Software Engineering 2 Project: PowerEnJoy
RASD: Requirements Analysis and Specification Document

- Alessio Mongelluzzo
- Michele Ferri
- Mattia Maffioli

Contents

1 Introduction

1.1 Purpose

This document is the Requirement Analysis and Specification Document for the PowerEnJoy application. Its aim is to completely describe the system to be developed and to lay out functional and non-functional requirements, constraints, relationships with the external world, and typical use cases describing user interactions that the software must provide. Moreover, this document will provide formal specification of some features of the applications, written in Alloy language. This document is written for project managers, developers, testers and software designers. It may be used in a contractual agreement between software suppliers and customers.

1.2 Scope

The system supports a car-sharing service employing electric cars.

The system consists of a server application (PowerEnJoy Server), a web application front-end (PowerEnJoy Web) and in a mobile application (PowerEnJoy Mobile).

The system has the customer and the employees of the car-sharing service as its only type of users. It must allow customers to subscribe to the service and identify themselves with their credentials. Only registered users should be able to make use of the service. It must also allow employees to receive notifications about cars that need to be fixed, i.e. cars with a low battery or which are parked outside of a safe area.

The system can show users the location of available cars.

The system allows users to reserve an available car for a limited amount of time. A user has to be able to tell to the system he/she is near the previously reserved car via mobile app or SMS in order to unlock it.

The system should keep track of information related to every active and completed ride.

The system shouldn't allow users to know the position of cars that are not available.

The system allows users to pay for the service at the end of each ride, by communicating with a third party digital payment system. The system should prevent users who failed to pay due to insufficient funds from using the service until they pay off their debt.

The system promotes diligent behaviour of drivers by applying a discount on the fare. The system should charge the user an additional fee when a car is unnecessarily reserved or when, at the end of the ride, the car is left in a state requiring specific intervention by PowerEnJoy employees.

The system interacts with PowerEnJoy employees in order to notice them when a car needs to be recharged.

The system is provided with an optional module:

- **Money saving option** allows the driver to specify the destination of the ride. The system will then suggest an optimal charging area to park at, in order to get a discount on the fare.

1.3 Goals

The goals of the PowerEnJoy software are the following:

1. A person can register to the system.
2. A person can log into the system.
3. Users can know the position of available cars.
4. Users can reserve a car.
5. Users can get into the reserved car.
6. Users can use the reserved car.
7. Users can pay for the car-sharing service.
8. Users can check the current fare while in the car.
9. Parked cars are automatically locked.
10. Discounts are applied to the fare of users who have a virtuous behaviour. Additional charges are applied to users who have an inconvenient behaviour.
11. Users can choose a money saving option.
12. A user who can't pay will be banned.
13. Users can manage their profile.
14. Employees can take care of cars needing an intervention, so that they can be reserved again.

1.4 Definitions, acronyms, and abbreviations

- **RASD**: Requirements Analysis and Specification Document (this document).
- **System**: the car-sharing application to be developed.
- **Back-end**: application logic and APIs elaborating data coming from users and cars. It allows the web and mobile application to be dynamic.
- **Module**: an optional software component which uses the core system APIs to provide additional features.

- **Client**: the web application or the mobile app.
- **Position**: geographic coordinates (latitude, longitude) specifying where an object is located in the world.
- **Car**: PowerEnJoy electric car used for the car-sharing service. It has a unique license plate.
- **Locked car**: car whose doors cannot be opened from the outside. If the doors of a locked car are opened from the inside, the car will remain in the locked state and the doors will lock automatically as soon as they are closed again.
- **Available car**: car which position is visible on the map and can be reserved by any non-banned user from the application.
- **Reserved car**: car which has been reserved by a user from the application. Its position is not visible on the map.
- **Running car**: car which has a running engine and is being driven by the user who has reserved it. Its position is not visible on the map.
- **Low battery car**: car which has less than 20% of total battery charge. A low battery car will be taken care of by a PowerEnJoy employee.
- **Ride**: operation performed by a customer when using a car. A ride begins when the user ignites the engine and ends when the user turns off the engine. Information related to every ride is tracked by the system. During a ride, the car is always a running car.
- **Money saving ride**: special type of ride that can be chosen optionally. In a money saving ride, the user specifies a destination and the system suggests an optimal charging area to park at, in order to get a discount on the fare.
- **Active ride**: ride that is currently in progress.
- **Completed ride**: ride that took place in the past.
- **User** or **Customer**: person who has registered to PowerEnJoy service.
- **Banned**: a user who has been temporarily prevented from using the service.
- **Credential**: unique set of username and password linked to each user.
- **Billing information**: data used to access a third party digital payment system.
- **Reservation**: operation performed by a user who wants to ensure the access to a given car within one hour.

- **Safe area**: place where cars can be legally parked without any risk for a fine. A car which is parked outside of a safe area will be taken care of by a PowerEnJoy employee.
- **Charging area**: station with a power grid and a set of plugs, where electric cars can be recharged. Charging areas are also safe areas.
- **Employee**: a worker for PowerEnJoy whose task is to recharge on-site cars with less than 20% of battery charge, and move to a safe area cars which are parked outside of a safe area.
- **Plug**: a device that allows cars to be recharged at charging areas.
- **DBMS**: Data Base Management System. It is a computer software application used for the administration of data bases.
- **API**: Application Programming Interface.
- **JVM**: Java Virtual Machine.
- **GPS**: Global Positioning System.
- **SMS**: Short Message Service.

1.5 References

This document refers to the project rules of the Software Engineering 2 project and to the RASD assignment. The structure of this document is inspired by that of the IEEE Standard 830-1998 for the format of Software Requirements specifications, with some variations. The main changes have been made in Chapter 2, where some general scenarios have been identified, and in Chapter 3, where the relationships between goals, domain assumptions and requirements have been identified.

1.6 Overview

This document is structured in three parts:

- **Chapter 1: Introduction**. This section explains the target audience, the purpose and the references of the document.
- **Chapter 2: Overall description**. This section provides a general description of the system and its features, constraints, and assumptions about the users and the environment.
- **Chapter 3: Specific requirements**. This section contains all the necessary software requirements to allow a proper implementation.

2 Overall description

2.1 Product perspective

2.1.1 User interfaces

All the user interfaces should be intuitive and user friendly, meaning that they should not require the reading of detailed documentation to be used. There should be very little difference between the user interface of the web application and that of the mobile applications, in order to provide a consistent user experience across all platforms.

2.1.2 Hardware interfaces

The main hardware interface of the system consists in the access to the GPS data in the mobile application. The application also requires Internet connectivity and internal storage access.

2.1.3 Software interfaces

The mobile application must support Android and iOS. The web application works on any web server that supports Java. The back-end stores its data in a DBMS and can run on every platform that supports the JVM.

2.2 Product functions

The system allows users to look for an available car, reserve it, unlock it, drive it, and finally pay for the ride. This is a list of what the users of the service can do.

All users can:

- create an account
- log in
- edit profile data and billing info
- delete their account
- look for available cars around them or near a particular address
- reserve an available car
- tell the system to unlock a car they have reserved
- choose a money saving option

Employees can:

- receive notifications about cars that need to be fixed

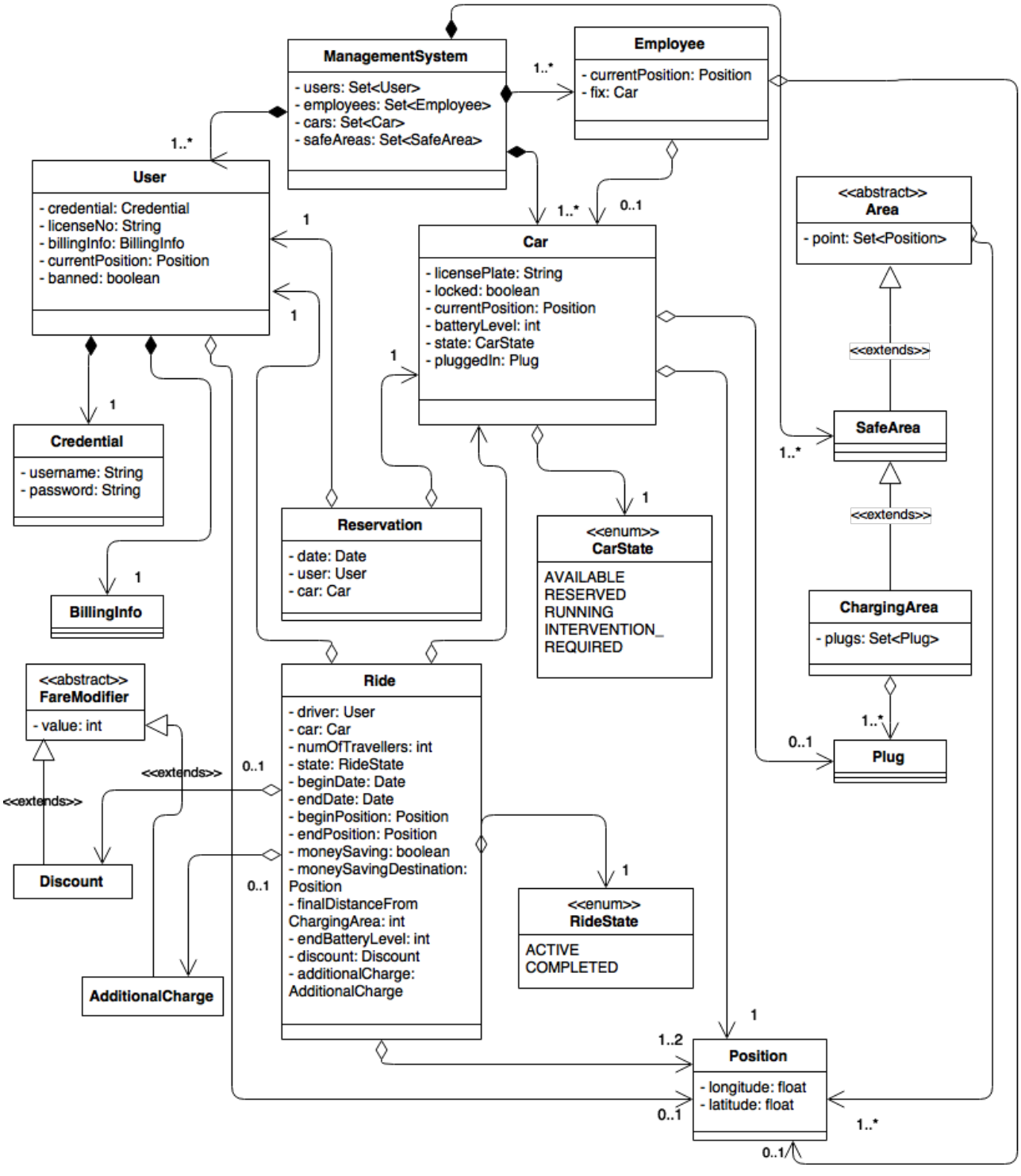

Figure 2.1: The comprehensive class diagram of the system.

