PowerEnJoy
Software Engineering II

Requirements Analysis and Specification Document

Giovanni Scotti, Marco Trabucchi

Document version: 3.1
February 27, 2017

Contents

Section 1

Introduction

1.1 Purpose

The Requirement Analysis and Specification Document for the *PowerEnJoy* digital management system is intended to describe the system itself, the functional and non-functional requirements, its components as well as its constraints and the relationship with the real world and the users by providing several use cases and scenarios. Furthermore part of the documentation makes use of Alloy, a language for describing structures and a tool for exploring them, and gives a formal specification of some features of the system to be.

This document establishes some baselines for the project planning, its estimation and evaluation and it may be legally binding; it is mainly addressed to developers and programmers, who have to implement the requirements, testers, who have to determine whether the requirements have been met, project managers, who control the development process and - last but not least - users, who validate the system goals.

1.2 Scope

The product is a digital management system to support a car-sharing service that exclusively employs electric cars.

The system consists of a back-end server application that manages rental requests remotely and three front-end applications:

- A web-based application to provide the final user with a friendly interface to take advantage of the services of *PowerEnJoy*;

- An application that runs on the existing on-board computers provided on each vehicle, used to interact with the car itself, unlock it and access the GPS/sat-nav service;
- A mobile application that allows the user to easily access the service anywhere he/she needs to.

The system is mainly intended for one type of user: drivers, who should be allowed to register and access the system via username and password, in order to make the renting and payment processes easier and quicker to carry out. Moreover, the system aids the users by locating nearby available vehicles and keeps track of the distance driven, all while notifying them about the amount of money they are being charged. A defined Safe Area stands as a limit for the service; properly equipped charging spots are signaled by the on-board computers.

The system aims to motivate drivers to maintain a virtuous behavior providing discounts when it detects signs of responsible and ecologic actions.

Lastly, the system is also in charge to inform - by marking the vehicles as out-of-service when necessary - the existing maintenance system about out-of-order vehicles so that the technicians can work out any kind of issue; the existing maintenance system will be able to do so by simply accessing part of the database of the system-to-be.

1.3 Goals

The goals of *PowerEnJoy* are the following:

1. Let the user register to the service and login via the provided credentials;
2. Let the user manage his/her own profile;
3. Let the driver find the location of nearby available cars and reserve a chosen car;
4. Improve the efficiency of the service by assuring that no car stays reserved for more than an hour if not actually in use;
5. Allow the user to easily access the cars by unlocking them once the driver is in proximity;
6. Automatically manage payments in order to make the service quicker and more dynamic to use;

7. Allow the user to start a ride, drive to his/her destination and finish the ride with the car he/she reserved;

8. Incentivize responsible behaviors, providing discounts for the worthiest users and additional charges for bad users.

1.4 Definitions, Acronyms and Abbreviations

Active reservation: a reservation of a vehicle that is not older than an hour, meaning that the user that performed it can still claim and use the reserved vehicle.

API: Application Programming Interface. A set of tools, protocols and libraries for building software and applications.

Authentication method: the to-be-defined method for users to authenticate to the on-board system in order to start their ride.

Back-end application: any computer program that remains in the background and offers application logic and communication interfaces to work with the front-end counterpart. It does not involve any graphical user interface, but it can provide a data access layer.

Charging spot: a particular parking spot equipped with a power grid and plugs to recharge the cars.

CAN bus: the Controller Area Network is a vehicle bus standard designed to allow microcontrollers and other devices to communicate with each other without a host computer.

DBMS: Database Management System.

Driver: See **User**.

Front-end application: any application the users interact with directly. It provides the so called presentation layer.

GPS: Global Positioning System.

Guest: any person who is not registered or logged in to the *PowerEnJoy* service.

m, km: meter, kilometer *Interational System of Units (SI)* units of measure.

Mobile broadband: it is the marketing term for wireless internet access delivered through mobile phone towers to any digital device using a portable modem.

OS: Operating System.

Out-of-service: the state of a vehicle that either broke down or is out of battery charge.

Parking spot: a generic place whitin the Safe Area where the driver can park the vehicles.

PC: Personal Computer.

RASD: Requirements Analysis and Specification Document.

Ride: the trip that involves the use of a *PowerEnJoy* electric car.

Safe Area: the predefined area where it is possible to start and end a ride; it marks the boundaries within which the user can park a vehicle and conclude the ride so that the system stops charging him/her.

System: The software system-to-be, in all of its entirety.

UML: Unified Modeling Language, the modeling language used in this document.

UMTS: Universal Mobile Telecommunications System.

User: Any person subscribed and logged in to the service who hence can rent a car using *PowerEnJoy*.

Vehicle: Any of the electric cars provided by *PowerEnJoy*.

W3C: World Wide Web Consortium. It is the main international standards organization for the World Wide Web, founded and led by Tim Berners-Lee.

1.5 References

This document follows the guidelines provided by ISO/IEC/IEEE 29148:2011 [1] and IEEE 830-1998 [2] respectively related to the requirements engineering for systems and software products and the recommended practice for software requirements specifications.

Moreover it is strictly based on the specifications concerning the RASD assignment [3] for the Software Engineering II project, part of the course held by professors Luca Mottola and Elisabetta Di Nitto at the Politecnico di Milano, A.Y. 2016/17.

1.6 Overview

This document consists of three sections:

Section 1: Introduction. A general introduction and overview of the system-to-be purpose, scope and goals, along with some important information about this document.

Section 2: Overall description. It describes the general factors that affects the product and its requirements. The section provides a background for those requirements which are defined in detail in Section 3 and makes them easier to figure out.

Section 3: Specific Requirements. All the software requirements are specified to a level of detail which is sufficient to let the designers satisfy them. Both functional and non-functional requirements are mentioned.

At the end of the document are an **Appendix** and a **Bibliography**, providing additional information about the sections listed above.

Section 2

Overall Description

2.1 Product Perspective

2.1.1 User interfaces

The users have several ways to access the system: a web application can be executed on any personal computer while a mobile application provides flexibility, portability and can be used literally everywhere. Despite the fact that the hardware interfaces running the application are rather different, a unified and common user interface is provided. It should be user friendly and very intuitive to allow everyone to easily use it without any specific knowledge.

Moreover the users have to interact with the on-board computer installed on each electric vehicle, therefore it should offer an interface as straightforward as the one implemented by the web and mobile applications.

2.1.2 Hardware interfaces

The web application can be executed on any general purpose computer that complies with the minimum system requirements specified in subsection 2.4.2.

The mobile application has to exchange data with the GPS module located on any recent smart-phone. Moreover it has to access the mobile broadband in order to communicate with the main system server.

An on-board computer is already set up in each electric car and it talks to the vehicle control unit through the CAN bus and to the system server via the mobile broadband. All sensors and hardware components needed to support the functionalities of the system are already installed on the vehicles.

2.1.3 Software interfaces

The web based application must support the main browsers such as IE, Google Chrome and Mozilla Firefox.

The mobile application has to be compatible with iOS and Android. The server side of the application, that is the system back-end, stores data in a relational DBMS and runs on any web server supporting Java.

The on-board computers run a customized Linux-based kernel and provide adequate APIs for the developers to communicate easily with the existing vehicle systems. This is to make communication between the system-to-be and the existing software easier.

The back-end has mainly to deal with the rental service and data management, essentially aimed towards keeping track of transactions and customers information, as well as the cars states; this last information is tracked in order to notify an existing maintenance system about out-of-service cars.

2.2 Product Functions

The system allows users to reserve available cars, to manage their requests and charge them for the rental.

The users can:

- Register to the service;
- Login with their personal account;
- Edit and manage personal information and delete their accounts;
- Locate nearby available-to-rent vehicles;
- Reserve one of said cars for a fixed amount of time;
- Unlock the reserved vehicle based on their mobile device GPS position;
- Alternatively, unlock the reserved car by inserting a vehicle-specific code into the application;
- Start the engine through a to-be-defined authentication method;
- Provide and later modify a favourite payment method for the system to manage actual due payments.

Figure 2.1: The comprehensive use-case diagram of all the functionalities provided by the system.
Note: all directed arrows from the system functions towards actor imply a notification.

2.3 User Characteristics

Target of the service are *drivers* that wish to rent a car.

The base assumption is that every registered user is in possess of a driving license which is valid in the country where he/she makes use of the service.

Moreover, the user must have the means of accessing the service from anywhere: this includes having the possibility to use a stable internet connection both from a mobile device and a stationary one such as a laptop or desktop PC.

Lastly, the user should provide the system a valid payment method upon the act of registration to the service.

2.4 Constraints

2.4.1 Regulatory policies

The system must be allowed by the user to collect, process and store personal data. Furthermore, the system is capable to delete all personal data upon user request and to keep track of each payment.

The user has the responsibility to use the system properly in order to comply with the local laws and policies.

2.4.2 Hardware limitations

The system should comply to these following minimum hardware requirements:

- Mobile application:
  - 3G UMTS connection at its maximum speed of 2 Mb/s
  - 50 MB of available space
  - 1 GB of RAM
  - GPS module
- Web application:
  - Internet connection at 7 Mb/s
  - 800x600 screen resolution

2.4.3 Parallel operation

The system must support parallel operations from different users and the DBMS relies heavily on concurrent transactions.

2.4.4 Reliability requirements

The system reliability, that is the probability to operate without a failure for a specific period of time, must be 99%.

2.4.5 Criticality of the application

Life-critical applications do not concern the system to be developed.

2.4.6 Safety and security consideration

The user must have a valid driving license in order to take advantage of the car sharing service. The license number is asked and stored by the system for security reasons. The locations and the travels performed by the users must be tracked but kept private.

2.5 Assumptions and Dependencies

For the rest of the RASD, the following assumptions are given for granted:

1. The GPS location of users and cars is always functioning and accurate, with an uncertainty of ± 1 m;
2. All users can access a reliable and stable internet connection;
3. The users' mobile devices feature a working GPS;
4. All users are always charged the correct amount after the ride;
5. The on-board computers always notify the correct charge to the driver during the ride;
6. All cars unlock properly upon insertion of a vehicle-specific code by the user who reserved them or in case he/she is detected to be in proximity;
7. The equipment of the cars always gives a correct reading of the number of passengers, driver included, for the current ride;

8. Whenever a car is marked as "out-of-service" and the system sends a notification, it is always refueled or fixed by the existing maintenance service before becoming available again;
9. If a user reserves and uses a car he/she is the one who drives it and is responsible for the associated trip;
10. The maintenance service technicians process each request in no more than 1 hour;
11. The maintenance system has the ability to mark "out-of-service" cars as "available" again after each maintenance intervention;
12. If a user is fined while driving a rented car, *PowerEnJoy* sends the fine at his/her address and he/she takes care of its payment;
13. Every type of vehicle damage is properly reported by the system;
14. Each car is provided with a unique, vehicle-specific code;
15. All users provide correct and valid data at the time of registration;
16. All cars always read properly their respective battery level.

2.6 Future Extensions

The system implementation will open the possibility of including new features. For example:

- The possibility of including more categories of vehicles, such as motorcycles and bycicles;
- The possibility of receiving active feedback from the users, mainly concerning car conditions;
- The possibility for everyone, registered and unregistered people, of notifying *PowerEnJoy* of incorrect or unsafe situations involving their vehicles;

2.7 World and Machine model interpretation

In this section, a description of the system-to-be is provided following the World and Machine model introduced by Jackson and Zave [4].

In their model, they indicate as the *Machine* the product-to-be. The *Machine* domain is the set of phenomena that the machine can control: data structures it can manipulate, algorithms it can run, devices it can control, inputs it can get from the world, and so on.

In contrast, the *World* domain is the real-life context into which the *Machine* will be introduced, and in particular, is that part of the world in which the *Machine*'s actions will be observed and evaluated.

The *World* and the *Machine* are connected, because the second must interact with the first in order to be useful. The connection is via *Shared Phenomena* – things that are observable both by the *Machine* and by the *World*. *Shared Phenomena* include events in the real world that the *Machine* can directly sense and actions in the real world that the *Machine* can directly cause.

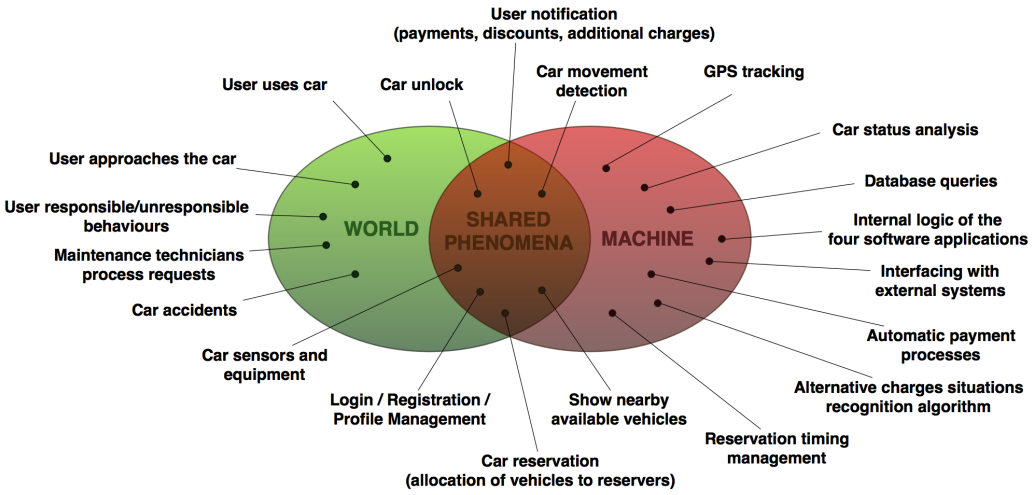


Figure 2.2: World and Machine model for the main functionalities to be provided by the system.

Section 3

Specific Requirements

3.1 External Interface Requirements

3.1.1 User interfaces

The user interfaces must be strongly user-friendly, in order to provide an easy and intuitive way to access the system. To achieve this, the following constraints have to be satisfied by the web and mobile applications:

- The first page must always ask the user to login or register to the service;
- After the login, the system must load a user home page;
- (Web) A toolbar must be accessible from every screen except the login/register page;
- (Mobile) A toolbar must be accessible from the home page;
- The toolbar must offer a quick and simple overview of the main system functionalities (car reservation, show personal information area, log out...);
- The "Car reservation" page must show a map with a search box in order to find all nearby available cars with respect to the specified address OR the information about the active reservation;
- The interface must offer the possibility to choose between a set of different languages;
- The user interface must dynamically adapt to the screen size;

- In order to enhance the user experience, the web application and the mobile one must use the same graphical elements;
- The user interface shall show the profile picture uploaded by the user, if any.

With respect to the on-board computer application, the restrictions are the following:

- After the car has been unlocked, the display of the on-board computer must turn on and show a screen that allows the user to perform the provided authentication method in order to start the ride;
- Once the ride starts, the screen must clearly show the current charges;
- Similarly, during the ride, the screen must clearly show the nearby charging spots and visually signal the Safe Area boundaries;
- When the car is locked, the screen must always be off.

