

**POLITECNICO DI MILANO**  
**Computer Science and Engineering**  
**Project of Software Engineering 2**

# **Requirements Analysis and Specification Document**

Authors: Falci Angelo 875123  
Lanzuise Valentina 807364  
Lazzaretti Simone 875326

Reference Professor: Di Nitto Elisabetta

# SUMMARY

1. INTRODUCTION.....	3
1.1 Description of the given problem.....	3
1.2 Goals.....	3
1.3 Domain Properties.....	3
1.4 Glossary.....	4
1.5 Assumptions.....	4
1.6 Proposed System.....	5
1.7 Identify Stakeholders.....	6
2. ACTORS IDENTITYFING.....	6
3. REQUIREMENTS.....	6
3.1 Functional requirements.....	8
3.2 Non-functional requirements.....	9
4. SCENARIOS IDENTIFYING.....	14
5.UML.....	16
5.1 Use Cases Definition.....	16
5.2 Class Diagram.....	30
5.3 Sequence Diagrams.....	31
5.4 State Chart Diagram.....	44
6. ALLOY MODELLING.....	45
6.1 Alloy Code.....	45
6.2 Alloy Result.....	51
6.3 Alloy Graphics.....	51
7. OTHER INFORMATION.....	55
7.1Software, Tool and Documentation.....	55
7.2Hours of Work and Effort.....	55

# **1 Introduction**

## **1.1 Description of the given problem**

We will develop PowerEnJoy, a digital management system for car-sharing service of electric cars.

Users will register to the system, giving basic information, such as nickname and the payment mode they prefer, while the system will give them a password.

Once they will be registered they could use the service thanks to the password given.

The system will show them the available cars near their position and the cars' features, so that the users could choose one of them (they will have to be near the car to unlock it), taking the reservation ( they will have an hour to pick it up).

Based on the number of passengers, the power used, and the position in which the car will be parked, the service will give them some discounts. When the car will be parked the system will charge the amount to users and will lock the car after users will be exited.

## **1.2 Goals**

We will develop these features:

- Users will be able to: register to the system, modify their personal information, ask for a reservation, call for assistance, rent a car, choose the destination, use the car, leave it momentarily and use it again.
- The system will be able to: show users the nearest cars available, unlock cars, reveal hypothetical fee, charge the amount, apply any discount, send notifications of execute operations to users.

## **1.3 Domain Proprieties**

We suppose that the system has this proprieties:

- User nickname is unique for every user.
- Every car has a tracking device.
- User must insert the code of a valid identity document and of driving license to complete the registration.
- When user books a car the car can't be booked from other user for one hour.
- User must have a valid payment method if it wants to rent or book a car.
- A user can rent only one car for time.
- A user can book only one car for time.
- A car can be rented by a user for time.
- Safe areas can have or not a power grid to recharge the cars.
- Park function permit to the user to leave the cars without stop the renting.

## 1.4 Glossary

Here we write the main terms we will use in the project with their meaning.

- GUEST: identify the person not registered yet.
- REGISTERED USER: identify the person registered who can use the service.
- USER: identify the generic person who is using the service.
- SAFE AREA: identify the area where the user can leave a car to have a discount.
- POWER GRID: identify the power station that a safe area can have where the user can recharge the car to obtain a discount.
- SYSTEM: identify server and database that manage the web-application service, and the software that manages the car.
- CAR: identify every single car provided by PowerEnJoy.
- POSITION: indicate the specific position about car, safe area and power grid using latitude and longitude.
- ASSISTANCE: identify service who the user can call if it has a problem with the car.
- ASSISTANT: identify both the operator that manages the maintenance of the cars and the telephone operator who helps clients.
- HOMEPAGE: indicate the page in which are addressed either guests and registered users before signing up.
- USER HOME: identify the page in which users are addressed after login where they have access to the services.

## 1.5 Assumptions

The problem doesn't give details for everything, so we do some assumptions.

- User can view only its own information.
- When user subscribes itself it chooses a nickname and a password, then user receives a email that confirms the registration and gets a special code that user uses to turn on the car.
- If a safe area gets full while user is going there, the safe area changes its colour in the map.
- If a power grid gets empty of energy or it isn't available while user is going there, the power grid disappears from the map.
- If a user books a car it can't rent another different car.
- User must vote the conditions of the car when he/she renting.
- The system has a database that contains: user information, car information, safe areas information.
- User can view all information about himself/herself, cars, safe areas and power grids.
- User can't modify its information while it is using a car.

- In every car there is a device whose user can use to interact with the system.
- User must insert the plate of the car and a special code unique for each car in order to rent the car.
- User can rent the car, if it is available, without booking the car previously.
- User can obtain different discounts at once (discounts are cumulated).
- User can cancel the booking.
- When user presses the button "Contact Assistance", starts a call and an assistant will answer.
- In order to turn on the car user must insert its personal code in the car display and if the code is right user can turn on the car with a key that is always inserted in the car ignition.
- User can park the car using "Park Function" (after turn off the car) on the device in the car. When user activates this function the key is unlocked; he/she can close the car with the key; user continues to pay though it parks the car.
- User can't leave the car where it wants but only in the safe areas unless he/she activates the "Park Function", otherwise he/she pays a fee and an assistant will bring the car in the nearest safe area (while the car is out of the safe area it isn't available for other renting/booking).
- Renting ends when the user turns off and closes the car, unless he activated "Park Function".
- Car is available if it is in safe area with more than 20% of energy.
- The cars with less than 20% of energy will be connected by some assistant to a power grid.
- The service is guaranteed only in one city, so it means that safe areas and power grids aren't available out of it.
- Users can leave the city, but they will not be able to park the car, because there will not be safe area. However, they can choose to activate "Park Function".
- User can't use a service if it didn't pay previous debts (rent or fee).
- The system can't manage the users that didn't pay for a long time a debit, it can only exclude them.
- User must have an internet connection to use the service.
- Cars are provided of sensors to know how many passengers there are in the car.
- Assistants is responsible for maintaining the cars clean and working.
- If a car gets fined, the fine must be paid by the user that was renting it when the car got fined.

## 1.6 Proposed Systems

The project aim is to create a web application, based on the client-server paradigm architecture.

In the server (managed as a single mainframe) there will be all the data about cars,

registered users and particular locations. This server will be called by the clients in order to perform every action allowed by the system: borrow car, find special parking areas and others.

Clients will communicate with the server thanks to a web browser and a graphical user interface by phone.

Users, before proceeding with the allowed actions, must register themselves in order to make the system memorizes their information.

Moreover in every car there is a device that permit to the user to have access to some functions of the system like: turn on the car, show map (and optionally safe area and power grid in the map), show charge, use park function and MSO.

### **1.7 Identifying Stakeholders**

Because of the actual phase of the project (starting phase/alpha phase), the main stakeholder is the teacher who held this project. Following her/his indications, the main tasks that we have to care about are requirements analysis, design, architecture and testing. Also, we have to care about some functionalities given by the teacher, based and advanced, in order to make an application close to a real one.

For this reason, we think that a final stakeholder could be a car-sharing online activity.

## **2 Actor Identifying**

The actors in our system are just two:

- GUEST: we already mentioned in the glossary that guest represents the person that hasn't been registered yet. They can interact with the system just to sign up themselves.
- REGISTRED USER: as we said in glossary they are the registered people. User can interact with the system in this way: renting a car, booking a car, refuelling a car, searching for a car, choosing a destination, modifying its personal information, viewing car, power grid or safe areas information.
- ASSISTANT: it is the person to whom the user is addressed when he/she is calling for help, and also the person delegate to displace the cars not left in safe areas, that refuels and manages the maintenance of cars.

## **3 Requirements**

We write the requirements in order to satisfy the goals of the project. We assume that the domain properties, which are described above, holds for our purposes.

- Registration of a person to the system
  - ➔ The system has to provide a sign up functionality.

- Show map
  - ➔ System will show map with all cars available (and eventually safe area and power grid).
  - ➔ If position user is available the map is centred in position user.
- Searching an address in the map
  - ➔ System look for the address inserted.
  - ➔ If it is possible system centres the map in the address.
- Renting a car
  - ➔ System will check the plate and special code the user inserted and will unlock the car if the information is correct.
  - ➔ System will change the state of the car in "used".
  - ➔ System will lock the car when user will be exited from it without active "park function".
  - ➔ System will change again the state of the car in "free" after lock it.
- Booking a car
  - ➔ System will change the state of the car in "booking".
  - ➔ System will save the booking and user who did it.
  - ➔ System will hold the booking for one hour, then will remove the booking and will take money from user who books the car if user doesn't rent the car or cancel the booking.
  - ➔ Booking is stopped if the user deletes it or rents the car.
- Turn on car
  - ➔ System check the personal code that user inserted in the car's device.
  - ➔ If the code is right user can turn on the car.
- Park function
  - ➔ System check if the user pressed "Park function" when he/she turn off and exit from the car.
  - ➔ System doesn't stop the renting and waits the user return.
- Adding or modifying payment method
  - ➔ System will check if the information inserted by user are correct.
  - ➔ System will save information if they are correct, will warn user if it is an error.
- Active saving option
  - ➔ System will elaborate the location where user want to go and it will find the station where user can leave the car to get a discount.
- Checking discount for passengers
  - ➔ System will check if user that rent the car has passengers in the car.
  - ➔ To the road driven with at least 2 passengers user obtains discount of 10%.
  - ➔ System will elaborate the costs of the rent and it will apply the discount obtained.

- Checking discount for battery empty
  - ➔ System will check the battery power when user leaves the car.
  - ➔ System will elaborate the costs of the rent and it will apply discount of 20% if the car is left with no more than 50% of battery empty.
- Checking discount for power grid
  - ➔ System will check if the car is connected into the power grid when user leave it.
  - ➔ System will elaborate the costs of the rent and it will apply discount of 30% if the car is connected into the power grid.
- Checking raises
  - ➔ System will check if the car is left at more than 3 Km from the nearest grid station or with more than 80% of the battery empty.
  - ➔ System will elaborate the costs of the rent and it will increase by 30% if the car is left at more than 3 Km from the nearest grid station or with more than 80% of the battery empty.
- View current charge
  - ➔ System will show how much user is spending until that moment since he/she will have started to drive.

### **3.1 Functional Requirements**

Now that we have defined the main features of PowerEnJoy, we can find some functional requirements concerning each defined actor:

- Guest can:
  - ➔ Sign up.
- Registered user (or simply user) with his/her device can:
  - ➔ Log in.
  - ➔ Rent a car.
  - ➔ Booking a car.
  - ➔ Delete booking he did previously.
  - ➔ View the map with the available cars.
  - ➔ View in the map safe areas.
  - ➔ View in the map safe areas that have some power grid.
  - ➔ Searching an address in the map.
  - ➔ Modify personal information.
  - ➔ Contact assistance.
  - ➔ Log out.
- Registered user with car's device can:
  - ➔ Insert his/her in order to turn on the car.
  - ➔ View the map centred in the car he/she is using.
  - ➔ Searching a destination in the map.



- ➔ View the current charge.
- ➔ View safe areas in the map around his/her position.
- ➔ View only safe area with some power grid around his/her position.
- ➔ Activate MSO.
- ➔ Activate “Park Function”.

### 3.2 Non-functional Requirement

#### *User-Interface of web-application*

The interface of our application it is thought to be used for every device. It will be composed principally by the following pages.

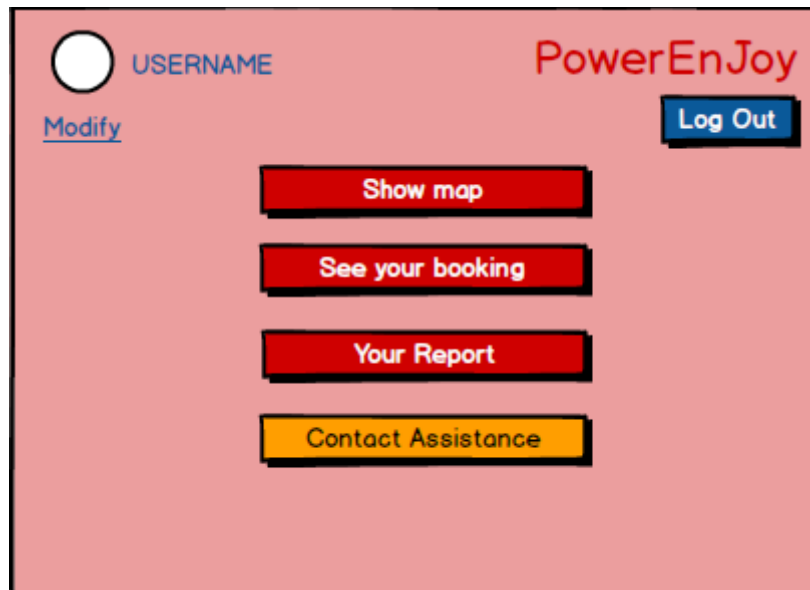
#### *Home Page*

At first both guests and users are addressed to these pages. If a registered user wants to login, it has to insert its nickname and its password, and then to push the special button. Instead, if they are guests and want to subscribe, they have to insert their credentials, in particular a nickname, a password, their name and surname, date of birth, an e-mail address, their Identity Card number, their driving license and the mode of Payment they prefer. After filling this forms, they must push the button "Sign up", and if all fields will be valid, the system will send an e-mail to users containing a special code. We can see below a first sketch of this web-page.

#### *User Home*

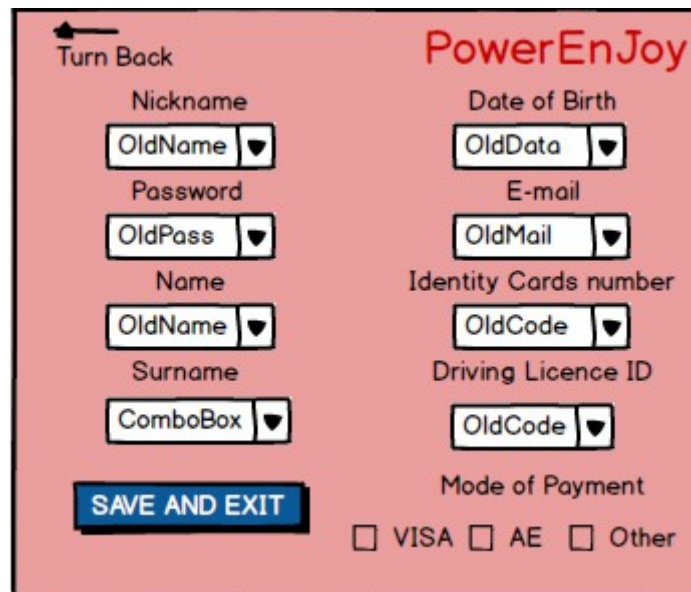
After login, users are addressed to their home, where they can choose what to do pressing the special buttons below: they can view the map with their position and available cars, they can see details about their booking if they have taken one, they can visualize the report about historic payment and system notifications, they can contact assistance and start a call with PowerEnJoy assistants. User can also modify their old data and insert new ones, pushing 'Modify', located below their profile icon.

If they want to exit, they will click on "Log Out" on the right.

A screenshot of a user profile page titled "PowerEnJoy". At the top left, there is a circular profile icon, the text "USERNAME", a blue "Modify" link, and a blue "Log Out" button. In the center, there are four stacked buttons: "Show map" (red), "See your booking" (red), "Your Report" (red), and "Contact Assistance" (yellow).

#### *Modifications Personal Information*

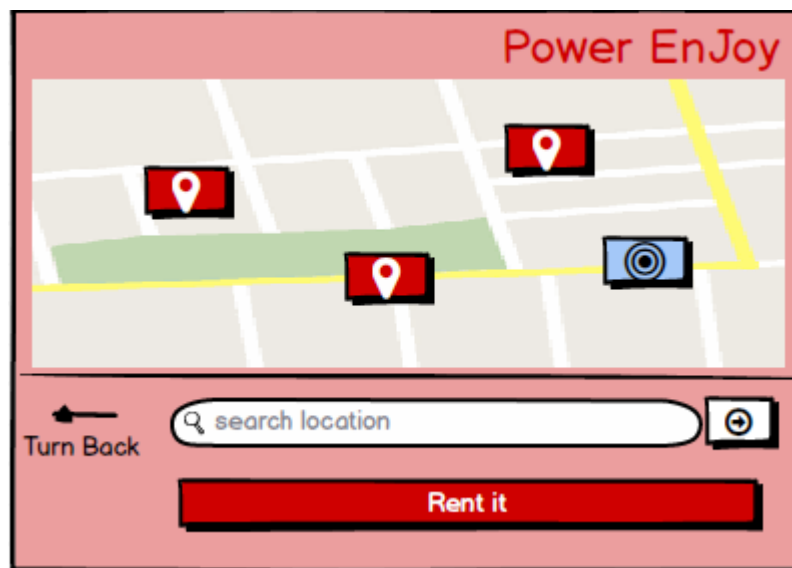
After click "Modify" users are addressed to this page, in which they can insert new information, change old ones or just turn on the previous pages.