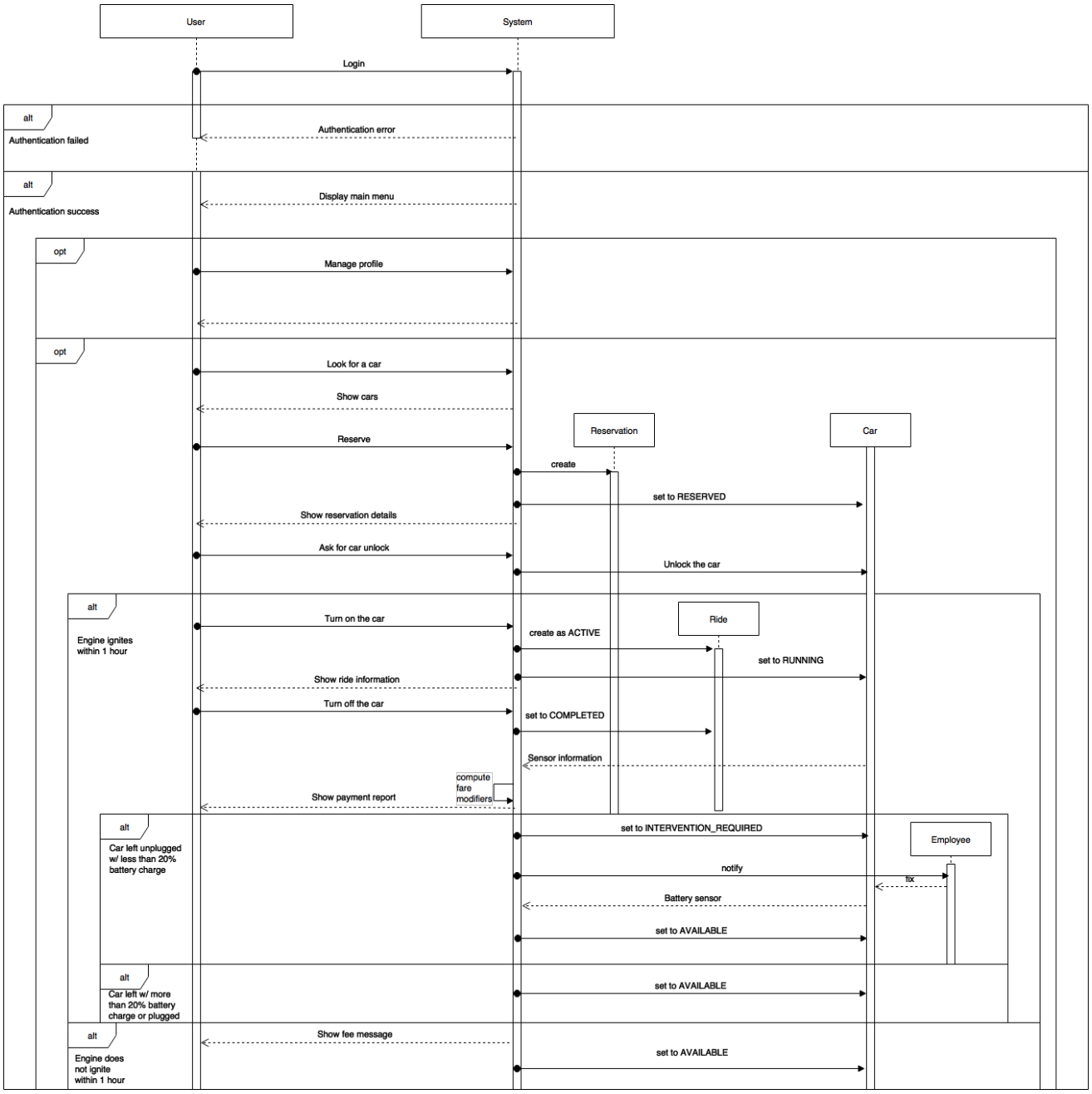

Figure 2.2: General sequence diagram of the system. The scenario where a car is left outside of a safe area is similar to the one where the car is left with less than 20% of battery charge.

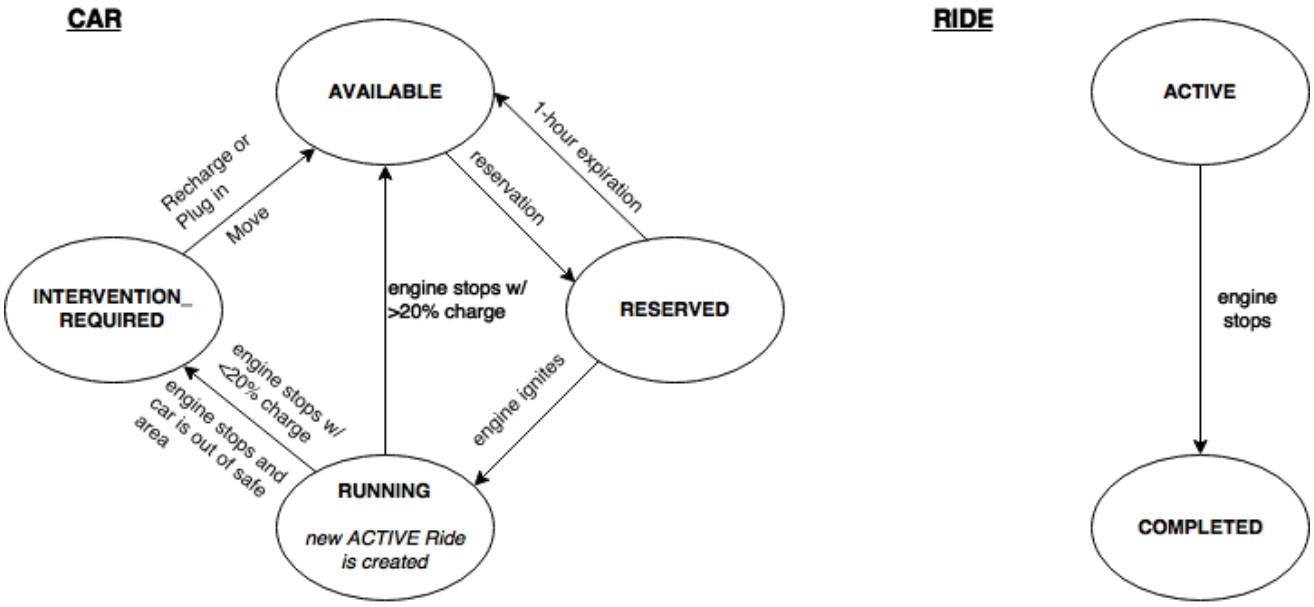

Figure 2.3: Statechart of Car and Ride objects.

2.3 User characteristics

The main user of the system is the customer of the car sharing service. We give for granted that users have access to the Internet. Users can look for a car and reserve it both via the web and the mobile application. Also, PowerEnJoy employees can use the system to receive a push notification when a car needs an intervention.

2.4 Constraints

2.4.1 Regulatory policies

It's user responsibility to ensure that the use of the system complies with the local laws and policies. The system must ask the user for the permission to acquire, store and process personal data and web cookies. The system must offer to the user the possibility to delete all personal data.

2.4.2 Hardware limitations

The system has to run under the following worst-case conditions:

- Mobile applications:
  - 3G connection, at 2 Mb/s
  - 50 MB of free space

  - 1 GB of RAM

- Web application:
  - 2 Mb/s Internet connection
  - 800x600 resolution

2.4.3 Reliability requirements

The system must have a minimum availability of 98%.

2.4.4 Criticality of the application

The system is not employed in life-critical applications.

2.4.5 Parallel operation

Every feature offered by the system must support, on the back-end, simultaneous operations from multiple users.

2.4.6 Safety and security considerations

Information concerning every active or completed ride, as its duration, destination (if any) and position of the car, must be kept private and available only to the user who is driving the car. Also, information concerning every reservation must be kept private and available only to the user who made the reservation.

Only people with a valid driving license must be able to register to the service and use it.

Sensitive user information such as credentials and billing information must be stored securely (i.e. not in plain text).

2.5 Assumptions and dependencies

We assume that:

1. Users and Employees have access to the Internet.
2. Every PowerEnJoy car is equipped with a GPS, which always provides an accurate location.
3. Every PowerEnJoy car is connected to the Internet.
4. Every PowerEnJoy car is equipped with an information display.
5. PowerEnJoy cars' battery level and charge status can be monitored remotely.
6. PowerEnJoy cars' engine status can be monitored remotely.

7. The number of people inside a PowerEnJoy car can always be monitored remotely.
8. PowerEnJoy cars can be locked and unlocked remotely.
9. PowerEnJoy car doors can always be opened from the inside.
10. When a PowerEnJoy employee receives a notification about a car which needs an intervention, he/she will always take care of the issue.
11. PowerEnJoy ensures no mechanical failures can happen to cars.
12. PowerEnJoy ensures a car is always nearby. The number of cars is not fixed and depends on the trend of reservations.
13. Cars are equipped with sensors to check whether doors are open or closed.

2.6 Scenarios identification

Scenario 1

Stefano resides in a student campus in Milan. He wants to go to a concert, but there is a public transport strike until 11 p.m. and unfortunately the concert starts at 10 p.m.; so, he decides to reserve a vehicle with PowerEnJoy to get there. Since he has used the service before, he is already registered and, well aware of the reservation time constraints, he reserves a car parked just 100 meters far at 7.30 p.m.. He leaves at 8.30 p.m. and reaches its destination one hour later. He has already been there, so he knows that there is a power grid station nearby, but unfortunately he finds it full: in fact, several other people have had his same idea of using the car sharing service to bypass the strike. Luckily, he finds a parking spot just 50 meters far from the power grid station and leaves the car there, with 40% of the battery still available. On the way back, the transport strike is finished, so he manages to take the train.