Some mockups that give an overall idea of the interface structure of the applications are provided in the next section (3.2). In addition to that, several application-dependent constraints are provided:

- Web application

  All the web pages must abide by the W3C standards to ensure a long-term growth of the application and take the full advantage of HTML5 markup language and CSS.

- Mobile application

  Both the Android and iOS versions must follow the design guidelines provided by the respective platform manufacturers.

- Server application

  The server application must be easily configurable via a XML file and use the APIs provided by the payment handler to interface with its information system.

- On-board computers application

  The application mounted on on-board computers must use the APIs provided by the vehicle manufacturer to interface with the cars.

3.1.2 Hardware interfaces

The most relevant aspect to highlight, in addition to section 2.1.2, is the ability of the system-to-be to communicate with the car systems via the embedded computer located in the car itself that takes the advantage of the on-board computer application to be developed. The on-board computer can communicate with the mobile broadband and provides a GPS module exploitable by the car-side of the application.

3.1.3 Software interfaces

The server side of the application, that represents the back-end of the system-to-be, requires the following software products:

- Java EE 7 - `http://www.oracle.com/technetwork/java/javaee/overview/index.html`
- MySQL 5.7 - `http://dev.mysql.com/`

The embedded on-board computer located in each car provides a Linux OS that is capable to run C/C++ programs. Moreover the car-side of the application can use specific software libraries provided by the manufacturer to access the car systems the computer is connected with.

3.1.4 Communication interfaces

The clients communicate with the server via HTTPS protocol by using TCP and default port 443. Moreover a safe and stable connection is also established between the server and the payment handler whenever the user performs a payment. The existing maintenance system accesses the database of the system-to-be with an equally safe and stable connection.

3.2 Functional Requirements

3.2.1 Register

Purpose

Anyone who wants to subscribe to the *PowerEnJoy* service can carry out the registration process through the mobile application or the web one. First of all the guest has to fill a registration form with his/her personal information and accept the Terms and Conditions of use otherwise the procedure will be aborted.

As soon as the guest submits the data, the system checks all the information and sends a confirmation e-mail to the specified e-mail address including the password to access the service and the information about the provided authentication method required to start the car engine.

Scenario 1

Marta would like to register to the *PowerEnJoy* rental service from her computer. She opens the browser and loads the registration page by clicking on the *"sign up"* button located in the *PowerEnJoy* login page. She fills in the registration form and agrees the service Terms and Conditions. Then Marta submits everything to the system, that checks the personal data. Authentication information is sent to Marta in a confirmation e-mail notifying the successful registration.

Scenario 2

Yuri cannot afford a car and therefore decides to take advantage of the *PowerEnJoy* service. He accesses the login page from his mobile phone and taps the *"sign up"* button since he is not a registered user. Then he fills in the form and accepts the Terms and Conditions of use.

Unfortunately he forgets to fill in his own tax code and the system does not allow him to carry out a successful registration notifying the forgetfulness. Afterwards Yuri completes the procedure by filling the whole form. The system communicates the successful registration and sends him a confirmation e-mail containing authentication information.

Use-case

The use-case of the registration procedure is analyzed in Table 3.1.
A more detailed use-case diagram about the registration procedure is shown in Figure 3.1.

Activity diagram

The activity diagram of the registration procedure is shown in Figure 3.2.

Functional requirements

1. The system must not accept an already registered e-mail address for the registration process;

2. The system must request for the following mandatory personal information:
    - e-mail address
    - name
    - surname
    - ID card number
    - tax code
    - license ID number
    - address
    - phone number
    - favourite payment method
3. There must not be another subscribed user with the same identity card number, tax code or license ID;
4. Until the guest does not accept the Terms and Conditions of use, the registration process must not be concluded by the system;
5. The system must send an e-mail containing authentication information to the e-mail address specified by the guest when he/she clicks on *"Submit"* button and all the required information has been entered;
6. The system allows the guest to exit the registration process at any time;

Figure 3.1: Detailed use-case diagram of the registration process.
Note: the authentication information includes both password and the detailed explanation of the provided authentication method used to unlock cars

Actor	Guest
Goal	Goal 1
Input Condition	A person wants to subscribe to the service.
Event Flow	1. The guest opens the *PowerEnJoy* home page or the mobile application and clicks or taps on the "Sign up" button; 2. The registration form is loaded and the guest fills in all the fields with his/her personal information; 3. The guest agrees the service Terms and Conditions by checking the corresponding box; 4. The guest clicks on *"Submit"* button; 5. The system saves the data in its database and sends a confirmation e-mail containing the authentication information to the newly registered user.
Output Condition	The system tells the user he/she is successfully subscribed and lets him/her to log in with the provided credentials.
Exception	When some exceptions occur, the system reloads the registration form and goes back to step 2 of Event Flow notifying the guest about the error. This happens when the following requirements are not satisfied: 1, 2, 3. The registration process is aborted when the guest decides not to carry it out completely or when requirement 4 generates an exception.

Table 3.1: Register use-case

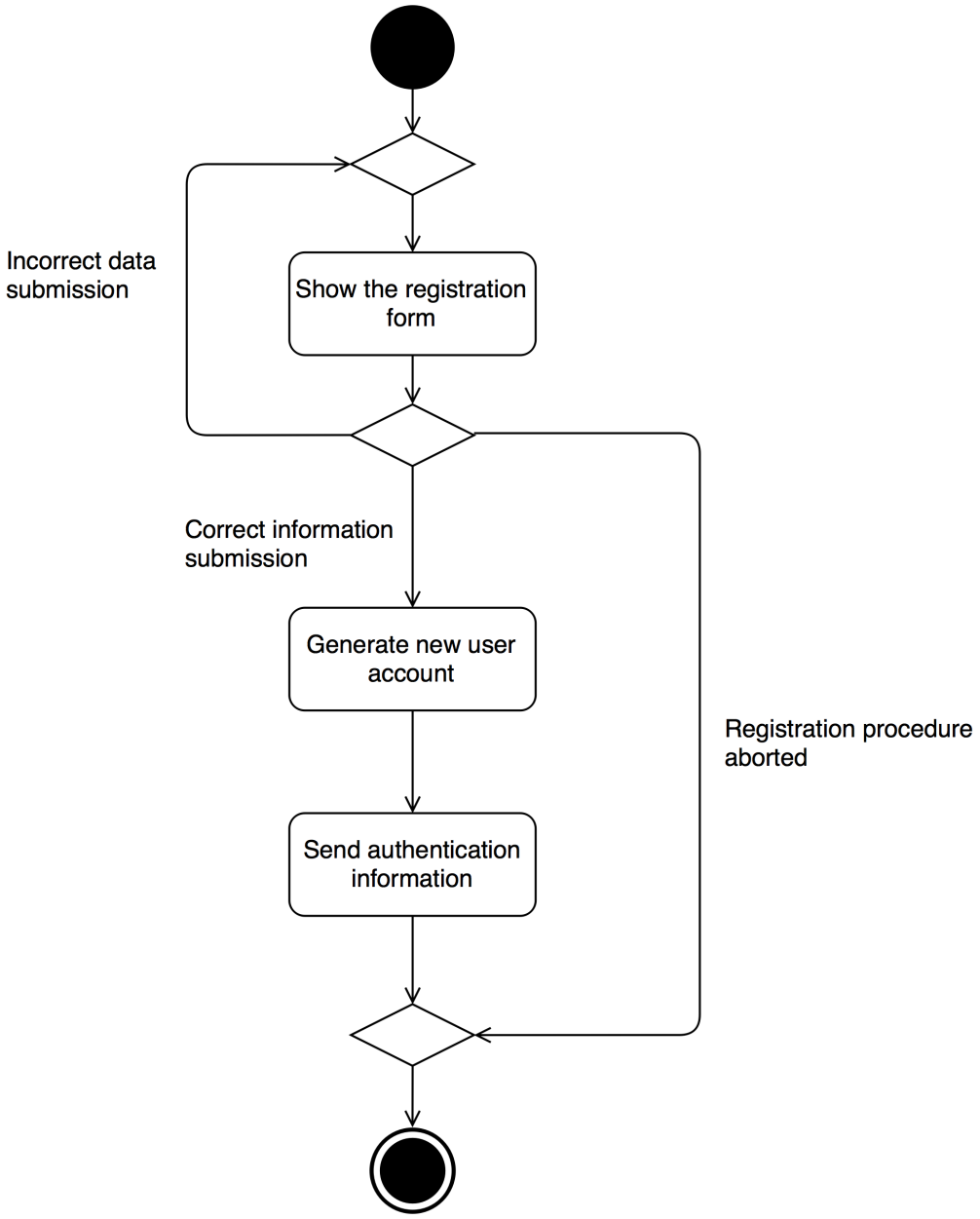


Figure 3.2: Activity diagram of the registration process from the system point of view.

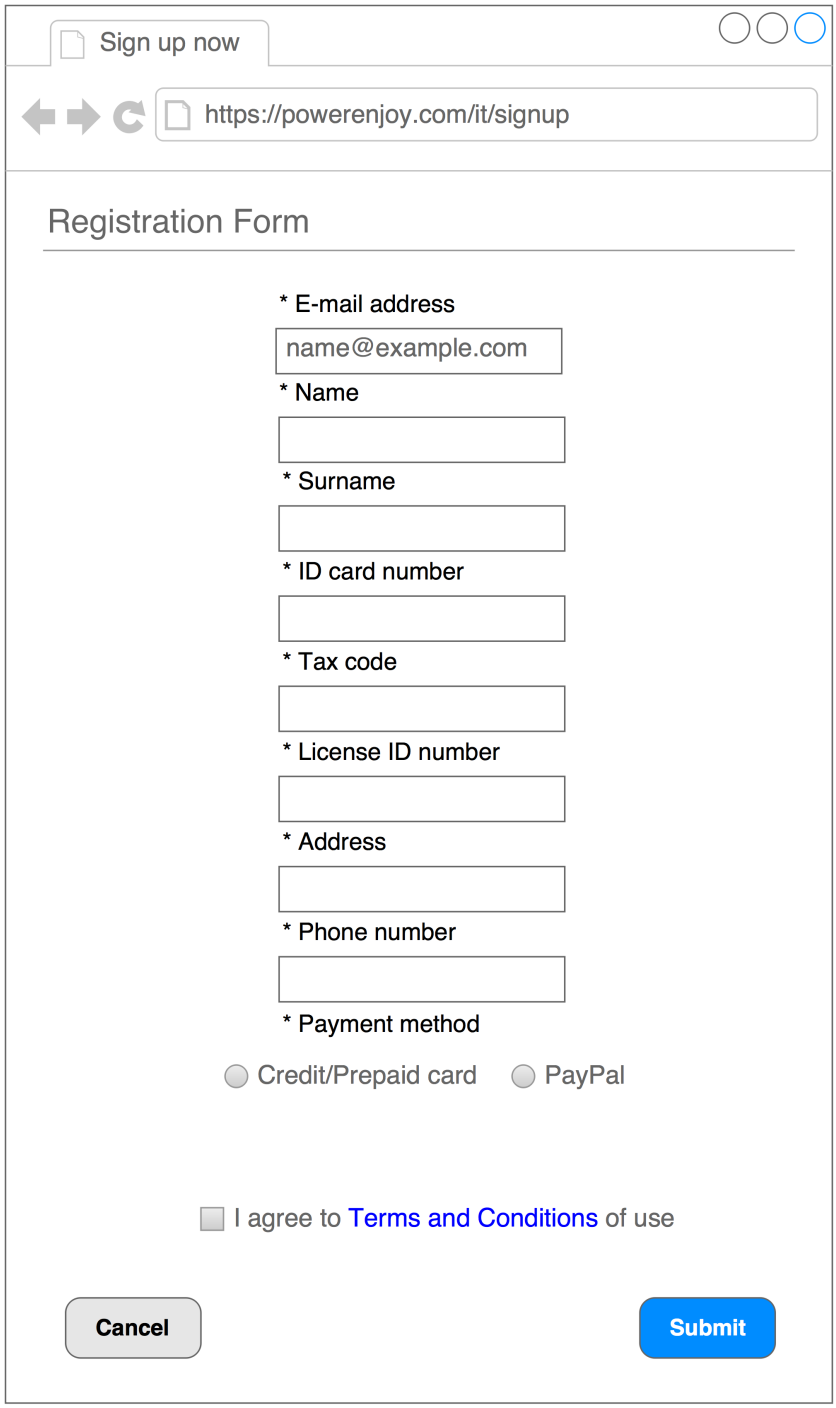


Figure 3.3: Mockup of the sign up webpage.
Note: the *"Payment Method"* field contains an example of possible choices that will be defined in the implementation phase.

3.2.2 Login

Purpose

The main purpose of the login feature is to grant the access to the *PowerEnJoy* service to any registered user. The system requires a valid e-mail address and password to login without generating any error.

In addition to that, the login screen offers the chance to recover a forgotten password. The user clicks on *"Forgot password?"*, a new one is sent to his e-mail address immediately and he/she can carry out the login procedure with the new password provided by the system.

Scenario 1

Mike would like to rent a car for having a ride in a beautiful sunny day. In order to do that he needs to access the system by means of his credentials. He opens the *PowerEnJoy* home page and enters his e-mail address and his password. Then Mike clicks on *"login"* and, due to the fact that everything is correct, he obtains the access as a logged user.

Scenario 2

Bill wants to take the advantage of the *PowerEnJoy* service. He opens the home page and he is asked for his e-mail address and password. He is aware of his e-mail address, but he does not remember the password. After some time spent trying to recall his personal password, Bill inputs his e-mail address and decides to click on *"Forgot password?"*. Right away the system sends a new password to the specified e-mail address. Bill can now enter the system using the new password.

Use-case

The login use-case is analyzed in Table 3.2.

Activity diagram

The login activity diagram is shown in Figure 3.4.

Sequence diagram

The sequence diagram of the login procedure is shown in Figure 3.5.

Functional requirements

1. The user must be already registered to the system in order to perform a successful login;
2. The user must be aware of his e-mail address and password to successfully obtain the system access;
3. The password provided by the user must correspond to the specified e-mail address;
4. Wrong credentials prevent the user to access the system;
5. The system sends a new password to the specified e-mail address if and only if the specified e-mail address is valid, registered to the system and the user clicks on *"Forgot password?"*;
6. After requesting a new password, the system must allow the user login with the new provided password;
7. After three times the entered password is wrong, the system prevents a new attempt for the following 30 seconds;
8. The system must not allow a registered user with a suspended account due to pending payments to log in.

