POLITECNICO DI MILANO

POWERENJOY

SOFTWARE ENGINEERING 2

# Requirements Analysis and Specification Document

*Authors:*
Giancarlo COLACI
Giulio DE PASQUALE
Francesco RINALDI

*Supervisor:*
Elisabetta DE NITTO

January 22, 2017

# Contents

# 1 Introduction

## 1.1 Revision History

| Version | Date | Author(s) | Summary |
|---|---|---|---|
| 1.3 | 22/01/2017 | Giancarlo Colaci<br>Giulio De Pasquale<br>Francesco Rinaldi | Updated Alloy |
| 1.2 | 14/01/2017 | Giancarlo Colaci<br>Giulio De Pasquale<br>Francesco Rinaldi | Name refactoring, updated Class Diagram |
| 1.1 | 12/12/2016 | Giancarlo Colaci<br>Giulio De Pasquale<br>Francesco Rinaldi | Further details, new functional requirement |
| 1.0 | 14/11/2016 | Giancarlo Colaci<br>Giulio De Pasquale<br>Francesco Rinaldi | Initial Release |

## 1.2 Description of the given problem

We will project and implement PowerEnjoy, which is a new car-sharing service that exclusively employs electric cars. The system that will be developed has to allow the registration of a new user with all his personal information (like name, age, driving license and credit card information), log in credentials and the possibility to find the locations of available cars within a certain distance from his current location or from a specified address. Moreover, according to some policies (specified later) the user can obtain exclusive discounts and offers.

## 1.3 Purpose

The principal purpose of this document, the Requirement Analysis and Specification Document, is to show the functional and non-functional requirements of the system-to-be. They will be based on different aspects: the needs expressed by the stakeholders, the constraints which it is subject to, the typical scenarios that will happen after its deployment. The targeted audience in this case will be mainly made of software engineers and developers who have to actually develop the service here described.

## 1.4 Scope

The system will be an optimization of a pre-existing system for renting electric cars already in use in some cities. The new system will let users to check reservability and status of available cars, rent or reserve them through a mobile or a web application in a more simple and effective way. In addition to a better user interface, the new system will guarantee a smarter uniform distribution of cars in the city, in order to offer a better service for the citizens, and will also offer some special discount, in order to incentivize the virtuous behaviors of the users.

## 1.5 Goals

PowerEnJoy's users will be able to:

- register themselves to the system;
- log into the system;
- find the location of available cars in a specified area;
- book a car with the possibility to cancel the reservation;
- open his / her reserved car;
- be notified of active reservation status;
- end the rental;
- know the total cost at the end of the lease;

## 1.6 Stakeholders

Our primary stakeholder is the professor who gave us this didactical project. Our stakeholder's main need is to review the complete project at the end of the semester. Our objective is to show him/her we have followed the development process in all its parts and that we are able to carry out a challenging task from the beginning to the end fully comprehending all its internal phases. We want to show that we can identify key requirements and specifications, design and test our web/mobile application while providing all documentation backing up the source code. Finally, our typical users will be people that don't have a car or that prefer to rent a car instead of a public service transportation to go around the city.

## 1.7 Glossary

Below there are definitions of some terms that will be used in the document, in order to avoid any ambiguity in their use and their understanding.

**Guest:** a person that has never signed up to the system. He can only sign up and view available cars

**User:** a person already registered in the system who can log in and has a personal profile and can use all the functionalities described. Unless specified, each user is active: therefore can use the service with no restrictions

**Deactivated user:** a user with revoked privileges. He/she cannot use the service until the issues with his/her account are solved (e.g., expired license)

**Login:** the action of accessing the system via a username and related password

**Notification:** a real-time alert that warns a user when there are updates about what he is dealing with

**Reservation:** the action performed by the registered user when he chooses to drive an available car. It expires as soon as one of these conditions is met: one hour is elapsed from the user's reservation request or the car is unlocked

**Rent:** the temporary possession and use of one of the PowerEnjoy's cars in return for payment by a user. It starts once a minute is elapsed from the car doors unlock or the engine is ignited and lasts until the doors are re-locked by the system

**Available:** a fully functional car ready to be used by a registered user

**Unavailable:** a car which is currently reserved or used by a registered user or is not entirely working (e.g. exhausted battery, mechanical damage)

**Reservable:** an available car located in a geographical region where a registered user is enabled to reserve it

**Parking area:** circumscribed public area where cars can be parked (e.g. no parking lots). The set of safe areas for parking cars is pre-defined by the management system.

**Recharging area:** it is included in the parking area; cars parked here can also be recharged thanks to several charging stations

**Special area:** specific recharging areas decided by the system to ensure a uniform distribution of cars in the city

**RMSS:** Request Management Sub-System, it is an already developed part of our system which stores and manages all the information about the PowerEnjoy's cars, users and about their request (reservation or rent).

**MES:** Maintenance External Service, it is an external service that takes care of ordinary or extraordinary car maintenance.

**ADS:** Auto Diagnosis System, an always on embedded peripheral which continuously monitors the status of the car (e.g. battery charge, tires pressure, impact detection)

**ECS:** Emergency Call System, an always on embedded peripheral which can be used to call the consumer service or an emergency number quickly

**Verification code:** it is a four digits code chosen by the user during the registration procedure that adds another level of security to our service; the user will need it to unlock the car

**Countdown:** it is the time given to the user to pick-up the reserved car before the reservation expires (during this period the reservation is active)

**EULA:** End User License Agreement

# 2 Overall Description

## 2.1 Proposed System

The server-side implementation will be structured through a net of micro-services mainly written in Python and managed through Docker. The server will run the business logic, host PowerEnjoy's website, and users' data. Besides, we propose a web/mobile application which allows to registered users to use PowerEnjoy's services. The web client will use Python/Javascript and HTML5/CSS for web pages generation and formatting respectively. We will also use open-source libraries such as Flask to speed up the development. The mobile applications will be deployed on the main platforms currently available (iOS, Android, Windows Phone) and will be written accordingly to each devices' programming language. Client-server communication will be platform agnostic since it will happen through an encrypted RESTful API which returns its data through JSON.

## 2.2 Domain Properties

We suppose these conditions will be respected at any given time:

- the user will request help only if he/she needs it;
- the user will never try to exploit the system's features (e.g. sleep in the car);
- the user will never fake his/her position to cause a denial of service;
- the user will only provide correct and authentic information while signing up;
- a booked car will always be driven by the user who reserved it;
- the user will never be robbed of the access credentials and the verification code;
- the user agrees to be geolocalized;
- each available car is fully functional;
- each car has an embedded key to turn on the engine;
- each car is equipped with a properly working ADS and an ECS;
- each car is fitted with a properly working notification touchscreen display;
- an available car will always be found in the supposed position;

## 2.3 Assumptions

This section explores some of the vague concepts or loosely explained ones: further hypotheses have been added to expand the description of the requirements and to account better for the interaction between the external environment and the developed application.

Due to some unclear points in the specification we made some assumptions, which are:

- the system allows users to locate any car in PowerEnjoy's area of operation;
- the system's money saving option has to be enabled or disabled in the user profile;
- the system permits users to reserve an available car only in his/her city;
  - with "a certain geographical region" we mean that the user can reach the vehicle in a reasonable amount of time;
- the system can locate its users through Cellular Data or GPS;
- the system's proximity check is done through one of these: numerical code on each cars' windshield or GPS data from user's phone;
- the fees are applied as soon as one of these conditions is met: one minute is elapsed from the car's doors opening, or the engine is revved;
- each seat is equipped with a sensor which is used to detect the number of passengers into each car;
- each user can't reserve more than one car at the same time;
- to be eligible for the 10% discount, almost two passengers have to remain seated at the same time for at least one minute;
- if a car is parked in a special area, the discount equals to 40%;

## 2.4 Actors Identifying

The system provides the interaction of two different types of actors who can use different functionalities of the application system. The types are set out below along with a brief description of their privileges.

### 2.4.1 Guest

A person that is not registered (yet!) so can check for cars' position, register or ask for help/advice.

**Privileges:**

- Register to the system by creating a new account;
- Check available cars' position and status;
- Contact the customer service;

### 2.4.2 User

A person that is already registered to the system with his personal information; a guest can become a user after the authentication to the system using the login form.

**Privileges:**

- Log into the system;
- Consult reservations' history;
- Edit account information:
  - personal and billing data;
  - enable/disable money saving option;
- Check available cars' position and status;
- Reserve a reservable car;
- Rent the reserved car;
- Check active reservation status:
  - Remaining time until the reservation expires;
  - Reserved car's position and status;
  - Elapsed rental time;
- Request help through the ECS;
- Terminate the rent;
- Contact the customer service;

### 2.4.3 Deactivated User

A user with revoked privileges. He/she cannot use the service until the issues with his/her account are solved (e.g expired license).

**Privileges:**

- Log into the system;
- Consult reservations' history;
- Edit account information:
  - personal and billing data;
  - enable/disable money saving option;
- Check available cars' position and status;
- Contact the customer service;

## 3 Specific Requirements

In this section will be analyzed in detail functional and non-functional requirements that the system developed has to satisfy, when the domain properties previously denoted hold and referring to the declared goals.

### 3.1 Functional requirements

The functional requirements include the functionalities that the system must necessarily have and describe the interactions between the system developed and the external environment independently from the implementation.

1. Registration of a guest to the system:
   (a) The system has to guarantee the registration to a new user by creating a new account.
2. Login of a user into the system:
   (a) The system has to allow the login to an already registered user when he types the correct username and the password in the login form.
3. Find the location of available cars in a specified area:
   (a) The system has to allow both guests and users to specify the address where they want to locate an available car or to use their position;
   (b) The system has to guarantee that a car is showed on the map if and only if it is available (that means that every available car is showed and that every showed car is available);
   (c) The system has to guarantee that the car's position is regularly showed up to date.
4. Book a car with the possibility to cancel the reservation:
   (a) The system has to guarantee that only a user (i.e. not a guest) can reserve a car;
   (b) The system must be able to check the position of the user;
   (c) The system has to guarantee that any user can reserve only one car at a time;
   (d) The system has to ensure that the same car cannot be reserved at the same time by different users;
   (e) The system has to guarantee that any user can have at most an active reservation;
   (f) The system has to guarantee that every reservable car is available;
   (g) The system has to ensure that only a reservable car can be reserved;
   (h) The system has to guarantee that when a user reserves a car, the latter is no more available;
   (i) The system has to ensure that only an active reservation can expire or be canceled;
   (j) The system has to make sure that when a reservation expires or is revoked, the car becomes available again;
   (k) When a reservation is activated, the system starts a countdown at the end of which the reservation expires if the user did not pick-up the reserved car;

5. Open his/her reserved car;
    (a) The system must be able to check the proximity of the user to the reserved car;
    (b) The system must be able to check if the money saving option is enabled for the current reservation;
    (c) The system has to allow the user to insert the verification code;
    (d) The system has to unlock the reserved car only if the entered verification code is correct;
6. Be notified of active reservation status:
    (a) The system must be able to retrieve the starting time of the rental;
    (b) The system must be able to calculate and notify the elapsed time of the rental periodically;
    (c) The system must be able to communicate with every grid power station to check their power plugs' availability;
    (d) The system, to ensure a uniform distribution of parked cars, must be able to notify the user the nearest selected station where to leave the car to get a discount only if he/she inputs a destination and the money saving option is active;
7. End the rental:
    (a) The system has to allow the user to communicate that he/she wants to terminate the rental using the car's display;
    (b) The system has to allow the user to end the rental only if he is in a safe parking area;
    (c) The system has to guarantee that when a reservation is ended, the car becomes available again;
8. Know the total cost at the end of the rental:
    (a) The system has to communicate with the ADS to retrieve the car status and rental information;
    (b) The system must be able to apply the discounts, if any, handling the previously retrieved information;
    (c) The system must communicate to the user the final cost of the rental;
9. Modify the profile information:
    (a) The system has to allow the user to modify his/her personal information;
    (b) The system has to allow the user to enable or disable the money saving option;

## 3.2 Non-functional requirements

The non-functional requirements are those not related to the functionality, but rather consider the quality of the system to be implemented (Quality of Service, QoS), regardless of the application domain.