Scenario 2

Giacomo and Francesca need to go to the shopping centre to get food for the week, but their car is broken. They decide to reserve a car with PowerEnJoy to get there. This is their first time using the service, so Giacomo has to register to it; then he logs and reserves a car after having searched among the nearest ones to his position. He also enables the money saving option and inserts the shopping center address. A few minutes later, one of Giacomo's friend, Marco, asks him if he can have a lift to the shopping centre. Giacomo agrees, and he, Francesca and Marco start their trip and get to their destination in just fifeteen minutes, so the car still has 75% of the battery available. Giacomo leaves the car in the power grid station and plugs the car to charge the battery, so he gets a total of 30% discount on the ride. Giacomo reserves the car again, since it will take them just 30 minutes to buy what they need. After having done the shopping, Giacomo, Francesca and Marco go back home and Giacomo leaves

the car in a parking spot, because this time he has forgot to enable the money saving option and he isn't able to find a station. The car still has 75% of the battery available, so he gets a 20% discount on the ride.

Scenario 3

Michele is a college student and he is trying to save money as much as possibile since he is very tight on money at the moment. Therefore he decides to reserve a PowerEnJoy car to go to university enabling the money saving option. He is accustomed to using PowerEnJoy so he just logs in and reserves the nearest car without modifying his billing information. That day the traffic jam gets him stuck for a long time so that by the time he reaches college he is charged with 11EUR. Unfortunately Michele is unaware of having only 10EUR on his prepaid card used as payment method, so the payment transaction goes wrong and the system blocks the user linked to Michele until he will pay off his debt. Michele won't be able to get back home with a PowerEnJoy car by the end of the lessons.

Scenario 4

Alessio is a college student who, during his spare time, goes to the gym. He needs a medical certificate for this activity, so he has arranged a doctor's appointment during the 2 hours break between two lectures on Monday, that is the only gap Alessio could find inside his very busy weekly schedule. On that day, before the first lecture, he thought that he may not get back in time if he took public transport for the whole round trip, so he decided to reserve a car with PowerEnJoy at least to get there much faster. At 10 a.m. he reserves a car near the university. Unfortunately, he has forgotten that car reservation lasts up to one hour before picking it up and the break starts only at 12 a.m.. Alessio receives a fine of 1 EUR for not having abided by the car reservation time constraint and he will have to reserve a car again if he wants to get quickly to his appointment.

Scenario 5

Mattia is a student who is keen on Drum and Bass music. He would really like to go to a gig tonight but his parents left him home alone with no car. He chooses to reserve a PowerEnJoy car. He needs as well to pick his ticket at the mall, which is on the way to the gig. He is in a hurry when he gets out, but he manages to find a very near car. The ticket shop is right at the entrance to the mall, so he just stops his car nearby not to waste much time. He forgets that as he turns off the car PowerEnJoy charges him with the proper fare and turns the car into available to the clients. Fortunately the car is still there once he gets back, but he needs to make another reservation for the same car.

2.7 Future extensions

The system will be implemented foreseeing the possibility of further extensions, for example:

1. Create a fidelity score for every user, which gives them the opportunity to benefit from special offers. The score increases every time the user drives a PowerEnJoy electric car, proportionally with the time spent.

2. Implement a report system so that users can signal to PowerEnJoy vehicles that are damaged, very dirty, or generally not fit for use.

3. Implement a web and/or mobile graphical interface for employees to easily manage the system (cars, safe areas...). For now this kind of data is managed by directly querying the DBMS, because it is rarely modified.

3 Specific requirements

3.1 External interface requirements

3.1.1 User interfaces

- The iOS version of the app must adhere to iOS Human Interface Guidelines.
- The Android version of the app must adhere to Android design guidelines.
- The web pages of the web application must adhere to the W3C standards.

3.1.2 Hardware interfaces

- The app must be able to access the GPS data of the user's device.

3.1.3 Software interfaces

- The system interacts with a driving license database, in order to check whether the license numer provided by the user during the registration is valid or not.
- The system interacts with a web mapping service, in order to retrieve maps when the user looks for a car and when the user needs to be shown the route to the destination (money saving option).
- The system interacts with a third party digital payment system, which at least needs to support transactions involving PayPal and the major credit card companies.
- The core part of the system should allow the building of new interfaces and modules.

3.1.4 Communications interfaces

- The communications between clients and server should be HTTP requests/responses.

3.2 Requirements identification

Goal 1: A person can register to the system.

- Domain assumption 1: Users and Employees have access to the Internet.

R1. On registration, a person must be asked: e-mail address, username, password, name, surname, birth date, address, phone number, driving license number, billing information.

R2. There mustn't be another user already registered with the same e-mail address, username, or driving license number.

R3. The password must be at least 8 characters long and it must contain at least one lowercase character, one uppercase character, and one digit.

R4. The password must be asked twice. The two passwords must match.

R5. The system must send the user a confirmation email with an activation link, and it must effectively insert the new user only when he/she has confirmed his/her registration.

Goal 2: Only registered users can log into the system.

- Domain assumption 1: Users and Employees have access to the Internet.

R6. On login, the system must grant the user access to the account if and only if the following conditions are met:

  (a) The inserted username corresponds to a username of an existing user, or the inserted e-mail corresponds to the registration e-mail of an existing user.

  (b) The inserted password matches with that of the user identified above.

R7. If the entered password is wrong, a new attempt can be made only 10 seconds later.

Goal 3: Users can know the position of available cars.

- Domain assumption 1: Users and Employees have access to the Internet.
- Domain assumption 2: Every PowerEnJoy car is equipped with a GPS, which always provides an accurate location.
- Domain assumption 3: Every PowerEnJoy car is connected to the Internet.
- Domain assumption 10: When a PowerEnJoy employee receives a notification about a car which needs an intervention, he/she will always take care of the issue.
- Domain assumption 12: PowerEnJoy ensures a car is always nearby. The number of cars is not fixed and depends on the trend of reservations.

R8. The user can choose to find cars around his/her position, or around a specified address.

R9. The user must be shown a map of the selected position, on which the locations of AVAILABLE cars are marked.

Goal 4: Users can reserve a car.

- Domain assumption 1: Users and Employees have access to the Internet.
- Domain assumption 3: Every PowerEnJoy car is connected to the Internet.

R10. The system charges the user in advance with 1 EUR deposit. The deposit is given back to the user if he turns on the engine.

R11. If an error occurs while charging the user with the down payment then an error message is displayed and the reservation operation is aborted.

R12. The user must be able to click on a marked location on the map in order to view additional information on the car at that position.

R13. The user must be able to view at least the exact address and the battery level of the selected car.

R14. The user must be able to reserve the selected car by clicking on a button in the additional information screen. After the button has been clicked, the car state must be set to RESERVED and a new reservation must be created associated to the user and the selected car.

R15. The user must not be able to reserve the selected car if there already exists a reservation made by the user himself/herself.

Goal 5: Users can get into the reserved car.

- Domain assumption 1: Users and Employees have access to the Internet.
- Domain assumption 3: Every PowerEnJoy car is connected to the Internet.
- Domain assumption 8: PowerEnJoy cars can be locked and unlocked remotely.

R16. Any user who reserved a car in the last hour must be able to unlock the car he/she has reserved in one of the following ways:

  (a) If the reservation has been made via mobile app, the user must be able to unlock the car by clicking on a button in the reservation screen of the mobile app.

  (b) If the reservation has been made via web app, the user must be able to unlock the car by sending an SMS.

Goal 6: Users can use the reserved car.

- Domain assumption 11: PowerEnJoy ensures no mechanical failures can happen to cars.
- Requirement 16: Any user who reserved a car in the last hour must be able to tell the system to unlock the car he/she has reserved.

Goal 7: Users can pay for the car-sharing service.

- Domain assumption 1: Users and Employees have access to the Internet.
- Domain assumption 3: Every PowerEnJoy car is connected to the Internet.
- Domain assumption 4: Every PowerEnJoy car is equipped with an information display.
- Domain assumption 6: PowerEnJoy cars' engine status can be monitored remotely.

R17. The system must keep track of the time spent from when the engine turns on until the engine turns off. This is done by means of the Ride object: whenever a car engine is ignited, a new ACTIVE Ride must be created, which is associated to the car and the user who reserved it, and contains the time at which the engine has been ignited; whenever a car engine is stopped, the ride for that car must be set to COMPLETED and the time at which the engine has been stopped must be recorded.

R18. When a ride is COMPLETED, the system waits for possible discounts to be applied to the final fare (see Goal 10), then updates the state of the car to either AVAILABLE or INTERVENTION_REQUIRED and interacts with the third party payment service, notifying the user with the outcome of the payment operation.

Goal 8: Users can check the current fare while in the car.

- Domain assumption 4: Every PowerEnJoy car is equipped with an information display.
- Requirement 17: The system must keep track of the time spent from when the engine turns on until the engine turns off.

Goal 9: Parked cars are automatically locked.

- Domain assumption 3: Every PowerEnJoy car is connected to the Internet.
- Domain assumption 6: PowerEnJoy cars' engine status can be monitored remotely.
- Domain assumption 8: PowerEnJoy cars can be locked and unlocked remotely.
- Domain assumption 9: PowerEnJoy car doors can always be opened from the inside.
- Domain assumption 13: Cars are equipped with sensors to check whether doors are open or closed.

R19. The system locks the car if it detects that nobody is inside the vehicle and doors are closed.

R20. If the system fails in locking the car (e.g. doors are not closed) then a warning message in sent to the respective user.

R21. If a user unlocks a locked car from the inside, then the system locks the car again.

Goal 10: Discounts are applied to the fare of users who have a virtuous behaviour. Additional charges are applied to users who have an inconvenient behaviour.

- Domain assumption 2: Every PowerEnJoy car is equipped with a GPS, which always provides an accurate location.
- Domain assumption 3: Every PowerEnJoy car is connected to the Internet.
- Domain assumption 5: PowerEnJoy cars' battery level and charge status can be monitored remotely.
- Domain assumption 8: PowerEnJoy cars' engine status can be monitored remotely.
- Domain assumption 9: The number of people inside a PowerEnJoy car can always be monitored remotely.

R22. If the car is detected to be at a charging area, then the system must wait for a fixed timeout of 2 minutes before charging the user.

R23. If the number of people inside the car is measured to be 3 or more, the system must apply a 10% discount on the final fare.

R24. If the battery level at the end of the ride is measured to be at least 50% of the total charge, the system must apply a 20% discount on the final fare.

R25. If the position of the car at the end of the ride is a charging area, and the car is plugged in within a 2 minutes time frame, the system must apply a 30% discount on the final fare.

R26. If a ride is eligible for multiple discounts, only the largest is effectively applied to the final fare.

R27. If the position of the car at the end of the ride is more than 3 km away from the nearest charging area or if the battery level is lower than 20% of the total charge, the system must apply a 30% additional charge on the final fare.