Actor	Guest
Goal	Goal 1
Input Condition	The guest is already registered to the system and wants to login.
Event Flow	1. The guest opens the *PowerEnJoy* home page or the mobile application and the system shows the login page; 2. The guest enters his/her e-mail address and password; 3. The guest clicks on *"login"* button.
Output Condition	The system lets the guest log in and loads his/her personal home page.
Exception	If the e-mail address provided by the guest has never been registered to the system or if the password does not correspond to the entered e-mail, the system notifies the guest with an error message. If the guest enters wrong credentials three times in a row, the system will prevent any attempt for the following 30 seconds informing him/her.

Table 3.2: Login use-case

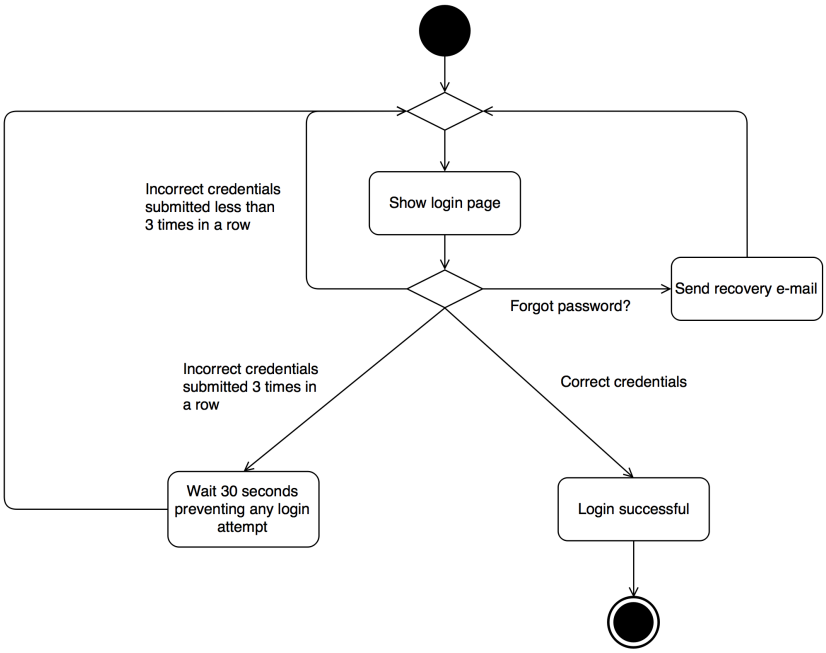


Figure 3.4: Activity diagram of the login process from the system point of view.

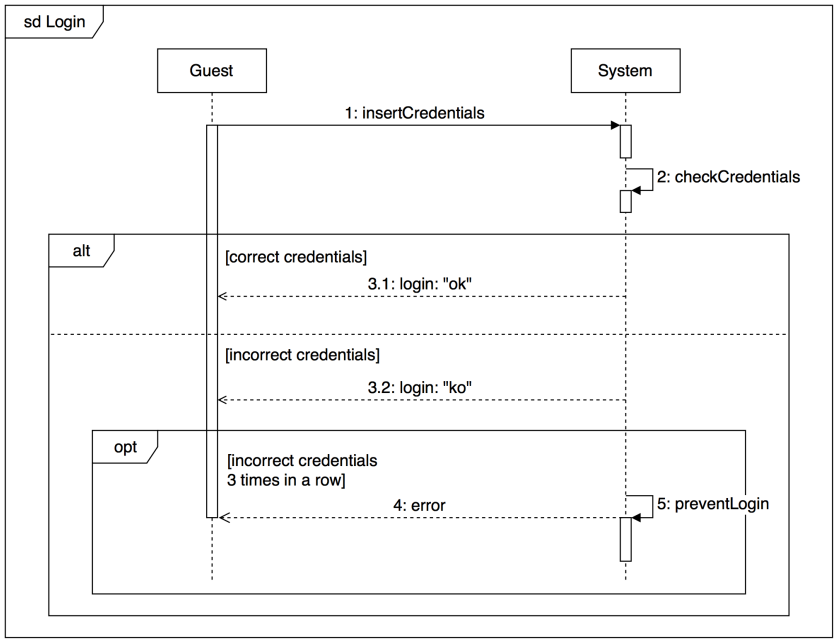


Figure 3.5: Sequence diagram of the login process.

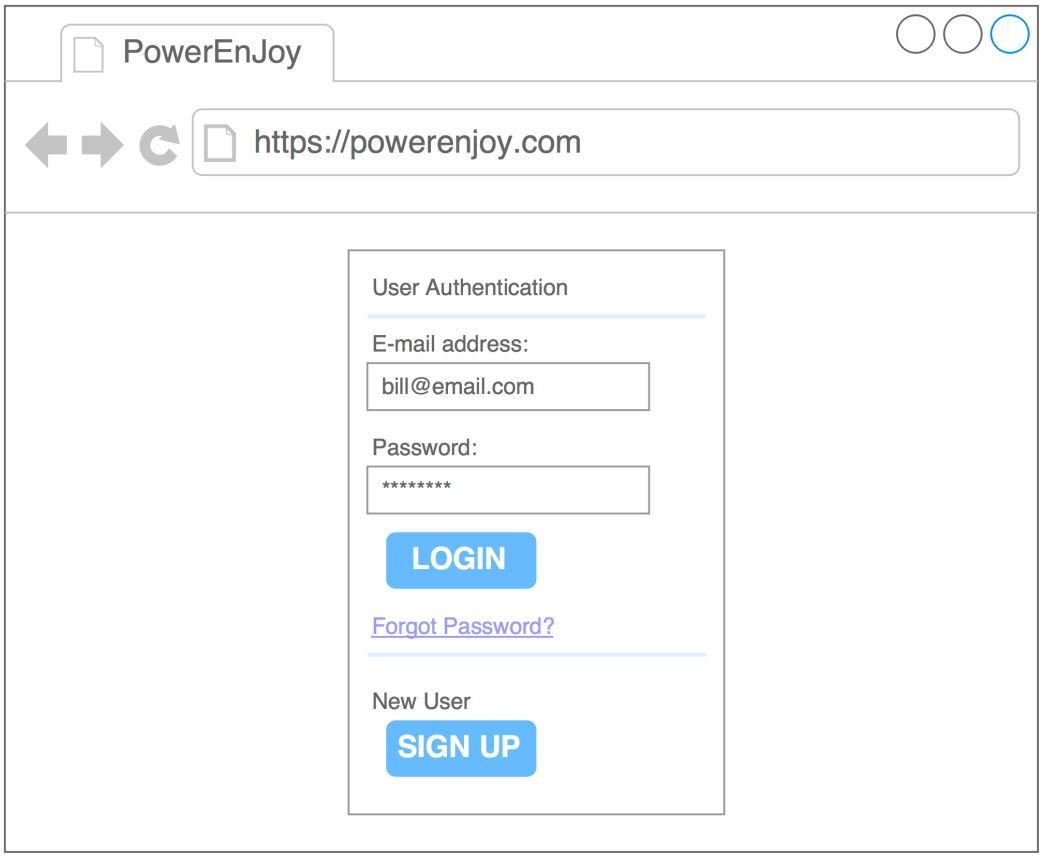


Figure 3.6: Mockup of the login webpage.

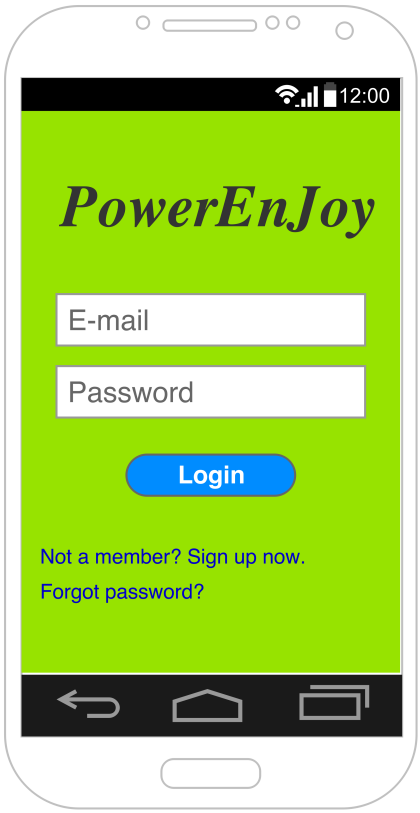


Figure 3.7: Mockup of the mobile version of the login screen.

3.2.3 Manage Profile

Purpose

Both the mobile application and the web one provide the chance to any user to view and update the personal profile information. The system is able to provide also information about the reservations and rides history as well as the respective payments.

Moreover the user is allowed to delete his/her account if the service is no longer needed.

Scenario 1

Anna has already carried out the registration process, but she does not like the password the system provided her because it is hard to remember. Therefore she decides to log in from her mobile phone and access the personal information area through the designated toolbar.

She taps *"Edit Profile"* and the system allows her to edit profile information. Anna enters the old password in the designated field as well as the new one, that must be entered twice to avoid unwanted errors. Then Anna selects *"Confirm"* and the system notifies her about the successful update.

Scenario 2

Flavio has recently renewed his identity card. In order to take the advantage of the *PowerEnJoy* service, he must update his profile information. Flavio opens the browser, reaches the *PowerEnJoy* home page and authenticates himself carrying out the login procedure.

Then he access his personal information area and clicks *"Edit Profile"*. He fills in the gap corresponding to the ID card number and, as soon as Flavio is ready, he clicks on the *"Confirm"* button and the system confirms the ID card number has been successfully updated.

Scenario 3

Emma wants to delete her *PowerEnJoy* account because she does not need the service anymore. She logs in from her mobile phone and enters her personal information area. She selects *"Edit Profile"*, taps *"Delete Account"* and confirms her choice twice. The system removes immediately all Emma's information from the database. From now on she needs a new account in order to access the system again.

Use-case

The overall profile management use-case diagram is shown in Figure 3.8.
The use-case of view profile procedure is analyzed in Table 3.3.
The use-case of modify profile procedure is analyzed in Table 3.4.
The use-case of delete profile procedure is analyzed in Table 3.5.

Functional requirements

1. The system allows the user to view his/her profile. When the user enters his/her personal information area, the system must display:
    - all the user's personal information that has been entered during the registration process;
    - the rides and reservations history;
    - the payments history.

2. The system allows user to modify his/her personal information as needed:
    - the system allows to change any information except e-mail address and authentication method related ones when the user selects *"Edit Profile"*;
    - the system asks the user to enter the old password and two times the new one whenever he/she needs to change it;
    - the system allows the user to change the password if and only if the old one has been entered correctly;
    - the system must not allow the user to change the password if the new one has not been entered twice;
    - the system makes the change permanent as soon as the user clicks the *"Confirm"* button;
    - the system allows the user to abort the profile editing at any time.
3. The system allows the user to delete his/her account:
    - the system asks for a confirmation twice whenever the user wants to delete his/her account;
    - the system completely destroys the user's personal information and removes his/her account from the database.

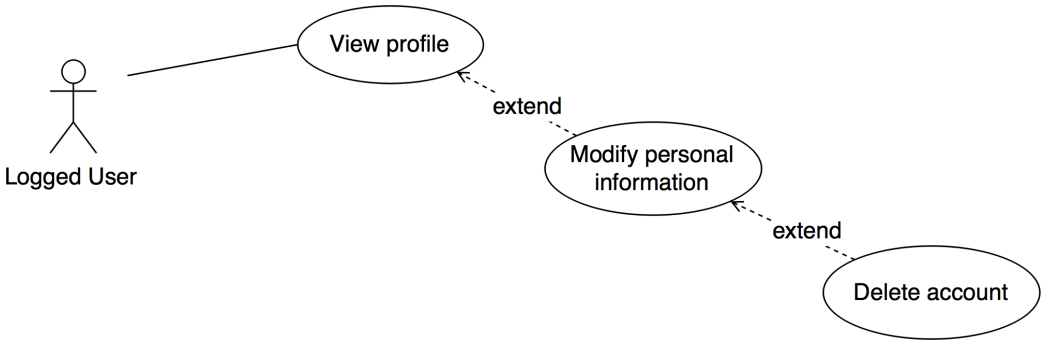


Figure 3.8: Profile management use-case diagram.

Actor	User
Goal	Goal 2
Input Condition	The authenticated user wants his/her profile information to be displayed.
Event Flow	1. The user enters his/her personal information area. 2. The system loads the page.
Output Condition	The system lets the user check his/her profile, the reservations and rides history as well as the completed payments.
Exception	None.

Table 3.3: Profile visualization use-case.

Actor	User
Goal	Goal 2
Input Condition	The authenticated user wants to edit his/her profile information.
Event Flow	1. The user enters his/her personal information area; 2. The user selects *"Edit Profile"*; 3. The system loads the *"Edit Profile"* page; 4. The user enters the new information; 5. The user confirms the desired changes.
Output Condition	The system updates the user's personal information.
Exception	Every time the user does not satisfy one of the requirements listed as number 2, the system simply ignores the modification and reloads the *"Edit Profile"* page.

Table 3.4: Modify profile use-case.

Actor	User
Goal	Goal 2
Input Condition	The authenticated user wants to delete completely his/her profile.
Event Flow	1. The user enters his/her personal information area; 2. The user selects *"Edit Profile"*; 3. The system loads the *"Edit Profile"* page; 4. The user selects *"Delete Account"*; 5. The system asks to confirm and the user confirms; 6. The system asks again to confirm and the user agrees;
Output Condition	The system deletes the user's account.
Exception	If the user decides not to confirm twice, the deletion process is aborted and the *"Edit Profile"* page is reloaded.

Table 3.5: Delete profile use-case.

3.2.4 Reserve Car

Purpose

The reserve car feature is a key service of *PowerEnJoy*. Its main purpose is that of allowing users to choose a nearby car from the ones marked as available on the map, hence signaling their intention of reserving it for an hour. The service can be accessed from the home page of both the web and mobile applications, clicking on the corresponding icon on the toolbar.

The reservation screen shows the user only available cars, based on the information provided by the server. The user can either search for an available car using the mobile application or the web application, indicating an address around which he wishes to search for a vehicle. If the user is logged in through the mobile application, he/she can also choose to use his/her GPS

position - as provided by the mobile device - instead of a specific address.

Only one car should be rented at a time, so the system will prevent the user from trying to reserve several cars simultaneously.

Scenario 1

Alice is logged in to the *PowerEnJoy* service with her laptop, because she wants to find a car to go meet a friend downtown. To do so, she clicks on the reservation icon and opens the city map. She inputs her address, to indicate that she wants a car not far from her house. The system proceeds to mark the vehicles within the desired location. Alice finds a car that she thinks is reasonably close and clicks on it. The system prompts her asking: *"Do you really want to reserve this car?"*. She clicks on the *"Confirm"* button, hence forwarding the request of the specific car to the system.

Scenario 2

Bob is in town and has just finished shopping. He wants to go back home, but does not want to catch a crowded subway train during the rush hour. For this reason he has logged in to the *PowerEnJoy* application and taps on the reservation icon. The system opens a map that shows the GPS position - provided by Bob's smartphone - and several icons marking nearby available cars. He chooses a vehicle just around the corner and properly confirms his intention of reserving the car for some time; by doing so, his request is forwarded to the server. A few minutes later, he meets a friend that offers him a ride, and decides to accept. So, he logs in to his account again and opens the reservation function. He taps on the *"Release Reservation"* button and the system processes his request by marking the car as available again.

