POLITECNICO
MILANO 1863

PowerEnjoy

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

Version 1.0

Software Engineering 2 (A.A. 2016/2017)
- Simone Boglio (mat. 772263)
- Lorenzo Croce (mat. 807833)

Table of Contents

1. Introduction

1.1 Purpose
This document describes the RASD (Requirements Analysis and Specification Document) of the PowerEnjoy system. In the next pages you can find the description of the whole system; its components, the structure of the service, the interfaces with the old systems, requirement (functional and non) and the constraints.

This document provides also some scenarios that represent some real situation and how these situations are managed.

This document is useful for developers, managers, tester and you. It is also helpful to have a general view of the entire project.

1.2 Scope
PowerEnjoy is a digital management system for a car-sharing service thought to allow clients to search, reserve (and drive) a car using only a web service. In this way clients have user-friendly access to the car-sharing world in the city of Milan.

Moreover PowerEnjoy provides only electric cars and this is a good initiative on the way to a "green city".

As mentioned above, users can search and reserve a car using any devices with an internet access, like a pc or a smartphone. They can search a car using their current position (GPS localization) or submitting a specific address, the system will propose a list of the nearest cars and the clients will only have to choose which one they want to reserve. Customers who reserve cars have one hour's time to reach and begin to drive it, otherwise the car will be set as available and a fee is charged on the user.

To access the service, clients must be registered in the system.

When a client stops using the car, he has to drive it to one of the specific areas and it will be set as available.

The locking and unlocking of the car is managed by the system, users haven't got any material key.

Other functionalities are explained in the following part of the document.

1.3 Identifying stakeholders
The main stakeholders for this project are the actual car-sharing company who want to increase the number of clients and the users of the service.

1.4 Identifying actors
The actors of the system are only the clients.

After registration, clients can search and reserve a car (and then drive it) using a mobile application.

1.5 Definitions, acronyms, abbreviations

1.5.1   Definitions
- Client: people who want to access the service.
- User (or Customer): people who use the service.
- Passengers: the user that reserved a car can bring with themselves more travellers.
- Safe area: it's a specific area where cars are parked and can be parked by the user of the service.
- Special area: it's like a safe area but in this place there are power grid stations to recharge cars.
- Power grid station: it looks like a column where a car can be connected so that the car battery is recharged.
- Ride: term used to indicate the actual use of a car by the user. The use starts when the car engine is power on and ends when the engine is power off and the car is located in a safe area.
- Current position: position identified by GPS coordinates (of the car user).
- Available: it's a status of the car. It means that the car can be chosen by someone for a ride (using the web or app). If someone reserves an available car, this car status is changed to "reserved".
- Reserved: it's a status of the car. It means that the car cannot be chosen by anyone because a user is still using it. When a reserved car is parked in a safe area and its engine is power off, the status changes to "Available".

1.5.2   Acronyms
- RASD: Requirements Analysis and Specification Document
- API: Application Programming Interface.
- UI: User Interface.
- GPS: Global Position System.
- DBMS: Database Management System.
- RDBMS: Relational DBMS.
- DB: Database.
- HTTP: Hypertext Transfer Protocol.
- HTTPS: HTTP over SSL/HTTP Secure.
- OS: Operating System.
- JVM: Java Virtual Machine.

1.5.3 Abbreviations
- [Gn]: n(th) goal.
- [Rn]: n(th) functional requirement.
- [Dn]: n(th) domain assumption.

1.6 Goals
- [G1] The client shall be able to access the service through web service.
- [G2] The client shall be able to sign in and log in to the service.
- [G3] The user shall be able to manage his profile.
- [G4] The user shall be able to search cars in a specific zone.
- [G5] The user shall be able to reserve a car from a list up to one hour in advance.
- [G6] The user shall be able to pick up and drive the reserved car.
- [G7] The user shall be able to know where the safe areas for parking the car are.
- [G8] The user shall be able to know the current charges during the ride.

- [G9] The user shall be able to end the ride when he leaves the car.
- [G10] The system must incentivize the virtuous behaviour of users.
- [G11] The system has to offer public APIs to enable the possibility to develop additional services on top of the basic ones.

1.7 Pre-existent company situation

Until now the electric car company has a system where the clients have to call a call centre and communicate their position, the operator searches for the nearest available car and proposes it to the client, if the client accepts the proposed car the operator reserves it for him using the internal information system.

Cars are located in specific parking areas owned by the company. In each area there are some electric power stations for charging cars and a small office for an operator.

The operator manages all the cars in his station.

When the client reaches the indicated station, the operator checks the reservation and verifies the client's identity, after this he gives the key of the car to him.

When the client doesn't need the car any more, he has to give it back in one of the specific parking areas and pay the ride to the operator.

There are also some operators entrusted to move the cars from one parking area to another when these are full or empty.

1.8 Reference Documents

This document refers to the project rules of the Software Engineering 2 project and to the RASD assignment.

This document follows the IEEE Standard 830-1998 for the format of Software Requirements specifications.

1.9 Overview
- 1: Introduction, it gives a brief description of the purpose, functionalities and goals of the application.
- 2: Overall Description, focuses more in-depth on features of the software, constraints and assumptions.
- 3: Specific Requirements, lists of requirements, typical scenarios and use cases, both with UML diagrams to provide an easy understanding at the several functionalities of the software.

2 Overall Description

2.1 Product perspective

2.1.1 User Interfaces
The clients can access the service in two ways: web pages or mobile application. It is necessary to provide a common and uniform look and feel among the different hardware interfaces.
All the interfaces shall be intuitive and user-friendly. They should not require the reading of long document or special skills or knowledge to be able to use the application.

2.1.2 Hardware Interfaces
The main hardware interface of the system consists in the access to the system of the car, in particular we need to check some transductors and interact with some actuators like:
Transductor that checks the status of the car engine.
Transductor for every car seat that checks if a seated person is there.
Transductor that checks the status of the car doors (locked/unlocked).
Actuator that enables the opening and closing of the car doors.
GPS to know in every moment the position of the car.
We need also to access the GPS position data in the user's mobile application, the device that is able to respect these requirements is a smartphone.

2.1.3 Software interfaces
The mobile application must support Android and iOS (the most used mobile OS), for other devices (deprecated utilization) a web browser is enough.
The back-end store its data in a RDBMS and can run on every platform that supports JVM.
The back-end must offer programmatic interfaces (APIs) for user interfaces and external modules, like:
- Cars search
- Car reservation
- Online Payments
- Web interface

2.1.4 Communication interfaces
The Communication between users and system uses best known protocols. They are TCP, HTTP and HTTPS.
Those protocols must be supported by the devices used.

2.2 Product function

The system allows the user to search, reserve and use a car. The system must also promote the good behaviour of the user.
This is a list of what the users of the service can do.

Client:
- Create an account

User:
- Login.
- Edit profile data.
- Delete account.
- Search cars in a definite zone from his position or specific address.
- Reserve a car from a list.
- Ask the system to unlock his reserved car when he is nearby.
- See safe areas where they can leave the car.
- Pay for the ride when he stops using the car.

2.3 User characteristics

We give for granted that users of the power-enjoy service have access to the Internet.
Passengers have to use the browser application or the mobile app. They move alone or with other people (passengers).

2.4 Actual information system

Presently an information system where the company store all data already exists, in particular we are interested in the database where information about cars, payments and reservations are stored.
This database is used by:
- The administrator of the system that manages all the data about cars used by the company (like adding, removing new cars in the system).
- The call-centre operators that search and reserve a car for the client.
- The company that is responsible of car maintenance.
- The operators responsible of moving cars in case of car position imbalance in the city.
- The parking office operators that insert data about car reservation, client information and payments.

The cars used by the system are also provided with a GPS to check the current car position. It is used in some exceptional cases (such as someone who tries to steal a car) by an external company to trace cars following an order of the system administrator.

2.5 Constraints

2.5.1 Regulatory policies & safety and security

All cars are provided with all the necessary documents to circulate in the city: insurance, possession tax, revision and mechanic coupon.
It's the user's responsibility to act according to traffic law and for their own safety (in addition to the safety of others).

The system must guarantee user's privacy both over profile data and rides. The system must grant the user the possibility to delete their profile.

The mobile application requires only basic permission.

2.5.2 Hardware limitations

Every car must be provided with a GPS module to locate its position and an information system to communicate with the central system.

Every car must be provided with a control unit where all the actuators and transductors are connected.

Every car must be provided with engine status sensor, doors state sensor, presence sensor for each seat.

To access the service users must use a device that can access the internet and that is equipped with a GPS module (the latter is optional).

2.5.3 Interfaces with other applications

PowerEnjoy requires to access the Internet and the Google Maps APIs to provide map visualizations and map-related services.

2.5.4 Interfaces with actual system

PowerEnjoy requires to interface with the database of the present system where all the data of the service are registered and to keep all the relevant data updated since this system is used actually by different groups of people.

2.5.5 Reliability of the services

The system must have a minimum availability of 99%.

2.5.6 Parallelism

The system will be able to support operations from different users at the same time without conflict.

2.6 Assumption and dependencies

- The user has a valid driving licence.
- The user has enough money at the end of the ride to pay for the bill.
- The user can pay the fee in case the car isn't picked up within one hour since the reservation.
- If a user switches off the engine and leaves the car in a safe area, , he must start a new operation if he wants to use the car again.
- If a user switches off the engine and leaves the car in a no safe area the system continues to charge the user.
- The user drives the car only in the city.
- The user will drive the car back into a safe area (they don't steal it).
- GPS position is always correct.
- If a car is set as "reserved" the system doesn't propose it when a user is searching for a car.
- The GPS installed on the car is always on.
- Each user can reserve only one car at a time.
- Set of safe areas is pre-defined by the management system.
- Safe areas are equivalently distributed in the city.
- Only the user who reserved the car can drive it.
- Every available car has enough charge to end a ride.