R28. Any user who doesn't ignite the engine of the car he/she has reserved within an hour is charged with a fee of 1 EUR and his/her reservation is deleted.

Goal 11: Users can choose a money saving option.

- Domain assumption 2: Every PowerEnJoy car is equipped with a GPS, which always provides an accurate location.
- Domain assumption 4: Every PowerEnJoy car is equipped with an information display.
- Requirement 26: If the position of the car at the end of the ride is a charging area, and the car is plugged in within a 2 minutes time frame, the system must apply a 30% discount on the final fare.

R29. The user must be able to enable a money saving option as soon as he/she enters the car, by entering a specific destination.

R30. The system estimates at which charging area the user has to park the car, depending on the specified destination and plug availability, in order to achieve a uniform distribution of cars.

R31. The optimal route to reach the charging area is displayed to the user.

Goal 12: A user who can't pay will be banned.

R32. If the payment operation of a user fails due to insufficient funds, then that user is banned until he/she pays it off.

R33. If a user is banned, the system must prevent him/her from having access to the service.

Goal 13: Users can manage their profile.

- Domain assumption 1: Users and Employees have access to the Internet.

R34. The user must be able to view the information in his/her profile.

R35. The user must be able to edit the information in his/her profile, according to information fields' constraints.

R36. The user must be able to delete his/her profile.

Goal 14: Employees can take care of cars needing an intervention, so that they can be reserved again.

- Domain assumption 1: Users and Employees have access to the Internet.
- Domain assumption 2: Every PowerEnJoy car is equipped with a GPS, which always provides an accurate location.
- Domain assumption 3: Every PowerEnJoy car is connected to the Internet.
- Domain assumption 5: PowerEnJoy cars' battery level and charge status can be monitored remotely.

- Domain assumption 6: PowerEnJoy cars' engine status can be monitored remotely.
- Domain assumption 8: PowerEnJoy cars can be locked and unlocked remotely.
- Domain assumption 10: When a PowerEnJoy employee receives a notification about a car which needs an intervention, he/she will always take care of the issue.

R37. The system must generate special credential for employees.

R38. Employees can lock and unlock every car.

R39. Whenever a car's state is INTERVENTION_REQUIRED, an employee which is not fixing any car at the moment receives a notification containing information about the issue.

3.2.1 Other requirements

R40. The system must store information about users, employees, safe/charging areas, and cars into a database.

3.3 System features

3.3.1 Registration

Actor	Person, System.
Goals	Goal 1: A person can register to the system.
Input condition	/
Event Flow	1. Person clicks the ”Register” button on the home page of PowerEnJoy app or website and is redirected to the registration form page. 2. Person fills the form with his email, a chosen password, his personal data and billing information. 3. Person accepts the terms and conditions of use of PowerEnJoy service. 4. Person clicks on the ”confirm” button. 5. Person accesses the provided email to verify it. 6. System stores information provided by person. 7. Person is shown a page that confirms registration has succesfully completed.
Output condition	Person becomes a user of the system and is now able to log into PowerEnJoy
Exception	• Person fills the form with invalid information: system highlights the fields with invalid information and informs person that these fields have to be filled with valid information. • Form is filled with information belonging to an already existing account: system informs the person that there already exists a user associated to provided information and prevents person to register with provided information. • Person does not fill all the mandatory fields of the form: system highlights the fields with missing information and informs person that these fields have to be filled.

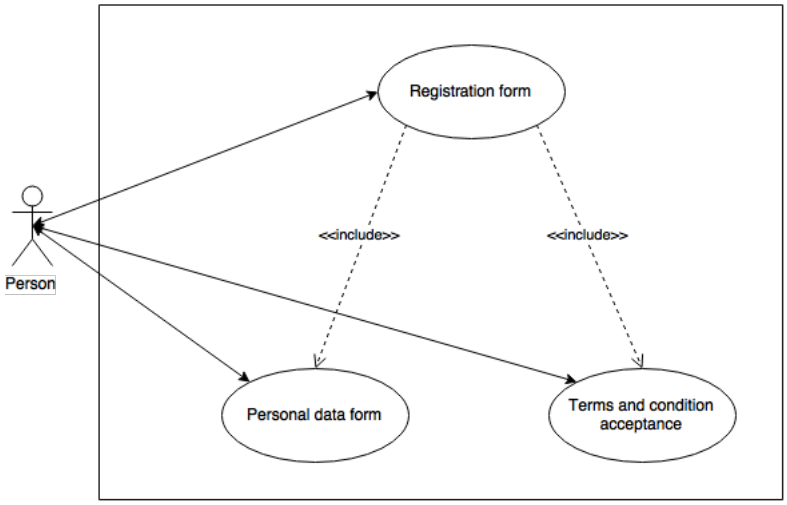

Figure 3.1: Use case diagram for the registration feature.

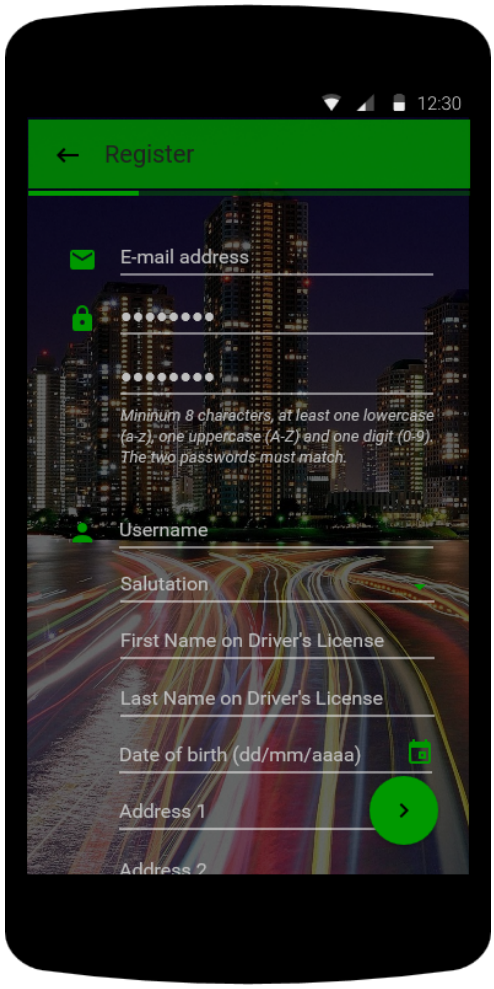

Figure 3.2: Mockup for the registration feature.

3.3.2 Login

Actor	User, Employee, System.
Goals	Goal 2: Only registered users can log into the system.
Input condition	/
Event Flow	1. User or Employee completes the form in the home page of PowerEnJoy app or website by inserting his username and password. 2. User or Employee clicks the ”login” button on the page.
Output condition	User or Employee's credentials are validated and he can access to PowerEnJoy services.
Exception	• User types an incorrect username and/or password and the system notifies him the error, by asking his credentials again. • If user is banned and tries to login, the system prevents him from performing this operation and instead shows him an error page.

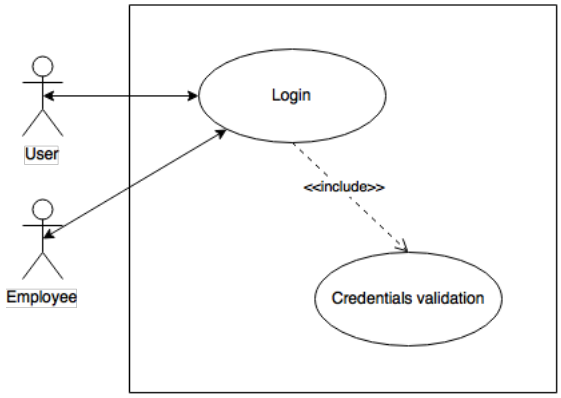

Figure 3.3: Use case diagram for the login feature.

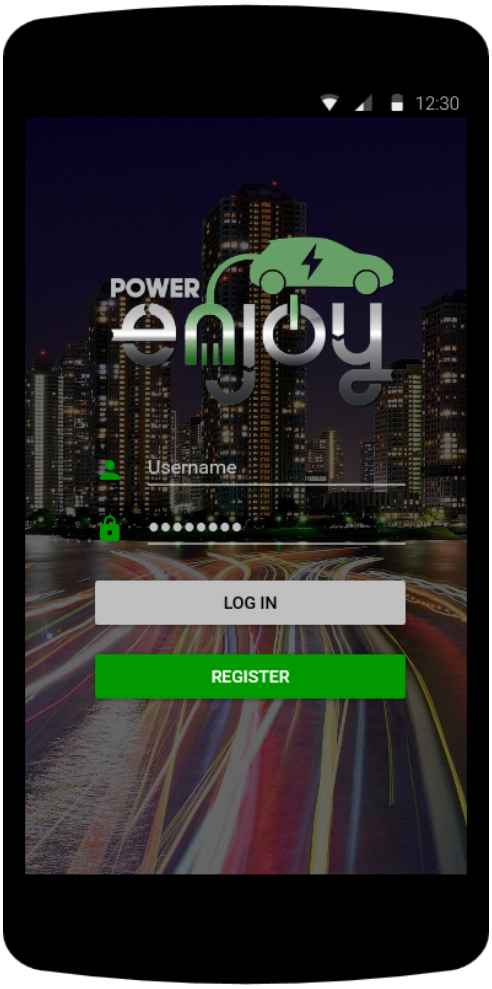

Figure 3.4: Mockup for the login feature.

3.3.3 Car lookup

Actor	User, System.
Goals	Goal 3: Users can know the position of available cars.
Input condition	User is logged into the app or the website.
Event Flow	1. User accesses the ”Find a car” area. 2. User inserts an address, otherwise by default his current address is found by the GPS and inserted. 3. User clicks on the ”Find a car” button.
Output condition	User is shown a page with the location of all the AVAILABLE cars nearby the selected position.
Exception	• User inserts an incorrect address and the system notifies him the error, by asking an address again.

