POLITECNICO
MILANO 1863

Politecnico di Milano
A.A. 2016–2017
Software Engineering 2: "PowerEnJoy"
Requirements Analysis and Specification Document

Pietro Ferretti, Nicole Gervasoni, Danilo Labanca

November 13, 2016

Contents

1 Introduction **3**
- 1.1 Purpose . . . 3
- 1.2 Scope . . . 3
- 1.3 Definitions, Acronyms, Abbreviations . . . 4
  - 1.3.1 Definitions . . . 4
  - 1.3.2 Acronyms . . . 5
  - 1.3.3 Abbreviations . . . 5
- 1.4 Reference Documents . . . 5
- 1.5 Document Overview . . . 6

2 Overall Description **7**
- 2.1 Product Perspective . . . 7
  - 2.1.1 Integration with external systems . . . 7
  - 2.1.2 Domain model . . . 7
- 2.2 Product Functions . . . 9
- 2.3 User Characteristics . . . 10
- 2.4 Constraints . . . 11
  - 2.4.1 Regulatory policies . . . 11
  - 2.4.2 Hardware limitations . . . 11
  - 2.4.3 Criticality of the application . . . 11
  - 2.4.4 Parallel operations . . . 11
  - 2.4.5 Safety and security considerations . . . 11
- 2.5 Assumptions and Dependencies . . . 12

3 Specific Requirements **14**
- 3.1 External Interface Requirements . . . 14
  - 3.1.1 User Interfaces . . . 14
  - 3.1.2 Hardware Interfaces . . . 18
  - 3.1.3 Software Interfaces (API) . . . 18
  - 3.1.4 Communication Interfaces . . . 19
- 3.2 Functional Requirements . . . 20
  - 3.2.1 Requirements . . . 20
  - 3.2.2 Use Cases . . . 26
- 3.3 Performance Requirements . . . 61
- 3.4 Software System Attributes . . . 62

4 Appendix **63**
- 4.1 Alloy Model . . . 63
- 4.2 Software and tools used . . . 74
- 4.3 Hours of work . . . 74

5 Revisions **75**
- 5.1 Changelog . . . 75

1 Introduction

1.1 Purpose

The purpose of this document is to provide a complete and detailed description and specification of a digital management system for PowerEnJoy, an electric car sharing service. This document will illustrate functional and non-functional requirements of the software to-be, outlining costraints, showing potential user interfaces for the software, and explaining the domain assumptions made. Additionally, at the end of the document we will present an Alloy model to further specificate the world and environment our system will have to manage.

This document is first and foremost a description of the project for all interested stakeholders and a reference for future development, but can also be used as a basis for legally binding agreements.

1.2 Scope

The aim of this project is to specify in detail a new digital management software for PowerEnJoy, a car-sharing service that employs electric cars only.

Users must be able to register to PowerEnJoy providing their personal data, driving license and payment information. After registering, users should be able to look for available cars near them or to a specific location. If they find a car that matches their needs, they can reserve the car up to one hour before using it. After approaching a reserved car users should have a way to unlock the car and enter it. While using a car, users are charged for a given amount of money per minute; a display on the car will inform the user of the amount of money they will have to pay at the end of the ride. The system stops charging the user after the car is parked in a safe area. Furthermore the system incentivizes virtuous behaviours by offering several discounts if certain conditions are met (e.g. charging a car at a power grid station).

To accomplish these goals we need to bridge the gap between the world (cars and passengers) and our system (servers, databases, etc.). The users will be able to access the system functionalities through a web and a mobile application, and a dedicated touchscreen installed in every car. A central system will manage the cars and the reservations, and will communicate with the cars through the Internet. The system will follow the cars' movements thanks to the GPS tracker installed in every car. Furthermore all cars will be equipped with a great variety of sensors to check the cars' status at any time.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- *Guest*: a person that is not registered to the system.
- *User*: a person that is registered to the system. Users can log in to the system with their email or username and their password. Their first name, last name, date of birth, driving license ID are stored in the database.
- *Safe area*: a location where the user can park and leave the car. Users can end their ride and park temporarily only in these locations. The set of safe areas is predefined by the system.
- *Power grid station*: also called charging station. A place where cars can be parked and plugged in. While a car is plugged in a power grid station its battery will be recharged. Power grid stations are by definition safe areas.
- *Available car*: a car that is currently not being used by any user, and has not been reserved either. Available cars are in good conditions (not dirty nor damaged) and don't have dead batteries.
- *Reservation*:
  - the operation of making a car reserved for a user, i.e. giving permission to unlock and use the car only for that user, forbidding reservations by other users.
  - the time period between the moment a reservation is requested and the moment the user unlocks the car, or the reservation is canceled.
- *Ride*: the time period from the moment a reserved car is unlocked to the moment the user notifies that he wants to stop using the car and closes all the doors. A ride doesn't stop when a car is temporarily parked, but continues until the user chooses to leave the car definitely.
- *Possession*: users that have reserved and unlocked a car are said to have possession of the car. While a user has possession of a car they are the only person that can drive it, lock or unlock it, and no other person can take possession of it until the user frees it. Users lose possession of a car when their ride ends.
- *Temporary parking*: the act of parking a car in a safe area and, after notifying the system, locking it and leaving it for a finite amount of time. The user that does this retains the right to use the car and can unlock it later to use it again.
- *Bill*: a record of the money owed by the user at the end of a ride.
- *Outstanding bill*: a bill that hasn't been paid yet.

- *Suspended user*: a user that cannot reserve or use cars. Usually users are suspended because they have outstanding bills.
- *Payment method*: a way to transfer money from the user to the system. Our system will only accept credit cards and online accounts like Paypal.
- *Payment API*: an interface to carry out money transactions, offered by the external provider associated to the payment method used (e.g. a bank).
- *CAN bus*: a vehicle bus standard designed to allow micro controllers and devices to communicate with each other.

1.3.2 Acronyms

- **RASD**: Requirements Analysis and Specification Document
- **DB**: Database
- **CVV**: Card Verification Value
- **DOB**: Date of birth
- **PGS**: Power Grid Station
- **GPS**: Global Positioning System
- **CAN bus**: Controller Area Network bus
- **MSO**: Money Saving Option

1.3.3 Abbreviations

- **[Gx]**: Goal
- **[RE.x]**: Functional Requirement
- **[UC.x]**: Use Case

1.4 Reference Documents

- ISO/IEC/IEEE Std. 29148:2011, "Systems and software engineering – Life cycle processes – Requirements engineering"
- Specification document: "Assignments AA 2016-2017.pdf"

1.5 Document Overview

This document is structured as follows:

- **Section 1 - Introduction:** it is a presentation of the document and the product; it contains the information needed to understand the whole document.
- **Section 2 - Overall Description:** gives general information about the software product with more focus about constraints and assumptions.
- **Section 3 - Specific Requirements:** it contains the product requirements list, use cases and associated UML diagrams; this section will delve deeper into the software functionalities. In addition, it includes some application mockups in order to produce a clearer vision on how the final product will look like.
- **Section 4 - Appendix:** it contains an Alloy model of the application domain, a list of all tools used to draw up this document and the hours of work each collaborator put into the creation of this document.

2 Overall Description

2.1 Product Perspective

2.1.1 Integration with external systems

Since there are no existing modules our system will need to integrate, PowerEnjoy is mainly a standalone system. The PowerEnjoy software will use APIs in order to accomplish few specific goals such as driving licence verification and card payment processing, more details about it can be found in section 3.1.3. The PowerEnjoy system is not planned to offer any APIs to external systems.

The whole system, software and databases, will be stored on the central servers. These will communicate with the car software which will control the car application given the sensors' feedbacks. Every request or data query from the application (both web and mobile version) will be sent to and elaborated by the servers.

2.1.2 Domain model

Here are presented both the state chart and the class diagram models in order to show a more formal and rigorous description of how the system will work and will be developed.

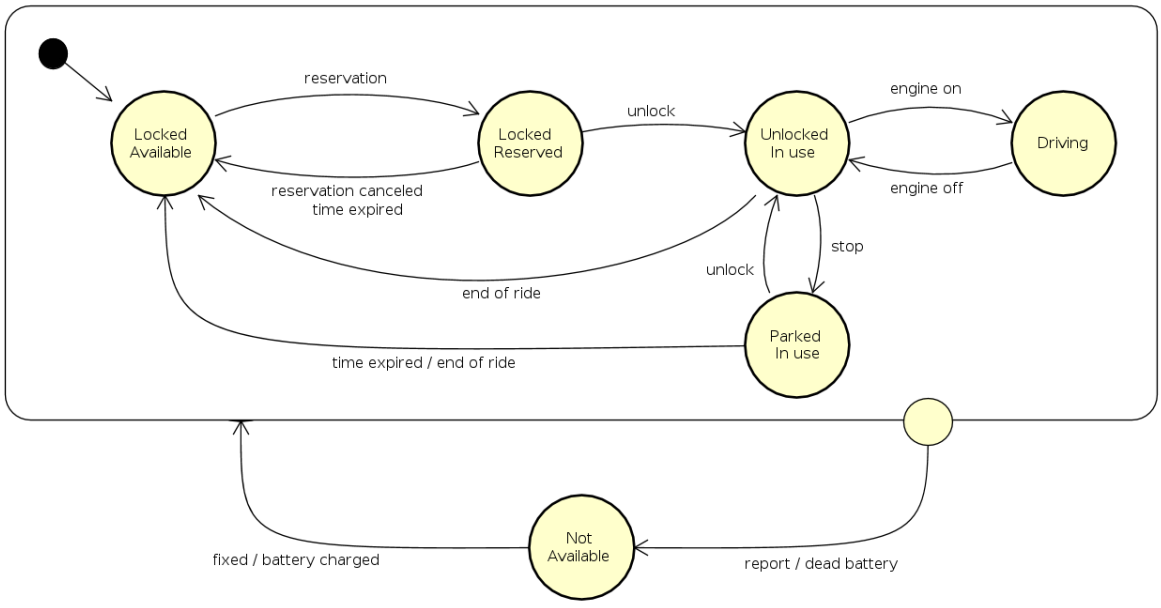


Figure 1: State Chart for cars

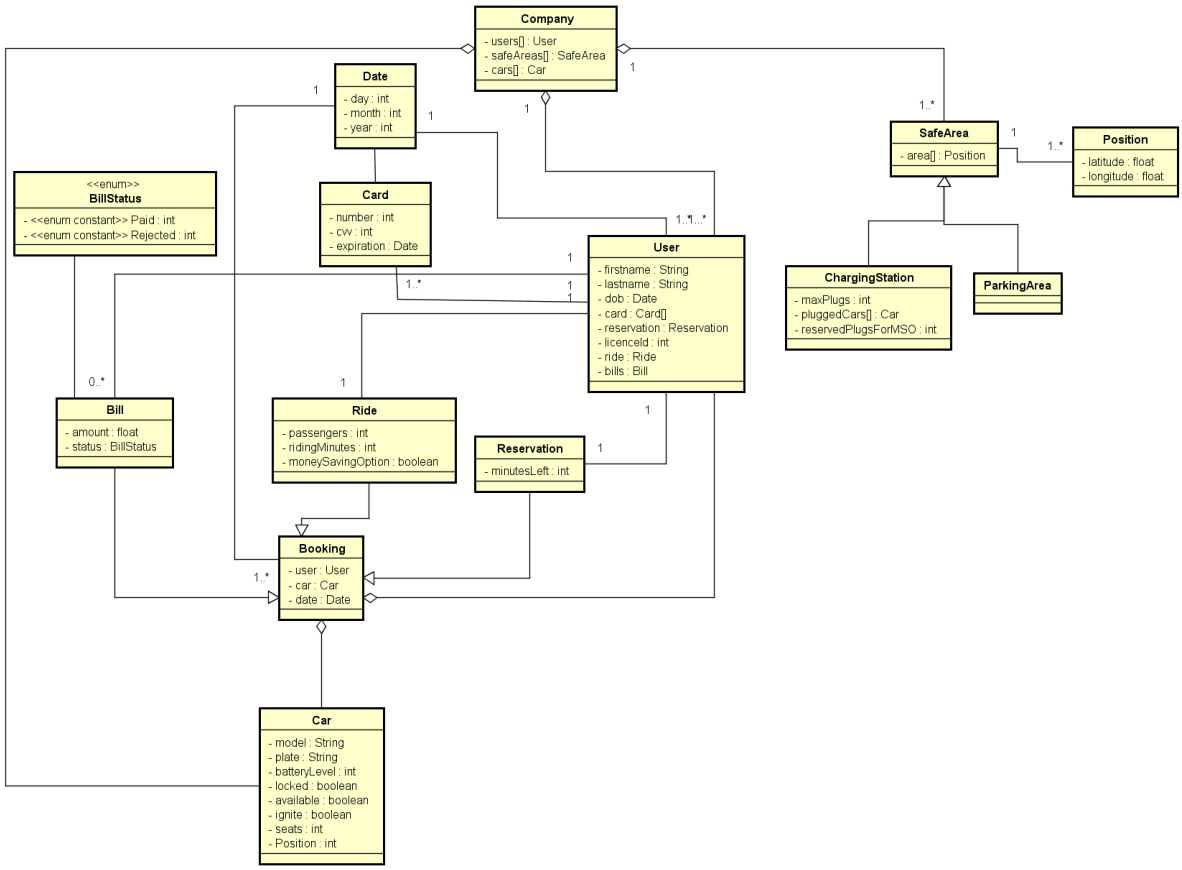


Figure 2: Class diagram

2.2 Product Functions

The PowerEnJoy application will offer many different functionalities; specifically, the goals we want to accomplish are the following:

[G1] Guests must be able to register as users by choosing a username and providing their personal data, driving license and payment information. They will receive a password at the email address they specified.

[G2] Users must be able to login with the username/email inserted and the password received on registration.

[G3a] Users must be able to find the location and battery charge of all the available cars within a certain distance from their current location.

[G3b] Users must be able to find the location and battery charge of all available cars around a specified location.

[G4] Users must be able to reserve a car for up to one hour before they pick it up. If they don't take the car before the time expires they are charged a fixed fee of 1 EUR.

[G5] Users must be able to cancel a reservation if they decide to not actually use the car.

[G6] A user that reaches a car reserved by them must have a way to tell the system they're nearby, so that the system unlocks the car and the user may enter.

[G7] The system must charge the user who reserved the car from the moment the engine is ignited after unlocking it. The user is charged for a given amount of money per minute.

[G8] The system must allow the user to see the amount they're being charged through a screen on the car.

[G9] The system must stop charging the user as soon as the car is parked in a safe area and the user exits the car notifying the system that they ended their ride.

[G10] Users must be able to leave a car without losing the reservation by informing the system that they're only temporarily parking; in this case the system continues on charging the user. The user should be able to end the ride without coming back to the car if they so choose.

[G11] The system must lock the car automatically after the user exits the car.

[G12] The system must provide a money saving option to the user, proposing a suitable power grid station near the user's final destination. The user will get a discount if they park the car there and plug it in the power grid.

[G13] The stations proposed by the system with the money saving option must be chosen in a way that ensures a uniform distribution of cars in the city.

[G14] The system must apply a discount of 10% on the last ride if the user took at least two other passengers onto the car.

[G15] The system must apply a discount of 20% on the last ride if the car is left with more than 50% of remaining battery charge.

[G16] The system must apply a discount of 30% on the last ride if the car is left at special parking areas where they can be recharged, and the user takes care of plugging the car into the power grid.

[G17] The system must charge 30% more if the car is left at more than 3 km from the nearest power grid station or with less than 20% remaining battery charge.

[G18] Users with outstanding bills cannot have access to the cars and reservations until the bills are paid.

[G19] Users with outstanding bills should have a way to pay them.

[G20] Available cars should actually be available to the user and in good conditions (i.e. not dirty, damaged, and/or with low battery).

[G21] Users must be able to easily find power grid stations.

2.3 User Characteristics

Users will use our application mainly to borrow cars and drive from one place to another to reach their destination. Considering that the users will have to drive cars, the typical user will at the very least have a driving license and be able to drive; users will also be necessarily of age.

Nonetheless, our users' level of skill and technological expertise can vary widely, from the young 20-something digital native with a modern smartphone, to the old couple that has some trouble with mobile applications and technology.

This means that our apps/interfaces must be as user-friendly as possible to allow a wide audience of people to easily use our applications and system.