A screenshot of a "Personal Information" modification form titled "PowerEnJoy". It features a "Turn Back" link with a left arrow at the top left. The form is organized into two columns. The left column contains fields for "Nickname" (OldName), "Password" (OldPass), "Name" (OldName), and "Surname" (ComboBox). The right column contains fields for "Date of Birth" (OldData), "E-mail" (OldMail), "Identity Cards number" (OldCode), and "Driving Licence ID" (OldCode). At the bottom right, there is a "Mode of Payment" section with checkboxes for "VISA", "AE", and "Other". A blue "SAVE AND EXIT" button is located at the bottom left of the form.

#### *Show Map*

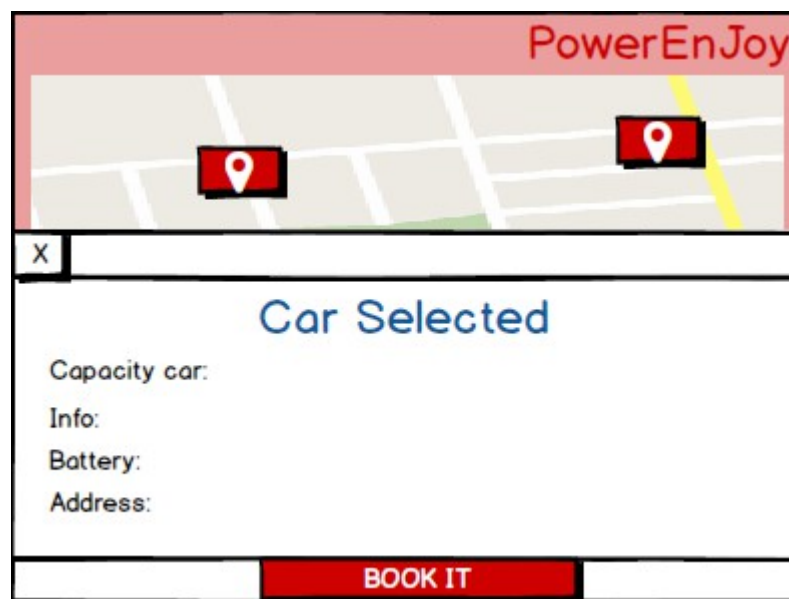
If user chooses "Show map" from the User Home, he/she accesses to the service. She/he can see his/her position in the map (the blue rectangle in the picture) and the position of the nearest cars available (the red rectangles in the picture). If user wants to know which car are available but in a location far away from him/her, she/he can insert the name of the place he/she wants to search, and can visualize it on the map. If users select cars, they can take a booking (as we are going to see in the next sketch),

instead if they are close to a car, they can take it also without booking it, pushing the bottom "Rent it".



### *Selecting car*

When users select cars clicking on them, it opens a window which shows the main information of the car, so users can choose to book it pushing the button below, or to turn on the map just closing the window.



### *Booking Page*

When users decide to book a car they can access to this page from the User Home, in which there are shown information about the reservation, clicking on the button "See your booking". When users are near the car they must click the button "Rent it", while they can always deleting the reservation before the deadline of the booking hour pushing on "Delete Booking". If they click on "Show map" they turn on the

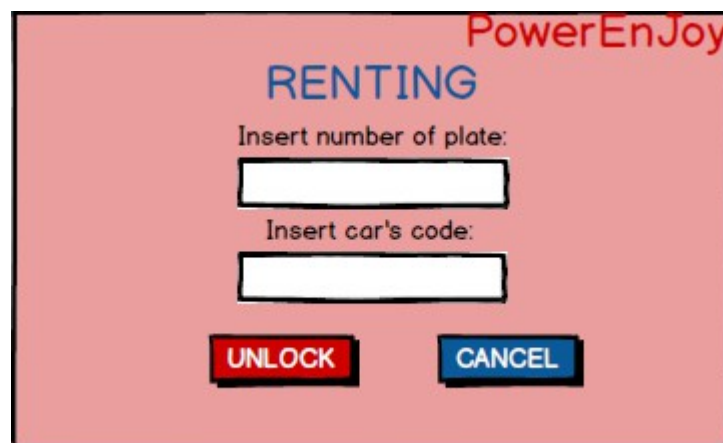
map, instead if they press "Turn Back" they are addressed to the User Page.



The screenshot shows a mobile application interface for "PowerEnJoy" with a pink background. At the top left, there is a black arrow pointing left and the text "Turn Back". At the top right, the "PowerEnJoy" logo is displayed in red. In the center, the word "BOOKING" is written in large blue letters. Below this, there are three labels: "Booking time:", "Last time:", and "Info car:". At the bottom, there are three red buttons with black text: "Rent it", "Delete Booking", and "Show Map".

### *Rent a car*

When users choose to rent a car, either from the map page or from the booking page, they are sent to this renting page you can see below. If users want to unlock the car they want to use, they have to insert the number of plate and the car's code that is located inside the car but visible from the outside, and then push "Unlock". While they want to delete the renting they have to click on "Cancel", and they will be addressed to the previous page.



The screenshot shows a mobile application interface for "PowerEnJoy" with a pink background. At the top right, the "PowerEnJoy" logo is displayed in red. In the center, the word "RENTING" is written in large blue letters. Below this, there are two labels: "Insert number of plate:" and "Insert car's code:". Each label is followed by a white rectangular input field. At the bottom, there are two buttons: a red button with black text labeled "UNLOCK" and a blue button with white text labeled "CANCEL".

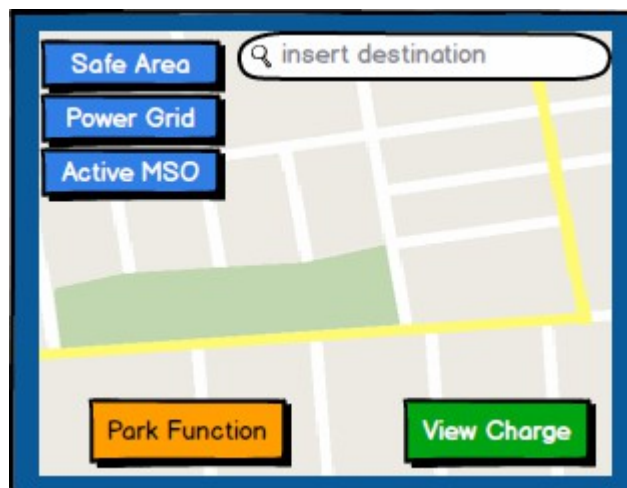
### *Report Page*

Clicking on "Report" from the User Page, users are sent to the report page, in which they can see the system notifications received. These notifications can be of different type, such as payment messages, confirm or delete booking, notification of fee, or others.

← Turn back
REPORT
PowerEnJoy

TypeOfMessage	Data	Cost	Information
Payment	1/01/2001	23\$	Partenza: Via Carlini
Confirmation Booking	1/01/2001		Time: 1.34
Delete Booking	1/01/2001		Time: 3.45
Fee For Booking	2/12/2000	1\$	Car 3432432
Generics	1/01/2001	-	Error

### *User-interface of the car's display*



This is the interfaces which is visible from the screen on the dashboard of the car. Clicking “Safe Area” or “Power Grid” the map would show respectively the safe areas and the safe areas with power grid situated near the car, while clicking “Active MSO” the modality “Money saving option” would be activated. Inserting the destination, the device's system would search the address and show the route, instead the “View Charge” would permit to visualize for few second the current charge. At last, with the special button “Park Function” the users would be able to leave the car at the moment and then take it again.

## **Other non-functional requirements**

The application should provide:

- User friendliness: the interface would be the most intuitive and simple as possible.
- Compatibility: the application would be compatible with the major devices on the market (available for Android, iOS, etc.)
- Security: the system would ascertain that there wouldn't be violation, that Users data will remain undamaged and reserved, ensuring through authentication the access at their own credentials and preventing ill-intentioned people to access with data of others.
- Availability: the system would be always obtainable for the users with an Internet connection, and the service would be guarantee 24/24 h, 7/7 days.
- Accuracy: the application would show the exact location through GPS device on the phone, and would indicate precisely the position of the cars.
- Performance: to supply suitable service, the system would have to be reactive and able to answer to a high number of requests in a short time.
- Assistance: user would be always able to contact an assistant (calling a phone number) that would help him/her in case of problems of various types, or in case of clarifications or looking for information.

## **4 Scenarios Identifying**

The application must be able to help users in common situations/activity about car-sharing. Here there are some scenarios afforded by PowerEnJoy:

- Gino, because of the missing of a car to reach his office, needs an application to rent a car. Also, he pays attention to the pollution, so he wants to use electrical cars. So he decides to search an application with these prerequisites. Gino finds this application, he starts to read information about it and then he decides to try it.  
At the first access the system asks him to register. So, Gino enters his data (name, surname, nickname, email, password, number of ID, driving license ID and mode of payment ). After this registration, the system send to the user an identifying password, in order to recognize him at every access.
- Gino needs to reach a pub at the centre of Milan. He decides to open PowerEnJoy and (after the log-in) asks where is the nearest car to him. The system than takes the position of Gino and searches the nearest free-usable cars. When the system finds them, it's shown to Gino a map where it's shown how to reach the proposed cars. Because it's too early to take the car in that moment, Gino decides to book it for the 19.00.  
Thanks to the fact that he can take the car at most an hour from the booking,

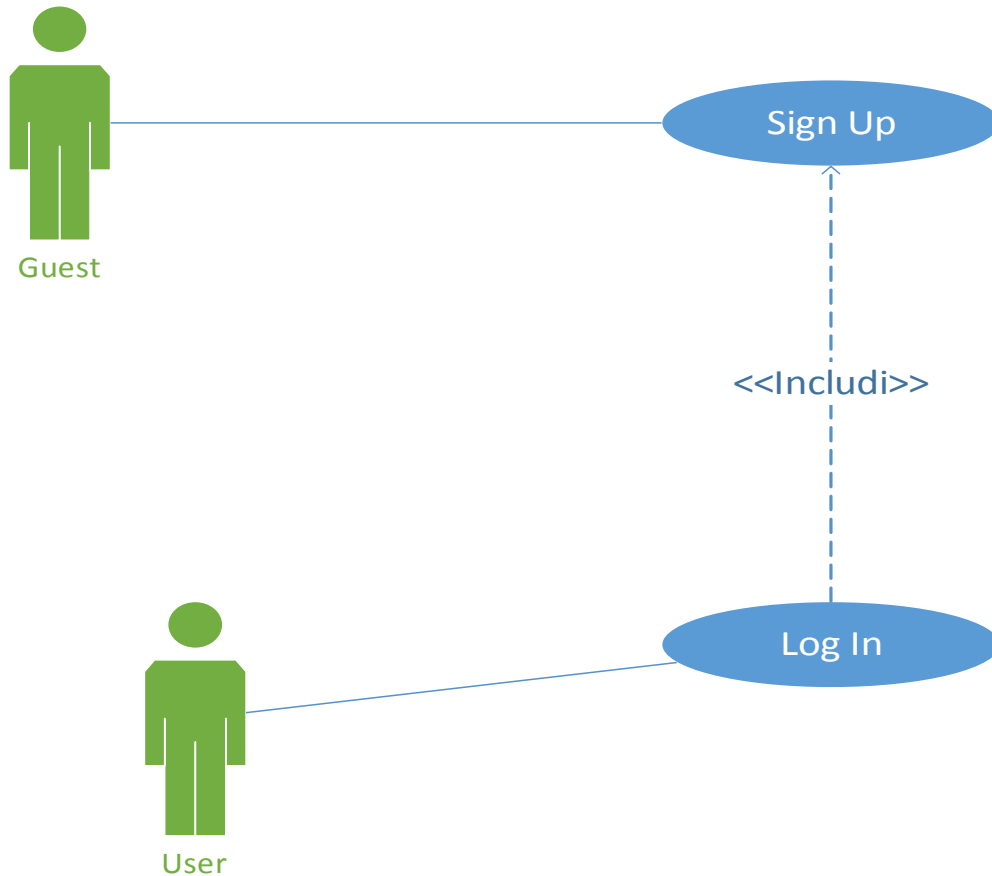
Gino takes the car at the 19.45 and tells the system that he is reaching it when it's nearby, inserts the numbers of plates and the car's code and then the system unlocks the booked car for usage.

- At the first stop, Gino wants to know how much is spending until that moment. So he changes the screen on the car from “GPS” to “CURRENT COST” by pressing the homonymous button. The system, as answer, shows the desired amount on the screen.
- When closed to the place to reach, Gino looks on the screen where are situated the safe areas for parking the car. After reaching the most comfortable one, Gino exits from the car and the system locks both the car and the cost. When Gino commits the payment, he sees a raise of 30% on the payment because the parking area where he left the car was distant 4,5 KM from the nearest power grid station.
- Frodo wants to have a party in Milan center with some friends, so they decides to take a car with PowerEnJoy. After taking it, they travelled from their flat in Piola to the destination and when they arrived, Frodo refuelled the car to a power grid station, whose position is shown on the screen in the car. At the payment, Frodo sees that the system decreased the original payment by 40%, 10% for driving with two other passengers and 30% for leaving the car with 46% battery.
- Gino is travelling from Milan centre to Milano Bovisa. He decides to leave the car in a power grid station, so he asks to the system through the function “MONEY SAVING OPTIONS” where is the optimal power grid station to leave the car. The system provides indications to Gino to reach the location. When the location is reached, Gino exits from the car and, Gino pays 50% of the original amount, 30% for leaving the car in the power grid and taking care of the power plugging, 20% for leaving the car with 66% power battery.
- Jack is an assistant working for PowerEnjoy. As usually, he checks cars used from a certain hour and, when users finish their renting time, reach them in order to control their status. After each check, if the status is acceptable, Jack continues checking the other cars of his list. If he finds that the car is dirty, has engine or/and external problems, Jack try to resolve the problem.
- Gino has a problem to turn on the car. To avoid to waste time, Gino decides to call assistance, so he presses the button "Contact Assistance" and waits for an assistant to answer. In the assistance center, Jack answers the call and starts the conversation with the user. At the explanation of the problem, the assistant gives directions about how to resolve it. Gino follows the instructions and turn on the car. Gino thanks the assistant and ends the call.

## 5 UML

### 5.1 Use Cases Definition

Let's start defining the use cases shown above in the previous graph:



As first step, we have to put attention to registration and log-in situation. The graphic above shows that log-in necessities of a previous registration. Below there are the explanations of these two cases.