Security:

- Each user will only be able to access functionality that competes to his category, and it is, therefore, necessary to provide an authentication method: it will be used the symmetric username/password one. Furthermore, each credential will not be stored in cleartext, and client/server communications will be protected by asymmetric encryption.

Portability:

- The client must be compatible with all hardware and software platforms to reach as many users as possible. This requirement is satisfied by realizing the client as a web application, in this way the only assumptions that must be met by users are the availability of a web browser and a connection to the Internet.

Stability and reliability:

- The system must notify the user the result of each his/her transaction request while maintaining an optimal level of reliability. Therefore it will be able to face with the possible loss of connection between clients and the central server by ensuring atomicity of all operations.

Performance:

- As for performance, the timing of getting and insertion the information in the database must be acceptable to do not block the whole system.

Concurrency management:

- The system must ensure data consistency while dealing with multiple concurrent accesses to the same resource in the database by multiple users authenticated to the system.

Graphical User Interfaces (GUIs):

- The system has to include several graphical user interfaces: intuitive, complete and exceptionally easy to use. An interface to contain areas of input, dialog boxes, buttons, links and drop-down menus will be developed. The interfaces will interact with business logic and then be differentiated according to user roles in the system: each interface will provide access to features that compete for the particular user who is logged in.

# 4 Scenarios

This section will present some possible situations that may occur from the interaction between a user and the system developed.

## 4.1 Alice is curious about PowerEnjoy

Lily tells Alice about a new service called PowerEnjoy: Alice, now curious, tries to register to PowerEnjoy. She visits the homepage, browses the site a little and then proceeds to the registration page. Alice then enters all the data needed by the system to complete the process: name and surname, username, email address, password, ID card number and license number. To complete the sign-up she finally enters the billing information. Once everything is done, the system sends her a confirmation mail with a link used to activate her new account. She has now completed the registration, and she can log into the system and power enjoy the service.

## 4.2 Grandpa Peter and his expired license

Grandpa Peter has been registered to PowerEnjoy for a quite long time. He loves the car sharing model, now more than ever since his old Panda broke a long time ago and he could not pick up his nephews, David and Goliath, from the pool. There is a problem, though: Peter is getting old, and his license has to be renewed to drive again. After a long day at the Driver and Vehicle Licensing Agency, Grandpa Peter successfully got his shiny license back! To reactivate his PowerEnjoy account, Peter has to edit his account data and insert the new license card number: after having logged in, he proceeds to his personal area by clicking “Edit Personal Data", he puts the new data in the system, and finally he saves the changes. Grandpa Peter is on the road, again!

## 4.3 Grandpa Peter and his nephews

It’s rainy outside, so Grandpa Peter has to pick up his two nephews from the pool: he picks up his smartphone and checks if there are any available cars near him. He is getting old so he cannot walk for a long time, so he reserves the nearest car which, unfortunately, has less than 50% battery charge remaining. Once in the car’s reach, Peter taps on “Open Vehicle” and inputs his verification code. His memory is aging just like him so the usual “1969” code, which corresponds to the year he married his beloved Franca, turned into “1967” which is not accepted by the system. The system prompts him again, and this time Peter does not fail: the doors unlock, and Grandpa is on the road once more! Peter’s driving skills are still amazing: he picks up the kids and gets home in no time, so he ends the rent. Unfortunately, he is not able to use the 20% discount due to the battery charge left, but he will be cut 10% since he took at least two other people in the car.

### 4.4 Pablo and Tata go to the cinema

Tata and Pablo make a great couple. It’s Friday night, and the theaters are flooded by the latest sci-fi movie everyone love. Pablo loves all those lightsabers and starships while Tata just can’t get the point out of them. The film will start in more than one hour: Pablo is ready for the night out, Tata has to fix her makeup. “Five minutes and I am ready!" - she said. Unfortunately, Pablo forgot to charge his phone so he can’t use his phone’s GPS to save precious battery juice and reserves his PowerEnjoy car by entering his address into the app. Time goes by; one hour has passed, and Pablo is still waiting for Tata who seems to have no intentions leaving the bathroom. The film has just begun, the reservation expired, the car is available to be reserved again, and Pablo has to pay 1€ fee: “One minute and we are rolling!” - she said. Pablo, may the force be with you!

### 4.5 Gustavo, the discount hunter

Gustavo is a saver, methodic and old-fashioned man, and he built an economic empire from scratch. He owns a vintage 1968 Ford Torino which is only used on Saturday afternoons with his closest friend to honor the times long gone: on the other days, he mainly uses PowerEnjoy. Gustavo has turned on the money saving option while signing up, and he loves it. Once in the car, he logs in and proceed to set up his seat and insert his destination: the system now calculates the overall cars distribution in the city, verifies the availability of power plugs in the nearest power grid stations to the arrival and suggests to park in a special area just 300 meters away from his destination. Once there, Gustavo, just before ending the rental, takes care of plugging the car into the power grid: now, the smartphone application shows him the total cost which includes the 40% discount.

# 5 Models

In this section we are going to abstract from the previously seen scenarios in order to have a more high-level description. For this purpose we will use UML (Unified Modeling Language) diagrams.

## 5.1 Use cases model

From previously denoted scenarios and from the whole analysis we did in this document, we individuated the use cases of the system to be developed. In these pictures there are some use Cases diagram which represent actors, their use cases and their interactions. Then some of them are better explained using a less formal and more narrative way.

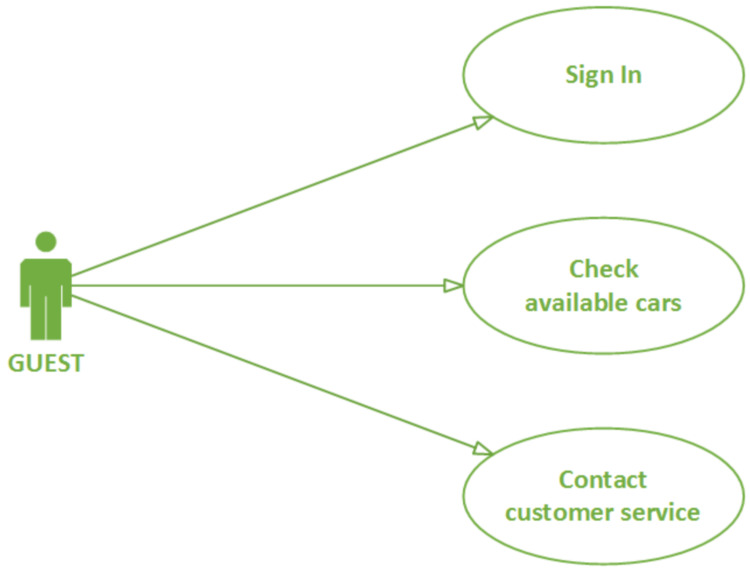


Figure 1: Guest Use Case

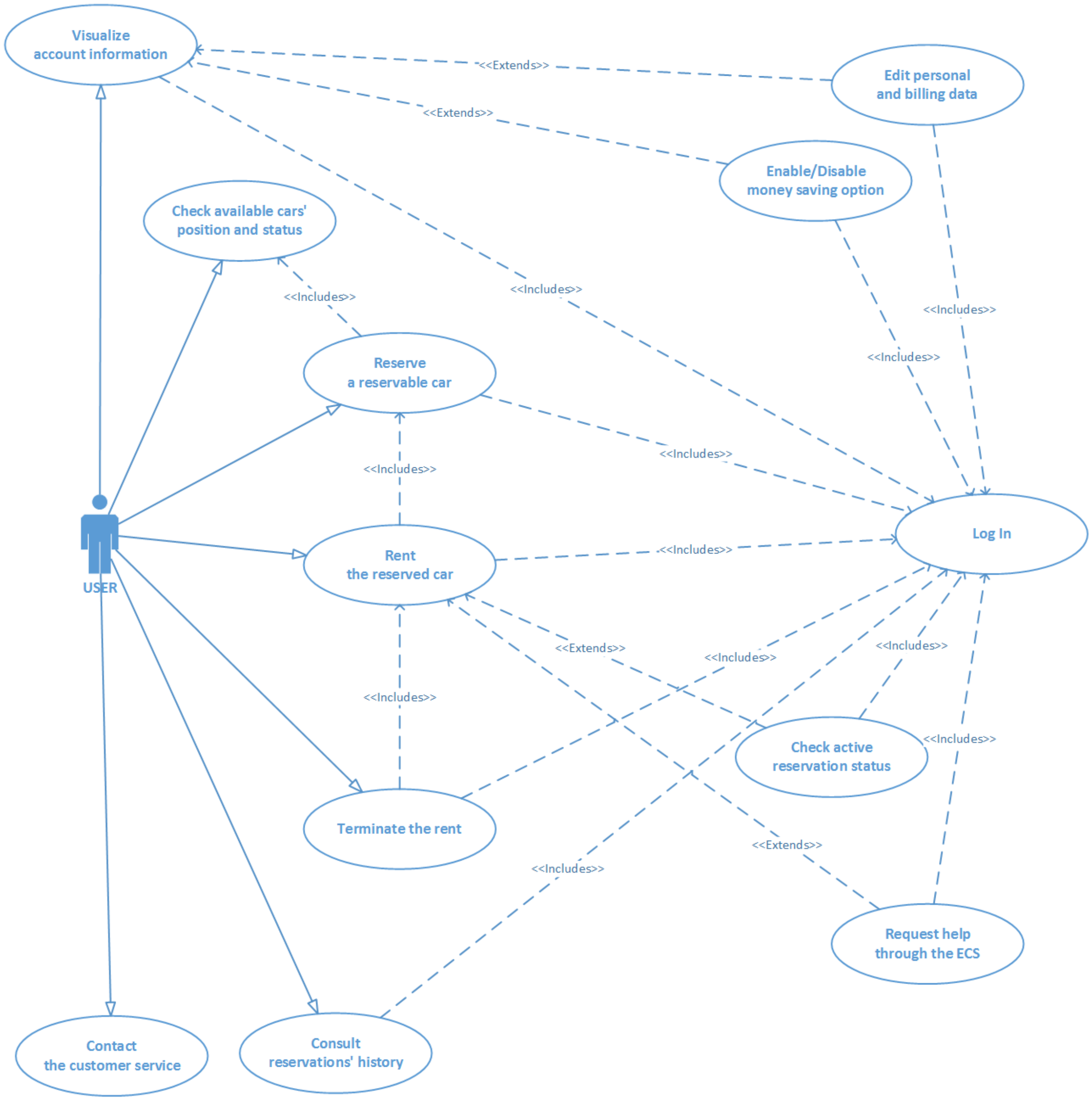

Visualize account information
Edit personal and billing data
Enable/Disable money saving option
Check available cars' position and status
Reserve a reservable car
Log In
USER
Rent the reserved car
Check active reservation status
Terminate the rent
Request help through the ECS
Contact the customer service
Consult reservations' history
<<Extends>>
<<Includes>>