2.4 Constraints

2.4.1 Regulatory policies

All the users are responsible for their behavior, from the insertion of truthful data to the respect of traffic laws.

The system must obtain the agreement from the user to acquire, process and store data.

The system must grant to the user the chance to delete the account and so remove from the DB each data concerning the user.

All the sensitive data are stored and transmitted to third parts society in accordance to the law.

2.4.2 Hardware limitations

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

- a mobile Internet connection (for example a 3G or 4G connection) that guarantee 1Mb/s.
- 50 MB of space

2.4.3 Criticality of the application

The system is not used in life-critical applications. The software doesn't have access to any critical hardware that could interfere with the car's functioning.

2.4.4 Parallel operations

The system must be able to respond multiple and simultaneous requests from users.

2.4.5 Safety and security considerations

All the knowledge regarding user's credentials, reservations, rides and bills must be kept private and accessible only to the user.

The position of a car could be accessible also both to employee and the user that reserved the car.

2.5 Assumptions and Dependencies

Car hardware dependencies To provide all the functionalities specified in this document, every car must have:

- a reliable GPS tracker;
- a mobile Internet connection (for example a 3G or 4G connection);
- sensors to collect data from the engine and the battery;
- an electronic switch to lock and unlock the doors;
- a weight sensor on each seat;
- a touchscreen to offer a dedicated interface to users;
- a GPS navigator to show the position of the power grid stations to the user and guide him;
- an internal system to control these components and communicate with the central system;
- a unique QR code printed on the outside of the car.

Web application dependencies The web application needs an Internet connection to work, for example to connect to the database and find available cars.

Mobile application dependencies The mobile application will need at least an Internet connection to receive data from and communicate with the central system. Access to the device's location is not mandatory, but is needed to provide the "find available cars near me" functionality. The mobile device will need a QR scanner to unlock the cars.

Power grid stations Power grid stations must have a sensor on each plug to know if it's free or in use, and an Internet connection to communicate this information to the system.

Perfect signal The car's data connection is always working and stable.
The car's GPS signal is always available and precise.

Predefined data The system has a predefined list of all the safe areas and their locations, list that can be changed by the people in control of the system. Likewise, the system has a predefined list of all the power grid stations, their locations and the number of plugs present in each station.

No idle users After unlocking a car users turn on the engine shortly after (i.e. the user doesn't stand in the car idly for long periods of time). We'd prefer if this happened rarely, because users aren't charged until the engine is ignited. We suppose that users want to get to their destination as soon as possible.

Users leave the car after parking When a user ends a ride or parks temporarily, we assume that the users and all the passengers have already exited the car when the last door is closed and the lock timeout expires.
This keeps the user or the passengers from getting stuck in the car when the system locks it automatically.

No cheating on the weight sensors When the weight measured by a seat weight sensor exceeds a certain threshold, we suppose that a person is present on that seat (and not an object).

Cars are recharged An external agency is entrusted with the task of recharging cars with low or dead batteries. The task of recharging cars doesn't involve our system, except for when a car is flagged as available again.

Users report problems Users always report dirty or damaged cars they find. Moreover, we trust users to never submit false reports.

Problems are fixed If a car is reported for a problem, an external agency is entrusted with the tasks of cleaning, repairing or fixing the car's problem. After the problem is fixed the agency marks the car as available in the system manually.

Theft is handled independently There is an office in charge of handling the cases when a car is flagged or reported as stolen. We suppose they're autonomous and don't need to interact with the system.

Cars aren't abandoned in bad spots We suppose users are reliable and never let the cars run out of battery before parking in a safe area.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

Our software will provide 3 main user interfaces: a mobile application, a web application and an application for the car's display. Through these interfaces the users must be able to access all the functionalities listed in section 2.2.

The design of:

- web application must follow W3C standard about HTML5 and CSS.
- iOS application must follow iOS Human Interface Guidelines described by Apple.
- Android application must follow Android Interface Guidelines about Material Design.

The interfaces must be user-friendly and easy to use. Every element's function must be clear, so that users can intuitively navigate the application without any confusion.

To avoid clunkiness and make the application easy to use, every functionality must be reachable in a maximum of 3 clicks and/or taps from any page.

- **Web Application UI**

The application will have a responsive interface to ensure the optimal view for every device screen. In this way owning the mobile application will not be mandatory in order to reserved a car.

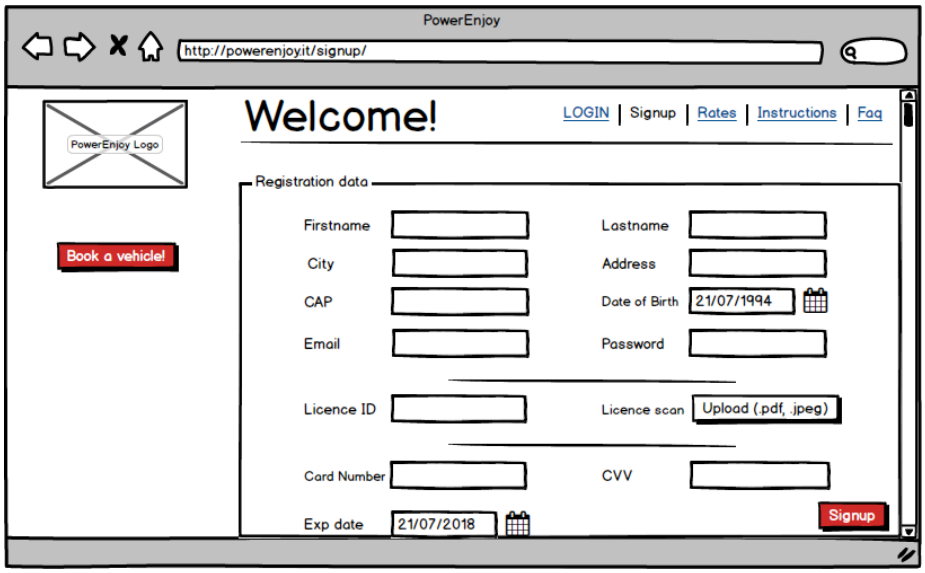


Figure 3: The web Sign Up form

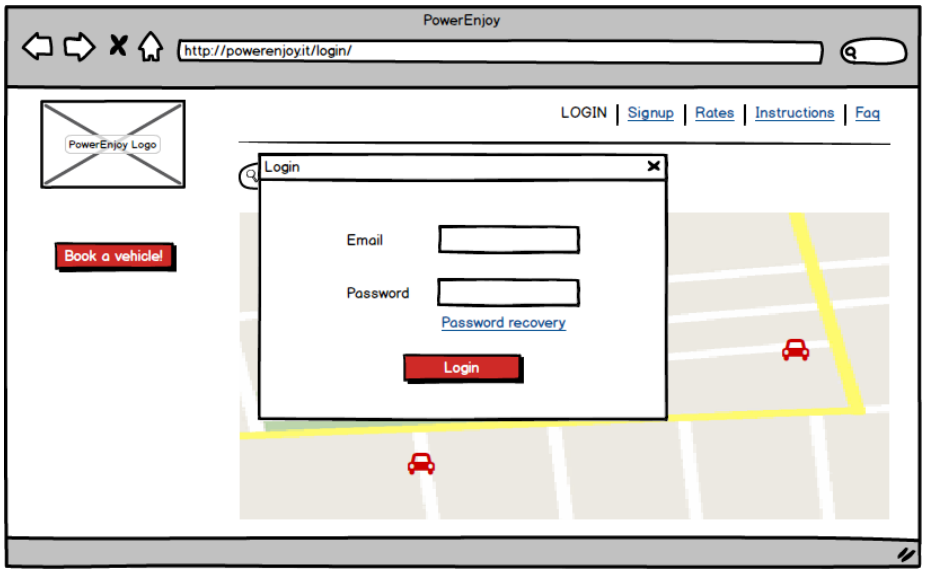


Figure 4: The web Login page

Figure 5: The dashboard, after logging in

- **Mobile application UI**

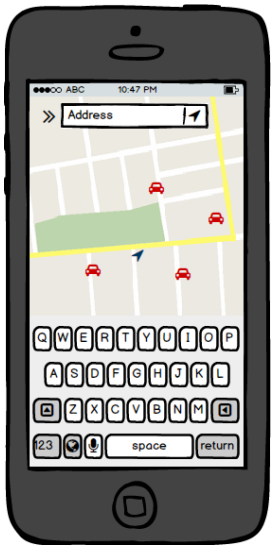


Figure 6: Map showing available cars

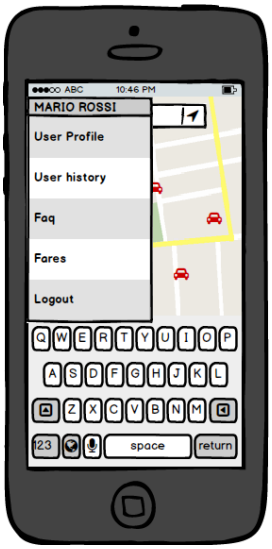


Figure 7: The mobile application menu

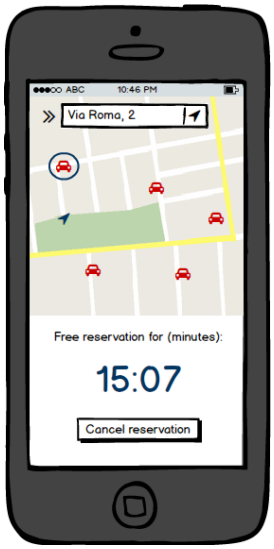


Figure 8: A car has been reserved

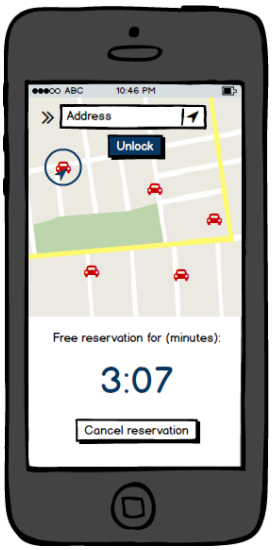


Figure 9: When the user is close to the car an unlock button is available

- **Car's display application UI**

The car's application interface, while driving, will always show the map, the battery level, the current cost of the journey and the number of passengers detected by the car's sensors.

Figure 10: The car's display while a user is driving

3.1.2 Hardware Interfaces

The car's internal system must interface with the car's CAN bus to collect information about the engine, battery and all the sensors installed on the car (GPS, weight sensors, etc.). The car's system will communicate this information to the central servers.

The car should also have a physical port to update the car's software if needed.

3.1.3 Software Interfaces (API)

The system requires:

- *Payment APIs*:
  - *VISA circuit API*: VISA Developer Center
  - *MasterCard circuit API*: MasterCard Developer Center
  - *PayPal API*: PayPal API Overview
- *Driving License Validation API*:
  - *Eucaris API*: Eucaris

- *Sign Up APIs*:
  - *Facebook API*: Login with Facebook
  - *Google API*: Google Identity Platform
- *Mapping/Routing API*:
  - *GraphHopper API*: GraphHopper

The mobile application must interface with the operating system:

- *iOS*
- *Android*
- *Windows Phone*

3.1.4 Communication Interfaces

All the communications between web application and system server or mobile application must be HTTPS request/response.

3.2 Functional Requirements

3.2.1 Requirements

[G1] Guests must be able to register as a user by choosing a username and providing their personal data, driving license and payment information. They will receive a password at the email address they specified.

- [RE.1.1] The system must offer an interface (e.g. a form on a web page or on the mobile application) where the user can enter and submit the data needed for registration: a username of their choice, their personal data (first name, last name, date of birth), a driving license ID or a photo of one, and a payment method.
- [RE.1.2] The system must verify that the data provided by a user on registration is valid.
- [RE.1.3] When a user registers the system must randomly generate a safe password and associate it to the user's account.
- [RE.1.4] After a user is registered the system must send the generated password to the user via email at the email address they provided.

[G2] Users must be able to log in with the username/email they submitted on registration and the password they received.

- [RE.2.1] The system must offer an interface (e.g. a form on a web page or on the mobile application) where the user can enter and submit their credentials (username/email and password) to log in.
- [RE.2.2] The system must compare the credentials submitted by the user with the ones saved in the database.
- [RE.2.3] If the credentials submitted on login are valid, the system authenticates the user and allows them to use the available functionalities.
- [RE.2.4] If the credentials submitted on login are not valid, the system denies access to the user and notifies them of the error.

[G3a] Users must be able to find the location and battery charge of all the available cars within a certain distance from their current location.

- [RE.3a.1] The system must offer an interface where the user can activate the option to look for nearby available cars if the user's location is available (e.g. thanks to a GPS receiver).
- [RE.3a.2] When the user activates the option to find nearby available cars, the system must get the user's location and look in the database for every available car near them.
- [RE.3a.3] The system must show the results of the search for nearby available cars on a suitable interface (e.g. a map).

[G3b] Users must be able to find the location and battery charge of all available cars around a specified location.

- [RE.3b.1] The system must offer an interface where the user can submit an address and search for available cars near that location.
- [RE.3b.2] When the user starts a search for available cars near a certain address, the system must look in the database for every available car close to the address provided by the user.
- [RE.3b.3] The system must show the results of the search for available cars on a suitable interface (e.g. a map).

[G4] Users must be able to reserve a car for up to one hour before they pick it up.

- [RE.4.1] The system must offer an interface that shows to the user the cars available for reservation (e.g. the search results specified in RE.3a.3) and allow the user to select a car.
- [RE.4.2] When a user selects a car for reservation the system must mark the car as reserved by that user.
- [RE.4.3] If the car has not been unlocked for an hour after reservation, the system must mark the car as available and charge the user a fixed amount (1 EUR).

[G5] Users must be able to cancel a reservation if they decide to not actually use the car.

- [RE.5.1] If the user has reserved a car the system must offer an interface to cancel the reservation.
- [RE.5.2] When the user selects to cancel the reservation, the system must free the car and allow the user to reserve another car.

[G6] A user that reaches a car reserved by them must have a way to tell the system they're nearby, so that the system unlocks the car and the user may enter.

- [RE.6.1] The system must offer an interface to unlock the car, with their position or the car's QR code.
- [RE.6.2] If the user is within a certain distance from the car, their device location is available and they choose from the interface to unlock the car with the position, the system must unlock the car.
- [RE.6.3] If the user chooses from the interface to unlock the car with the QR code and the user takes a clear picture of the QR code of the right car, the system must unlock the car.

[G7] The system must charge the user who reserved the car from the moment the engine is ignited after unlocking it. The user is charged for a given amount of money per minute.

- [RE.7.1] The system must keep track until the end of the reservation of the amount the user has to pay.
- [RE.7.2] The system starts to charge the user the moment the reserved car's engine is ignited after unlocking the car.
- [RE.7.3] The system increases the total amount charged by a given amount per minute.

[G8] The system must allow the user to see the amount they're being charged through a screen on the car.

- [RE.7.1] The system must keep track until the end of the reservation of the amount the user has to pay.
- [RE.8.1] After the system has started keeping track of the amount that the user has to pay, the system must show it in a clear way on the car screen while the car is in use and the reservation hasn't ended.

[G9] The system must stop charging the user as soon as the car is parked in a safe area and the user exits the car after ending the reservation.

- [RE.7.1] The system must keep track until the end of the reservation of the amount the user has to pay.
- [RE.9.1] The moment a car is parked in a safe area, the system must ask through an interface on the car screen if the user wants to end the reservation or just park temporarily.
- [RE.9.2] If a reserved car has been parked in a safe area and the user has chosen to end the reservation, then after the user has closed the doors the system must carry out the transaction for the amount that the system calculated, with the method of payment specified by the user on registration.

[G10] Users must be able to leave a car in a safe area without losing the reservation by informing the system that they're only temporarily parking; in this case the system keeps charging the user.

- [RE.7.1] The system must keep track until the end of the reservation of the amount the user has to pay.
- [RE.9.1] The moment a car is parked in a safe area, the system must ask through an interface on the car screen if the user wants to end the reservation or just park temporarily.
- [RE.10.1] When a car is temporarily parked and locked, the system keeps on charging the user but with a different rate than that used in RE.7.3.
- [RE.10.2] After a temporarily parked car is unlocked by the user and the engine is turned on, the system starts charging the user with the rate used in RE.7.3 again.