#### *Definition of “Sign Up”*

Name	Sign Up
Actors	Guests
Entry Conditions	The guest isn't registered on the website

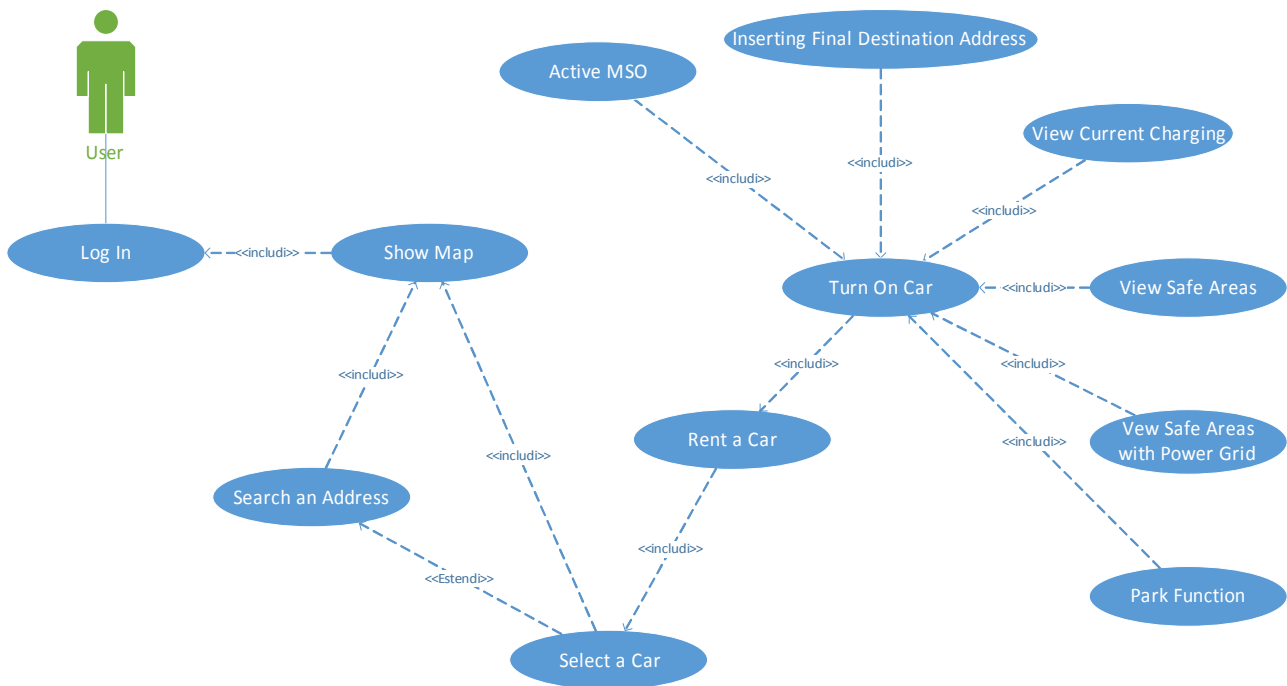


Flow of Events	<p>The guest enters the website;  The guest clicks on the “SIGN UP” button;  The guest fills the form with these informations:  -Nickname  -Password  -Name  -Surname  -E-mail  -Date of birth  -Address  -Identity card number.  -Driving Licence ID  -Mode of payment</p>
Exit Conditions	User has been successfully registered and the system sends to the user a mail with a special code that user use to turn on the cars he will rent
Exceptions	User didn't filled properly one of the field requested or he is registered yet

### *Definition of “Log In”*

Name	Log In
Actors	Registered user
Entry Conditions	The user has signed up correctly on the website
Flow of Events	<p>User enters the website;  User presses on the “LOG IN” button;  User inserts nickname and password he chose during the registration</p>
Exit Conditions	The system shows to the user the user page
Exceptions	User doesn't insert a valid nickname or password

## Renting & Driving



In this huge graph, we illustrate one of the two main part of our service: how a registered user can rent a car and what it can do when it's on it during the rent. An important thing to say is the double-way to select a car: even if the user is constrict to pass through the “Show Map” use case, we have decided to give two way to select our cars, in order to give more flexibility to our system. Below the definition of the represented use case:

### Definition of "Show Map"

Name	Show Map
Actors	Registered user
Entry Conditions	User logged in successfully
Flow of Events	User presses on the button "Show Map";
Exit Conditions	System shows the map near the user position if it active GPS. Or system show only the map
Exceptions	The map isn't available for technical reason.

### *Definition of “Search an Address”*

Name	Search an Address
Actors	Registered user
Entry Conditions	User is on the map
Flow of Events	User inserts a specified address in the dedicate text box; The user clicks the button on the right of the text box;
Exit Conditions	The system centre the map on the address inserted
Exceptions	User inserts a non valid address

### *Definition of “Select a Car”*

Name	Search a Car
Actors	Registered user
Entry Conditions	User is on the map (Eventually, but no mandatory) The user has searched a certain address
Flow of Events	User selects in the map the car that he is interesting in
Exit Conditions	The system displays to the user the car selected
Exceptions	No exceptions triggered

### *Definition of “Renting a Car”*

Name	Renting a Car
Actors	Registered user
Entry Conditions	User is on the map or user is on the booking page and booked a car
Flow of Events	User presses on the button “Rent It”;

	User must insert a vote from 0 to 5 about car conditions; User enters in a blank field a code situated inside the car, visible from the outside, and the plate of the car;
Exit Conditions	The system unlocks the car for user, ready to be driven; The car isn't available for other users
Exceptions	User doesn't insert a valid code in the blank field; Another user booked or rented the car when the user was going to insert the code on the blank

### *Definition of “Turn on Car”*

Name	Turn on Car
Actors	Registered user
Entry Conditions	User rented a car.
Flow of Events	User enters in the rented car; If turned off, user turns on the display inside the car; User inserts its personal code, received after the registration by the system, in the display inside the car; If code is right user can use the key to turn on the car
Exit Conditions	Car is turned on and display shows the map
Exceptions	User inserted wrong code

### *Definition of “Park Function”*

Name	Park Function
Actors	Registered user
Entry Conditions	User rented and turn off the car.
Flow of Events	User press the button “Park Function” on

	the device in the car; User pick up the key unlocked; User exit to the car with the key
Exit Conditions	Car get close
Exceptions	No exception

*Definition of “Visualize Current Charging”*

Name	Visualize Current Charging
Actors	Registered user
Entry Conditions	User must be inside the car
Flow of Events	If turned off, user turns on the display inside the car; User presses on the button “Current Charging”;
Exit Conditions	The system shows the current amount in a corner of the display for few seconds.
Exceptions	No exception triggered

*Definition of “Inserting final destination address”*

Name	Inserting final destination address
Actors	Registered user
Entry Conditions	User must be inside the car and successfully completed “Turn on Car” use case.
Flow of Events	User inserts in the display in the car the address where he want to go; The user confirm the address
Exit Conditions	The system shows on the display a the road the user can do to reach the address
Exceptions	The system, for technical reasons, is not available

### *Definition of “View Safe Areas”*

Name	View Safe Areas
Actors	Registered user
Entry Conditions	User must be inside the car and successfully completely turn on car use case.
Flow of Events	User presses on the button “View Safe Areas”;
Exit Conditions	The system shows on the map in the display all safe areas around the user position.
Exceptions	The system, for technical reasons, is not available

### *Definition of “View Safe Areas with Power Grid”*

Name	<i>View Safe Areas with Power Grid</i>
Actors	Registered user
Entry Conditions	User must be inside the car and successfully completely turn on car use case.
Flow of Events	User pushes the button “View Safe Areas with Power Grid”;
Exit Conditions	The system shows on the map in the display all safe areas with power grid around the user position.
Exceptions	The system, for technical reasons, is not available

### *Definition of “Active Money Saving Options”*

Name	Active Money Saving Options
------	-----------------------------



### *Definition of “Booking a Car”*

Name	Booking a Car
Actors	Registered user
Entry Conditions	User has chosen a car to be displayed on
Flow of Events	User presses on the button “Book It”;Or the user presses on the button "X" and turn to the map
Exit Conditions	If user pressed "Book It" the system shows the user until when the car is reserved and how to reach it; If user pressed "X" the system closes the window and turns to show the map without booking the car.
Exceptions	Another user booked the car in the time between the view of the car profile and the pressure of the “Book Car” button

### *Definition of “ See your Booking”*

Name	See the Booking
Actors	Registered user
Entry Conditions	User is on user page
Flow of Events	User presses on the button “See Booking”
Exit Conditions	User booked a car system show the booking
Exceptions	User didn't book any car

### *Definition of “Interacting with Booked Car”*

Name	<i>Interacting with Booked Car</i>
------	------------------------------------

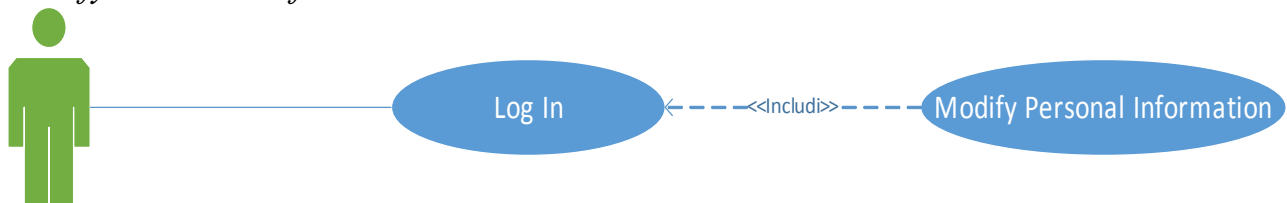


Actors	Registered user
Entry Conditions	User did successfully "see your booking" use case
Flow of Events	User presses on the button “rent it”; Or user presses on the button "show map"
Exit Conditions	If user presses "rent it" he is sent to "Renting Car" use case; If user pressed "show map" he is sent in the map, centred in the car's address.
Exceptions	No exception triggered.

### *Definition of “Delete a Booking”*

Name	Delete a Booking
Actors	Registered user
Entry Conditions	User must be logged in
Flow of Events	User presses on the button “DELETE BOOKING”; At the system question “Do you really want to delete this reservation?”, the user press “YES” or "NO";
Exit Conditions	If pressed “YES”, the reservation is deleted and user is sent to the user home; If pressed “NO”, user is sent to the previous page;
Exceptions	The system, for technical reasons, is not available; User doesn't booked any car;

### *Modify Personal Information*



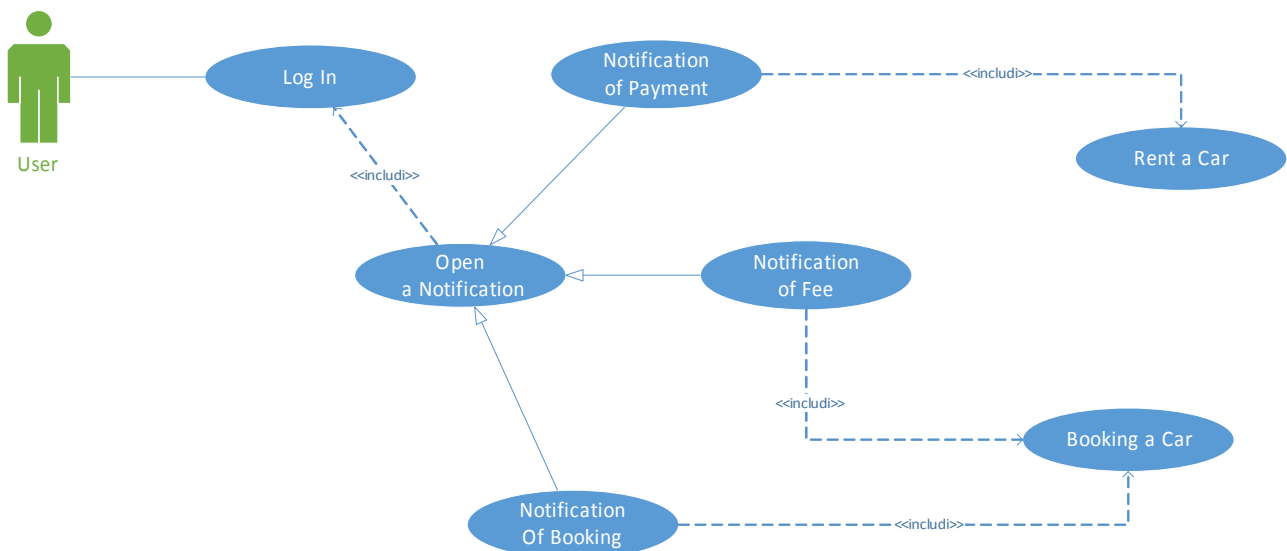
This little graph shows an useful feature for registered user: the modification of

personal information.  
Below the description of the use case.

### *Definition of “Modify Personal Information”*

Name	Modify Personal Information
Actors	Registered user
Entry Conditions	User must be logged in
Flow of Events	User presses on the button “Modify Personal Information”; User selects the information to modify; User inserts new information; If there is nothing else to modify, the user presses on the button “Save and Exit”
Exit Conditions	The system saves new information in the databases and user returns to the main page;
Exceptions	Any information entered is not correct

### *Notifications*



In this graph we analyse how the system provides to the user feedback about his actions and more information about payment. This notification system is divided in three category: notification about the payment and related informations (discount, raises), notification about fees and notification about booking.

Below the description of the use cases.  
*Definition of “Receive a Notification”*

Name	Receive a Notification
Actors	Registered user
Entry Conditions	The system sends a notification to the user
Flow of Events	User sees the notification sent; User presses on the "Report" button; User visualize correctly the list of notifications received;
Exit Conditions	User pushes the button “MENU” to return to the user home of the website;
Exceptions	The system didn't sent any notification to the user, so he cannot press on the relative button

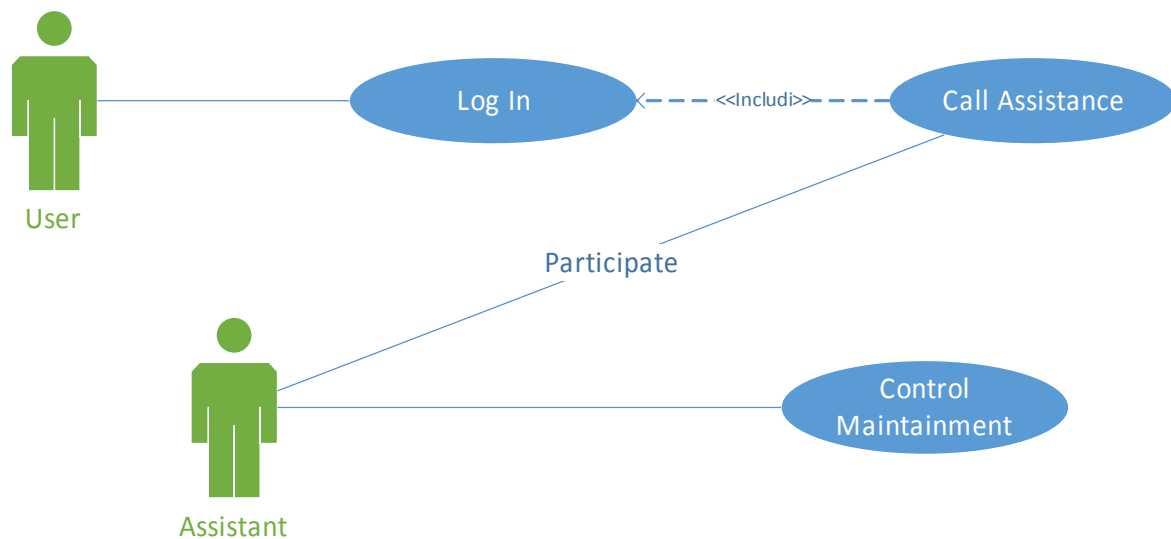
*Definition of “Notification of Fee”*

Name	Notification of Fee
Actors	Registered user
Entry Conditions	User doesn't rent a car or doesn't cancel the booking up to an hour after the “Booking a Car” use case
Flow of Events	User receives a notification from the system; User presses on the letter-shape button; User visualizes correctly a notification named “1h FEE”;
Exit Conditions	User pushes the button “MENU” to return to the user home of the web site;
Exceptions	No exceptions

### Definition of “Notification of Payment”

Name	Notification of Payment
Actors	Registered user
Entry Conditions	The user has just finished the renting
Flow of Events	<p>System elaborates the payment applying eventually discounts or raises;</p> <p>System add a fee if the user doesn't leave the car in safe area;</p> <p>System sends notification;</p> <p>The user receives a notification from the system;</p> <p>The user presses on the letter-shape button;</p> <p>The user views successfully a notification named “Notification of Payment” with all details about how much it paid and the discounts or raises it obtained</p>
Exit Conditions	The user presses on the button “OK” to return to the main page of the web site
Exceptions	No exceptions

### Assistance



We have decided also to give an assistance to our users, in order to resolve their

problems. Also, the assistant has a proper use case called “Control Maintenance”. Below the description of the use case.