- Special areas are at the same time safe areas.
- Available cars are always in a safe area.
- All cars are of the same model and have the same number of seats (1 driver, 4 passengers).
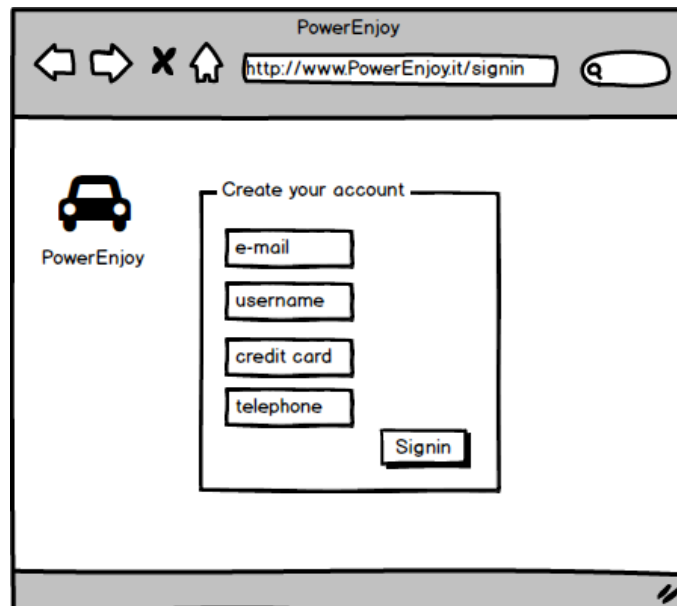- Car maintenance is assigned to an external company.

3   Specific Requirements
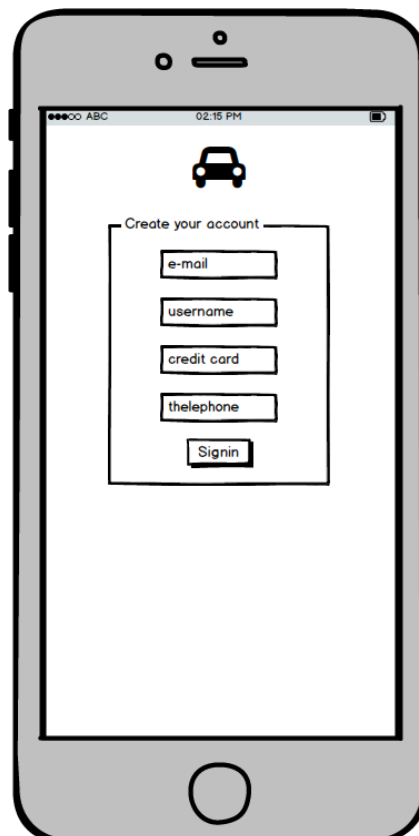
3.1 External Interface Requirements

3.1.1   User Interfaces

3.1.1.1 Sign in (Web and App)
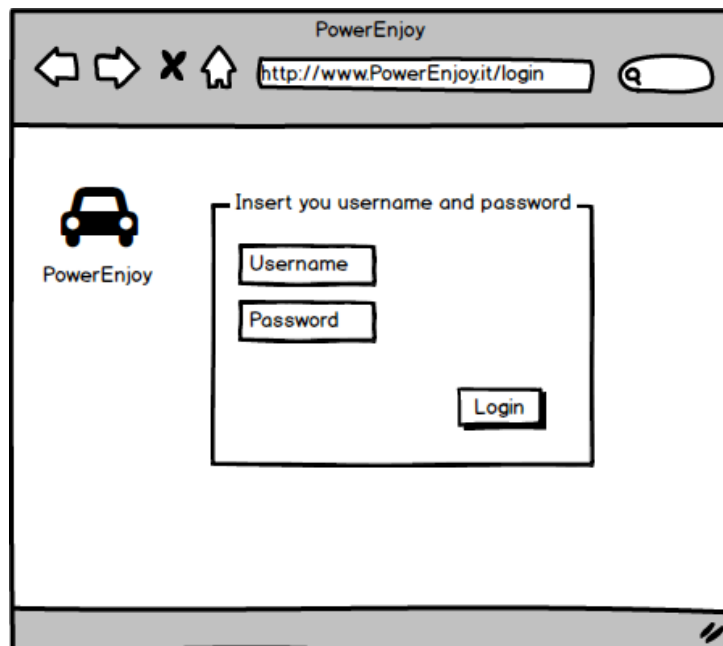This page represents the sign in form. The client must provide his personal information.

3.1.1.2 Login (Web and App)

This page represents the login form. The user must provide his credentials to access the service.

3.1.1.3 Research (Web and App)

This page represents the research form. The user can choose whether to search for the car from his current position or submitting a valid address.
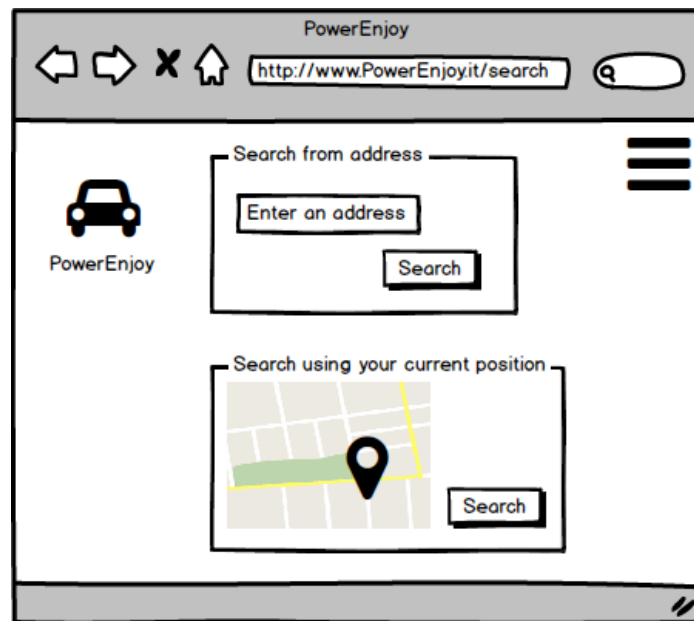
3.1.1.4 Reservation (Web and App)

This page represents the reservation form. The user can choose one car from the list and reserve it.
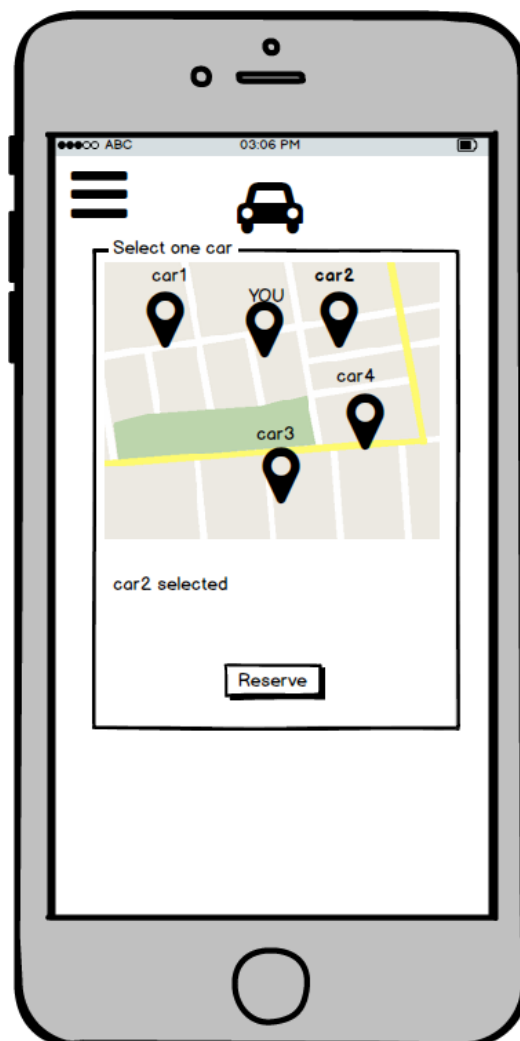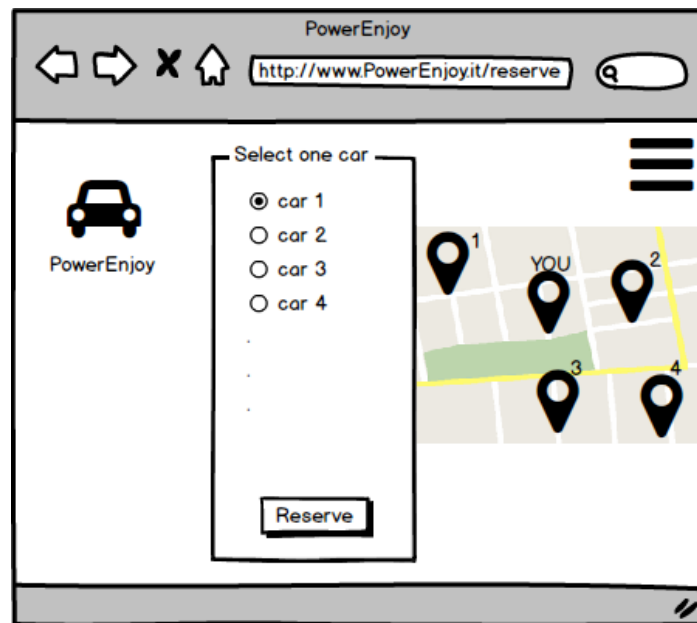
3.1.1.5 Car screen

