

Politecnico di Milano

Requirements Analysis and Specifications Document

"myTaxiService"

Nicolas Tagliabue(matr. 853097), Matteo Pagliari(matr. 854353)

November 6, 2015

Contents

1	Introduction	4
1.1	Purpose	4
1.2	Actual System	4
1.3	Scope	4
1.4	Actors	4
1.5	Goals	4
1.6	Definitions, Acronyms, Abbreviations	5
1.6.1	Definitions	5
2	Overall Description	5
2.1	Users	5
2.2	Constraints	6
2.3	Assumptions	6
2.4	Future implementations	7
2.5	Documents related	7
3	Specific Requirements	7
3.1	External Interface Requirements	7
3.1.1	User Interfaces	7
3.1.2	API Interfaces	12
3.1.3	Hardware Interfaces	12
3.1.4	Software Interfaces	13
3.2	Functional Requirements	13
3.3	Scenarios	14
3.3.1	Scenario 1	14
3.3.2	Scenario 2	14
3.3.3	Scenario 3	14
3.3.4	Scenario 4	15
3.4	UML Models	15
3.4.1	Use Case Diagram	15
3.4.2	Use Case 1	16
3.4.3	Use Case 2	18
3.4.4	Use Case 3	19
3.4.5	Use Case 4	20
3.4.6	Use Case 5	21
3.4.7	Use Case 6	22
3.4.8	Use Case 7	23
3.4.9	Use Case 8	24
3.4.10	Use Case 9	25
3.4.11	Use Case 10	26
3.4.12	Use Case 11	27
3.5	Class Diagram	28
3.6	Non Functional Requirements	28
3.6.1	Performance Requirements	28

3.6.2	Design Constraints	28
3.6.3	Software System Attributes	28
3.6.4	Security	29
4	Alloy	31
4.1	Signature	31
4.2	Facts	32
4.3	Asserts	33
4.4	Predicates	34
4.5	Results	35
4.6	Generated World	37
5	Appendix	39
5.1	Software used	39
5.2	Hours of Work	39

1 Introduction

1.1 Purpose

This is the Requirement Analysis and Specification Document (RASD) in which are described the properties of the system we are going to develop by means of its general goals, functional and non functional requirements , constraints and real case scenarios. Needless to say that the document is intended for developers that are going to implement or integrate the system.

1.2 Actual System

This is a brand new system and we assume that there is no legacy system.

1.3 Scope

Our task is to develop a web and mobile application whose aim is to let people call and reserve a taxi in a easier and faster way than usual.

1.4 Actors

- Guest user: user that has not yet been registered in the system, therefore can only see the login page on the app/website and access the registration form.
- Registered user: this user is divided into two subclasses, the taxi driver and the customer. The taxi driver can inform the system about his/her availability and confirm whether or not he/she will respond to the call. The customer can call a taxi or reserve a taxi by specifying the origin and the destination of the ride.

1.5 Goals

These are the goals of myTaxiService mobile and web app:

- [G1] Registration of a person to the system
The system has to provide a sign up functionally.
The system has to provide two types of personal pages: one for the customer and one for the taxi driver.
- [G2] Request call
The system has to provide a function that makes possible to do a request.
- [G3] Reservation
The system has to provide a function that makes possible to do a reservation.
The system has to provide a function that allows the customer to modify his/her reservation until 1 hour before the meeting.

The system has to provide a function that allows the customer to cancel his/her reservation until 1 hour before the meeting.

- [G4] Taxi Management System
The system has to provide a function that allows the taxi driver to inform the system about his/her availability.
The system has to provide a function that allows the taxi driver to confirm or not the request/reservation of the customers.
The system has to provide a function that finds an available taxi 30 minutes before a reservation.
- [G5] Notification System
The system has to notify the customer who wants a taxi, about the found taxi.
The system has to notify the taxi driver when he/she has been assigned to a new customer.

1.6 Definitions, Acronyms, Abbreviations

1.6.1 Definitions

- Client/Customer: identifies a person who is going to take usage of the myTaxiService for requesting/reserving a taxi. These two words will be used as synonymous. Note that a client/customer must be a registered user.
- Reservation: when a customer reserves a taxi via the "Reserve a taxi" functionality in either the web or mobile application.
- Request: when a customer requests a taxi via the "Request a taxi" functionality in either the web or mobile application.

2 Overall Description

The application consists in a web and mobile application which is not integrated with an existing system. The application has two main different interfaces: one for the user and one for the taxi drivers. The application also offers some APIs for future projects, for example it provides an interface for a taxi sharing service.

2.1 Users

The users of our application are people who need to use a taxi and taxi drivers inside the operational area of the taxi service of the city in which the application has been implemented. The users must be able to use a web browser in order to use the webapp and be able to download the app from their respective mobile app store. All the users must have access to Internet.

2.2 Constraints

- Concurrency
myTaxiService must support concurrent operation from many different users in order to function correctly.
- Hardware limitations
myTaxiService has no hardware limitations.
- Interface with other applications
myTaxiService has no interface requirements with other applications.
- Policies
myTaxiService has no policies constraints.

2.3 Assumptions

- In the text is not specified if the taxi driver uses the same mobile app as the users, so we assume that taxi drivers have the same app but they are required to log in as a different type of user.
- Users requiring a taxi should really have the need to use the service, but if a force majeure event occurs the user can always cancel/modify the reservation 1 hour before the scheduled time.
- Informations given by the taxi drivers about their availability should be valid.
- When a taxi driver decides to answer a call, he/she has to go to his/her customer as soon as possible.
- When a user reserves a taxi should insert a valid position in the city for the origin and destination of his route.
- When a user requests a taxi using the mobile app or the webapp, the pick up location can be provided by the GPS or input manually.
- When a user reserves a taxi, the system automatically provide the first taxi in the queue, where the origin of the ride is located, 30 minutes before the meeting. In case the first taxi driver rejects the call, the system behaves like in the request service.
- The system confirms instantly the user when he/her sends a reservation and a taxi always provide the service.
- The city zones are fixed and well defined, so every address corresponds with only a zone.