*Definition of “Contact Assistance”*

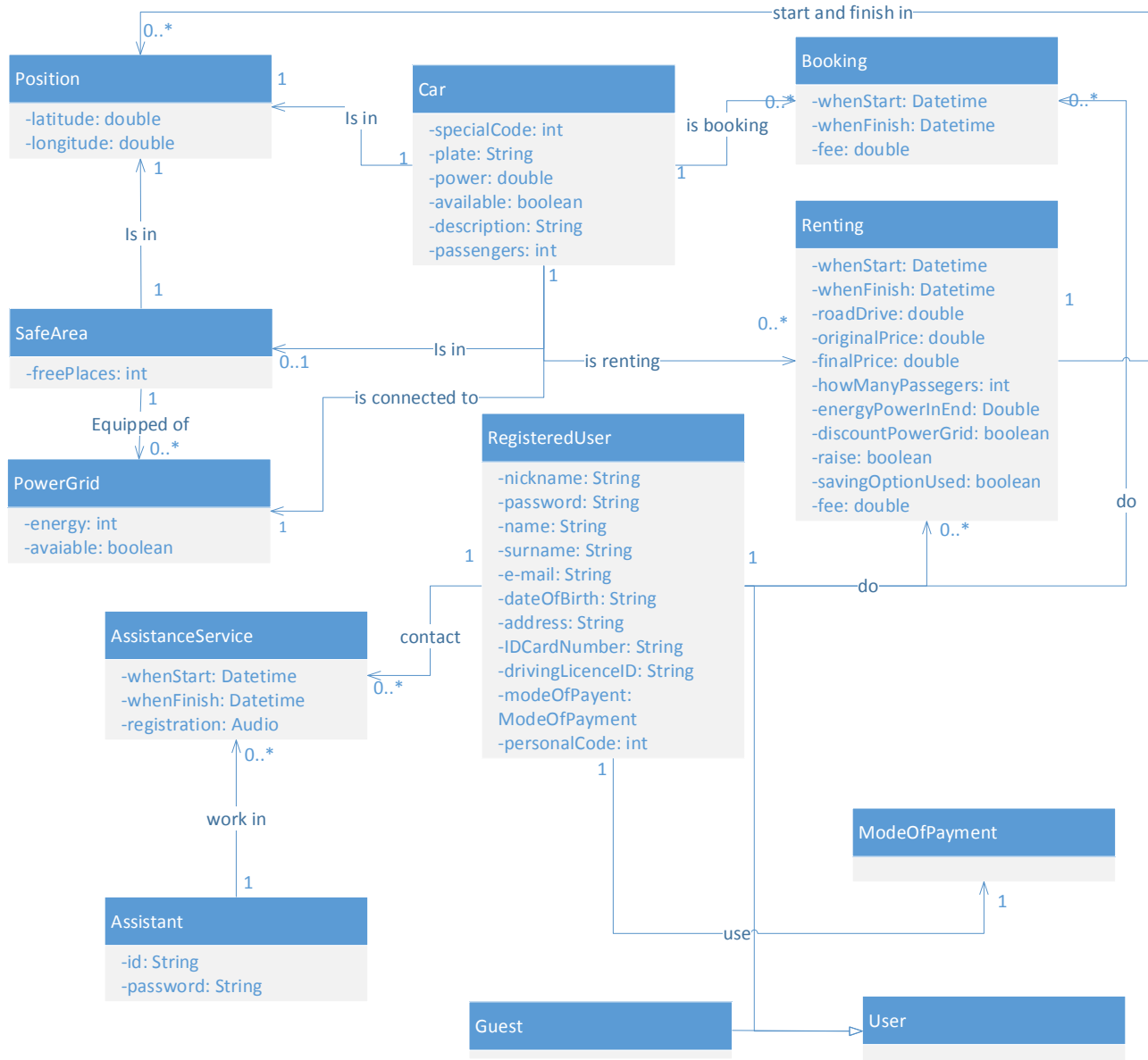
Name	Contact Assistance
Actors	Registered user, Assistant
Entry Conditions	The user has successfully logged in
Flow of Events	User clicks in "Contact Assistance" button; System starts a call between the user and an assistant
Exit Conditions	An assistant answer to user call
Exception	No assistant available

*Definition of “Control Maintenance”*

Name	Control Maintenance
Actors	Assistant
Entry Conditions	No entry conditions
Flow of Events	The assistant reach the place of the car to be examined; The assistant control if the car is dirty or has been damaged;
Exit Conditions	The assistant reports the condition of the car
Exception	No assistant available

## 5.2 Class Diagram

Now that we see all use case we can draw the class diagram that show how the main object of our system interact each other.



### Observation

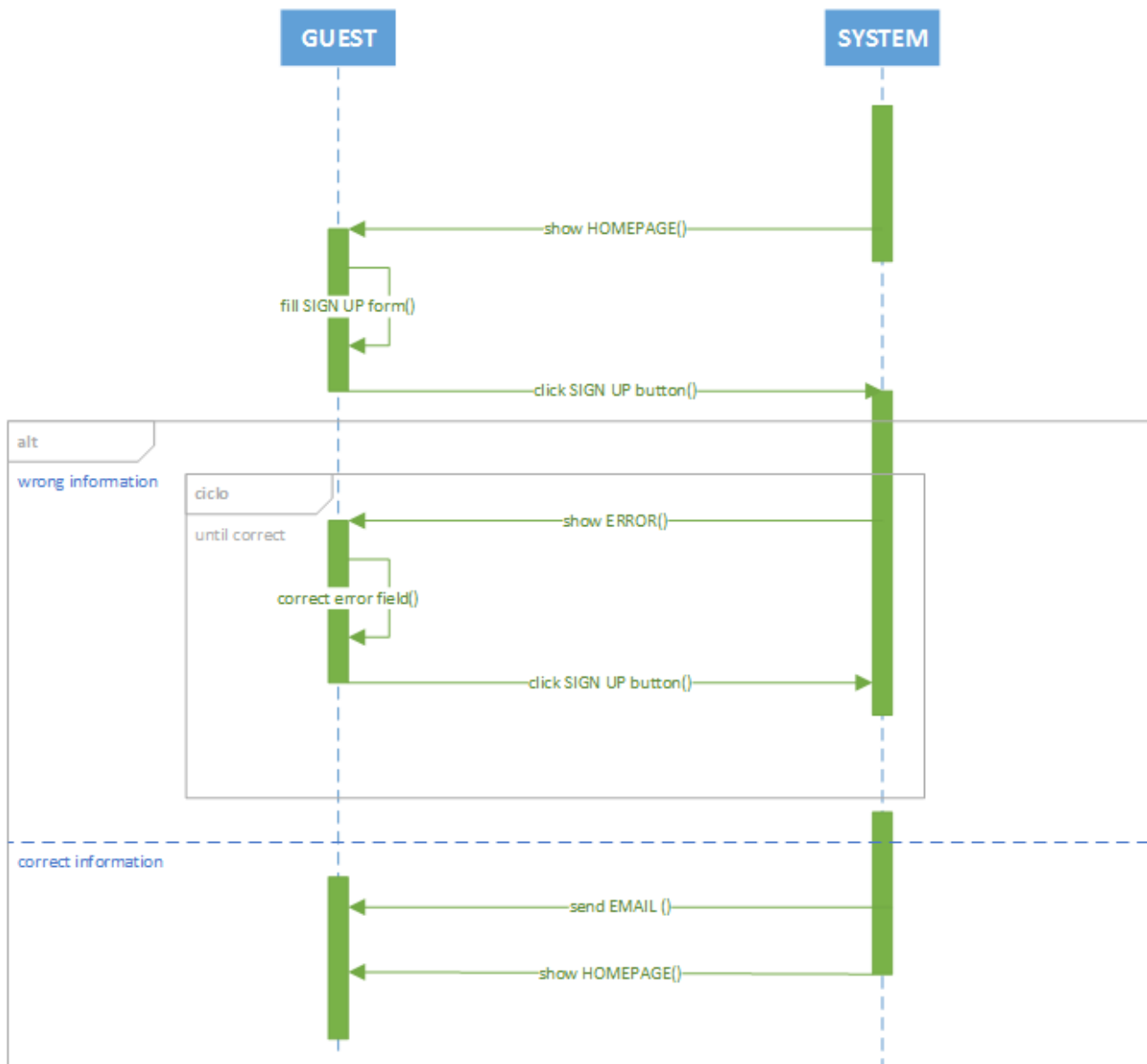
- when user rents a car a "Renting object" is created by the system; whenStart is set when user start the renting, the other value (like whenFinish or cost) in the beginning are null, they will be set when the user will finish the renting.
- the same procedure described above is applied for "Booking object" and "ContactAssistance object".
- discountPowerGrid in "Renting" is set TRUE if the user connects the car to a power grid when he finishes the renting.

- a power grid is available if no car is connected and energy is more than 0.

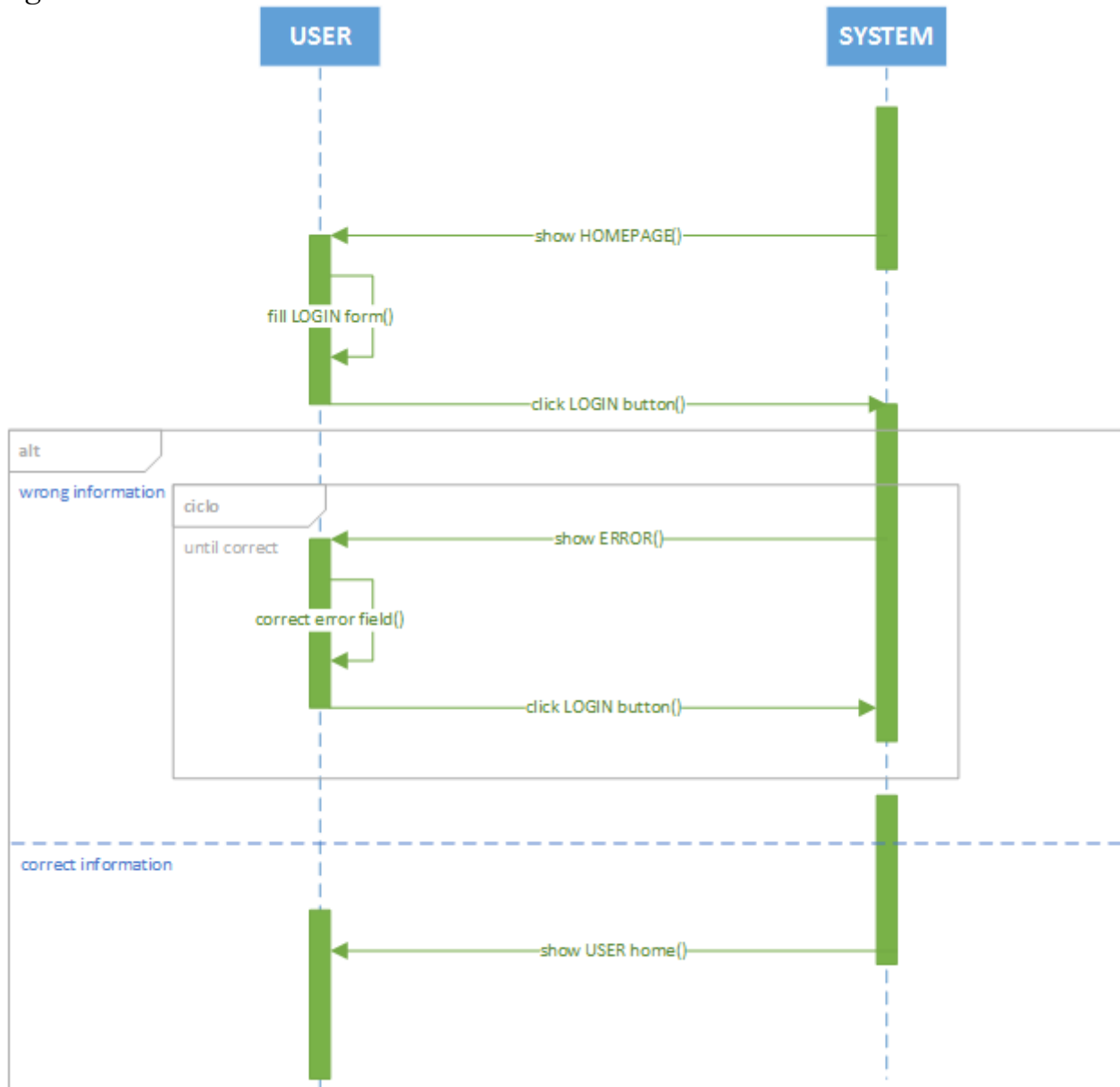
### 5.3 Sequence diagrams

Below we have realized the sequence diagram of the main action described in the use cases above, both of the web-application and of the system inside the car.