In each car there is a screen where some information is indicated. These information is referred to the battery level, the current charge for the ride and the positions of safe areas (including special areas).
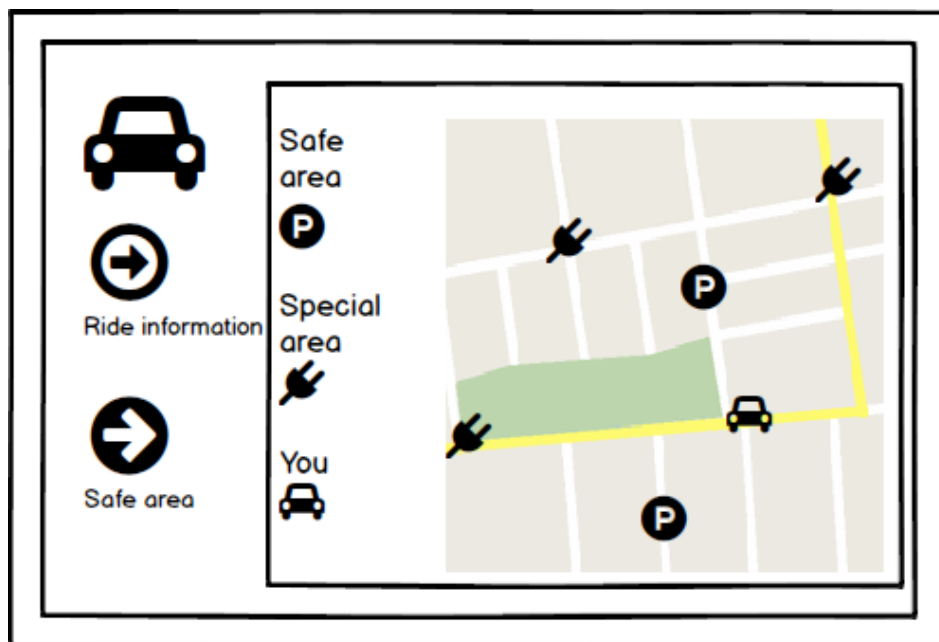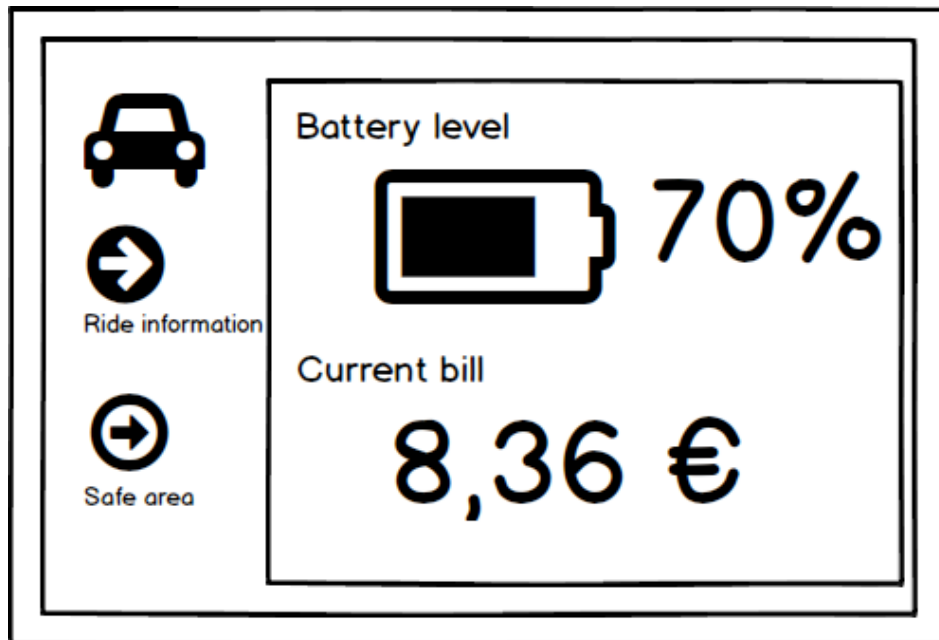
3.1.2 Hardware Interfaces

Every car needs a screen where the user can see the current charge for the ride.
Nothing else is required, to access the service the client doesn't need special hardware components, a compatible smartphone (IOS and Android) is enough or any supported browser on a computer.

3.1.3 Communication Interfaces

PowerEnjoy uses the TCP transport protocol and HTTP/HTTP over SSL application layer protocol to guarantee the top of the line security on the matter of transmission of private data.
PowerEnjoy also needs to be able to access the control car unit through appropriate protocols to check state values and control the lock system of the car.

3.2 Functional Requirements

3.2.1 [G1] The client shall be able to access the service through web service.
- [R1] The system must provide two different ways to access to the service with same functionalities: web page and mobile app.

3.2.2 [G2] The client shall be able to sign in and log in to the service.
- [R1] The client must be able to submit a valid email address, phone number and choose an available username, this fields are mandatory for successfully complete the registration.
- [R2] The system must be able to send back to the new user his password.
- [R3] The user can be able to log in into the service submitting his username and his password.
- [R4] The user must be able to submitted his credit card number in according with the privacy and security policies.
- [R5] The unregistered user can be see only sign in and log in pages.
- [R6] The system must implement a password retrieval mechanism.
- [D1] Phone number, email address and credit card data used for the registration must be valid.

3.2.3 [G3] The user shall be able to manage his profile.
- [R1] The logged user must be able to modify and delete his profile.
- [R2] The logged user must be able to edit his credit card number.

3.2.4 [G4] The user shall be able to search cars in a specific zone.
- [R1] The user must be able to search cars near his current position (using GPS coordinate).
- [R2] The user must be able to search cars near a specific address.
- [R3] The system must be able to verify that the submitted position is in the city.
- [R4] The system must be able to refuse request of searching car in position out of the city.
- [R5] The system must be able to provide a list of available cars in according with the position indicated by the user.
- [D1] There is always at least one available car thanks to the operators that manage the distribution of the cars in the city.
- [D2] The GPS coordinates of the car are always available and correct.
- [D3] All the cars have the same number of seats

3.2.5 [G5] The user shall be able to reserve a car from a list up to one hour.
- [R1] The user must be able to see the list of available car received as response to his request.
- [R2] The user must be able to select and reserve a car from the list for up to one hour.

- [R3] The user can reserve only one car by one.
- [R4] If after one hour of car reservation the user don't pick up the car, reservation expired and the user pays a fee of 1 EUR.
- [D1] The user knows that the allowance of the car is 4 passengers.
- [D2] The user can't delete a reservation.

3.2.6 [G6] The user shall be able to picks up and drives the reserved car.
- [R1] The user who reserved a car must be able to tell the system he is near the car.
- [R2] The system must be able to calculate the distance from the user and the car.
- [R3] If the distance between user and car is short, the system must be able to unlock the car.
- [R4] The system must be able to know if car's engine is power on.
- [R5] The system must be able to notifies user of the current charges.
- [D1] The user doesn't drive out of Milan more than 10km.
- [D2] The car doesn't run out of battery during the ride.

3.2.7 [G7] The user shall be able to know where are the safe area for parking the car.
- [R1] The system allows to user to see the position of safe areas.
- [R2] The system allows to user to see the position of special areas.

3.2.8 [G8] The user shall be able to know the current charges during the ride.
- [R1] The system allows to user to see the current charges by a screen installed on the car.
- [R2] The system must be able to send to the car the current charges.

3.2.9 [G9] The user shall be able to end the ride and pay it when he leaves the car in safe area.
- [R1] The system must be able to know if the car's engine is power of and the car is parked in a safe area.
- [R2] The system must be able to stop charging user if the car is parked in a safe area.
- [R3] The system must be able to notify the user the account of the ride.
- [R4] The user must be able to see the amount of the ride.
- [R5] The system must charge the total amount of the ride on the credit card user.
- [D1] The set of safe areas is pre-defined by the system.
- [D2] The user has enough money on his credit card.

3.2.10 [G10] The system must incentivize the virtuous behaviours of the users.
- [R1] The system must be able to know the number of passenger in the car.
- [R2] The system must be able to know the battery level of the car.
- [R3] The system must be able to know if the car is in a special area.
- [R4] The system must be able to detect the car's position.
- [R5] The system must be able to apply a discount of 10% on the last ride if it detects that there are at least two other passengers onto the car.
- [R6] The system must be able to apply a discount of 20% on the last ride if it detects that the car is left with no more than 50% of the battery empty.
- [R7] The system must be able to apply a discount of 30% on the last ride if it detects that the car is parked in a special area and it is plugged into a grid power.
- [R8] The system must be able to apply a charges of 30% more on the last ride if it detects that the car is left at more than 3Km from the nearest power grid station or with no more than 80% of the battery empty.
- [R9] Bonus for one ride are cumulative.

-   [R10] The discount percentage can't exceed 100% of the total amount.

3.2.11 [G11] The system has to offer public APIs to enable the possibility to develop additional services on top of the basic ones.
-   [R1] The system offers APIs to third party applications using web APIs as technology.
-   [R2] The system replies using current industry standards.
-   [D1] Access to APIs functionalities is provided only using the HTTPS protocol.

3.3 Scenarios

3.3.1   Scenario 1
Marco lives in Milan and he wants to go to shopping but he doesn't want to use public transports. Marco has already registered to PowerEnjoy service and he has the mobile app installed on his Android smartphone. He decides to use it. He starts the application, logs in to the system and starts searching for the nearest car using his GPS position. In few seconds a map with the list of available cars is shown on his smartphone and he only has to select and reserve one of them. When he reaches his reserved car he takes his phone and using the PowerEnjoy app asks to unlock the car. The car is immediately unlocked and he can drive it. When the engine is on the system starts charging and Marco can see current charges on the screen installed in the car.
When Marco arrives in the town centre, he searches for a safe-area, parks the car and the system charges Marco's credit card with the amount for the ride.

3.3.2   Scenario 2
Davide lives in Milan and today he has to go to the post office to send a letter, then he wants to go to his friend Luca. He wants to use public transport to go to the post office and a car to reach Luca's house since it isn't serviced by any public transport. So he decides to use PowerEnjoy service from the PC in his house to search for cars from the address of the nearest post office. Immediately a list of cars in the nearby submitted address appears on the pc screen and Davide can choose which car to reserve. In this way he has up to one hour to go to the post office and pick up the reserved car to reach his friend Luca.

3.3.3   Scenario 3
Andrea and his friends Lucia and Federico want to join a party tonight but they don't have a car and it's too late to take the underground, they are young and don't want to spend a lot of money to take a taxi, so they decide to use PowerEnjoy. After registration and reservation of the nearest car Federico and his friends pick up the car and since they want to save some money they decide to find the special area nearest to the party location, on the map on the screen of the car, to park and plug the car into the power grid station. In this way Federico gets two bonuses: one for bringing at least other 2 passengers and one for leaving the car in the special area plugged in the power station. Federico and his friends save some money and now they can happily join the party in few minutes.

3.3.4   Scenario 4
Luigi arrives this morning in Milan by train to join a conference of his company, he doesn't know which public transport he should take since the location is more than one hour far on foot, so he decides sadly to walk to the destination, after few metres he sees one of the cars of PowerEnjoy service and reads on one of its side "PowerEnjoy, green car-sharing for You, download our app and start to drive", intrigued by the message he searches for and downloads the app on his Android

smartphone, since he hasn't registered yet, he inserts his data (name, email, phone number, number of driving licence and credit card) and after few seconds he gets the password to join the system. Through his GPS position the system proposes him quickly the car in front of him, in one-click he reserves the car and the system confirms the reservation, at this point he clicks the button to signal he is near the car and the system unlocks the doors of the car and he finally can drive to the conference destination. Luigi is pleasantly surprised that it took him less than 5 minutes to do everything. After 15 minutes he arrives at his destination and thanks to the map on the screen he finds the nearest safe area where he leaves the car, he turns off the engine and gets out of the car and receives the bill automatically on his banking account.

