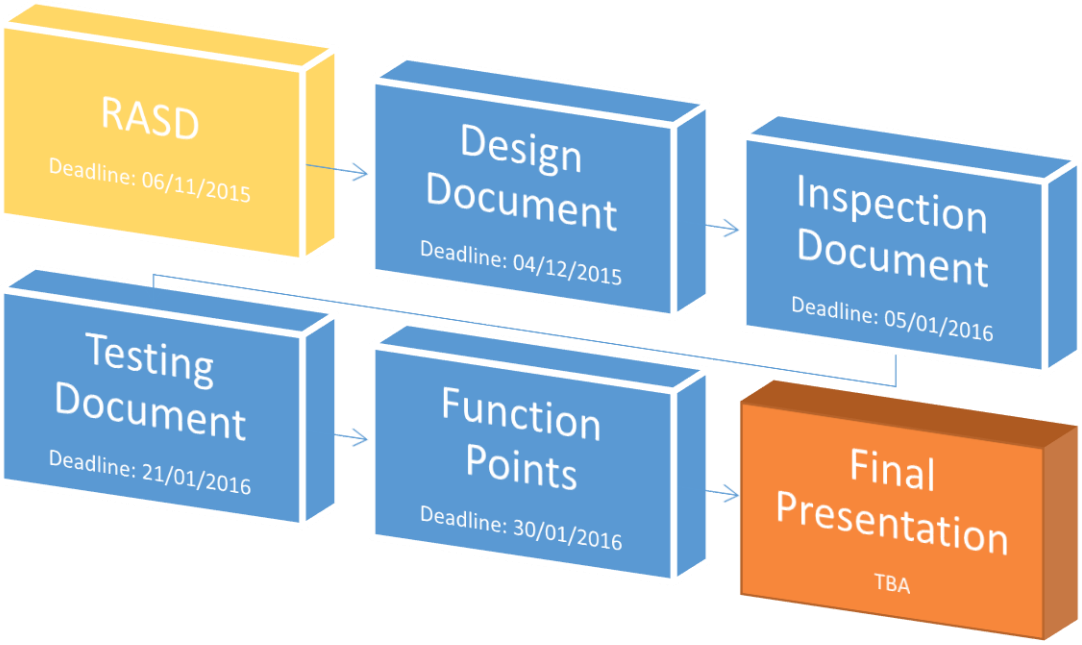
*Mirandola Raffaela*

1. Introduction ..... 3
1.1. Purpose of the requirements model ..... 3
1.2. RASD Approach: “The world and the machine” ..... 4
1.3. myTaxiService: main goals ..... 5
1.4. Current state of the service and future prospect ..... 6
1.5. Definitions, acronyms and abbreviations ..... 7
1.6. References ..... 8
2. Overall description ..... 9
2.1. Product perspective ..... 9
2.2. User characteristics ..... 9
2.3. Constraints ..... 9
2.4. Assumptions and Dependencies ..... 10
2.5. Future possible implementations ..... 11
2.6. Stakeholders identification ..... 11
3. Specific requirements ..... 12
3.1. External interface requirements ..... 12
3.2. Functional Requirements ..... 19
3.3 Scenarios ..... 20
3.4 Non-functional requirements ..... 22
4. Use Case and UML Models ..... 23
4.1. Use Case and related UML Models ..... 23
4.2. Class Diagram ..... 49
5. Appendix ..... 50
5.1. Alloy ..... 50
5.2. Software and Tools Used ..... 62
5.3. Hours of work ..... 62
5.4. Revision of the document ..... 62

# 1. Introduction

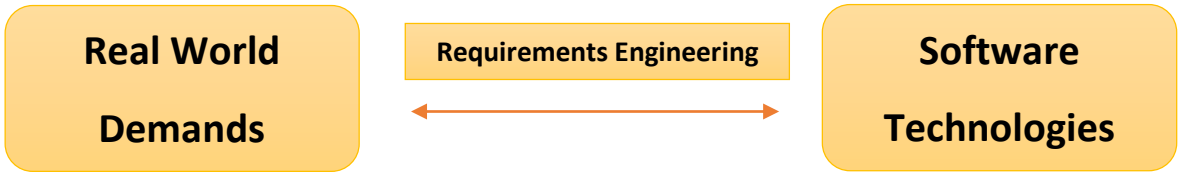
## 1.1. Purpose of the requirements model

The main purpose of this **RASD** (*Requirements Analysis and Specification Document*) is to examine in depth the phases of analysis and specification of the project requirements.
The project name is *myTaxiDriver*, which is the **Software Engineering 2 project** of year 2015/16 at **Politecnico di Milano**.
The reference model used in this project is **IEEE/ANSI 830-1998**, with little adjustments. This is one of the most widely known requirements document standard. It is important to underline that the specifications of this document may evolve in the future (this may occurs for several causes).
Anyway, we will try to maintain coherence with this document in the next steps as much as possible.



## 1.2. RASD Approach: "The world and the machine"

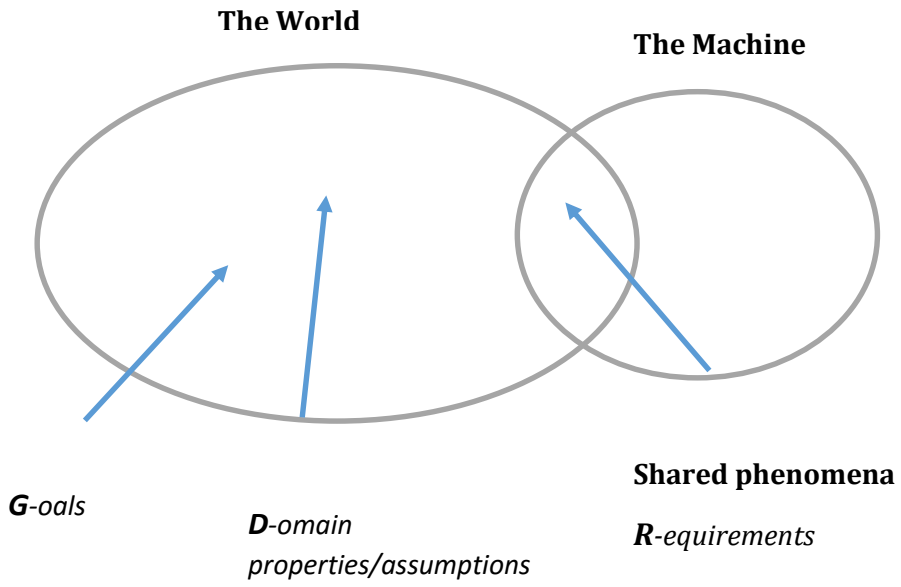
Identify the right **requirements** may be a difficult thing to do if the approach is not good enough. The main thing to understand is the link between what happens in the real world (*The World*) and the software technologies (*The Machine*). This link is Requirements Engineering.




The approach followed in this document is known as "*The world and the machine*". This one is the approach defined by Michael Jackson and Pamela Dave. There are two main entities in this approach:

- **The World**: part of the real World that interfaces with the software to be and which is influenced by him.

- **The Machine:** part of the software to be. That is the union of the developed software and the hardware where software will be executed.



## 1.3. myTaxiService: main goals

According to "*The World and The Machine*" model, *myTaxiService* project has the following goals:

- **[G0]** Allow a Visitor to sign-up to *myTaxiService*.
- **[G1]** Allow a Visitor to log-in to *myTaxiService*.
- **[G2]** Allow a Costumer to make a taxi reservation using the mobile application.
- **[G3]** Allow a Costumer to make a taxi reservation using the web application.
- **[G4]** Grant the possibility to delete a reservation using the mobile application.
- **[G5]** Grant the possibility to delete a reservation using the web application.
- **[G6]** Reduce Costumer waiting time.
- **[G7]** Minimize Taxi Driver down-time.
- **[G8]** Allow a Costumer to make a LiveReservation™ using the mobile application.
- **[G9]** Allow a Customer to pay the ride to TAXISPA.
- **[G10]** Allow the Taxi Driver to reach the right destination of the ride.
- **[G11]** Allow the Taxi Driver to reach the position of the Costumer.
- **[G12]** Notify the Taxi Driver when there is a ride possibility.
- **[G13]** Allow the Costumer to check the current price of a ride.
- **[G14]** Notify a Costumer with the taxi driver's response.
- **[G15]** Allow a Customer to see his/her current position in a ride.
- **[G16]** Allow the SysAdmin to register a new Taxi Driver.
- **[G17]** Allow a Costumer to check his reservation list.

## 1.4. Current state of the service and future prospect

### 1.4.1. SYSTEM AS IS

Taxi drivers are equipped with a cellphone and an earpiece to be able to answer calls during driving.

Taxis are equipped with a proprietary device placed on the cockpit that periodically sends GPS information to *TAXISPA* using GSM connection and acts as a taximeter.

Currently if a user wants to use a taxi he/her must call *TAXISPA* phone number and provide his/her position with accuracy, if possible. Every call is redirected from a switchboard system to an available employer that takes care of the customer. The employer watches a computer screen that shows every taxi driver location over a map to decide which one could be available to take the call.

Then the employer puts on hold the customer and calls the driver to check his/her availability and report the response to the customer.

Currently is not possible to reserve a taxi before the very same day.

### 1.4.2. SYSTEM TO BE

*myTaxiService* will be built from scratch. This new product is not a specific evolution of the existing system.

*myTaxiService* aim to provide new ways of organizing work efficiently to ensure an always growing customer-base and quality of service, reduce the total operating costs of *TAXISPA* and put *TAXISPA* in a stronger competitive position. *myTaxiService* will use mobile and web technologies and will add new features.

## 1.5. Definitions, acronyms and abbreviations

- **RSA**: one of the first practical *public-key cryptosystems*. It is widely used for secure data transmission. It takes the name from the algorithm's inventors (Rivest, Shamir, Adleman).
- **TAXISPA**: big taxi society that wants to develop *myTaxiService*.
- **QoS**: *Quality of Service* is the overall performance of a service (especially from the users' point of view).
- **2PL**: *Two-Phase Locking* is a concurrency control method used in the most recent databases with transaction processing.
- **Timestamp**: a *timestamp* is a sequence of characters or encoded information identifying when a certain event occurred. It can be used with the 2PL in DBMS to improve the concurrency control efficiency.
- **DBMS**: a *DataBase Management* System is a computer software application that interacts with the user, other applications, and the database itself to capture and analyze data.
- **GSM:** *Global System for Mobile Communications*, default global standard for mobile communications.
- **GPS:** *Global Positioning System*, satellite-based navigation system.
- **ADS**: also known as *Advertising,* is a form of marketing communication used to promote or sell something, usually a business's product or service.
- **Queueing Theory**: is the mathematical study of waiting lines, or queues.
- **M/D/k**: In *Queueing theory*, a discipline within the mathematical theory of probability, an M/D/k queue represents the queue length in a system having k servers, where a Poisson process determines arrivals and job service times are fixed.
- **UI:** User Interface, the space where interactions between humans and machines occur.
- **Internal functionality:** functionality is not directly available to the final user but only to system administrator.
- **User**: an abstract class of interest with two subclasses: *Customer* and *Taxi Driver*.
- **mTS**: myTaxiService's acronym.
- **Google Maps**: is a famous mapping service developed by Google.
- **Alloy**: is a declarative specification language for expressing complex structural constraints and behavior in a software system.
- **SAT**: *Boolean Satisfiability Problem* (sometimes called *Propositional Satisfiability Problem* and abbreviated as *SATISFIABILITY* or *SAT*) is the problem of determining if there exists an interpretation that satisfies a given Boolean formula.
- **ETA:** *Estimated Time of Arrival*.
- **Metamodel**: Also known as *Surrogate model*, is a model of a model
- **API**: *Application Programming Interface* is a set of routines, protocols, and tools for building software applications.

## 1.6. References

This document was produced by faithfully following the directives contained in the **IEEE/ANSII 830-1998** (as we said in the chapter *1.1*), with little adjustments.

It also revealed and very useful to consult some of the **RASD** presented over the previous academic years, trying to identify critical issues, patterns and isolate sections developed in an accurate, thorough and organic way.

Here are the documents used as reference:


- M. *Jackson, P. Zave, "Deriving Specifications from Requirements: An Example", Proceedings of ICSE 95, 1995*
- *M. Jackson, P. Zave, "Four Dark Corners of Requirements Engineering", TOSEM, 1997*
- *B. Nuseibeh, S. Easterbrook, "Requirements Engineering: A Roadmap", Proceedings ICSE 2000*
- *M. Jackson, Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices, ACM Press Books, 1995*
- *830-1998 IEEE/ANSII Recommended Practice for Software Requirements Specifications, http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=720574&tag=1&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D720574%26tag%3D1*
- *Various projects of the past years (from the **Beep** platform)*
- *Slides of the course by Prof. Raffaela Mirandola*

# 2. Overall description

## 2.1. Product perspective

Both the mobile application and the web application of *myTaxiService* will be new products of *TAXISPA*. In fact, there will not be an integration with *TAXISPA*'s legacy systems. There will be two different version of *myTaxiService*: the first one for the user and the other for the taxi driver. These versions will have different features and views, as it is possible to imagine.

*Note*: the taxi driver will only have access to the mobile app service (in his case, the web application version is not available).

## 2.2. User characteristics

The use of *myTaxiService* will be easy enough to allow a big number of people to use it. In fact, no special skills are requested. Users must be only able to use the service via mobile application or web application. There is not a "target age" of users: everyone is a potential user. Anyway, there is an age limitation for people under 18 years (*see also 2.3.1*).

## 2.3. Constraints

### 2.3.1. Regulatory policies

People under 18 years cannot use *myTaxiService*.

### 2.3.2. Hardware limitations

*myTaxiService* doesn't have hardware limitations.

### 2.3.3. Interfaces to other applications

There is only the integration with *Google Maps*. There are not interfaces between *myTaxiService* and other applications.

### 2.3.4. Parallel operation

Parallelism is very important for *myTaxiService* service. We attend many requests: so parallel processing and dynamic queue management are crucial.
Because of this fact, the system supports parallelism and simultaneous transactions according to the latest technologies in this field. For example, the **DBMS** uses *2PL + Timestamp* for the

concurrency control. Parallelism will be also used to manage the multiple queues for the different zones.

## 2.4. Assumptions and Dependencies

**Assumptions** and **Dependencies** of *myTaxiService* are in the following list:

We will use some hypothetical *variables names* to help us explain better the model. We will refer to those variables in *Class Diagram* and *Alloy document*.

- There are only three types of *RegisteredUser* accounts: standard user (*Customer*) account, taxi driver account (*TaxiDriver*) and system administrator (*SysAdmin*).
- Now there is not dependency between different costumers. Otherwise, taxi sharing may be an idea for the future.
- A user can have also access to the web application version using a browser (like *Chrome, Safari, Firefox*…).
- A user can do limitless reservations for future events.
- If a *Costumer* deletes a reservation, he can do again the same reservation in the future if necessary without particular problems.
- There are not ads both in the web and in the mobile application.
- There are not relevant differences from the functional point of view between the mobile application and the web application.
- The queue management model is based on an efficient *M/D/k* application of *Queuing Theory*.
- Two different types of position are handled by mTS System.
- When a *Costumer* is in the taxi cannot delete a reservation.
- According to a user request, a taxi driver can change destination while he is driving.
- The city is divided in *Area* (2km$^2$ each).
- Each *Area* is associated to a one and only one queue of *TaxiDriver* (*listOfDrivers*).
- There are two possible types of position: *GPSPosition* and *AddressPosition.*
- A *GPSPosition* could be associated to at most one valid (=in mTS range of expertise) *AddressPosition*
- Each *Area* is disjointed (it means that every *GPSPosition* belonging to a certain *Area* cannot belong to another *Area*).
- Each *Area* is associated to a one and only one (eventually empty) list of reservations (*listOfReservations).*
- Each *Area* must have a number of Taxi Drivers between two and five.
- To every *Reservation* is associated a time/date, and two ~~*GPS*~~ position ~~*GPSPosition*~~ *AddressPosition* (*from*, *to*).
- When a *LiveReservation* is done *Customer currentPosition* must be associated to a valid *AddressPosition*
- mTS will only accept reservations where starting point and destination point belongs to roads in mTS area of expertise.
- Every *Reservation* belongs to *listOfReservations* of the *Area* that *from* belongs to.
- Every *Reservation* has a destination (*to*) different from the starting point (*from*).

- A taxi driver cannot belong to more than one *listOfDrivers* at the same time.
- Every *listOfReservations* is ordered to always have on top the *Reservation* with maximum priority (the one that the associated time/date is the closest one).
- When a *TaxiDriver* is not working (break, vacation or holiday) does not belong to any *listOfDrivers.*
- *Customer* must make his/her reservation two hours or more before the chosen date.
- Each *Zone* has unique Coordinates.
- Every *RegisteredUser* has a different identifier (ID).
- A *TaxiDriver* cannot be *currentlyBusyOn* a *Reservation* before the acceptation of the reservation.
- Every *Reservation* has a different identifier (ID).

## 2.5. Future possible implementations

- At launch mTS mobile application will only be available for *iOS* and *Android*. We will try our best to release *Windows Phone* version before the end of the year.
- Public API will be accessible to the public 2 months after the release of the product. Documentation and SDK/emulator will be available at launch for independent developers.
- mTS mobile application will allow customers to pay through phone credit and In-App Purchase (iOS/Android).
- We will create mobile version of the app to reach old mobile devices, which cannot support our app.