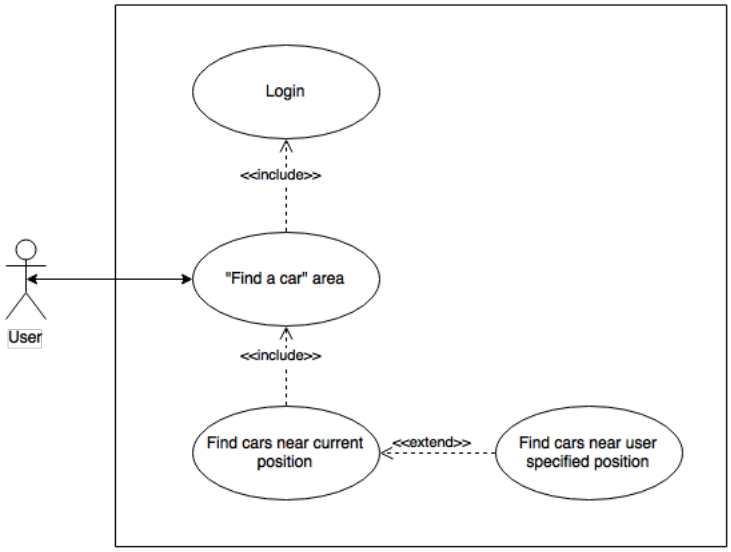

Figure 3.5: Use case diagram for the car lookup feature.

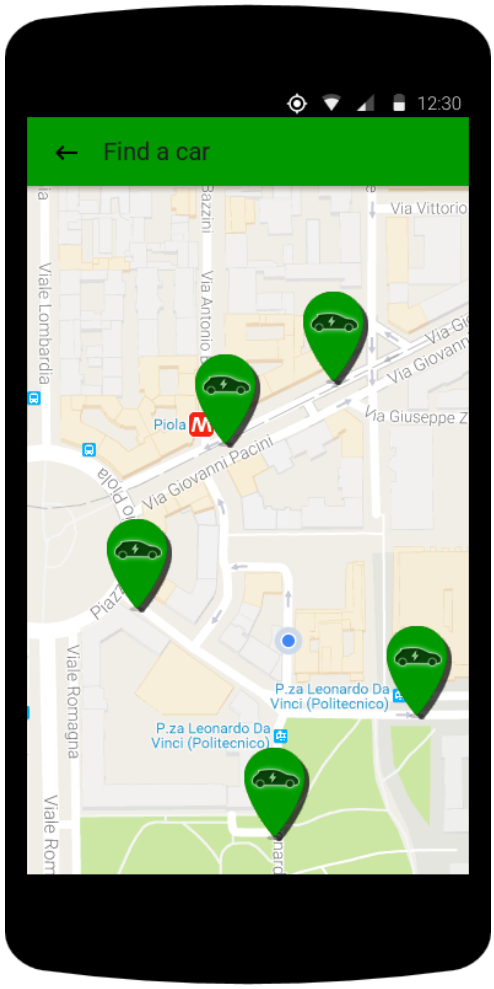

Figure 3.6: Mockup for the car lookup feature.

3.3.4 Car reservation

Actor	User, System.
Goals	Goal 4: Users can reserve a car.
Input condition	User looked for available cars nearby a position.
Event Flow	1. User chooses one of the available cars and selects it by clicking on it. 2. User is redirected to the "Reserve a car" page where he is shown the model name of the reserved car, its battery level and location. 3. User confirms the reservation request by clicking on the "√" button. 4. User is charged with 1 EUR deposit. 5. User is notified in another page that the operation was succesful.
Output condition	The system flags the car selected by the user as reserved and the user can unlock it.
Exception	• The operation is interrupted if the user cannot successfully make 1 EUR down payment. • The car the user is trying to reserve is not available anymore, the system notifies him the error and automatically refreshes the available cars page, so that the user can choose a different car.

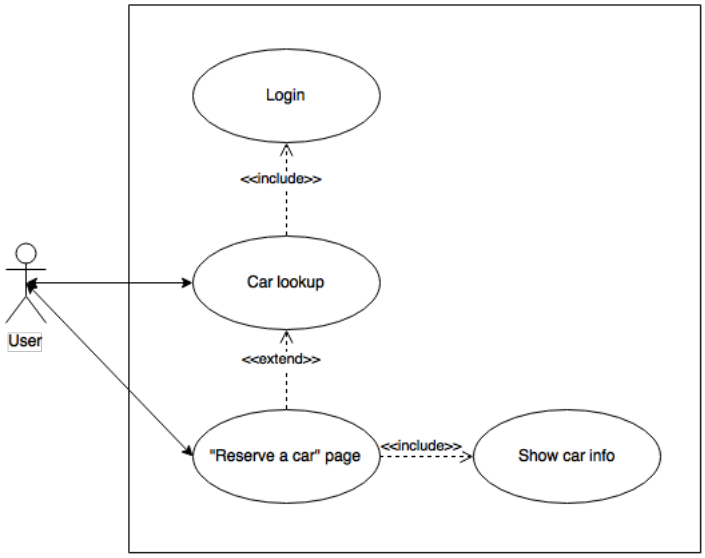

Figure 3.7: Use case diagram for the car reservation feature.

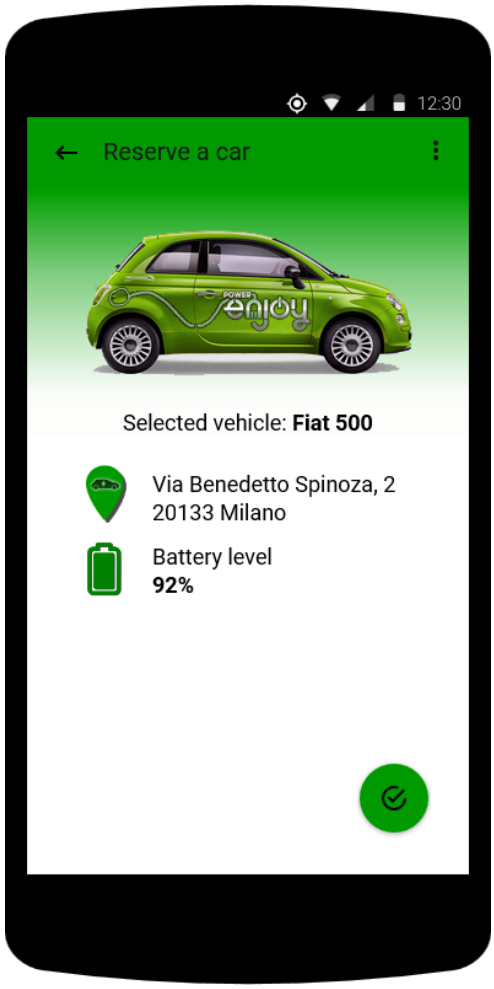

Figure 3.8: Mockup for the car reservation feature.

3.3.5 Car lock and unlock

Actor	User, System.
Goals	Goal 5: Users can get into the reserved car. Goal 9: Parked cars are automatically locked.
Input condition	User reserved a car from the PowerEnJoy website or app.
Event Flow	1. User accesses the "My reservation" area. 2. User selects the reserved car and clicks the "unlock" button. 3. Car doors are automatically locked by the system as soon as the user gets out of the vehicle and closes the doors.
Output condition	User can use the reserved car and when he has finished using it, it becomes available again or it is picked up by a PowerEnjoy employee if not parked in a safe area and/or with less than 20% battery charge of the total.
Exception	• The system cannot successfully lock a car and a warning message is sent to the user's profile.

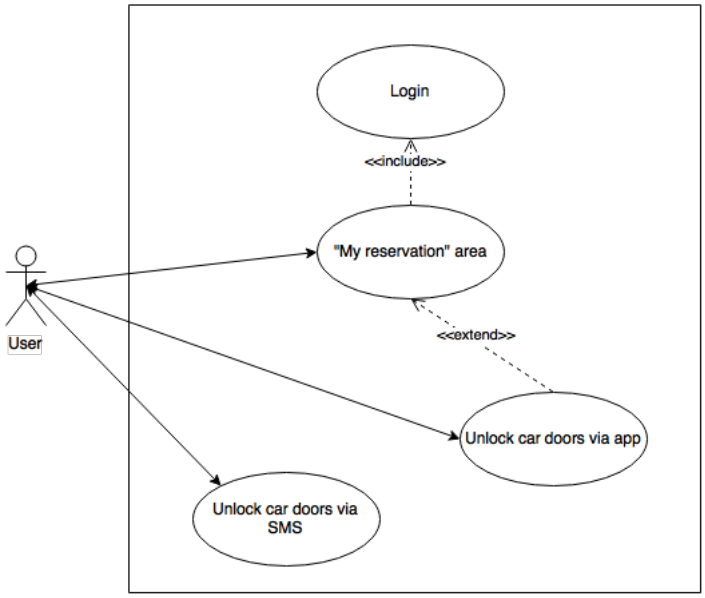

Figure 3.9: Use case diagram for the car unlock feature.

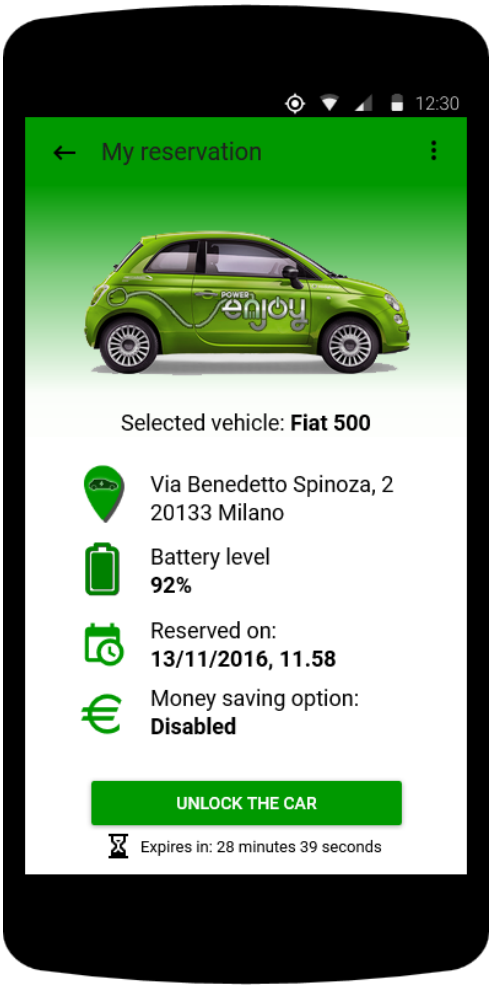

Figure 3.10: Mockup for the car unlock feature.