Luigi easily arrives 40 minutes in advance and in this time he can relax and have a coffee, he will use PowerEnjoy service again, that's for sure.

3.4 UML models

3.4.1 Use Case
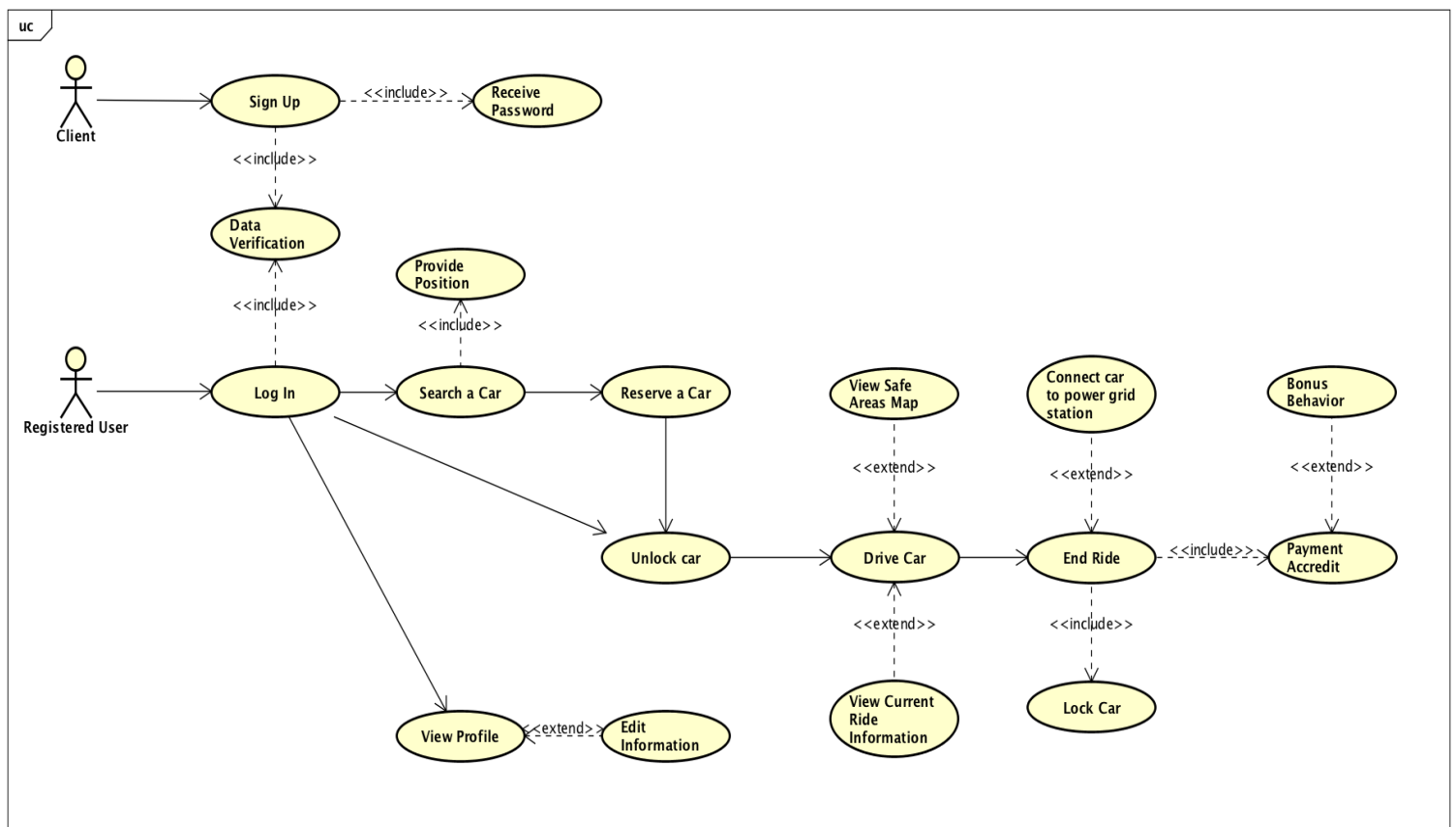
3.4.1.1 The client shall be able to sign in to the service

Actor	Client
Goal	[G2]
Entry conditions	There isn't any entry condition.
Flow of events	- The client goes to the home of the service (using the app or the website). - The client chooses "Sign in" option. - The client compile requested fields. - The client clicks on the button "Sign in". - The system saves information on a DB and send back to the client his password.
Exit condition	The client became a PowerEnjoy user and he can start to use the service.
Exceptions	- The submitted e-mail address is already registered on the service. - The submitted username is already used by someone. - Some fields are not compiled
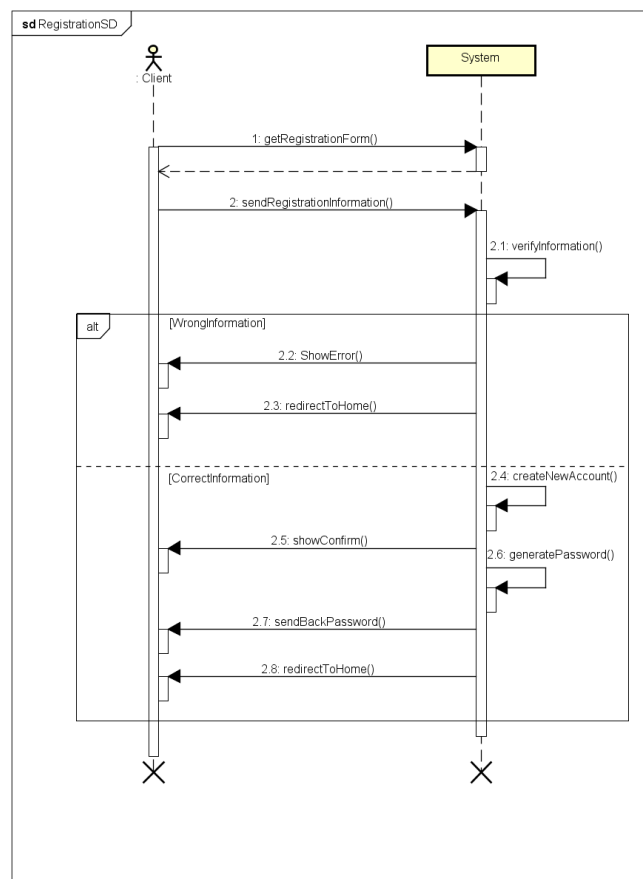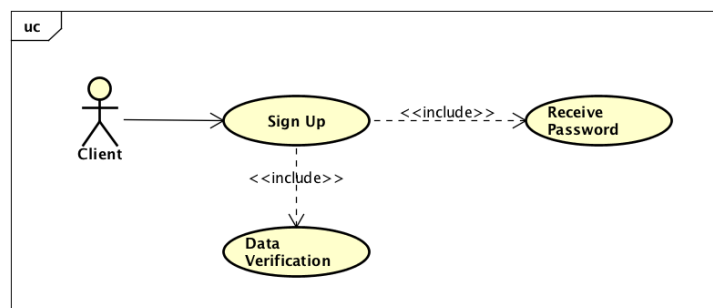
3.4.1.2 The client shall be able to log in to the service

Actor	Client
Goal	[G2]
Entry conditions	The client has already a valid account.
Flow of events	- The client goes to the home of the service (using the app or the website). - The client chooses "Log in" option. - The client inserts username and password. - The client clicks on the button "Login". - The system check if the credentials are correct.
Exit condition	The user is redirected to the search page.
Exceptions	- The credentials are incorrect.
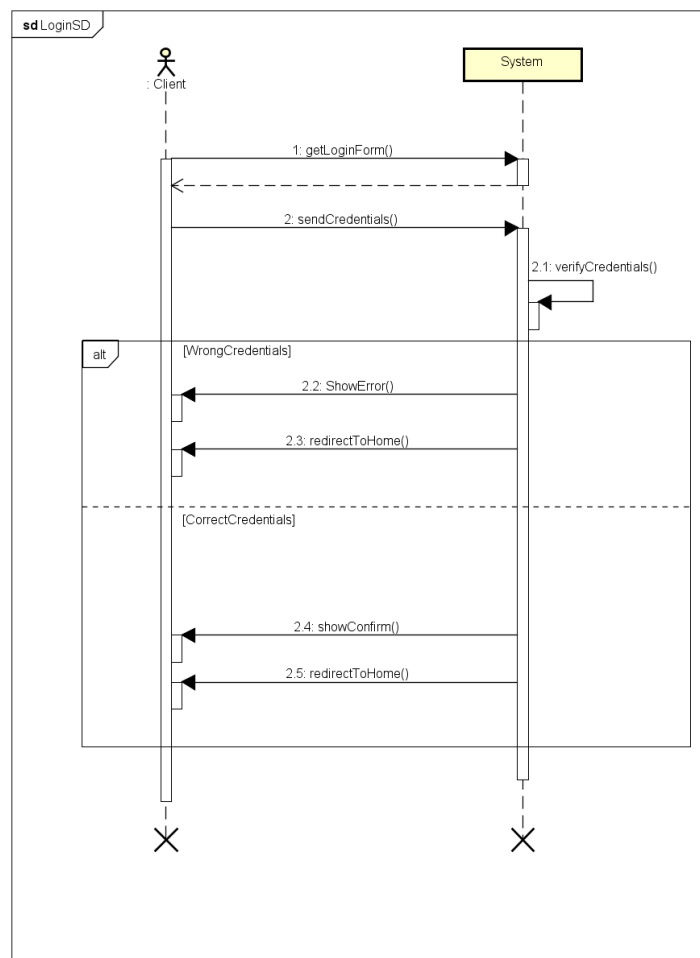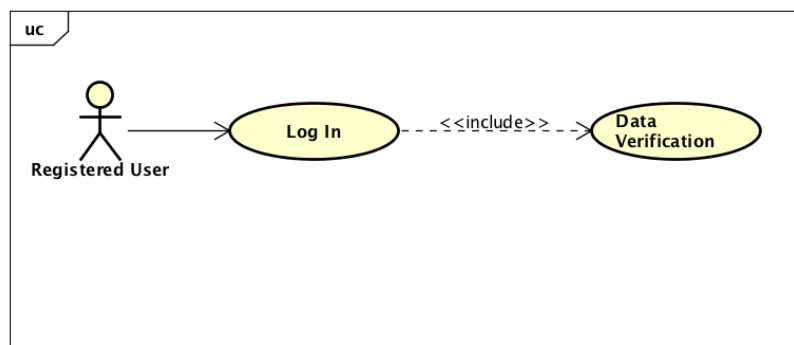
3.4.1.3 The user shall be able to manage his profile

Actor	User
Goal	[G3]
Entry condition	The user is already logged into the system.
Flow of events	- The user selects "Profile" option in the homepage. - The user modifies information (like credit card number or telephone number). - The user clicks "Save" button. - The system updates new data.
Exit condition	The system shows the update message and redirect user to the home.
Exception	- The user submits a blank field.