## 2.6. Stakeholders identification

It is possible to distinguish between two categories of stakeholders of *myTaxiService*:

### Internal Stakeholders

- Taxi drivers of *TAXISPA*
- Mangers of *TAXISPA*
- Employees of *TAXISPA*
- Other personnel of *TAXISPA*
- System Admin (*SysAdmin*)

### External Stakeholders

- Costumers

- Sponsors
- Direct Competitors
- Un-direct Competitors
- App developers
- App testers

# 3. Specific requirements

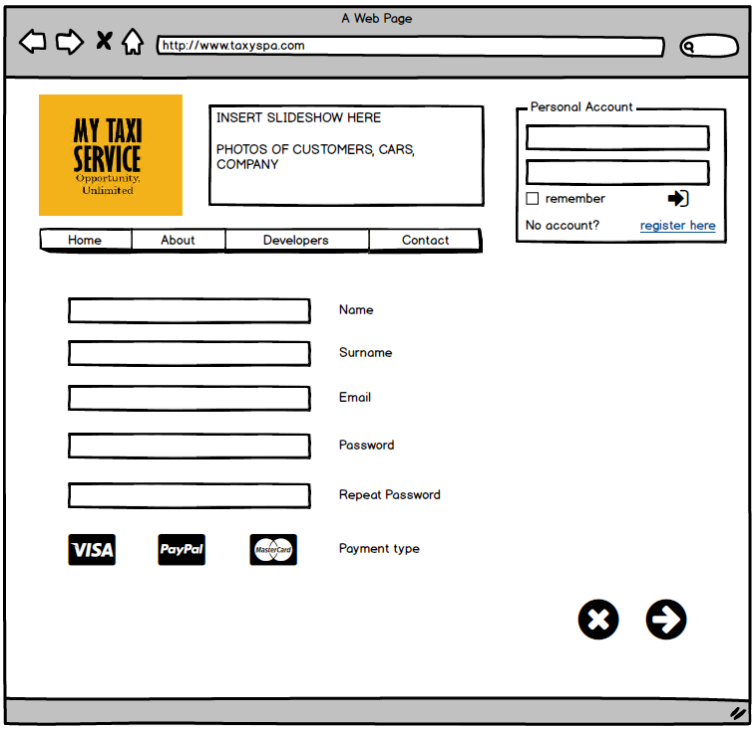
Identify the right requirements may be a difficult thing to do if the approach is not good enough. The main thing to understand is the link between what happens in the real world (*The World*) and the software technologies (*The Machine*). This link is *Requirements Engineering*.

## 3.1. External interface requirements

### 3.1.1. User Interfaces

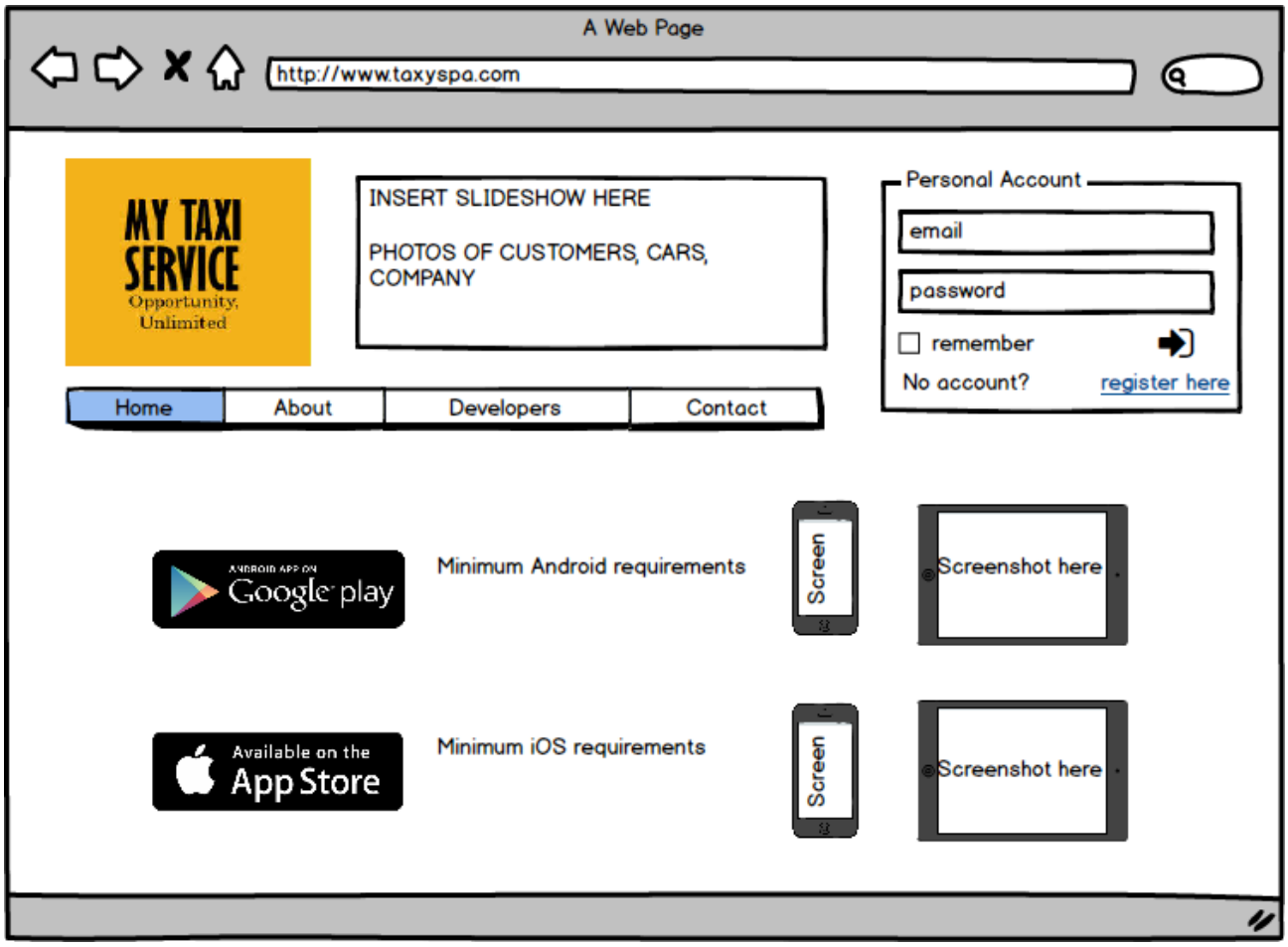
We have developed some *mock-ups* to give a general idea of *myTaxiService* application:

- REGISTRATION PAGE - This is the mock-up about the registration page. A Visitor must insert all the required data (name, surname, email, password, payment type):

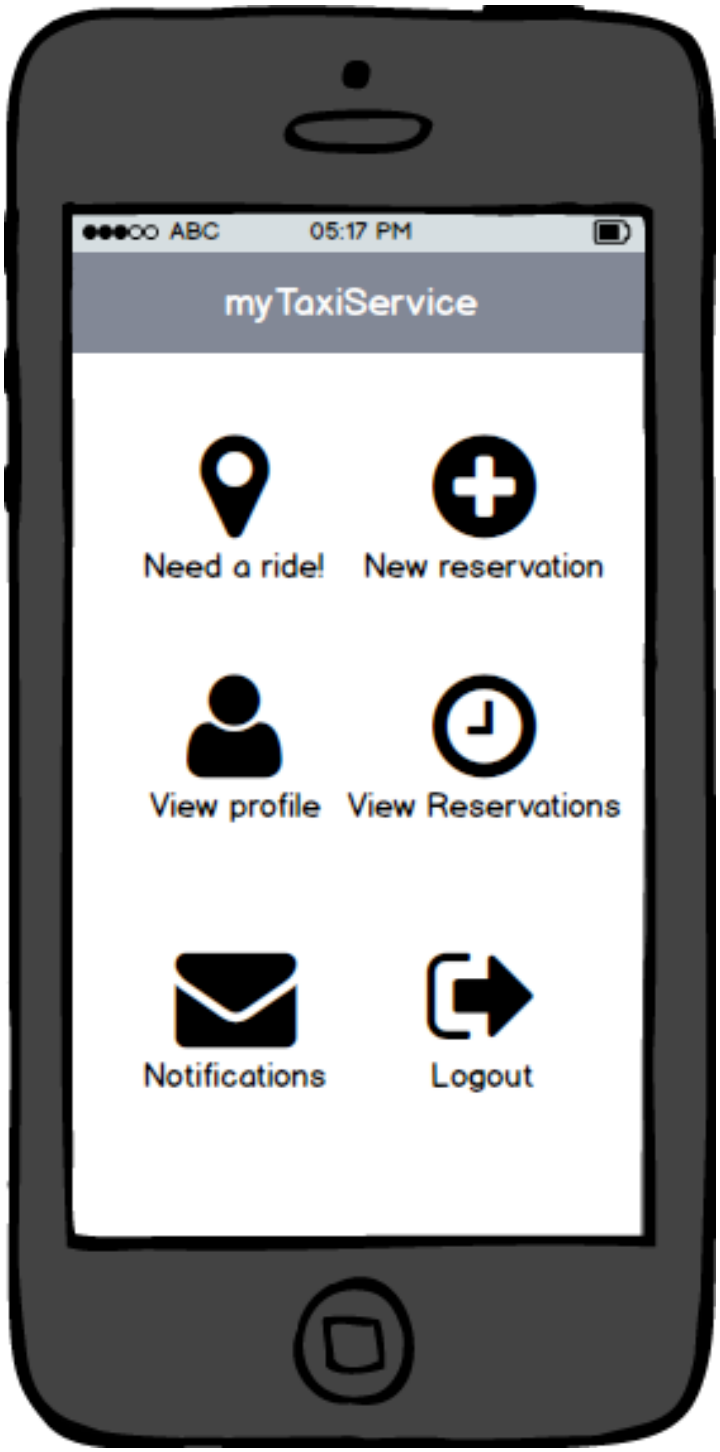


- MOBILE APP LOGIN - This is the mock-up about the login in the mobile app version of mTS. As the intuition suggests, a general user must insert his/her registration email and the chosen password to log-in.

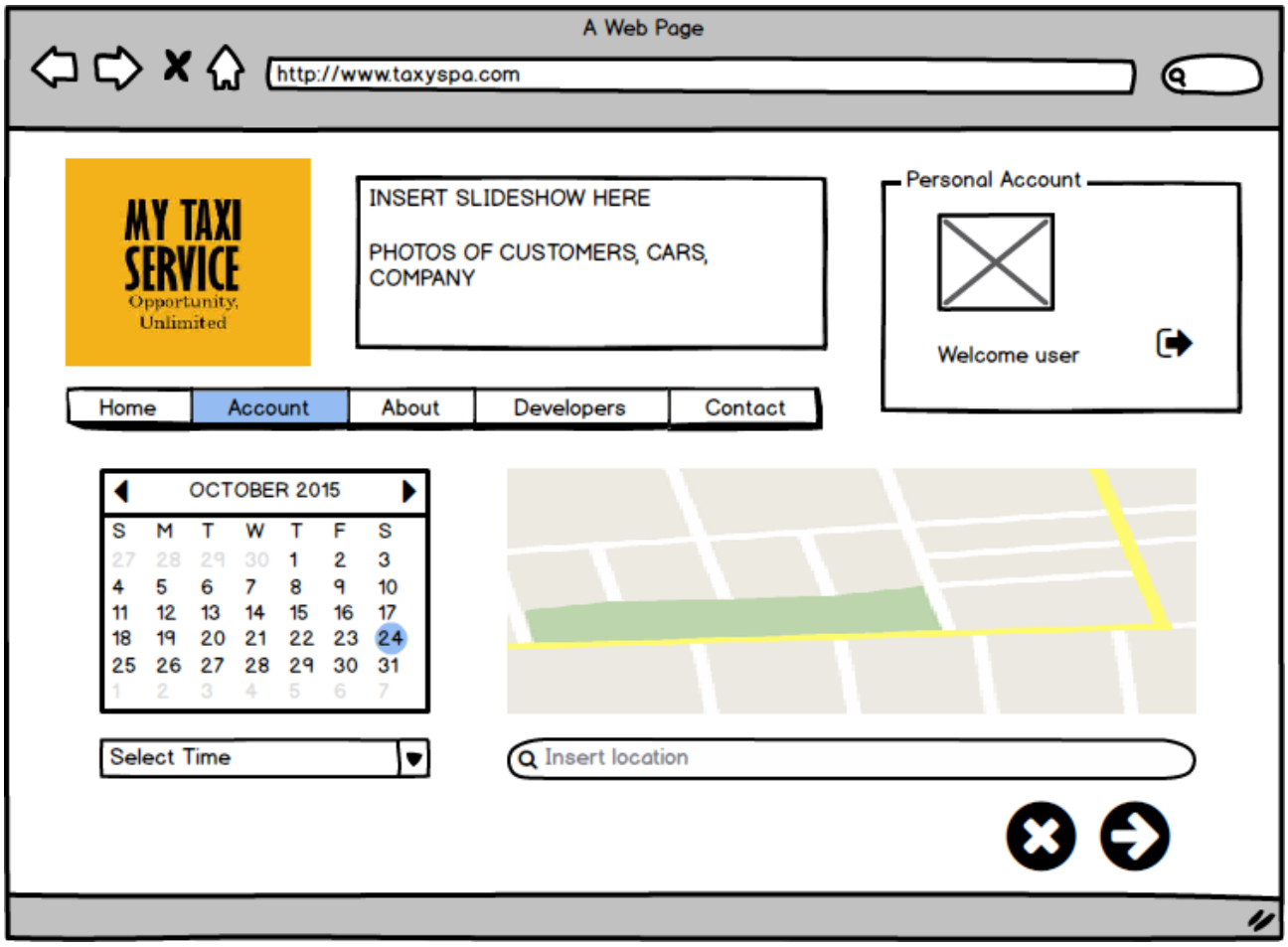
- WEB APPLICATION HOMEPAGE - This is the mock-up about the homepage of mTS (Web Application Version). There are also two banners on the left of the screen about the mobile versions of *myTaxiService* (for *Android* and *iOS*).



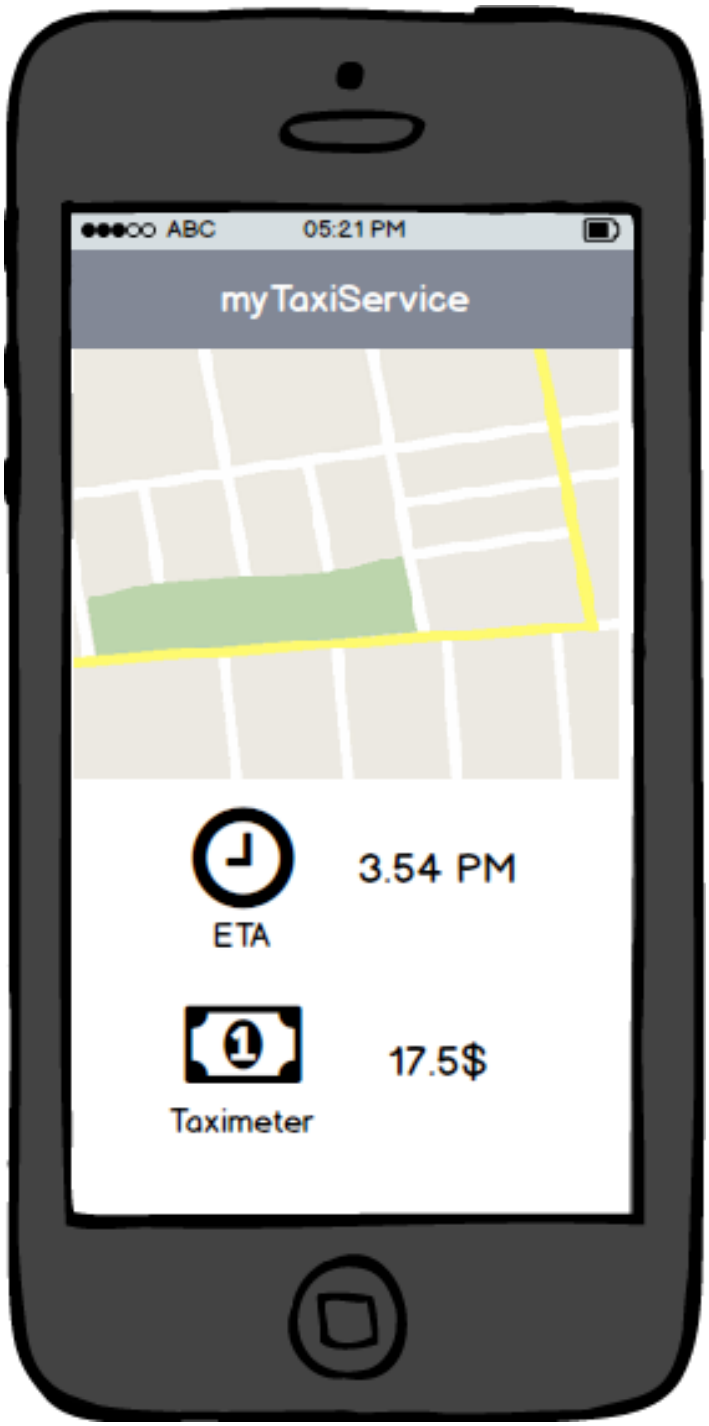
- MOBILE APP HOMEPAGE - This is the mock-up about the homepage of mTS (Mobile App Version). The menu has various features: use the LiveReservation™ with the *"Need a Ride!"* option, make a new reservation, view the profile, view the list of reservations, view the notifications and have the possibility of logout.



- WEB APPLICATION RESERVATION - This is the mock-up about the reservation of a ride (Web Application version). There is a calendar on the left of the screen, with the possibility of insert the time of the ride. On the right, there is the position of the location (in *Google Maps*).



- COSTUMER WITH MOBILE APP IN NAVIGATION - This is the mock-up about a costumer with the mobile app during a ride. On the bottom of the screen, it is possible to see ETA and the taximeter in real time.



- TAXI DRIVER NAVIGATION SYSTEM - This is the mock-up about the navigation system of Taxi Driver. It is possible to see the indications of GPS on the screen. There are also the messages about Costumer's reservations on the top of the screen (with time details).

## 3.2. Functional Requirements

### 3.2.1. Registration of a user to the system

- The system has to provide sign up functionality to every customer.
- The system has to provide system admins of an internal sign up functionality to register taxi drivers.

### 3.2.2. Login of a user to the system

- The system has to provide a login mechanism for user.
- The system has to provide system admins of an internal login mechanism to log-in drivers.

### 3.2.3. Make financial transactions

- The system has to provide a mechanism to allow customers to send their money to *TAXISPA*.