Scenario 3

Carla is in need of a car after work, and decides to use a *PowerEnJoy* one. She is looking at the reservation screen and decides to reserve a vehicle by selecting it. Since she is not sure that she will be able to reach the vehicle easily, she tries to reserve a second one after a few minutes. At that point, the system shows a visual recap of her current reservation. Carla tries to access the city map, but the system prevents her to do so, unless she cancels her current reservation. So she renounces to the second reservation and the system does not change anything about the active reservation status.

Use-case

A more detailed use-case diagram is provided in order to model the possibility of releasing an existing reservation. The detail of the use-case diagram is shown in Figure 3.9.
The use-case for the standard procedure needed to request a car is analyzed in Table 3.6.
The use-case for the procedure needed to release an existing reservation is analyzed in Table 3.7.

Sequence diagram

The diagram for the standard behavior of the system in case of a car reservation is shown in Figure 3.10.

Functional Requirements

1. The system must allow the user to find nearby cars based on a specified address;
2. The system must allow the user to find nearby cars based on their GPS-detected position;
3. The system must be able to connect to the GPS of the users' mobile devices;
4. The system must consider as valid positions for a car search all GPS positions within the Safe Area;
5. The system must consider as valid positions for a car search all addresses within the Safe Area;
6. If a user inputs a non-valid address, the system must notify him/her with an error message saying that he/she is too far from the Safe Area to locate a car nearby;
7. The system must only show as "available" cars within the Safe Area;
8. The system must provide the exact position of all nearby vehicles in a radius of 2 km for every valid position;
9. The system must mark as "available" only cars that are not already "reserved", "in-use" or "out-of-service";
10. The system must check that only "available" vehicles can be rented;

11. The system must mark as "reserved" any rented vehicle;
12. An "available" car can be rented by only one registered user at a time;
13. If a user tries to reserve a car, but that car has been reserved by another user during the visualization of the available vehicles, the system must notify it with an error message;
14. The system must allow car rental only to users that do not have any active reservation yet;
15. If a user has an active reservation, the system must show as first screen - upon the selection of the reservation function - the recap of said active reservation, including time of expiration and car location;
16. If a user has an active reservation, the system must allow him/her to renounce to it and ask him/her to confirm the choice;
17. The system must always notify the user of a successful reservation/recess from a reservation, immediately after the action is saved in memory by the server;
18. The system must allow the user to visualize the "available" cars without renting any, by canceling the current reservation procedure.

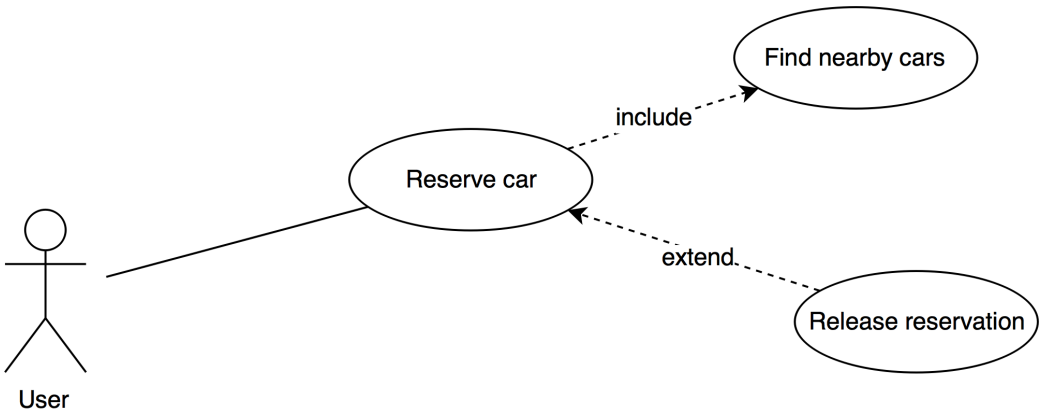


Figure 3.9: Extension of the use case diagram that includes the "Release reservation" feature

Actor	User
Goal	Goal 3
Input Condition	The user has no active reservations
Event Flow	1. In the main page for the *PowerEnJoy* website or mobile application, the user clicks or taps on the *"Car Reservation"* button; 2. The user inputs the desired location: specific address or current GPS location; 3. The system loads a map of the Safe Area, marking every available vehicle near the desired location; 4. The user clicks or taps on the car he/she wants to rent; 5. The user clicks or taps on the *"Confirm"* button when asked if he actually wants to rent the car; 6. The system saves the reservation in its database and logs it; 7. The system confirms the reservation to the user, reminding him/her of the fact that the reservation will expire in an hour of time with a small fee if the car is not actually used.
Output Condition	The system notifies the user that the reservation was successful and marks the chosen car as reserved.
Exception	If the user tries to reserve more than one car, the system notifies him/her that this action is forbidden. If the user changes his/her mind and decides not to reserve a car, the event flow stops at the point in which the map was shown. No car will be reserved in this eventuality.

Table 3.6: Reserve car use-case

Actor	User
Goal	Goal 3
Input Condition	The user has an active reservation not older than an hour
Event Flow	1. In the main page for the *PowerEnJoy* website or mobile application, the user clicks or taps on the *"Reserve car"* button; 2. Since the user already reserved a car, he is shown the location and time of his current reservation; 3. The user clicks or taps on the *"Release reservation"* button; 4. The system asks confirmation from the user; 5. The user confirms his/her choice of receding from his/her existing reservation; 6. The system modifies the state of the formerly reserved car marking it as available again and notifies the user of the fact that his/her reservation was successfully cancelled.
Output Condition	The system notifies the user that the reservation was successfully cancelled and marks the chosen car as available again. The user can reserve another car, since he/she has no more active reservations
Exception	If the user changes his/her mind and wants to keep the current reservation, the event flow stops at the point in which the current reservation was shown. The reservation will be kept unchanged in this eventuality.

Table 3.7: Reserve car use-case for the option of releasing a reservation

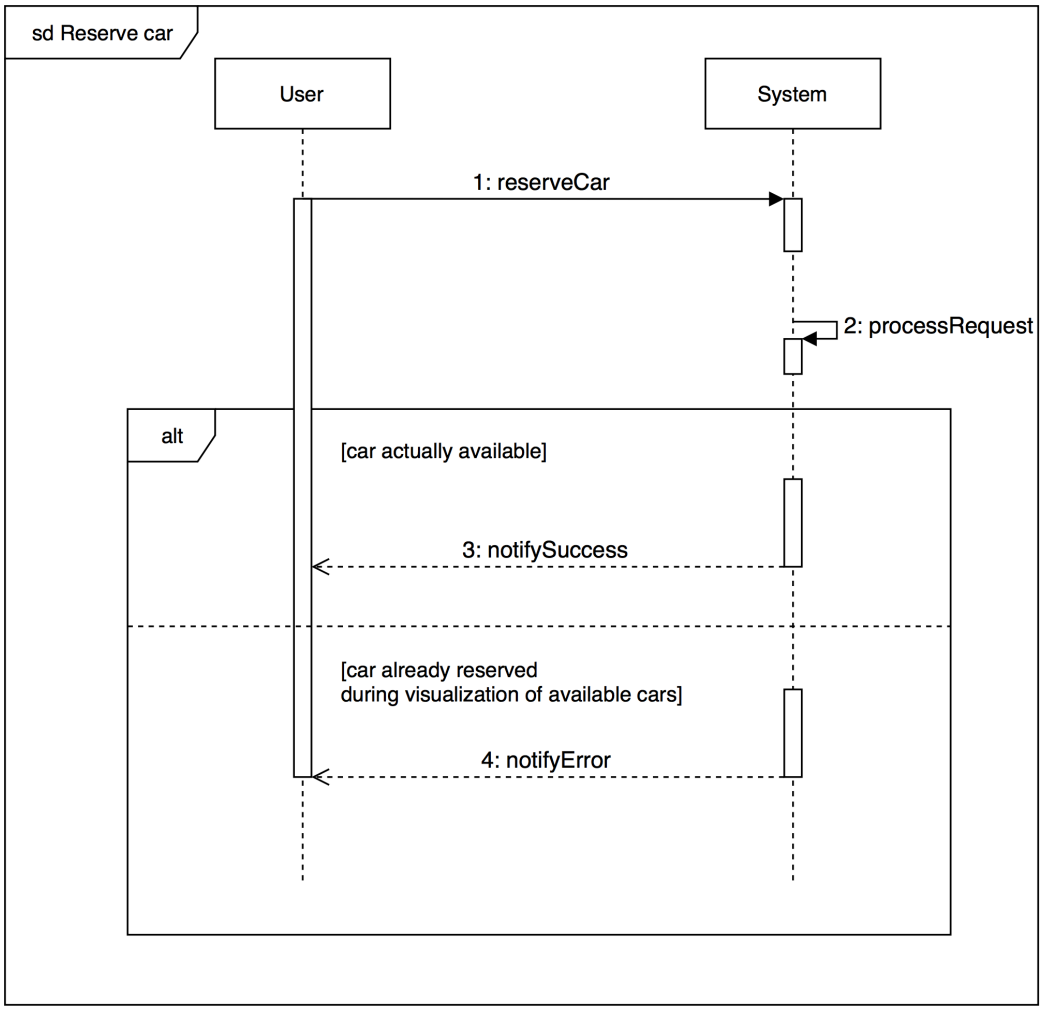


Figure 3.10: Sequence diagram that describes the event flow of a standard reservation request

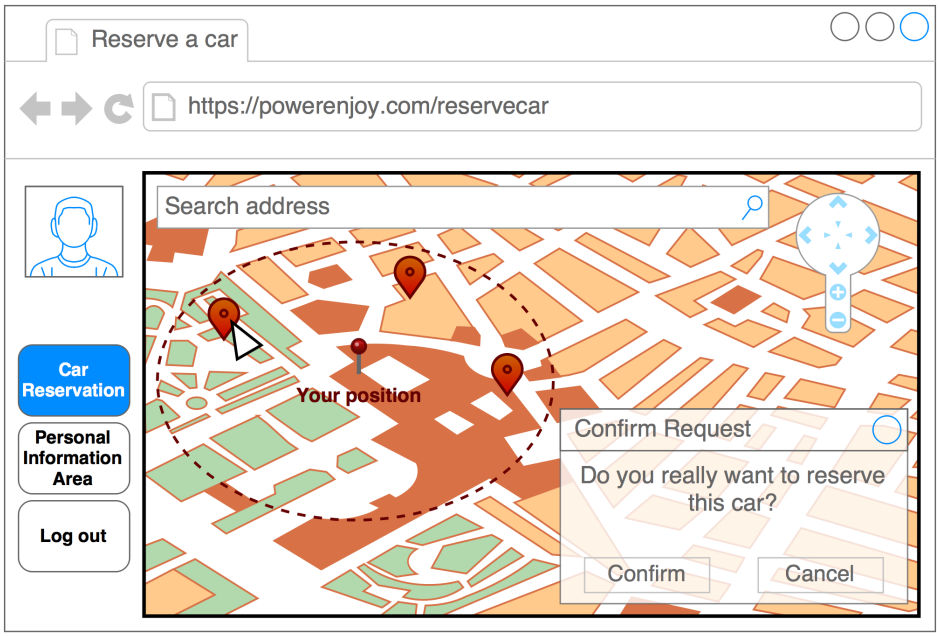


Figure 3.11: Mockup of the reservation web page

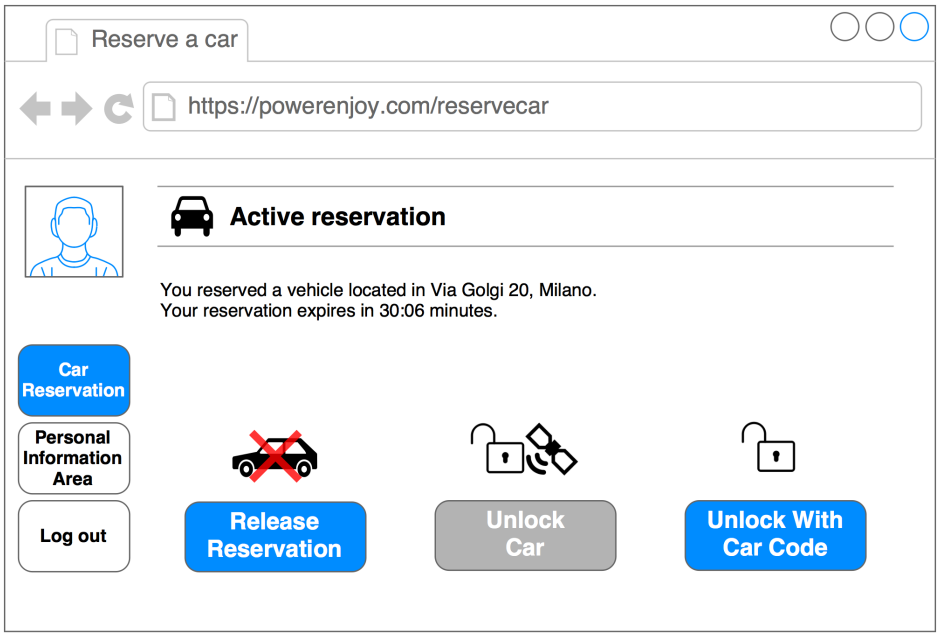


Figure 3.12: Mockup of the reservation web page with a pending active reservation.

3.2.5 Use Car

Purpose

The use car feature is the key feature of the whole system. For the sake of completeness, this requirement is described by detailing each of its sub-parts, as seen in Figure 2.1.

The detailed sub-parts are described in the subsections immediately after this one, namely: section 3.2.6, section 3.2.7, 3.2.8 and 3.2.10. As a consequence, the scenarios provided in this subsection are general and must be considered in conjunction with the more detailed one to have a full overview of the use-case.

Its purpose is that of allowing users to open, get on and drive the actual vehicle that they rented via the dedicated service. This feature requires the interaction of the user with both his/her mobile device and the on-board computer application.

The system must unlock a car upon the performance of the authentication method from the user that reserved it. The on-board computer application must keep track of every important detail of user rides, such as the current charges, the nearby free charging spots and the boundaries of the Safe Area, all while notifying correspondingly the driver.

The on-board computer application will detect the beginning and end of the ride, counting the charges accordingly. Also, the system will be able to detect whether the car is inside the Safe Area or not: if this is not the case, the user will be notified and the charges will keep increasing.

This feature triggers the notifications sent to the existing maintenance system, as it includes the detection of out-of-service vehicles.

Scenario 1

Vincent reserved a *PowerEnJoy* car to reach a far destination within his city. He reaches the vehicle and unlocks the car with his smartphone's GPS. Once aboard, he performs the authentication procedure and turns the car on, then starts his ride. Arrived at his destination, Vincent pulls over and checks that he did not park outside the Safe Area: since he did not, he turns the engine on and gets off the car. The system locks the car and applies all charges, initiating the automatic payment process.