3.4.1.4 The user shall be able to search cars in a specific zone

Actor	User
Goal	[G4]
Entry condition	The user is already logged into the system.
Flow of events	- The user selects "search" option on the homepage. - The user chooses kind of research: by current position or by address. - If user chose to search by address he submits the address information. - The user clicks "Search" button.
Exit condition	The system shows on the user's smartphone a list of available cars.
Exception	- If user chose to search by address and he doesn't submit any address, system return an error.
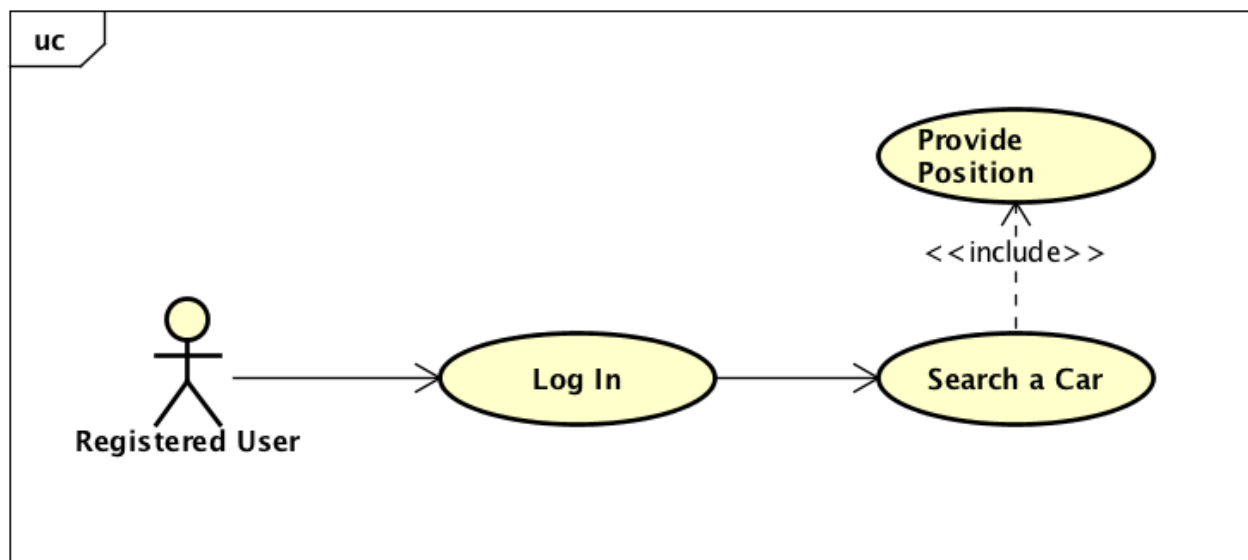
3.4.1.5 The user shall be able to reserve a car from a list up to one hour

Actor	User
Goal	[G5]
Entry condition	The user has already sent a research request.
Flow of events	- The user selects what car he prefers to reserve. - The user confirms his choice clicking "reserve" button.
Exit condition	The system shows the reservation message and redirect user to the home.
Exception	- If another user reserves the same car first that the user completes the operation, system shows a message and asks for a new selection.

3.4.1.6 The user shall be able to picks up and drives the reserved car

Actor	User
Goal	[G6]
Entry condition	The user is logged to the system and has already sent a reservation request.
Flow of events	- When the user is in the car's neighbourhood, it appears on his smartphone a notification and the he clicks on "Unlock" button. - The system unlocks the user's reserved car. - The user opens the car and drive it.
Exit condition	The user can drive his reserved car.
Exception	There isn't any exception.
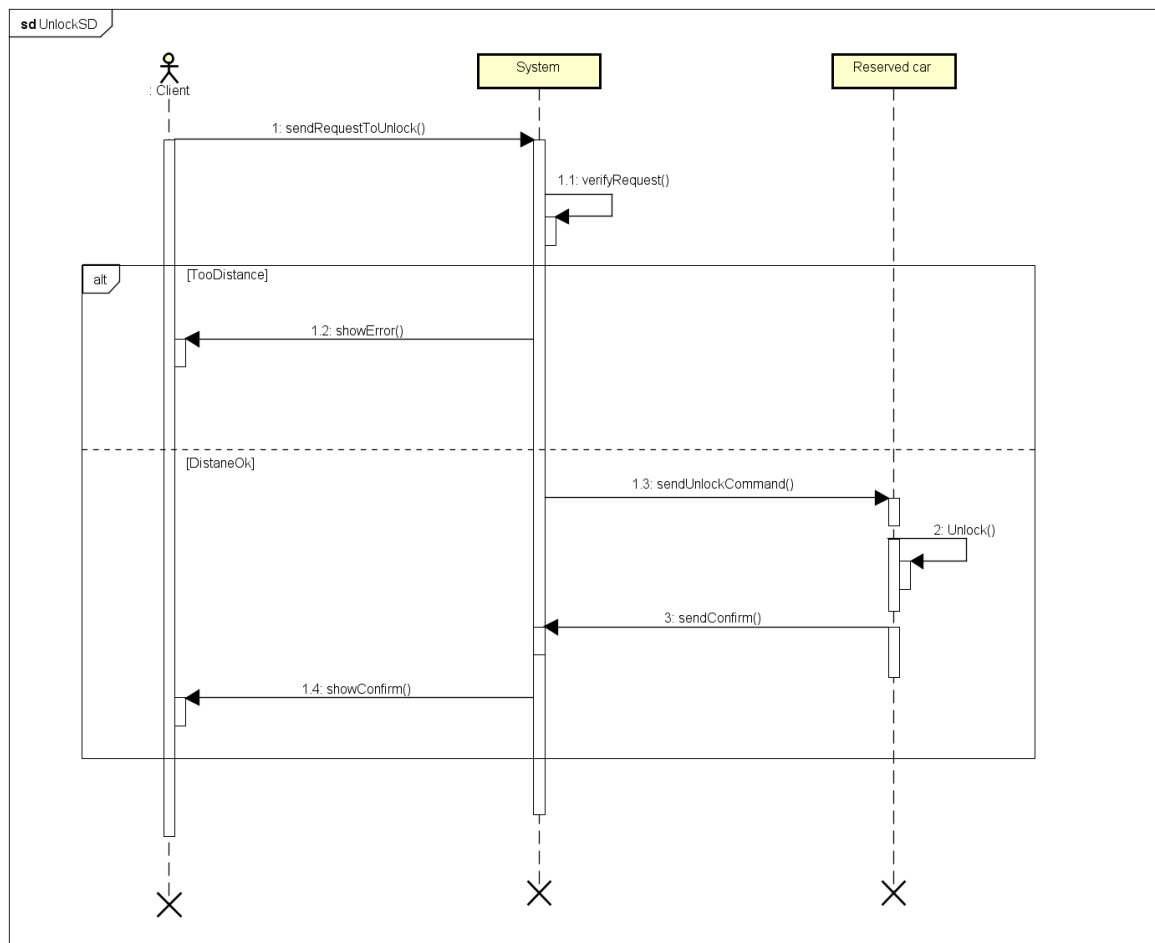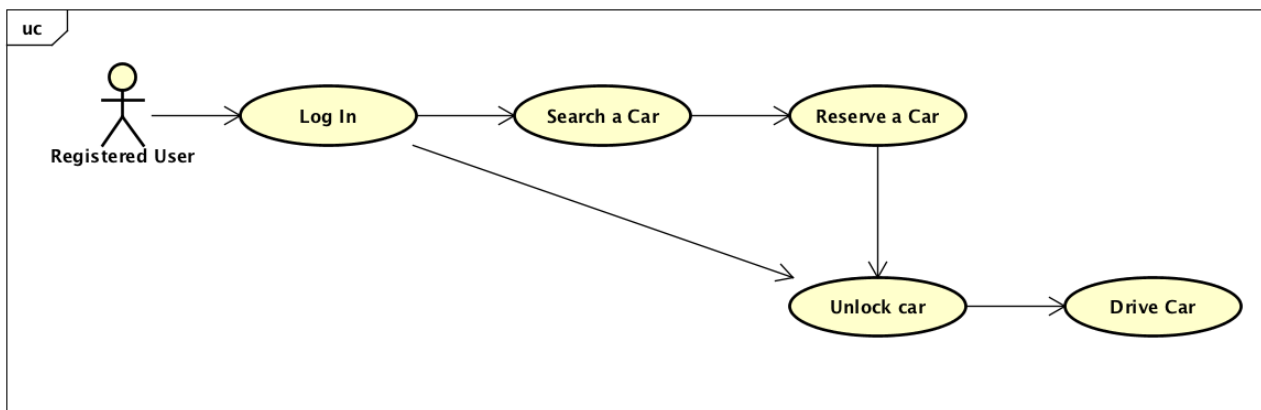
3.4.1.7 The user shall be able to know where are the safe area for parking the car

Actor	User
Goal	[G7]
Entry condition	The user is already driving the car.
Flow of events	- The user can see where the safe (and special) areas are clicking on the "Safe area" option on the screen's car.
Exit condition	A map of the city appears on the screen and in it are represented the safe and special areas.
Exception	There isn't any exception.
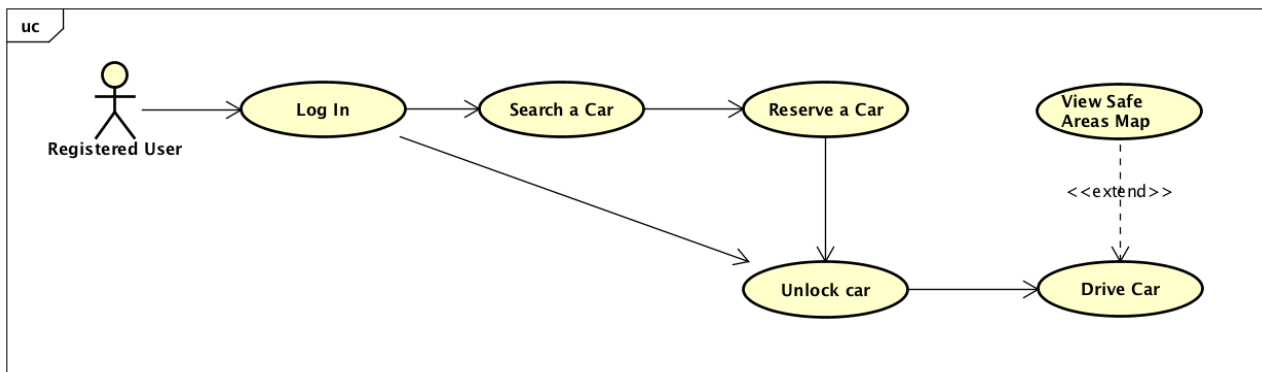
3.4.1.8 The user shall be able to know the current charges during the ride

Actor	User
Goal	[G8]
Entry condition	The user is already driving the car.
Flow of events	- The user can see the current charges clicking on "Ride information" option on the screen's car.
Exit condition	On the car's screen appears information about charges and battery status.
Exception	There isn't any exception.