*Sign up*

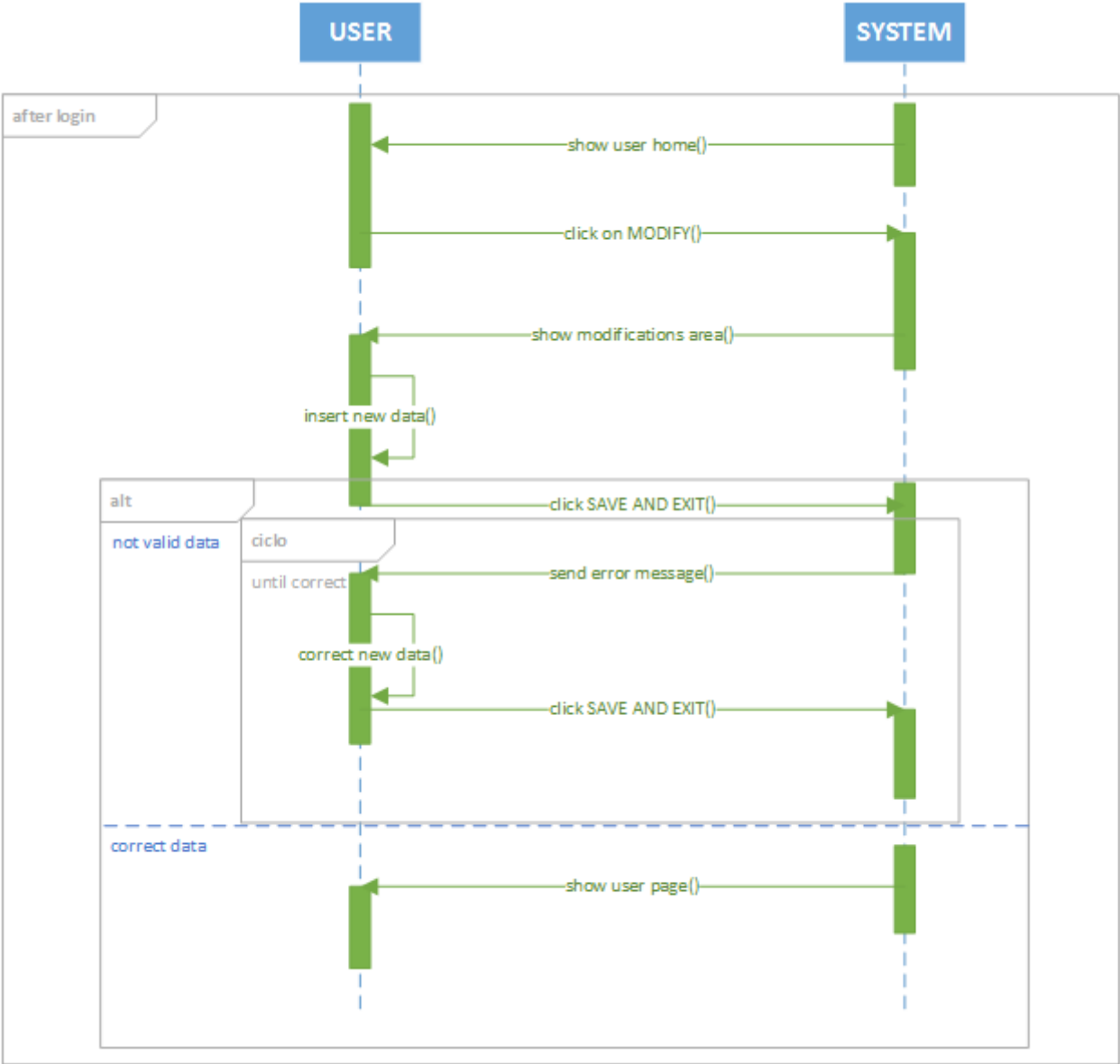


## Login

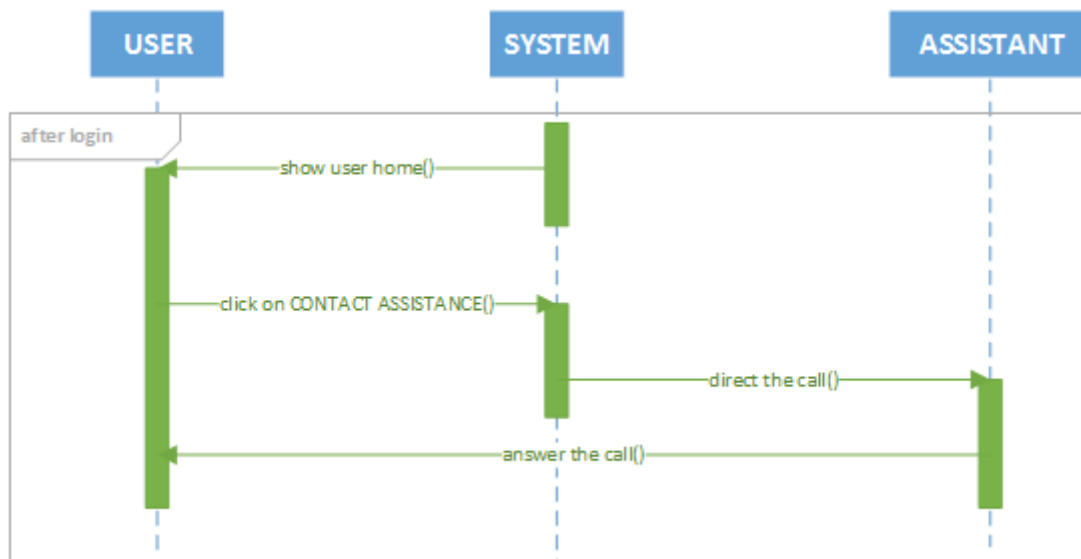




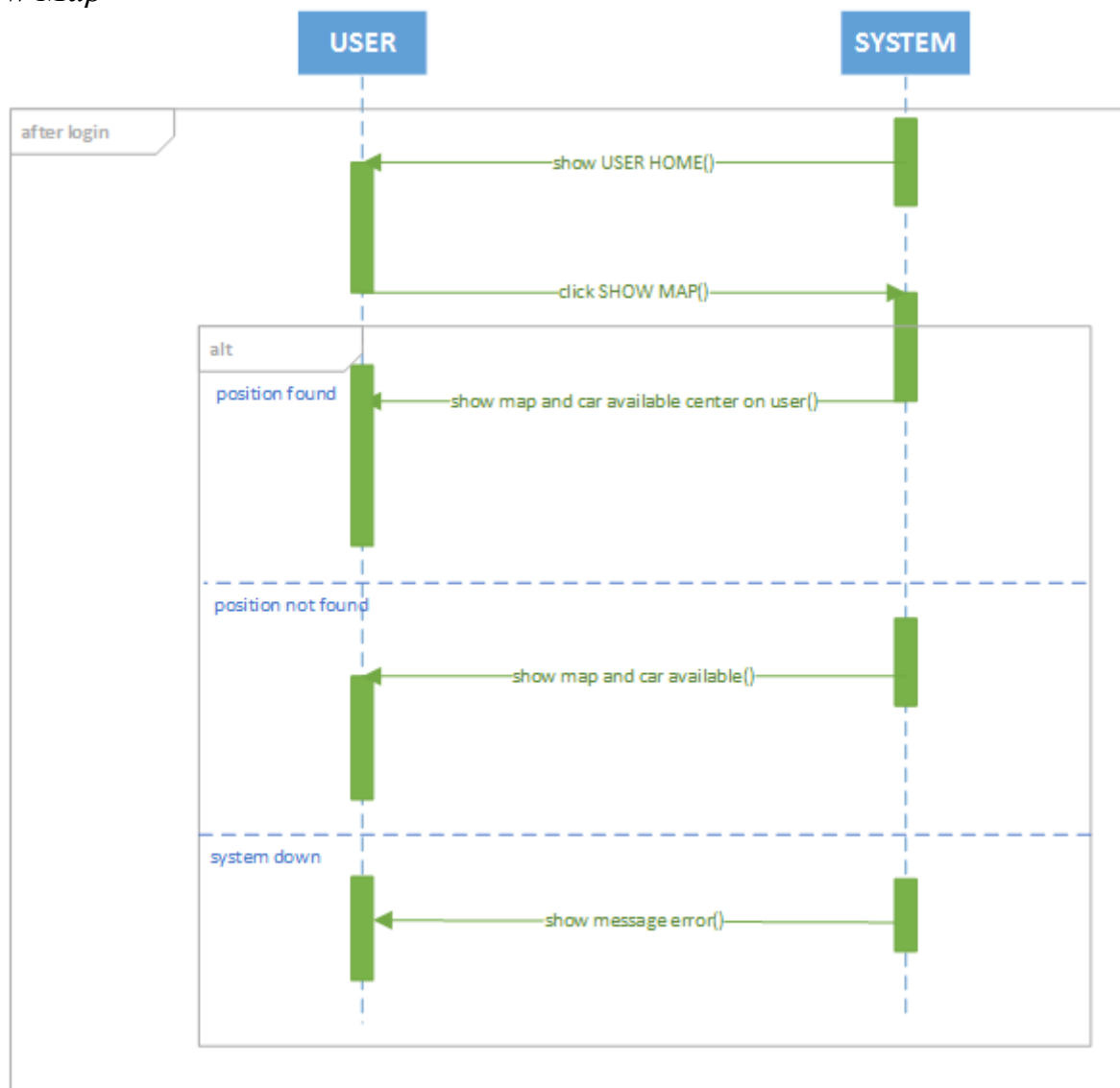
*Modify Data*



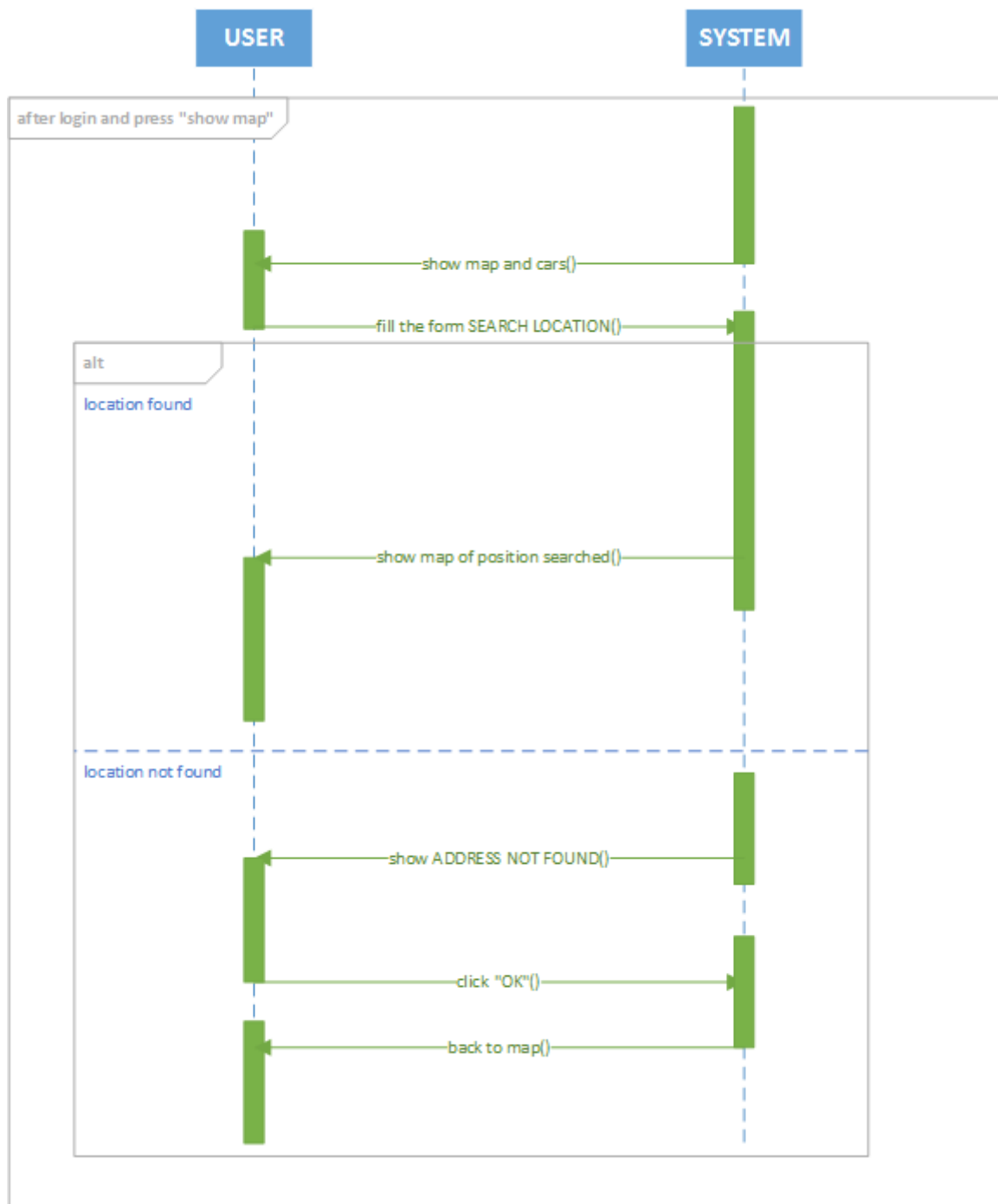
*Contact Assistance*



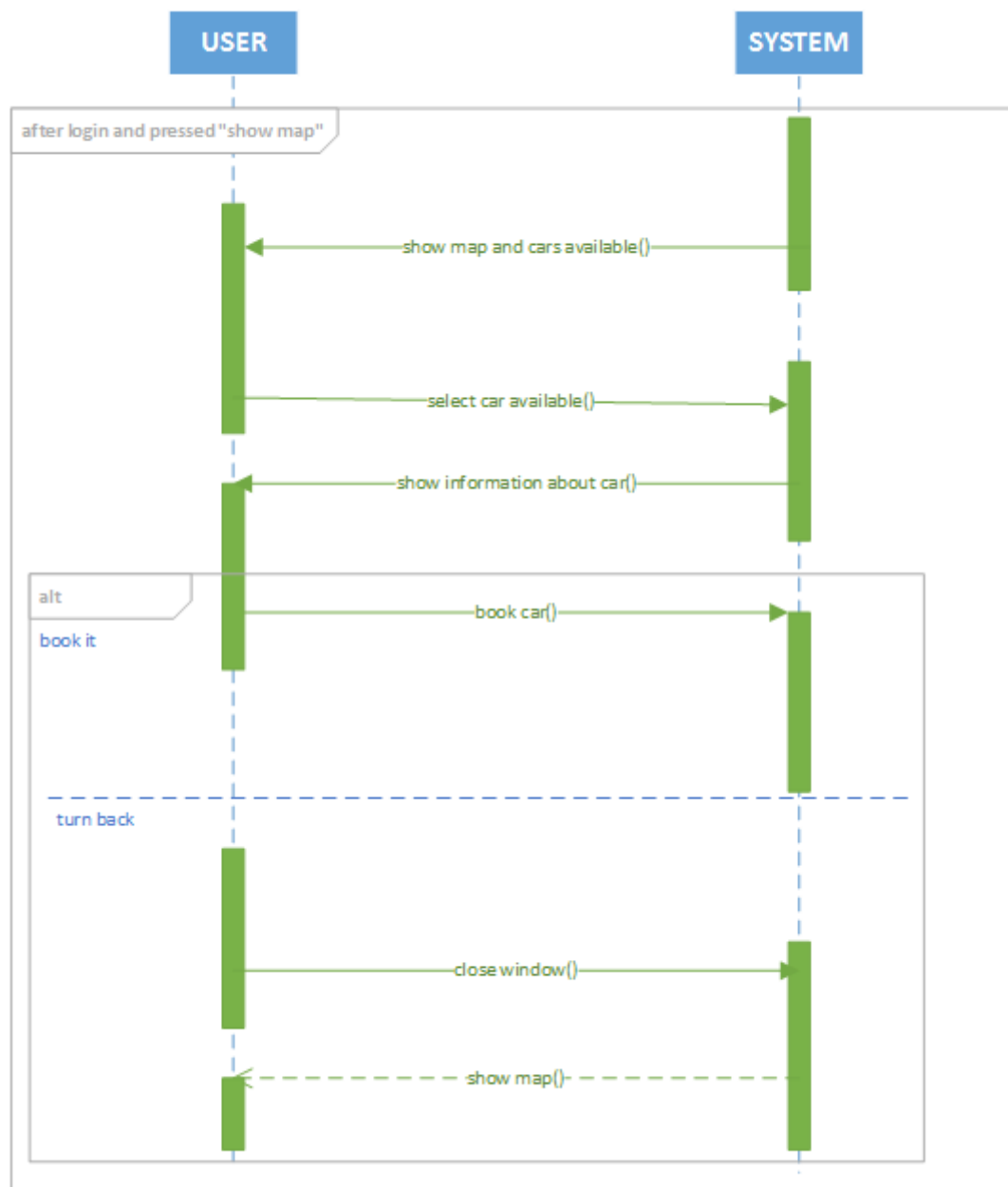
### Show Map



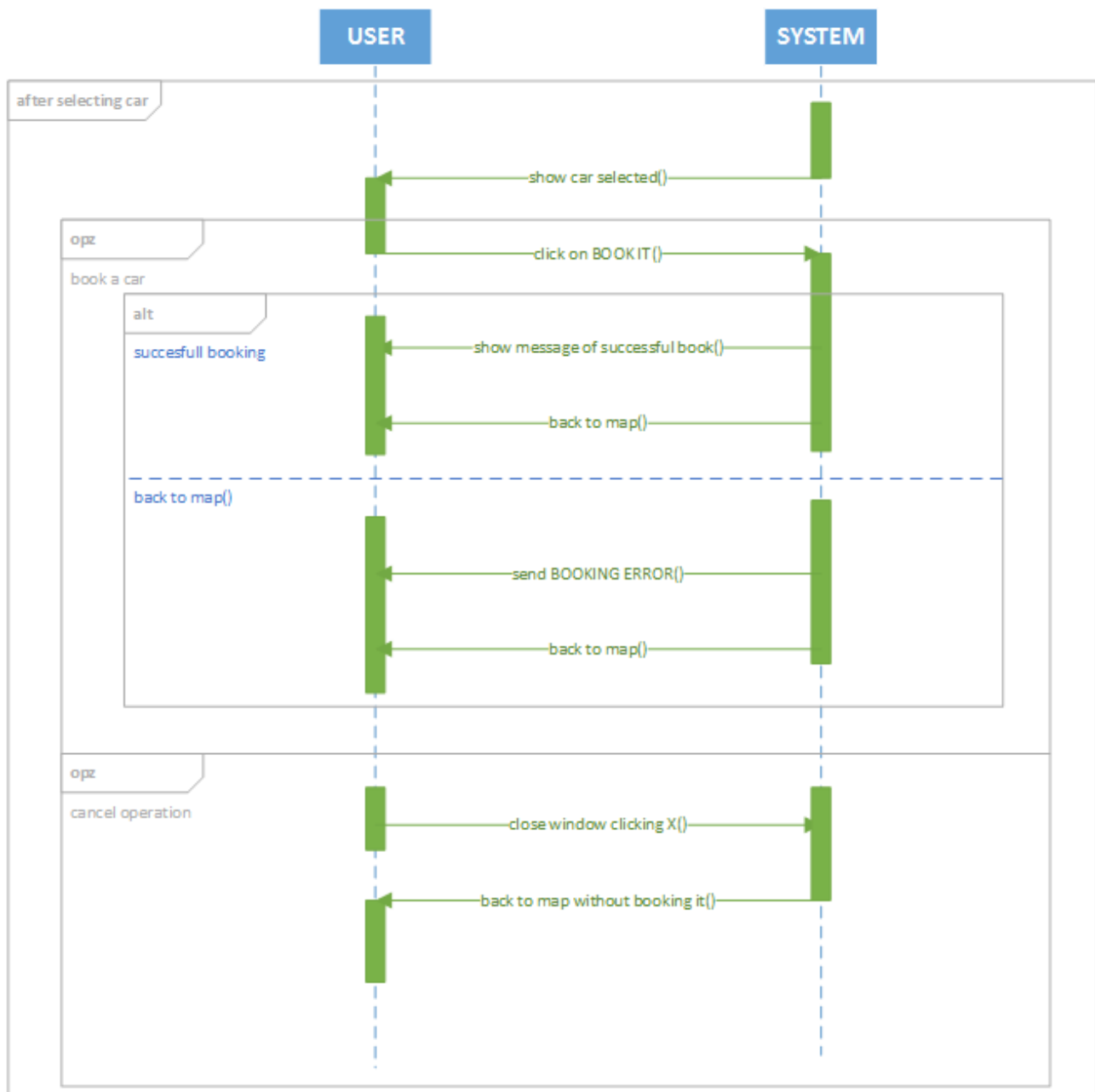
*Search Location*



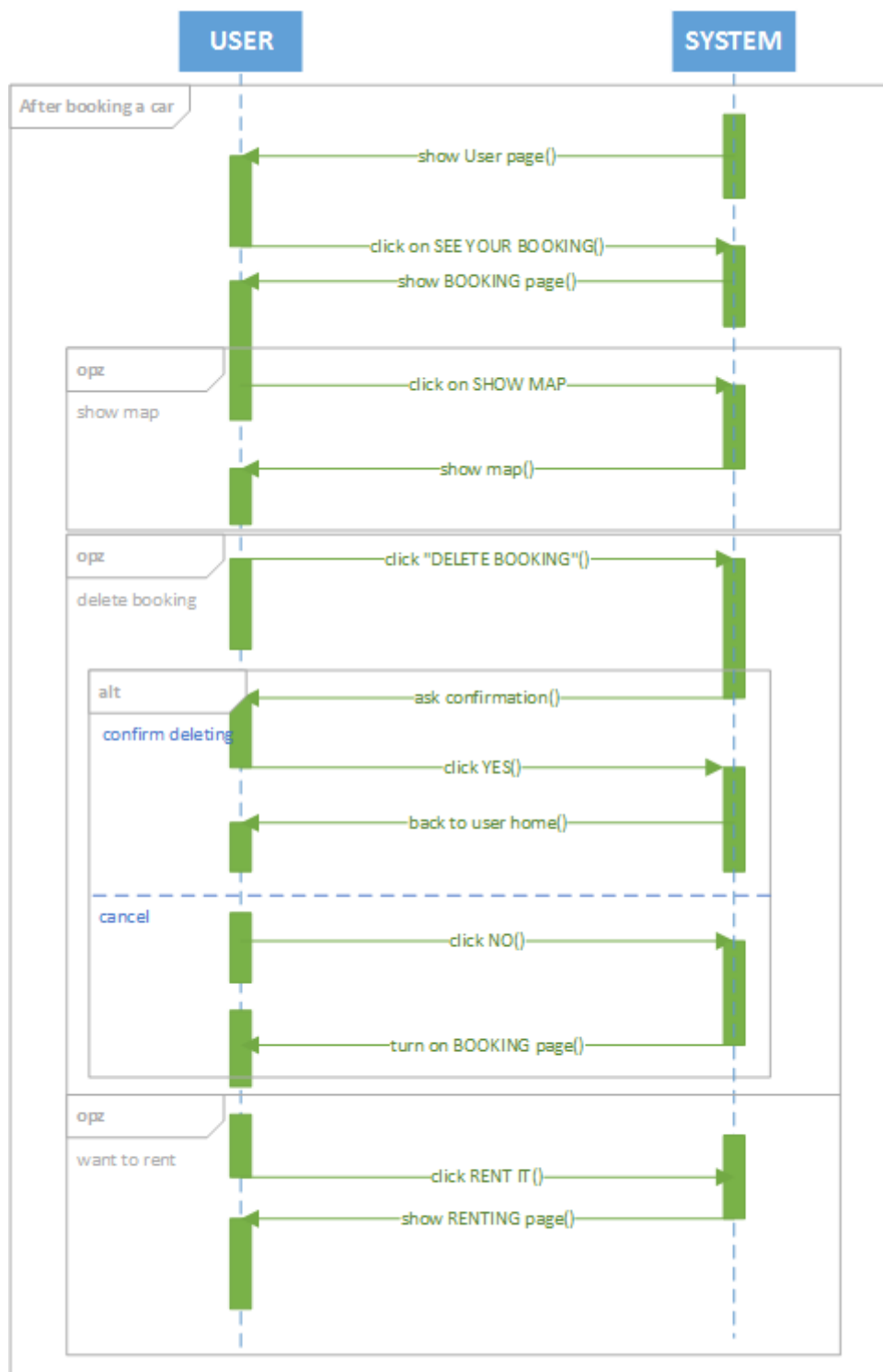
*Select a Car*

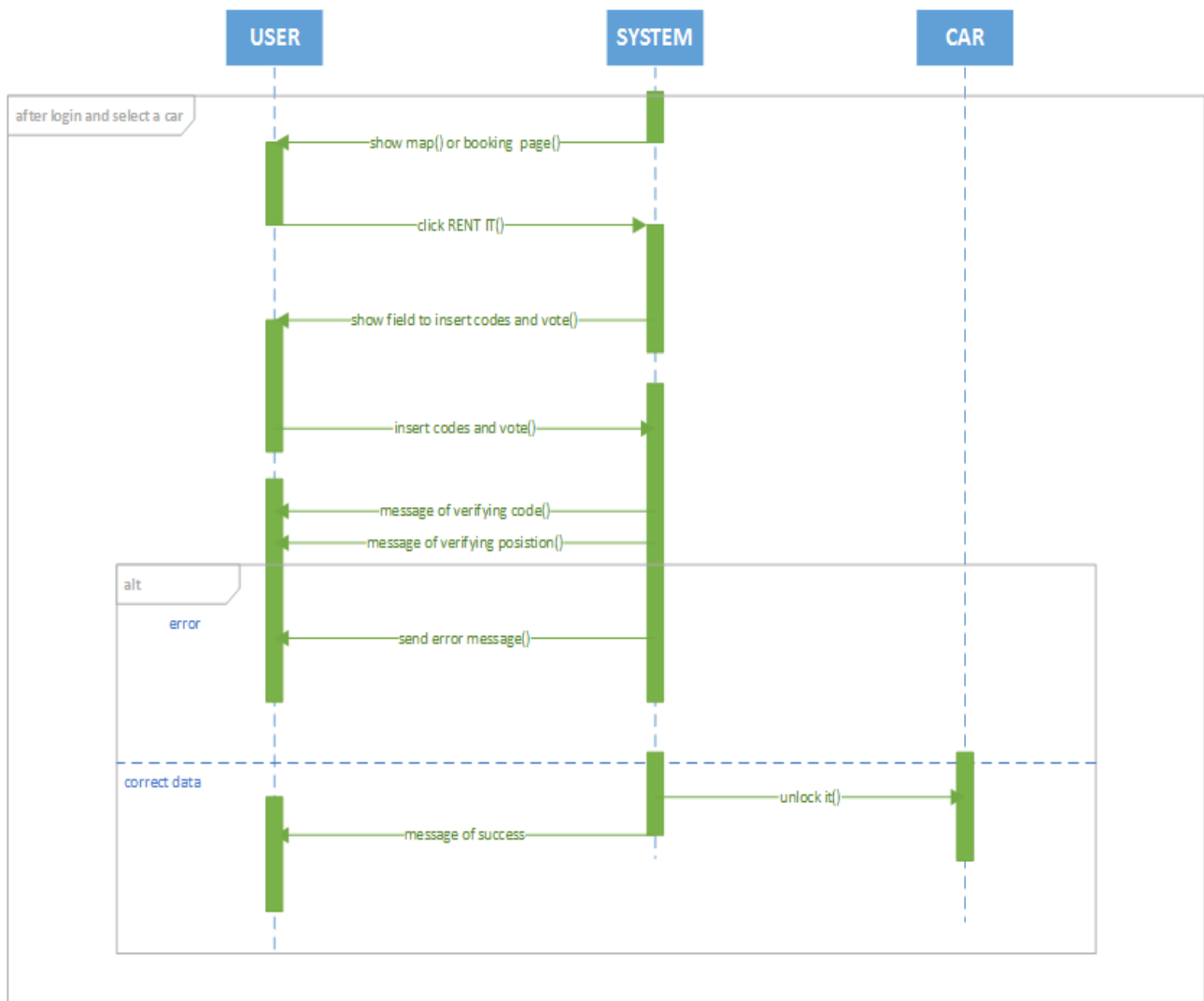


*Booking a car*



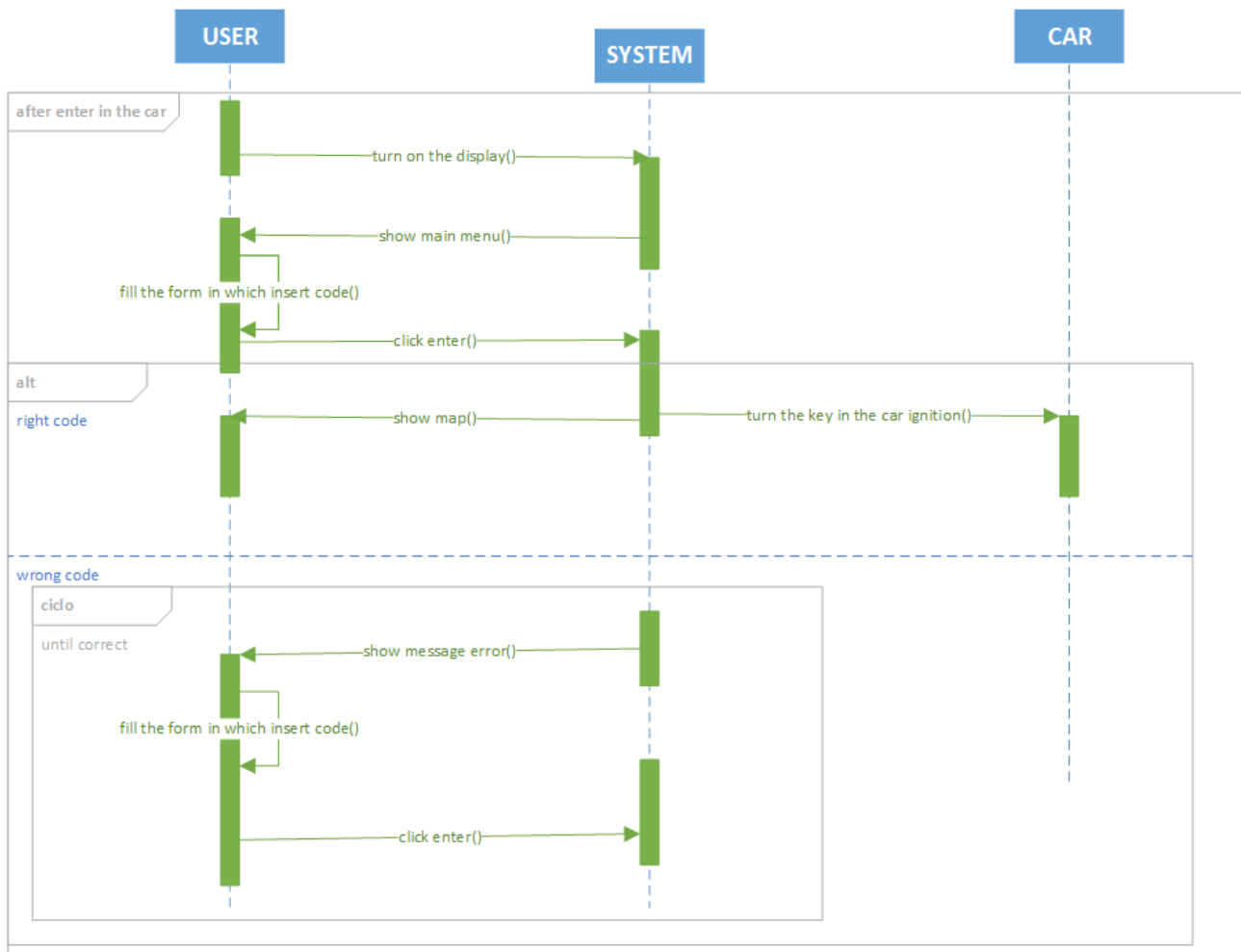
*See your booking/Delete booking*



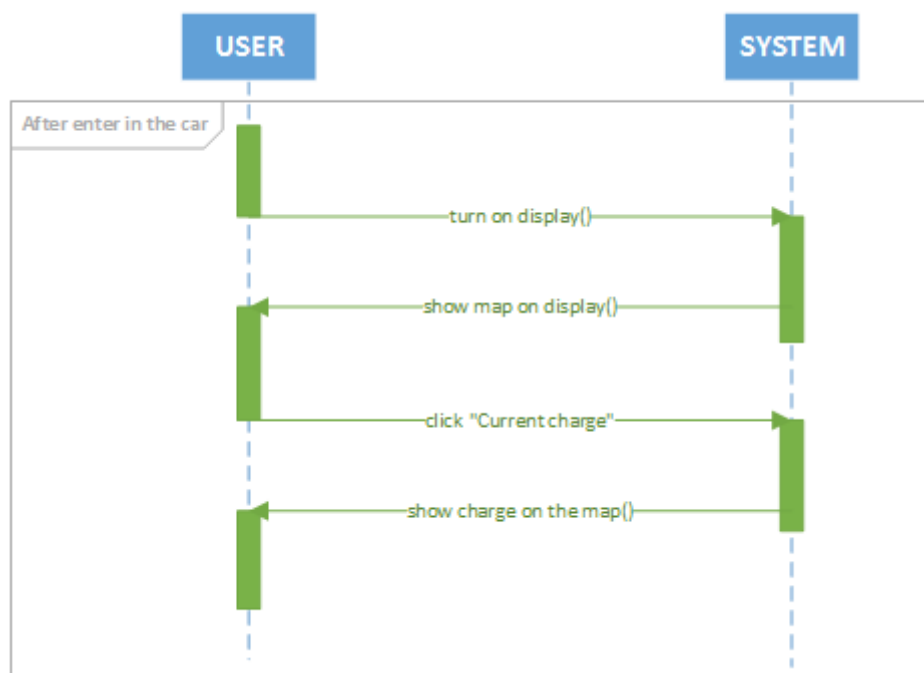


*Turn on Car*

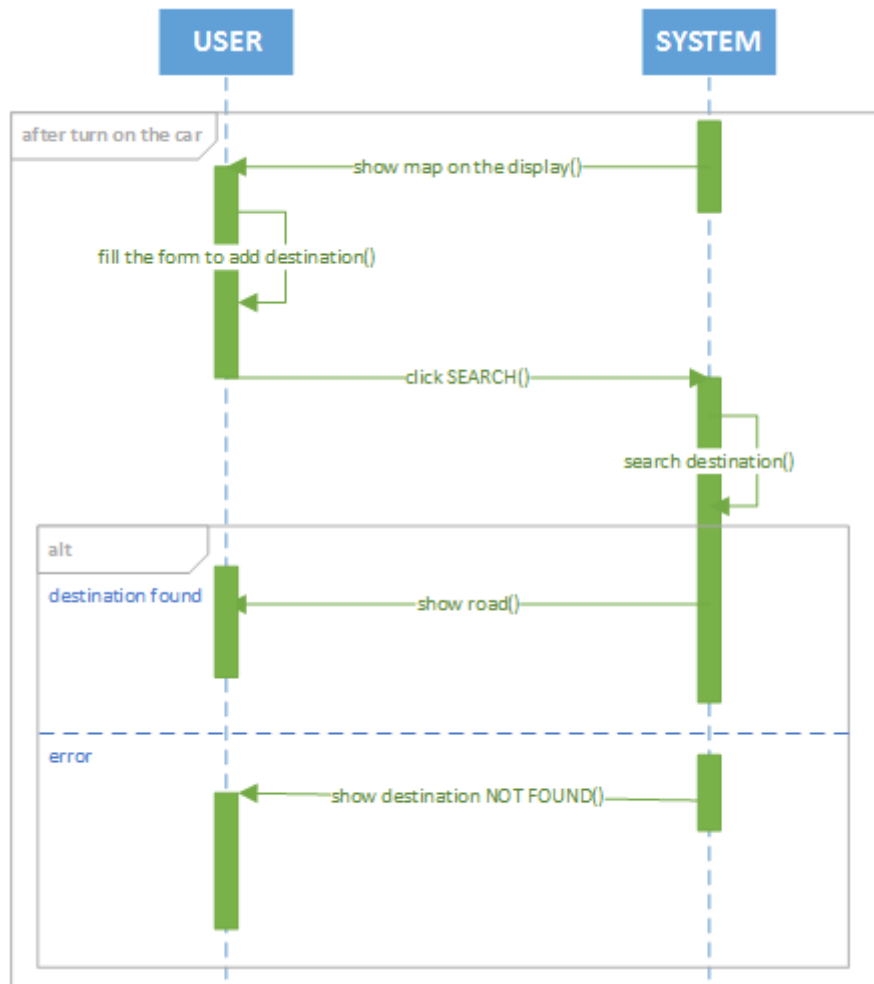




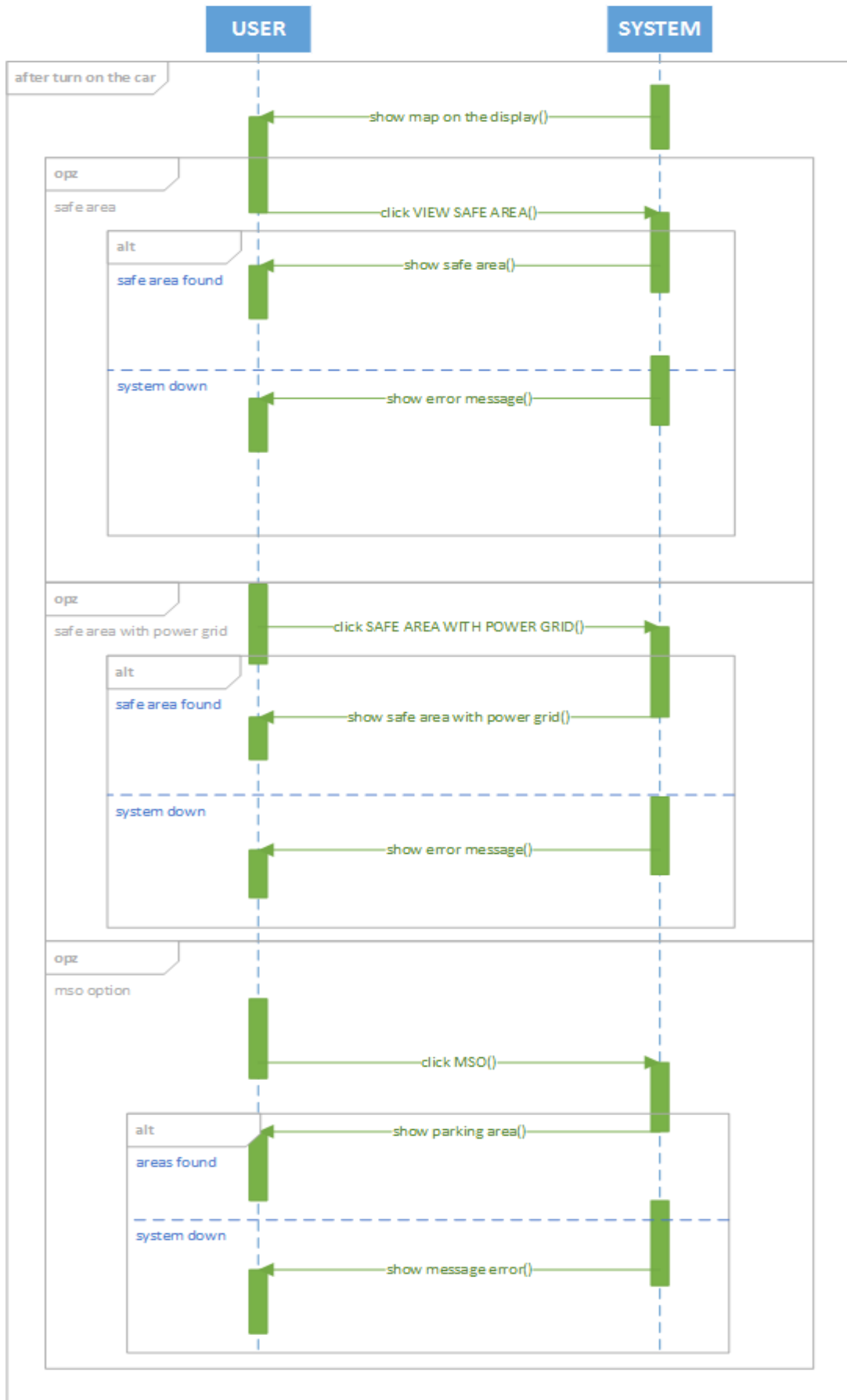
*Visualize charge on car display*



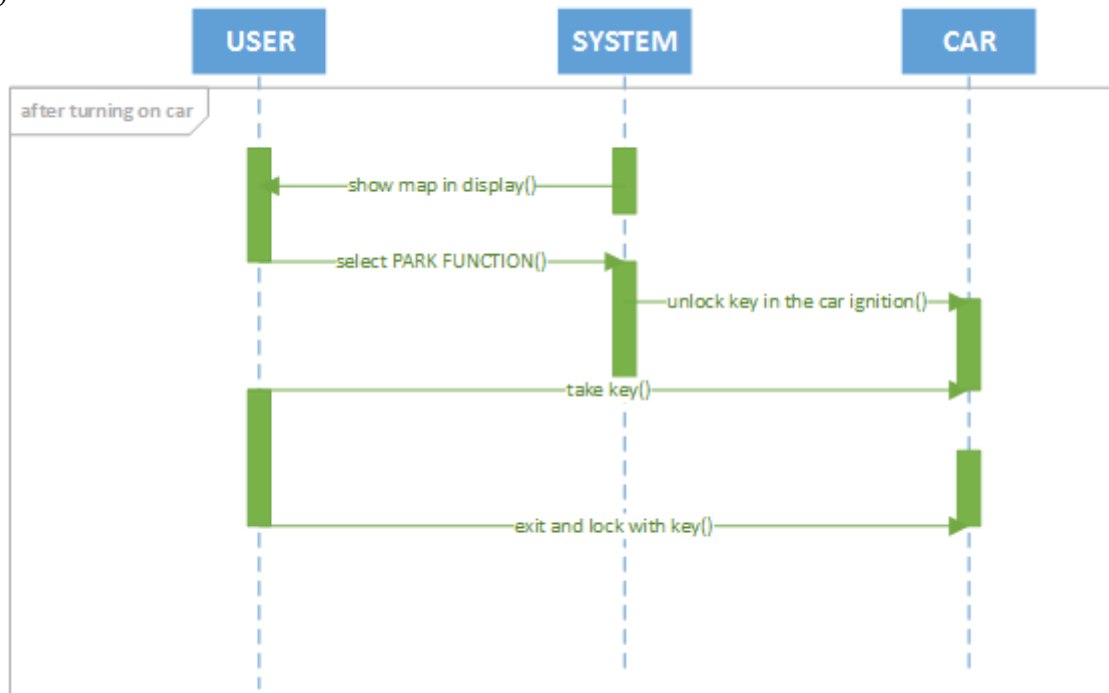
## *Insert address on car display*



## Safe area, power grid and MSO on car display

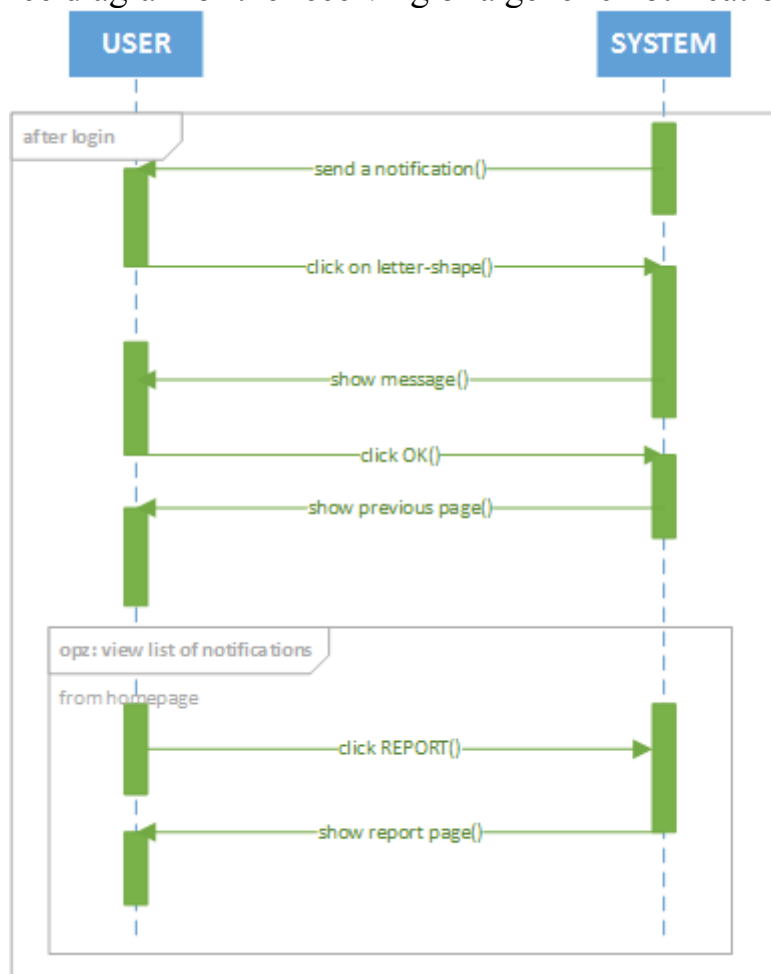


### *Park function*



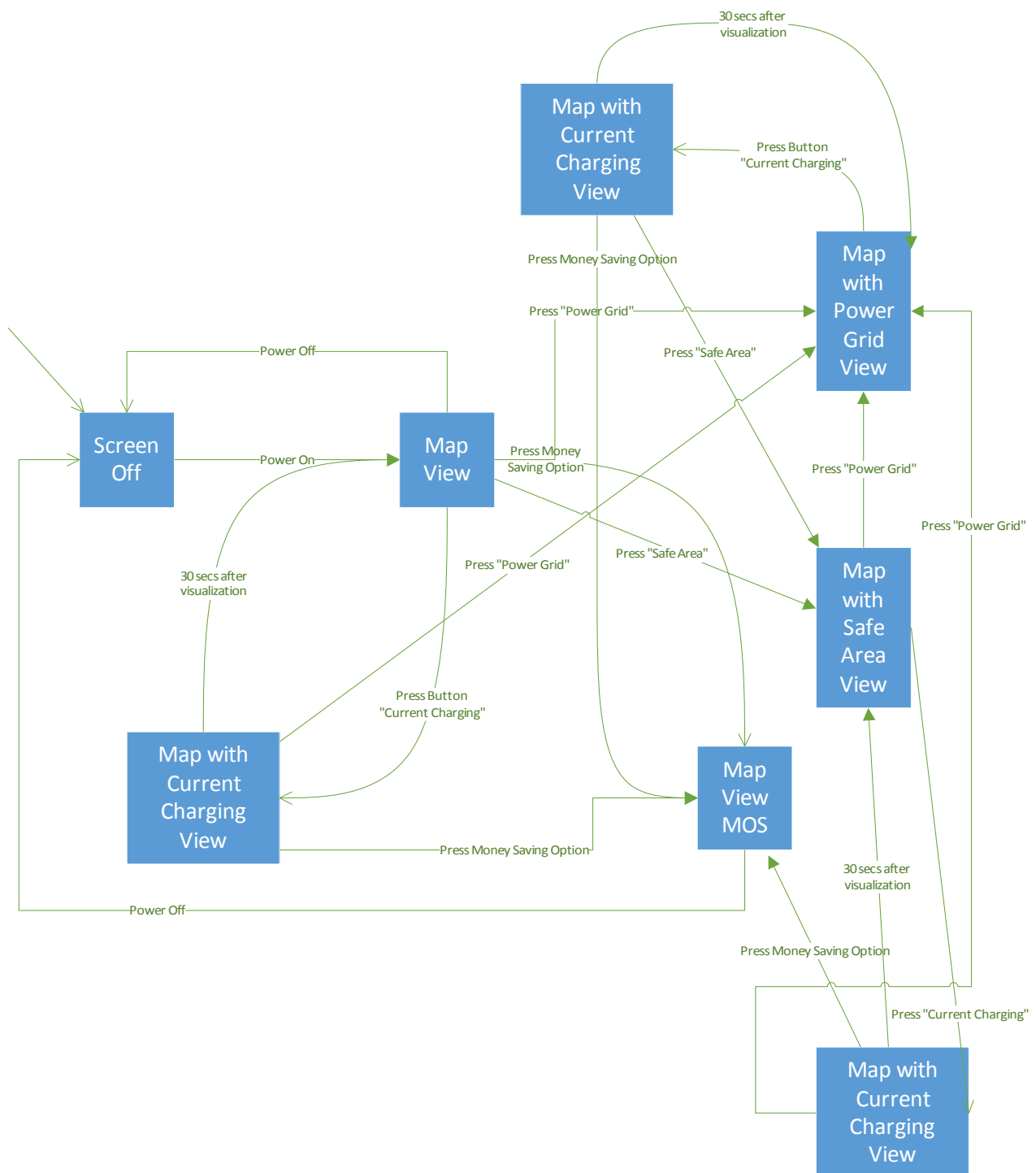
### *Receive a Notification*

Below the sequence diagram of the receiving of a generic notification.



## 5.4 State Chart Diagram

We want also to provide a graph showing how the services provided inside the car are shown and managed. This because the car-system is not a third party one, but a part managed by the main system



## **6 ALLOY MODELLING**

### **6.1 Alloy Code**

```
sig RegisteredUser{
    nickname: one string,
    email: one string
}

sig string {}

sig ID {}

abstract sig boolean {}

one sig True extends boolean {}

one sig False extends boolean {}

sig Float{
    left: Int,
    right: Int
}
{left >= 0 and right >= 0 and right < 100} -- precision of 2 decimal --

sig Position{
    latitude: one Float,
    longitude: one Float,
}

sig PowerGrid{
    energy: one Float,
    available: one boolean,
    car_linked: one boolean,
} {(available = True <=> car_linked = False) || (available = False <=> car_linked = True)}