Figure 2: User Use Case

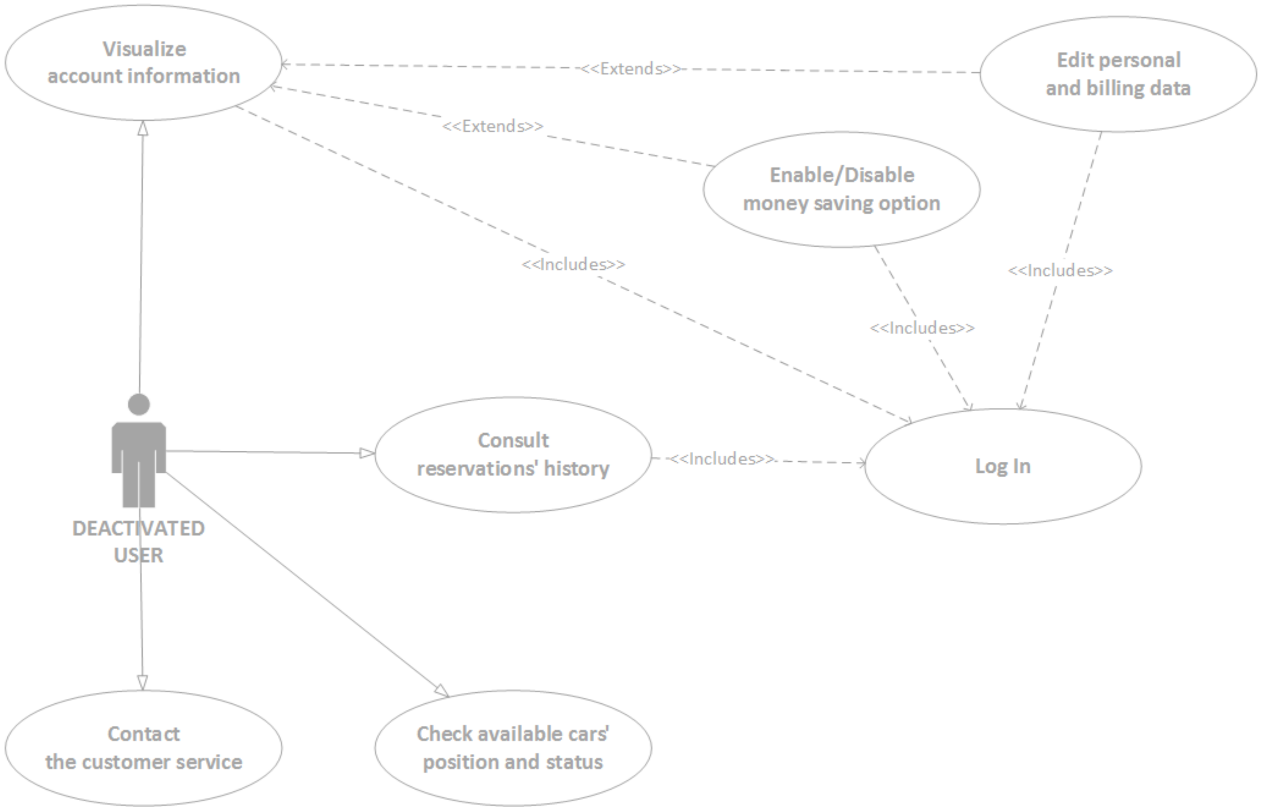

Visualize account information
<<Extends>>
Edit personal and billing data
<<Extends>>
Enable/Disable money saving option
<<Includes>>
<<Includes>>
<<Includes>>
Consult reservations' history
<<Includes>>
Log In
DEACTIVATED USER
Contact the customer service
Check available cars' position and status

Figure 3: Deactivated User Use Case

### 5.1.1 Registration

| | |
|---|---|
| Actors | Guest |
| Preconditions | The guest has a working Internet connection and he has not registered an account yet. |
| Events | 1. The guest reaches the registration page<br>2. The system requires the guest to enter all his/her personal information, driving license data and login credentials along with the verification code<br>3. The guest types the requested information and presses the 'Next' button<br>4. The system verifies the uniqueness of the email, the equality of the twice typed passwords and that the driving license is not expired yet<br>5. The system notifies the guest that he is going to be redirected to an external service web page where he will be required to enter his billing information<br>6. The system receives a confirmation from the external payment service about the correctness of the billing information<br>7. The system shows to the guest a recap of the information already provided and requires him to confirm them, to read and accept the EULA and complete the registration<br>8. The guest ticks the “I read carefully, and I accept the contract” box and presses the “Confirm and Complete Registration” button<br>9. The system sends an email to notify the correct registration to the guest<br>10. The system reports the registration and redirects the user to the login page |
| Postconditions | The user has signed up. |
| Exceptions | The email the guest typed has been already used. The second password does not match with the first one. One of the fields is empty. The SSN is not in compliance with the other guest's personal information. The driving license is already expired. The external payment service didn't confirm the billing information. The guest did not accept the EULA. In these cases the system notifies the error and cannot complete the registration. |

### 5.1.2 Log in

| | |
|---|---|
| Actors | User or Deactivated User |
| Preconditions | The actor has a working Internet connection and he has already registered his account. |
| Events | 1. The actor reaches the log in page<br>2. The system requires the actor to enter his email and password<br>3. The actor types the requested information and press the ’Log in’ button<br>4. The system verifies the correctness of the email and password<br>5. The system redirects the actor to his personal page |
| Postconditions | The actor is logged in. |
| Exceptions | The email or the password the actor typed are not correct. One of the fields is empty. In these cases the actor can’t complete the log in. The system notifies the error and cannot complete the login. |

### 5.1.3 Edit personal information

| | |
|---|---|
| Actors | User or Deactivated User |
| Preconditions | The actor has a working Internet connection and he is already logged into the system. |
| Events | 1. The actor reaches his personal area<br>2. The actor clicks the “Edit personal information” button<br>3. The system allows the actor to change his address, mobile phone number, password (and to confirm it), driving license number, the expiration date and the authority who released it<br>4. The user enters the new information and presses ’Save’<br>5. The system verifies the correctness and the completeness of the information<br>6. The system shows the actor his updated information |
| Postconditions | The user has modified his personal information. |
| Exceptions | One of the fields is empty. The second password does not match with the first one. The driving license is already expired. In these cases the system notifies the error and cannot complete the request. |

### 5.1.4 Edit billing information

| | |
|---|---|
| Actors | User or Deactivated User |
| Preconditions | The actor has a working Internet connection and he is already logged into the system. |
| Events | 1. The actor reaches his personal area<br>2. The actor clicks the “Edit billing information” button<br>3. The system notifies the actor that he is going to be redirect to an external service web page where he will be required to enter his new billing information<br>4. The system receives a confirmation from the external payment service about the correctness of the new billing information<br>5. The system shows the actor his updated information |
| Postconditions | The user has modified his billing information. |
| Exceptions | The external payment service did not confirm the billing information. In this case the system notifies the error and cannot complete the request. |

### 5.1.5 Check available cars' position and status

| | |
|---|---|
| Actors | Guest, User or Deactivated User |
| Preconditions | The actor has a working Internet connection. |
| Events | 1. The actor opens the map<br>2. The system requires the actor to enter an address or to use his position to localize cars<br>3. The actor types the address where he wants to find an available car or clicks "Use my position" button<br>4. The system verifies the correctness of the information and send a request to the RMSS<br>5. The system receives an answer from the RMSS with the available cars and shows them to the actor on the map<br>6. The actor taps the icon that stands for the available car he chooses<br>7. The system shows to the actor the status of the selected available car |
| Postconditions | The actor obtains all the information about the position and the status of any available car in a certain area. |
| Exceptions | The address field is empty. The inserted address is not found or the location service does not work. There is no available cars in the selected area. In these cases the system notifies the error and cannot complete the request. |

### 5.1.6 Reserve a car

| Actors | User |
|---|---|
| Preconditions | The user has a working Internet connection, he has already checked the position and the status of an available car and he is logged into the system for the whole reservation. |
| Events | 1. The user clicks on “Reserve car” button<br>2. The system send to the RMSS the user position and his reservation request<br>3. The system receives an affirmative answer from the RMSS<br>4. The system creates a new instance of the reservation<br>5. The system notifies the user the success of the reservation |
| Postconditions | The user reserved successfully a car. |
| Exceptions | The communication with the RMSS failed. The system says to the user that the service is temporarily not available. The localization service does not work. The system receives a negative answer from the RMSS. In these cases the system notifies the error to the user and he cannot complete the reservation. |

### 5.1.7 Open the reserved car

| Actors | User |
|---|---|
| Preconditions | The user has a working Internet connection, has already reserved a car and he is logged into the system. |
| Events | 1. The user reaches the car he reserved.<br>2. The system checks user position or requires him to enter the code on the windshield<br>3. If necessary, the user enters the code he can read on the windshield<br>4. The system requires the user enter his verification code<br>5. The user types his verification code<br>6. The system verifies the correctness of the verification code and the status of the reservation<br>7. The system unlocks the car doors, terminates the reservation and creates a rent instance on the RMSS<br>8. The user opens the car doors |
| Postconditions | The user can get in the car. |
| Exceptions | The user failed to unlock the car in less than one hour: the system notifies the end of the reservation to the user, declares the reserved car as available again and applies 1€ of fee to the user. \| The system is not able to check the user position. The user fails to enter the windshield code. The verification code is not correct. In these cases the system doesn't unlock the doors. |

### 5.1.8 Use the reserved car and terminate the rent

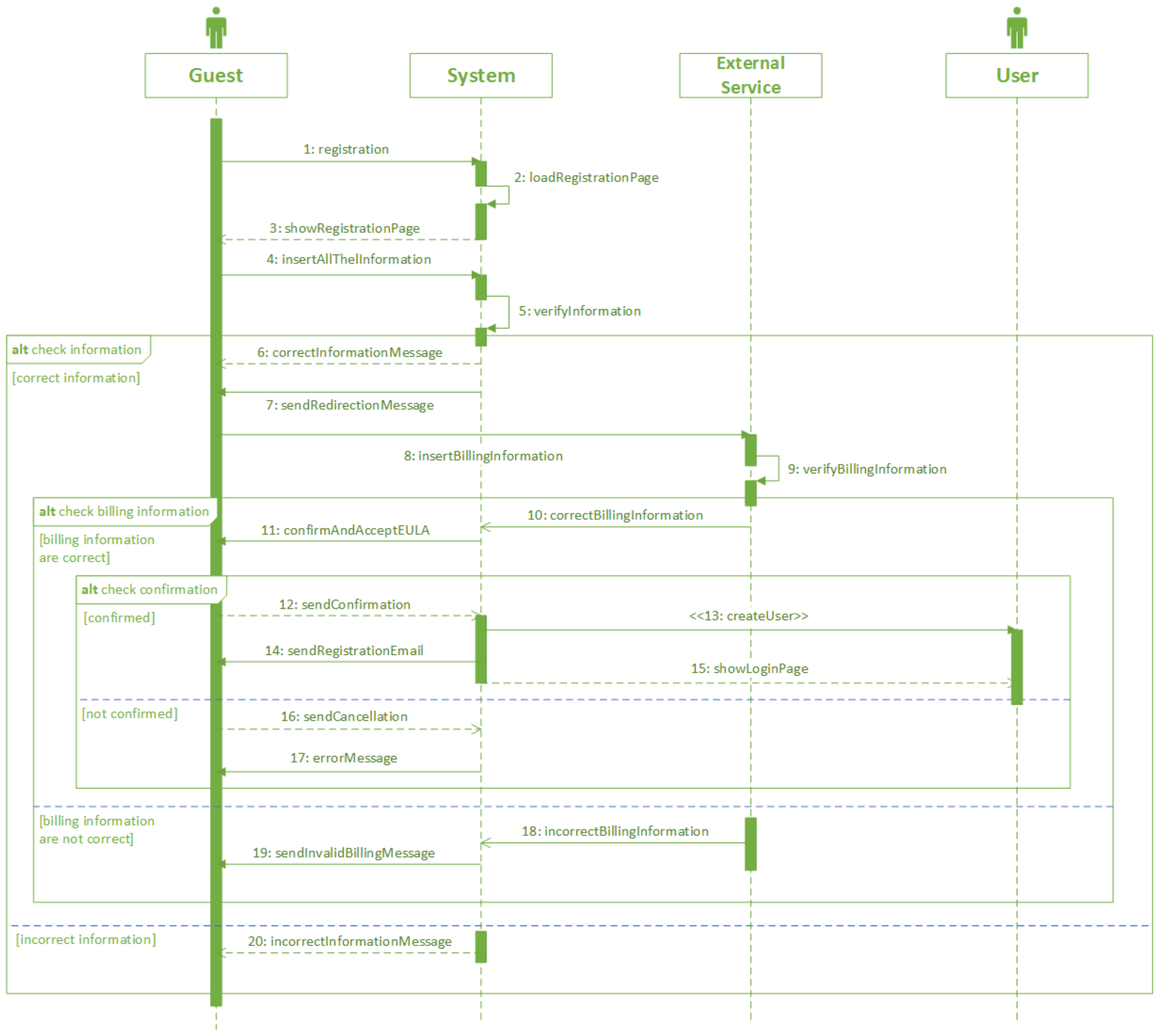
| Actors | User |
|---|---|
| Preconditions | The user has a working Internet connection, has already opened his reserved car and he is logged into the system. |
| Events | 1. If the money saving option is actived, the user can enter his final address on the touchscreen display in the car<br>2. If the user did it, the system shows a special parking area near his destination<br>3. The user starts the engine and uses the car<br>4. As a matter of choice, the user takes care of plugging the car into the power grid<br>5. The user clicks "Terminate rent"<br>6. The system verifies nobody is in the car and that it is in a safe parking area<br>7. The system locks the car doors again, verifies the car's status and updates the rent instance<br>8. If necessary, the system contacts the MES for an ordinary (or extraordinary) car maintenance.<br>9. The system calculates the total cost and notifies the end of the rental to the user along with the total cost |
| Postconditions | The user successfully rented a car. |
| Exceptions | The address is not correct or it is not found. The system notifies the error to the user and requires him to insert it again. \| The car is not in a safe parking area. The car is not empty. In these cases the system doesn't allow to terminate the rent. \| The user can't pay the total cost: the system deactivate the user. |