3.4.1.9 The user shall be able to end the ride and pay it when he leaves the car in safe area

Actor	User
Goal	[G9]
Entry condition	The user is already driving the car.
Flow of events	- The user drives the car until a safe or special area. - The user, if he is in a special area can plug the car into the power grid for take a discount. - The user power off the engine. - The user left the car. - The system checks that the car is parked in a safe area, engine is power off and there isn't anyone on the car and then locks it. - The system accredits the ride (applying discount if it is required) and notifies the user.
Exit condition	The car is locked and its status is changed in available.
Exception	There isn't exception.
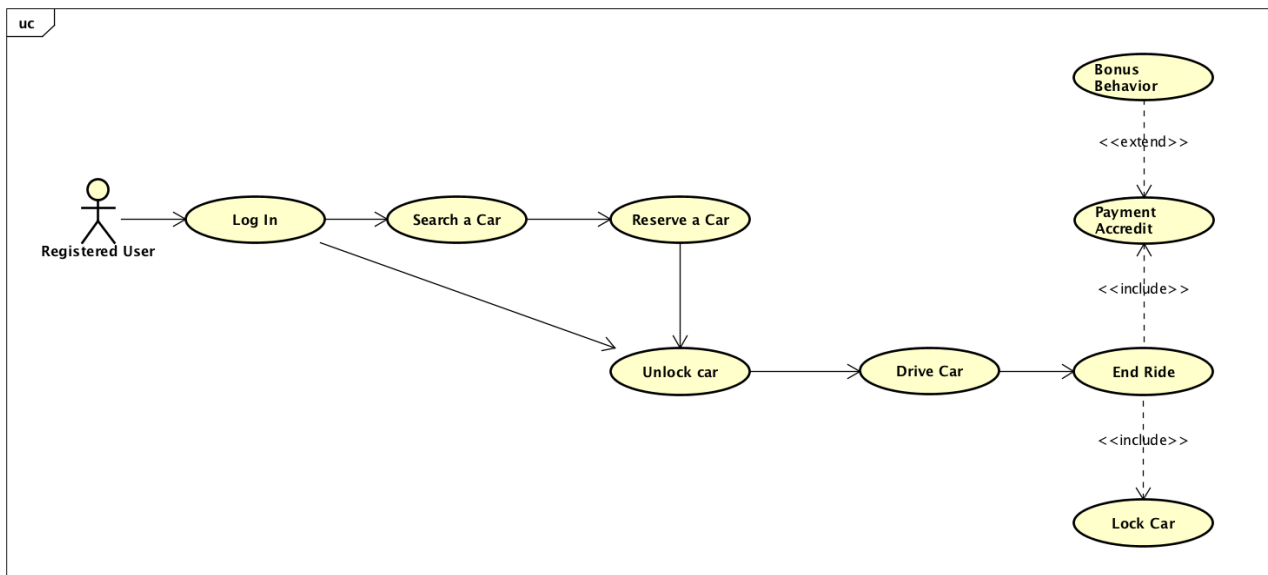
3.4.2 Class Diagram
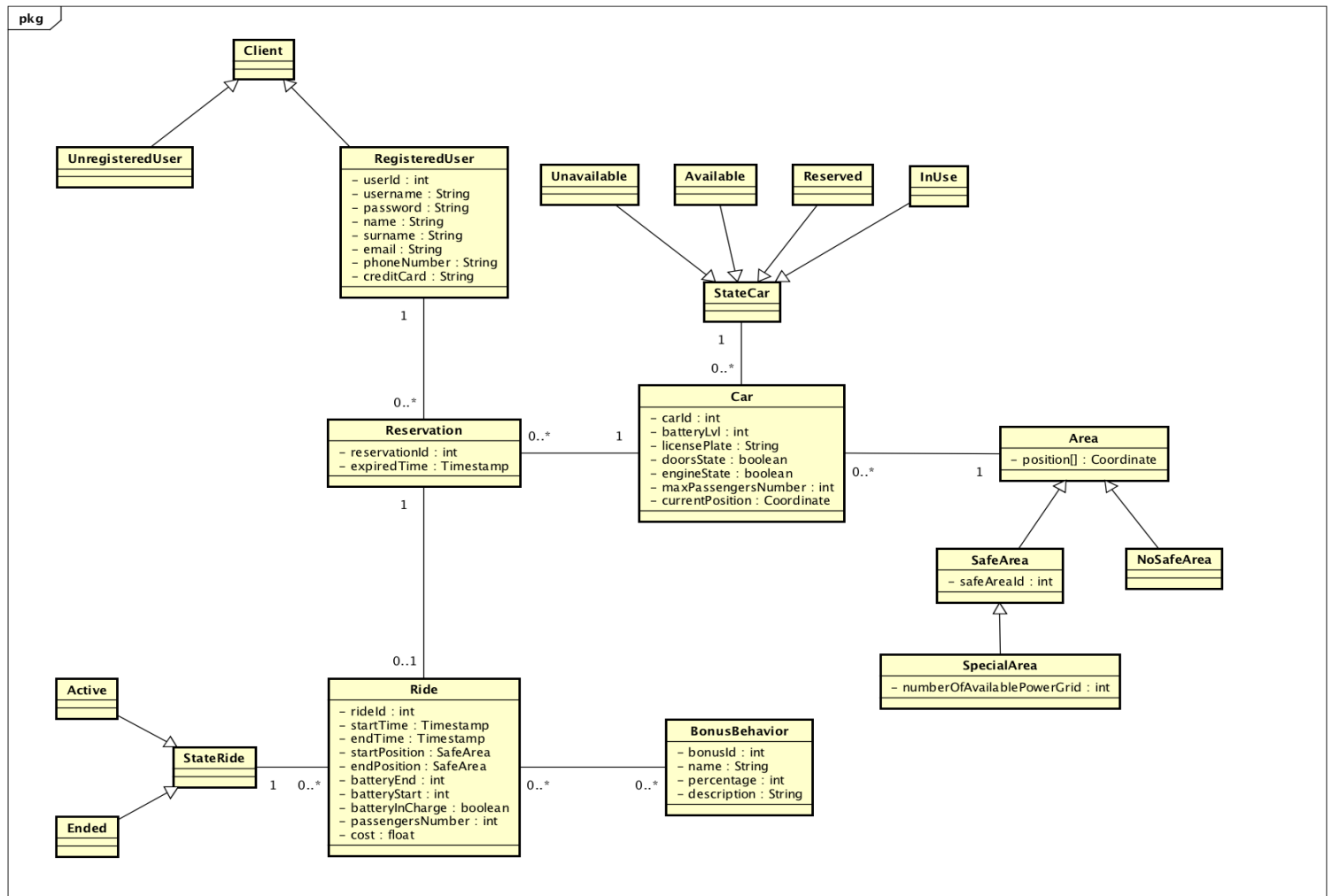
3.4.3   State machine Diagrams