2.4 Future implementations

- One major future expansion could be the implementation of a taxi sharing feature, that can allow different user with similar destinations to use the same taxi cab.
- The integration with Google Wallet and Apple Pay for an easy and fast payment method.
- Let the user choose from different taxi cabs available, for example an user can request a bigger cab or a different model.
- Expand the service to other locations.
- When a user requests a taxi using the mobile app or the webapp, the pick up location can be provided by the GPS or input manually.

2.5 Documents related

- Requirements and Analysis Specification Document.
- Design Document.

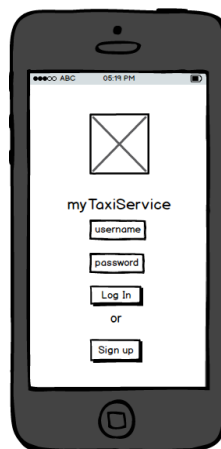
3 Specific Requirements

3.1 External Interface Requirements

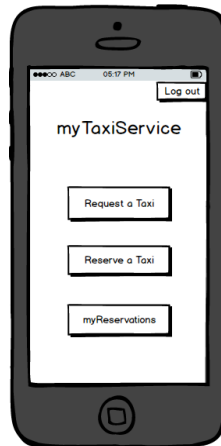
3.1.1 User Interfaces

these are some mockups for the user interface of both the mobile and web app

Home



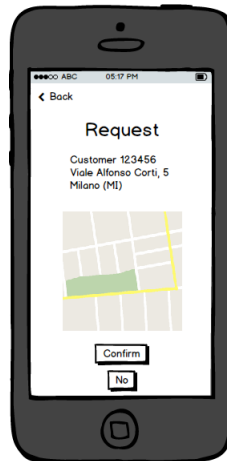
Home page Customer



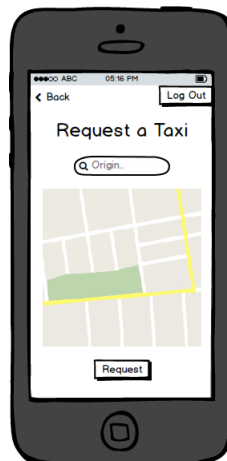
Home page Taxi Driver



Notification requested Taxi



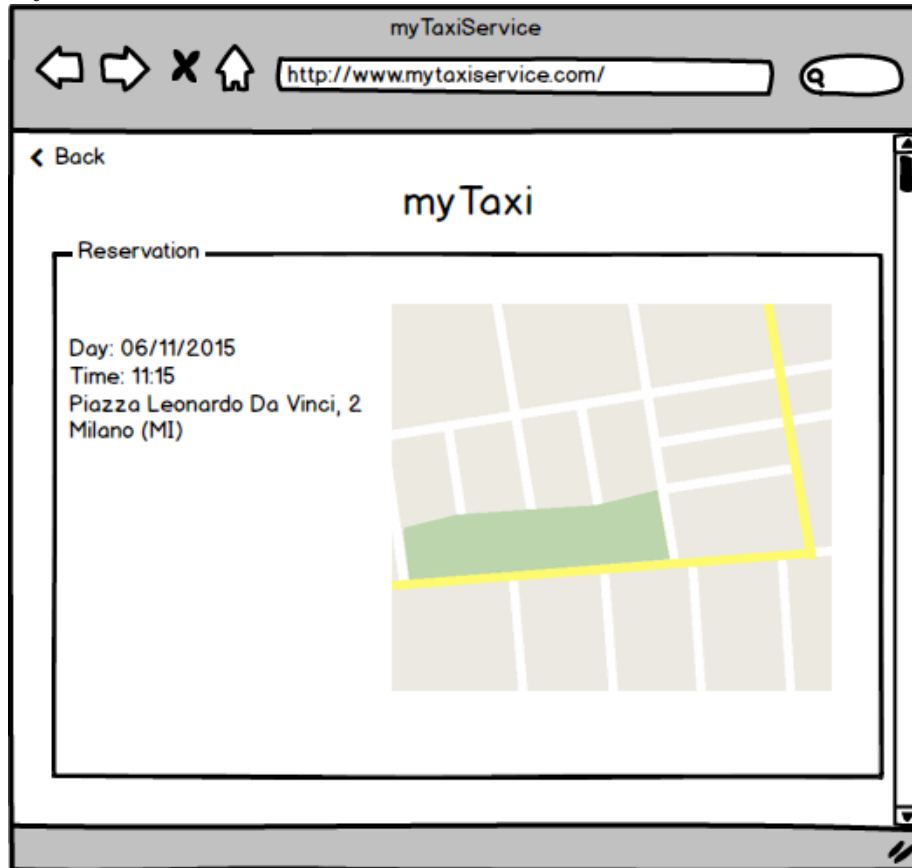
Request a Taxi



Reserve a Taxi



myReservations



Reserve a Taxi on the web app



3.1.2 API Interfaces

The application uses the Google Maps API to help the customers when they have to indicate the origin and the position of their routes. Also, Google Maps API helps taxi drivers find the locations of the trips. Google Maps API covers almost every streets on the Earth and it can be used in the future for possible new implementations like taxi sharing.

3.1.3 Hardware Interfaces

The application doesn't support hardware interfaces.

3.1.4 Software Interfaces

- Database Management System
 - Name: MySQL
- Java Virtual Machine
 - Name: JEE
- Operating System
 - Application must be able to run on any SO which supports JVM and DBMS specified before

3.2 Functional Requirements

- Guest:
 - Sign up
- Customer
 - Log in
 - Request
 - See request
 - Reservation
 - Modify reservation
 - Cancel reservation
 - See reservation
 - Receive a notification from the system about the found taxi
- Taxi Driver:
 - Log In
 - Inform the system about his/her availability
 - Confirm the system request/reservation call

3.3 Scenarios

3.3.1 Scenario 1

A is looking for a better way to call a taxi so, while searching the internet, he discovers a website called myTaxiService which has a mobile app too. In order to use the services offered by the newly found website, A has to register himself using the registration form on the homepage, choosing if he's a Taxi Driver or a Customer. A is just a potential Customer so he chooses the "Register as a Customer" functionality on the homepage of the site and fills in the registration form with all the required information like username, email and password. B instead is a Taxi Driver of city M, and he has been informed about the new taxi reservation system that has been adopted by the city government. In order to take advantage of it he has to connect to www.myTaxiService.it url with a web browser and choose "Register as a Taxi Driver". Then he will be prompted to complete the registration form with personal information and insert his taxi driver license number.