Scenario 2

Wanda decided to rent a shared car instead of taking the subway train home. She reserved one on *PowerEnJoy* and is now walking towards it. She unlocks

the car properly and gets on it; after proper authentication, she starts driving. Her house is a the opposite border of the Safe Area, so she drives for a long time before arriving to her destination. Once she stops the engine, the system recognizes that the battery level is below 20%: it immediately proceeds to check the vicinity for charging stations. The system does not find any nearby so, after 5 minutes, it marks the car as "out-of-service", and Wanda is charged the correct amount of money for the ride.

Use-case

A general use-case analysis is provided in Table 3.8. The subparts are detailed in the following subsections as specified in "Purpose" above.
A statechart of the possible states of a car is shown in Figure 3.13.

Functional Requirements

The requirements listed below do not include the ones covered within the sub-parts mentioned above. To have a detailed overview of all the requirements related to the sub-functionalities, see the subsections mentioned in "Purpose".

1. The system must keep track of the position of all vehicles;
2. The system must be able to detect car accidents, damages and faults;
3. The system must mark as "out-of-service" every vehicle that broke down;
4. The system must mark as "out-of-service" every vehicle that ran out of battery charge;
5. The system must mark as "out-of-service" the vehicles that were parked with more than 80% of the battery empty;
6. If a car is plugged into a power grid and has a battery level greater than 20%, the system must mark it as "available" again;
7. The system must mark as "available" the vehicles that were parked with more than 20% of the battery still full;
8. As soon as a vehicle is marked as "out-of-service", a notification is sent over to the maintenance system.
9. If more than 15 minutes pass between a car being unlocked and user authentication, the system must mark the vehicle as "out-of-service" and prevent it from being used;

Actor	User
Goal	Goal 7
Input Condition	The user reserved a car and reaches its location.
Event Flow	1. The user unlocks the car (see detail in section 3.2.6); 2. The user authenticates and ingnites the engine, starting his/her ride (see detail in section 3.2.7); 3. The user rides the rented car to his/her destination; 4. The user ends his/her ride (see detail in section 3.2.8); 5. The system applies all charges, including additional ones and eventual discounts (see detail in section 3.2.10);
Output Condition	The car is marked as "available" again; after the end of the ride and a 5 minute period used to perform eventual additional checks, the user ends the use of the service.
Exception	If the ride ends and the battery is more than 80% empty, the vehicle is marked as "out-of-order" instead of "available". If the car is involved in an accident or breaks down, the ride ends and the vehicle is marked as "out-of order" immediately. The user is charged only up to the amount corresponding to the instant before the accident. Note that accidents include the eventuality of the car running out of battery charge during the ride. If more than 15 minutes pass between a car being unlocked and it being turned on, the vehicle is marked as "out-of-service"; the maintenance team is in charge of checking the car situation.

Table 3.8: Use car general use-case

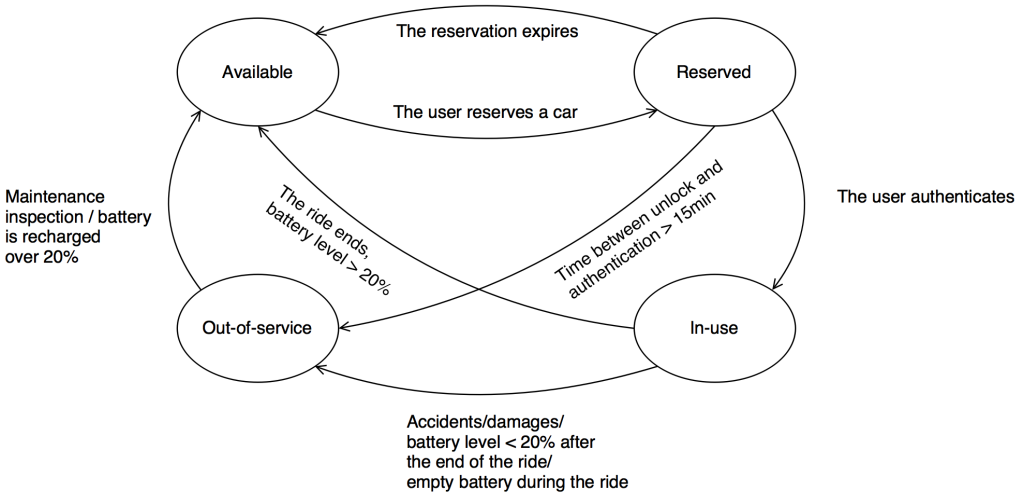


Figure 3.13: Statechart of the car usage.

3.2.6 Unlock Car

Purpose

This sub-part of the "Use Car" functionality focuses on the sequence of sub-functionalities involved in the car unlocking process.

The purpose of the functionality is that of allowing the user to unlock the car he/she reserved based on his/her actual proximity to the vehicle - given the GPS of a mobile device is available - or upon the insertion of a vehicle-specific code into the application.

Scenario 1

Damian reserved a *PowerEnJoy* car and is walking towards the vehicle location. When he reaches the car, he takes out his smartphone and opens the *"Car Reservation"* function of the mobile app. The system shows the current reservation: since Damian is close enough to the car and the GPS is active on his mobile device, the *"Unlock car"* button has been activated. He taps on the button and the car unlocks, enabling him to enter it.

Scenario 2

Elsa stayed out later than usual to study for her exams, and decided to rent a *PowerEnJoy* car with her laptop. Her smartphone has low battery, so she decided to turn the GPS off to make the battery last longer. Once

she gets to the car, she tries to unlock it by making the system detect her proximity. Since the button does not activate, she remembers that the GPS is off and decides to input the car code into the mobile app; she then finds the dedicated car code, opens the right function by tapping on the *"Unlock with car code"* button and inputs it. The system processes the request and, having verified that the user who inserted the code is the same who reserved the car, properly unlocks the vehicle and lets Elsa enter it.

Use-case

A detail of the use-case is provided in Table 3.9

Sequence diagram

A UML sequence diagram is provided to detail the alternative flows that can occur in case the user tries to unlock the car with the vehicle-specific code mentioned above. The diagram is shown in Figure 3.14

Functional requirements

1. Upon every try, the system must check that the user trying to unlock the car is the same that reserved it;
2. The system must guarantee to recognize the proximity of the user if he/she is within a 2 m radius from the car and has the GPS active on a mobile device;
3. The system must always guarantee the possibility for the user to unlock the vehicle he/she reserved by inserting the car code instead of using the GPS proximity detection system;
4. Upon every code insertion, the system must check that the code is the one associated with the car reserved by the user who inserted the code;
5. The vehicle-specific code must be unique for each car.
6. The reservation expiration timer must stop as soon as the car is unlocked;

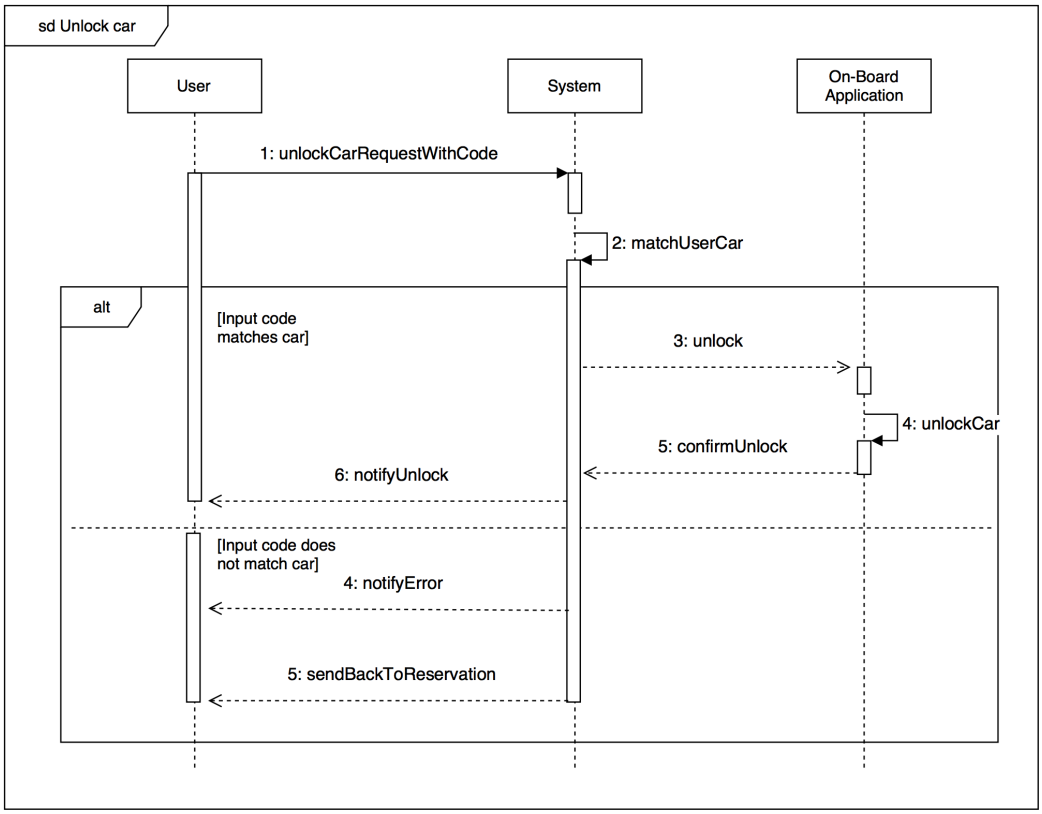


Figure 3.14: Sequence diagram for the unlock process in case the user decides to input the car code.

Actor	User
Goal	Goal 5
Input Condition	The user reserved a car and he wants to unlock it.
Event Flow	1. The user who reserved the vehicle approaches it and opens the mobile application; 2. The user opens the *"Car Reservation"* function and the system provides the sum-up of his/her active reservation; 3. The user either: has the GPS active and taps onto the *"Unlock car"* button OR reads and inserts the car code into the application, tapping the *"Unlock with car code"* button; 4. The system processes the request, by verifying that the user who tries to unlock the car is the one who reserved it; 5. The system unlocks the vehicle.
Output Condition	The car is unlocked and the user can access it.
Exception	If the user wants to unlock the car but he/she is further than 2 m away from it, the *"Unlock car"* button on the reservation screen will not be active. If the user inputs a wrong code when trying to unlock the car, the system will notify that the code is wrong and will not unlock the car, sending the user back to the reservation page on the application.

Table 3.9: Unlock car use-case

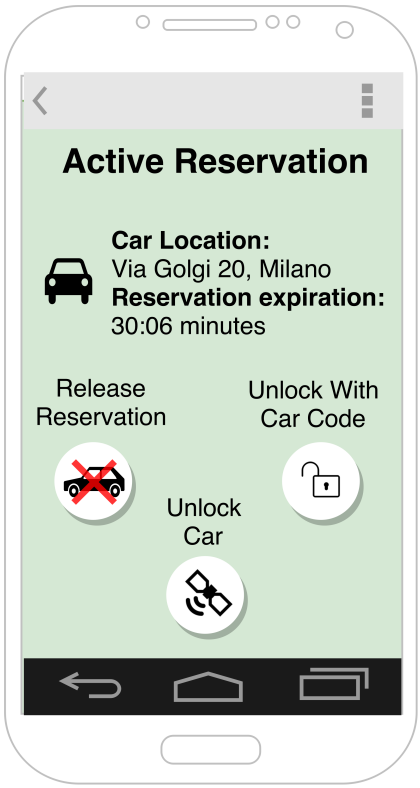


Figure 3.15: Mockup of the mobile unlock feature accessible from the reservation screen.

3.2.7 Start Ride

Purpose

This sub-part of the "Use Car" functionality focuses on the sub-functionalities concerned with the interaction between the user and the on-board application in order to allow the user to turn the engine on.

The system will interact with the user prompting him/her after the car has been unlocked and asking to proceed with the specified authentication method. This process will grant access to the car systems, hence allowing the user to start the ride and igniting the engine.

The on-board application will also start charging the user as soon as he/she authenticates, all while showing him/her the current charges on the screen, along with the sat-nav that is also going to signal nearby charging stations and the boundaries of the Safe Area.

Scenario 1

Francis just unlocked the car he reserved using the *PowerEnJoy* service. He sits on the driver's side and closes the door. The screen of the on-board computer is turned on and the system prompts Francis to authenticate. He performs the authentication procedure correctly; the message *"You can now turn the engine on. The system is charging you based on the duration of the ride."* appears on the screen. Francis turns the engine on and he then proceeds to drive the car, starting his ride.

Scenario 2

Graham mounted on a *PowerEnJoy* vehicle he just reserved and unlocked. When he authenticates after being prompted by the system, he makes an error. The system then shows an error message. Graham then repeats the procedure properly; this time it is correct, and the system allows him to drive the car.

Use-case

A detail of the use-case is provided in Table 3.10.

Sequence diagram

A UML sequence diagram is provided to detail the standard behavior of the system upon performance of the authentication method. The diagram is in Figure 3.16.

Functional Requirements

1. The system must check that the authentication method is followed properly;
2. The system must start charging the user only after he/she authenticates;
3. The system must always notify the user about his/her current charges;
4. The system must mark the car as "in-use" as soon as the user authenticates;

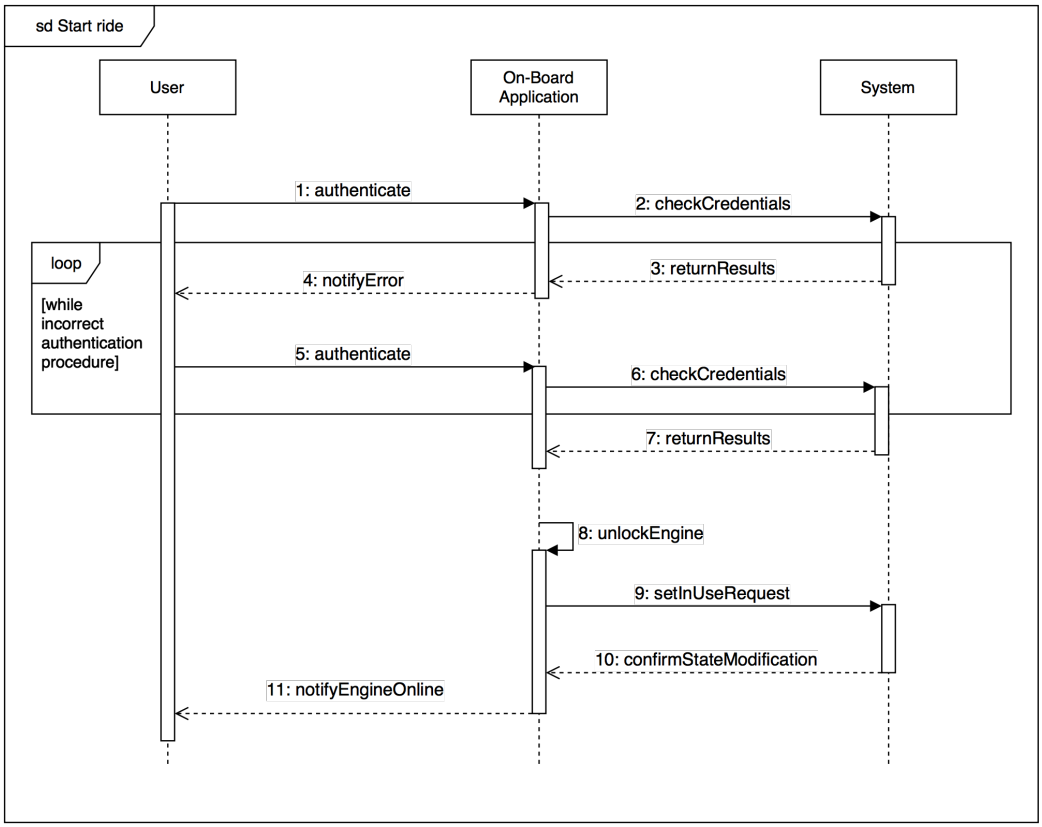


Figure 3.16: Sequence diagram for the process of starting a ride in a standard situation.