3.4.3.1 Car availability
The following finite state machine shows the possible states of Availability of a car and the triggers that enable the change of state.
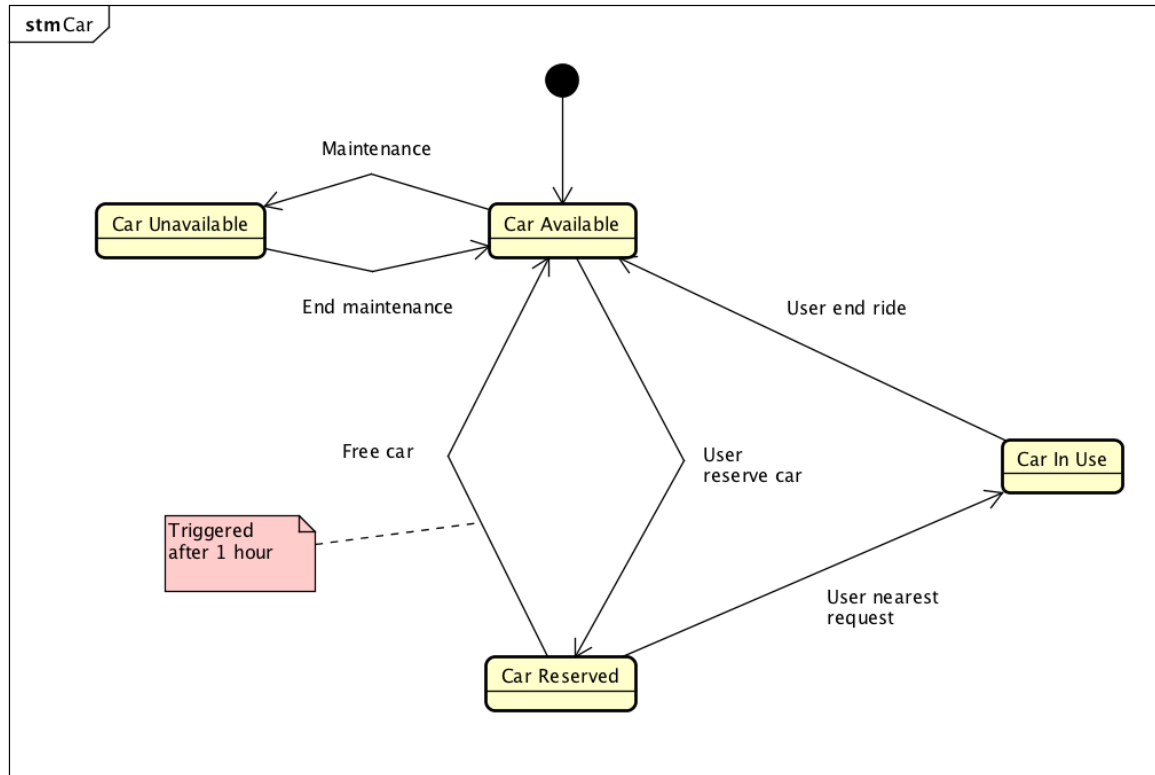
3.4.3.2 Ride state
The following finite state machine shows the possible states of a ride and the triggers that enable the change of the state.
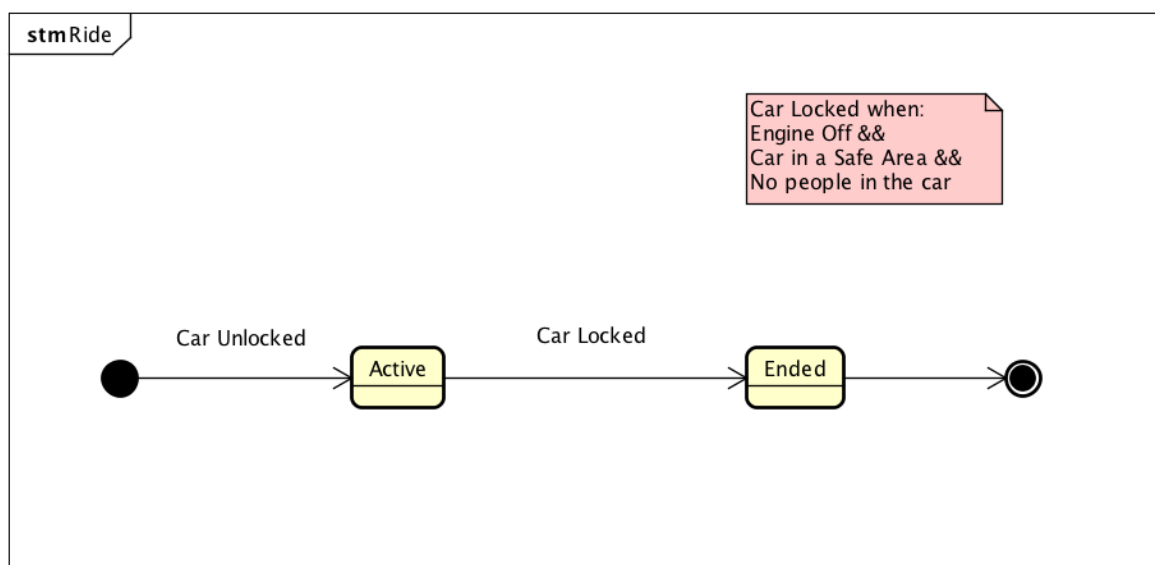
3.5 Non-functional Requirements

3.5.1   Performance Requirements
PowerEnjoy is a system that wants to help the largest number of people to move into Milan.
For this aim the system must be able to support a large number of requests (many of these simultaneous) and it must be able to respond to any request in few seconds. So the server must be correctly chosen to guarantee the best possible efficiency. It must be scalable.

3.5.2   Software System Attributes

3.5.2.1 Reliability
The system must guarantee that if the user makes a reservation, his reservation is available within the deadline.

3.5.2.2 Availability
The system must guarantee that the service is available 24/7.

3.5.2.3 Usability
The user interface must be user-friendly to guarantee a very quick access to the service in both cases; access through app and access through web page.

3.5.2.4 Security
Since the user must insert his credit card number at the registration moment, the system must guarantee that this information is hidden from anyone and the only permitted use for it is to pay for the ride. The system must guarantee the privacy of all registered users, too.

3.5.2.5 Maintainability
The software itself does not require any particular periodic update, unless the PowerEnjoy company wants to add some functionalities. The only feature that could require some maintenance are the APIs offered by the system.

4 Alloy

4.1 Code

//Alloy model for PowerEnjoy system

//Defines Bool -> True, False
open util/boolean

sig Stringa {}

sig Date{
        time: one Int
}
{
        time >0
}

sig Position {
        positionId: one Int
}
{
        positionId > 0
}

abstract sig User {}

sig UnregisteredUser extends User {}

sig RegisteredUser extends User {
        userId: one Int,
        username: one Stringa,
        email: one Stringa,
        password: one Stringa,
        name: one Stringa,
        surname: one Stringa,
        phoneNumber: lone Stringa,
        creditcardNumber: one Stringa
}
{
        userId>0
}

abstract sig StateCar {}
one sig AvailableCar extends StateCar{}
one sig UnavailableCar extends StateCar{}
one sig InUseCar extends StateCar{}
one sig ReservedCar extends StateCar{}

abstract sig StateBattery{}
one sig LowBattery extends StateBattery{}
one sig MediumBattery extends StateBattery{}
one sig HighBattery extends StateBattery{}