[G11] The system must lock the car automatically after the user exits the car.

- [RE.9.1] The moment a car is parked in a safe area, the system must ask through an interface on the car screen if the user wants to end the reservation or just park temporarily.
- [RE.11.1] After the user chose to end their reservation or park temporarily, the system must lock the car after 30 seconds from the moment the last door has been closed.

[G12] The system must provide a money saving option to the user, proposing a suitable station near the user's final destination. The user will get a discount if they park the car there and plug it in the power grid.

- [RE.12.1] The system must offer an interface where the user can select the money saving option and submit their final destination.
- [RE.12.2] When the user submits their final destination for the money saving option, the system must find a suitable power grid station near the ones close to the destination that have at least one free spot.
- [RE.12.3] The system must show on the car's navigator the location of the power grid station chosen for the money saving option and how to get there.
- [RE.12.4] The system must apply a discount on the amount the user has to pay if they selected the power saving option and they parked in the station the system chose for them.

[G13] The stations proposed by the system with the money saving option must be chosen in a way that ensures a uniform distribution of cars in the city.

- [RE.13.1] To choose a station in the process specified in RE.12.2, the system selects among all the stations with free spots within a certain distance of the user's final destination (e.g. a 10-minutes walk) the one with the fewest available cars nearby.

[G14] The system must apply a discount of 10% on the last ride if the user took at least two other passengers onto the car.

- [RE.7.1] The system must keep track until the end of the reservation of the amount the user has to pay.
- [RE.14.1] If the weight sensors in the car seats detect at least two passengers (excluding the driver) for at least half of the time the car has been driving, then the system must apply a discount of 10% at the end of the reservation on the amount the user has to pay.

[G15] The system must apply a discount of 20% on the last ride if the car is left with more than 50% of remaining battery charge.

- [RE.7.1] The system must keep track until the end of the reservation of the amount the user has to pay.

- [RE.15.1] When the car has been parked in a safe area and the user has chosen to end the reservation, the system must apply a discount of 20% on the amount the user has to pay if the car is left with more than 50% remaining battery charge.

[G16] The system must apply a discount of 30% on the last ride if the car is left at special parking areas where they can be recharged, and the user takes care of plugging the car into the power grid.

- [RE.7.1] The system must keep track until the end of the reservation of the amount the user has to pay.
- [RE.16.1] When the car has been parked in a power grid station, has been plugged in the power grid and the user has chosen to end the reservation, the system must apply a discount of 30% on the amount the user has to pay.

[G17] The system must charge 30% more if the car is left at more than 3 km from the nearest power grid station or with less than 20% remaining battery charge.

- [RE.7.1] The system must keep track until the end of the reservation of the amount the user has to pay.
- [RE.17.1] When the user has ended the reservation, if the car is parked more than 3 km from the nearest power grid station or has less than 20% remaining battery charge and is not plugged in the power grid, the system must increase the amount the user has to pay by 30%.

[G18] Users with outstanding bills cannot have access to the cars and reservations until the bills are paid.

- [RE.18.1] If a transaction fails the user must be marked as suspended until the outstanding bill is paid.
- [RE.18.2] Suspended users must not be allowed to reserve cars.

[G19] Users with outstanding bills should have a way to pay them.

- [RE.19.1] Users must have an option on the application to pay their outstanding bills.
- [RE.19.2] When the user chooses to pay their outstanding bills, if the transaction succeeds they are no longer marked as suspended, otherwise the user is notified and they remain suspended.
- [RE.19.3] Users must have a way to change the method of payment saved on the system through a dedicated option on the application.

[G20] Available cars should actually be available to the user and in good conditions (i.e. not dirty, damaged, and/or with low battery).

- [RE.20.1] Users should have a way to report problems with a car they reserved (excessive dirtiness, damage, etc.) through the interface on the car's display.
- [RE.20.2] If a user reports a car for dirtiness or damage, the system will mark it as unavailable, end the user's reservation and notify an employee to solve the problem.
- [RE.20.3] If a car has less than 20% battery charge at the end of a trip, the system will mark the car as unavailable and notify the employees in charge of recharging cars.
- [RE.20.4] If a car is unlocked by force or moves without any reservation, the system will consider it stolen and mark it as unavailable. The system notifies the people in charge of handling theft.
- [RE.20.5] If a car is repaired, cleaned or otherwise fixed, an employee will mark it as available through a dedicated interface.
- [RE.20.6] If a car flagged as unavailable because of low battery reaches 50% of battery charge while charging, the system will mark it as available again.

[G21] Users must be able to easily find power grid stations and safe areas.

- [RE.21.1] Users must have a way to search for power grid stations, for example with an option on the car navigator.
- [RE.21.2] Users must have a way to search for safe areas, for example with an option on the car navigator.

3.2.2 Use Cases

We identified the following use cases for our software:

- **[UC.1]** A guest registers to PowerEnJoy.
- **[UC.2]** A user logs in to the PowerEnJoy application.
- **[UC.3a]** A user searches for available cars near his position.
- **[UC.3b]** A user searches for available cars in a specific position.
- **[UC.4]** A user reserves a car.
- **[UC.5]** A user cancels a reservation for a car.
- **[UC.6a]** A user unlocks the car with the QR code printed on the car.
- **[UC.6b]** A user unlocks the car using his position.
- **[UC.7]** A user searches for a PGS.
- **[UC.8]** A user ends a ride.
- **[UC.9]** A user parks the car without ending the ride.
- **[UC.10]** A user ends a ride while the car is temporarily parked.
- **[UC.11]** The system suggests to the user a PGS to park the car at and save money.
- **[UC.12]** Payment
- **[UC.13]** A user pays an outstanding bill.
- **[UC.14]** A user reports a problem to the system.

Figure 11 shows the relationships between use cases and involved actors.

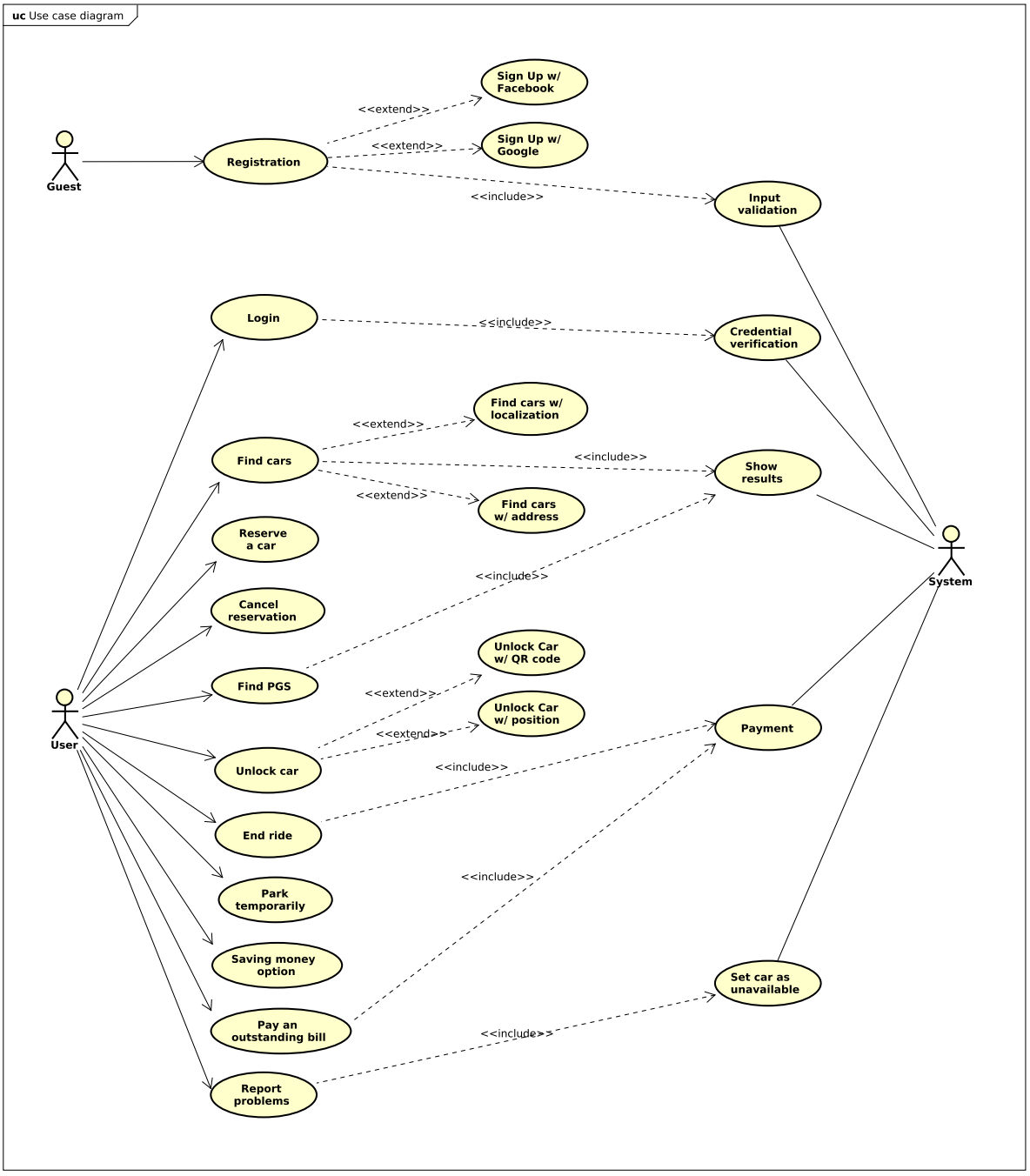
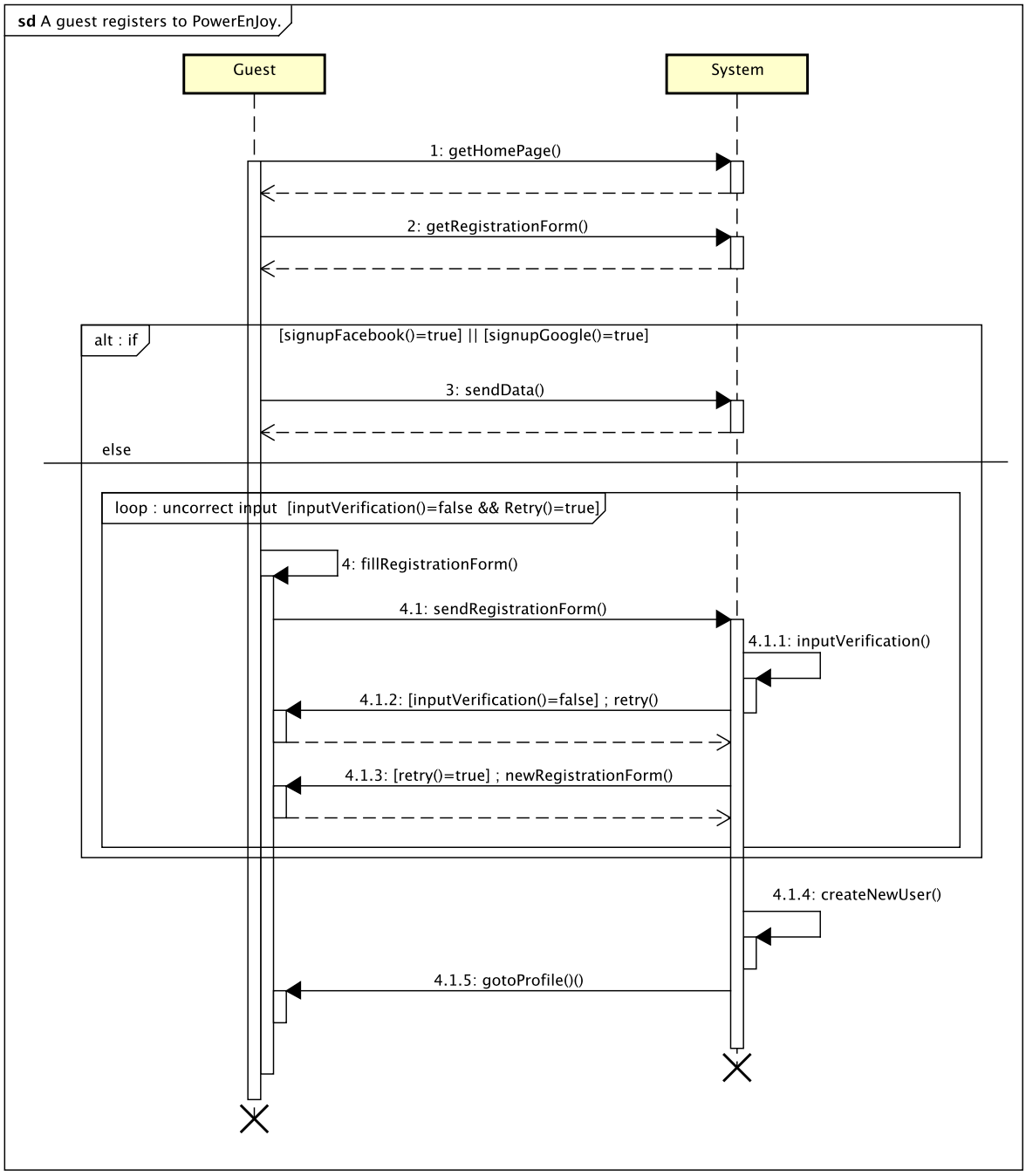


Figure 11: Use Case Diagram

[UC.1] A guest registers to PowerEnJoy.

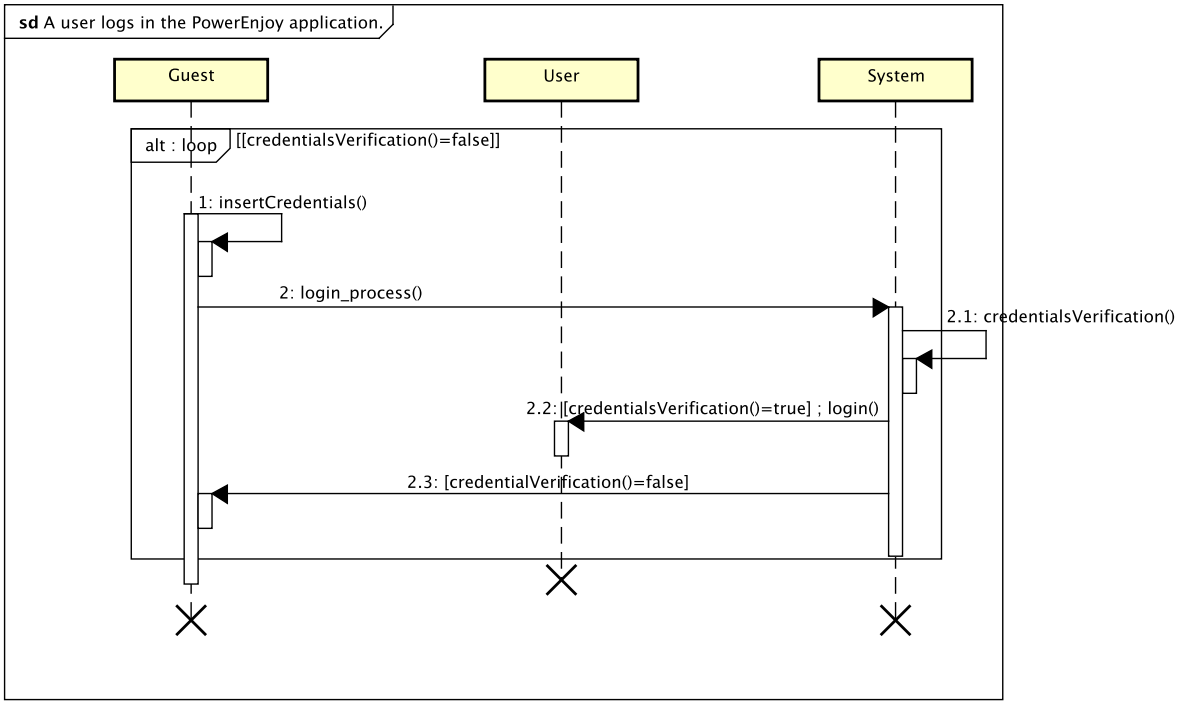
Actor	Guest
Goal	[G1]
Preconditions	The guest is not already registered
Execution Flow	1. The guest is on the home page and clicks on the "register" button to start the registration process. 2. The guest chooses between "Sign up with Google", "Sign up with Facebook" or inserting the data manually. 3. In the case of manually inserting, the guest fills in at least all mandatory fields with the required information (first name, last name, username, email address, DOB). 4. In the other cases, the guest connects to Google or Facebook and gives the system permission to take the data from there. 5. The guest uploads a photo of his driving license or inserts the information manually. 6. The guest inserts the number of the credit card and the corresponding CVV. 7. The guest gives their consent to the processing of their personal data by checking a box. 8. The guest clicks on the "confirm" button and submits the data. 9. In the case the guest uploads a photo, the system uses an OCR application to extract the data. 10. The system connects to the *Motorizzazione*'s data center to prove the validity of the driving license. 11. The system generates a password and sends it to the user's email address. 12. The system saves the data in the DB. 13. The system informs the user of the successful registration and redirects the user to the profile management page.