3.3.2 Scenario 2

It's 4 in the morning and C just finished attending a friend's birthday party. Instead of walking home he decides to call a cab, so he takes off his smartphone from the pocket and fires up the myTaxiService app. Luckily he has already registered, so he can simply log in and call for a taxi. The system receives the request and according to C's position provided by the app, forwards the request to the first taxi available near C which is driven by D. D receives a notification that a new client is available and as soon as he confirms the request he proceeds to reach C's position. After D has confirmed the request, the system sends C a notification containing D's taxi code number and an estimated waiting time.

3.3.3 Scenario 3

E has received a call from his boss asking him to take the next fly to Tokyo, which is scheduled for the next morning at 8 AM. E wants to be sure to reach the airport on time, so he goes to the website of myTaxiService, logs in and selects "Reserve a taxi". Then he just inserts starting location, destination and the desired time and date of meeting and hits the "Reserve" button. The system registers his request. 20 minutes later, E receives another call from his boss, now asking him to go to Moscow instead of Tokyo. Luckily, E has time to change his taxi request because 8 hours still remain to his previously scheduled meeting time with the taxi. Again he logs in to myTaxiService's website, accesses "My Reservation", selects his reservation and a new page opens up where he can modify the data. He then changes the meeting hour to 12 AM, according to the next fly available for Moscow, leaving origin and destination of the trip unchanged and then confirms everything by pressing the "Save" button. 30 minutes before the scheduled meeting date, the system sends a request for a new client to the first taxi available near the origin of the reserved trip, and

when the taxi driver confirms the request, the systems sends a notification to E confirming the reservation, providing also information about the taxi code number and estimated waiting time.

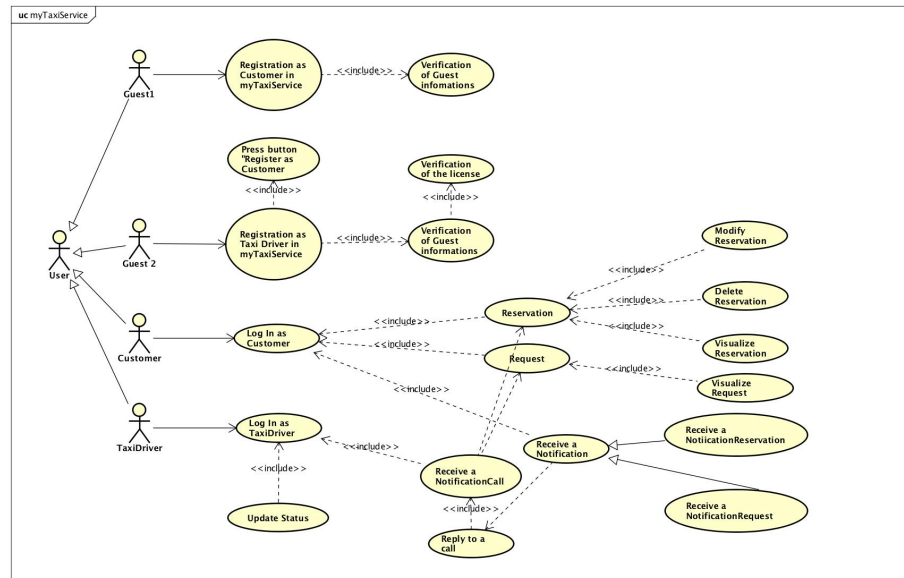
3.3.4 Scenario 4

F is a M taxi driver and uses myTaxiService for finding new customers. After a long day of work, and many many successful request answered he just wants to have a lunch break. In order to do so, he goes to the home of myTaxiService and presses the button "Not Available". The system updates his status on the DB, marking him as not available. Now he can enjoy his deserved break. When the break is over, he simply goes again to the home of myTaxiService, but this time he presses the button "Available" instead. The system once again updates his status on the DB, marking him as available, so he can get back to work.