### 5.1.9 Contact customer service

| | |
|---|---|
| Actors | Guest, User or Deactivated User |
| Preconditions | The actor has a working Internet connection. |
| Events | 1. The actor reaches the “Contact the Customer Service” page<br>2. The system requires the actor to enter his own name, surname, mobile phone number, email address and to explain his problem in less than 1000 words<br>3. The user types the requested information and press the ‘Contact the Customer Service” button<br>4. The system redirects the request of the user to the customer service. |
| Postconditions | The actor successfully contacted the customer service. |
| Exceptions | The actor uses more than 1000 words to explain his request. There is at least one empty field. In these cases the system notifies the error and cannot complete the request. |

## 5.2 Sequence Diagram

This section presents the sequence diagram of the most important interaction, in order to have a dynamic sight of the main entities too.



Guest
System
External Service
User
1: registration
2: loadRegistrationPage
3: showRegistrationPage
4: insertAllTheInformation
5: verifyInformation
alt check information
[correct information]
6: correctInformationMessage
7: sendRedirectionMessage
8: insertBillingInformation
9: verifyBillingInformation
alt check billing information
[billing information are correct]
10: correctBillingInformation
11: confirmAndAcceptEULA
alt check confirmation
[confirmed]
12: sendConfirmation
<<13: createUser>>
14: sendRegistrationEmail
15: showLoginPage
[not confirmed]
16: sendCancellation
17: errorMessage
[billing information are not correct]
18: incorrectBillingInformation
19: sendInvalidBillingMessage
[incorrect information]
20: incorrectInformationMessage