Postconditions	The guest successfully completes the registration process and become a user. From now on he or she can log in to the application using his credentials and use PowerEnJoy.
Exceptions	1. The guest is already registered. 2. The guest inserts invalid information. 3. The guest inserts a username used by another user. 4. The guest inserts an email address used by another user. 5. The guest doesn't give their consent to the processing of personal data. Each exception is handled by warning the guest of the problem and letting them insert the data again.


sd A guest registers to PowerEnJoy.
Guest
System
1: getHomePage()
2: getRegistrationForm()
alt : if
[signupFacebook()=true] || [signupGoogle()=true]
3: sendData()
else
loop : uncorrect input [inputVerification()=false && Retry()=true]
4: fillRegistrationForm()
4.1: sendRegistrationForm()
4.1.1: inputVerification()
4.1.2: [inputVerification()=false] ; retry()
4.1.3: [retry()=true] ; newRegistrationForm()
4.1.4: createNewUser()
4.1.5: gotoProfile()()

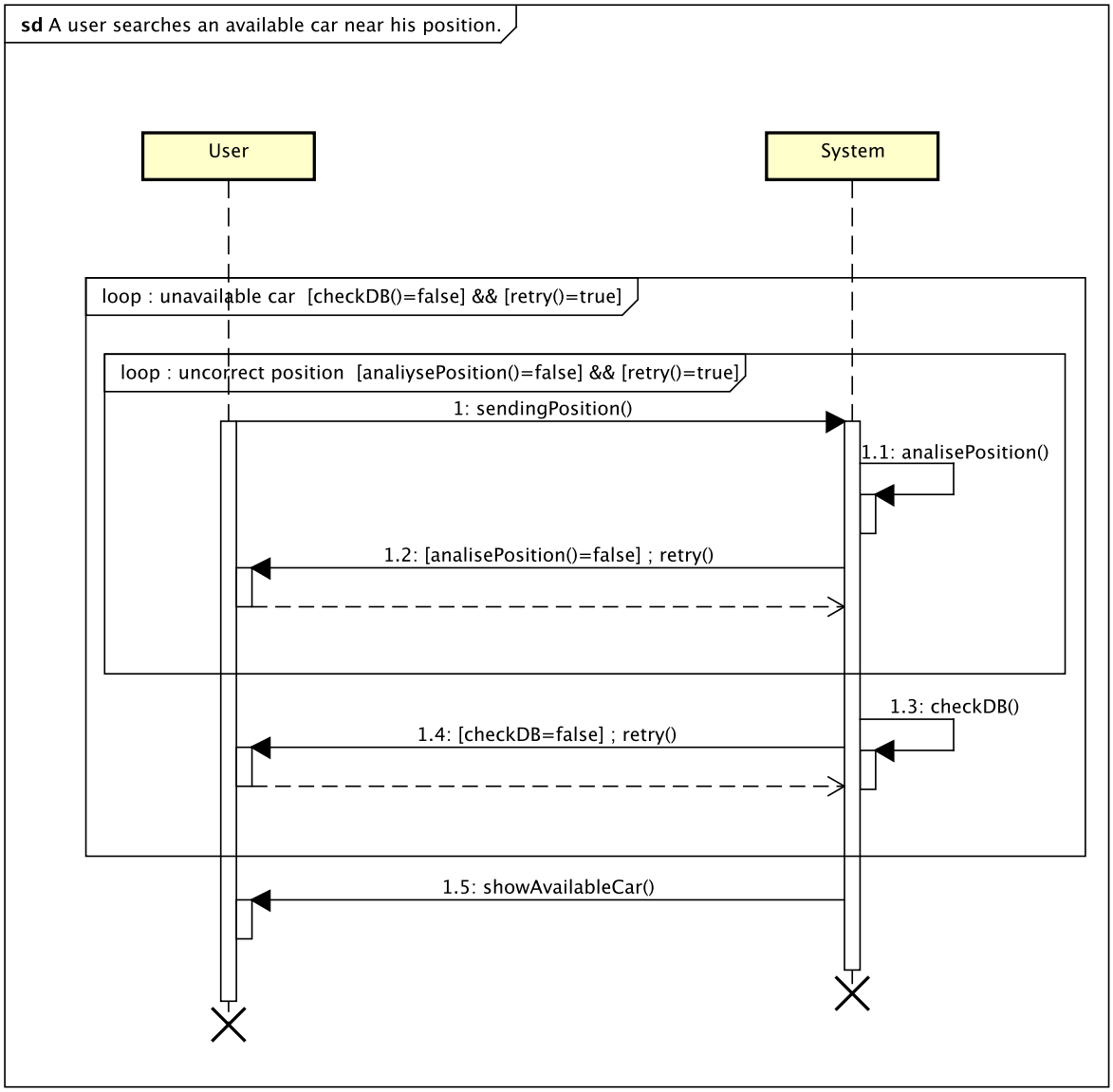
[UC.2] A user logs in to the PowerEnJoy application.

Actor	User
Goal	[G2]
Preconditions	The user must be registered in the system.
Execution Flow	1. The guest opens the PowerEnJoy application and presses on the login button. 2. The guest inserts the username or email and password received during registration and submits them. 3. The system checks the credentials submitted by the user. 4. The user is authenticated and redirected to the page where he can look for a car.
Postconditions	The user is now logged in and can use all the functionalities of the system.
Exceptions	1. The guest submits invalid credentials.


sd A user logs in the PowerEnjoy application.
Guest
User
System
alt : loop
[[credentialsVerification()=false]]
1: insertCredentials()
2: login_process()
2.1: credentialsVerification()
2.2: [credentialsVerification()=true] ; login()
2.3: [credentialVerification()=false]

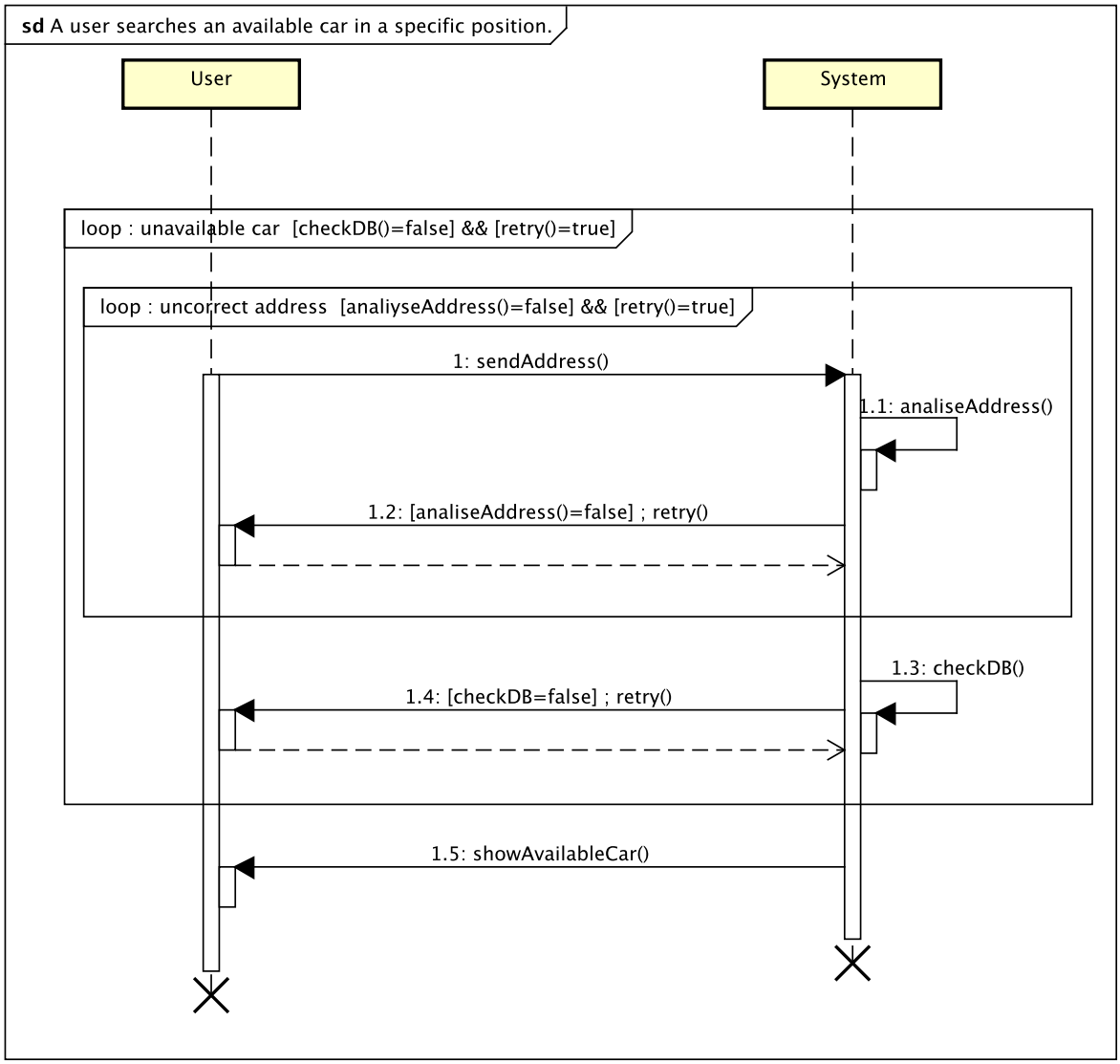
[UC.3a] A user searches for available cars near his position.

Actor	User
Goal	[G3a]
Preconditions	The user is logged in to the system and the GPS is active.
Execution Flow	1. The user presses the button to be localized on the map. 2. The system receives the user's position and searches the DB for all the available cars close to that position. 3. The application shows on a map all the cars that were found. 4. The user navigates on the map to choose a car. 5. The user selects a car he or her is interested in and sees its battery charge.
Postconditions	The user can see all the available cars and their battery charge.
Exceptions	1. There aren't any available cars, in this case the system will suggest to search in a different location.


sd A user searches an available car near his position.
User
System
loop : unavailable car [checkDB()=false] && [retry()=true]
loop : uncorrect position [analiysePosition()=false] && [retry()=true]
1: sendingPosition()
1.1: analisePosition()
1.2: [analisePosition()=false] ; retry()
1.3: checkDB()
1.4: [checkDB=false] ; retry()
1.5: showAvailableCar()

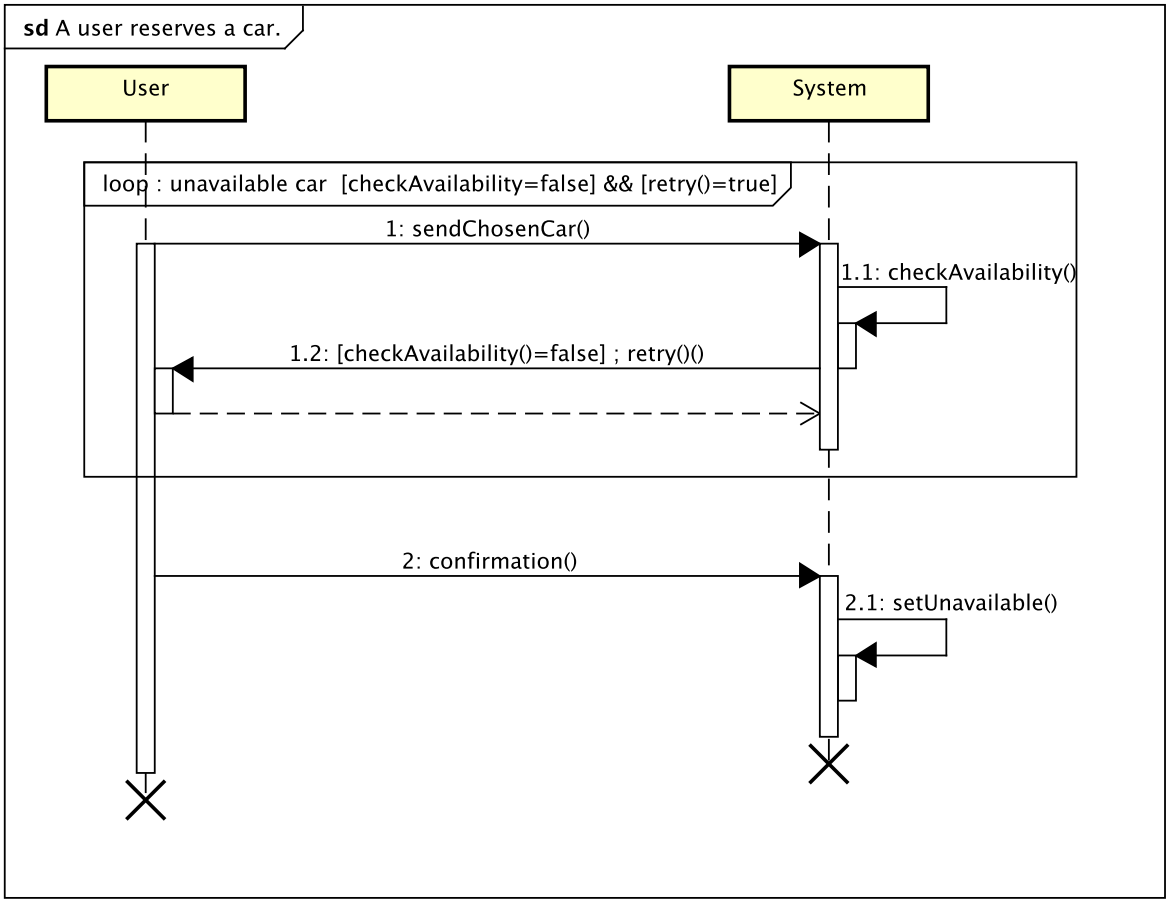
[UC.3b] A user searches for available cars in a specific position.