Actor	User
Goal	Goal 7
Input Condition	The user unlocked the car.
Event Flow	1. The user performs the authentication procedure interacting with the on-board screen; 2. The system analyzes the procedure correctness; 3. The system starts charging the user and marks the car as "in-use"; 4. The system lets the user to turn on the engine.
Output Condition	The user ignites the engine and starts the ride.
Exception	If the user makes an error while authenticating, the system will notify it and report an error message; in this case, the car still can not be turned on and its status remained "Reserved".

Table 3.10: Start ride use-case

3.2.8 End Ride

Purpose

This sub-part of the "Use Car" functionality focuses on allowing the user to conclude the current ride and assuring that, after the vehicles were used, they are parked within the Safe Area, properly locked and secured, ready to be rented and reserved again.

Scenario 1

Helen is driving a *PowerEnJoy* car in her city with her friend Irina. Once they arrive at their destination within the Safe Area, Helen pulls aside and parks the vehicle. She turns the engine off, and the system recognizes that the car is inside the Safe Area. As Helen gets off the vehicle and closes the door, the system detects that there still are people on board the car and waits before ending the ride. When Irina gets off as well, the system detects that the car is now empty and proceeds to lock the car and stop charging

Helen. The system marks the car as available again and Helen's current ride and its relevant data including the final cost are stored by the server.

Scenario 2

Lauren is outside the Safe Area, driving a *PowerEnJoy* car. She decides to end her ride, so she finds a spot where it is safe to leave the car. She stops and turns the engine off, but the on-board application warns her that, due to the fact that the car is outside the Safe Area, she is still being charged for the ride. Then, she turns the engine on again and moves towards the Safe Area to find a suitable parking spot.

Use-case

A detail of the use-case is provided in Table 3.11

Functional Requirements

1. The system must verify that every passenger - driver included - got off the cars before locking them and ending the ride;
2. The system must verify that the engine is turned off before ending the ride;
3. The system must check that the vehicle is stopped within the Safe Area before ending the ride;
4. The system must notify the user he/she didn't park the vehicle within the Safe Area;
5. The system must not allow to end a ride outside the Safe Area;
6. The system must keep charging the user if he/she tries to end the ride outside the Safe Area;
7. The system must stop charging the user as soon as the car is locked;
8. Every time a vehicle becomes "out-of-service" during a ride, the ride ends.

Actor	User
Goal	Goal 7
Input Condition	The user's ride is in progress.
Event Flow	1. The user stops the vehicle within the Safe Area and parks it, turning the engine off; 2. The system checks that every passenger, driver included, got off the vehicle; 3. The system locks the car; 4. The system stops charging the user.
Output Condition	The car is locked and the ride ends.
Exception	If a user tries to end a ride outside the Safe Area, he/she is notified that he/she is not allowed to and keeps being charged for the ride until he reaches the Safe Area and stops there. If a user does not turn the car engine off and gets off the car, the ride continues and he/she keeps being charged by the system.

Table 3.11: End ride use-case

3.2.9 Apply Additional Charges and Apply Discounts

Purpose

The system is able to incentivize virtuous users and discourage bad ones thanks to discounts and additional charges that are applied in the context of a specific ride. Whenever the system detects one or more of these situations, it notifies the user and updates the current charges for the ride.

Scenario 1

Glenn rents a car in order to go home after having fun with his friends. Two of them hitch a ride. As soon as the ride starts, Glenn receives a notification about a discount he will receive at the end of the ride itself. As a matter of fact, the system has detected the three people in the car as a possible discount situation and applies the discount accordingly.

Use-case

The apply additional charges use-case is shown in Table 3.12.
The apply discount use-case is shown in Table 3.13.

Functional requirements

1. The system must refer to a given set of virtuous and additional-charges situations;
2. The system must memorize, in the context of the current ride, the discounts and additional charges in order to correctly charge the user at the end of the ride;
3. The system must notify the user whenever a discount/additional charge situation is detected;
4. The system shall be aware of the number of people inside a rented car;
5. The system shall be aware of the battery level of any car;
6. The system must be able to detect whether a car is on charge;
7. The system must recognize if, within 5 minutes from the end of a ride, the user plugs the car to the grid;
8. The system must be able to identify the location of power grid stations in the SafeArea;
9. The system must be able to measure the distance between the cars and the power grids;
10. If the system detects two or more virtuous situations, only the largest discount will be applied;
11. If the system detects two or more situations of additional charges, all their effects on the current charge will be applied.

Actor	Bad user
Goal	Goal 8
Input Condition	In the context of the current ride, the system detects a situation that results in additional charges.
Event Flow	1. The system memorizes the additional charge and links it to the user current ride;
Output Condition	The system notifies the user about the additional charge.
Exception	None.

Table 3.12: Apply additional charges use-case

Actor	Virtuous user
Goal	Goal 8
Input Condition	In the context of the current ride, the system detects a situation that results in a discount.
Event Flow	1. The system checks for a pending discount on the current user ride; 2. If there is any, the system compares it with the new one and keeps the highest one; 3. The system memorizes the highest discount and links it to the user current ride;
Output Condition	The system notifies the user about the discount on the current charge.
Exception	None.

Table 3.13: Apply discount use-case

3.2.10 Apply Charges

Purpose

After 5 minutes from the end of the ride, that is the time the system gives to the user to plug the car to the grid, the systems computes the charge that has to be applied to the driver. The system takes into accounts discounts and additional charges as well as regular charges for the car utilization. After that, the system can manage the payment by sending a request to the payment handler (see section 3.2.12 for further details). If no errors occur, the system updates the rides history of the user.

Scenario 1

Rick has just safely parked his car into the boundaries of the Safe Area. He exits the car and end his ride. After 5 minutes, since during the ride no discounts/additional charges situations have been detected, the system charges him the only car utilization fare.

Scenario 2

During the ride performed by Carl, the systems has detected some additional charges situations. As soon as he parks and locks the car and 5 minutes pass, the system charges Carl not only the simple utilization fare, but takes also into account the additional charges.

Use-case

The apply charges use-case is shown in Table 3.14.

Activity diagram

The activity diagram showing the event flow is shown in figure 3.17

Functional requirements

1. The system must take into account discounts and additional charges when the final charge is calculated;
2. The system shall apply charges to the user within 5 minutes from the end of the ride;
3. The system shall calculate the charges with respect to the duration of the ride.

Actor	User
Goal	Goal 6
Input Condition	The user's ride comes to an end and 5 minutes pass.
Event Flow	1. The system calculates the amount of money the driver has to pay taking into account discounts and additional charges; 2. The system manages the payment.
Output Condition	The system updates rides history of the user.
Exception	If an error occurs at point 2 of the Event Flow, the system does not update the rides history and waits for the user's payment.

Table 3.14: Apply charges use-case

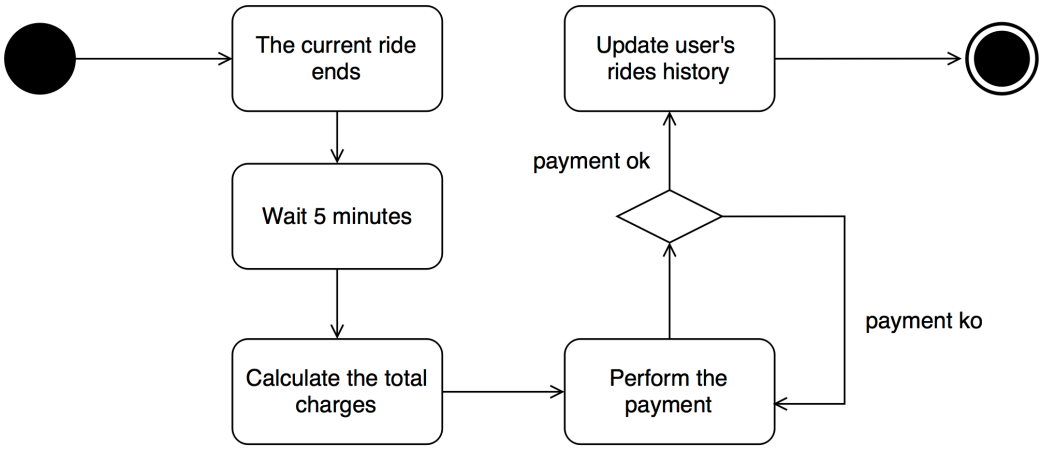


Figure 3.17: Activity diagram of the apply charges use-case from the system point of view.

3.2.11 Apply Fee

Purpose

The system must prevent any user's unfair behavior. Whenever the user decides to reserve a car for a ride, he/she has a limited amount of time to

reach the car and use it. This is reasonable because during this time nobody can rent that precise car. If the user does not come and use the reserved car, he/she must be penalized and fined by the system.

Moreover, in this case, the previously reserved car becomes available again and the system notify the bad user about the fine he/she has just received. Afterwards the system takes care to carry out the payment.

Scenario 1

Negan is dancing at the disco with his friends when he realizes that it's very late. He opens the *PowerEnJoy* application on his smartphone and checks for an available and nearby car to rent to go home. He finds one and reserve it.

Suddenly he notices a very beautiful girl and decides to speak to her in order to know her better. The clock is ticking and he forgets about his previous reservation. After more than one hour, Negan checks his mobile phone and notices the system has notified him about the expired reservation charging a fine.

Use-case

The apply fee use-case is shown in Table 3.15.

Functional requirements

1. The system must not allow a car to be reserved for more than 1 hour;
2. The system must notify the user about the expired reservation;
3. The system automatically charges the user in case of reservation expiration;
4. If the reserved car is not picked-up by the user within 1 hour from the reservation, the system tags the car available again;

Actor	Bad user
Goal	Goal 4
Input Condition	The user has reserved a car and he/she does not pick it up within 1 hour from the reservation
Event Flow	1. The system notifies the user about the expiration of his/her reservation and the application of a fee; 2. The system tags the previously reserved car as available;
Output Condition	The system takes care to perform the payment automatically.
Exception	None.

Table 3.15: Pay fee use-case

3.2.12 Manage Payments

Purpose

Every time a payment operation has to be carried out, the system takes care to interface with the payment handler and automatically charge the user. It could happen ether when a reservation expires or when a ride ends and charges are applied.

The user will pay by the payment method specified during the registration process. If a problem during the payment operation occurs because the payment handler refuses to complete it, the user account has to be suspended and the user is notified by the system. The user is notified even if the payment operation is successful.

Scenario 1

Federico concludes his current ride and parks within the Safe Area. As soon as the system applies the current charges, the payment is performed. Since Federico has enough money on his credit card, the operation is successfully completed and a notification is sent to the user with a summary of the charges.

Scenario 2

Vladimir has just bought a new deadly-expensive smartphone with his prepaid card and rents a car in order to get home from the shopping mall. After safely parking the car and ending the ride, the system applies the current charges and tries to carry out the payment. Unfortunately Vladimir has run out of money because of the purchase of the new mobile phone. The payment handler refuses the operation and the system immediately suspends Vladimir's account and sends him a notification. As soon as Vladimir has enough money on his card, the system automatically performs the payment and unlocks his account.

Use-case

The use-case about managing payments is shown in Table 3.16.

Sequence diagram

The sequence diagram describing the payment procedure is shown in Figure 3.18.

Functional requirements

1. The system must automatically perform payments after a ride ends and the charge is applied to the driver;
2. The system must perform payments whenever a user is fined due to a reservation expiration;
3. The system must suspend the user's account if the user cannot pay the charged amount of money;
4. The system must unlock a suspended user's account as soon as the payment can be carried out;
5. The system shall always notify the user after performing the payment;
6. The system must use the payment method specified by the user during the registration process;
7. The system must include an inteface to communicate with the payment handler system.

Actor	User, payment handler
Goal	Goal 6
Input Condition	The current charges have been applied to the user taking into account possible discounts/additional charges and fees.
Event Flow	1. The system interfaces to the payment handler; 2. The system waits for a positive answer; 3. As soon as the payment handler grants the money exchange, the system performs the payment;
Output Condition	The system notifies the user and updates the payment history. Moreover the system unlocks the user's account if it has been previously suspended.
Exception	If the payment handler blocks the payment request, the system suspends the user's account, restarts from point 1 of the Event Flow and notifies the user.