3.3.6 Price discounts/increases management

Actor	System.
Goals	Goal 10: Discounts are applied to the fare of users who have a virtuous behaviour. Additional charges are applied to users who have an inconvenient behaviour.
Input condition	User parked the car he was using and the system saved the number of the people that were inside the car.
Event Flow	1. The system checks if the position of the car in at least 3 km far from that of the nearest charging station: if this is the case the system increases the fare for the ride by 30%. 2. The system checks if the battery level of the car is lower than 20% of the total: if this is the case the system increases the fare for the ride. 3. The system checks if the position of the car coincides with that of a charging station and if the sensor signals the car is plugged within 2 minutes: if this is the case the system decreases the fare for the ride by 30% and skips controls relative to lower discounts. 4. The system checks if the sensor signals that the battery level of the car is at least 50% of the total: if this is the case the system decreases the fare for the ride by 20% and skips controls relative to lower discounts. 5. The system checks if the number of the people that were inside the car is at least equal to 3: if this is the case the system decreases the fare for the ride by 10% and skips controls relative to lower discounts.
Output condition	The fare is possibly increased or decreased by the appropriate amount according to the user's behaviour.
Exception	/

3.3.7 Money saving option

Actor	User, System.
Goals	Goal 11: Users can choose a money saving option.
Input condition	User reserved a car from the PowerEnJoy website or app.
Event Flow	1. User clicks on the money saving option "€" button directly from the car information display or from the "My reservation" area in the website or in the app. 2. User inserts his destination by typing its address. 3. The system estimates a charging station position by taking into account the distance from the user's destination and the available plugs.
Output condition	Money saving option is enabled and the user's car destination is automatically set to the charging station estimated by the system.
Exception	/

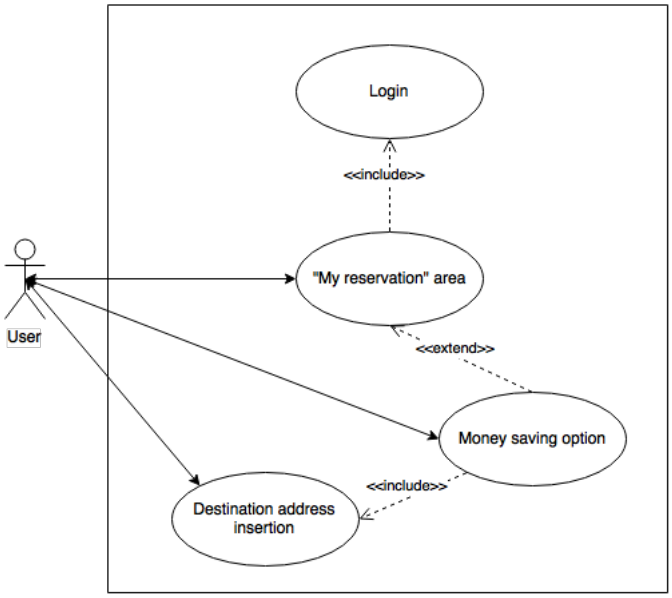

Figure 3.11: Use case diagram for the money saving option feature.

3.3.8 Electronic payment

Actor	System, Third party digital payment system.
Goals	Goal 4: Users can reserve a car. Goal 7: Users can pay for the car-sharing service. Goal 12: A user who can't pay will be banned.
Input condition	Price discounts and increase management possibly increased or decreased the fare by the appropriate amount according to the user's behaviour.
Event Flow	1. The system sends the payment data to the third party digital payment system and waits for the outcome of the payment operation. 2. The system receives the succesful outcome of the payment operation.
Output condition	The user's bill is decreased by the fare amount and he can check the receipt on his account
Exception	• The system receives an unsuccessful outcome w.r.t. a reservation deposit, then the transaction is aborted and the user is not banned. • The system receives an unsuccesful outcome w.r.t. a ride from the third party digital payment system because there are not enough funds on the user's bill to perform the payment operation and the user's account is banned until the system receives a succesful outcome.

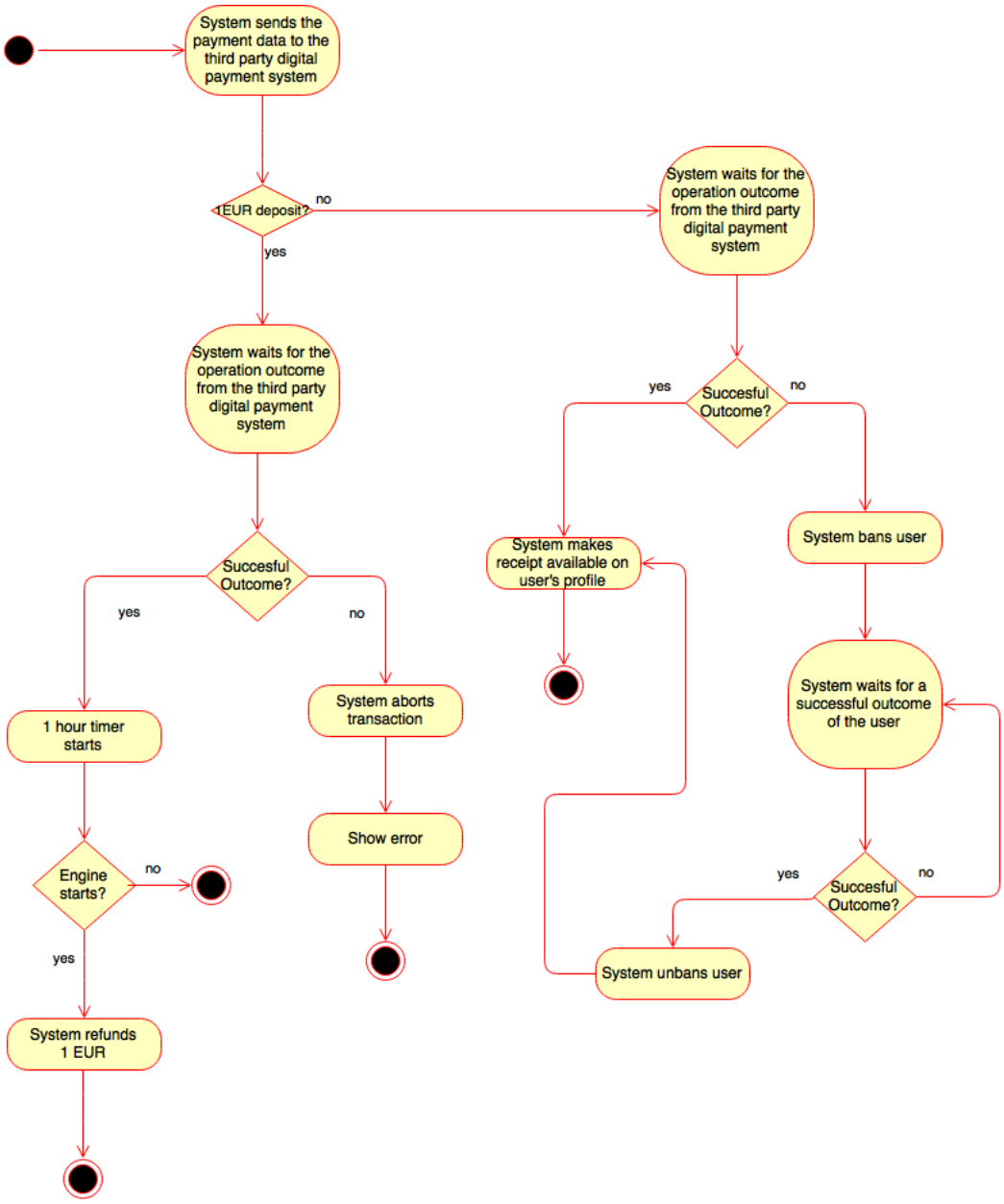

Figure 3.12: Activity diagram for the electronic payment feature.

3.3.9 Car pickup request

Actor	Employee, System.
Goals	Goal 6: Users can use the reserved car. Goal 14: Employees can take care of cars needing an intervention, so that they can be reserved again.
Input condition	A car is parked outside of a safe area and/or with a low battery level, not plugged in.
Event Flow	1. The system sends a push notification to the employee that is nearest to the car and available (i.e. he/she is not fixing any other car). 2. The employee gets there and unlocks the car through the app. 3. The system estimates the nearest power grid station and automatically sets it as the destination of the car. 4. If the battery level is not enough to reach the destination, the employee charges the car in place. 5. The employee drives the car to the destination and plugs the car in.
Output condition	The car is put into charge in a power grid station.
Exception	/

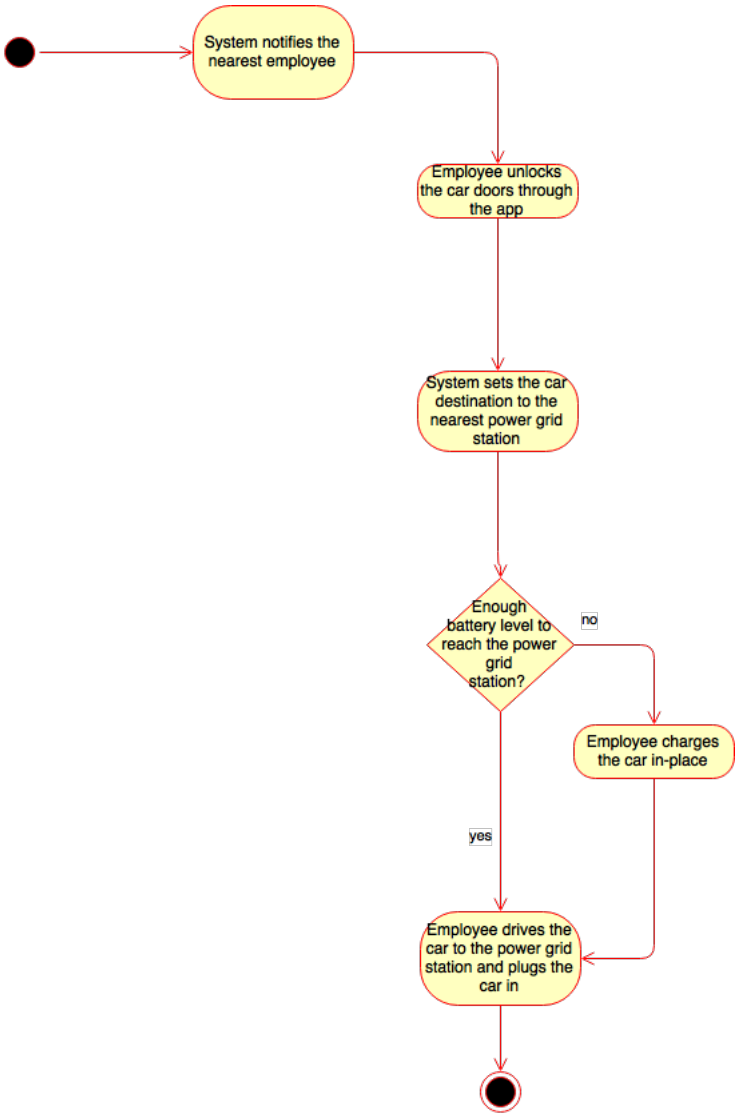

Figure 3.13: Activity diagram for the car pickup feature.