3.4 UML Models

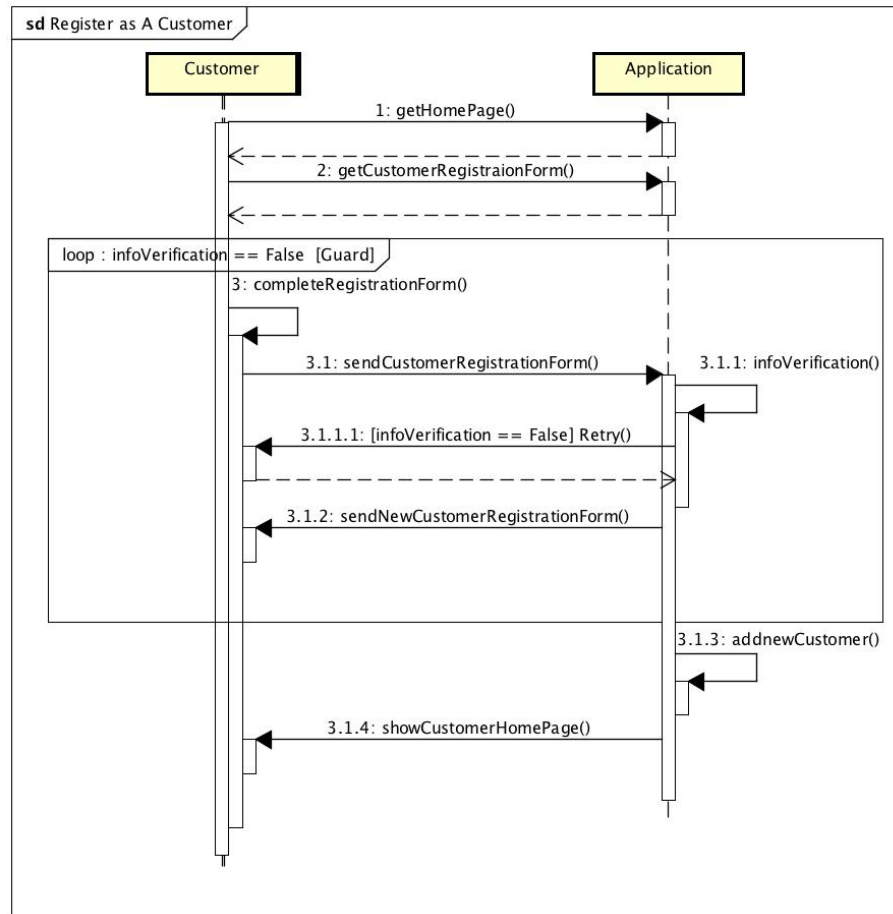
3.4.1 Use Case Diagram

This is the total Use Case Diagram of the application



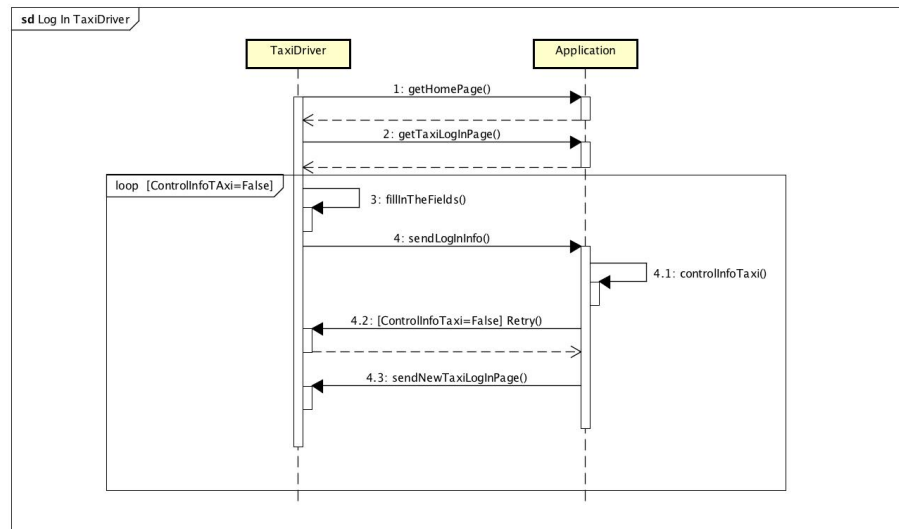
3.4.2 Use Case 1

Actor	Guest
Name	Sign up
Input condition	NULL
Event flow	<p>There are two cases:</p> <p>1- Customer Guest clicks on the home page on the button "Register as a Customer" to start the registration. Guest inserts his/her info in the registration form. Guest clicks on "Confirm" button. The system saves all the infos about the guest in the DB and shows his/her personal page.</p> <p>2- Taxi Driver Guest clicks on the home page on the button "Register as taxi driver" to start the registration</p>
Output condition	<p>1- Guest successfully complete the registration process and become a Customer and he/she can use the service.</p> <p>2- Guest successfully complete the registration process and become a Taxi driver and he/she can use the service.</p>
Exception	<p>Guest is already a registered user.</p> <p>Guest doesn't complete all the mandatory fields, in particular if he/she wants to register as a taxi driver he/she has to insert his/her license and it has to be correct.</p> <p>Username chosen is already used by another registered user.</p> <p>Email chosen is already used by another registered user.</p>