Actor	User
Goal	[G3b]
Preconditions	The user is logged in to the system
Execution Flow	1. The user presses the search bar to insert a location. 2. The user inserts an address or a name of a place 3. The system receives the user's input. 4. The system interprets the address or place's name and converts it into a position. 5. The system checks in the DB all the available cars close to that position. 6. The application shows on a map all the cars that were found. 7. The user navigates on the map to choose a car. 8. The user selects a car he or her is interested in and sees its battery charge.
Postconditions	The user can see all the available cars and their battery charge.
Exceptions	1. The address inserted by the user doesn't exist. 2. There aren't any available cars, in this case the system will suggest to search in a different location.


sd A user searches an available car in a specific position.
User
System
loop : unavailable car [checkDB()=false] && [retry()=true]
loop : uncorrect address [analiyseAddress()=false] && [retry()=true]
1: sendAddress()
1.1: analiseAddress()
1.2: [analiseAddress()=false] ; retry()
1.3: checkDB()
1.4: [checkDB=false] ; retry()
1.5: showAvailableCar()

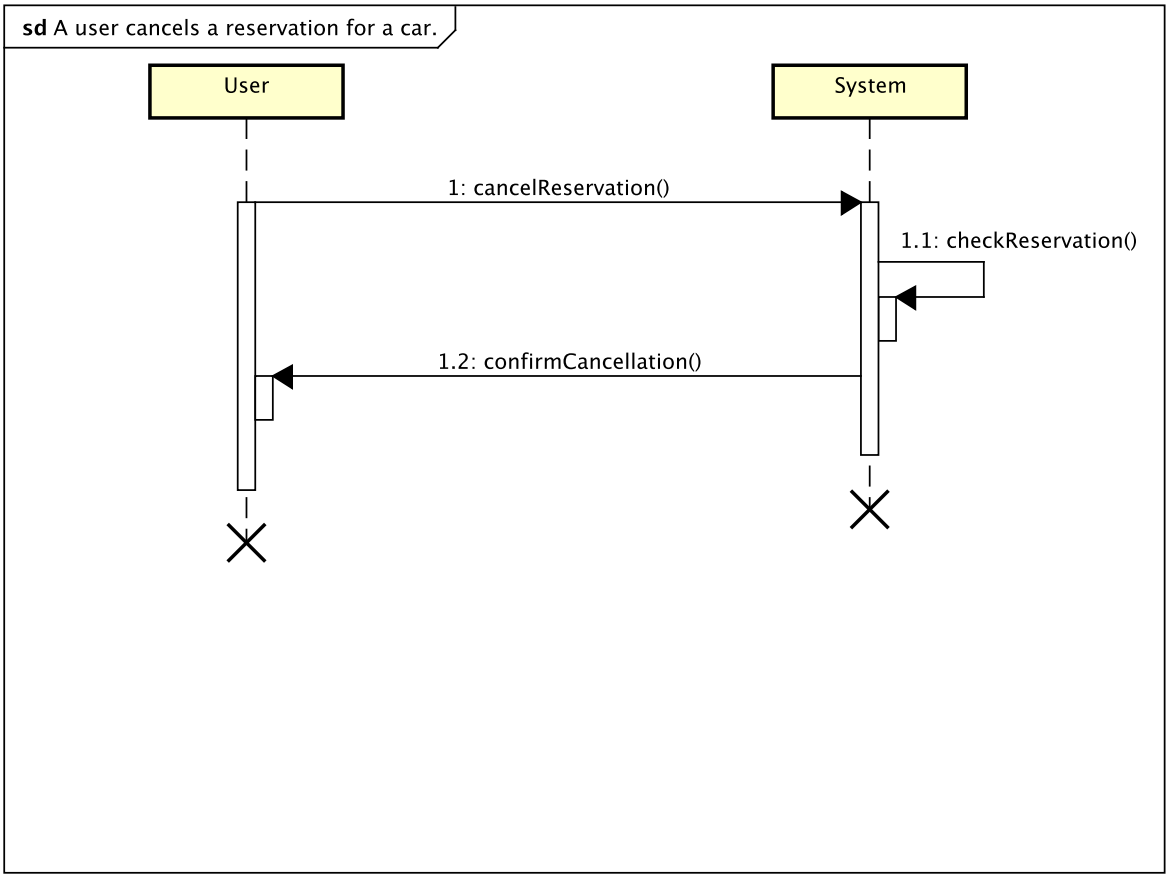
[UC.4] A user reserves a car.

Actor	User
Goal	[G4]
Preconditions	The user is logged in and there is at least an available car.
Execution Flow	1. The user selects a car on the map. 2. The system shows to the user the battery remaining charge. 3. The user confirms the selected car as their choice for reservation.
Postconditions	The car is reserved for the user for an hour.
Exceptions	1. The car is reserved by an another user before the user confirms the reservation. 2. The user is suspended by the system. In this case, the application doesn't let them reserve the car and reminds them that they have to pay the outstanding bill(s).


sd A user reserves a car.
User
System
loop : unavailable car [checkAvailability=false] && [retry()=true]
1: sendChosenCar()
1.1: checkAvailability()
1.2: [checkAvailability()=false] ; retry()()
2: confirmation()
2.1: setUnavailable()

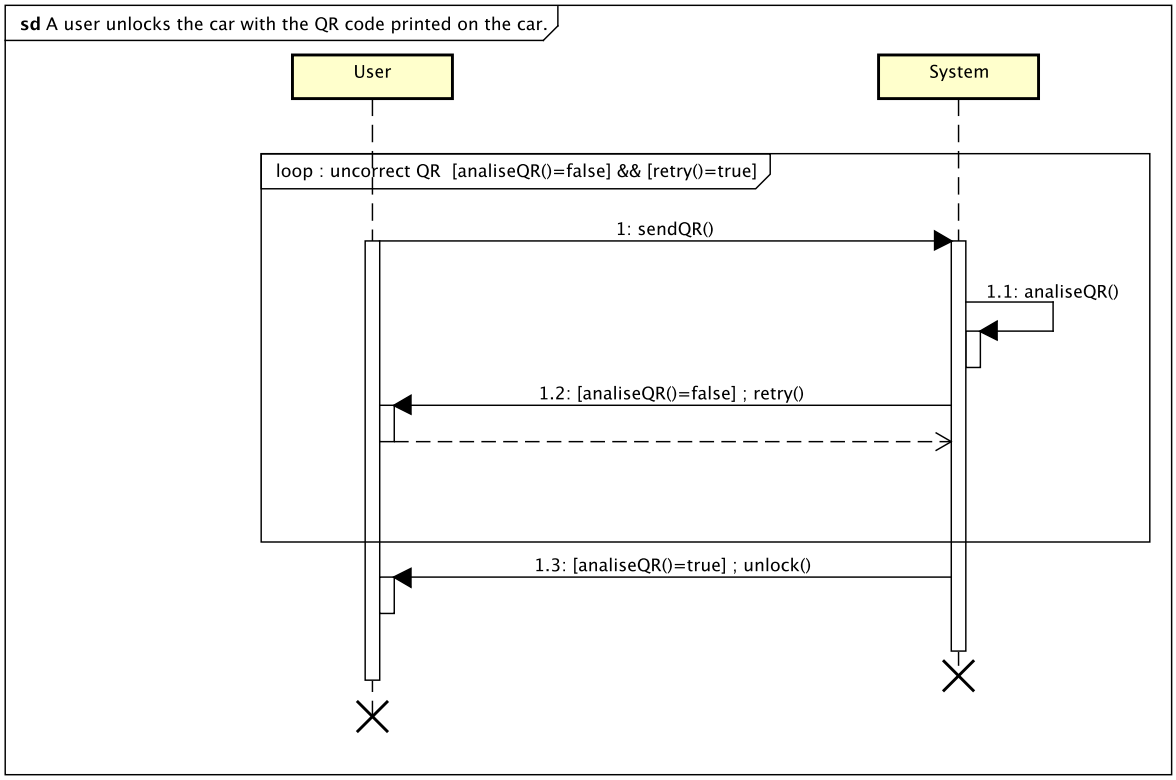
[UC.5] A user cancels a reservation for a car.

Actor	User
Goal	[G5]
Preconditions	The user is logged in, has reserved a car and hasn't unlocked the car yet.
Execution Flow	1. The user goes to the main page of the application. 2. The user presses the "Cancel reservation" button. 3. The user confirms that they want to cancel the reservation. 4. The system marks the car as available again.
Postconditions	The car is available and the user has no pending reservation.
Exceptions	1. The reservation has already expired. The system charges the user a fee of 1 EUR and cancels the reservation.


sd A user cancels a reservation for a car.
User
System
1: cancelReservation()
1.1: checkReservation()
1.2: confirmCancellation()

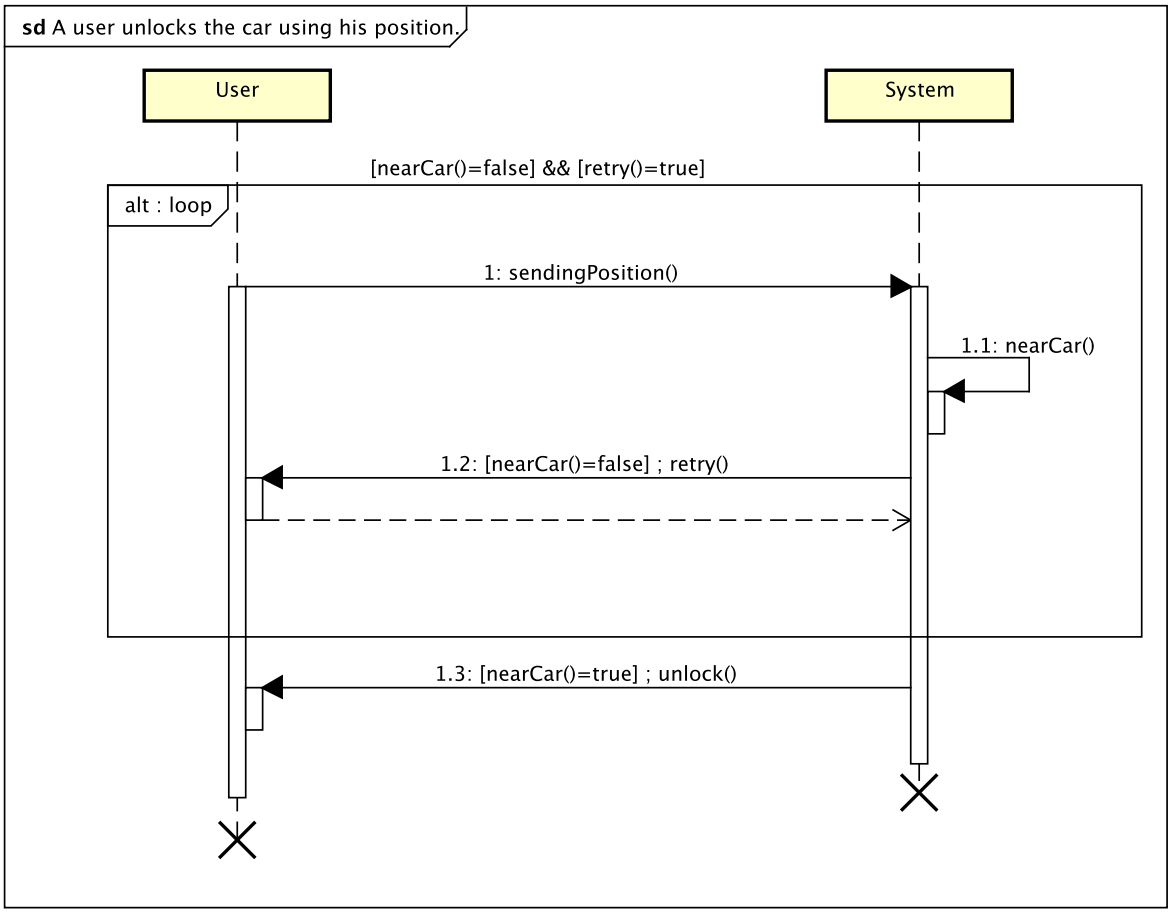
[UC.6a] A user unlocks the car with the QR code printed on the car.

Actor	User
Goal	[G6]
Preconditions	The user is logged in and close to the car he reserved.
Execution Flow	1. The user presses on the camera button, takes a picture of the QR code printed on the car and submits it to the system. 2. The system identifies the car with the QR code and checks the reservation. 3. The system enables the button to unlock the car on the application. 4. The user presses the button. 5. The system unlocks the car.
Postconditions	The car unlocked and ready to be started.
Exceptions	1. The user sends a QR code of a car he didn't reserve.


sd A user unlocks the car with the QR code printed on the car.
User
System
loop : uncorrect QR [analiseQR()=false] && [retry()=true]
1: sendQR()
1.1: analiseQR()
1.2: [analiseQR()=false] ; retry()
1.3: [analiseQR()=true] ; unlock()

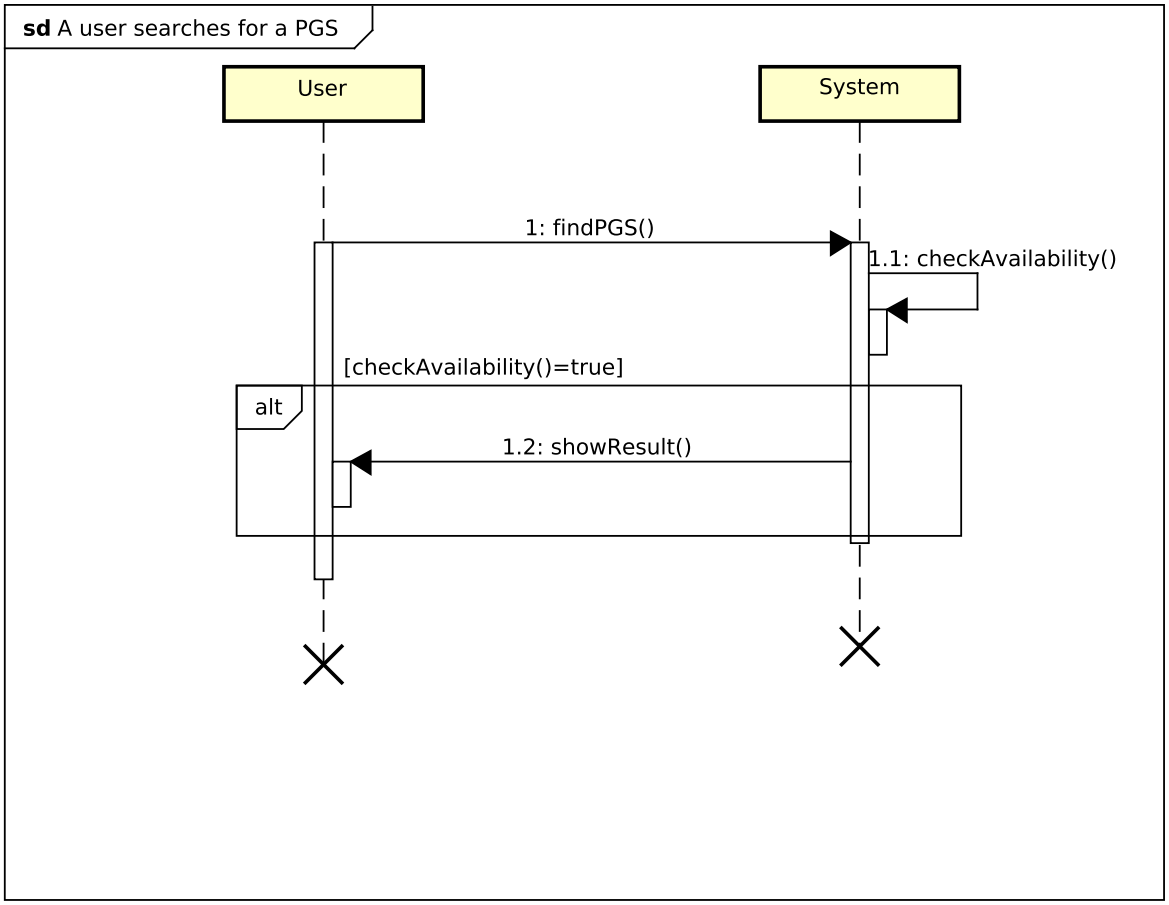
[UC.6b] A user unlocks the car using his position.