3.3.10 User profile visualization

Actor	User, System.
Goals	Goal 13: Users can manage their profile.
Input condition	User is logged in.
Event Flow	1. User selects the ”profile settings” button. 2. A dropdown menu with ”View profile” and ”Delet profile” options is shown to the user. 3. User selects ”View profile” option.
Output condition	User is shown his profile page and can check his profile information.
Exception	/

3.3.11 User profile modification

Actor	User, System.
Goals	Goal 13: Users can manage their profile.
Input condition	User is viewing his profile page.
Event Flow	1. User selects the ”Edit profile” button. 2. User is shown the profile edit page. 3. User changes the contents of the fields he want to edit. 4. User selects the ”Save changes” button at the end of the page.
Output condition	User profile information is updated.
Exception	If the user fills at least one of the fields with invalid strings and then he tries to save changes, the system notifies him the error and shows him his profile unedited.

3.3.12 User profile deletion

Actor	User, System.
Goals	Goal 13: Users can manage their profile.
Input condition	User is logged in.
Event Flow	1. User selects the ”profile settings” button. 2. A dropdown menu with ”View profile” and ”Delet profile” options is shown to the user. 3. User selects ”Delete profile” option. 4. User is asked to confirm his choice by clicking ”Yes” or ”No” in a pop-up. 5. User clicks the ”Yes” button
Output condition	User is logged out and his profile is deleted from the system.
Exception	If the user tries to delete his profile while having reserved a car, the system notifies him that it is not possible to delete a profile that is reserving a car.

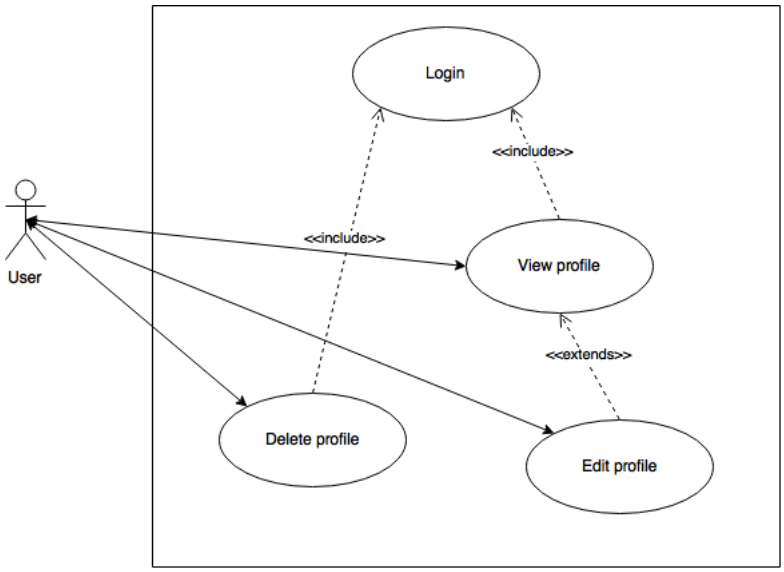

Figure 3.14: Use case diagram for the user profile management.

3.4 Performance requirements

- The system must support at least 1000 simultaneously active rides.
- Any request must be processed in less than 5 seconds.
- The payment operation should be processed in less than 5 seconds after the end of the ride.

3.5 Software system attributes

3.5.1 Reliability

- The reliability of the system is related to the one of the server it runs on.
- The system should grant a MTTF (Mean Time To Failure) greater than 1 year.
- The system should grant a MTTR (Mean Time To Repair) smaller than 24 hours.

3.5.2 Availability

- The system must have a minimum availability of 98%.

3.5.3 Security

- All the communications must be protected by encryption using the TLS (Transport Layer Security) protocol.
- Passwords must not be stored in plain text in the database, but should be hashed and salted instead.

3.5.4 Maintainability

- The code should be documented using JavaDoc.
- The system must support modules extensions, keeping a stable core decoupled from the modules.

3.5.5 Portability

- The software must be written in Java. It should run on every platform that supports the JVM.