### 3.2.4. Route taxi driver to certain location

- The system must be able to send driving directions (to a certain location) to taxi drivers.

### 3.2.5. Create/delete/update taxi reservations

- The system has to provide a functionality to allow a customer to create a taxi reservation.
- The system has to provide a functionality to allow customer view taxi reservations.
- The system has to provide a functionality to allow a customer to delete taxi reservations.
- The system has to provide a functionality to allow a customer to update taxi reservations.

### 3.2.6. Send information about travels/reservations to a customer

- The system will be able to notify users about their reservations.
- The system will be able to update users during their travels in mTS taxis.

### 3.2.7. Accept/Decline system request to take care of a customer

- The system will allow taxi driver to accept a new customer.
- The system will allow taxi driver to decline a new customer.

### 3.2.8 Request/acquire user ~~GPS~~ position

- The system will provide a way to acquire periodically taxi drivers GPS position.
- The system will provide a way to request customers ~~GPS~~ position (privacy concerns, user can deny).

## 3.3 Scenarios

### SCENARIO 1

Sergey Brin is planning to hang out with other colleagues of Google this Friday after work. However, there is a problem: Friday nights are famous for the traffic jams. He is a very precise person and he prefers to book a taxi in advance to reach the destination. Therefore, he decides to login to his account at MyTaxiService website and insert all the data for the taxi reservation. He selects one taxi, only for him, near Google Maps research center in Mountain View at 8.30 pm. After the submission of the data, the system confirms Sergey's reservation without problems. The request is stored in the system's database.

### SCENARIO 2

Mark Zuckerberg is waiting his taxi in front of Facebook Park in the Silicon Valley: he is finally going to meet the famous Russian Professor Markesanskjy. However, while he was attending his guest, he receives a WhatsApp message from the Professor. The Professor is still not available for the meeting… so Mark Zuckerberg can come back in the park. While he is walking, he realizes that has to delete the taxi reservation. Therefore, he immediately opens the mobile app of *myTaxiService* and delete the reservation. The system sends him an acknowledge confirmation.

## SCENARIO 3

Bill Gates wants to buy a new big house with many rooms to earn a Guinness World Record award. In order to visit the big house, he is waiting for his taxi, but in the meantime a crash occurs and his booked taxi cannot reach the location. Fortunately, the system provides Bill an alternative taxi (he is an important and rich client!) and send him a notification with the announcement of the possible delay, of course!

## SCENARIO 4

Robert De Niro is a young taxi driver that has just left a passenger in Martin Scorsese Road, the destination place. While Robert is waiting for another call (he is in the area's queue of course), he can drink something (non-alcoholic of course!) with his old girlfriend Jodie Foster.

## SCENARIO 5

Night: 11 pm. Robert De Niro is driving his taxi, carrying two passengers, on the 1st Avenue. He is going to reach the destination in about five minutes. In the meantime, the system signals him another call from the 2nd Avenue (very close to the first one, as it is possible to imagine). The system notifies him that he should reach the 2nd Avenue in about ten minutes. In addition, he is a very good and fast taxi driver. Therefore, he confirms with his earphones. The system sends a notification to the client. It seems that Robert is very lucky this night… he will earn a lot of money!

## SCENARIO 6

Marty McFly is a student from California. At 1.30 a.m., he receives a phone call from his friend: the professor Emmet Brown alias "Doc". The professor wants to see Marty as fast as possible: the question is very important! Thus, Marty immediately picks up his smartphone and opens *myTaxiService* mobile application. Unfortunately, no taxi is currently available in his area. How to solve this problem? Just waiting a few minutes: the service will provide a taxi as soon as possible. No worries Marty, you will see Doc soon!

## 3.4 Non-functional requirements

### 3.4.1 Performance requirements

The system will send the notification of taxi availability time after the user's reservation almost tree minute after the reservation. The taxi driver on the top of reservation's queue will have one minute to accept the user reservation. If the taxi driver declines the reservation, the system puts him in the end of the queue and sends the notification to another taxi driver (the following in the queue). The percentage of *crash for usage* in the mobile app will be under 1% for the mobile app and under 0.5% for the web app.

### 3.4.2 Availability

*myTaxiService* motto is "*Opportunity Unlimited*", so the system will be always 24 hours per day and 7 days a week (*full time*) in standard days. In case of updates, the system will be down only form *3 am* to *5 am* (when the number of users' requests is small). To guarantee the availability the company will also buy powerful servers (a server farm, in fact).

### 3.4.3 Maintainability

The application's code will follow the principles and standards of "*good programming*" (right commenting, clean and simple coding, design patterns using and so on). The full documentation of *myTaxiService* will be stored in *TAXISPA*. With these precautions, new developers of the service will know how the system works in detail in order to ensure an optimal maintainability.

### 3.4.4 Portability

Thanks to the mobile application support, the service will easily run on millions of devices. In fact, the mobile application will be compatible with a big amount (*>=90%*) of devices with Android and iOS (the most common operative systems in mobile devices, like smartphones and tablets).

### 3.4.5 Scalability

It is important to consider the scalability factor. Possibilities of new modules may be:

- *Taxi Sharing Service:* that allows two or more users to share a taxi and save a little amount of money.
- *Ride Review System:* the user has the possibility to evaluate the ride and the taxi driver (like a feedback).
- *Other modules:* to be defined in the future.

### 3.4.6 Security

Sensible data will be stored in a database with firewall (in both hardware and software) protection. Another crucial aspect is the payment service via app: this will happen according to an asymmetric *256-bit RSA cryptography system* (in order to gain protection in the transaction).

# 4. Use Case and UML Models

## 4.1. Use Case and related UML Models

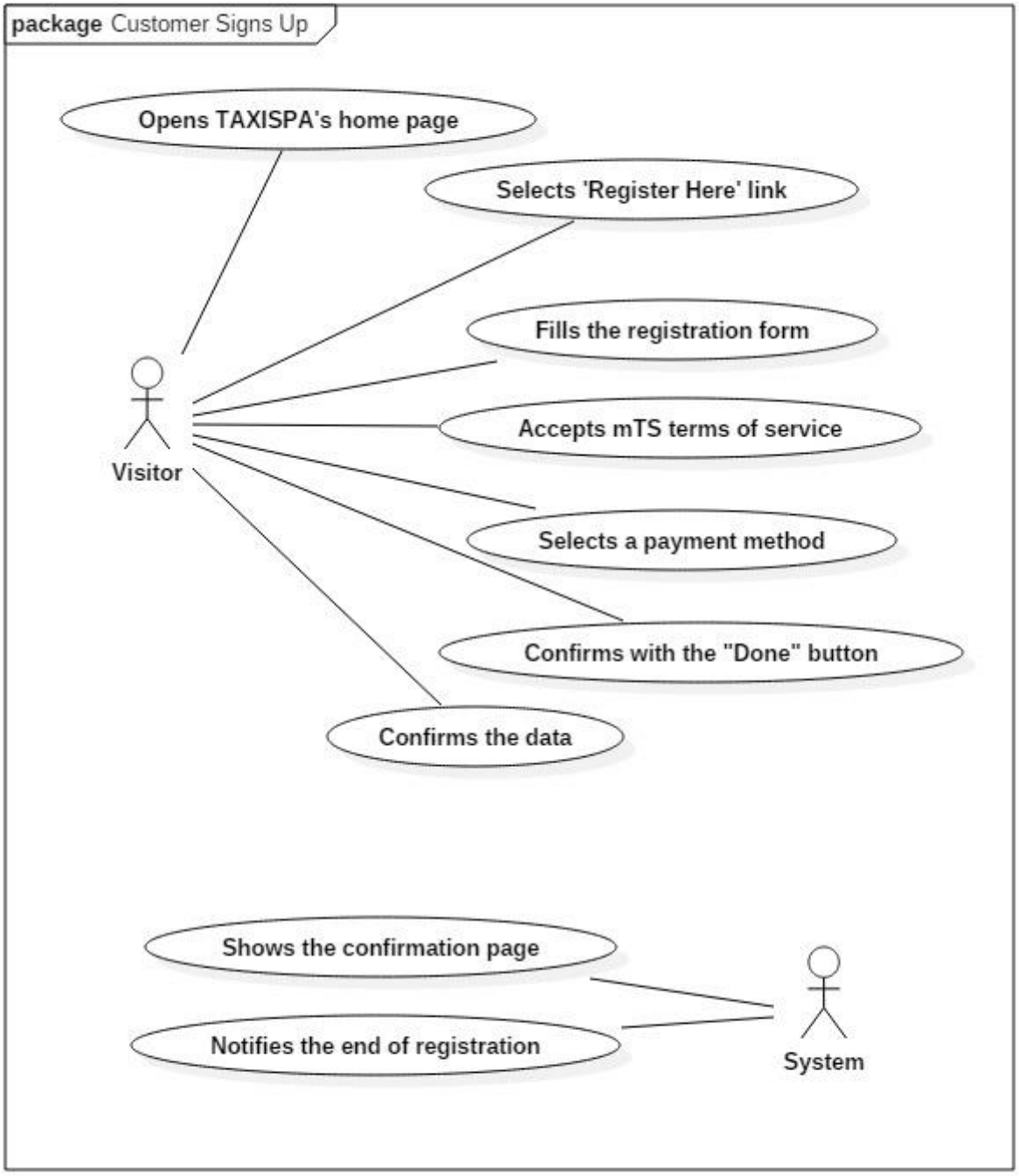
We describe the most important use cases from our point of view. For simplicity, we have decided to omit some use cases.
Please, keep in mind that all references to “pages”, “fields” or “buttons” are only hypothesis to make the situation clearer. With the help of mockups, the reader would not find difficult to imagine the different features of the service. A more detailed point of view about actual webpage/application interaction is exposed in *Design Document.*

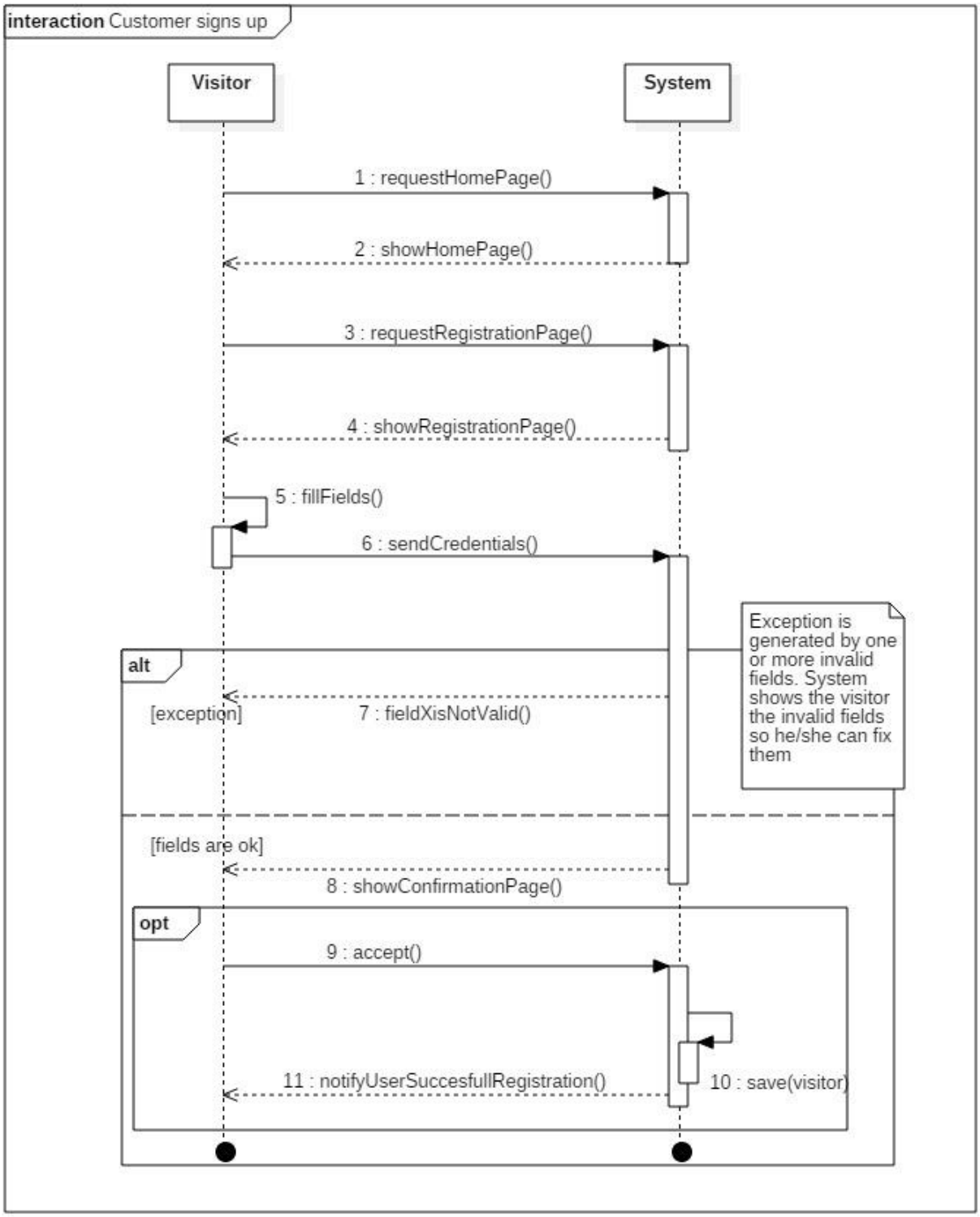
**LIST OF POSSIBLE ACTORS:**

- **Visitor**
- **Customer**
- **System Admin (SysAdmin)**
- **Taxi Driver**
- **System**

### 4.1.1. Customer Signs Up

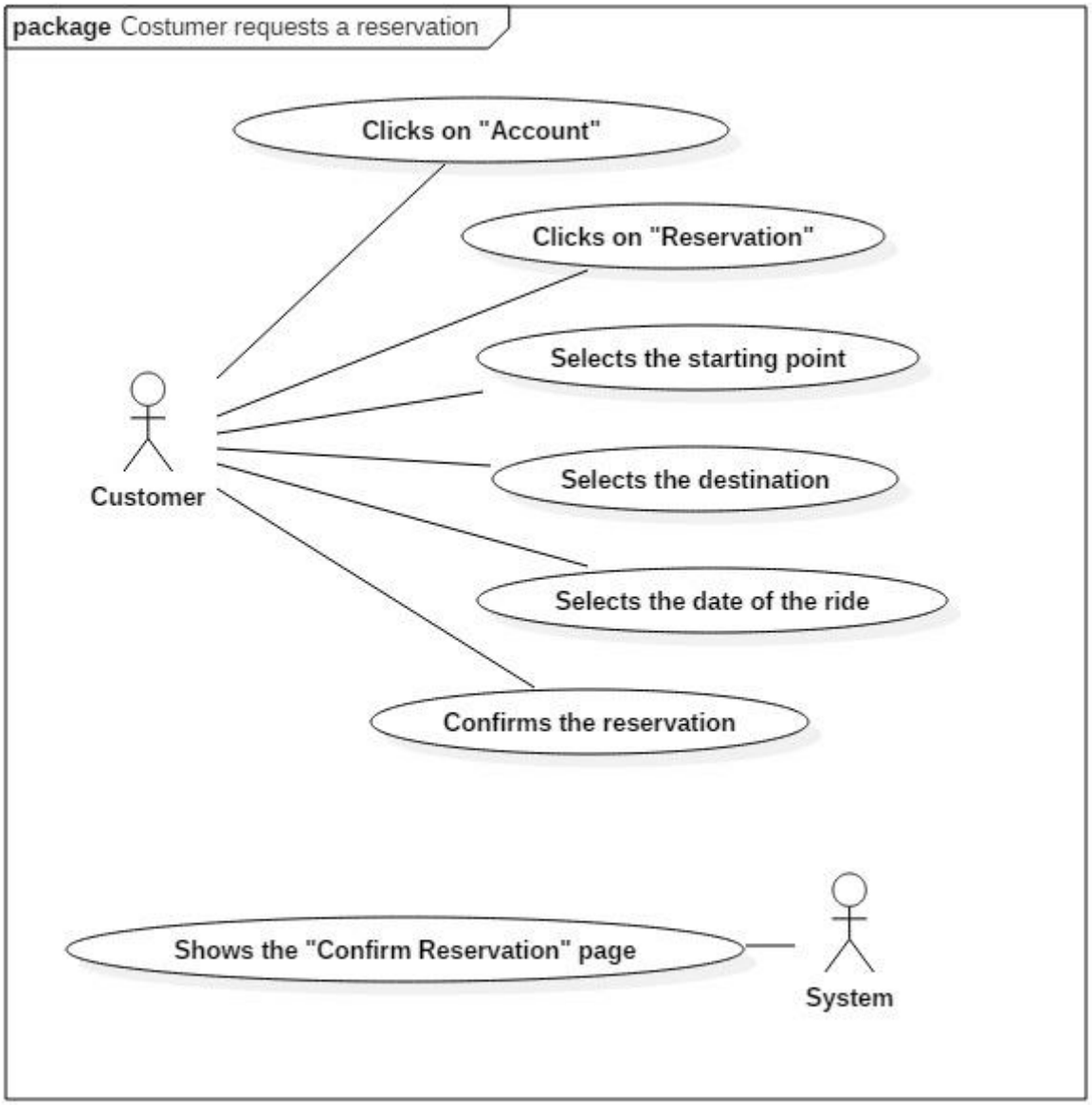
| | |
|---|---|
| Actors | Visitor, System |
| Goals | [G0] |
| Input conditions | • A visitor wants to sign up to use mTS service. |
| Flow of events | 1. A visitor opens the home page of *TAXISPA*<br>2. The visitor selects *"register here"* link<br>3. The visitor fills in the required fields:<br>• Name<br>• Surname<br>• Email<br>• Password<br>• Repeat Password<br>• Birthdate<br>• SSN<br>4. The visitor accepts mTS terms of service<br>5. The visitor picks a payment method from a list of provided ones<br>6. The visitor clicks the "done" button<br>7. The system shows the visitor the confirmation page<br>8. The visitor clicks the "confirm reservation" button<br>9. The system notifies the visitor that he/she has completed the registration |
| Output conditions | • A new customer with provided credentials is registered. |
| Exceptions | • Malformed mail<br>• Mail is already used<br>• Repeat password field not equal to Password field<br>• Visitor is not at least eighteen<br>• Payment method not accepted/rejected<br>• System down<br>• Visitor has the same SSN of an already registered Customer<br>• At least one required fields is not filled in |