sig SafeArea{
    position: one Position,
    freePlaces: one Int,
    powerGrid: set PowerGrid,
    car: set Car
} { #car >= 0 and freePlaces >= 0 }
```

```

sig Car{
    power: one Float,
    available: one boolean,
    passengers: one Int,
    position: one Position,
    nearestPowerGridStation: one Float,
}
{power.left >= 0 and power.left <= 100 and passengers >=1}

sig Booking{
    user: one RegisteredUser,
    minutesDuration: one Int,
    fee: one Float,
    car: one Car,
} {#car = 1 and (minutesDuration<60 <=> fee.left=0 and fee.right=0) and
minutesDuration>0}

sig Renting{
    user: one RegisteredUser,
    car: one Car,
    originalPrice: one Float,
    finalPrice: one Float,
    discountPassenger: one Float,
    discountBattery: one Float,
    discountPowerGrid: one Float,
    raise: one Float,
    savingOptionUsed: one boolean,
}

sig Assistant{
    id: one ID
}

sig AssistanceService{
    user: one RegisteredUser,
    assistant: one Assistant
}

--FACT--
--In the same time T can't exist 2 AssistanceService with the same user or assistant
fact AssistanceService{
    all disj as1, as2: AssistanceService | (as1.user != as2.user and as1.assistant !=

```

```

as2.assistant)
}
--There aren't assistants with the same id
fact UniqueAssistant{
    all a1,a2: Assistant | (a1!=a2 => a1.id != a2.id)
}
--There aren't users with the same nickname
fact UniqueRegisteredUser{
    all u1,u2: RegisteredUser | (u1!=u2 => (u1.nickname != u2.nickname and
u1.email!=u2.email))
}
--For any renting the car and the user exist
fact RentingWithAUser{
    all r: Renting | ( one u: RegisteredUser, c: Car | r.user = u and r.car = c)
}
--For any booking the car and the user exist
fact BookingWithAUserAndACar{
    all b: Booking | ( one u: RegisteredUser, c: Car | b.car = c and b.user = u)
}

fact CarForBookingSameForRenting{
    all b: Booking, r: Renting | (r.user != b.user and r.car != b.car)
}