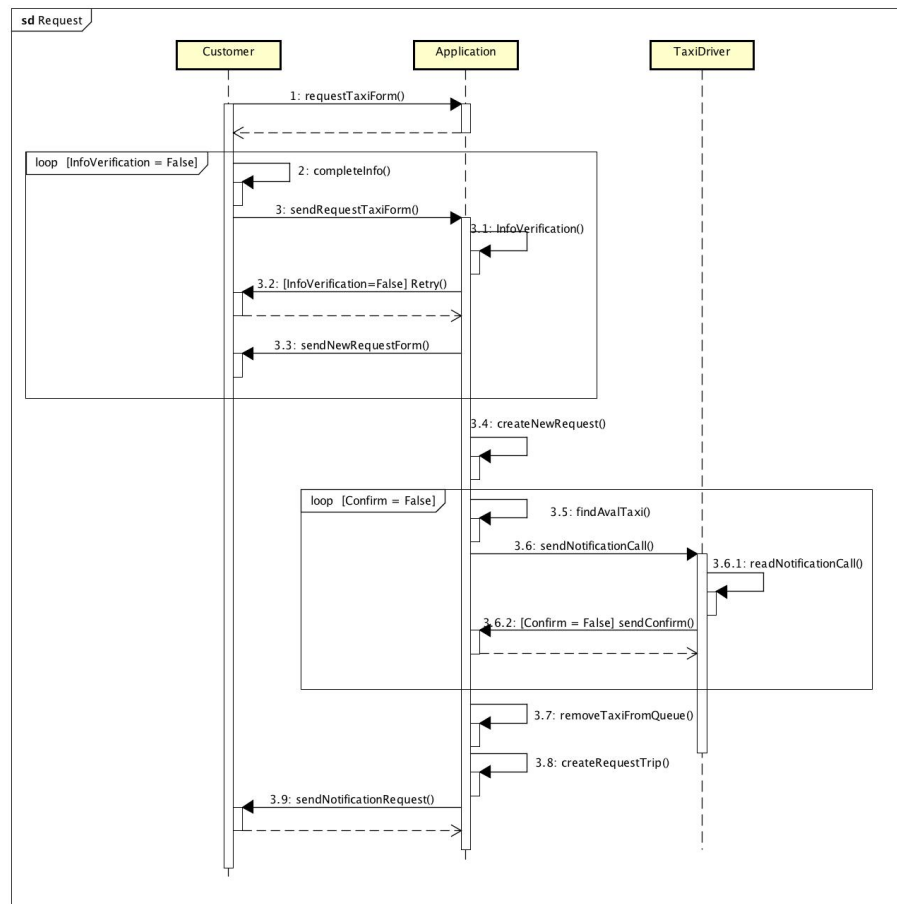
3.4.3 Use Case 2

Actor	Customer, Taxi Driver
Name	Log in
Input condition	Customer/Taxi Driver must be registered into the system.
Event flow	Customer/Taxi Driver enters in the website or in the webapp. Customer/Taxi Driver fills in the text field in the home page with the username and the password. Customer/Taxi Driver clicks on the button "Log in".
Output condition	The systems shows his/her personal page.
Exception	The username or the password inserted where wrong. The systems shows an error message to the Customer/Taxi Driver.



3.4.4 Use Case 3

Actor	Customer
Name	Request
Input condition	Customer must be logged into the system
Event flow	Customer clicks on the button "Request a taxi". Customer inserts his/her position in the correct fields. Customer clicks on the button "Request". The system contacts an available taxi driver for the request.
Output condition	The system shows a new page with the infos about the route.
Exception	Customer didn't indicate a valid position and the system shows an error message.



3.4.5 Use Case 4

Actor	Customer
Name	Reservation
Input condition	Customer must be logged in to the system
Event flow	Customer clicks on the button "Reserve a taxi". Customer inserts the informations about his route: origin destination date time Customer clicks on the button "Reserve".
Output condition	The system shows a new page with the infos about the route.
Exception	Customer didn't indicate a correct position of the origin/destination. Customer didn't insert a correct date. Customer didn't insert a correct time. In these cases the systems shows an Error message.

3.4.6 Use Case 5

Actor	Customer
Name	Modify reservation
Input condition	Customer must be logged into the system. Customer must have done a reservation.
Event flow	Customer clicks on the button "My Reservations". The systems shows all the reservations created. Customer selects the reservation that he/she wants to modify. The system shows him a page where he/she can modify the informations about the reservation. Customer clicks on the button "Save" to apply his/her changes.
Output condition	The system shows the modified reservation.
Exception	Customer didn't insert valid informations about the route. Customer violates the time constraints. In these cases the system shows him an Error message. Customer presses the button "Cancel" to abort the process and the system shows him the personal page.

3.4.7 Use Case 6

Actor	Customer
Name	Delete reservation
Input condition	Customer must be logged into the system. Customer must have done a reservation.
Event flow	Customer clicks on the button "My Reservations". The systems shows all the reservations created. Customer selects the reservation that he/she wants to cancel. The system shows the infos about the reservation. Customer clicks on the button "Delete" to remove the reservation.
Output condition	The system removes the reservation from the DB and shows to the customer the personal page.
Exception	Customer violates the time constraints. Customer presses the button "Cancel" to abort the process and the system shows the personal page.

3.4.8 Use Case 7

Actor	Customer
Name	See reservations
Input condition	Customer must be logged into the system.
Event flow	Customer clicks on the button "My Reservations".
Output condition	The system shows him a new page with a list of his/her reservations.
Exception	The system shows the message "Reservations can't be found" if there aren't reservations in his/her list.


3.4.9 Use Case 8

Actor	Taxi Driver
Name	Update status
Input condition	Taxi Driver must be logged into the system.
Event flow	Taxi Driver clicks on the home page on the button "Available" or "Not Available" to inform the system about his/her status.
Output condition	The system updates his/her status on the DB.
Exception	Taxi driver informs the system about his/her status but he/she doesn't change it. The system shows the message "Error update".

3.4.10 Use Case 9

Actor	Taxi Driver
Name	Confirmation
Input condition	Taxi Driver must be logged into the system. Taxi Driver must have received a request call.
Event flow	Taxi Driver clicks on the home page on the button "Request call". The system shows the informations about the request/reservation. Taxi Driver clicks the button "Confirm" to inform the system that he/she is going to take care of the call.
Output condition	The system shows the informations about the request/reservation.
Exception	Taxi Driver clicks on the button "No" and the system tries to contact another taxi driver. The system shows the personal page.

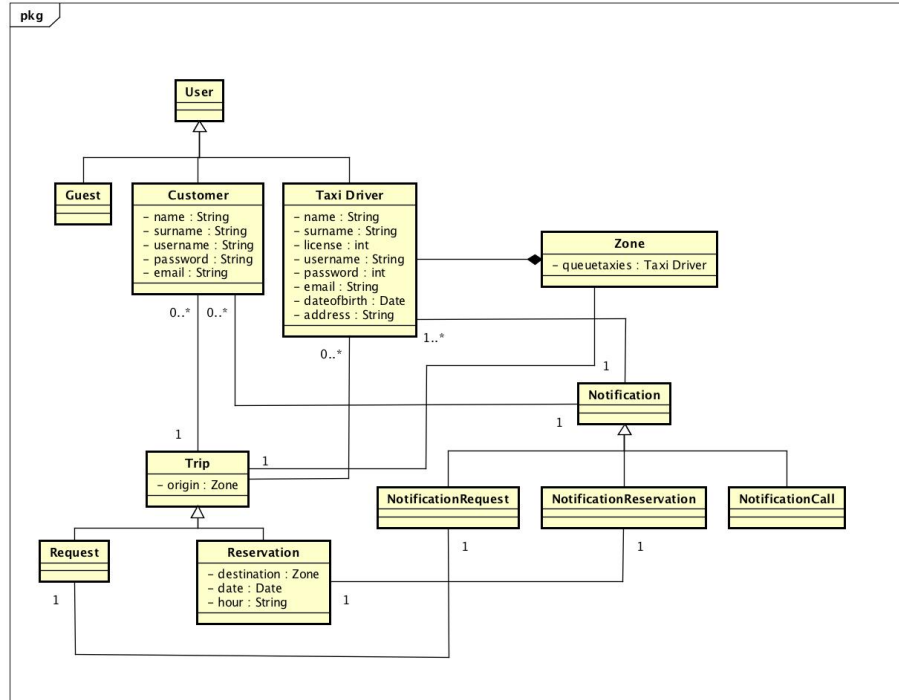
3.4.11 Use Case 10

Actor	Customer
Name	Taxi found Notification 
Input condition	Customer must be logged into the system. Customer must have done a request/reservation call. One taxi driver has to accept the request/reservation call.
Event flow	The system notifies the customer about the found taxi.
Output condition	The system shows the personal page.
Exception	No exception.