Figure 4: Registration Sequence Diagram

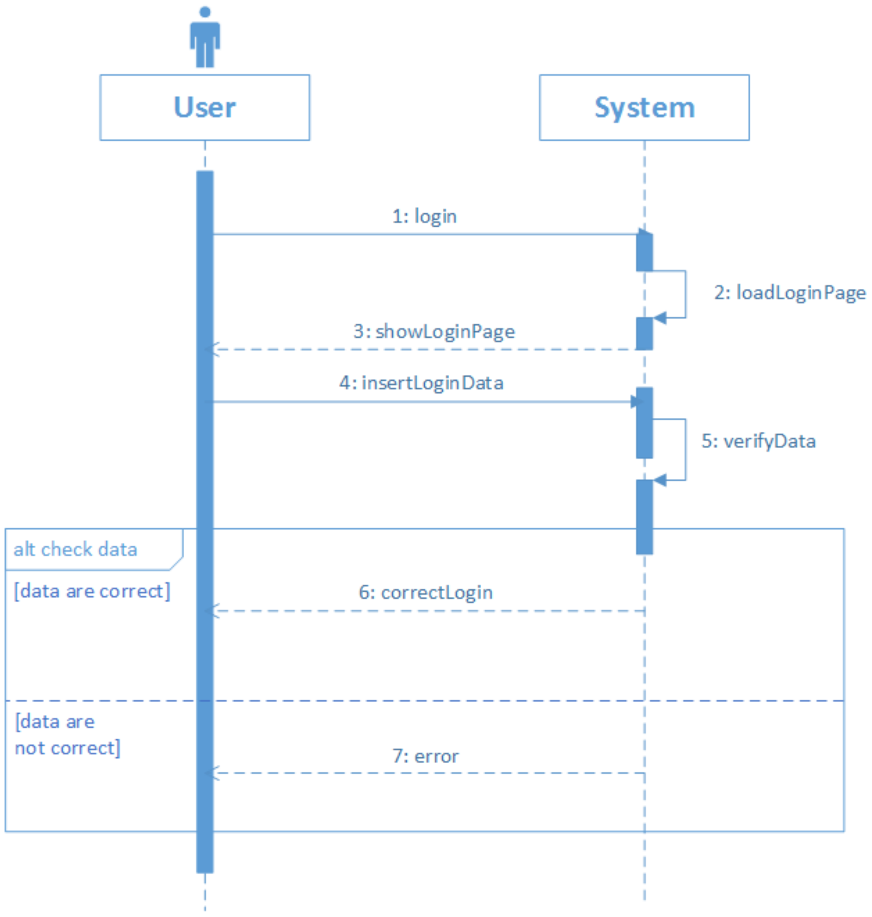


Figure 5: Login Sequence Diagram

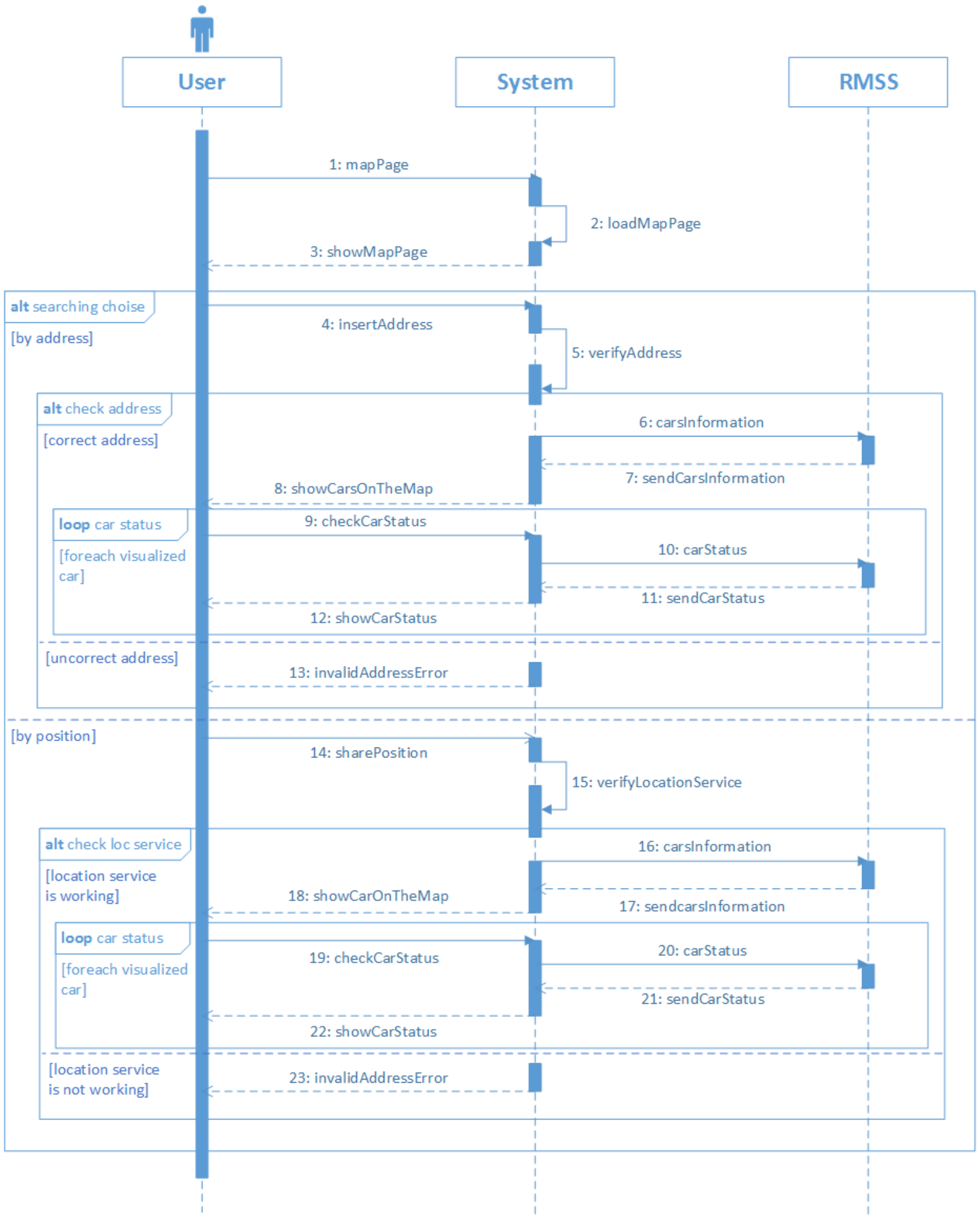


Figure 6: Check Cars' Availability Sequence Diagram

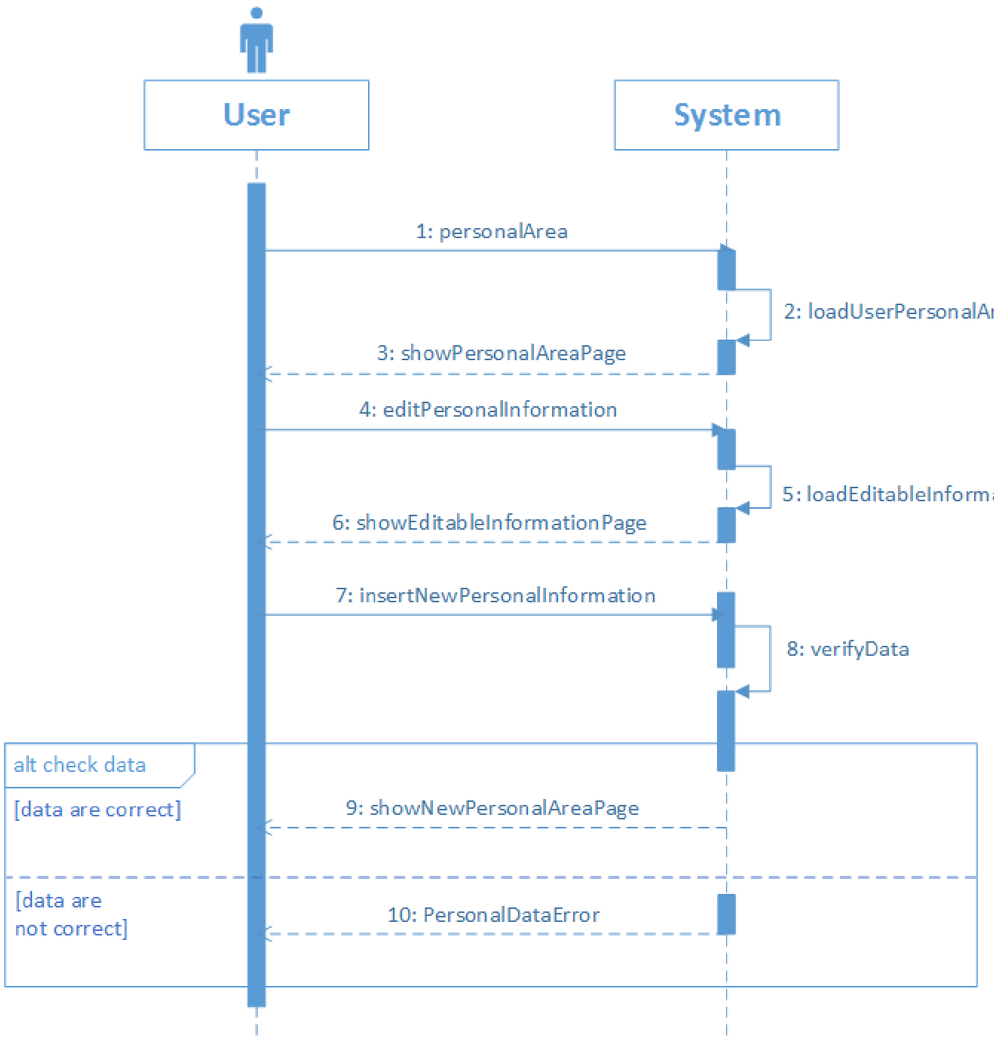

User
System
1: personalArea
2: loadUserPersonalAreaPage
3: showPersonalAreaPage
4: editPersonalInformation
5: loadEditableInformationPage
6: showEditableInformationPage
7: insertNewPersonalInformation
8: verifyData
alt check data
[data are correct]
9: showNewPersonalAreaPage
[data are not correct]
10: PersonalDataError

Figure 7: Modify Personal Information Sequence Diagram

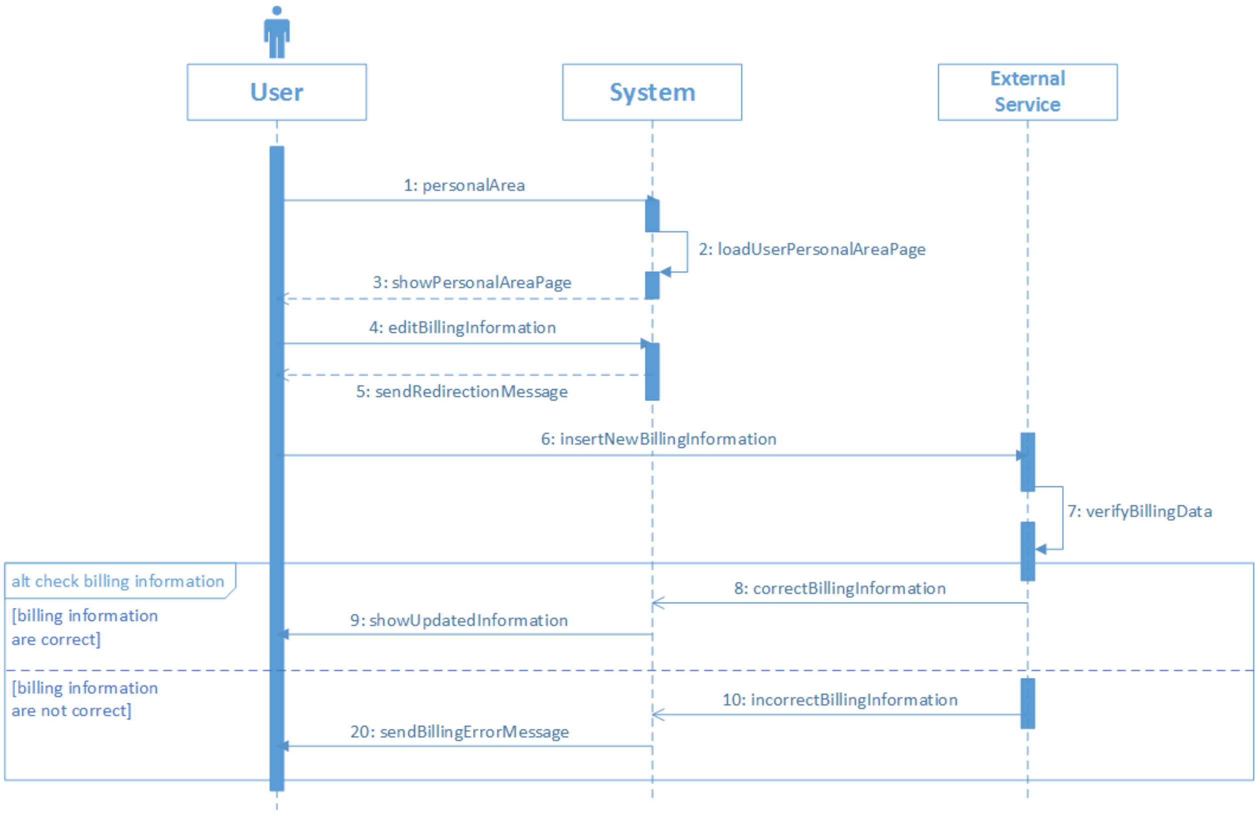

User
System
External Service
1: personalArea
2: loadUserPersonalAreaPage
3: showPersonalAreaPage
4: editBillingInformation
5: sendRedirectionMessage
6: insertNewBillingInformation
7: verifyBillingData
alt check billing information
[billing information are correct]
8: correctBillingInformation
9: showUpdatedInformation
[billing information are not correct]
10: incorrectBillingInformation
20: sendBillingErrorMessage