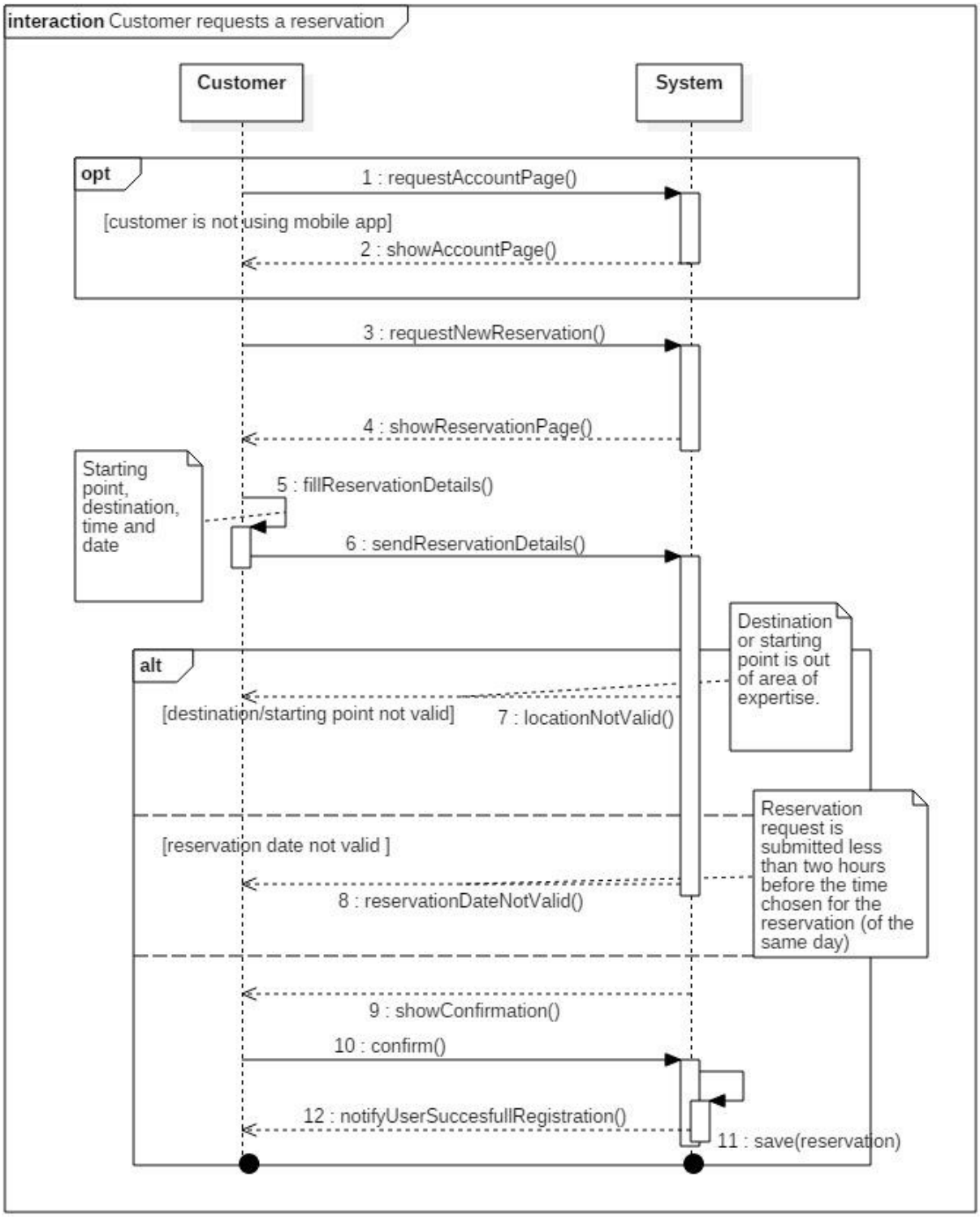

package Customer Signs Up
Opens TAXISPA's home page
Selects 'Register Here' link
Fills the registration form
Accepts mTS terms of service
Selects a payment method
Confirms with the "Done" button
Confirms the data
Visitor
Shows the confirmation page
Notifies the end of registration
System


interaction Customer signs up
Visitor
System
1 : requestHomePage()
2 : showHomePage()
3 : requestRegistrationPage()
4 : showRegistrationPage()
5 : fillFields()
6 : sendCredentials()
Exception is generated by one or more invalid fields. System shows the visitor the invalid fields so he/she can fix them
alt
[exception]
7 : fieldXisNotValid()
[fields are ok]
8 : showConfirmationPage()
opt
9 : accept()
10 : save(visitor)
11 : notifyUserSuccesfullRegistration()

### 4.1.2. Customer requests a reservation

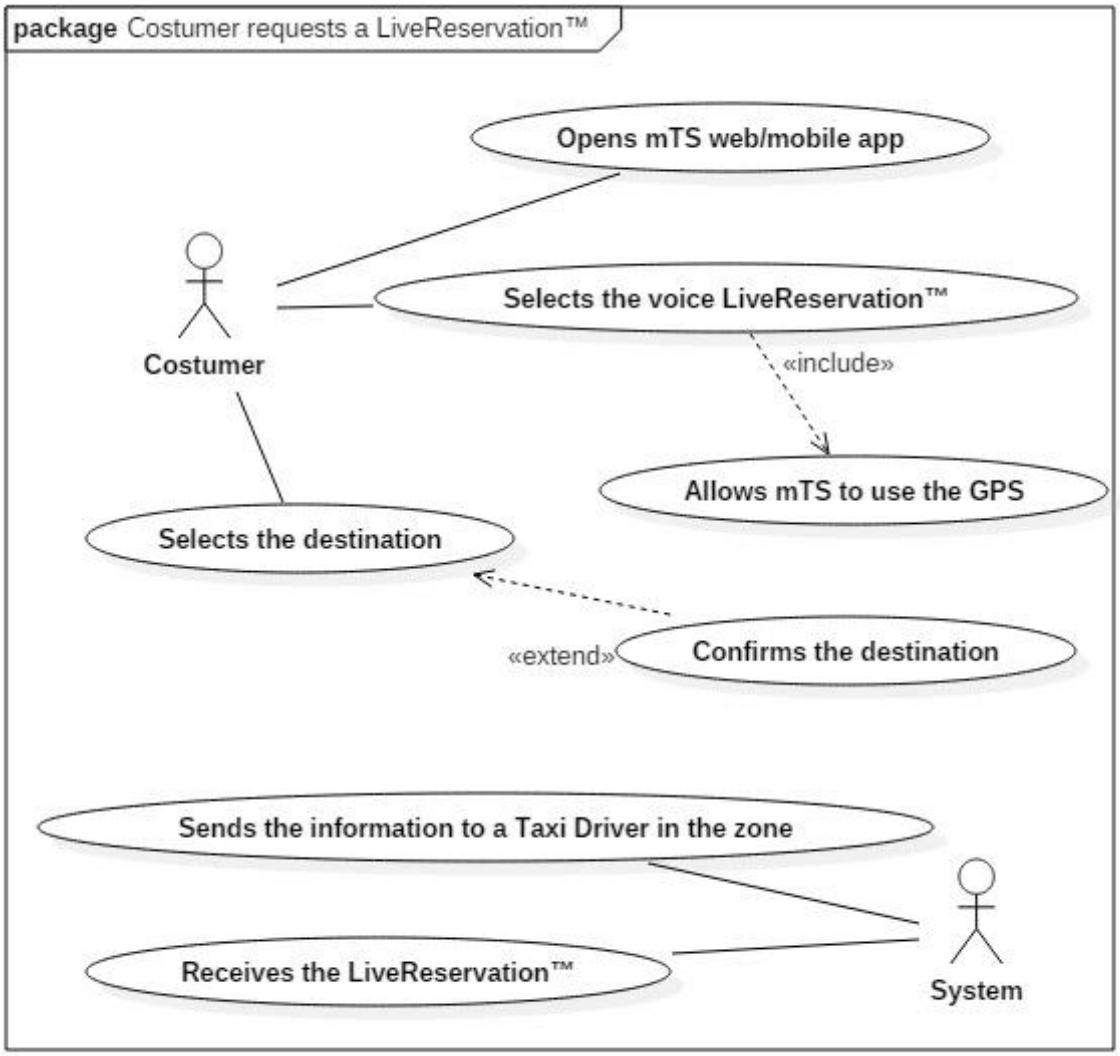
| | |
|---|---|
| Actors | Customer, System |
| Goals | [G2], [G3] |
| Input conditions | • The user is already registered on mTS.<br>• The user is already logged in as a Customer. |
| Flow of events | 1. Customer clicks “account” link<br>2. Customer clicks on “new reservation”<br>3. Customer selects starting point (from)<br>4. Customer selects destination (to)<br>5. Customer selects date<br>6. Customer selects time<br>7. System shows “confirm reservation” page<br>8. Customer confirms reservation |
| Output conditions | • A new reservation associated to the customer with the inserted data is saved on mTS system.<br>The reservation is now added to the customer’s reservation list (*listOfReservations*).<br>• The reservation is put in the priority queue of the *from* area. |
| Exceptions | • Customer is making his/her reservation less than two hours before the chosen date<br>• Chosen starting point and/or destination does not belong to *TAXISPA*. area of expertise<br>• System down |

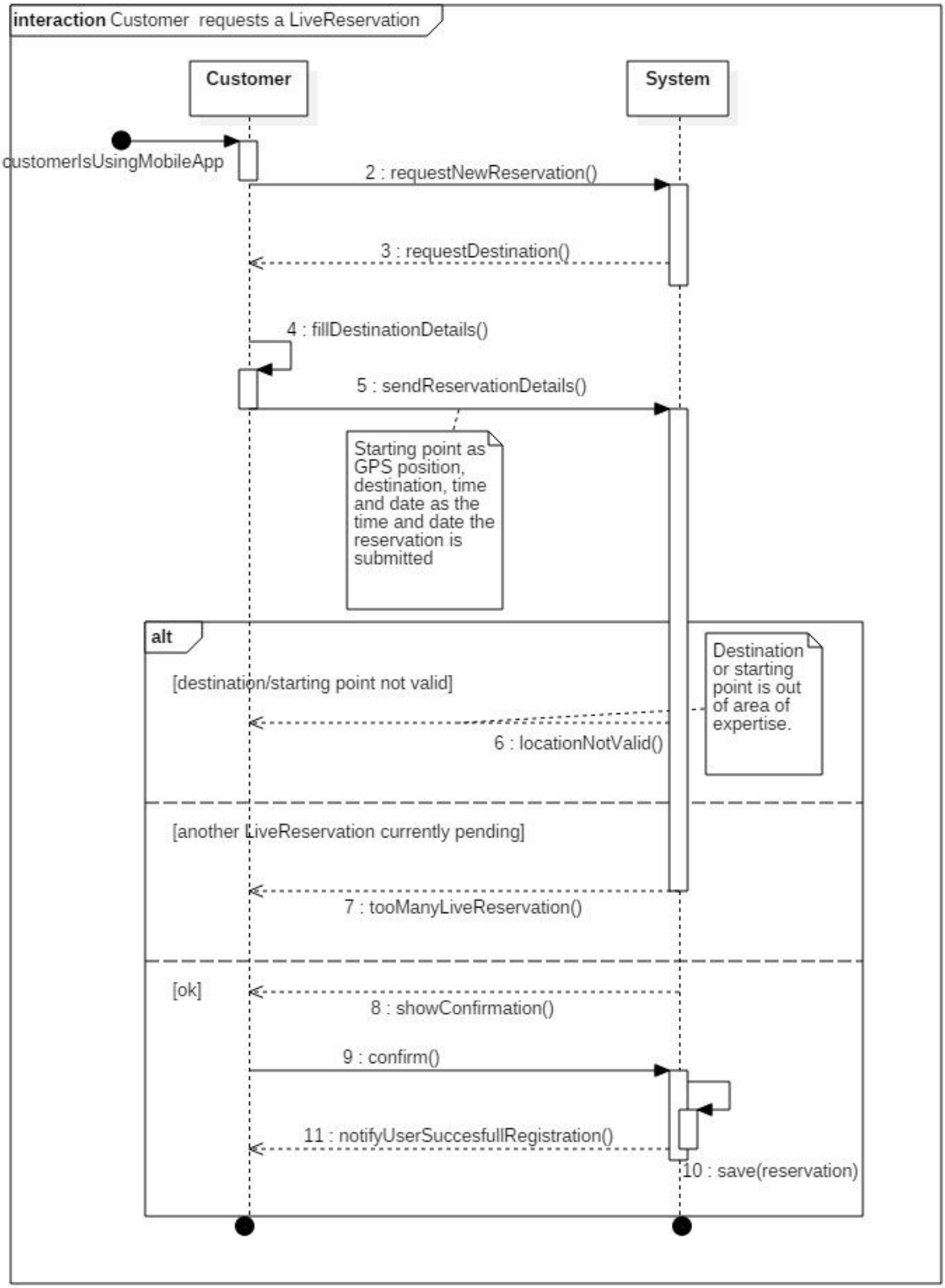

package Costumer requests a reservation
Clicks on "Account"
Clicks on "Reservation"
Selects the starting point
Selects the destination
Selects the date of the ride
Confirms the reservation
Customer
Shows the "Confirm Reservation" page
System


interaction Customer requests a reservation
Customer
System
opt
1 : requestAccountPage()
[customer is not using mobile app]
2 : showAccountPage()
3 : requestNewReservation()
4 : showReservationPage()
Starting point, destination, time and date
5 : fillReservationDetails()
6 : sendReservationDetails()
Destination or starting point is out of area of expertise.
alt
[destination/starting point not valid]
7 : locationNotValid()
[reservation date not valid ]
Reservation request is submitted less than two hours before the time chosen for the reservation (of the same day)
8 : reservationDateNotValid()
9 : showConfirmation()
10 : confirm()
12 : notifyUserSuccesfullRegistration()
11 : save(reservation)

### 4.1.3. Customer requests a LiveReservation™

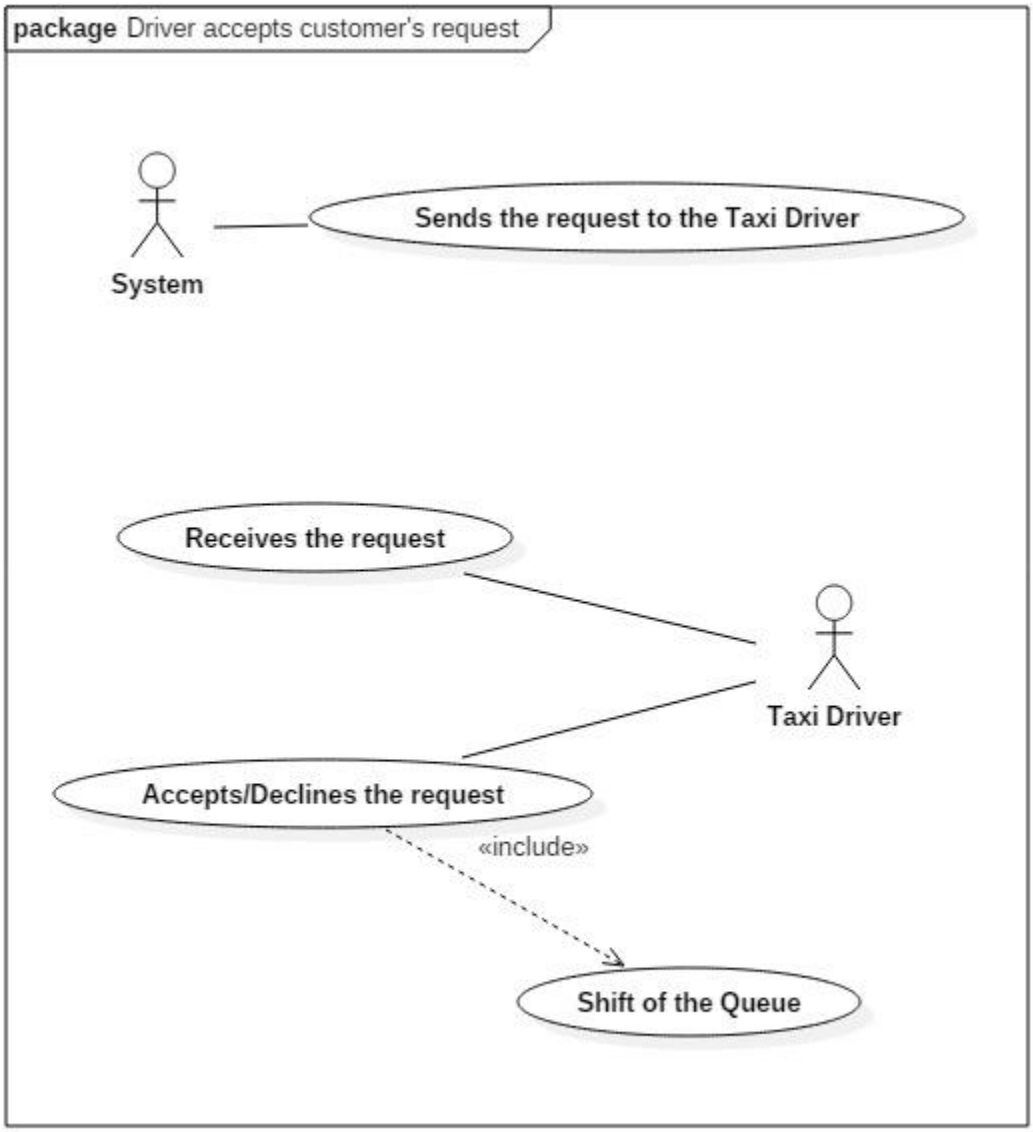
| | |
|---|---|
| Actors | Customer, System |
| Goals | [G8] |
| Input conditions | • The user is already registered on mTS.<br>• The user is already logged in (mobile app) as a Customer.<br>• mTS mobile app is available to the User.<br>• Customer has accepted mTS privacy policy about location sharing. |
| Flow of events | 1. Costumer opens mTS app<br>2. Costumer selects the voice "LiveReservation" and his/her GPS<br>3. Costumer selects the desired destination<br>4. Costumer confirms the LiveReservation<br>5. The System receives the LiveReservation<br>6. The System sends the information about the LiveReservation to a Taxi Driver in the zone |
| Output conditions | • A new reservation associated to the customer with the inserted data is saved on mTS system.<br>• The reservation is now added to the customer's reservation list (*listOfReservations*).<br>• The reservation is put in the priority queue of the *from* area. |
| Exceptions | • GPS customer signal has less than 50 meters of accuracy<br>• Chosen starting point and/or destination doesn't belong to TAXISPA area of expertise<br>• System down |