3.4.12 Use Case 11

Actor	Taxi Driver
Name	New Customer Notification
Input condition	Taxi Driver must be logged into the system. The system must have assigned a Customer to the Taxi Driver.
Event flow	The system notifies the Taxi Driver about the new Customer.
Output condition	The system shows the personal page.
Exception	No exception.

3.5 Class Diagram



3.6 Non Functional Requirements

3.6.1 Performance Requirements

Performance of the system should be fast enough to permit a great user experience for both the customers and the taxi drivers. It should be optimized also for low connection bandwidth.

3.6.2 Design Constraints

The application will be developed with Java EE so it will inherit all language's constraints.

3.6.3 Software System Attributes

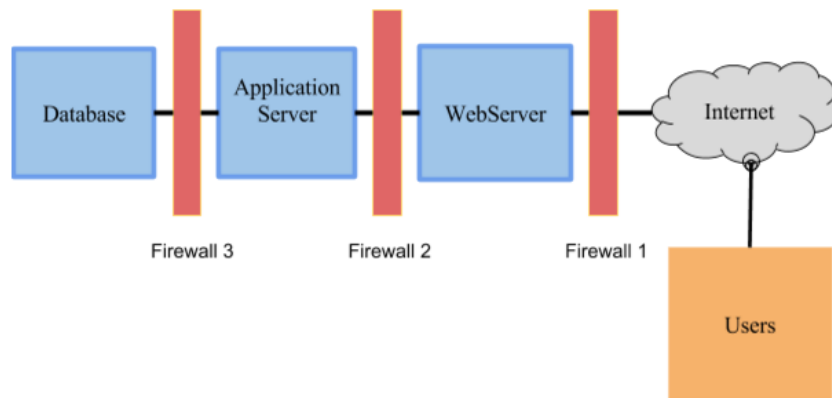
- **Availability:** The application has to be online 24/7 so it is necessary to have a dedicated server.
- **Maintainability:** The application offers API so they will have to be constantly updated and documented in order to respect the current development of the app.

- Portability: The application could be used on any SO which supports JVM and DBMS.

3.6.4 Security

Application Side Every connection should use the HTTPS protocol, this is because the possible future implementation with online payment methods.

Server Side Here is presented a schematic representation of the server architecture: the idea is to separate the raw data from the proper application server, in order to increase the security level.



4 Alloy

4.1 Signature

```
//-----Signatures-----//

abstract sig User{}

sig TaxiDriver extends User {
    license: one License,
    aval: one Availability
}

sig License{}

sig Customer extends User {}

abstract sig Trip {
    customer: one Customer,
    taxidriver: one TaxiDriver,
    zone: one Zone
}

sig Request extends Trip{}

sig Reservation extends Trip{
    destination: one Zone
}

abstract sig Notification {}

sig NotificationRequest extends Notification{
    request: one Request
}

sig NotificationReservation extends Notification {
    reservation: one Reservation
}

sig NotificationCall extends Notification{
    customer: one Customer,
    taxidriver: one TaxiDriver,
    zone: one Zone
}

sig Availability {
    aval: one Int
}{aval>=0 aval<=1}

sig Zone {
    queuetaxi: set TaxiDriver
}
```

4.2 Facts

```
//-----Facts-----//

//Each Request corresponds only to one NotificationRequest
fact OneNotReqPerRequest{
    all r:Request | (one nr:NotificationRequest | nr.request=r)
}

//Each Reservtion corresponds to only one NotificationReservation
fact OneNotiResPerReservation{
    all r:Reservation | (one nr:NotificationReservation | nr.reservation=r)
}

//There are only different Trips
fact OneTaxiDriverCustomePerTrip{
    no disj tp1,tp2: Trip | (tp1.customer=tp2.customer or
                           tp1.taxidriver=tp2.taxidriver )
}

//Each TaxiDriver has different license
fact DifferentLicense{
    all l:License | (one t:TaxiDriver | t.license = l)
}

//A TaxiDriver is not available iff he does a Trip
fact NotAvailableStatus{
    all t:TaxiDriver | (one tp:Trip | tp.taxidriver=t) <=> t.aval.aval=0
}

//A TaxiDriver is available iff he's in a queue of taxis
fact AvailableStatus{
    all t:TaxiDriver | (one z:Zone | t in z.queue taxi) <=> t.aval.aval=1
}

//Each TaxiDriver can be only in one queue
fact TaxiOnlyOneQueue{
    all t:TaxiDriver | (lone z:Zone | t in z.queue taxi)
}

//Only available TaxiDrivers can receive a NotificationCall
fact CallAvailTaxi{
    all t:TaxiDriver | (one c:NotificationCall | c.taxidriver=t) => t.aval.aval=1
}

//Only Customer that they aren't on a taxi can call one
fact CallCustomer{
    all c:Customer | (one nc:NotificationCall | nc.customer=c) => (all t:Trip | c!=t.customer)
}
```



```

//A Customer can do only one request/reservation at time
fact OneNotificationCallPerCustomer{
    all c:Customer | (!one n:NotificationCall | n.customer=c)
}

//Each NotificationCall corresponds to only one available TaxiDriver
fact OneNotCallPerTaxi{
    all t:TaxiDriver | (!one n:NotificationCall | n.taxidriver=t)
}

//A NotificationCall must be send to a TaxiDriver in the right Zone
fact NotificationCallInRightZone{
    all n:NotificationCall | (one z:Zone | n.taxidriver in z.queue taxi and n.zone=z)
}