Table 3.16: Manage payments use-case

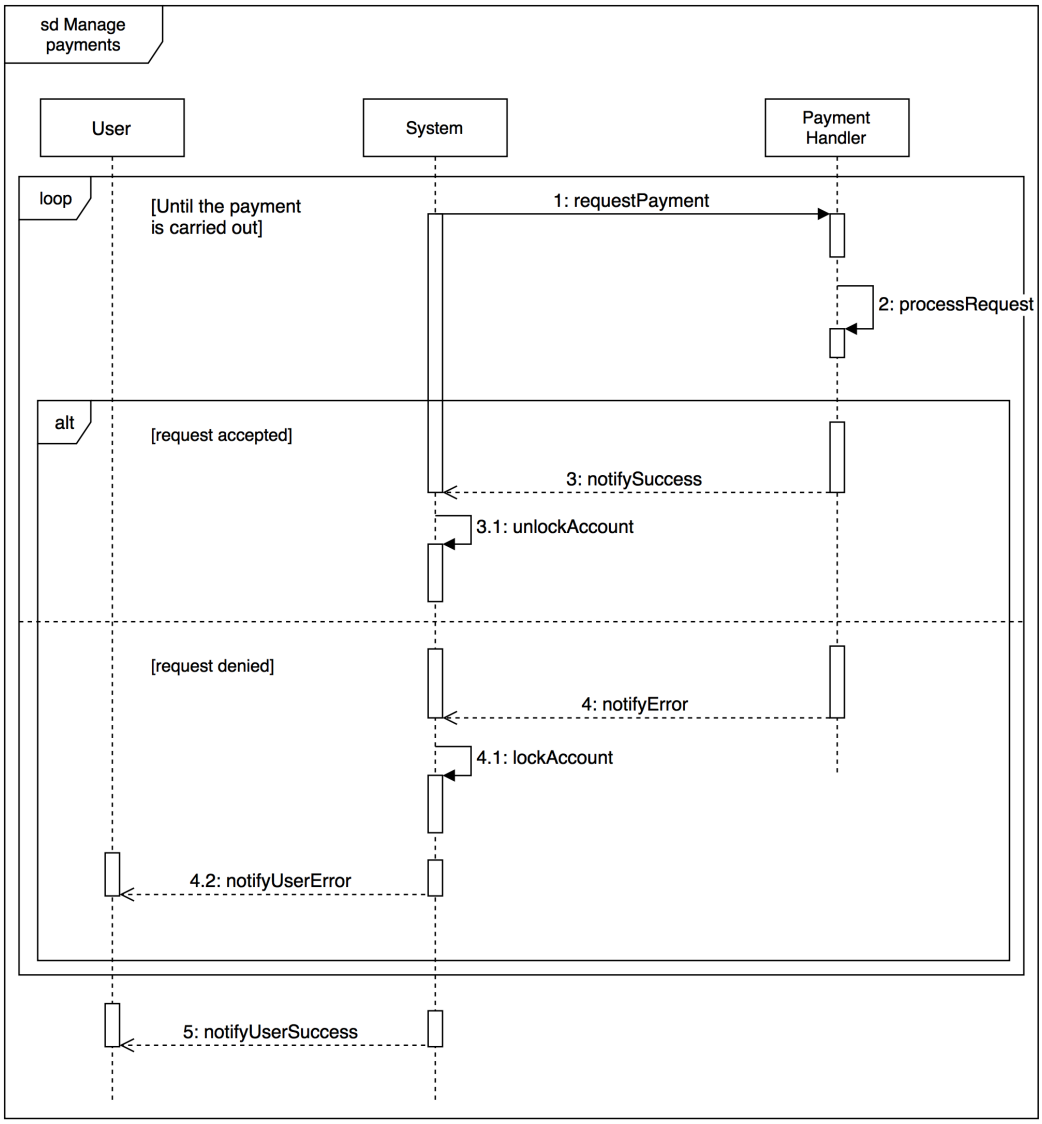


Figure 3.18: Sequence diagram of the payment procedure.

3.3 Performance Requirements

The following performance requirements must be satisfied by the system-to-be in order to guarantee the best user experience:

- The system must not have an upper-bound to the number of registered users;

- The system must handle at least 1000 active car reservations at the same time;
- The system must simultaneously support at least 1000 logged users who want to take the advantage of the service performing any kind of operation;
- 90% of car unlock requests shall be processed in less than 1 s;
- 100% of car unlock requests shall be processed in less than 2 s;
- 100% of any other kind of request shall be processed in less than 5 s.

  Since the performance is essentially bound by the internet connection, the above timing constraints are to be intended from when the system receives the request to the dispatch of the reply.

3.4 Software System Attributes

Reliability

The reliability of the system depends on several factors. First of all it is related to the reliability of the servers that run the back-end application. In addition to that, the reliability of the on-board computers must be also taken into account. Therefore the probability of failure of the devices mentioned above heavily affects the overall system reliability, that can be evaluated as:

$$Reliability = 1 - Probability\ of\ failure$$

Availability

The system must offer an availability of 99%. The remaining 1% shall take into account the time spent for ordinary maintenance sessions.

Security

- All the connections established between users and server must use the HTTPS protocol;
- All the communications between the server and the payment handler shall be encrypted;
- The users' passwords are stored in the database using a proper hashing mechanism.

Maintainability

The code of the whole software-to-be-developed must be thoroughly documented in order to let future developers easily know how the system works and how it has been designed. A version control system must be used to manage and organize all the different code revisions.

Portability

The server-side of the application shall be written in Java. Therefore it can be executed on any platform that supports the Java Virtual Machine and runs JEE. The web application must support the latest version of the main browsers like IE, Google Chrome and Firefox. The mobile application must be developed for both Android and iOS architectures. The car-side of the application is designed to run on the existing on-board computers.

3.5 Alloy

This section includes a class diagram which is representative of the relevant features of the system-to-be, followed by the Alloy code that describes and checks the consistency of the model provided in this document, as well as an example of the model-compliant world generated by the Alloy tool.

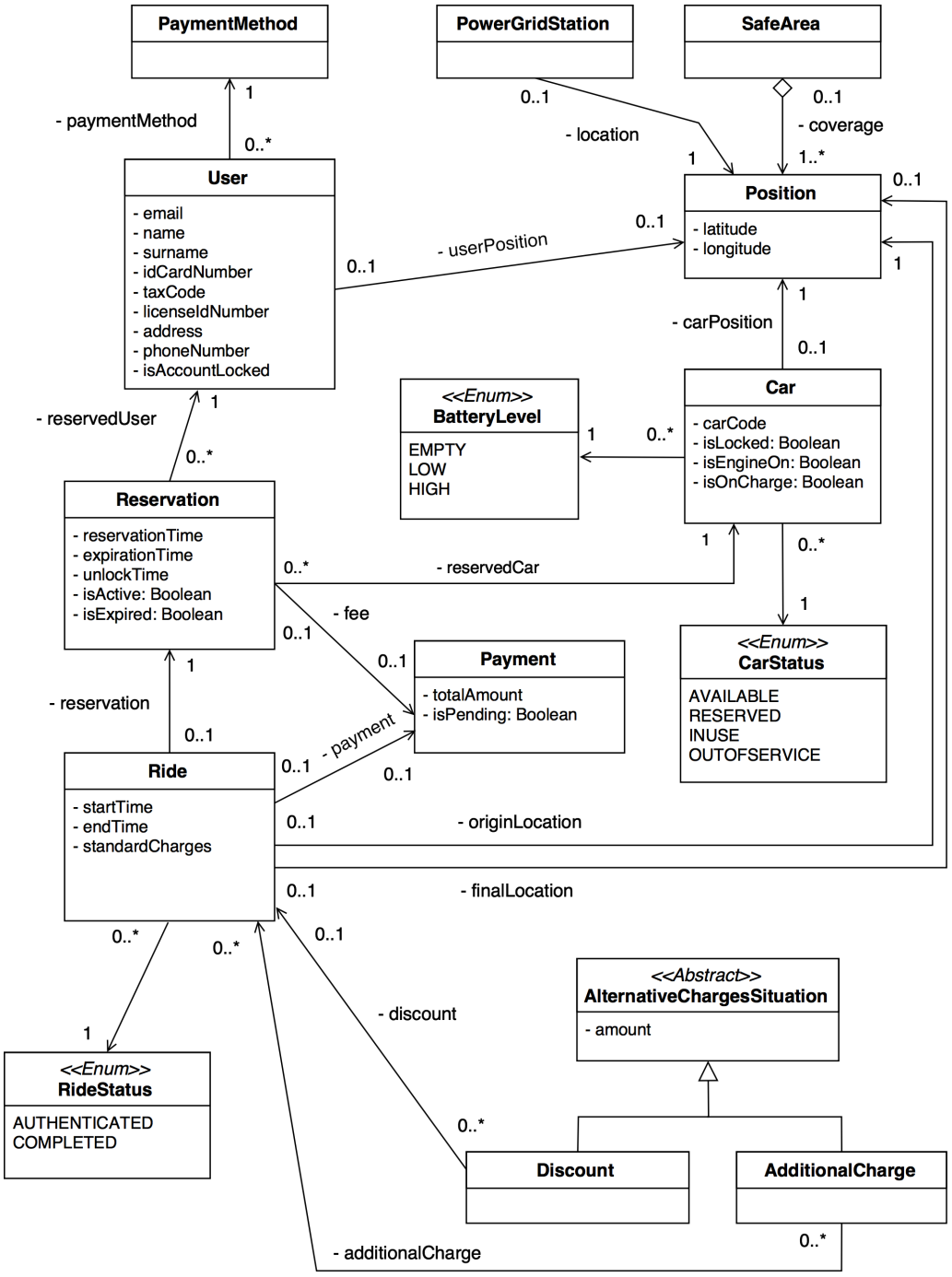


Figure 3.19: Class diagram illustrating the structure of the system-to-be.

```
open util/boolean

//UNIX time notation is used
//Dates are calculated as the number of seconds from the
//      1st of January 1970

//--------------SIGNATURES--------------

//String abstraction
sig StringPE {}

sig User {
    //<<email>> is modeled as an Int to allow comparisons
    email: one Int,
    name: one StringPE,
    surname: one StringPE,
    idCardNumber: one Int,
    taxCode: one Int,
    licenseIdNumber: one Int,
    address: one StringPE,
    phoneNumber: one Int,
    paymentMethod: one PaymentMethod,
    isAccountLocked: Bool,
    userPosition: lone Position,
}

sig PaymentMethod {}

sig Car {
    carCode: one Int,
    status: one CarStatus,
    batteryLevel: one BatteryLevel,
    //<<isLocked = True>> until the car door and
    //      the engine are unlocked
    isLocked: Bool,
    isEngineOn: Bool,
    isOnCharge: Bool,
    carPosition: one Position
} {
    carCode > 0
    status = AVAILABLE implies batteryLevel = HIGH
    status = RESERVED implies batteryLevel = HIGH
    status = INUSE implies not (batteryLevel = EMPTY)
```

```
44      batteryLevel = EMPTY implies status = OUTOFSERVICE
        isOnCharge = True implies isLocked = True
46      isOnCharge = True implies
            (one pg: PowerGridStation | pg.location = carPosition)
48  }

50  abstract sig CarStatus {}
    one sig AVAILABLE extends CarStatus {}
52  one sig OUTOFSERVICE extends CarStatus {}
    one sig RESERVED extends CarStatus {}
54  one sig INUSE extends CarStatus {}

56  abstract sig RideStatus {}
    one sig AUTHENTICATED extends RideStatus{}
58  one sig COMPLETED extends RideStatus{}

60  sig Reservation {
        reservedCar: one Car,
62      reservedUser: one User,
        reservationTime: one Int,
64      expirationTime: one Int,
        unlockTime: lone Int,
66      //<<isActive = True>> until the car becomes "INUSE"
        isActive: Bool,
68      isExpired: Bool,
        fee: lone Payment
70  } {
        reservationTime > 0
72      unlockTime > 0
        expirationTime > 0
74      reservationTime < expirationTime
        reservationTime < unlockTime
76      unlockTime < expirationTime

78      isExpired = True implies isActive = False
        one fee <=> isExpired = True
80  }

82  sig Ride {
        reservation: one Reservation,
84      startTime: one Int,
        endTime: lone Int,
86      originLocation: one Position,
```

```
    finalLocation: lone Position,
    rideStatus: one RideStatus,
    standardCharges: one Int,
    payment: lone Payment,
    discount: lone Discount,
    addCharges: set AdditionalCharge
} {
    rideStatus = COMPLETED <=> (startTime < endTime)
    startTime > reservation.unlockTime
    standardCharges > 0

    //If a ride exists, its associated reservation is not active
    //      anymore!
    reservation.isActive = False
    reservation.isExpired = False

    originLocation in SafeArea.coverage
    not (finalLocation in SafeArea.coverage)
        implies (reservation.reservedCar.status = OUTOFSERVICE)
}

//Float abstraction
sig Float {}

sig PowerGridStation {
    location: Position,
}

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

//The Safe Area is UNIQUE
one sig SafeArea {
    coverage: set Position,
} {
    #coverage>0
}

sig Payment {
    totalAmount: one Int,
    //<<isPending = True>> if the system was not able
```

```
130     //      to perform the payment
        isPending: Bool,
132 } {
        totalAmount > 0
134 }

136 abstract sig BatteryLevel {}
    one sig EMPTY extends BatteryLevel {}
138 one sig LOW extends BatteryLevel {}
    one sig HIGH extends BatteryLevel {}
140
    abstract sig AlternativeChargesSituation {
142     amount: one Int
    }{
144     amount > 0
    }
146 sig Discount extends AlternativeChargesSituation {}
    sig AdditionalCharge extends AlternativeChargesSituation {}
148

150 //--------------FACTS--------------

152 //AVAILABLE cars conditions
    fact AvailableForRentCar {
154     all c: Car | (c.status = AVAILABLE)
            implies (c. isLocked = True and c.isEngineOn = False
156             and c.carPosition in SafeArea.coverage and
                (no r: Reservation | r.reservedCar = c and
158             r.isActive = True))
    }
160
    //RESERVED cars conditions
162 fact ReservedCar {
        all c: Car | (c.status = RESERVED)
164         implies ((one res: Reservation | res.reservedCar = c
                                and (no r: Ride | r.reservation = res)
166             and res.isActive = True)
                and c.carPosition in SafeArea.coverage
168             and c.isLocked = True
                and c.isEngineOn = False)
170 }

172 //INUSE cars conditions
```

```
fact InUseCar {
    all c: Car | (c.status = INUSE)
        implies (c.isLocked = False and c.isOnCharge = False
            and (one r: Ride | r.rideStatus = AUTHENTICATED
                        and r.reservation.reservedCar = c))
}

//OUTOFSERVICE cars conditions
fact OutOfServiceCar {
    all c: Car | (c.status = OUTOFSERVICE)
        implies (c.isEngineOn = False and
                        (no res: Reservation | res.isActive = True
                        and res.reservedCar = c))
}

//All completed rides have a payment
fact AllCompletedRidesHavePayments {
    all r: Ride | r.rideStatus = COMPLETED
        <=> (one p: Payment | r.payment = p)
}

//All completed rides have a destionation and an end time
fact CompletedRidesHaveDestEnd {
    all r: Ride | r.rideStatus = COMPLETED
        <=> (one r.endTime and one r.finalLocation)
}

//If a ride has an end time, it must have a destination
fact NoEndDestWithoutEndTime {
    all r: Ride | no r.endTime <=> no r.finalLocation
}

//No two rides with the same payment
fact NoRidesWithSamePayment {
    no disj r1, r2: Ride | r1.payment = r2.payment
}

//No two reservations with the same fees
fact NoReservationsWithSameFee {
    no disj r1, r2: Reservation | (r1.fee = r2.fee
        and r1.isExpired = True and r2.isExpired = True)
}
```

```
216 //No couple <ride, reservation> that share a pay-transaction
    fact noRideAndResWithCoincidingFeeAndPayment {
218     no r: Ride, res: Reservation | (r.payment = res.fee
            and res.isExpired = True)
220 }

222 //No payment is not associated to any pay-transaction
    fact NoIsolatedPayment {
224     no p: Payment | ((no res: Reservation | res.fee = p)
            and (no r: Ride | r.payment = p))
226 }

228 //No two coinciding but distinct positions
    fact NoPositionOverlapping {
230     no disj p1, p2: Position |
            (p1.latitude = p2.latitude and p1.longitude = p2.longitude)
232 }

234 //No two distinct coinciding users
    fact UniqueUser {
236     no disj u1, u2: User | (u1 != u2 and
            (u1.email = u2.email or
238         u1.idCardNumber = u2.idCardNumber or
            u1.taxCode = u2.taxCode or
240         u1.licenseIdNumber = u2.licenseIdNumber))
    }
242
    //No two active reservations by the same user
244 fact UniqueReservation {
        no disj res1, res2: Reservation | (res1 != res2 and
246         (res1.reservedUser = res2.reservedUser
            and res1.isActive = True
248         and res2.isActive = True))
    }
250
    //Only one active reservation per RESERVED car
252 fact OnlyOneActiveResPerCar {
        all c: Car | c.status = RESERVED
254         <=> (one r: Reservation | r.isActive = True
            and r.reservedCar = c)
256 }

258 //No users with a locked account have active reservations
```

```
fact NoLockedUserActiveReservation {
    no res: Reservation | (res.isActive = True
        and res.reservedUser.isAccountLocked = True)
}

//No two distinct cars with the same car code
fact UniqueCarCode {
    no disj c1, c2: Car | (c1 != c2 and c1.carCode = c2.carCode)
}



//No two cars in the same position
fact NoCarPositionOverlapping {
    no disj c1, c2: Car | c1.carPosition = c2.carPosition
}

//No two users in the same position
fact NoUserPositionOverlapping {
    no disj u1, u2: User | u1.userPosition = u2.userPosition
}

//There can not be two rides for the same user that overlap in time
fact NoOverlappingRidesPerUser {
    all disj r1, r2: Ride |
        (r1.reservation.reservedUser = r2.reservation.reservedUser)
        implies
        (r1.endTime < r2.startTime or r2.endTime < r1.startTime)
}

//No two distinct rides with the same associated reservation
fact NoDisjRidesWithSameReservation {
    no disj r1, r2: Ride | r1.reservation = r2.reservation
}

//All charging stations are within the Safe Area
fact PowerGridInsideSafeArea {
    all pgs: PowerGridStation | pgs.location in SafeArea.coverage
}

//No two power grids in the same position
fact NoPowerGridOverlapping {
    no disj pg1, pg2: PowerGridStation |
```

```
302         pg1.location = pg2.location
    }
304
    //Conditions of AUTHENTICATED rides
306 fact NoAuthRideWithCarNotInUse {
        all r: Ride | r.rideStatus = AUTHENTICATED
308         <=> r.reservation.reservedCar.status = INUSE
    }
310
    //No AUTHENTICATED ride has a payment (yet)
312 fact NoPaymentIfAuthRide {
        no p: Payment | (some r: Ride | r.payment = p
314                                 and r.rideStatus = AUTHENTICATED)
    }
316
    //All pending pay-transactions correspond to an locked users
318 fact AllPendingRidePaymentsHaveLockedUser {
        all r: Ride | r.payment.isPending = True
320         <=> (r.reservation.reservedUser.isAccountLocked = True)
    }
322
    fact AllPendingReservationFeesHaveLockedUser {
324     all r: Reservation | r.fee.isPending = True
            <=> (r.reservedUser.isAccountLocked = True)
326 }

328 //No locked user can have an AUTHENTICATED ride
    fact NoLockedUserWithAuthRide {
330     all u: User | u.isAccountLocked = True
            implies (no r: Ride | (r.rideStatus = AUTHENTICATED
332                         and r.reservation.reservedUser = u))
    }
334
    //No locked user can have an active reservation
336 fact NoLockedUserWithActiveReservation {
        all u: User | u.isAccountLocked = True
338         implies (no res: Reservation | (res.isActive = True
                            and res.reservedUser = u))
340 }

342 //If a ride is not expired nor active, there is an associated ride
    fact NoUnexpiredReservationWithoutRide {
344     all res: Reservation |
```

```
            (res.isExpired = False and res.isActive = False)
346         <=> (one r: Ride | r.reservation = res)
    }
348
    //Account locked <=> (pending payment XOR pending fee)
350 fact NoTwoPendingPaymentsWithSameUser {
        all u: User | u.isAccountLocked = True <=>
352         (((one r: Ride | r.payment.isPending = True
                    and r.reservation.reservedUser = u)
354             and
                (no res: Reservation | res.fee.isPending = True
356                 and res.reservedUser = u))
            or
358             ((no r: Ride | r.payment.isPending = True
                    and r.reservation.reservedUser = u)
360             and
                (one res: Reservation | res.fee.isPending = True
362                 and res.reservedUser = u)))
    }
364
    //There are not locked users that have no reservations at all
366 fact LockedUsersHaveAtLeastOneReservation {
        all u: User | u.isAccountLocked = True
368         implies (some r: Reservation | r.reservedUser = u)
    }
370
    //All reservations correspond to a RESERVED car
372 fact AllReservationsHaveOneReservedCar {
        all r: Reservation | r.isActive = True
374         <=> r.reservedCar.status = RESERVED
    }
376
    //--------------ASSERTIONS--------------
378
    //An AUTHENTICATED ride must not have a payment
380 assert NoAuthRideWithPayment {
        no r: Ride, p: Payment | r.payment = p
382         and r.rideStatus = AUTHENTICATED
    }
384
    //A COMPLETED ride must have a destination and end time
386 assert RideCompleted {
        all r: Ride | (r.rideStatus = COMPLETED) <=>
```

```
388         (one r.endTime and one r.finalLocation)
    }
390
    //There must be no rides with expired reservations
392 assert NoRideIfReservationExpired {
        no r: Ride | (r.reservation.isExpired = True)
394 }

396 //An AUTHENTICATED ride must not have a destination and
    //      end time
398 assert NoEndTimeFinalDestForAuthRides {
        all r: Ride | r.rideStatus = AUTHENTICATED <=>
400         (no r.endTime and no r.finalLocation)
    }
402
    //There must be no two authenticated rides for the same user
404 assert NoTwoAuthRidesSameUser {
        no disj r1, r2: Ride |
406         (r1.reservation.reservedUser = r2.reservation.reservedUser
                and r1.rideStatus = AUTHENTICATED
408             and r2.rideStatus = AUTHENTICATED)
    }
410
    //All cars INUSE must be unlocked
412 assert AllInUseCarsAreUnlocked {
        all c: Car | c.status = INUSE
414         implies c.isLocked = False
    }
416
    pred show() {}
418
    run show for 5
```