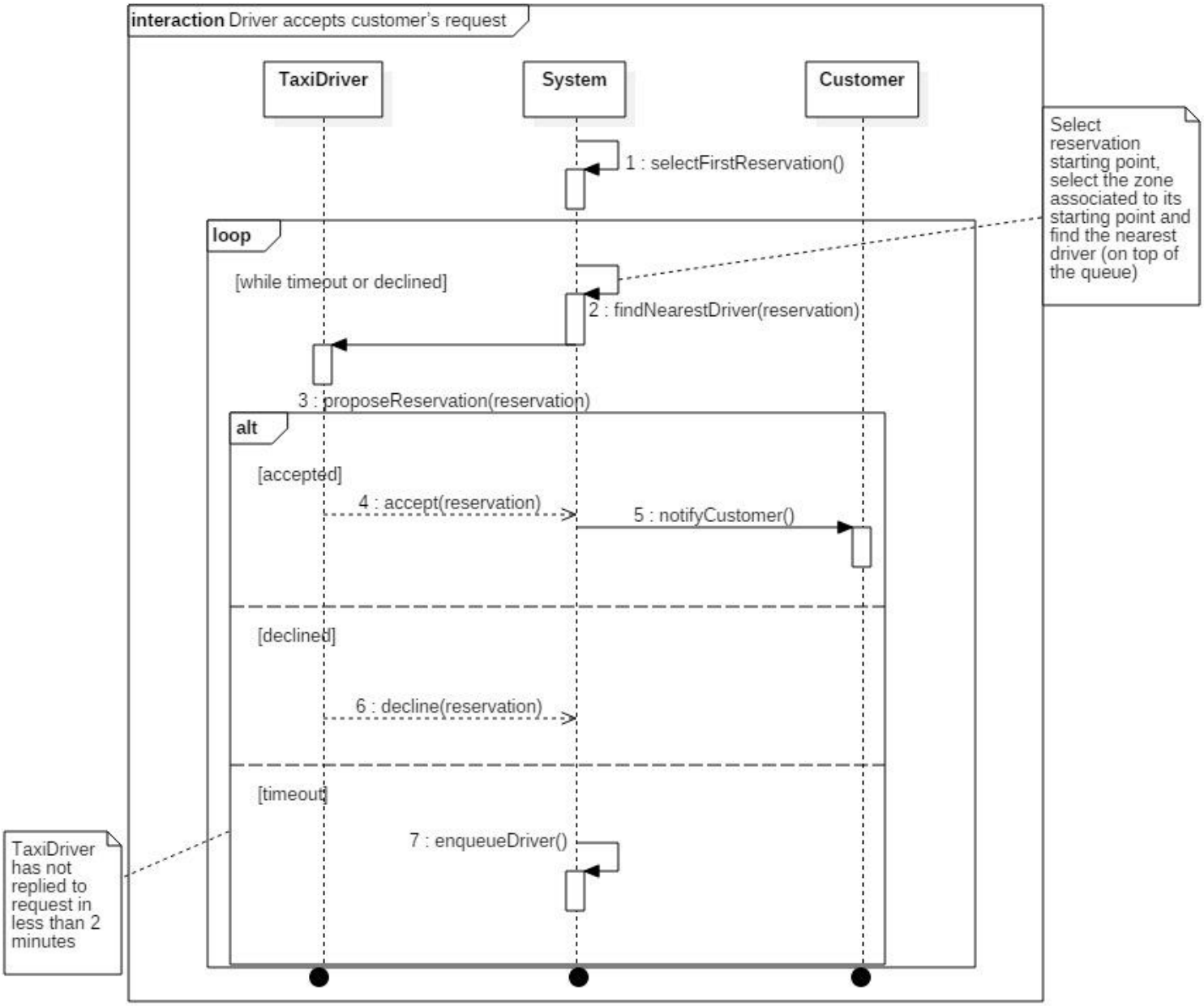

package Costumer requests a LiveReservation™
Opens mTS web/mobile app
Costumer
Selects the voice LiveReservation™
«include»
Allows mTS to use the GPS
Selects the destination
«extend»
Confirms the destination
Sends the information to a Taxi Driver in the zone
Receives the LiveReservation™
System


interaction Customer requests a LiveReservation
Customer
System
1 : customerIsUsingMobileApp
2 : requestNewReservation()
3 : requestDestination()
4 : fillDestinationDetails()
5 : sendReservationDetails()
Starting point as GPS position, destination, time and date as the time and date the reservation is submitted
alt
[destination/starting point not valid]
Destination or starting point is out of area of expertise.
6 : locationNotValid()
[another LiveReservation currently pending]
7 : tooManyLiveReservation()
[ok]
8 : showConfirmation()
9 : confirm()
11 : notifyUserSuccesfullRegistration()
10 : save(reservation)

### 4.1.4. Driver accepts customer's request

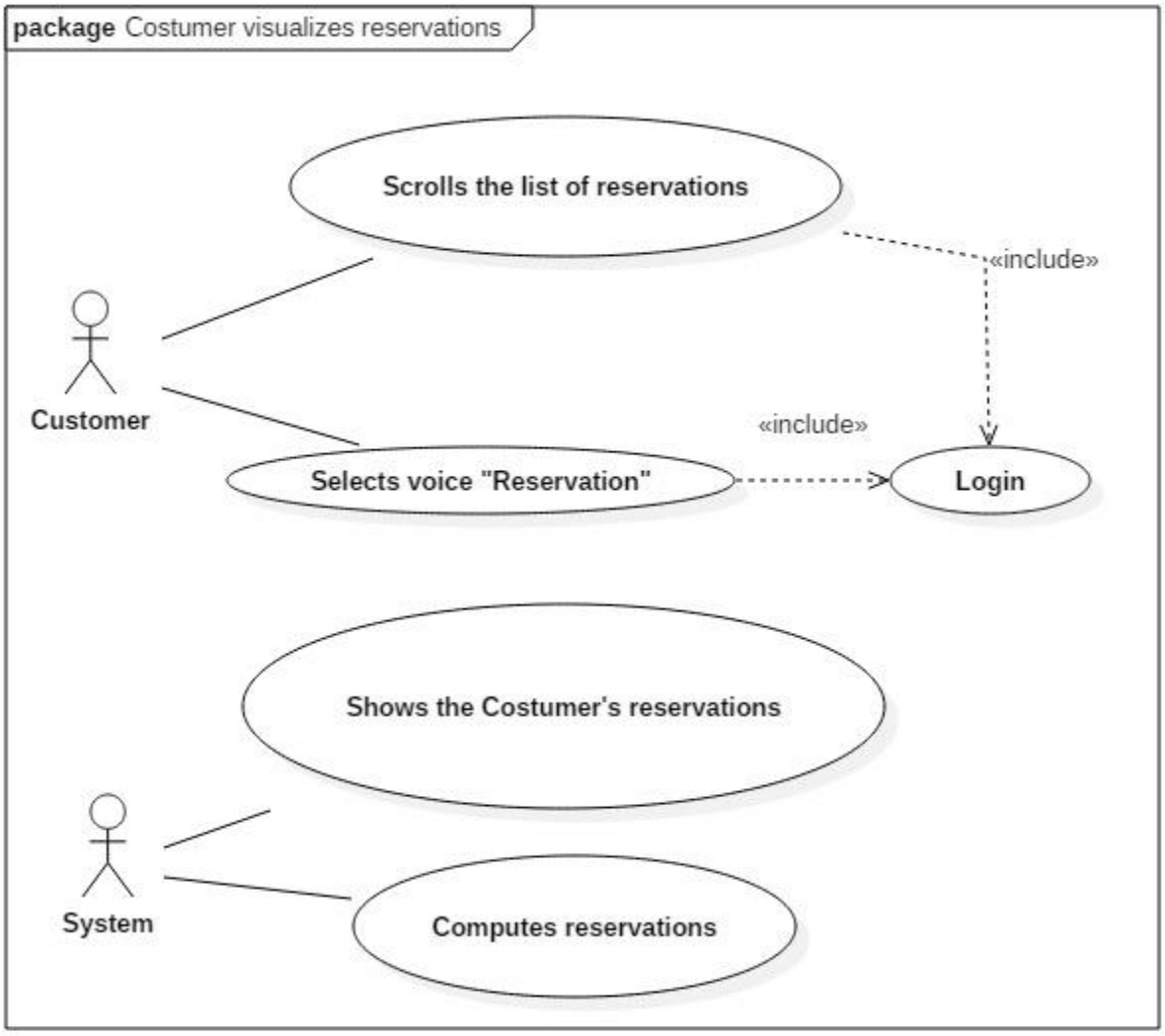
| | |
|---|---|
| Actors | System, Taxi Driver |
| Goals | [G11] |
| Input conditions | • System takes care of a reservation ten minutes before the reservation hour.<br>• Taxi Driver is on top of the queue of the starting area chosen in the reservation. |
| Flow of events | Repeat until you have a positive response from a Taxi Driver:<br><br>1. System sends request to the Taxi Driver<br>2. Driver receives the request<br>3. Driver accepts/declines request<br>4. If a driver accepts, System notifies the Customer associated with the reservation with an ETA |
| Output conditions | • If the Taxi Driver accepts then he/she is moved to the bottom of the destination area queue and the Customer is notified.<br>• If the taxi driver declines, then he/she is moved to the bottom of the starting area queue. |
| Exceptions | Taxi Driver doesn't reply in time<br>System down |


package Driver accepts customer's request
System
Sends the request to the Taxi Driver
Receives the request
Taxi Driver
Accepts/Declines the request
«include»
Shift of the Queue


interaction Driver accepts customer's request
TaxiDriver
System
Customer
1 : selectFirstReservation()
Select reservation starting point, select the zone associated to its starting point and find the nearest driver (on top of the queue)
loop
[while timeout or declined]
2 : findNearestDriver(reservation)
3 : proposeReservation(reservation)
alt
[accepted]
4 : accept(reservation)
5 : notifyCustomer()
[declined]
6 : decline(reservation)
[timeout]
7 : enqueueDriver()
TaxiDriver has not replied to request in less than 2 minutes

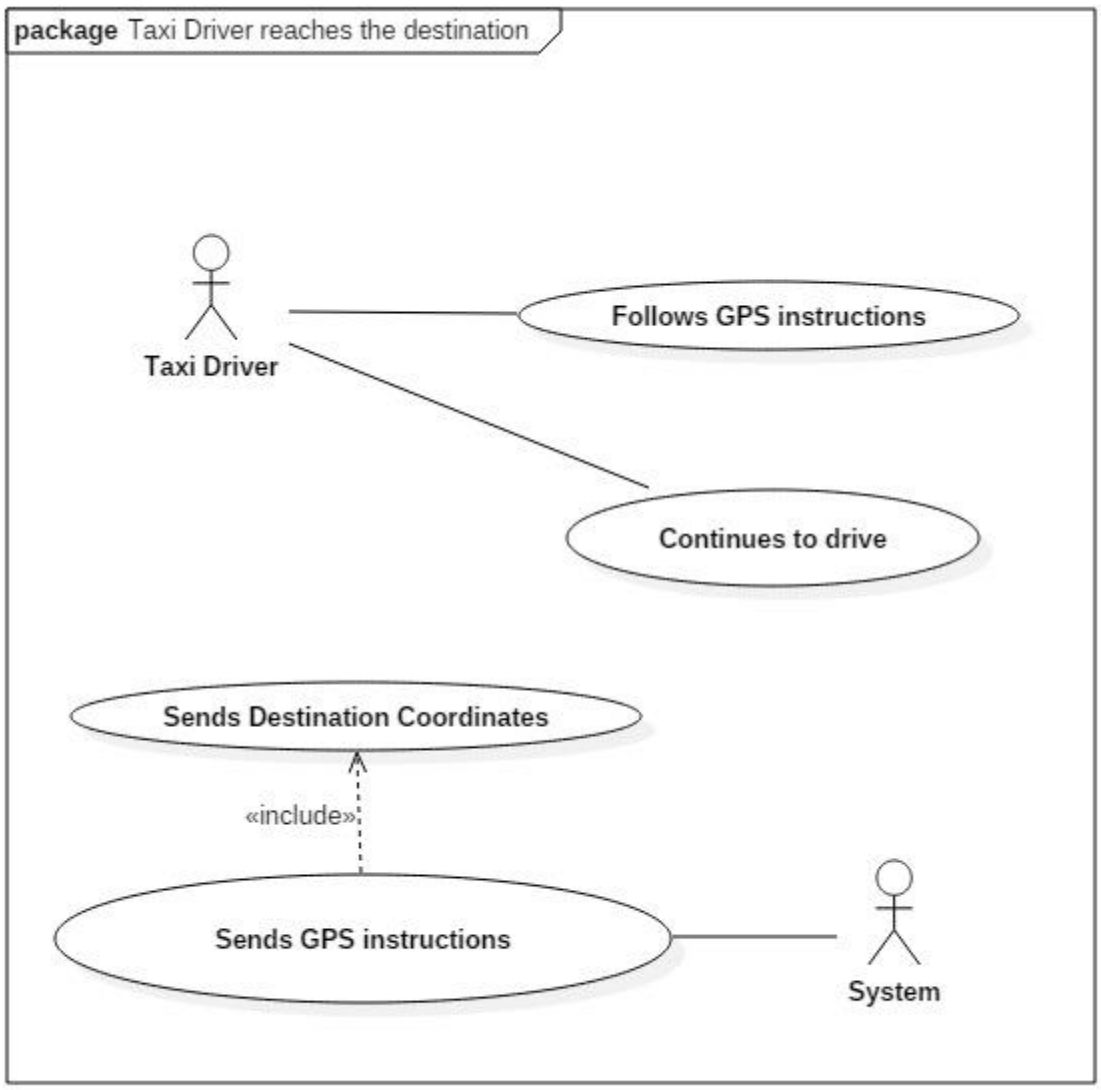
### 4.1.5. Customer visualizes reservations

| | |
|---|---|
| Actors | Customer, System |
| Goals | [G17] |
| Input conditions | • The customer must go on the web application page or must open the mobile application. |
| Flow of events | 1. The costumer does the login.<br>2. The costumers select the voice “Reservations”.<br>3. The system computes all the costumer’s reservations.<br>4. The costumer can scroll the list of his reservation(s). |
| Output conditions | • The costumer has seen his/her reservation(s), if any. |
| Exceptions | System down |


package Costumer visualizes reservations
Scrolls the list of reservations
«include»
Customer
«include»
Selects voice "Reservation"
Login
Shows the Costumer's reservations
System
Computes reservations

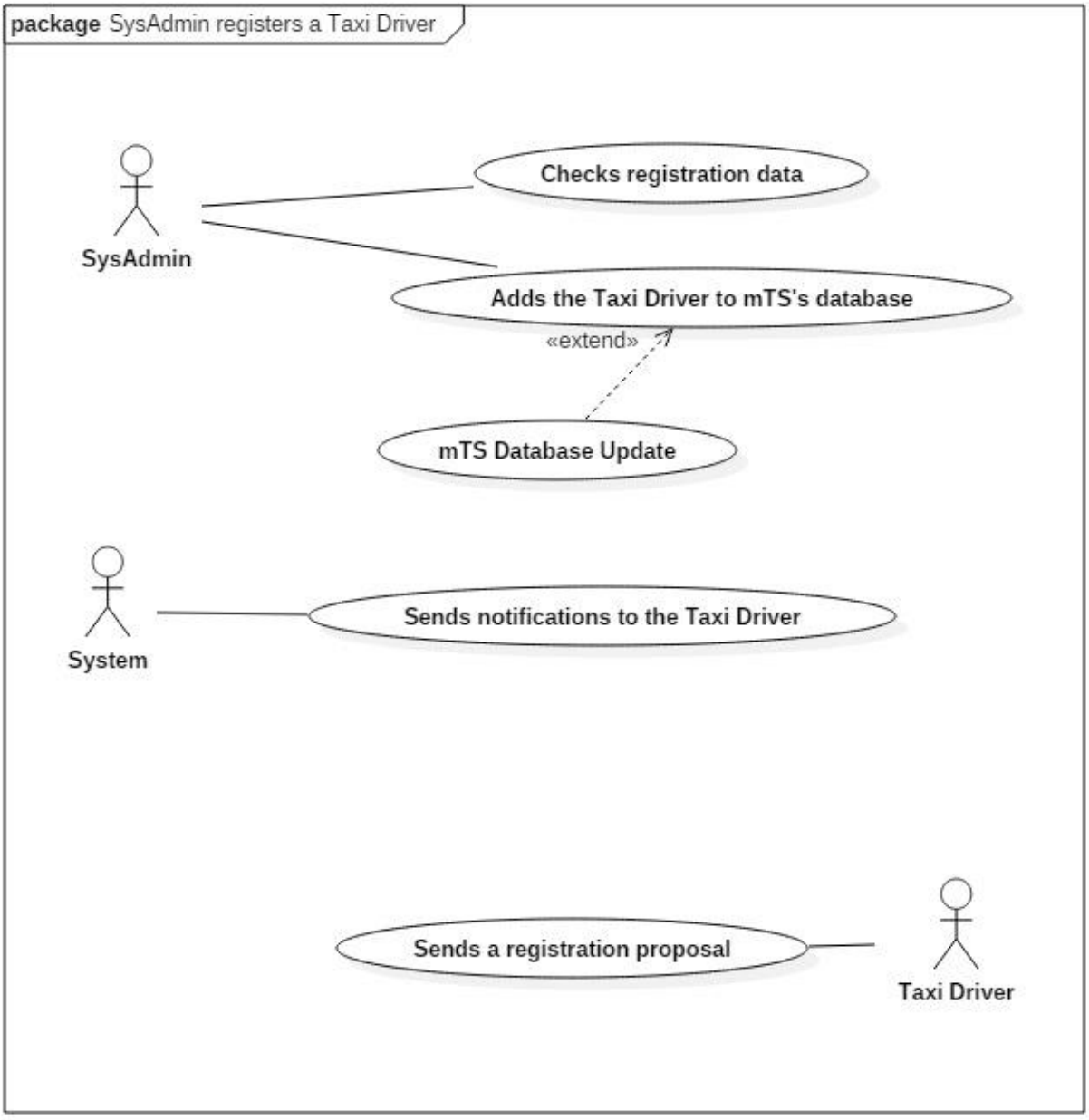
### 4.1.6. Taxi Driver reaches the destination