Actor	User
Goal	[G6]
Preconditions	The user is close to the car he reserved and has the localization activated.
Execution Flow	1. The user presses on the localization button and sends his position to the system. 2. The system checks if the user's position is close to the car's. 3. The system enables the button to unlock the car on the application. 4. The user presses the button. 5. The system unlocks the car.
Postconditions	The car is unlocked and ready to started.
Exceptions	1. The user is too far from the car he reserved


sd A user unlocks the car using his position.
User
System
[nearCar()=false] && [retry()=true]
alt : loop
1: sendingPosition()
1.1: nearCar()
1.2: [nearCar()=false] ; retry()
1.3: [nearCar()=true] ; unlock()

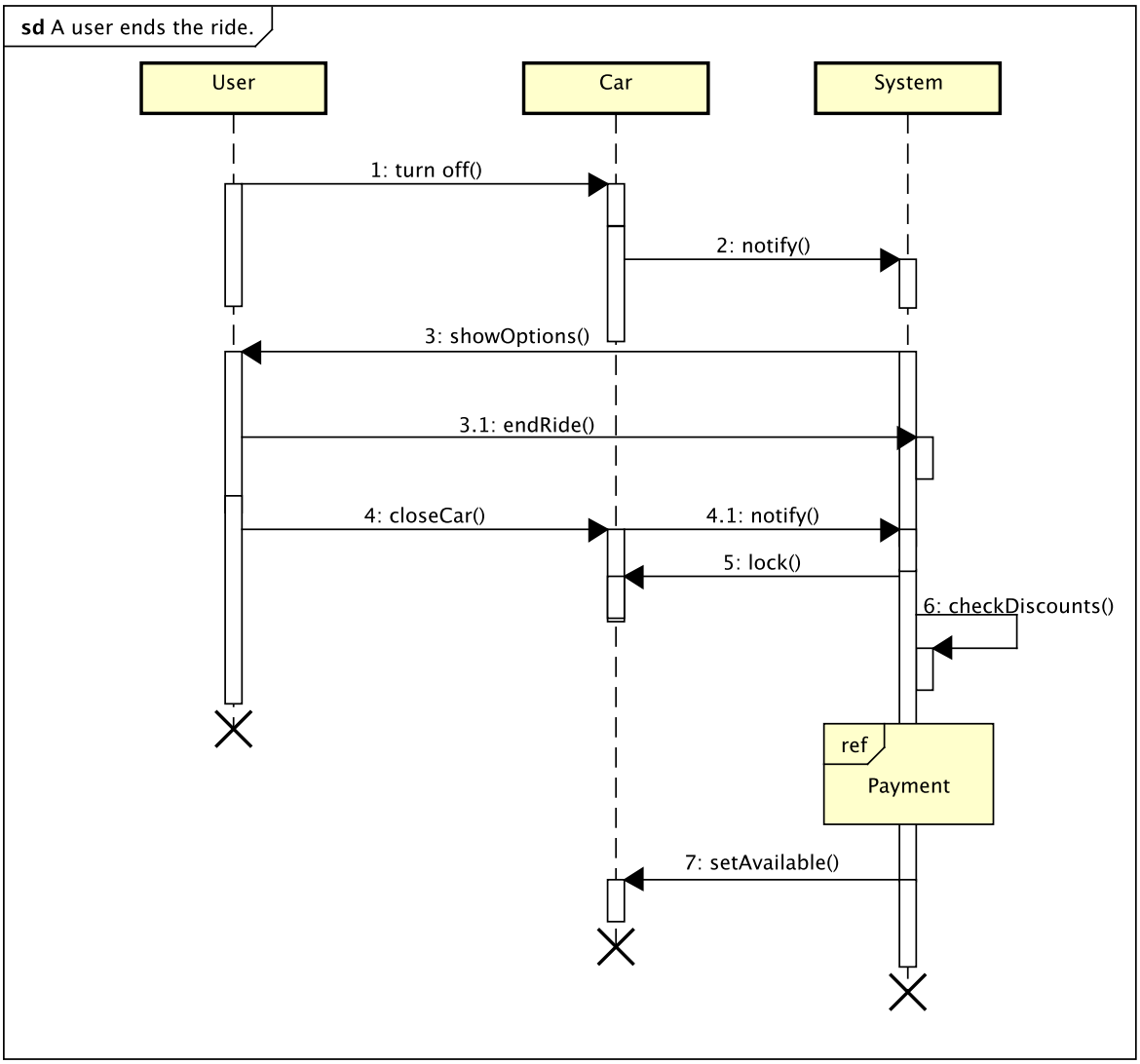
[UC.7] A user searches for a PGS.

Actor	User
Goal	[G21]
Preconditions	The user is inside a car and has access to the car screen.
Execution Flow	1. The user presses on the "find charging stations" button on the display of the car. 2. The user inserts the destination. 3. The system looks in the DB for available PGSs. 4. The system shows to the user the list of PGS in order of distance. 5. By pressing on a PGS shown by the display, the user sets the chosen PGS as destination.
Postconditions	The user knows the PGSs near his destination and has set one as destination.
Exceptions	1. There aren't any available PGSs, the user can try again with a different destination


sd A user searches for a PGS
User
System
1: findPGS()
1.1: checkAvailability()
[checkAvailability()=true]
alt
1.2: showResult()

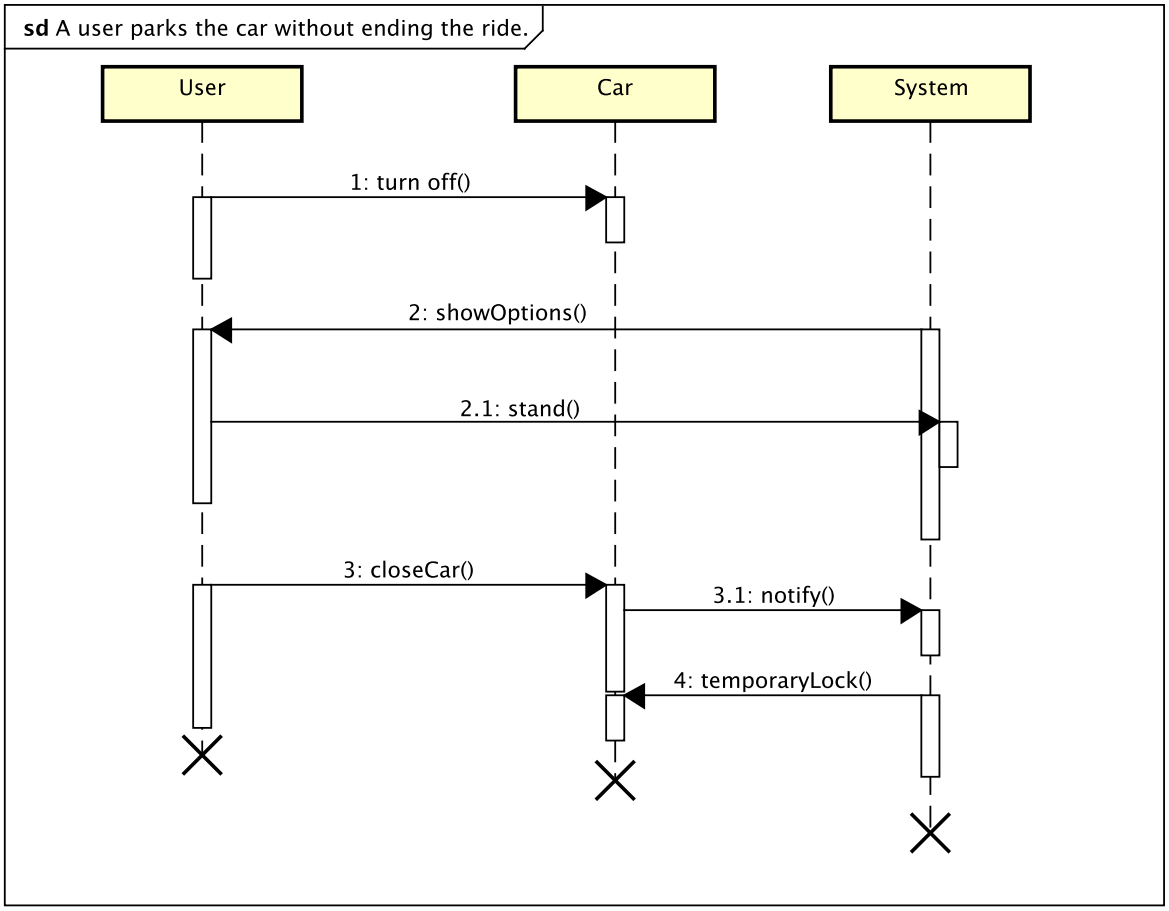
[UC.8] A user ends a ride.

Actor	User
Goal	[G9], [G15], [G16], [G17]
Preconditions	The user is using the car.
Execution Flow	1. If the user is driving he or her stops the car and turns the engine off. 2. The system checks if the car's position corresponds to a safe area. 3. The display shows to the user two options: **a)** to end the ride; **b)** to park the car temporarily. 4. The user presses the button a). 5. The user exits the car and closes the doors. 6. The system locks the car. 7. The system stops charging the user. 8. The system verifies if one or more discount can be applied and in that case applies them. 9. The system sets the car as available. 10. The system carries out the payment transaction.
Postconditions	The car is stopped in a safe area, available and ready to be reserved again.
Exceptions	• The car is not parked in a safe area. In this case the system tells the user the closest safe areas to park in. • The user drives away instead of exiting the car. In this case the process is canceled.


sd A user ends the ride.
User
Car
System
1: turn off()
2: notify()
3: showOptions()
3.1: endRide()
4: closeCar()
4.1: notify()
5: lock()
6: checkDiscounts()
ref
Payment
7: setAvailable()

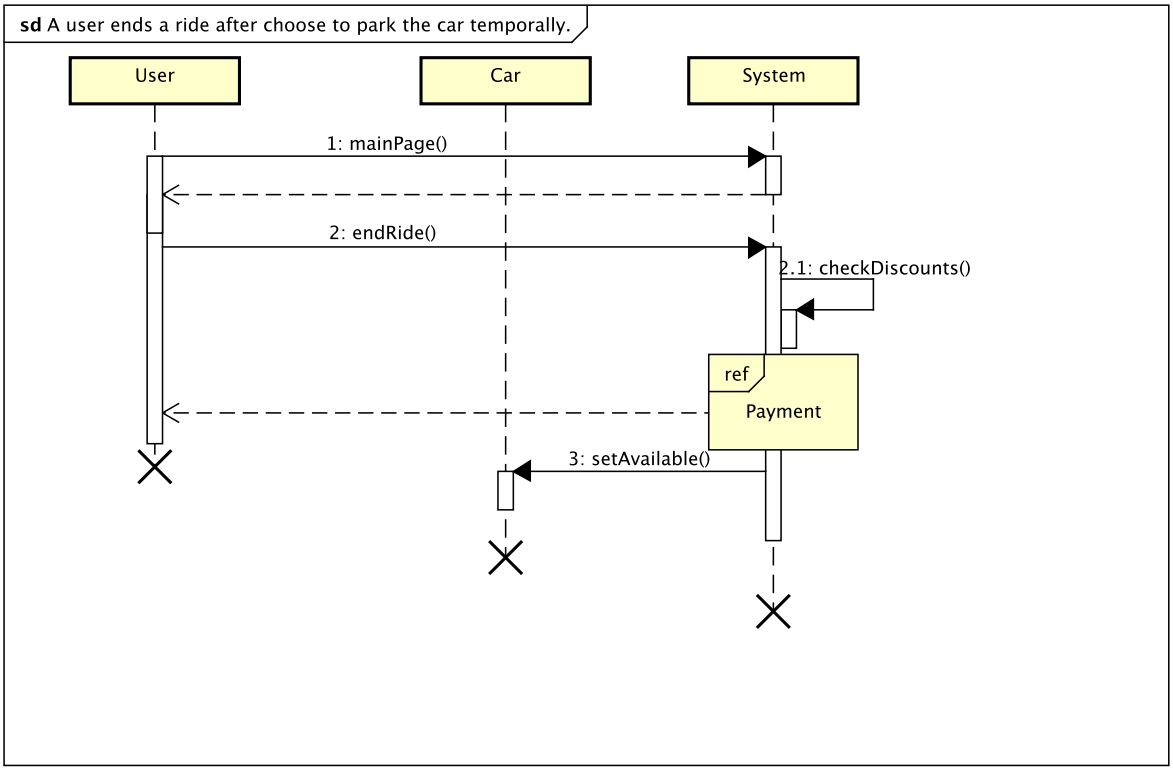
[UC.9] A user parks the car without ending the ride.

Actor	User
Goal	[G10]
Preconditions	The user is using the car.
Execution Flow	1. If the user is driving he or she stops the car and turns the engine off. 2. The system checks if the car's position corresponds to a safe area. 3. The display shows to the user two options: **a)** to end the ride; **b)** to park the car temporarily. 4. The user presses the button b). 5. The user exits the car and closes the doors. 6. The system locks the car automatically.
Postconditions	The car is stopped in a safe area, ready to be unlocked by the same user.
Exceptions	• The car is not parked in a safe area. In this case the system tells the user the closest safe areas to park in. • The user drives away instead of exiting the car. In this case the stop is canceled.


sd A user parks the car without ending the ride.
User
Car
System
1: turn off()
2: showOptions()
2.1: stand()
3: closeCar()
3.1: notify()
4: temporaryLock()

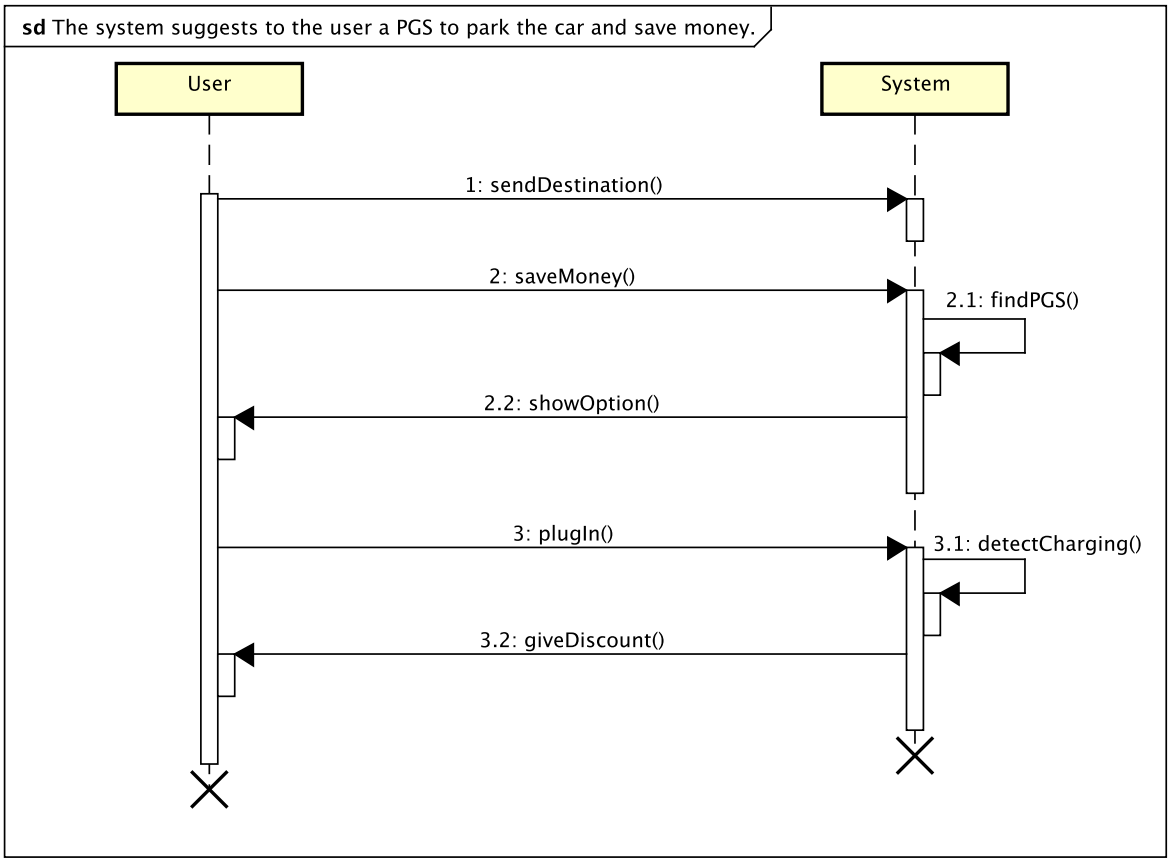
[UC.10] A user ends a ride while the car is temporarily parked.