Here is the execution of checks on the listed assertions:

Executing "Check RideCompleted"
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=2 Symmetry=20
11829 vars. 945 primary vars. 28200 clauses. 77ms.
No counterexample found. Assertion may be valid. 42ms.

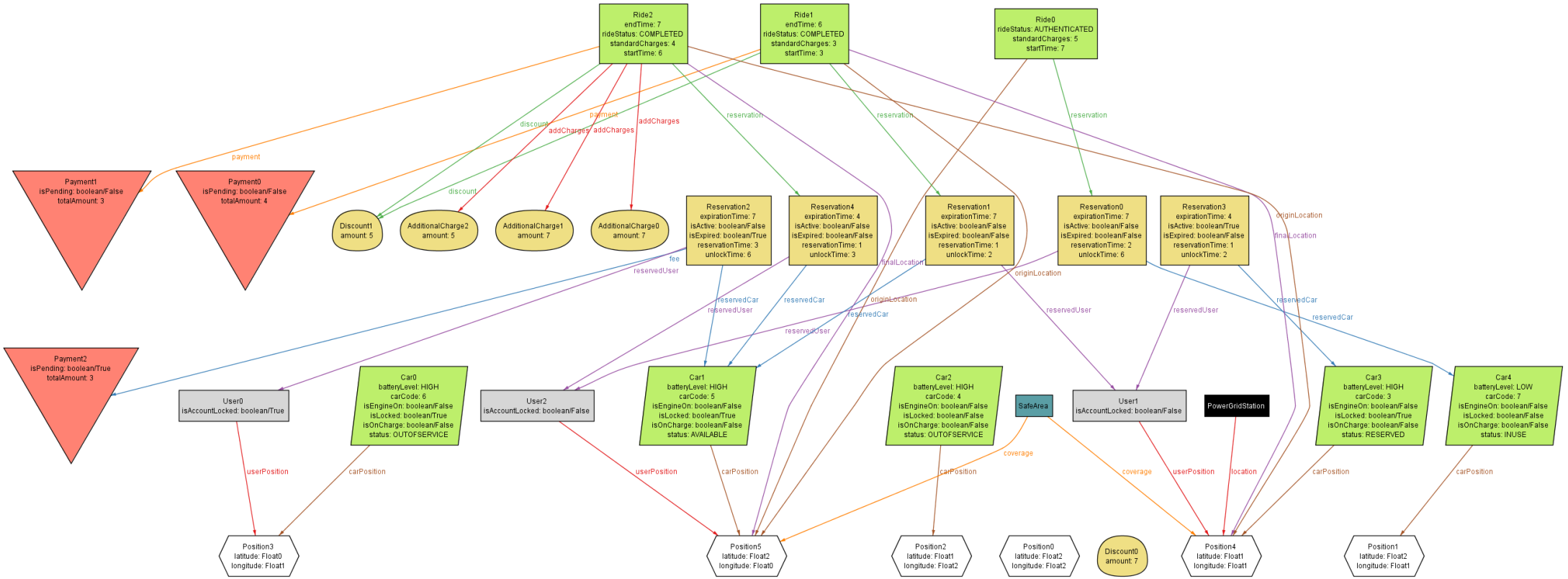
Executing "Check NoRideIfReservationExpired"
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=2 Symmetry=20
11727 vars. 945 primary vars. 27849 clauses. 81ms.

No counterexample found. Assertion may be valid. 7ms.

Executing "Check NoEndTimeFinalDestForAuthRides"
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=2 Symmetry=20
11795 vars. 945 primary vars. 28098 clauses. 51ms.
No counterexample found. Assertion may be valid. 9ms.

Executing "Check NoTwoAuthRidesSameUser"
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=2 Symmetry=20
11808 vars. 948 primary vars. 28038 clauses. 55ms.
No counterexample found. Assertion may be valid. 42ms.

Executing "Check AllInUseCarsAreUnlocked"
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=2 Symmetry=20
11734 vars. 945 primary vars. 27841 clauses. 41ms.
No counterexample found. Assertion may be valid. 3ms.


Ride2
endTime: 7
rideStatus: COMPLETED
standardCharges: 4
startTime: 6
Ride1
endTime: 6
rideStatus: COMPLETED
standardCharges: 3
startTime: 3
Ride0
rideStatus: AUTHENTICATED
standardCharges: 5
startTime: 7
Payment1
isPending: boolean/False
totalAmount: 3
Payment0
isPending: boolean/False
totalAmount: 4
Discount1
amount: 5
AdditionalCharge2
amount: 5
AdditionalCharge1
amount: 7
AdditionalCharge0
amount: 7
Reservation2
expirationTime: 7
isActive: boolean/False
isExpired: boolean/True
reservationTime: 3
unlockTime: 6
Reservation4
expirationTime: 4
isActive: boolean/False
isExpired: boolean/False
reservationTime: 1
unlockTime: 3
Reservation1
expirationTime: 7
isActive: boolean/False
isExpired: boolean/False
reservationTime: 1
unlockTime: 2
Reservation0
expirationTime: 7
isActive: boolean/False
isExpired: boolean/False
reservationTime: 2
unlockTime: 6
Reservation3
expirationTime: 4
isActive: boolean/True
isExpired: boolean/False
reservationTime: 1
unlockTime: 2
Payment2
isPending: boolean/True
totalAmount: 3
User0
isAccountLocked: boolean/True
Car0
batteryLevel: HIGH
carCode: 6
isEngineOn: boolean/False
isLocked: boolean/True
isOnCharge: boolean/False
status: OUTOFSERVICE
User2
isAccountLocked: boolean/False
Car1
batteryLevel: HIGH
carCode: 5
isEngineOn: boolean/False
isLocked: boolean/True
isOnCharge: boolean/False
status: AVAILABLE
Car2
batteryLevel: HIGH
carCode: 4
isEngineOn: boolean/False
isLocked: boolean/False
isOnCharge: boolean/False
status: OUTOFSERVICE
SafeArea
User1
isAccountLocked: boolean/False
PowerGridStation
Car3
batteryLevel: HIGH
carCode: 3
isEngineOn: boolean/False
isLocked: boolean/True
isOnCharge: boolean/False
status: RESERVED
Car4
batteryLevel: LOW
carCode: 7
isEngineOn: boolean/False
isLocked: boolean/False
isOnCharge: boolean/False
status: INUSE
Position3
latitude: Float0
longitude: Float1
Position5
latitude: Float2
longitude: Float0
Position2
latitude: Float1
longitude: Float2
Position0
latitude: Float2
longitude: Float2
Discount0
amount: 7
Position4
latitude: Float1
longitude: Float1
Position1
latitude: Float2
longitude: Float1
payment
discount
addCharges
reservation
originLocation
finalLocation
fee
reservedUser
reservedCar
coverage
userPosition
carPosition
location

Appendix A

Appendix

A.1 Software and tools used

- LaTeX, used as typesetting system to build this document.
- draw.io - `https://www.draw.io` - used to draw diagrams and mock-ups.
- GitHub - `https://github.com` - used to manage the different versions of the document and to make the distributed work much easier.
- GitHub Desktop, the GitHub official application that offers a seamless way to contribute to projects.
- Alloy Analyzer, used for analyzing our specifications.

A.2 Hours of work

The absolute major part of the document was produced in group work. The approximate number of hours of work for each member of the group is the following:

- Giovanni Scotti: ~65 hrs;
- Marco Trabucchi: ~60 hrs;

NOTE: indicated hours include the time spent in group work.

A.3 Changelog

Here are listed the changes and additions between different versions of the document:

V2 Added assumptions 16 and 17;

V2 Added functional requirement 7 in "Manage Payments";

V2 Added missing arc "reservedCar" in the class diagram;

V2 Fixed functional requirement 9 in "Use Car": "it being turned on" has been turned into "user authentication".

V2 Modified functional requirement 1 in "End Ride": added "and ending the ride";

V2 Added functional requirement 8 in "End Ride";

V3 Added "profile picture" requirement in Section 3.1.1;

V3 Added Subsection 2.7 containing information and picture about the World and Machine interpretation of the system-to-be;

V3 Removed not-so-reasonable domain assumption number 8 from the previous version;

V3.1 Small fix in World and Machine diagram.

Bibliography

[1] ISO/IEC/IEEE 29148:2011 *Systems and software engineering - Life cycle processes - Requirements engineering*

[2] IEEE 830:1998 *Recommended Practice for Software Requirements Specifications*

[3] AA 2016/2017 Software Engineering 2 - *Project goal, schedule and rules*

[4] Jackson, Zave - *The World and the Machine*