Figure 8: Modify Billing Information Sequence Diagram

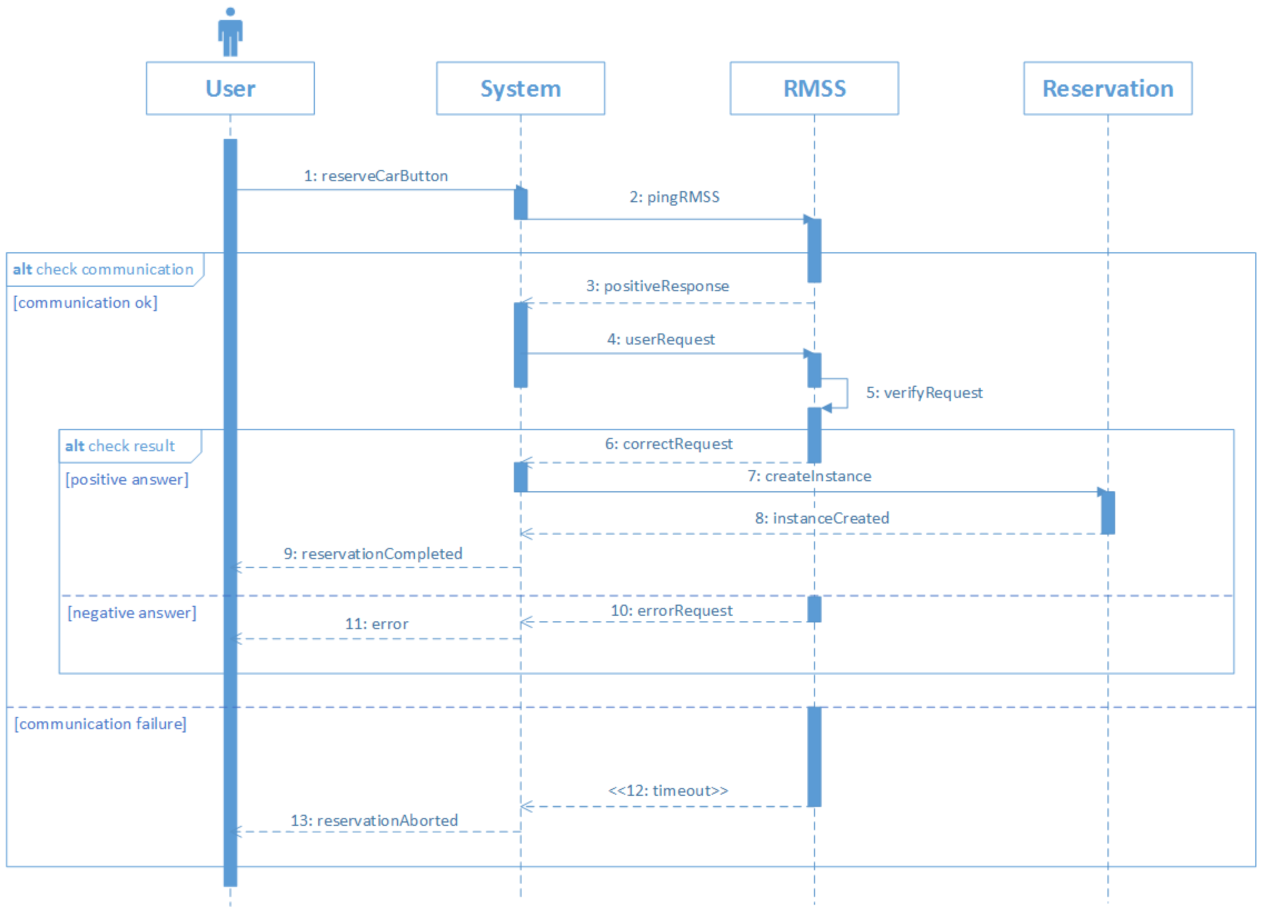


Figure 9: Car Reservation Sequence Diagram

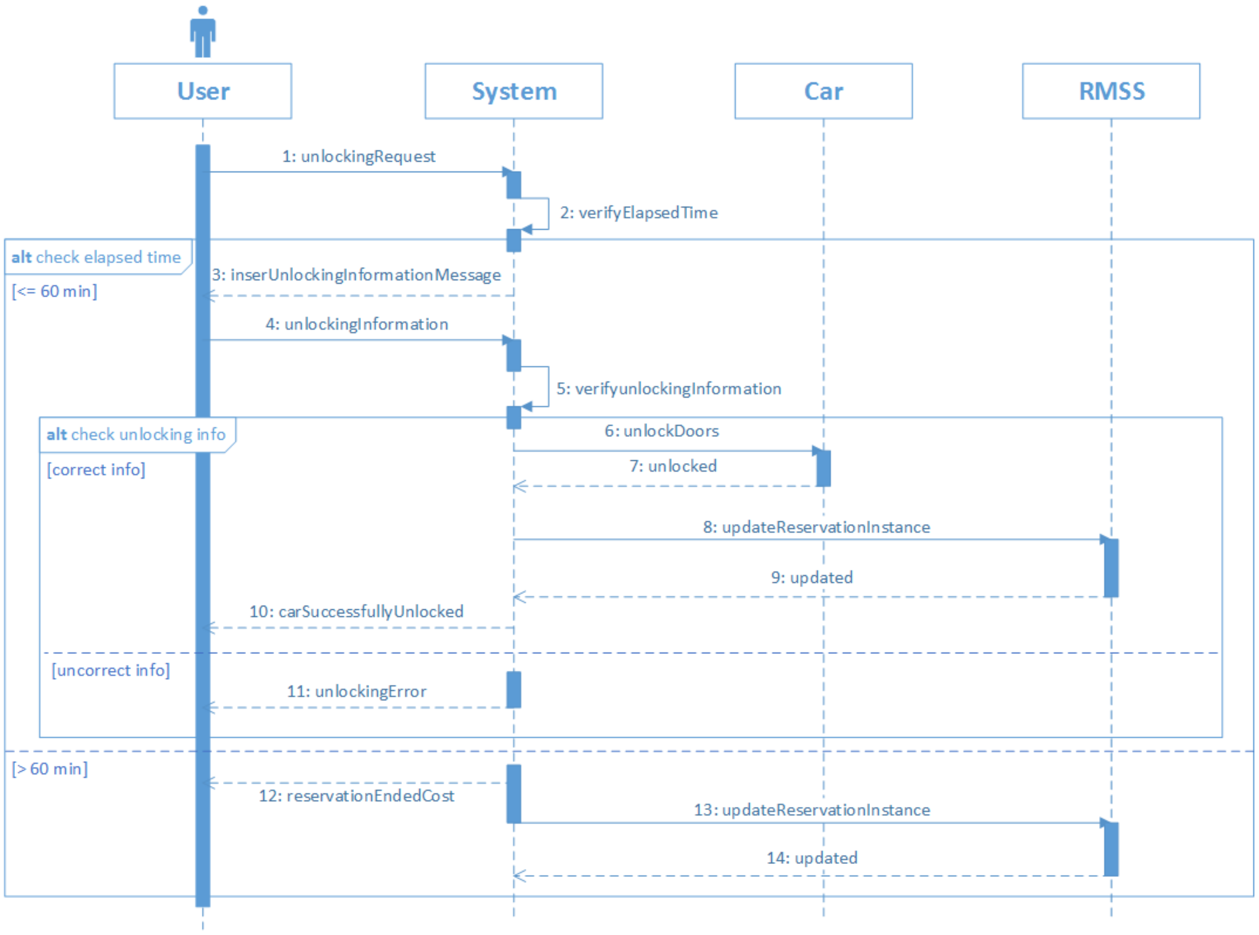


Figure 10: Car Unlock Sequence Diagram

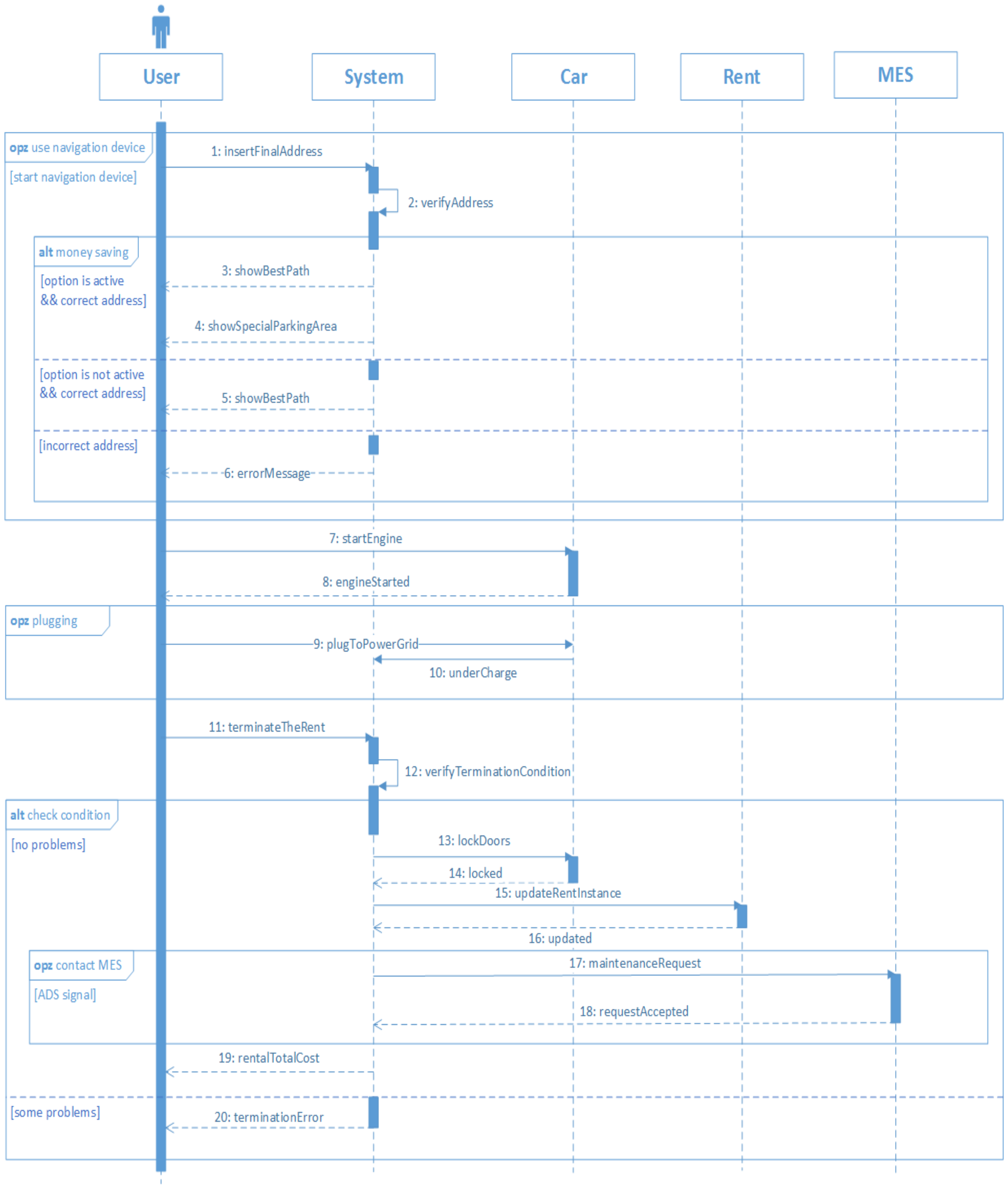


Figure 11: Car Rental Sequence Diagram

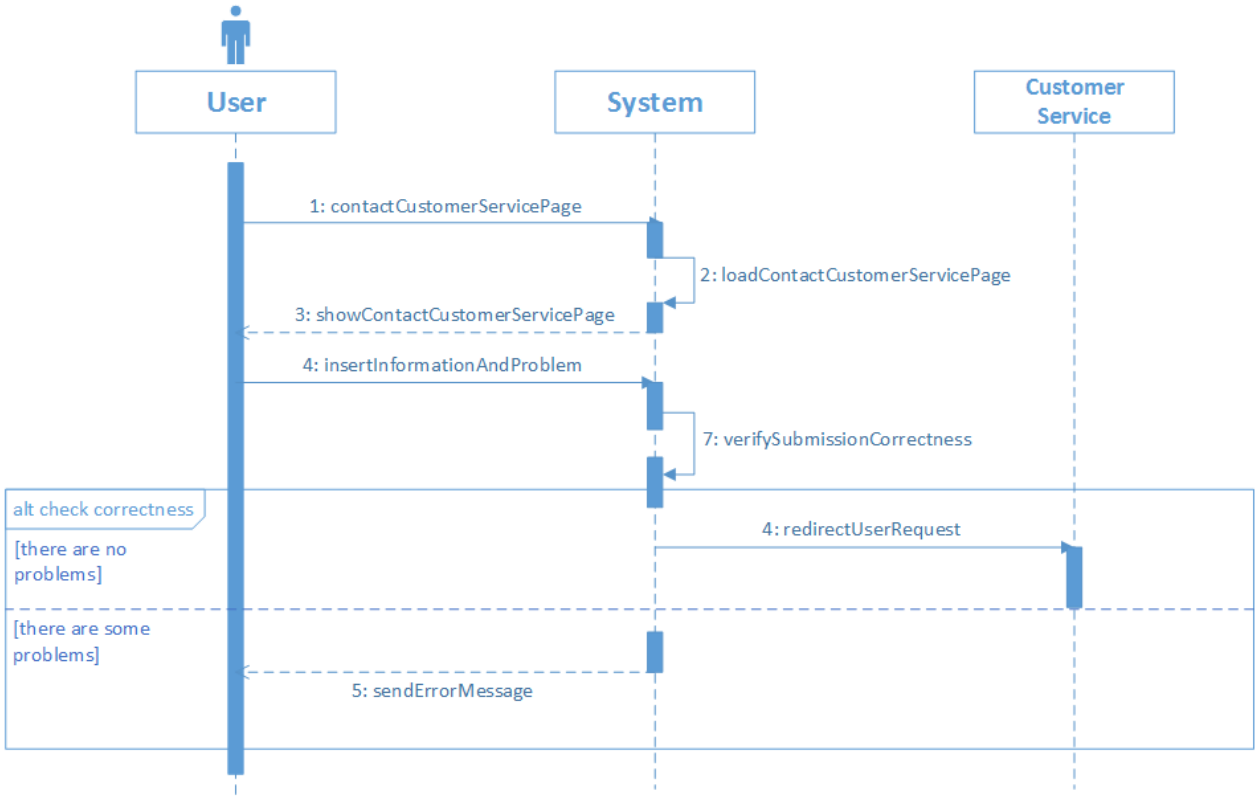

User
System
Customer Service
1: contactCustomerServicePage
2: loadContactCustomerServicePage
3: showContactCustomerServicePage
4: insertInformationAndProblem
7: verifySubmissionCorrectness
alt check correctness
[there are no problems]
4: redirectUserRequest
[there are some problems]
5: sendErrorMessage

Figure 12: Contact Customer Service Sequence Diagram

## 5.3 Class Diagram

Here it is shown a class diagram in order to give a static sight of the main involved entities and of their relations. The diagram, whose aim is just to better specificy requirements, is shown in this picture.

This diagram is intended as a starting draft to be expanded further in the design phase: moreover, it doesn't represent the final class infrastructure which will be used in our system. Finally, this diagram doesn't include the classes that will be used in the Mobile/Web applications.

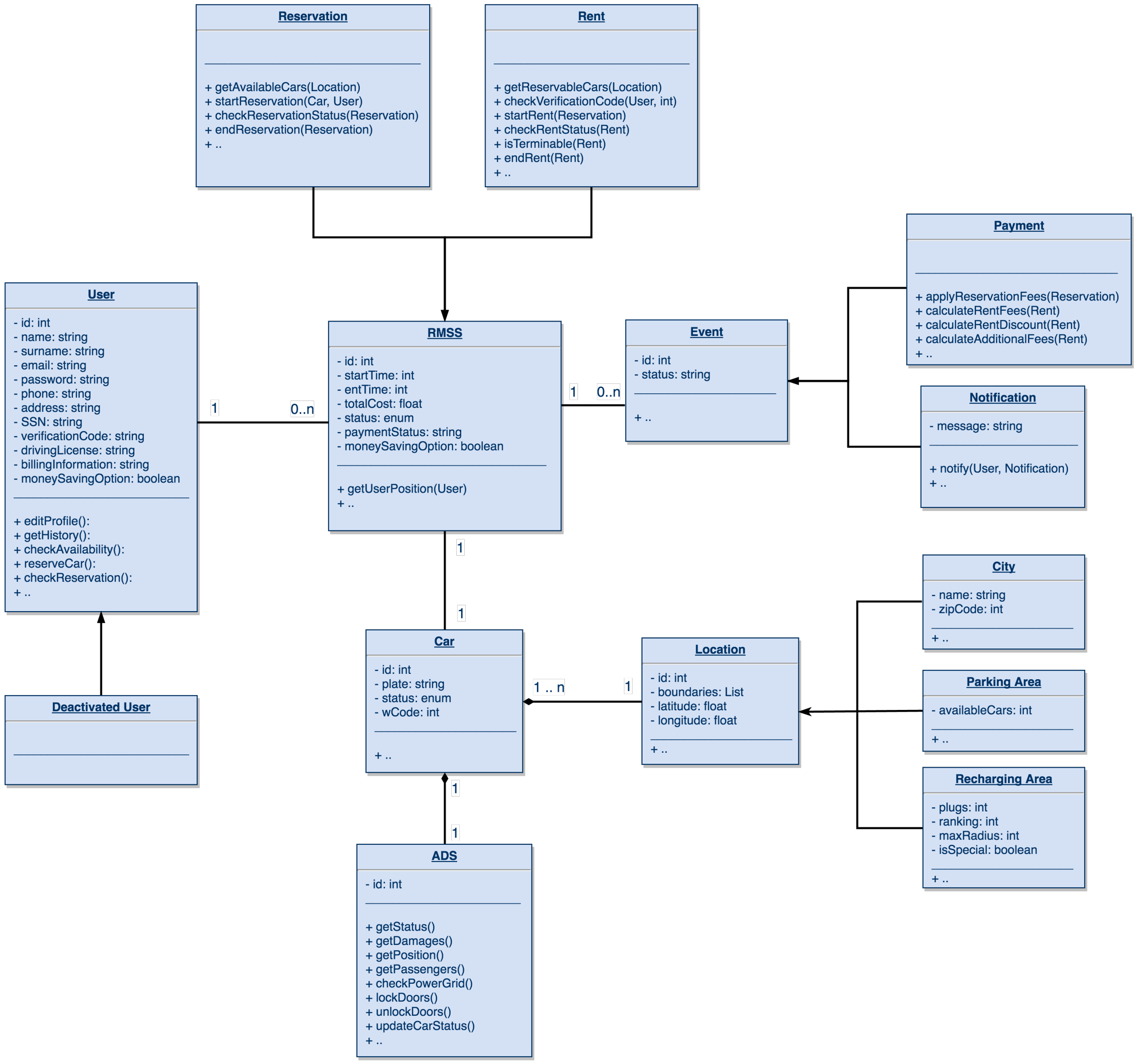


Figure 13: Class Diagram

## 6 Alloy

In this section we tried to verify the consistency of our class diagram. To this goal, we realized a formal model of the system based both on the class diagram and in the previously done assumptions and considerations about constraints. We realized this model using Alloy syntax, thanks to which we get on formally describing the domain of our application and its properties. Then we verified the consistency of our model using Alloy Analyzer. Here there is the code of the model of the system.