fact NoRentingShareSameUsersandCars{
    all disj r1,r2: Renting | (r1.user != r2.user and r1.car != r2.car)
}

fact NoBookingShareSameUsersandCars{
    all disj b1,b2: Booking | (b1.user != b2.user and b1.car != b2.car)
}

-- Only the registered user can book --
fact onlyRegisteredUserCanBook{
    all b: Booking | b.user in RegisteredUser
}

-- Power Grid is available if no car is connected and energy is more than 0% --
fact availablePowerGrid{
    all p: PowerGrid | (p.available = True ) <=>
    (p.energy.left >= 0 and p.energy.right > 0)
}

```



```

fact NoSafeAreasOverlapped{
    all disj s1,s2: SafeArea | (s1.position != s2.position)
}
--The same car can't be situated in different safe areas in the same time
fact NoSafeAreasSameCars{
    all disj s1,s2: SafeArea | (s1.car != s2.car)
}
--The same power grid can't be situated in different safe areas
fact NoSafeAreasSharedPowerGrid{
    all disj s1,s2: SafeArea | (s1.powerGrid != s2.powerGrid)
}

--If the final and original price are equal the discounts are equal to raises
fact FinalPriceNoSameOriginalPrice{
    all r: Renting | ((r.originalPrice != r.finalPrice) ||
    (r.originalPrice = r.finalPrice and r.discountPassenger.right = 10 and
r.raise.right = 30
    and r.discountBattery.right = 20) )
}

--If a user connected the car in a power grid and leave the car with more than 20% of
energy he/she doesn't have raise
fact DiscountAndRaise{
    all r: Renting | (r.discountBattery.right>0 and r.discountPowerGrid.right >0
    <=> (r.raise.left = 0 and r.raise.right = 0))
}

fact CarInAreaSharePosition{
    all s:SafeArea | (s.car.position = s.position)
}

fact CarNotOverlapped{
    all disj c1,c2: Car | (c1.position != c2.position)
}

fact PowerGridMustBeInSafeArea{
    all p: PowerGrid | (one s: SafeArea | (s.powerGrid=p))
}

fact CarAvailable{
    all c: Car, r: Renting, b: Booking | c.available=True <=> (r.car != c and b.car !=c)
}