3.6 Alloy

```
// Alloy model for PowerEnJoy system.

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

// Dates are expressed as the number of seconds
// that have passed from January 1, 1970,
// because other signatures are not comparable.

// === SIGNATURES ===

sig Stringa {}
sig Float {}
sig BillingInfo {}
sig Plug {}

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

abstract sig Area {
    point: some Position
    }

sig SafeArea extends Area {}

sig ChargingArea extends SafeArea {
    hasPlug: some Plug
    }

one sig ManagementSystem {
    registeredUser: some User,
    registeredCar: some Car,
    registeredEmployee: some Employee,
    safeArea: some SafeArea
    }

sig Credential {
    username: one Stringa,
    password: one Stringa
    }{
    username != password
    }

sig Employee {
    position: one Position,
    fix: some Car
```

```
    }{
    fix.state = INTERVENTION_REQUIRED
    }

/*
 * This signature represents a registered User.
 */
sig User {
    credential: one Credential,
    licenseNo: one Stringa,
    billingInfo: one BillingInfo,
    currentPosition: lone Position,
    banned: one Bool
    }

abstract sig CarState {}
one sig AVAILABLE extends CarState {}
one sig RESERVED extends CarState {}
one sig RUNNING extends CarState {}
one sig INTERVENTION_REQUIRED extends CarState {}

sig Car {
    licensePlate: one Stringa,
    locked: one Bool,
    currentPosition: one Position,
    batteryLevel: one Int,
    state: one CarState,
    pluggedIn: lone Plug
    }{
    batteryLevel >= 0
    batteryLevel =< 100
    batteryLevel < 20 and state != RUNNING implies state = INTERVENTION_REQUIRED
    state != RUNNING and all a: SafeArea | currentPosition & a.point = none
        implies state = INTERVENTION_REQUIRED
    state = AVAILABLE implies locked = True
    state = RESERVED <=> one r: Reservation | r.selectedCar = this
    state = RUNNING <=> one r: Ride | r.state = ACTIVE and r.car = this
    state = INTERVENTION_REQUIRED implies locked = True
    }

sig Reservation {
    date: one Int,
    madeBy: one User,
    selectedCar: one Car
    }{
    date > 0
    }

abstract sig RideState {}
one sig ACTIVE extends RideState {}
```

```
one sig COMPLETED extends RideState {}

/*
 *      Fare modifiers are expressed in percentage.
 *      Discounts are expressed as a positive integer between 1 and 100.
 *      Additional charges are represented as a "negative discount",
 *      so a negative integer between -infinity and -1.
 */
abstract sig FareModifier {
    value: one Int
    }{
    value <= 100
    }
abstract sig Discount extends FareModifier {}{
    value >= 1
    }
abstract sig AdditionalCharge extends FareModifier {}{
    value <= -1
    }

// Discounts
one sig THREE_PEOPLE_DISCOUNT extends Discount {}{
    value = 10
    }
one sig HIGH_BATTERY_DISCOUNT extends Discount {}{
    value = 20
    }
one sig PLUGGED_CAR_DISCOUNT extends Discount {}{
    value = 30
    }

// Additional charges
one sig LOW_BATTERY_ADDITIONAL_CHARGE extends AdditionalCharge {}{
    value = -30
    }
one sig HIGH_DISTANCE_ADDITIONAL_CHARGE extends AdditionalCharge {}{
    value = -30
    }

sig Ride {
    driver: one User,
    car: one Car,
    numOfTravellers: one Int,
    state: one RideState,
    beginDate: one Int,
    endDate: lone Int,
    beginPosition: one Position,
    endPosition: lone Position,
    finalDistanceFromChargingArea: lone Int,
    endBatteryLevel: lone Int,
```

```
    moneySaving: one Bool,
    moneySavingDestination: lone Position,
    discount: lone Discount,
    additionalCharge: lone AdditionalCharge,
    paymentSuccessful: lone Bool
    }{
    numOfTravellers > 0
    numOfTravellers <= 4
    beginDate > 0
    endBatteryLevel >= 0
    endBatteryLevel <= 100
    finalDistanceFromChargingArea >= 0
    endDate != none implies endDate > beginDate
    state = ACTIVE implies discount = none and additionalCharge = none
    state = COMPLETED <=> endPosition != none
    state = COMPLETED <=> endDate != none
    state = COMPLETED <=> paymentSuccessful != none
    state = COMPLETED <=> endBatteryLevel != none
    state = COMPLETED <=> finalDistanceFromChargingArea != none
    moneySaving = True implies some a: ChargingArea |
        moneySavingDestination & a.point != none
    moneySaving = False implies moneySavingDestination = none
    paymentSuccessful = False implies driver.banned = True
    discount != none implies state = COMPLETED
    additionalCharge != none implies state = COMPLETED
    }

// === FACTS ===
fact licensePlatesAreUnique {
    all c1, c2: Car | (c1 != c2) => c1.licensePlate != c2.licensePlate
    }

fact drivingLicensesAreUnique {
    all u1, u2: User | (u1 != u2) => u1.licenseNo != u2.licenseNo
    }

fact usernamesAreUnique {
    all c1, c2: Credential | (c1 != c2) => c1.username != c2.username
    }

fact credentialsAreUnique {
    all u1, u2: User | (u1 != u2) => u1.credential != u2.credential
    }

fact billingInfoIsUnique {
    all u1, u2: User | (u1 != u2) => u1.billingInfo != u2.billingInfo
    }

fact plugCannotBelongToDifferentChargingAreas {
    no disjoint c1, c2: ChargingArea | c1.hasPlug & c2.hasPlug != none
```

```
    }

fact carsCannotBePluggedInTheSamePlug {
    all c1, c2: Car | ((c1 != c2) and (c1.pluggedIn != none) and
        (c2.pluggedIn != none)) implies c1.pluggedIn & c2.pluggedIn = none
    }

fact chargingCarIsAtChargingArea {
    all c: Car | c.pluggedIn != none implies
        c.currentPosition & c.pluggedIn.(~hasPlug).point != none
    }

fact noOverlappingPositions {
    all p1, p2: Position | (p1 != p2) implies (p1.latitude != p2.latitude) ||
        (p1.longitude != p2.longitude)
    }

fact carsCannotHaveTheSamePosition {
    all c1, c2: Car | (c1 != c2) implies c1.currentPosition != c2.currentPosition
    }

fact userHasSamePositionAsCarHeIsDriving {
    all r: Ride | r.state = ACTIVE implies
        r.driver.currentPosition = r.car.currentPosition
    }

fact userHasOnlyOneReservation {
    no disjoint r1, r2: Reservation | r1.madeBy = r2.madeBy
    }

fact carHasOnlyOneReservation {
    no disjoint r1, r2: Reservation | r1.selectedCar = r2.selectedCar
    }

fact userHasOnlyOneActiveRide {
    no disjoint r1, r2: Ride | r1.state = ACTIVE and r2.state = ACTIVE and
        r1.driver = r2.driver
    }

fact carHasOnlyOneActiveRide {
    no disjoint r1, r2: Ride | r1.state = ACTIVE and r2.state = ACTIVE and
        r1.car = r2.car
    }

fact ridesDoNotOverlapForSameUserOrCar {
    all r1, r2: Ride | {
            (r1 != r2) and (r1.driver = r2.driver or r1.car = r2.car) and
                (r1.state = COMPLETED) and (r2.state = COMPLETED) implies
                r1.endDate < r2.beginDate or r2.endDate < r1.beginDate
            (r1 != r2) and (r1.driver = r2.driver or r1.car = r2.car) and
```

```
                (r1.state = ACTIVE) and (r2.state = COMPLETED) implies
                r2.endDate < r1.beginDate
            (r1 != r2) and (r1.driver = r2.driver or r1.car = r2.car) and
                (r1.state = COMPLETED) and (r2.state = ACTIVE) implies
                r1.endDate < r2.beginDate
            }
    }

fact bannedUserHasNoReservationOrActiveRide {
    all u: User | u.banned = True implies (no r: Ride | r.state = ACTIVE and
        r.driver = u) and (no r: Reservation | r.madeBy = u)
    }

fact bannedUsersHaveExactlyOneUnsuccessfulPayment {
    all u: User | u.banned = True implies (one r: Ride | (r.driver = u) and
        (r.paymentSuccessful = False))
    }

fact noRidesAfterUnsuccessfulPayment {
    all r1, r2: Ride | r1 != r2 and r1.driver = r2.driver and
        r1.paymentSuccessful = False implies r1.beginDate > r2.endDate
    }

fact areasOfTheSameTypeDoNotOverlap {
    no disjoint s1, s2: SafeArea | s1.point & s2.point != none
    no disjoint c1, c2: ChargingArea | c1.point & c2.point != none
    }

fact noUnusedPlug {
    #(Plug) = #(ChargingArea.hasPlug)
    }

fact noUnusedPosition {
    #(Position) = #(User.currentPosition + Car.currentPosition)
    }

fact noUnusedBillingInfo {
    #(BillingInfo) = #(User.billingInfo)
    }

fact everyUserBelongsToManagementSystem {
    #(User) = #(ManagementSystem.registeredUser)
    }

fact everyCarBelongsToManagementSystem {
    #(Car) = #(ManagementSystem.registeredCar)
    }

fact everySafeAreaBelongsToManagementSystem {
    #(SafeArea) = #(ManagementSystem.safeArea)
```

```
    }

fact everyEmployeeBelongsToManagementSystem {
    #(Employee) = #(ManagementSystem.registeredEmployee)
    }

fact everyCredentialBelongsToAUser {
    #Credential = #(User.credential)
    }

fact threePassengersDiscountRides {
    all r: Ride | (r.state = COMPLETED and r.numOfTravellers >= 3 and
        r.endBatteryLevel < 50 and r.car.pluggedIn = none) implies
        r.discount = THREE_PEOPLE_DISCOUNT
    }

fact highBatteryDiscountRides {
    all r: Ride | (r.state = COMPLETED and r.endBatteryLevel >=50 and
        r.car.pluggedIn = none) implies r.discount = HIGH_BATTERY_DISCOUNT
    }

fact pluggedInDiscountRides {
    all r: Ride | r.car.pluggedIn != none implies r.discount = PLUGGED_CAR_DISCOUNT
    }

fact noFreeDiscountsApplied {
    no r: Ride | r.numOfTravellers < 3 and r.endBatteryLevel < 50 and
        r.car.pluggedIn = none and r.discount != none
    }

fact pluggedInCarsAreLeftInChargingStations {
    all r: Ride | r.state = COMPLETED and r.car.pluggedIn != none implies
        r.finalDistanceFromChargingArea=0
    }

fact lowBatteryAdditionalCharge {
    all r: Ride | r.state = COMPLETED and r.endBatteryLevel < 20 implies
        r.additionalCharge = LOW_BATTERY_ADDITIONAL_CHARGE
    }

fact highDistanceAdditionalCharge {
    all r: Ride | r.state = COMPLETED and r.finalDistanceFromChargingArea > 3 implies
        r.additionalCharge = HIGH_DISTANCE_ADDITIONAL_CHARGE
    }

fact noDiscountIfAdditionalCharge {
    all r: Ride | r.additionalCharge != none implies r.discount = none
    }

fact brokenCarsAreFixedByOneEmployee {
```

```
    all c: Car | (c.state = INTERVENTION_REQUIRED and c.pluggedIn = none and no r: Ride |
        r.state = ACTIVE and r.car = c) => one e: Employee | e.fix = c
    }

fact pluggedCarsAreNotFixedByEmployees {
    all c: Car | (c.pluggedIn != none => no e: Employee | e.fix = c)
    }

fact noNewCarsRequireIntervention {
    no c: Car | (no r: Ride | r.car = c) and c.state = INTERVENTION_REQUIRED
    }

fact noRandomFloatsShown {
    all f: Float | (some p: Position | p.latitude = f || p.longitude = f)
    }

fact noRandomStringaShown {
    all s: Stringa | (some u: User, c: Car | (u.licenseNo = s ||
    u.credential.username = s || u.credential.password = s || c.licensePlate = s))
    }

fact interventionRequiredCarHasLastCompletedRideWithSomethingWrong {
    all r1: Ride | (no r2: Ride | r2 != r1 and r2.car = r1.car and
        r2.beginDate > r1.endDate) and (r1.endBatteryLevel < 20 or
        all a: SafeArea | r1.car.currentPosition & a.point = none)
<=>
            r1.car.state = INTERVENTION_REQUIRED
    }

// === ASSERTIONS ===
assert noEmployeeFixesOKCar {
    no e: Employee | e.fix.state != INTERVENTION_REQUIRED
    }
check noEmployeeFixesOKCar

assert noUnreservedCarInReservation {
    all r: Reservation | r.selectedCar.state = RESERVED
    }
check noUnreservedCarInReservation

assert noCarReservedMoreThanOnce {
    no disjoint r1, r2: Reservation | r1.selectedCar = r2.selectedCar
    }
check noCarReservedMoreThanOnce

assert noUserWithMoreThanOneReservation {
    no disjoint r1, r2: Reservation | r1.madeBy = r2.madeBy
    }
check noUserWithMoreThanOneReservation
```

```
assert moneySavingRideHasDestination {
    no r: Ride | r.moneySaving = True and r.moneySavingDestination = none
    }
check moneySavingRideHasDestination

assert allRunningCarsHaveActiveRide {
    no c: Car | c.state = RUNNING and (no r: Ride | r.car = c and r.state = ACTIVE)
    }
check allRunningCarsHaveActiveRide

pred show() {
    some r: Ride | r.state = COMPLETED
    some r: Ride | r.state = ACTIVE
    some r: Ride | r.numOfTravellers >1
    some c: Car | c.state = INTERVENTION_REQUIRED
    some r: Ride | r.discount != none
    some r: Ride | r.discount != HIGH_BATTERY_DISCOUNT
    }

run show for 4 but 8 int
```

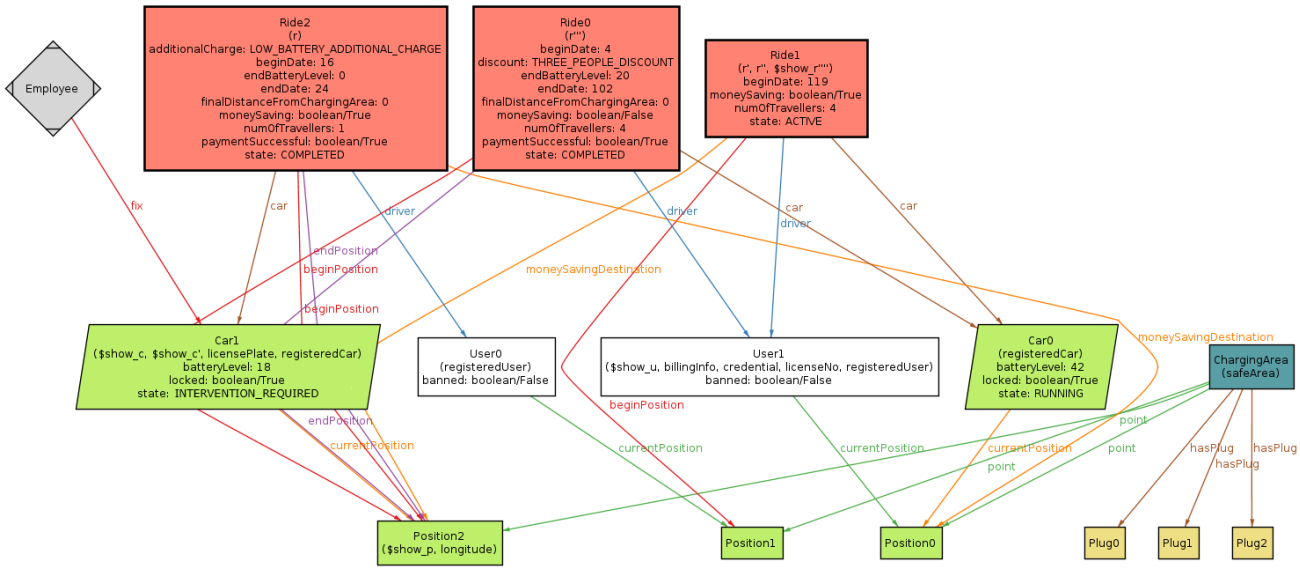

Figure 3.15: One possible solution of the Alloy logic model.

A Appendix A

A.1 Software and tools used

- LaTeX for typesetting the document.
- LyX as a document processor.
- GitHub for version control and teamwork.
- Pencil for mockups.
- Draw.io for UML diagrams.
- Alloy Analyzer to check the consistence of the model.

A.2 Hours of work

- Alessio Mongelluzzo: 30 hours
- Michele Ferri: 35 hours
- Mattia Maffioli: 30 hours

A.3 Version history

- 06/02/2017: RASD v. 1.1: General coherence update. Some things were incorrectly stated by the RASD document due to different choices made later on during the project (in particular during the Design phase). This update should fix the inconsistencies.