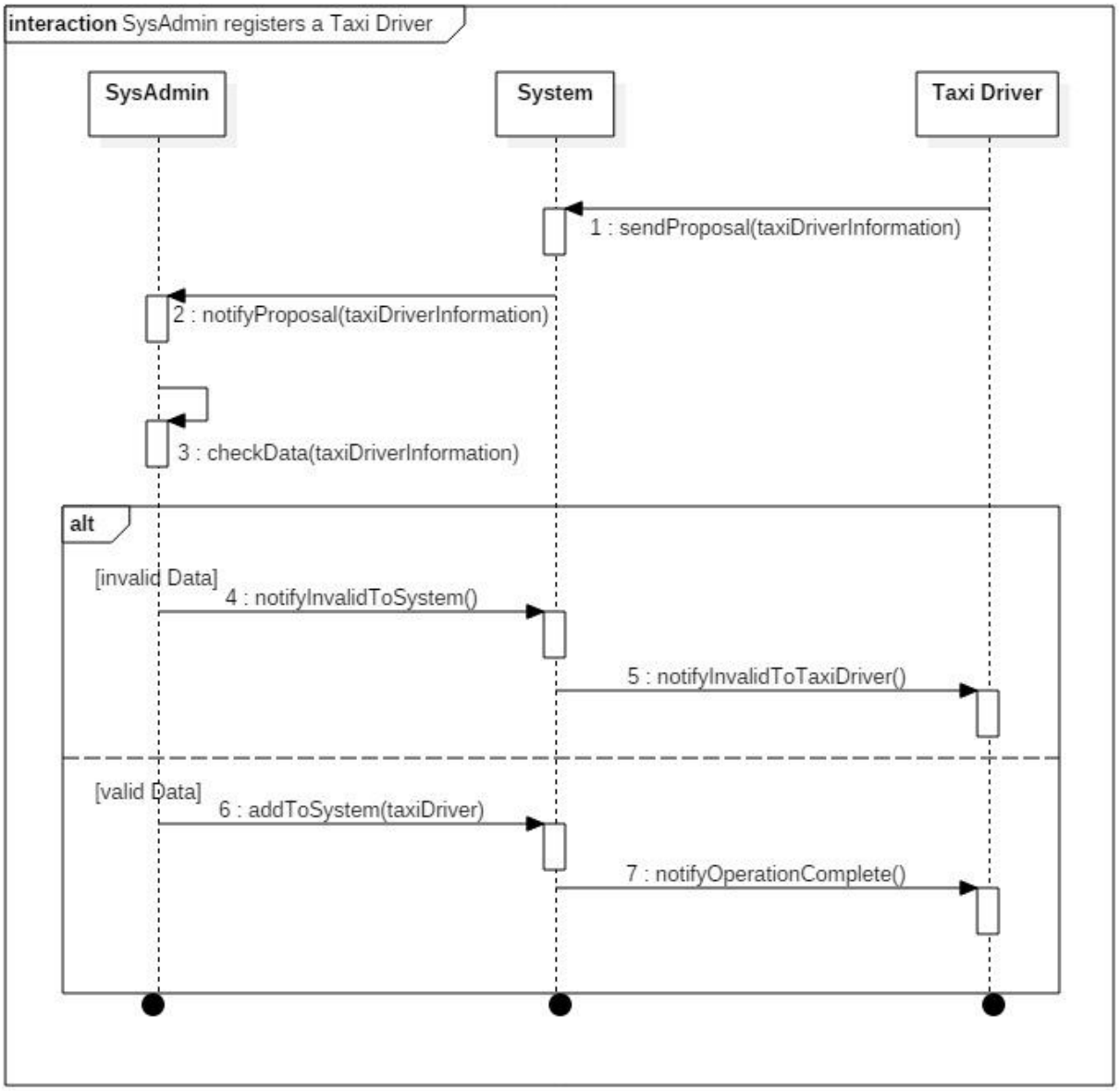
| | |
|---|---|
| Actors | Taxi Driver, System |
| Goals | [G10] |
| Input conditions | • Taxi Driver must have his mobile application available.<br>• The GPS position must be available. |
| Flow of events | 1. The System sends him the step-to-step instructions to reach the destination through GPS.<br>2. He continues to drive, following the GPS indications, until he reaches the target position. |
| Output conditions | • Taxi Driver reaches the right coordinates of destination. |
| Exceptions | • GPS not available<br>• Crash during the ride<br>• Emergency during the ride<br>• Reservation deleted<br>• Invalid coordinates<br>• System down |


package Taxi Driver reaches the destination
Taxi Driver
Follows GPS instructions
Continues to drive
Sends Destination Coordinates
«include»
Sends GPS instructions
System

### 4.1.7. SysAdmin registers a Taxi Driver

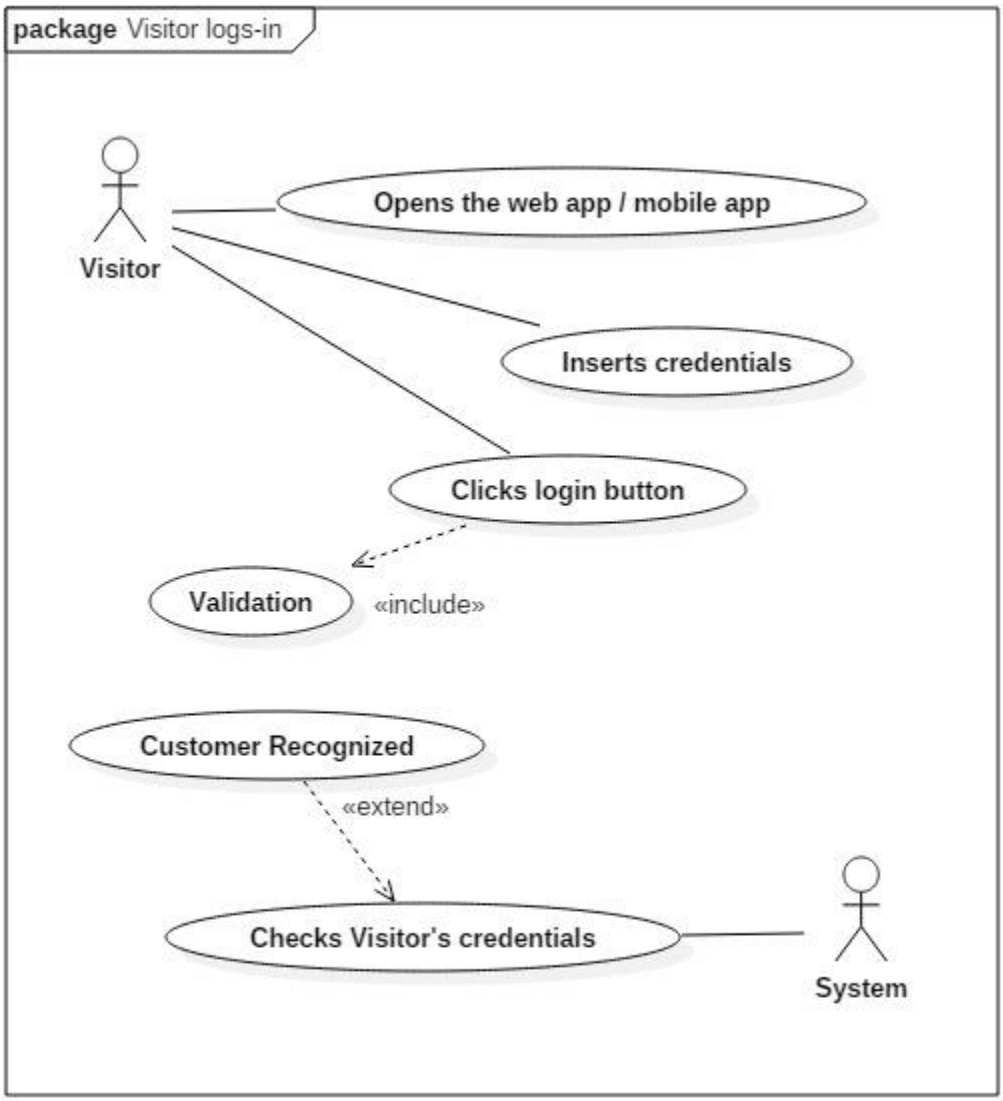
| | |
|---|---|
| Actors | SysAdmin, System, Taxi Driver |
| Goals | [G16] |
| Input conditions | • A new Taxi Driver hired from *TAXISPA*.<br>• An old Taxi Driver member of *TAXISPA* wants to join *myTaxiService*. |
| Flow of events | 1. The new Taxi Driver sends a proposal to the system with the information requested for the registration<br>2. The SysAdmin checks if all the data are correct<br>3. The SysAdmin insert the new Taxi Driver in the System's database, with a Taxi Driver account (of course)<br>4. When the database update is committed, the System sends an acknowledge to the Taxi Driver |
| Output conditions | • The selected Taxi Driver becomes an active and available member of *myTaxiService*. |
| Exceptions | • Taxi Driver already registered<br>• System down<br>• Missing information |

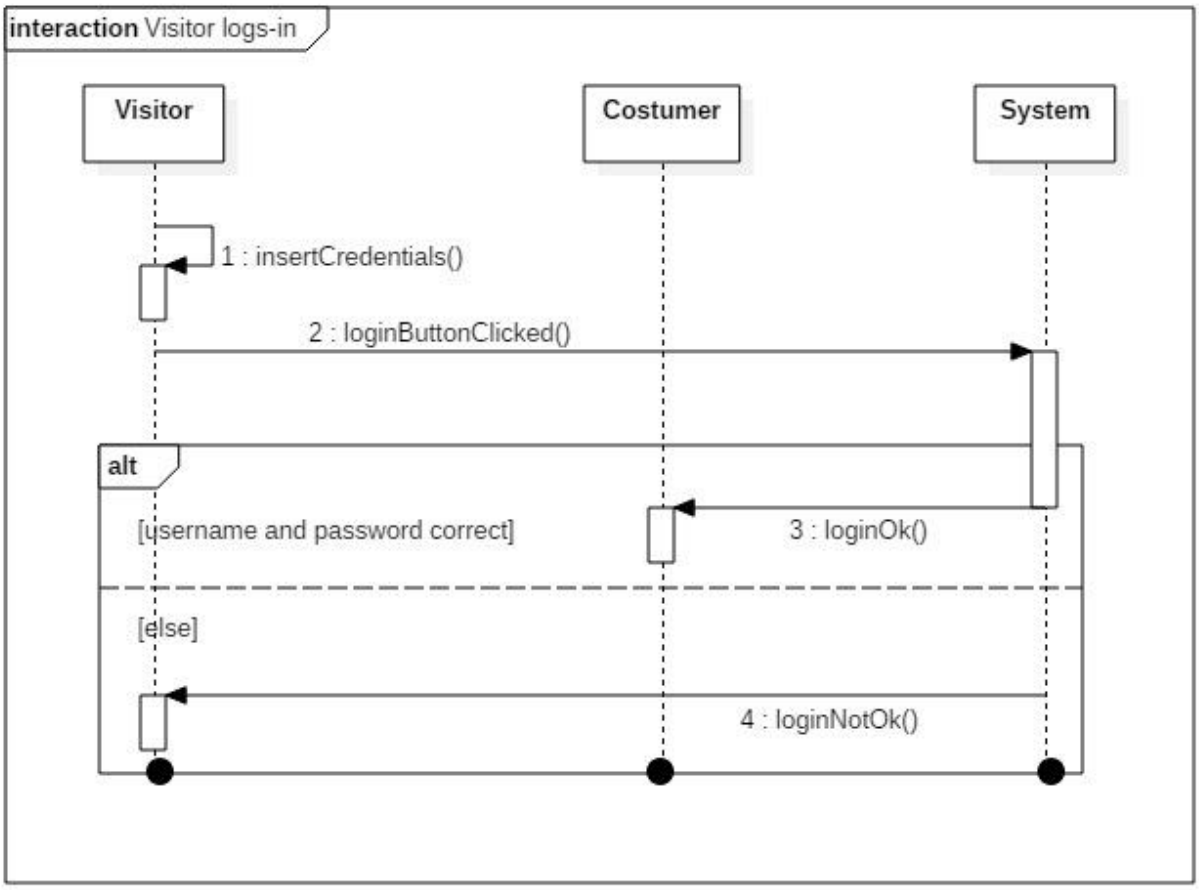

package SysAdmin registers a Taxi Driver
SysAdmin
Checks registration data
Adds the Taxi Driver to mTS's database
«extend»
mTS Database Update
System
Sends notifications to the Taxi Driver
Sends a registration proposal
Taxi Driver


interaction SysAdmin registers a Taxi Driver
SysAdmin
System
Taxi Driver
1 : sendProposal(taxiDriverInformation)
2 : notifyProposal(taxiDriverInformation)
3 : checkData(taxiDriverInformation)
alt
[invalid Data]
4 : notifyInvalidToSystem()
5 : notifyInvalidToTaxiDriver()
[valid Data]
6 : addToSystem(taxiDriver)
7 : notifyOperationComplete()

### 4.1.8. Visitor logs-in

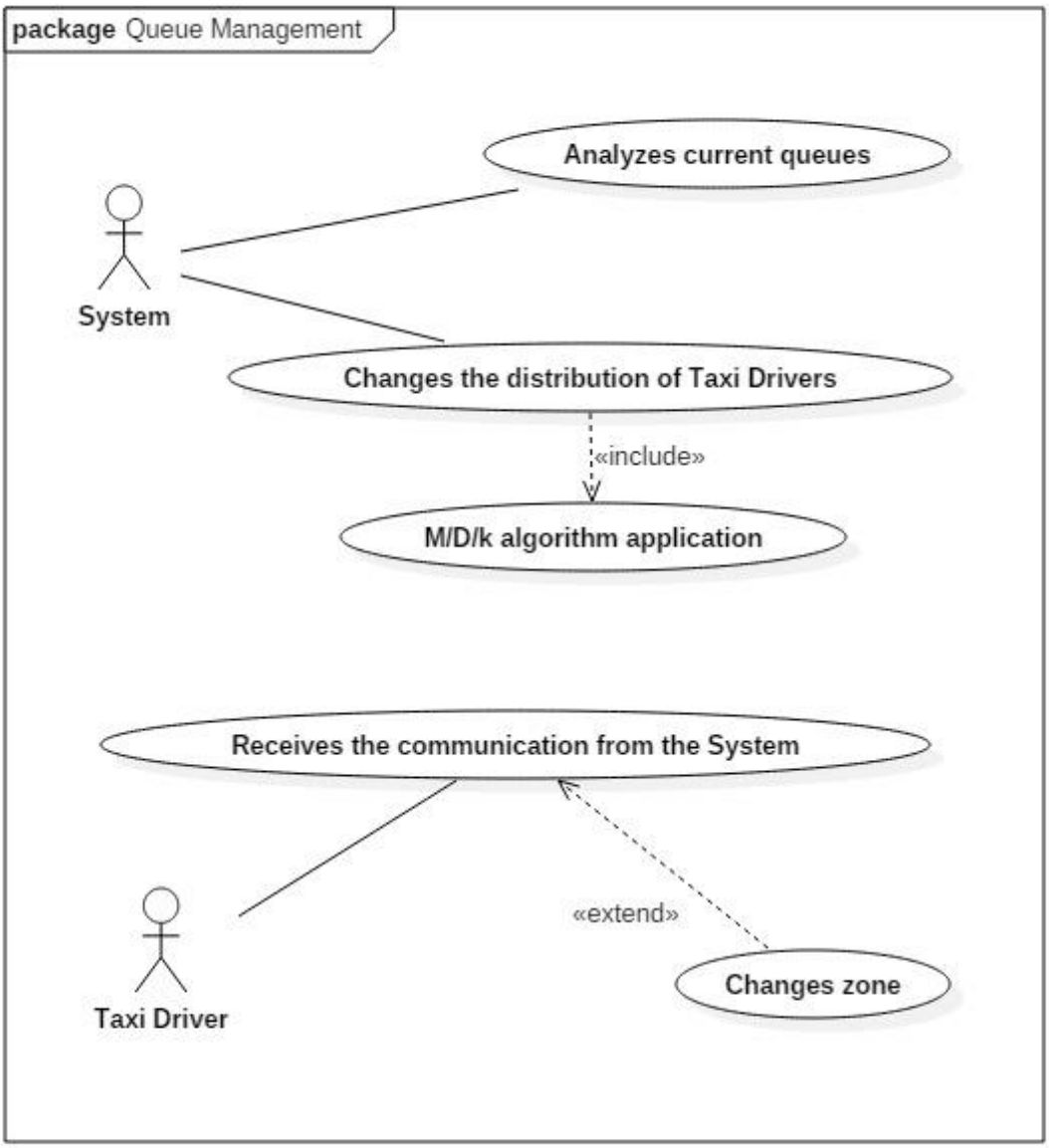
| | |
|---|---|
| Actors | Visitor, Costumer, System |
| Goals | [G1] |
| Input conditions | • The Visitor has a Costumer Account, but he/she is not logged yet. |
| Flow of events | 1. The Visitor opens the web application or the mobile app<br>2. The Visitor inserts his account's data (Username and Password)<br>3. The Visitor clicks on the Login button<br>4. The System checks if exists an account with the inserted credentials |
| Output conditions | • The Visitor is formally recognized as Costumer from *myTaxiService* System. |
| Exceptions | • Unregistered User<br>• Wrong Access Credentials<br>• System Down |