```

4.3 Asserts

```

//-----Asserts-----//

//Check that a Customer can requests/reserves only one Taxi at time
assert OneRequestPerCustomer{
    no n1,n2:Request | (n1.customer=n2.customer and n1!=n2)
}

check OneRequestPerCustomer

//Check that each TaxiDriver can do at most one Trip
assert NoOneTaxiForDifferentTrips{
    no t1,t2:Trip | (t1.taxidriver=t2.taxidriver and t1!=t2)
}

check NoOneTaxiForDifferentTrips

//Check when a TaxiDriver confirms a call
assert RequestTrip{
    all t:Trip, z1,z2:Zone, t1,t2:TaxiDriver |
        ((t1.aval.aval=1 and t1 in z1.queue taxi and RequestTrip[t,z1,z2,t1,t2]) implies
            (t2.aval.aval=0) and t2 not in z2.queue taxi and t.taxidriver=t2)
}

check RequestTrip

//Check when a TaxiDriver became available after a Trip
assert AvailableTaxi{
    all t1,t2:TaxiDriver, z1,z2:Zone, t:Trip |
        (t1.aval.aval=0 and t.taxidriver=t1 and t1 not in z1.queue taxi and AvailableTaxi[t1,t2,z1,z2,t] implies
            (t2.aval.aval=1) and t.taxidriver!=t2 and t2 in z2.queue taxi)
}

check AvailableTaxi

```

4.4 Predicates

```
//-----Predicates-----//

//Show a world with a TaxiDriver that confirms a call
pred RequestTrip(t: Trip, z1, z2:Zone, t1, t2:TaxiDriver){
    t1.aval.aval=1 implies t2.aval.aval=0 and
    (t1 in z1.queue taxi implies t2 not in z2.queue taxi) and
    t.taxidriver=t2
    #Zone =2
    #Availability =2
}
run RequestTrip

//Show a world with a TaxiDriver that finishes his trip and he became available
pred AvailableTaxi(t1,t2:TaxiDriver, z1,z2: Zone, t:Trip){
    t1.aval.aval=0 implies t2.aval.aval=1 and (t1 not in z1.queue taxi implies t2 in z2.queue taxi)
    and (t.taxidriver=t1 implies t.taxidriver!=t2)
    #Availability=2
    #Zone=2
}
run AvailableTaxi

//Show a complete world
pred show1 {
    #TaxiDriver=7
    #Customer=6
    #Request=2
    #Reservation=2
    #Zone=4
    #Availability=2
    #NotificationCall=2
}
run show1 for 15

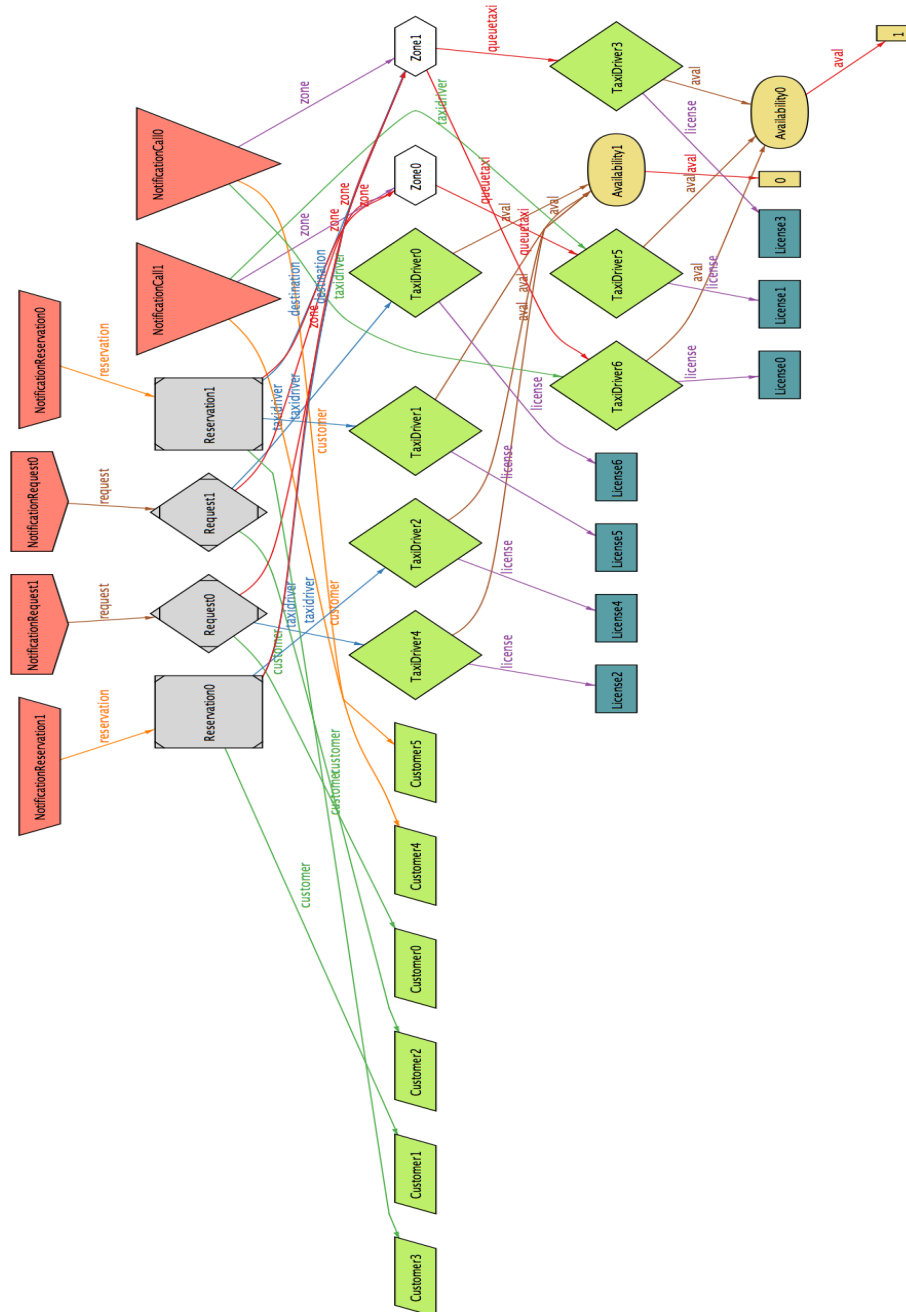
//Show another a complete world
pred show2{
    #Notification=3
    #Zone=2
    #Availability=2
}
run show2 for 8
```