sig Car {
        carId: one Int,

```
        batteryLvl: one StateBattery,
        stateAvailability: one StateCar,
        licensePlate: one Stringa,
        doorState: one Bool, // False = locked door, True = unlocked door
        engineState: one Bool, // False = engine off, True = engingeOn
        numberOfSeats: one Int,
        position: one Area,
        batteryInCharge: one Bool,
        displayInformation: one Bool //current charge, map, battery level
}
{
        (batteryInCharge = True) => (engineState = False)
        (displayInformation = True) => (stateAvailability = InUseCar)
        (stateAvailability = UnavailableCar) => (engineState = False)
        (stateAvailability = AvailableCar) => (engineState = False and position = SafeArea)
        (stateAvailability = ReservedCar) => (engineState = False and position = SafeArea)
        (stateAvailability = InUseCar) <=> (doorState = True)
        (engineState = True) => (stateAvailability = InUseCar)
        (doorState = True) => (stateAvailability = InUseCar)
        carId > 0
        numberOfSeats = 4
}

sig Reservation {
        reservationId: one Int,
        reservationTime: one Date,
        reservedCar: one Car,
        expired: one Bool,
        user: one RegisteredUser,
        fee: one Bool
}
{
        (fee = True) => (expired = True)
        reservationId >0
}

abstract sig Area {
        position: one Position
}
sig NoSafeArea extends Area{}
sig SafeArea extends Area {}
sig SpecialArea extends SafeArea{
        numberOfPowerGrid: Int,
        car: set Car,
}
{
car.position = SpecialArea
}

abstract sig StateRide {}
one sig ActiveRide extends StateRide{}
one sig EndedRide extends StateRide{}

sig Ride{
        rideId: one Int,
        reservation: one Reservation,
        state: one StateRide,
        startTime: one Date,
```

```
        endTime: lone Date,
        startPosition: one SafeArea,
        endPosition: lone SafeArea,
        batteryStart: one StateBattery,
        batteryEnd: lone StateBattery,
        passengersNumber: Int,
        cost: one Int,
        bonusSet: set Bonus,
        specialAreaTooFar: lone Bool,
        batteryInCharge: lone Bool
}
{
        rideId > 0
        cost >= 0
        passengersNumber >= 0
        passengersNumber < reservation.reservedCar.numberOfSeats //< because 1 is the user
        startTime.time >= reservation.reservationTime.time
        startTime.time < endTime.time
        specialAreaTooFar = True <=> (endPosition != SpecialArea and batteryInCharge = False)
        batteryInCharge = True => endPosition = SpecialArea
        state = ActiveRide <=> (no endPosition) and (no batteryInCharge) and (no specialAreaTooFar)
        state = EndedRide <=> (one endPosition) and (one batteryInCharge) and (one specialAreaTooFar)
        (BonusInCharge in bonusSet) <=> (BonusDistanceLowBattery not in bonusSet)
        (BonusHighBattery in bonusSet) <=> (BonusDistanceLowBattery not in bonusSet)
        state = ActiveRide => (no bonusSet)
}

abstract sig Bonus {}
one sig BonusPassenger extends Bonus{}
one sig BonusHighBattery extends Bonus{}
one sig BonusInCharge extends Bonus{}
one sig BonusDistanceLowBattery extends Bonus{}

//////////Facts

fact registeredUserAreUnique{
        all u1, u2: RegisteredUser | (u1 != u2) => (u1.userId != u2.userId)
        all u1, u2: RegisteredUser | (u1 != u2) => (u1.email != u2.email)
}

fact carAreUnique{
        all c1, c2: Car | (c1 != c2) => (c1.carId != c2.carId)
        all c1, c2: Car | (c1 != c2) => (c1.licensePlate != c2.licensePlate)
}

fact positionAreUnique{
        all a1, a2: Area | ((a1 != a2) <=> (a1.position.positionId != a2.position.positionId))
}

fact stateRide{
        all r1: Ride | (r1.state = ActiveRide) => (one r1.startTime and one r1.startPosition and one r1.reservation)
        all r1: Ride | (r1.state = EndedRide) => (one r1.startTime and one r1.startPosition and one r1.reservation and
one r1.endTime and one r1.endPosition)
}

fact noTwoConcurrentReservationSameUser{
        no r1, r2: Reservation | (r1 != r2 and r1.user = r2. user and (r1.expired = False and r2.expired = False))
}
```

```
fact noReservationIfUserIsDriving{
        no r1: Reservation, rd1: Ride | (r1 != rd1.reservation and r1.user = rd1.reservation.user and rd1.state =
ActiveRide)
}

fact noTwoConcurrentReservationSameCar{
        no r1, r2: Reservation | (r1 != r2 and r1.reservedCar = r2. reservedCar and (r1.expired = False and r2.expired =
False))
}

fact unavailableCarCannotBeReserved{
        no c1: Car, r1: Reservation | (r1.reservedCar = c1 and c1.stateAvailability = UnavailableCar)
}

fact reservedCard{
        all c1: Car, r1: Reservation | (r1.reservedCar = c1 and r1.expired = False) <=> (c1.stateAvailability =
ReservedCar)
}

fact expiredReservation{
        all r1: Reservation, rd1: Ride | (r1 = rd1.reservation) => (r1.expired = True)
        all r1: Reservation | (r1.fee = True) => (r1.expired = True and r1.reservedCar.stateAvailability = AvailableCar)
}

fact inUseCar{
        all rd1: Ride | (rd1.state = ActiveRide) <=> (rd1.reservation.reservedCar.stateAvailability = InUseCar)
        all c1: Car | (c1.position != SafeArea) => (c1.stateAvailability = InUseCar and c1.position = NoSafeArea)
}

fact inUseCarCannotBeReserved{
        no r1: Reservation,  rd1: Ride | (r1 != rd1.reservation and r1.reservedCar = rd1.reservation.reservedCar and
rd1.state = ActiveRide and r1.expired = False)
}

fact noTwoConcurrentRideSameReservation{
        no rd1, rd2: Ride | (rd1 != rd2 and rd1.reservation = rd2.reservation)
}

fact noTwoConcurrentRideSameCar{
        no rd1, rd2: Ride | (rd1 != rd2 and rd1.state = ActiveRide and rd2.state = ActiveRide and
rd1.reservation.reservedCar = rd2.reservation.reservedCar)
}

fact fee{
        all r1: Reservation | (r1.fee = True) => (no rd1: Ride | rd1.reservation = r1)
}

fact safeAreaForEnededRide{
        all rd1: Ride | (rd1.state = EndedRide) <=> (rd1.endPosition = SafeArea)
}

fact specialArea{
        all rd1: Ride | (rd1.state = EndedRide and rd1.batteryInCharge = True) => (rd1.endPosition = SpecialArea)
        all c1: Car | (c1.batteryInCharge = True) => (c1.position = SpecialArea)
        all sp: SpecialArea | (sp.numberOfPowerGrid <= #{ c1: Car | (c1.batteryInCharge = True and c1.position = sp)})
}
```

```
fact bonusPassenger{
        all rd1: Ride | (rd1.state = EndedRide and rd1.passengersNumber>=1) <=> (one b: BonusPassenger | b in
rd1.bonusSet)
}

fact bonusHighBattery{
        all rd1: Ride | (rd1.state = EndedRide and rd1.batteryEnd = HighBattery) <=> (one b: BonusHighBattery | b in
rd1.bonusSet)
}

fact bonusInCharge{
        all rd1: Ride | (rd1.state = EndedRide and rd1.batteryInCharge = True ) <=> (one b: BonusInCharge | b in
rd1.bonusSet)
}

fact bonusLowbatteryDistance{
        all rd1: Ride | (rd1.state = EndedRide and (rd1.specialAreaTooFar = True or rd1.batteryEnd = LowBattery))
<=> (one b: BonusDistanceLowBattery | b in rd1.bonusSet)
}

fact availableCarInSafeArea{
        all c1: Car | (c1.stateAvailability = AvailableCar) <=> (c1.position = SafeArea and (one sa: SafeArea | c1 in
sa.car))
}

fact availableAndReservedCarInSpecialAreaAreInSpecialAreaSet{
        all c1: Car, sa: SpecialArea | (((c1.position = sa) and (c1.stateAvailability = AvailableCar or c1.stateAvailability
= ReservedCar)) => (c1 in sa.car) )
}

//////////Asserts

assert noPassengersNumberExceedSeats{
        no rd1: Ride | (#rd1.passengersNumber > rd1.reservation.reservedCar.numberOfSeats)
}

assert carCondition{
        no c1: Car | (c1.engineState = True and c1.stateAvailability != InUseCar)
        no c1: Car | (c1.doorState = True and c1.stateAvailability != InUseCar)
        no c1: Car | (c1.batteryInCharge = True and c1.position != SpecialArea)
        no c1: Car | (c1.batteryInCharge = True and c1.engineState = True)
}

assert rideCondition{
        no r1: Ride | (r1.specialAreaTooFar = True and r1.endPosition = SpecialArea)
        no r1: Ride | (r1.batteryInCharge = True and r1.endPosition != SpecialArea)
        no r1: Ride | (r1.state = ActiveRide and (some r1.bonusSet or one r1.endPosition))
        no r1: Ride | (r1.state = EndedRide and (no r1.endPosition))
        no r1: Ride | ((BonusInCharge in r1.bonusSet) and (BonusDistanceLowBattery in r1.bonusSet))
        no r1: Ride | ((BonusHighBattery in r1.bonusSet) and (BonusDistanceLowBattery in r1.bonusSet))
}

assert bonusInChargeSpecialArea{
        all r1: Ride | (r1.endPosition = SpecialArea and r1.batteryInCharge = True => (BonusInCharge in r1.bonusSet))
        all r1: Ride | (r1.endPosition = SpecialArea and r1.batteryInCharge = False => not(BonusInCharge in
r1.bonusSet))
}
```

```
assert carAvailableAreAlwaysInSafeArea{
        all c1: Car | (c1.stateAvailability = AvailableCar) => (c1.position = SafeArea)
}
assert carReservedAreAlwaysInSafeArea{
        all c1: Car | (c1.stateAvailability = ReservedCar) => (c1.position = SafeArea)
}

assert noRideWithoutReservation{
        all rd1: Ride | (one r1: Reservation | rd1.reservation = r1)
}

pred show() {
        #{c1: Car |c1.batteryInCharge = True} >=1
        #{c1: Car |c1.batteryInCharge = False} >=1
        #{c1: Car |c1.position = NoSafeArea} >=1
        #Ride = 1
        #User =2
        #Reservation = 2
}

check rideCondition
check carCondition
check bonusInChargeSpecialArea
check noPassengersNumberExceedSeats
check carReservedAreAlwaysInSafeArea
check carAvailableAreAlwaysInSafeArea
check noRideWithoutReservation

run show for 4
```

4.2 Run of the checks

8 commands were executed. The results are:
 #1: No counterexample found. rideCondition may be valid.
 #2: No counterexample found. carCondition may be valid.
 #3: No counterexample found. bonusInChargeSpecialArea may be valid.
 #4: No counterexample found. noPassengersNumberExceedSeats may be valid.
 #5: No counterexample found. carReservedAreAlwaysInSafeArea may be valid.
 #6: No counterexample found. carAvailableAreAlwaysInSafeArea may be valid.
 #7: No counterexample found. noRideWithoutReservation may be valid.
 #8: **Instance found.** show is consistent.

4.3 Model

One possible solution of the Alloy logic model.
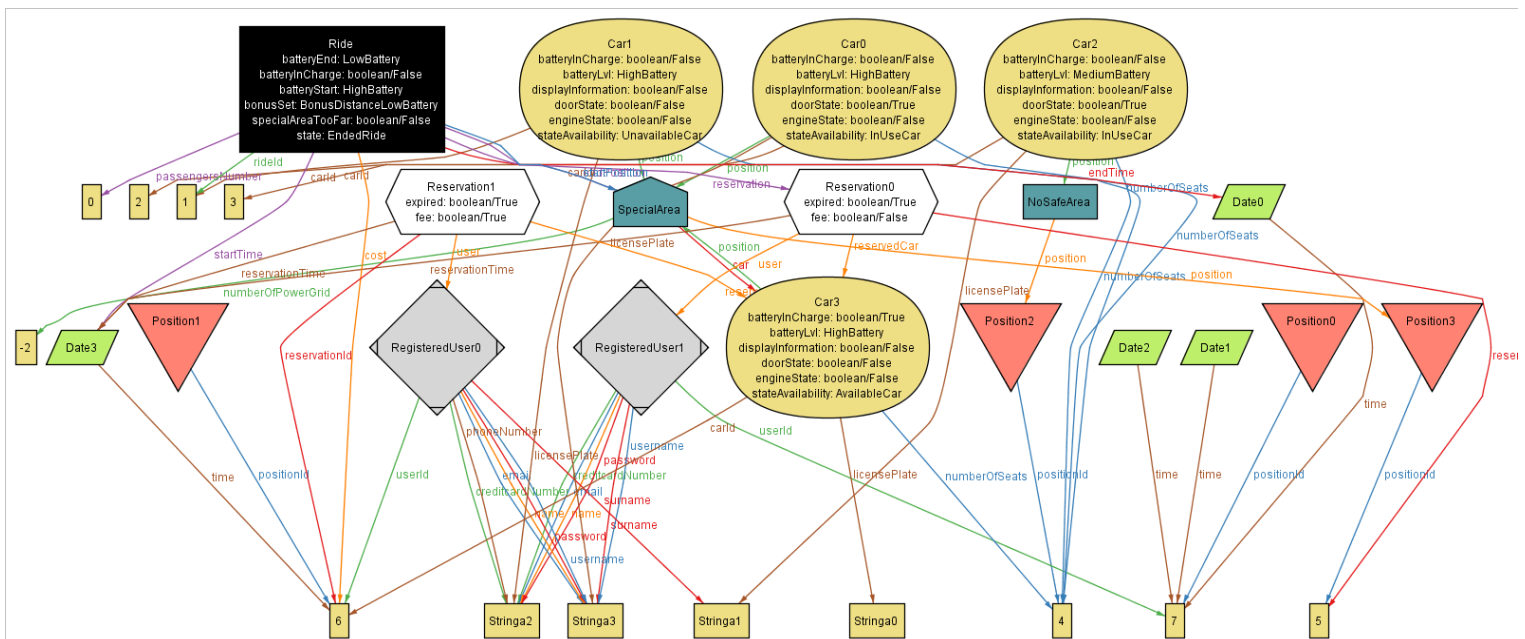
5 Appendix

5.1 Software and tool used
- Microsoft Office Word to redact and format this document (docx format).
- Google Drive and Documents to redact the first copy of this document (it supports multiple access to the same document, so we can work on the same file and see real time changes)
- Astah Professional 7.1.0 (http://astah.net/editions/professional): to create Use Cases Diagrams, Sequence Diagrams, Class Diagrams and State Machine Diagrams.
- Alloy Analyzer 4.2 (http://alloy.mit.edu/alloy/) to prove the consistency of the model.
- Balsamiq Mockups 3.5.5 (http://balsamiq.com/products/mockups/) to create the mockups.
- GitHub for version control.

5.2 Hours of work
- Simone Boglio: 45 hours.
- Lorenzo Croce: 45 hours.