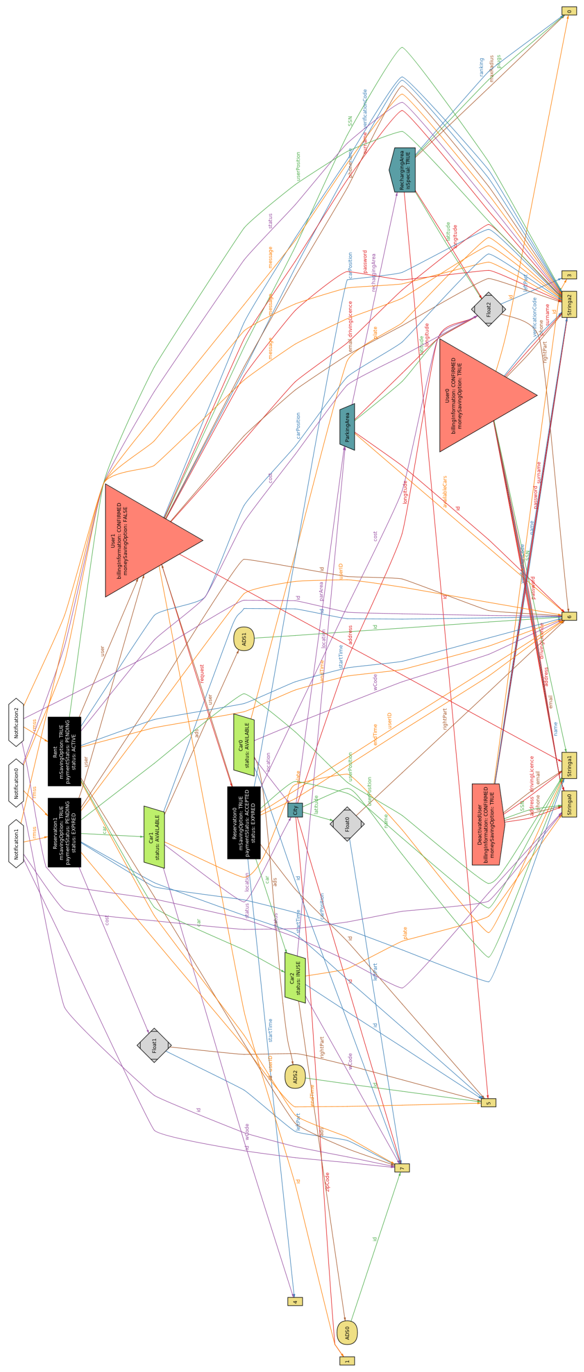
```
module PowerEnjoy

//SIG

sig Stringa {}

sig Float {
    leftPart : one Int,
    rightPart : one Int
} {
    rightPart ≥ 0
}

sig User {
    id : one Int,
    name : one Stringa,
    surname : one Stringa,
    email : one Stringa,
    password : one Stringa,
    phone : one Stringa,
    address : one Stringa,
    SSN : one Stringa,
    verificationCode : one Stringa,
    drivingLicence : one Stringa,
    billingInformation:  one BillingInformation,
    moneySavingOption : one Bool,
    request : set RMSS
} {
    id ≥ 0
}

sig Car {
    id : one Int,
    plate : one Stringa,
    wCode : one Int,
    ads : one ADS,
    location:  one Location,
    status : one CarStatus
} {
    id ≥ 0
    wCode ≥ 0
}

abstract sig RMSS {
    id : one Int,
    startTime : one Int,
    endTime : one Int,
    cost : one Float,
    status : one RequestStatus,
    paymentStatus : one PaymentStatus,
    userID : one Int,
    userPosition : one Stringa,
    carPosition : one Stringa,
    mSavingOption : one Bool,
    car : one Car,
    user : one User,
    events : set Event
} {
    id ≥ 0
    startTime ≥ 0
    endTime ≥ 0
    endTime = none or endTime ≥ 0
    userID ≥ 0
    endTime ≥ startTime
}

sig Reservation extends RMSS {}

sig Rent extends RMSS {}

sig DeactivatedUser extends User {}
sig ADS {
    id : one Int
} {
    id ≥ 0
}

abstract sig Event {
  id : one Int,
  status:  one Stringa,
  rmss:  one RMSS
} {
  id ≥ 0
}

sig Payment extends Event {}

sig Notification extends Event {
  message:  one Stringa
}

abstract sig Location {
  id : one Int,
  boundaries : set Int,
  latitude : one Float,
  longitude : one Float
} {
  id ≥ 0
  latitude.leftPart ≥ 0
  longitude.leftPart ≥ 0
}

sig City extends Location {
  name : one Stringa,
  zipCode : one Int,
  parArea : set ParkingArea
} {
    zipCode > 0
}

sig ParkingArea extends Location {
  availableCars : one Int,
  rechargingArea : one RechargingArea
} {
  availableCars ≥ 0
}

sig RechargingArea extends Location {
    plugs : one Int,
    ranking : one Int,
    maxRadius : one Int,
    isSpecial : one Bool
} {
    plugs ≥ 0
    ranking ≥ 0
    maxRadius ≥ 0
}

// ENUMS

enum Bool {
    TRUE,
    FALSE
}

enum BillingInformation {
    CONFIRMED,
    NOTCONFIRMED
}

enum PaymentStatus {
    ACCEPTED,
    PENDING,
    DENIED
}

enum CarStatus {
    AVAILABLE,
    RESERVED,
    UNAVAILABLE,
    INUSE
```