Actor	User
Goal	[G10], [G15], [G16], [G17]
Preconditions	The user is logged in, and the car is locked and temporarily parked.
Execution Flow	1. The user goes to the main page of the application. 2. The user presses the "End ride" button. 3. The system stops charging the user. 4. The system verifies if one or more discount can be applied and in that case applies them. 5. The system sets the car as available. 6. The system carries out the payment transaction.
Postconditions	The car is stopped in a safe area, ready to be used again.
Exceptions	-


sd A user ends a ride after choose to park the car temporally.
User
Car
System
1: mainPage()
2: endRide()
2.1: checkDiscounts()
ref
Payment
3: setAvailable()

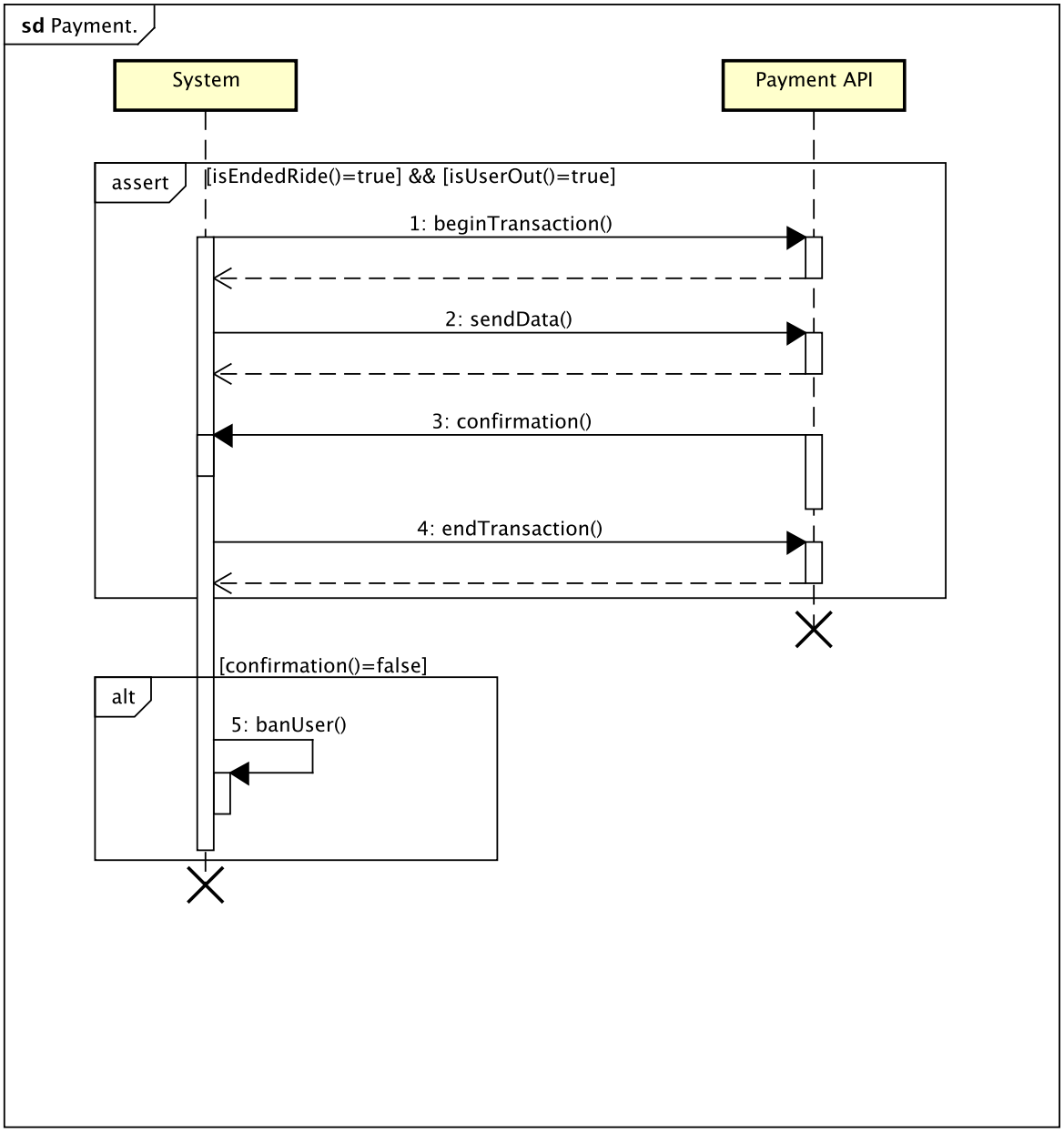
[UC.11] The system suggests to the user a PGS to park the car at and save money.

Actor	System, User
Goal	[G12]
Preconditions	The user is using the car.
Execution Flow	1. The user inserts the destination on the car's navigator. 2. The user chooses the option "Find charging station" 3. The system calculates the best solution, so it searches the nearest PGS to the destination keeping in mind a uniform distribution of the cars. 4. The system suggests the PGS to the user through the display. 5. The user drives to the PGS, parks the car and turns the engine off. 6. The user plugs the car in the power grid. 7. The user chooses from the car's display to end the ride. 8. The system detects that the car's position corresponds to the right PGS and that the car is plugged in. 9. The system applies a discount on the amount the user has to pay.
Postconditions	The car is ready to be locked.
Exceptions	1. There isn't any available PGS near the user's destination. 2. The user doesn't park the car at the PGS suggested by the system. 3. The user doesn't plug the car in the power grid. 4. The user drives away after choosing to end the ride. The discount is canceled.


sd The system suggests to the user a PGS to park the car and save money.
User
System
1: sendDestination()
2: saveMoney()
2.1: findPGS()
2.2: showOption()
3: plugIn()
3.1: detectCharging()
3.2: giveDiscount()

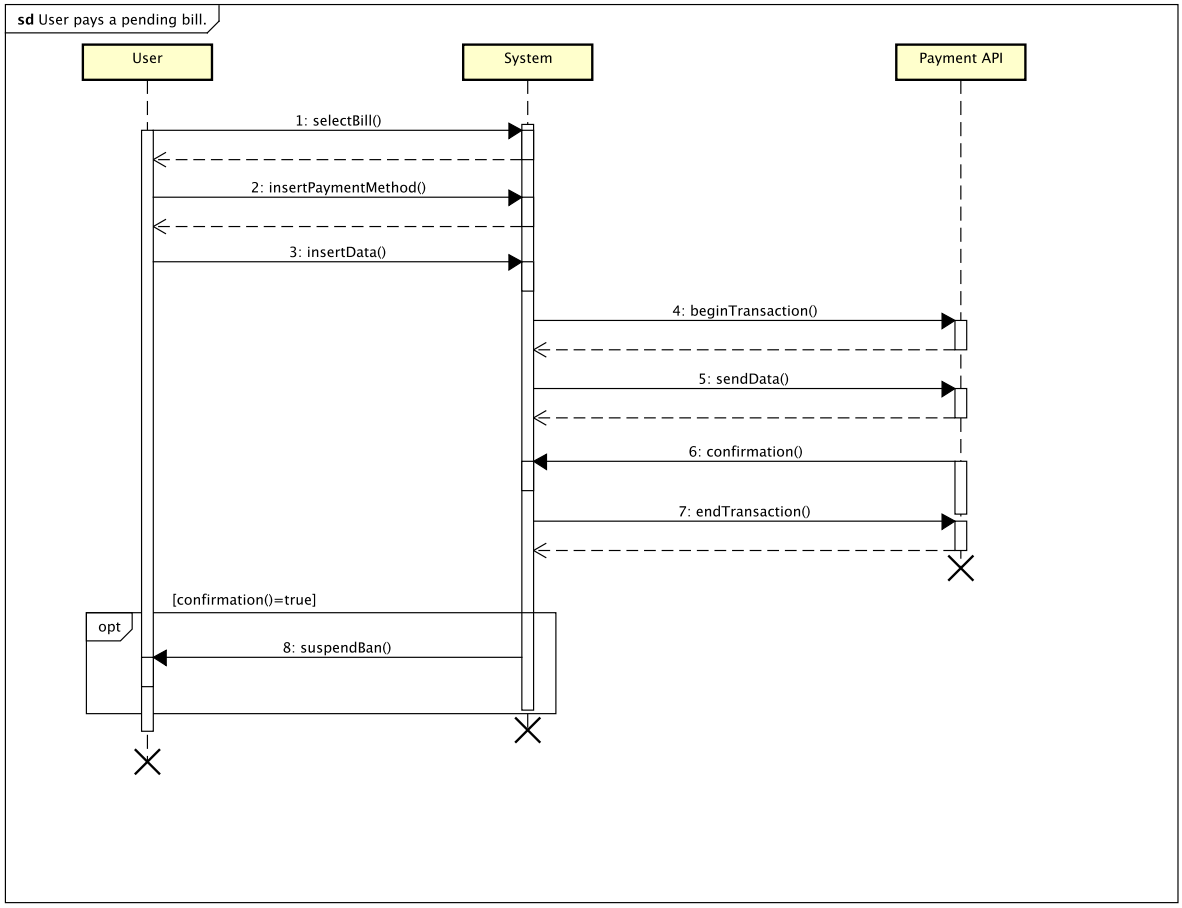
[UC.12] Payment

Actor	System, Payment API
Goal	[G9], [G19]
Preconditions	The user ended a ride and has a pending charge.
Execution Flow	1. The system starts the transaction. 2. The system sends its credentials to the Payment API. 3. The system sends the user's account data to the API. 4. The system sends the amount to receive to the API. 5. The API confirms the payment. 6. The system closes the transaction.
Postconditions	The bill is paid
Exceptions	1. The Payment API is not available. In this case, the system marks the bill as "waiting" and retries later. 2. The transaction is rejected. In this case, the system suspends the user until he pays the outstanding bill.


sd Payment.
System
Payment API
assert
[isEndedRide()=true] && [isUserOut()=true]
1: beginTransaction()
2: sendData()
3: confirmation()
4: endTransaction()
[confirmation()=false]
alt
5: banUser()

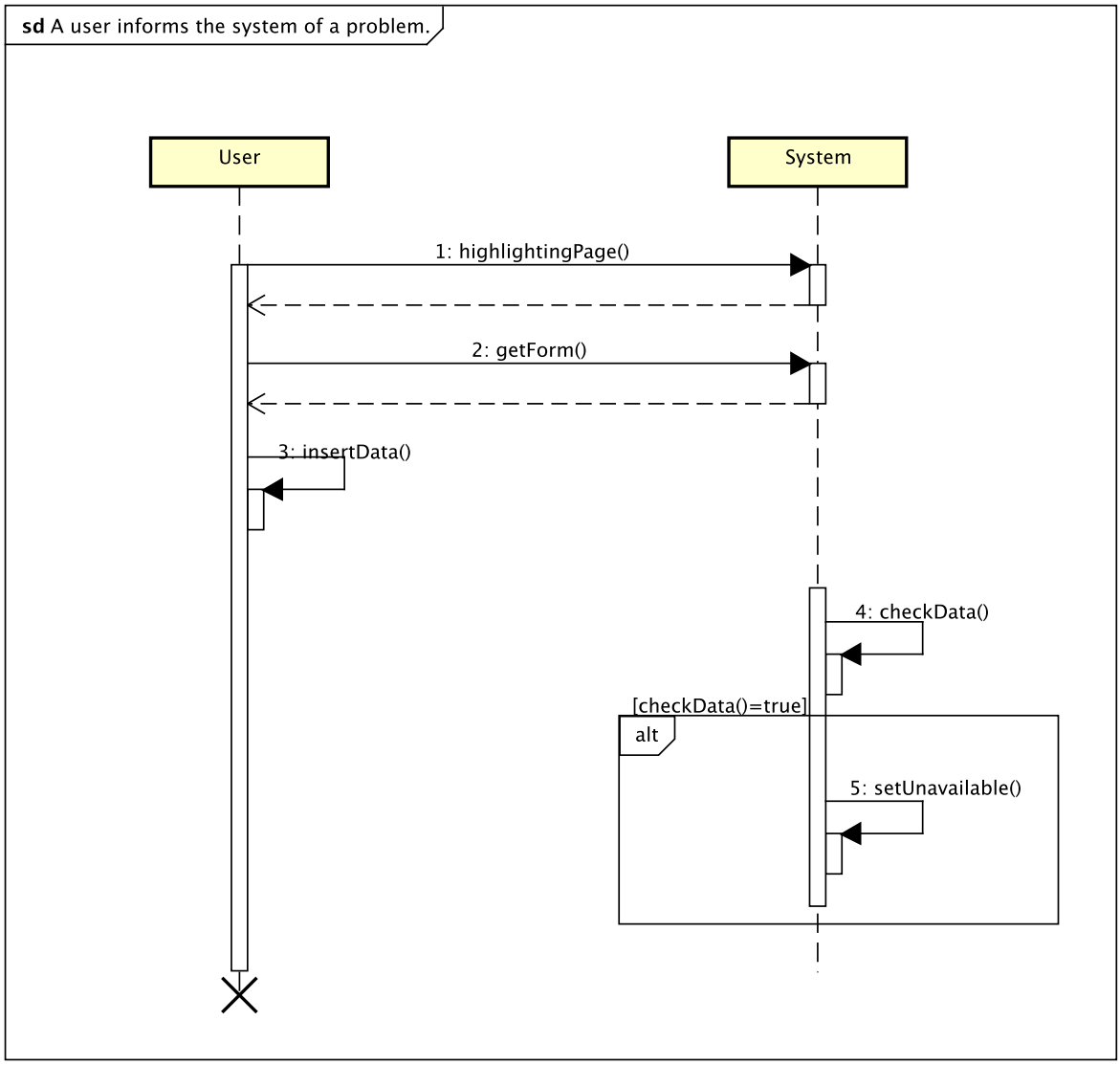
[UC.13] A user pays an outstanding bill.

Actor	User, System, Payment API
Goal	[G18]
Preconditions	The user is logged in, has a outstanding bill and is suspended.
Execution Flow	1. The user goes to his profile page. 2. The user selects the outstanding bill. 3. The user selects the payment method. 4. The user confirms the transaction. 5. The system carries out the payment as in the "Payment" use case. 6. The API confirms the payment. 7. The system cancels the suspension.
Postconditions	The bill is paid and the user is no longer suspended.
Exceptions	1. The Payment API is not available. In this case, the system marks the bill as "waiting" and retries later. 2. The transaction is rejected. In this case, the user is notified of the problem and remains suspended.


sd User pays a pending bill.
User
System
Payment API
1: selectBill()
2: insertPaymentMethod()
3: insertData()
4: beginTransaction()
5: sendData()
6: confirmation()
7: endTransaction()
[confirmation()=true]
opt
8: suspendBan()

[UC.14] A user reports a problem to the system.

Actor	User, System
Goal	[G20]
Preconditions	The user is logged in and has reserved a car
Execution Flow	1. The user goes to the reporting page of the application. 2. The user selects the type of problem the car has (dirty, damaged, ...). 3. The user confirms and submits the data to the system. 4. The system checks the data inserted by the user. 5. The system sets the car as unavailable. 6. The system ends the user's reservation and ride.
Postconditions	The car is marked as unavailable and the user has no reservation.
Exceptions	1. The user inserts wrong data.


sd A user informs the system of a problem.
User
System
1: highlightingPage()
2: getForm()
3: insertData()
4: checkData()
[checkData()=true]
alt
5: setUnavailable()

3.3 Performance Requirements

- *Response Time*: we want that our software will satisfy the 100% of requests in maximum 0.5 seconds. For this reason, the software, in order to guarantee a good grade of usability, has to be very reactive and we assume that the user's perception is bounded to his connection and his device.

- *Workload*: we assume that the workload will be distributed following the profile shown in figure 12.

  We assume that our software will be able to manage a large number of connection - we estimates the the peak load could be about 3000 connections - between users and the system.

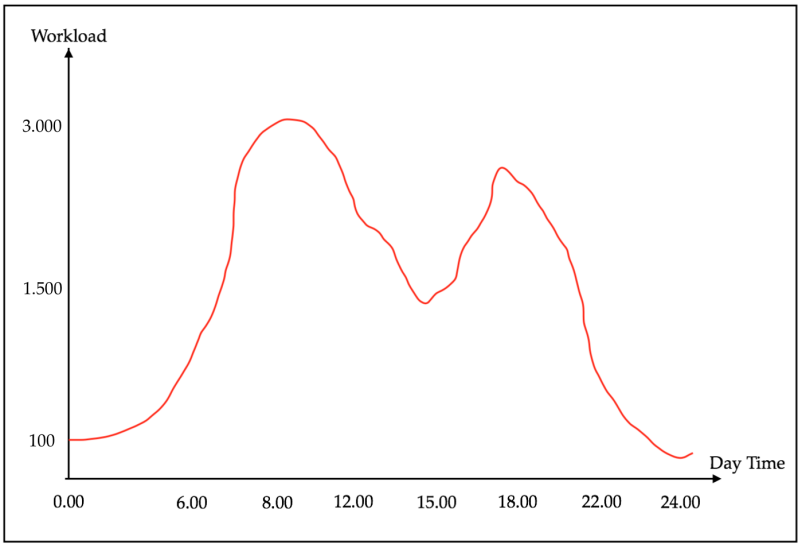


Figure 12: Expected workload by time of day

3.4 Software System Attributes

- *User Friendliness*: The user interface has to be as simple and intuitive as possible as the application will also be used by users that could feel uncomfortable with the technology.
- *Reliability*: we assume that our software will give the right result for at least 99% of the requests.
- *Availability*: we assume that our software will be available with a rate of 99%. In order to ensure that rate, patches will deployed during the night and time of low traffic.
- *Security*:
  - every message exchanged between the system and the user will be protected by using the TLS protocol.
  - sensitive data like names, driving license IDs and so on will be encrypted on the database, and passwords will be hashed using the SHA-256 algorithm.
- *Maintainability*: the software must be accompanied by a complete documentation to let other developers correct, improve the code or integrate new functions more easily.
- *Portability*: server-side, the software must be developed so that it can be executed on most commercial servers; client-side the software must be compatible with iOS, Android and Windows Phone regarding the mobile application, while the web app must be compatible with at least the most used browser, i.e. Chrome, Firefox, Opera, Safari, Internet Explorer.