```

```

fact CarNotAvailable{
    all c: Car, r: Renting, b: Booking | c.available=False <=> (r.car = c or b.car=c)
}
--ASSERTION--

-- If the system detects the user took at least two other passengers onto the car, the
system applies a discount --
-- of 10% on the last ride --
assert DiscountPassengers{
    all r: Renting | (r.discountPassenger.left = 0 and r.discountPassenger.right = 10
and r.raise.left = 0 and r.raise.right = 0) =>
    ((r.finalPrice.left < r.originalPrice.left) || (r.finalPrice.left = r.originalPrice.left
and r.finalPrice.right < r.originalPrice.right))
}

-- If a car is left at a special parking areas where they can be recharged and the user
takes care of --
--plugging the car into the power grid, the system applies a discount of 30% on the
last rid --
assert DiscountPowerGrid{
    all r: Renting | (r.discountPowerGrid.left = 0 and r.discountPowerGrid.right =
30 and r.raise.left = 0 and r.raise.right = 0) =>
    ((r.finalPrice.left < r.originalPrice.left || (r.finalPrice.left = r.originalPrice.left
and r.finalPrice.right < r.originalPrice.right))
    and (r.car.power.left > 20))
}

assert DiscountPowerBattery{
    all r: Renting | (r.discountBattery.left = 0 and r.discountBattery.right = 20 and
r.raise.left = 0 and r.raise.right = 0) =>
    ((r.finalPrice.left < r.originalPrice.left || (r.finalPrice.left = r.originalPrice.left
and r.finalPrice.right < r.originalPrice.right))
    and (r.car.power.left >= 50))
}

assert RaiseOnPrice{
    all r: Renting | (r.raise.left >= 0 and r.raise.right > 0) =>
    ((r.finalPrice.left = r.originalPrice.left and r.finalPrice.right =
r.originalPrice.right) ||
    ((r.car.power.left <= 20) || (r.car.nearestPowerGridStation.left >= 3
and r.car.nearestPowerGridStation.right >= 0)))
}

```

```
--PREDICATE--
pred showRentingandBooking {
    #RegisteredUser = 3
    #Renting = 1
    #Booking = 1
    #SafeArea = 0
    #PowerGrid = 0
    #Assistant = 0
    #ID = 0
}
```

```
run showRentingandBooking
```

```
pred showCarAndAreas{
    #SafeArea = 2
    #PowerGrid = 2
    #Car = 2
    #Booking = 0
    #Renting = 0
    #RegisteredUser = 0
    #Assistant = 0
    #ID = 0
    #string=0
}
```

```
run showCarAndAreas
```

```
pred showUserAndAssistant{
    #RegisteredUser = 3
    #Assistant = 3
    #Car = 0
    #Booking = 0
    #Renting = 0
    #SafeArea = 0
    #PowerGrid = 0
}
```

```
run showUserAndAssistant
```

```
pred show{}
```

```
run show
```

check DiscountPassengers

check DiscountPowerGrid

check DiscountPowerBattery

check RaiseOnPrice

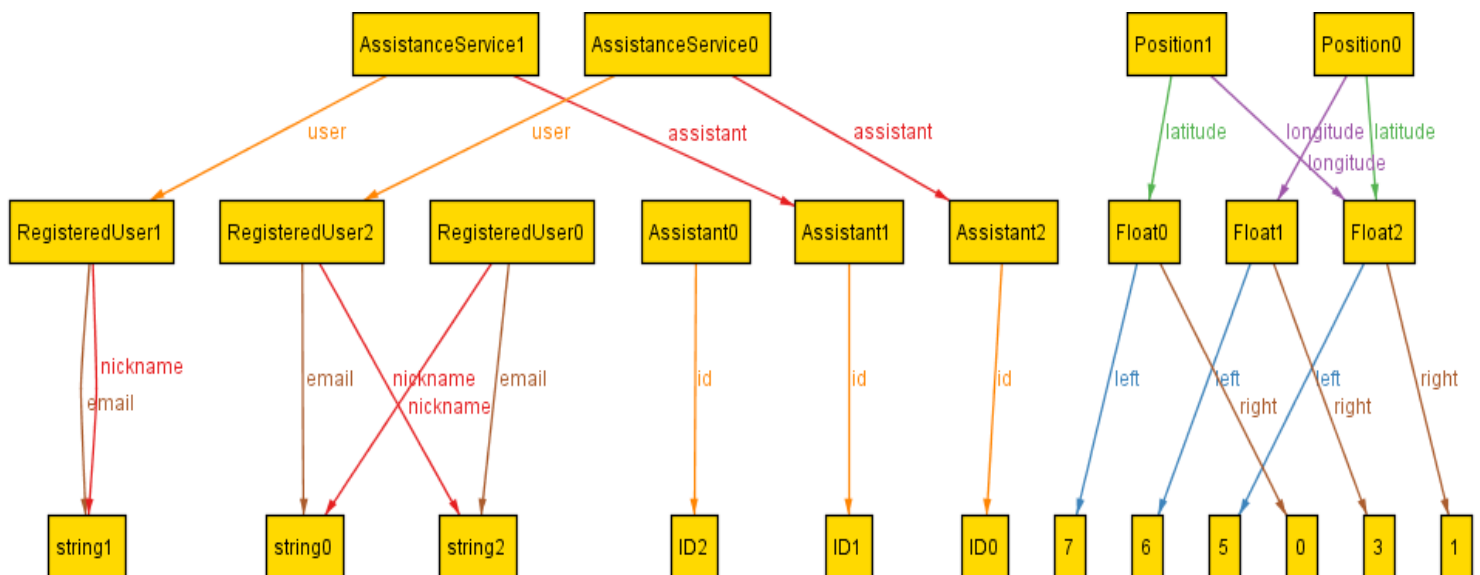
## 6.2 Alloy result

**8 commands were executed. The results are:**

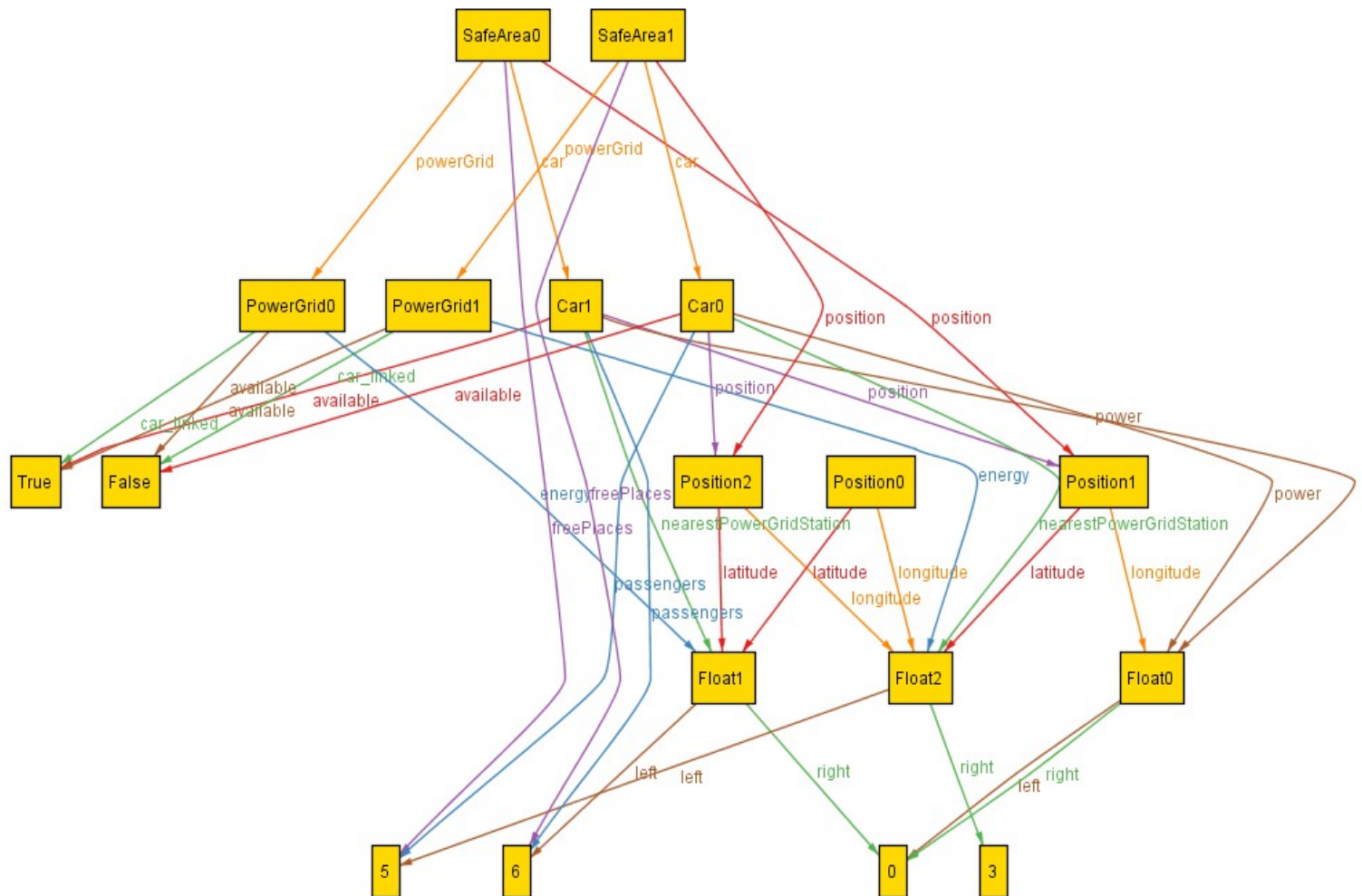
- #1: **Instance found.** showRentingandBooking is consistent.
- #2: **Instance found.** showCarAndAreas is consistent.
- #3: **Instance found.** showUserAndAssistant is consistent.
- #4: **Instance found.** show is consistent.
- #5: No counterexample found. DiscountPassengers may be valid.
- #6: No counterexample found. DiscountPowerGrid may be valid.
- #7: No counterexample found. DiscountPowerBattery may be valid.
- #8: No counterexample found. RaiseOnPrice may be valid.

## 6.3 Alloy graphics

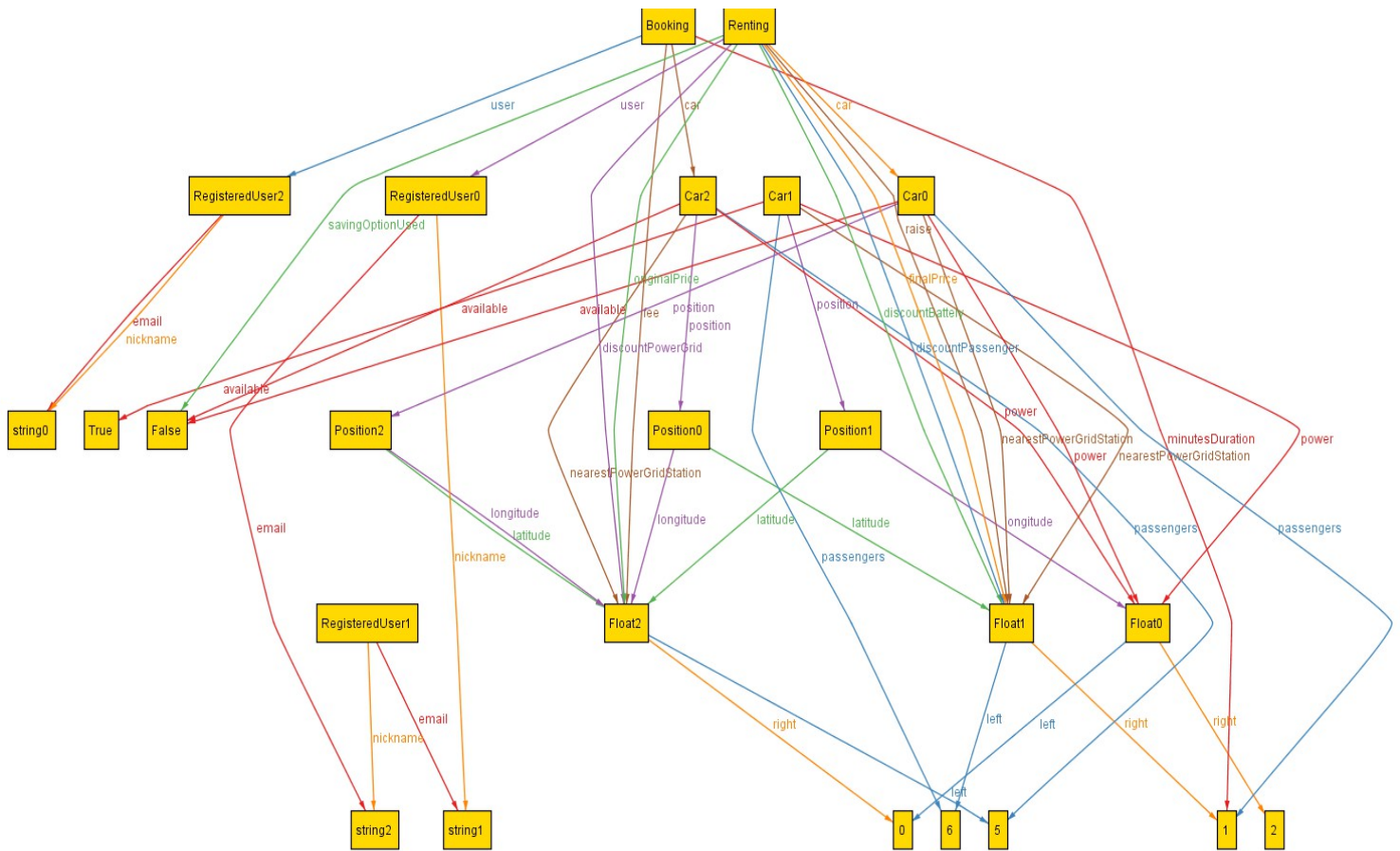
*Show Assistance Service with Use*



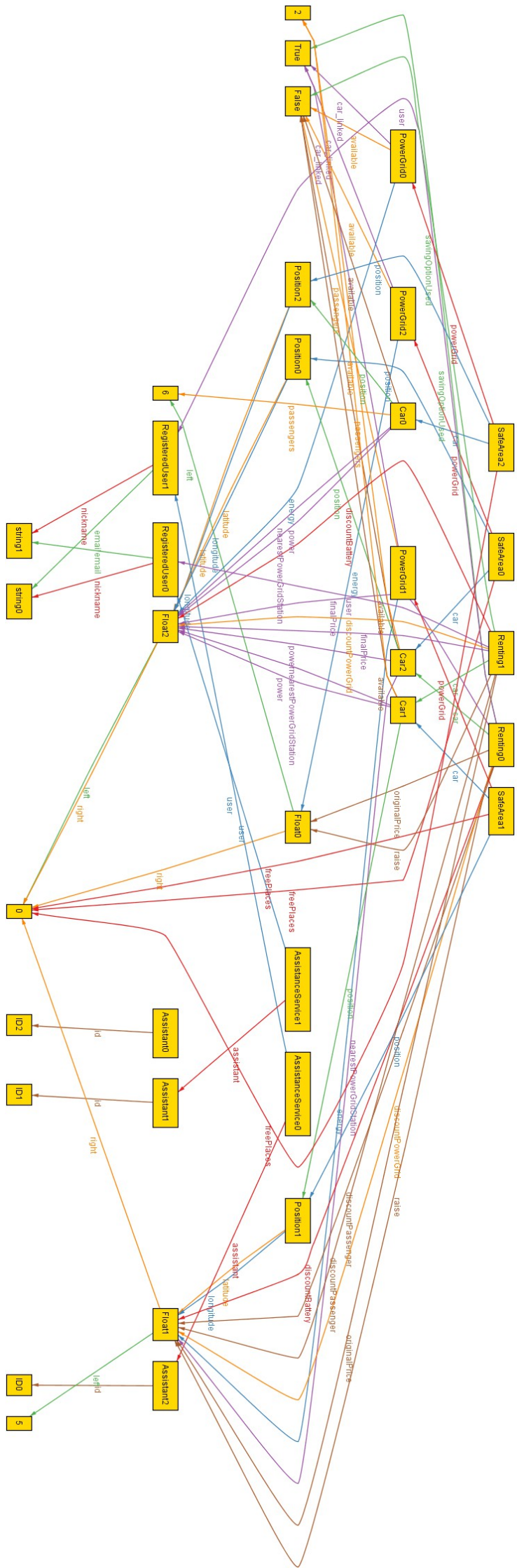
Show cars and safe areas



## Show renting and booking









## **7 Other Information**

### **7.1 Software, Tool and Documentation**

Below the list of the main programs we used and the papers we consulted for the realization of this document:

- Visio Standard 2016: for diagrams, such as use cases and sequence diagrams.
- MyBalsamiq: to draw the part related to the mockups.
- Notepad++: to write the alloy code in a clear way.
- Alloy4.2: for the implementation of Alloy code in order to verify that data were consistent.
- Paint: to model some pictures.
- Old assigned projects: as a reference for the structure of the document we consulted RASD Meteocal samples.
- Lectures slides.
- IEE Standard on requirement engineering.

### **7.2 Hours of work and effort**

- Angelo Falci: 17 hours; domain proprieties, glossary, assumptions, actor identifying, requirements, functional requirements, use cases, class diagram, alloy.
- Valentina Lanzuise: 17 hours; description of the given problem, goals, glossary, non-functional requirements, sequence diagrams, other information.
- Simone Lazzaretti: 20 hours; proposed system, identifying stakeholder, scenarios identifying, use cases, state chart, alloy.