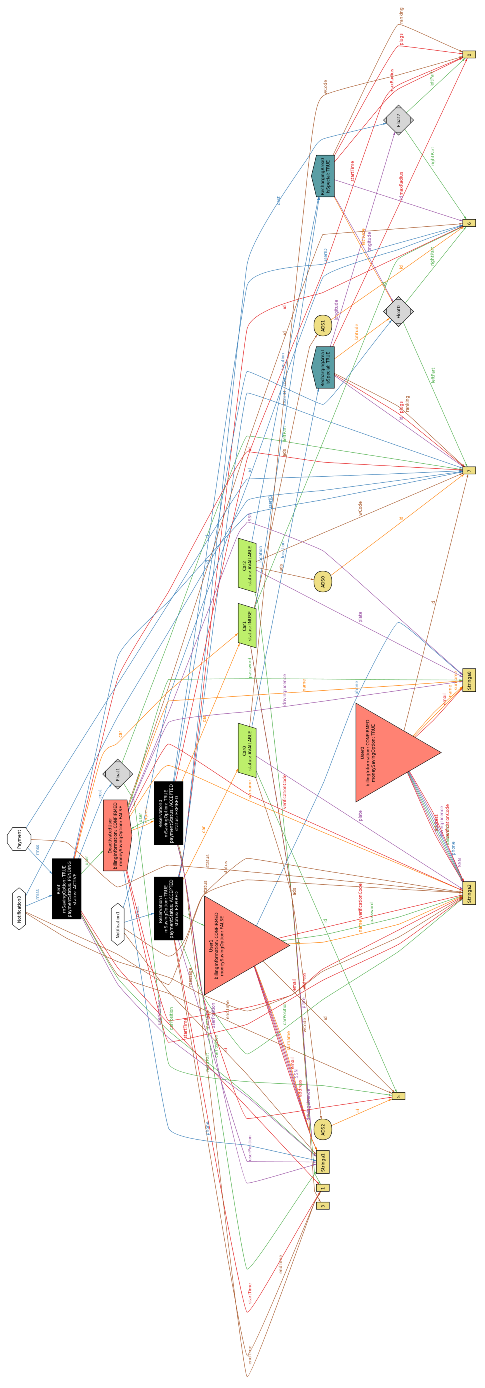
```
}

enum RequestStatus {
    ACTIVE,
    EXPIRED
}

// FACTS

// In any city there is at least a parking area
fact atLeastAParkingArea {
    (all c : City | #c.parArea ≥ 1)
}


// In any parking area there could be zero or more recharging area
fact presenceOfRechargingArea {
    (all p : ParkingArea | #p.rechargingArea ≥ 0)
}

// In any parking area there could be zero or more cars
fact presenceOfCars {
    (all p : ParkingArea | #p.availableCars ≥ 0)
}

// No ADS with the same ID
fact noDuplicatedADS {
    (no ads1 , ads2 : ADS | ads1.id = ads2.id and ads1 ≠ ads2)
}


// The same ADS cannot be used by two different Cars
fact theSameADSCannotBeUsedByDifferentCars {
    (no disj c1, c2 : Car | c1.ads = c2.ads)
}

// No Duplicated Users
fact noDuplicatedUser {
    (no u1 , u2 : User | u1.id = u2.id and u1 ≠ u2) and
    (no u1 , u2 : User | u1.email = u2.email and u1 ≠ u2) and
    (no u1 , u2 : User | u1.SSN = u2.SSN and u1 ≠ u2) and
    (no u1 , u2 : User | u1.drivingLicence = u2.drivingLicence and u1 ≠ u2)
}

// A reservation and its associated rent have the same user
fact noPhantomResRent {
  (all c : Car | c.status = INUSE implies
    (no res : Reservation, ren : Rent | res.car = c and ren.car = c and res.user ≠
ren.user)
  )
}

// No rent is possible if the reservation payment was denied or pending
fact noRentIfPaymentUncertain {
  (all c : Car, res : Reservation| c.status = INUSE and res.car = c and (res.paymentStat
DENIED or res.paymentStatus = PENDING) implies
    (no ren : Rent | ren.car = c)
  )
}

// No Users with NOTCONFIRMED Billing Information
fact noUserWithNotConfirmedBilling {
    no u : User | u.billingInformation = NOTCONFIRMED
}

// Every rent starts after every reservation
fact noEarlyRent {
  (all c : Car, res : Reservation, ren : Rent | c.status = INUSE and res.car =
c and ren.car = c implies
    ren.startTime ≥ res.endTime
  )
}

// No Cities with the same ID
fact noDuplicatedCities {
    no c1 , c2 : City | c1.id = c2.id and c1 ≠ c2
}

// No Parking Areas with the same ID
fact noDuplicatedParkingAreas {
    no pa1 , pa2 : ParkingArea | pa1.id = pa2.id and pa1 ≠ pa2
}

// No Recharging Areas with the same ID
fact noDuplicatedRechargingAreas {
    no ra1 , ra2 : RechargingArea | ra1.id = ra2.id and ra1 ≠ ra2
}

// No Duplicated Requests
fact noDuplicatedRMSS {
    (no req1 , req2 : RMSS | req1.id = req2.id and req1 ≠ req2) and
    (no req1, req2 : RMSS | req1.userID = req2.userID and
    req1.car = req2.car and req1.startTime = req2.startTime
    and req1.endTime = req2.endTime and req1 ≠ req2)
}

// No Duplicated Cars
fact noDuplicatedCars {
    (no c1 , c2 : Car | c1.id = c2.id and c1 ≠ c2) and
    (no c1 , c2 : Car | c1.plate = c2.plate and c1 ≠ c2) and
    (no c1 , c2 : Car | c1.wCode = c2.wCode and c1 ≠ c2)
}

// When a car is RENTED the related RENT is ACTIVE and viceversa
fact aRentedCarIsRelatedToAnActiveRent {
    (all c : Car | c.status = INUSE implies
    (one ren : Rent | ren.car = c and ren.status = ACTIVE) and
    (no res : Reservation | res.car = c and res.status = ACTIVE) )
    and
    (all ren : Rent | ren.status = ACTIVE implies
    (one c : Car | ren.car = c and c.status = INUSE)
    )
}

// When a car is RESERVED the related RESERVATION is ACTIVE and viceversa
fact aReservedCarIsRelatedToAnActiveReservation {
    (all c : Car | c.status = RESERVED implies
    (one res : Reservation | res.car = c and res.status = ACTIVE) and
    (no ren : Rent | ren.car = c and ren.status = ACTIVE)
    ) and
    (all res : Reservation | res.status = ACTIVE implies
    (one c : Car | res.car = c and c.status = RESERVED)
    )
}

// When a car is UNAVAILABLE, it cannot be RESERVED nor RENTED
fact noUnavailableReservedCar {
    all c : Car | c.status = UNAVAILABLE implies (
    (no res : Reservation | res.status = ACTIVE and res.car = c) and
    (no ren : Rent | ren.status = ACTIVE and ren.car = c)
  )
}

// When a request is ACTIVE the status of the payment is PENDING
fact pendingPaymentForActiveRequest {
    (all r : RMSS | r.status = ACTIVE implies r.paymentStatus = PENDING)
}

// There are no duplicate payments
fact noDuplicatePayments {
  (all r1, r2 : RMSS | r1 ≠ r2 and r1.paymentStatus = PENDING and r2.paymentStatus =
PENDING implies
    (all p1, p2 : Payment | p1 ≠ p2 and p1 in r1.events and p2 in r2.events)
  )
}
// No Multiple Users for the same Request
fact noMultipleUsersForTheSameRequest {
     no disj u1, u2 : User | u1.request & u2.request ≠ none
}

// The same Request cannot be performed by two different User
fact noDifferentUserForTheSameRequest {
    (all u1, u2 : User | u1 ≠ u2 implies
      (no r : RMSS | r in u1.request and r in u2.request)
    )
}

// The same User cannot have two ACTIVE Requests
fact theSameUserCannotPerformTwoActiveRequests {
    no disj r1, r2 : RMSS | r1.user = r2.user and
```

```
        r1.status = ACTIVE and r2.status = ACTIVE
}

// The same User cannot start two Request contemporary
fact noSimultaneousActions {
   no disj r1, r2 : RMSS | r1.user = r2.user and r1.startTime = r2.startTime
}

// Relation between deactivated users and active requests (reservation or rent)
fact noActiveRequestForDeactivatedUser {
   (all dU : DeactivatedUser | no r : dU.request | (r.status = ACTIVE))
     // No deactivated users can have an active request
}

//Consistency of the MoneySavingOption for the ACTIVE Requests
fact consistencyOfMoneySavingOptionForActiveRequests {
   (all u : User | u.request.status = ACTIVE implies
      u.moneySavingOption = u.request.mSavingOption)
}

// A Rent is possible only as a consequence of a Reservation
fact rentIsAPossibleConsequenceOfReservation{
   (all r : Rent | one res : Reservation | res in r.user.request and
             r.startTime = res.endTime and r.car = res.car)
}



// ASSERTIONS

/******* WORKING *******/       //[no counterexamplefound]
// The number of active rents is equal to the number of cars in use
assert equalityOfRentedCarsAndActiveRents {
   #{r : Rent | r.status = ACTIVE} = #{c : Car | c.status = INUSE}
}
// check equalityOfRentedCarsAndActiveRents for 10

/******* WORKING *******/       //[no counterexamplefound]
// The number of active reservations is equal to the number of cars reserved
assert equalityOfReservedCarsAndActiveReservations {
    #{r : Reservation | r.status = ACTIVE} = #{c : Car | c.status = RESERVED}
}
// check equalityOfReservedCarsAndActiveReservations for 10

/******* WORKING *******/       //[no counterexamplefound]
//The number of the Reservation is greater or equal to the number of Rent
assert noRentWithoutReservation {
    all u : User |
    #{res:  Reservation | res.user = u } ≥ #{ren : Rent | ren.user = u}
}
// check noRentWithoutReservation for 10

/******* WORKING *******/       //[no counterexamplefound]
//If there is an end time for a Reservation, that must be after the start
assert requestTimeConsistency {
    all r : RMSS | r.endTime > 0 implies r.endTime > r.startTime
}
// check requestTimeConsistency for 10



pred show {
    #User ≥ 2
    #DeactivatedUser = 1
    #RMSS ≥ 2
    #Car = 3
    #{ r : RMSS | r.status = ACTIVE} ≥ 1
    #{ r : Rent | r.status = ACTIVE} ≥ 1
}

run show for 3
```

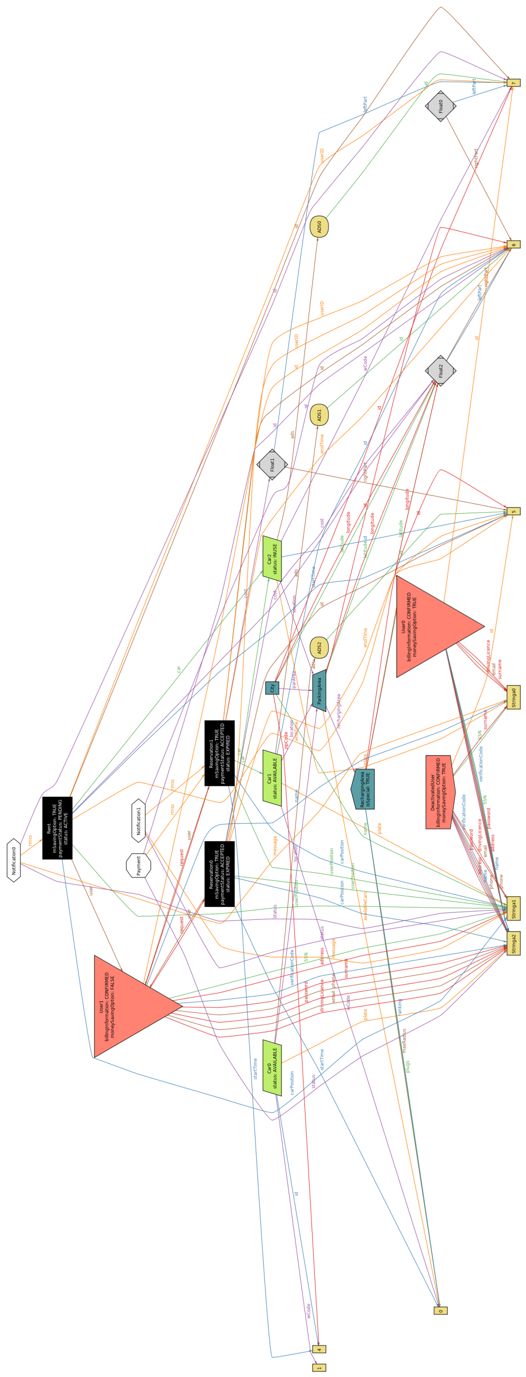
## 6.1 Generated Worlds

Here are presented three generated worlds, according to the model specified in Alloy.





Notification0
Payment
Rent mSavingOption: TRUE paymentStatus: PENDING status: ACTIVE
DeactivatedUser billingInformation: CONFIRMED moneySavingOption: FALSE
Float1
Notification1
Reservation1 mSavingOption: TRUE paymentStatus: ACCEPTED status: EXPIRED
Reservation0 mSavingOption: TRUE paymentStatus: ACCEPTED status: EXPIRED
User1 billingInformation: CONFIRMED moneySavingOption: FALSE
Car0 status: AVAILABLE
Car1 status: INUSE
Car2 status: AVAILABLE
ADS0
ADS1
ADS2
User0 billingInformation: CONFIRMED moneySavingOption: TRUE
RechargingArea1 isSpecial: TRUE
RechargingArea0 isSpecial: TRUE
Float0
Float2
Stringa0
Stringa1
Stringa2
7
6
0
5
3
1



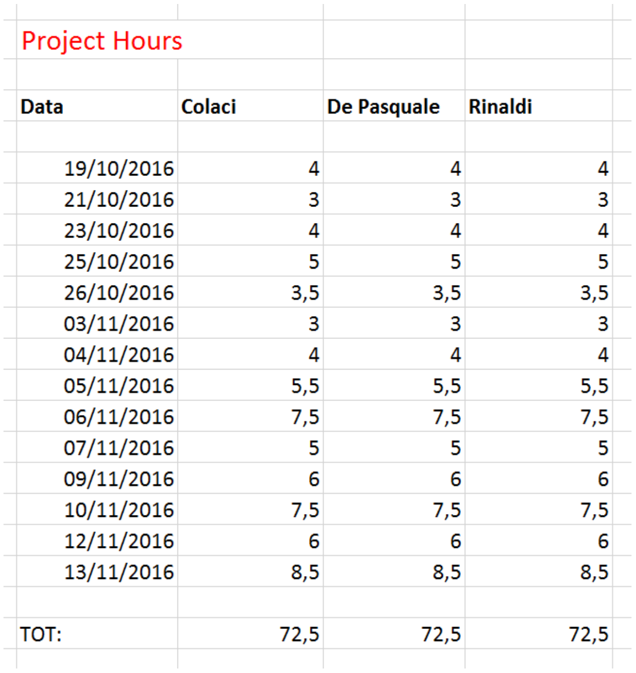
# 7 Appendix

## 7.1 Tools used

We used the following tools to produce this document:

- **LaTex** as typesetting system to write this document
- **LyX** as editor
- **Visio Professional** to draw all the diagrams
- **Alloy Analyzer 4.2** to write and verify alloy models

## 7.2 Hours spent



Project Hours

| Data | Colaci | De Pasquale | Rinaldi |
|---|---|---|---|
| 19/10/2016 | 4 | 4 | 4 |
| 21/10/2016 | 3 | 3 | 3 |
| 23/10/2016 | 4 | 4 | 4 |
| 25/10/2016 | 5 | 5 | 5 |
| 26/10/2016 | 3,5 | 3,5 | 3,5 |
| 03/11/2016 | 3 | 3 | 3 |
| 04/11/2016 | 4 | 4 | 4 |
| 05/11/2016 | 5,5 | 5,5 | 5,5 |
| 06/11/2016 | 7,5 | 7,5 | 7,5 |
| 07/11/2016 | 5 | 5 | 5 |
| 09/11/2016 | 6 | 6 | 6 |
| 10/11/2016 | 7,5 | 7,5 | 7,5 |
| 12/11/2016 | 6 | 6 | 6 |
| 13/11/2016 | 8,5 | 8,5 | 8,5 |
| TOT: | 72,5 | 72,5 | 72,5 |