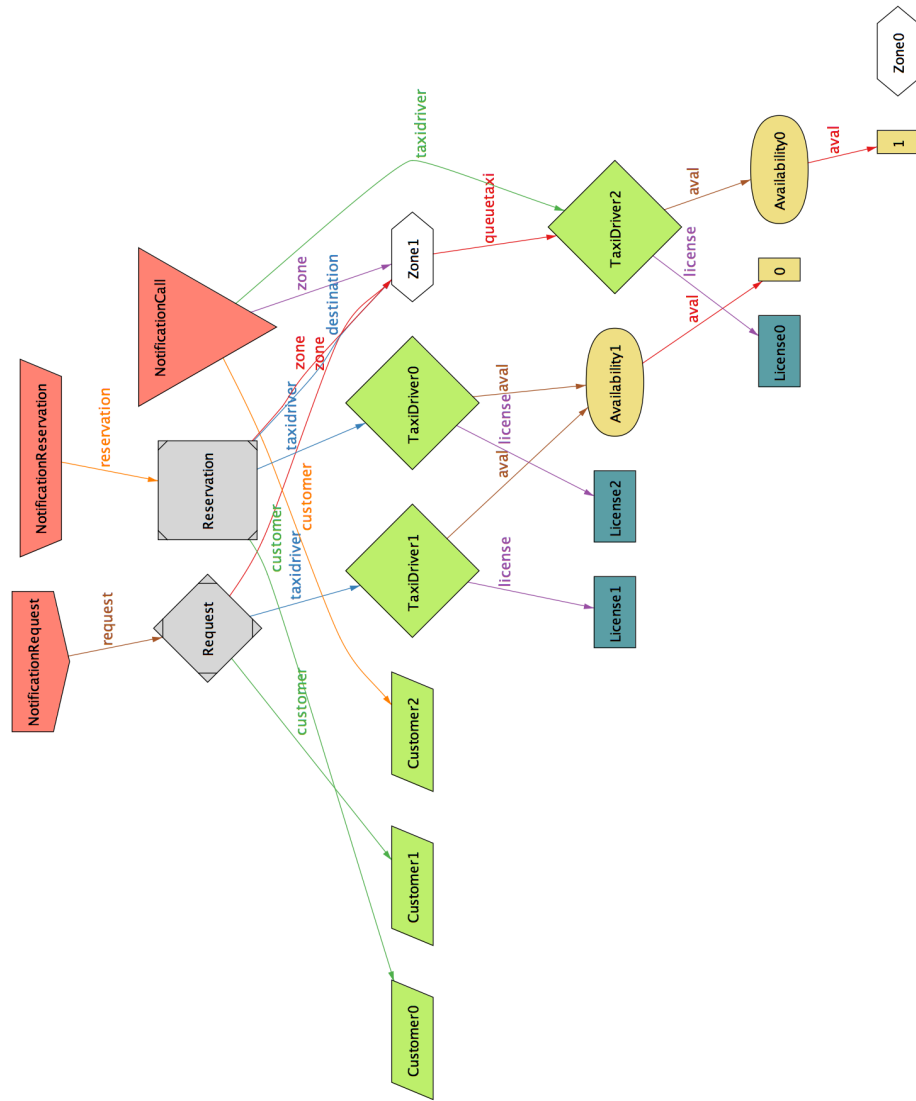
4.5 Results

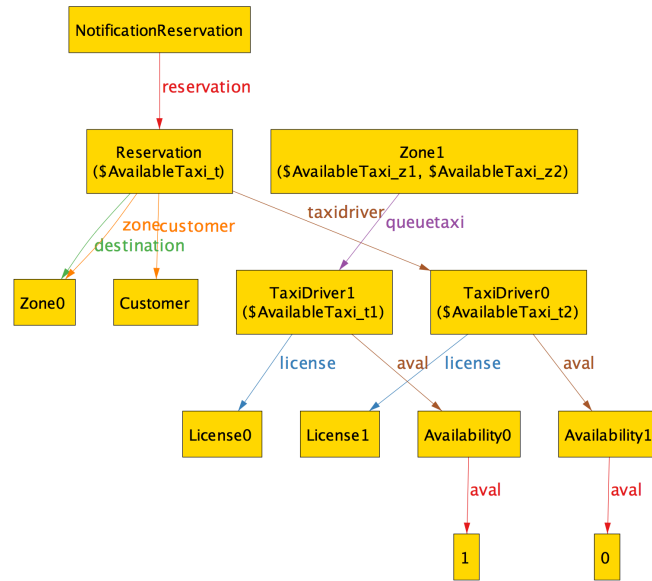
8 commands were executed. The results are:

- #1: No counterexample found. OneRequestPerCustomer may be valid.
- #2: No counterexample found. NoOneTaxiForDifferentTrips may be valid.
- #3: No counterexample found. RequestTrip may be valid.
- #4: No counterexample found. AvailableTaxi may be valid.
- #5: **Instance found.** RequestTrip is consistent.
- #6: **Instance found.** AvailableTaxi is consistent.
- #7: **Instance found.** show1 is consistent.
- #8: **Instance found.** show2 is consistent.

4.6 Generated World







5 Appendix

5.1 Software used

- Lyx
- Alloy Analyzer
- Astah
- Balsamiq Mockups

5.2 Hours of Work

We have spent approximately 30 hours each for preparing this document