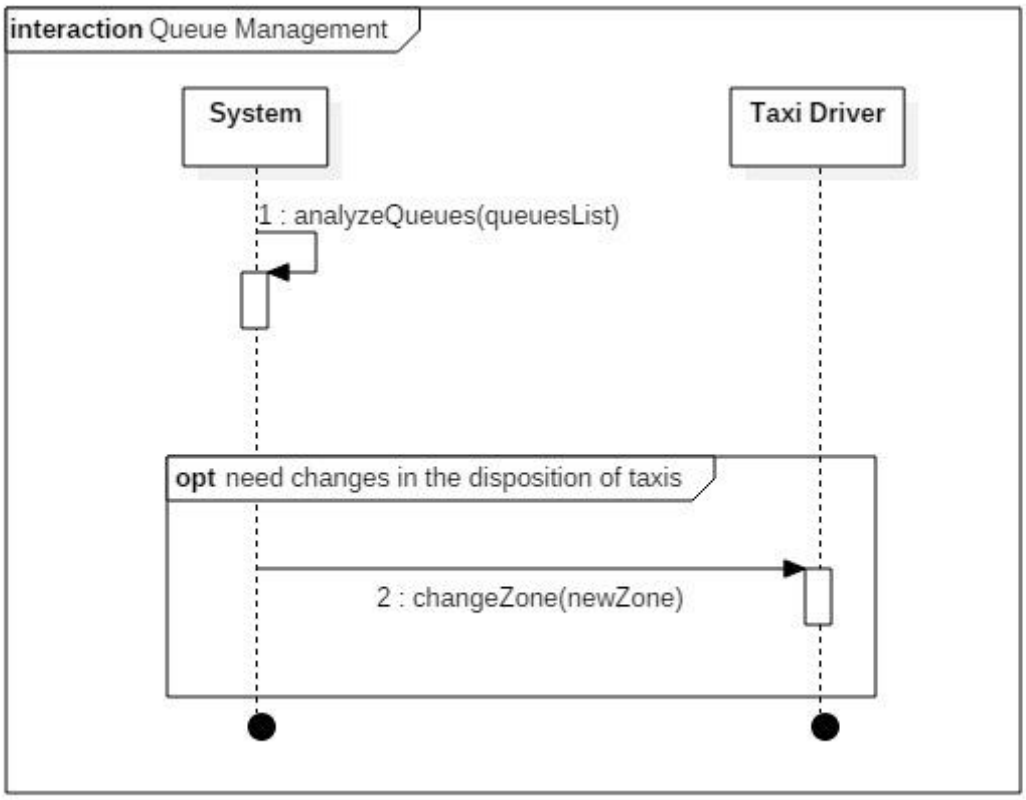

package Visitor logs-in
Visitor
Opens the web app / mobile app
Inserts credentials
Clicks login button
Validation
«include»
Customer Recognized
«extend»
Checks Visitor's credentials
System


interaction Visitor logs-in
Visitor
Costumer
System
1 : insertCredentials()
2 : loginButtonClicked()
alt
[username and password correct]
3 : loginOk()
[else]
4 : loginNotOk()

### 4.1.9. Queue management

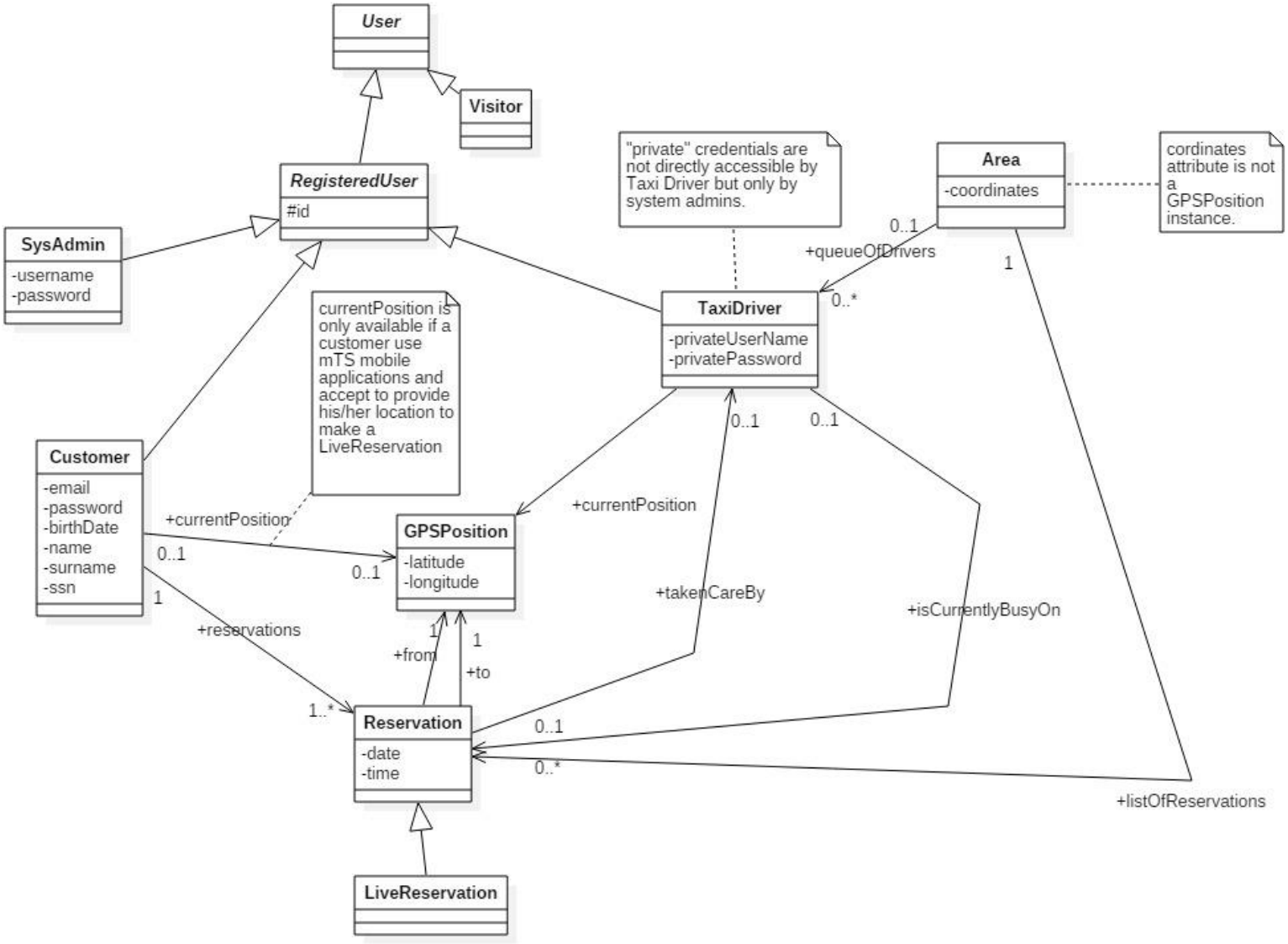
| | |
|---|---|
| Actors | System, Taxi Driver |
| Goals | [G6], [G7] |
| Input conditions | • The current situation of different queues (in different zones). |
| Flow of events | 1. The System periodically analyzes the situation of queues<br>2. According to the M/D/k model, the System changes the distribution of Taxi Drivers in the different areas of the city<br>3. Taxi Drivers receive the communication of change zone or continue to stay in the same zone |
| Output conditions | • An optimal distribution according to the probabilistic M/D/k model of queue management. |
| Exceptions | • System down<br>• No Taxi Available<br>• Queue Overflow |


package Queue Management
Analyzes current queues
System
Changes the distribution of Taxi Drivers
«include»
M/D/k algorithm application
Receives the communication from the System
«extend»
Changes zone
Taxi Driver


interaction Queue Management
System
Taxi Driver
1 : analyzeQueues(queuesList)
opt need changes in the disposition of taxis
2 : changeZone(newZone)

## 4.2. Class Diagram

This is the **Class Diagram** of our *myTaxiService* model. All the **UML Diagrams** are developed with this *Class Diagram* as main reference:



# 5. Appendix

## 5.1. Alloy

### 5.1.1. What is Alloy?

**alloy**: a language & tool for relational models

**Alloy** is a declarative specification language for expressing complex structural constraints and behavior in a software system. It is a very powerful support in the *Software Engineering* field.

**Alloy** provides a simple structural modeling tool based on *first-order predicate calculus* (*logic*).

***Alloy*** is targeted at the creation of micro-models (finite, of course) that can then be automatically checked for correctness in a formal way.

**Alloy** specifications can be checked using the **Alloy Analyzer**, which works with reduction to *SAT* (*Propositional Satisfiability Problem*).

We will use **Alloy** to check the correctness and the consistency of *myTaxiService* model.

### 5.1.2. Data types and abstract entities

These are the definitions of **data types** and **abstract data entities** of *myTaxiService*:

```
sig SSN {}
sig ID {}
sig GPSPosition {}


abstract sig User {}


abstract sig RegisteredUser extends User {
	id: one ID
}


sig Visitor extends User {}


sig Customer extends RegisteredUser{
```

```
        email: one Email,
        ssn:  one SSN,
        reservations: set Reservation,
        currentPosition: lone GPSPosition

}


sig TaxiDriver extends RegisteredUser{


        currentPosition: lone GPSPosition,
        isCurrentlyBusyOn: lone Reservation


}


sig SysAdmin extends RegisteredUser {}


sig Reservation {


        to: one GPSPosition,
        from: one GPSPosition,
        takenCareBy: lone TaxiDriver

}


sig LiveReservation extends Reservation {}


sig Area {

        queueOfDrivers: set TaxiDriver,
        listOfReservation: set Reservation,
        xPosition: Int,
        yPosition: Int

}
```

### 5.1.3. Facts

These are the **Facts** of *myTaxiService* model:

```
fact customerProperties {

        //Cannot exist two different users with the same ssn or the same email address
        all s:SSN | one cus:Customer | cus.ssn = s
        all e:Email | one cus:Customer | cus.email = e
        //a customer can have at most one livereservation in his/her reservations
        all cus:Customer | lone reg:LiveReservation | reg in cus.reservations
```

```
	//system uses customer positions iff a livereservation is saved in customer
reservations
	all cus:Customer | no gps:GPSPosition | cus.currentPosition = gps && no
res:LiveReservation | res in cus.reservations

}



fact registeredUserProperties {

	//id value is used by the system to identify every different registered user. two
different users with the same id are not allowed
	all i:ID | one user:RegisteredUser | user.id = i


}


fact areaProperties {


	//every area must have more than one driver and less than 6 drivers
	all area:Area | #area.queueOfDrivers>=2 && #area.queueOfDrivers<=5
	//area coordinates must be positive numbers
	&& area.xPosition>0 && area.yPosition>0
	//every area has different set of coordinates
	all disj area1, area2 : Area | area1.xPosition!=area2.xPosition or
area1.yPosition!=area2.yPosition


}


fact reservationProperties {


	//Every reservation is associated to one and only one Customer
	all res:Reservation | one cust:Customer | res in cust.reservations
	//Is not possible to have a reservation with starting point equal to destination point.
	//It's meaningless to have a reservation with no movement at all
	all res:Reservation | not (res.from = res.to)
	//Every reservation is in one and only one list
	all res: Reservation | one area: Area | res in area.listOfReservation


}


fact taxiDriverProperties {
```

```
	//if a taxi driver is busy on a reservation then this reservation is taken care by this driver
	all drv:TaxiDriver | all res:Reservation | (drv.isCurrentlyBusyOn=res implies res.takenCareBy=drv)
	//a taxi driver is in a list when he/she is working. gps position must be saved only when the driver is working.
	all drv:TaxiDriver | (one gps:GPSPosition | drv.currentPosition=gps) iff (one area:Area | drv in area.queueOfDrivers)
	//every taxi driver is in at most one area at time
	all drv: TaxiDriver | lone area1: Area | drv in area1.queueOfDrivers


}



fact gpsPositionProperties {

	//No gps position in the system not associated to any entity that requires a gps position
	no gps:GPSPosition | (no drv:TaxiDriver | drv.currentPosition=gps) and (no res:Reservation | res.from=gps) and (no res:Reservation | res.to = gps)


}

fact liveReservationProperties {


	//if a user has a live reservation in his/her list then is current position is equal to from field of live reservation
	all res:LiveReservation | all cus:Customer | res in cus.reservations implies res.from=cus.currentPosition


}
```

### 5.1.4. Assertions

These are the **Assertions** of *myTaxiService* model:

```
assert noCustomersWithSameSSN {

	all ssn1, ssn2: SSN | all disj cus1, cus2: Customer | cus1.ssn=ssn1 && cus2.ssn=ssn2 implies ssn1!=ssn2


}
```

```
check noCustomersWithSameSSN for 10


assert noCustomersWithSameEmail {

	all email1, email2: Email | all disj cus1, cus2: Customer | cus1.email=email1 &&
cus2.email=email2 implies email1!=email2


}


check noCustomersWithSameEmail for 10


assert noRegisteredUserWithSameID {


	all id1, id2: ID | all disj reg1, reg2: RegisteredUser | reg1.id = id1 && reg2.id = id2
implies id1!=id2


}


check noRegisteredUserWithSameID for 10


assert disjointedAreas {

	all area1, area2 : Area| all disj x1, x2: Int | all disj y1,y2: Int |
area1.xPosition=x1 && area1.yPosition=y1 && area2.xPosition=x2 && area2.yPosition=y2 implies
(x1!=x2 or y1=y2)


}


check disjointedAreas for 10


assert driverInQueueAndGPS {

	all drv:TaxiDriver | ( one area:Area | drv in area.queueOfDrivers ) implies one
gps:GPSPosition | drv.currentPosition=gps


}

check driverInQueueAndGPS for 10
```

### 5.1.5. Predicates

In conclusion, these are the **Predicates** of myTaxiService model:

```
pred showComplex (drv:TaxiDriver, drv2:TaxiDriver) {


        no area:Area | drv in area.queueOfDrivers
        one res:Reservation | drv.isCurrentlyBusyOn=res
        one res:Reservation | drv.isCurrentlyBusyOn!=res && res.takenCareBy=drv
        #Area=2
        #Reservation=5
        #LiveReservation=2
        #Customer=2
        #TaxiDriver=5
        #GPSPosition>0
        #Visitor=2
        #SysAdmin=1


}


pred showSimple() {


        #Area=1
        #Reservation=2
        #LiveReservation=1
        #Customer=2
        #TaxiDriver=2
        #GPSPosition>0
        #Visitor=0
        #SysAdmin=0


}

run showSimple for 10


run showComplex for 10
```

### 5.1.6. Analyzer Results

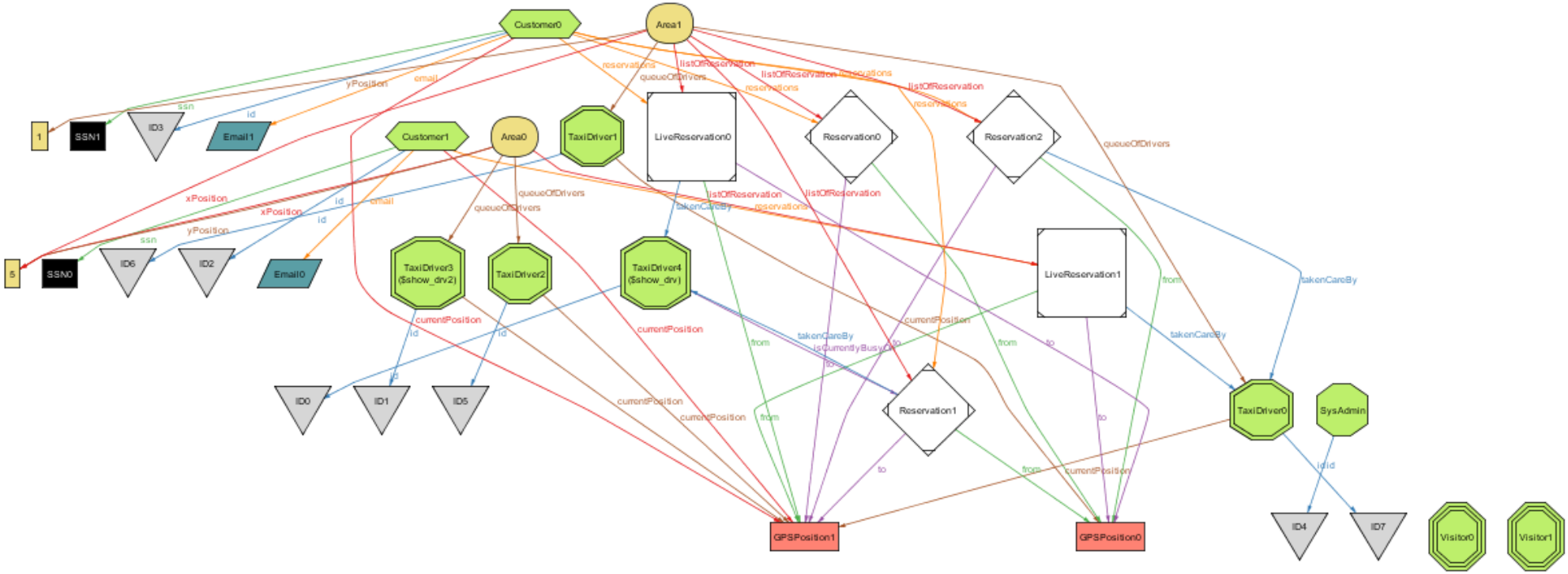
This is the screen of the **results** of **Alloy Analyzer**. We can see that the model is **consistent**:

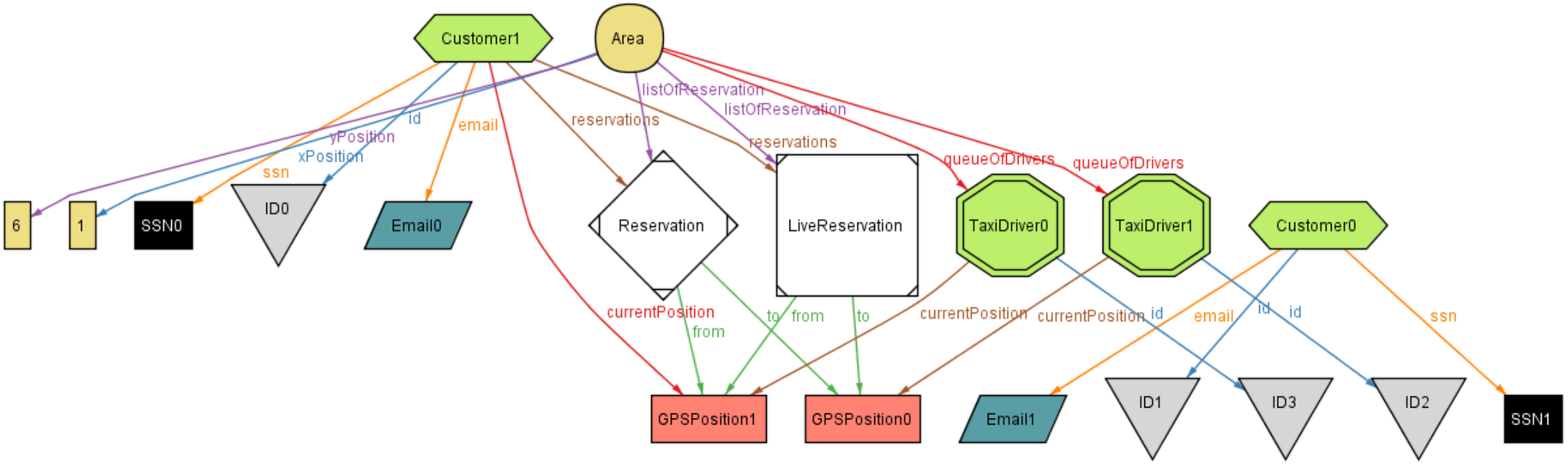
```
7 commands were executed. The results are:
  #1: No counterexample found. noCustomersWithSameSSN may be valid.
  #2: No counterexample found. noCustomersWithSameEmail may be valid.
  #3: No counterexample found. noRegisteredUserWithSameID may be valid.
  #4: No counterexample found. disjointedAreas may be valid.
  #5: No counterexample found. driverInQueueAndGPS may be valid.
  #6: Instance found. showSimple is consistent.
  #7: Instance found. showComplex is consistent.
```

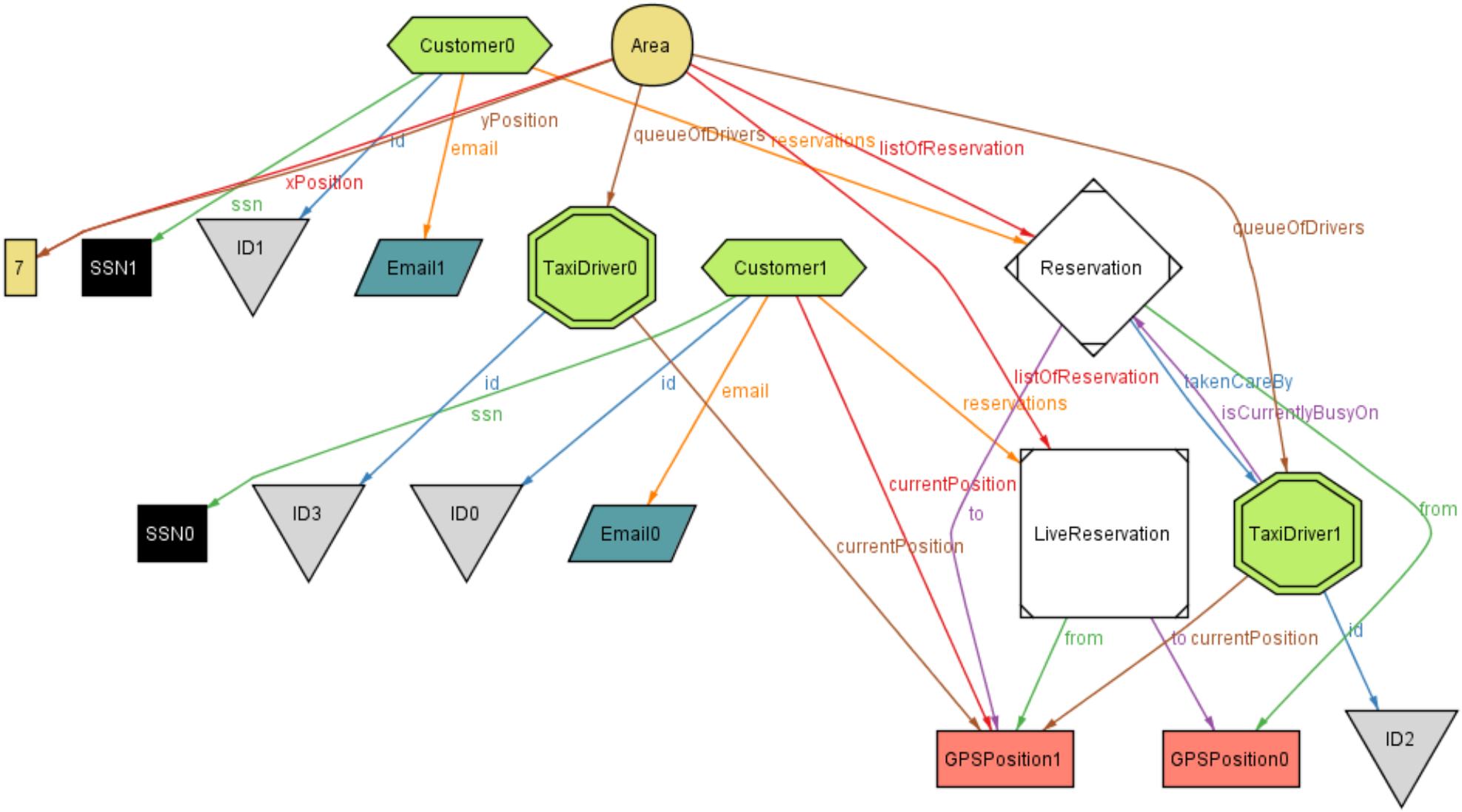
### 5.1.7. Population models

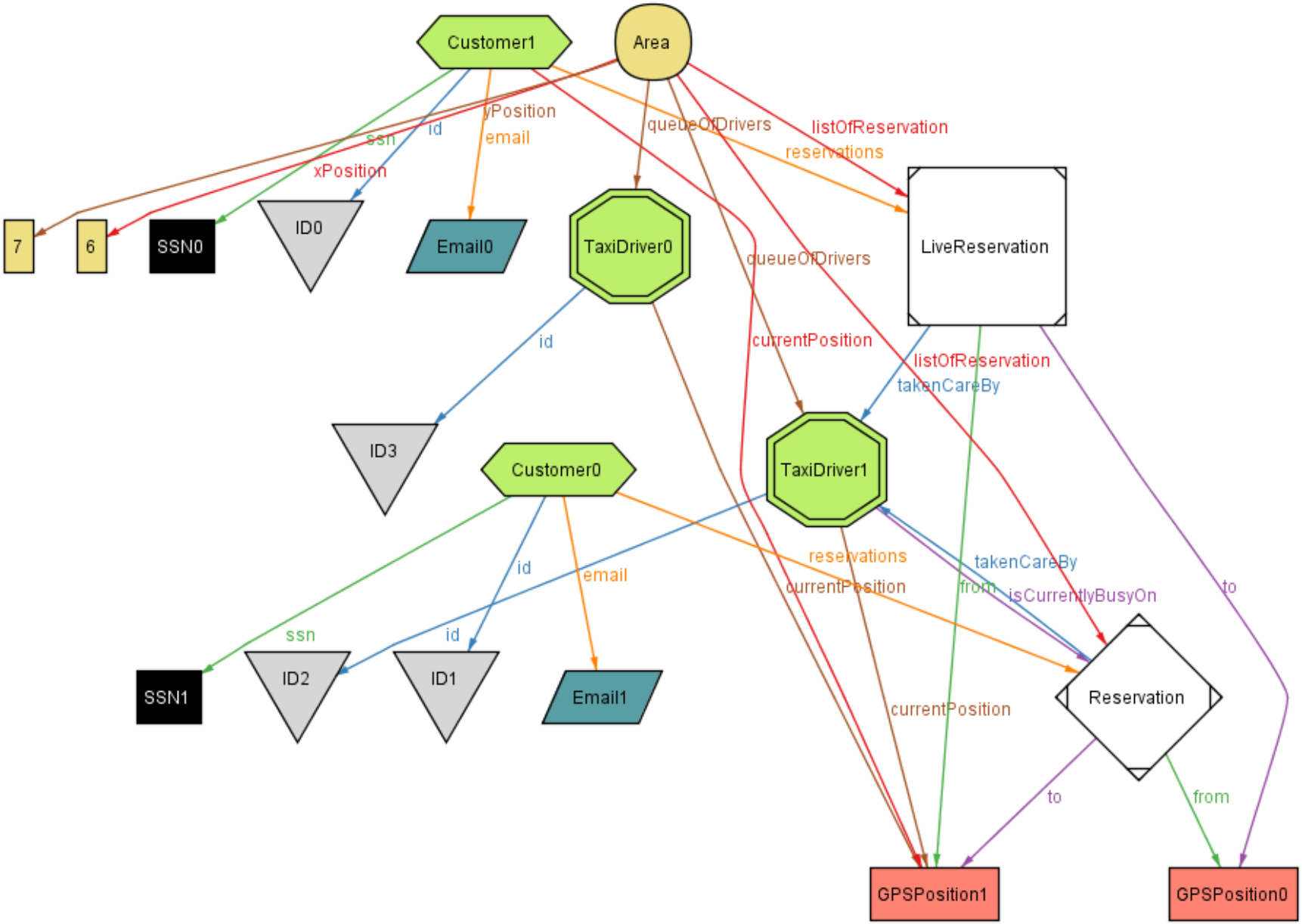
Here are some examples of **population models** obtained using **Alloy**. The generated models contain the key aspects of the project. In fact, there are entities and relations, with the suitable constraints.
Of course it is possible do generate other similar models using the *Alloy code*, but this is redundant information. Therefore, we included only some examples, as said before.

**Note:** *the source code is available in the RASD folder (GitHub repository).*


Customer0
Area1
1
SSN1
ID3
Email1
ssn
id
yPosition
email
reservations
queueOfDrivers
listOfReservation
LiveReservation0
Reservation0
Reservation2
Customer1
Area0
TaxiDriver1
xPosition
5
SSN0
ID6
ID2
Email0
TaxiDriver3 ($show_drv2)
TaxiDriver2
TaxiDriver4 ($show_drv)
takenCareBy
LiveReservation1
currentPosition
ID0
ID1
ID5
isCurrentlyBusy
Reservation1
from
to
TaxiDriver0
SysAdmin
GPSPosition1
GPSPosition0
ID4
ID7
Visitor0
Visitor1

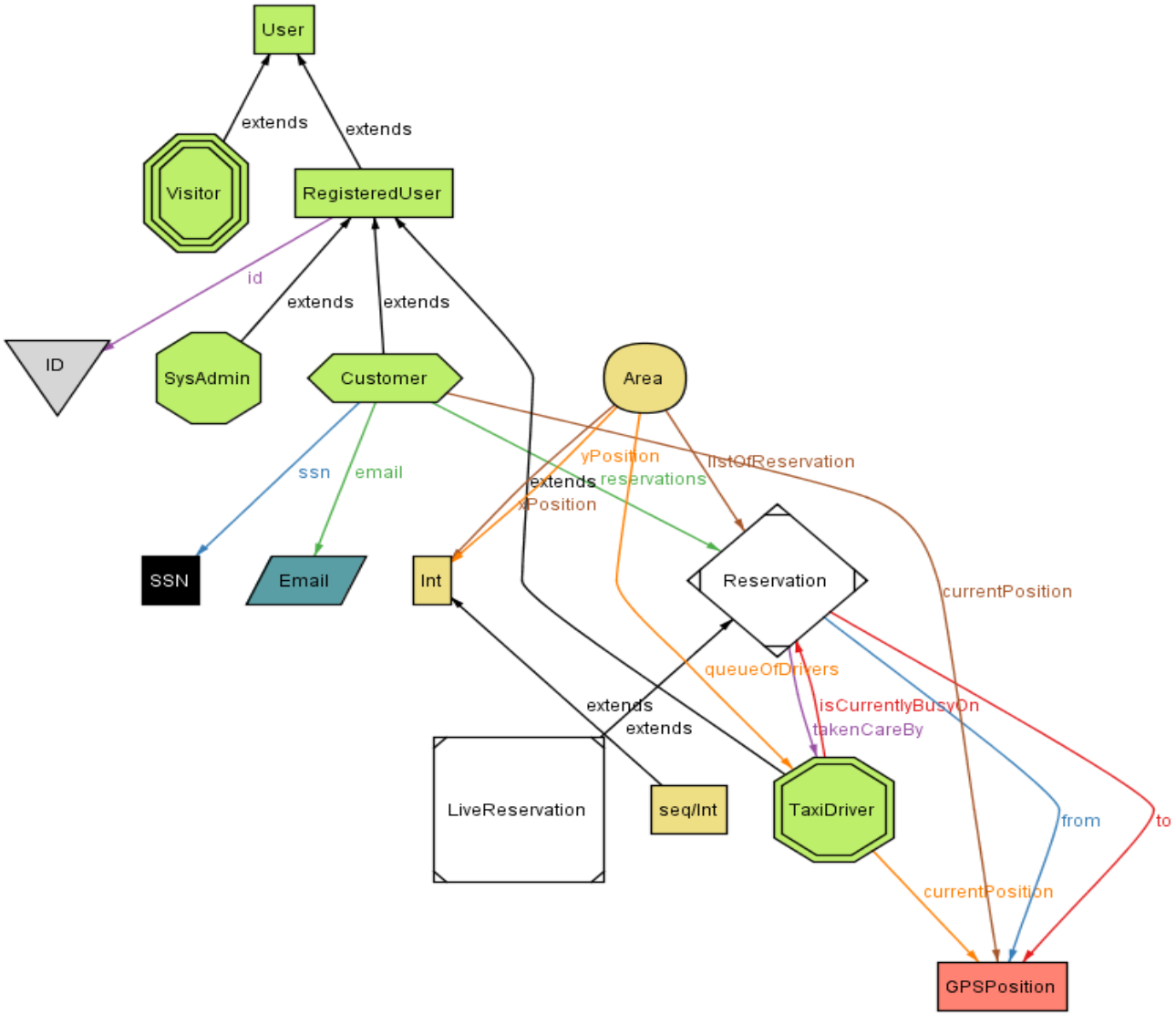

Customer1
Area
6
1
SSN0
ID0
Email0
Reservation
LiveReservation
TaxiDriver0
TaxiDriver1
Customer0
GPSPosition1
GPSPosition0
Email1
ID1
ID3
ID2
SSN1
yPosition
xPosition
id
ssn
email
reservations
listOfReservation
queueOfDrivers
currentPosition
to
from


Customer0
Area
yPosition
id
email
queueOfDrivers
reservations
listOfReservation
xPosition
ssn
queueOfDrivers
7
SSN1
ID1
Email1
TaxiDriver0
Customer1
Reservation
id
ssn
id
email
listOfReservation
reservations
takenCareBy
isCurrentlyBusyOn
SSN0
ID3
ID0
Email0
currentPosition
to
currentPosition
LiveReservation
TaxiDriver1
from
from
to currentPosition
id
GPSPosition1
GPSPosition0
ID2


Customer1
Area
ssn
id
yPosition
email
xPosition
queueOfDrivers
listOfReservation
reservations
7
6
SSN0
ID0
Email0
TaxiDriver0
LiveReservation
queueOfDrivers
id
currentPosition
listOfReservation
takenCareBy
ID3
Customer0
TaxiDriver1
reservations
currentPosition
takenCareBy
from
isCurrentlyBusyOn
to
ssn
id
email
id
SSN1
ID2
ID1
Email1
Reservation
currentPosition
to
from
GPSPosition1
GPSPosition0

### 5.1.8. Metamodel

We have also produced the **Metamodel**, using the **Alloy** *Metamodel function:*



## 5.2. Software and Tools Used

- **Microsoft Word** (https://products.office.com/it-it/word): redaction, formatting and revision of the RASD.
- **Alloy Analyzer** (http://alloy.mit.edu/alloy/): to verify in a formal way the consistency of *myTaxiService* model.
- **StarUML** (http://staruml.io/): to create and develop incrementally the UML models of the project (*UseCase diagrams, Sequence diagrams, Class diagram*).
- **Gimp** (http://www.gimp.org/): graphic elaborations of the project.
- **Balsamiq** (https://balsamiq.com/products/mockups/): mockups creation.
- **GitHub** (https://github.com/): to develop the project using a distributed repository.
- **Dropbox** (https://www.dropbox.com/): to share other contents step to step.

## 5.3. Hours of work

- **Andrea Martino**: ~33 Hours
- **Francesco Marchesani**: ~33 Hours

## 5.4. Revision of the document

This is the second version of the RASD (**RASD 2.0**).
We changed some **assumptions** in the paragraph *2.4 Assumptions and Dependencies*.
We highlighted the changes with the blue color.