4 Appendix

4.1 Alloy Model

We developed an Alloy model following the requirements we specified to check if they were consistent. This section contains the entire source code of our model and a picture of the world it generated. The original file can be found in the same directory where this document was published, with the name `'powerenjoy_alloy.als'`.

```
open util/integer as integer

//--- SIG ---

// Geographical position (simplified in this model
// using a pair of Int instead of Float)
sig Position{
    lat: one Int,
    lon: one Int
}

// Container of ∀ positions recognized by our system
sig Map{
    area : some Position
}

/*
The set of safe areas for parking cars is predefined
by the management system.
Areas can have any shape: we'll consider them
as a predefined set of Positions.
We assume there is at least one safe area.
*/

abstract sig SafeArea extends Map{}

sig ParkingArea extends SafeArea{}

abstract sig PlugStatus{}
sig P_Reserved, P_Available, P_InUse extends PlugStatus{}
// reserved if user has a MSO enabled

sig Plug{
    status : one PlugStatus,
    pluggedCar : lone Car
} {
    status = P_InUse implies pluggedCar ≠ none
    pluggedCar ≠ none implies status = P_InUse
}

```

```
// We assume that charging stations in our model ∀ have at
    least one working plug
sig ChargingStation extends SafeArea{
    plugs : some Plug
}{
    #(area) = #(plugs)
    ∀ c : Car • c in plugs.pluggedCar implies c.pos in area
}

fact {
    no disjoint c1, c2 : ChargingStation • some p : Plug • p in
        c1.plugs and p in c2.plugs
}

sig Plate{}

sig Seat{}

abstract sig CarStatus{}
sig Available, Charging, Reserved extends CarStatus{}
sig Used, TemporaryBreak extends Reserved{}

abstract sig BatteryLevel{}
sig MoreThanHalf, ChargeNeeded /*≤20%*/ , OtherBatteryLevel
    extends BatteryLevel{}

abstract sig MotorStatus{}
sig On, Off extends MotorStatus{}

abstract sig CarDoors{}
sig Locked, Unlocked extends CarDoors{}

sig Car{
    plate : one Plate,
    battery : one BatteryLevel,
    status: one CarStatus,
    doors: one CarDoors,
    pos : one Position,
    motor: one MotorStatus,
    seats : set Seat
}{
    #seats ≥ 2
    (doors = Locked or status = Charging or status =
        TemporaryBreak  ) implies motor = Off
    (status = Charging or status = Available) implies doors =
        Locked
    status = Available implies pos in SafeArea.area
    motor = On implies doors = Unlocked
}
```

```
one sig Company{
    cars: set Car,
    parkingAreas : set ParkingArea,
    stations : set ChargingStation,
    users : set User,
    costPerMinute : one Int,
}{
    costPerMinute > 0
}

sig ReservationCode{}

sig Licence{}


sig User{
    licenceNumber : one Licence,
    curRes : lone Reservation,
    ride : lone Ride,
    bills : set Bill
} {
    curRes ≠ none implies ride = none
    ride ≠ none implies curRes = none
}

sig Minute{}

sig Reservation{
    code : one ReservationCode,
    user : one User,
    car : one Car,
    minutesLeft : set Minute
}{
    #minutesLeft < 60
    car.battery ≠ ChargeNeeded
    car.doors = Locked
    car.motor = Off
}

sig Passenger{}

abstract sig RideStatus{}
sig RideCompleted, Riding extends RideStatus{}

sig Ride {
    code : one ReservationCode,
    status : one RideStatus,
    car : one Car,
    user : one User,
    ridingMinutes: some Minute,
```

```
136     passengers: set Passenger,
        MSO : lone ChargingStation
138     // If MoneySavingOption is enabled it specified the chosen
            ChargingStation
    }{
140     #passengers ≤ #(car.seats) − 1
        MSO ≠ none implies {
142         one p : MSO.plugs • p.status = P_Reserved
        }
144 }

146
    abstract sig BillStatus{}
148 sig Paid, Rejected extends BillStatus{}

150 sig Bill{
        ride : one Ride,
152     status : one BillStatus,
        amount : one Int
154 }{

156     (!getTenPercentDiscount[ride] and !getTwentyPercentDiscount
            [ride] and !getThirtyPercentDiscount[ride] and !
            getThirtyPercentMore[ride] and !
            getTwentyPercentDiscountMSO[ride]) implies amount = (
            mul[Company.costPerMinute, #(ride.ridingMinutes)])
        (getTenPercentDiscount[ride] and !getTwentyPercentDiscount[
            ride] and !getThirtyPercentDiscount[ride] and !
            getThirtyPercentMore[ride] and !
            getTwentyPercentDiscountMSO[ride]) implies amount = (
            mul[Company.costPerMinute, #(ride.ridingMinutes)]).
            applyDiscount[10]
158     (!getTenPercentDiscount[ride] and getTwentyPercentDiscount[
            ride] and !getThirtyPercentDiscount[ride] and !
            getThirtyPercentMore[ride] and !
            getTwentyPercentDiscountMSO[ride]) implies amount = (
            mul[Company.costPerMinute, #(ride.ridingMinutes)]).
            applyDiscount[20]
        ((!getTenPercentDiscount[ride] and !
            getTwentyPercentDiscount[ride] and
            getThirtyPercentDiscount[ride] and !
            getThirtyPercentMore[ride] and !
            getTwentyPercentDiscountMSO[ride]) or (
            getTenPercentDiscount[ride] and
            getTwentyPercentDiscount[ride] and !
            getThirtyPercentDiscount[ride] and !
            getThirtyPercentMore[ride] and !
            getTwentyPercentDiscountMSO[ride])) implies amount = (
            mul[Company.costPerMinute, #(ride.ridingMinutes)]).
            applyDiscount[30]
```

```
160     (!getTenPercentDiscount[ride] and !getTwentyPercentDiscount
            [ride] and getThirtyPercentDiscount[ride] and !
            getThirtyPercentMore[ride] and !
            getTwentyPercentDiscountMSO[ride]) implies amount = (
            mul[Company.costPerMinute, #(ride.ridingMinutes)]).
            applyDiscount[40]
        (!getTenPercentDiscount[ride] and getTwentyPercentDiscount[
            ride] and getThirtyPercentDiscount[ride] and !
            getThirtyPercentMore[ride] and !
            getTwentyPercentDiscountMSO[ride]) implies amount = (
            mul[Company.costPerMinute, #(ride.ridingMinutes)]).
            applyDiscount[50]
162     (getTenPercentDiscount[ride] and getTwentyPercentDiscount[
            ride] and getThirtyPercentDiscount[ride] and !
            getThirtyPercentMore[ride] and !
            getTwentyPercentDiscountMSO[ride]) implies amount = (
            mul[Company.costPerMinute, #(ride.ridingMinutes)]).
            applyDiscount[60]
        (!getTenPercentDiscount[ride] and !getTwentyPercentDiscount
            [ride] and !getThirtyPercentDiscount[ride] and
            getThirtyPercentMore[ride] and !
            getTwentyPercentDiscountMSO[ride]) implies amount = (
            mul[Company.costPerMinute, #(ride.ridingMinutes)]).
            applyDiscount[-30]
164     (getTenPercentDiscount[ride] and !getTwentyPercentDiscount[
            ride] and !getThirtyPercentDiscount[ride] and
            getThirtyPercentMore[ride] and !
            getTwentyPercentDiscountMSO[ride]) implies amount = (
            mul[Company.costPerMinute, #(ride.ridingMinutes)]).
            applyDiscount[-20]
        (!getTenPercentDiscount[ride] and getTwentyPercentDiscount[
            ride] and !getThirtyPercentDiscount[ride] and
            getThirtyPercentMore[ride] and !
            getTwentyPercentDiscountMSO[ride]) implies amount = (
            mul[Company.costPerMinute, #(ride.ridingMinutes)]).
            applyDiscount[-10]
166     (getTenPercentDiscount[ride] and !getTwentyPercentDiscount[
            ride] and !getThirtyPercentDiscount[ride] and !
            getThirtyPercentMore[ride] and
            getTwentyPercentDiscountMSO[ride]) implies amount = (
            mul[Company.costPerMinute, #(ride.ridingMinutes)]).
            applyDiscount[30]
        (!getTenPercentDiscount[ride] and getTwentyPercentDiscount[
            ride] and !getThirtyPercentDiscount[ride] and !
            getThirtyPercentMore[ride] and
            getTwentyPercentDiscountMSO[ride]) implies amount = (
            mul[Company.costPerMinute, #(ride.ridingMinutes)]).
            applyDiscount[40]
168     (getTenPercentDiscount[ride] and getTwentyPercentDiscount[
            ride] and !getThirtyPercentDiscount[ride] and !
```

```
            getThirtyPercentMore[ride] and
            getTwentyPercentDiscountMSO[ride]) implies amount = (
            mul[Company.costPerMinute, #(ride.ridingMinutes)]).
            applyDiscount[50]
        (!getTenPercentDiscount[ride] and !getTwentyPercentDiscount
            [ride] and getThirtyPercentDiscount[ride] and !
            getThirtyPercentMore[ride] and
            getTwentyPercentDiscountMSO[ride]) implies amount = (
            mul[Company.costPerMinute, #(ride.ridingMinutes)]).
            applyDiscount[50]
170     (getTenPercentDiscount[ride] and !getTwentyPercentDiscount[
            ride] and getThirtyPercentDiscount[ride] and !
            getThirtyPercentMore[ride] and
            getTwentyPercentDiscountMSO[ride]) implies amount = (
            mul[Company.costPerMinute, #(ride.ridingMinutes)]).
            applyDiscount[60]
        (!getTenPercentDiscount[ride] and getTwentyPercentDiscount[
            ride] and getThirtyPercentDiscount[ride] and !
            getThirtyPercentMore[ride] and
            getTwentyPercentDiscountMSO[ride]) implies amount = (
            mul[Company.costPerMinute, #(ride.ridingMinutes)]).
            applyDiscount[70]
172     (getTenPercentDiscount[ride] and getTwentyPercentDiscount[
            ride] and getThirtyPercentDiscount[ride] and !
            getThirtyPercentMore[ride] and
            getTwentyPercentDiscountMSO[ride]) implies amount = (
            mul[Company.costPerMinute, #(ride.ridingMinutes)]).
            applyDiscount[80]

174     ride.status = RideCompleted

176 }

178 //--- FACT ---

180 fact {
        ∀ CS : CarStatus • CS in Car.status
182 }

184 fact {
        ∀ BL : BatteryLevel •BL in Car.battery
186 }

188 fact {
        ∀ r : ReservationCode • r in Reservation.code or r in Ride.
            code
190 }

192 fact {
        ∀ b : BillStatus • b in Bill.status
```

```
194 }

196 fact {
        ∀ p : Passenger • p in Ride.passengers
198 }

200 fact {
        ∀ p : PlugStatus • p in Plug.status
202 }

204 fact {
        ∀ ms : MotorStatus • ms in Car.motor
206 }

208 fact {
        ∀ cd : CarDoors • cd in Car.doors
210 }

212 fact {
        ∀ m : Minute • m in Ride.ridingMinutes or m in Reservation.
            minutesLeft
214 }

216 //--- USER ---

218 fact licenceNumberIsUnique {
        no disjoint u1, u2: User  •  u1.licenceNumber = u2.
            licenceNumber
220 }

222 //--- POSITION ---
    fact eachPositionPairIsUnique{
224     no disjoint p1,p2 : Position• p1.lat=p2.lat and p1.lon=p2.
            lon
    }
226
    //--- PLATE ---
228 fact eachPlateIsUnique{
        no disjoint c1, c2 : Car • c1.plate = c2.plate
230 }

232 fact noUnusedPlate{
        ∀ p : Plate • one c : Car • p = c.plate
234 }

236 //--- RESERVATION ---

238 fact reservationCodeIsUnique {
        no disjoint a, b : Reservation •  a.code = b.code
240 }
```

```
242 fact OneReservationPerUser{
        no disjoint r1, r2 : Reservation • r1.user = r2.user
244 }

246 fact OneCarPerReservation{
        no disjoint r1, r2 : Reservation • r1.car = r2.car
248 }

250 fact OnlyUsersBookCars{
        ∀ r : Reservation • r.user in Company.users
252 }

254 fact {
        ∀ u : User • u.curRes ≠ none implies u.curRes.user = u
256 }

258 //—— RIDE ——

260 fact reservationCodeIsUnique2 {
        no disjoint r1, r2 : Ride •  r1.code = r2.code
262 }

264 fact OneRidePerUser{
        no disjoint r1, r2 : Ride • r1.user = r2.user
266 }

268 fact OneCarPerRide{
        no disjoint r1, r2 : Ride • r1.car = r2.car and r1.status =
            Riding and r2.status = Riding
270 }

272 fact OnlyUsersBookCars2{
        ∀ r : Ride • r.user in Company.users
274 }

276 fact {
        ∀ u : User • u.ride ≠ none implies u.ride.user=u
278 }

280 //—— CHARGING STATION ——

282 fact chargingCarsAreInAStation{
        ∀ c : Car •  c.status = Charging <=> c in ChargingStation.
            plugs.pluggedCar
284 }

286 fact carsArePluggedInOnlyOneStation {
        no cs1, cs2 : ChargingStation, c : Car • c=cs1.plugs.
            pluggedCar and c=cs2.plugs.pluggedCar
```

```
288 }

290 fact onlyLockedCarCanBeRecharged {
        no c : Car,  cs :  ChargingStation • c.doors = Unlocked and
            c in cs.plugs.pluggedCar
292 }

294 fact{
        ∀ p : Plug • one cs : ChargingStation • p in cs.plugs
296 }

298 fact {
        ∀ p : Plug • p in ChargingStation.plugs
300 }

302 fact{
        no p : Plug • p.status = P_InUse and p.pluggedCar= none
304 }

306 fact{
        ∀ c : Car• one cs : ChargingStation • c.status = Charging
            implies c.pos in cs.area
308 }

310 //--- CAR ---

312 fact carAvailability{
        ∀ c : Car • c.status = Available implies (c.battery ≠
            ChargeNeeded and c.doors = Locked and c.motor = Off)
314 }

316 fact carsUnlockedOnlyForARide{
        ∀ c : Car• c.doors = Unlocked implies one r : Ride• r.car =
            c
318 }

320 //--- COMPANY ---

322 fact companyCars{
        ∀ c : Car • c in Company.cars
324 }

326 fact {
        ∀ cs :  ChargingStation  • cs in Company.stations
328 }

330 fact {
        ∀ u :  User  • u in Company.users
332 }
```

```
fact {
    ∀ p :  ParkingArea • p in Company.parkingAreas
}

//--- BILL ---

fact  noReservationOrRideUntillBillIsPaid{
    ∀ u : User, b : Bill • ((u.curRes ≠ none or u.ride ≠ none)
       and b in u.bills) implies b.status = Paid
}

fact {
    ∀ b : Bill • b in User.bills
}

fact {
    no disjoint u1, u2 : User • u1.bills & u2.bills ≠ none
}

fact reservationCodeIsUnique3 {
    no disjoint b1, b2 : Bill •  b1.ride.code = b2.ride.code
}

//--- MISC ---

fact{
    ∀ rs : RideStatus • rs in Ride.status
}

fact {
    ∀ b : Bill, u: User • b in u.bills implies b.ride.user=u
}

fact {
    ∀ r : Ride • one b : Bill • r. status = RideCompleted
       implies b.ride=r
}

fact {
    no disjoint c1, c2 : Car • c1.seats & c2.seats ≠ none
}

fact{
    ∀ c : Car • one p : Plug • ( c.status = Charging implies (p
       .pluggedCar = c and p.status= P_InUse))
}

fact {
    ∀ l : Licence • l in User.licenceNumber
}
```

```
382 fact {
        ∀ p : Plug • p.status = P_Reserved implies (one r : Ride •
            r.MSO ≠ none and r.car.status = Reserved and r.status =
             Riding)
384 }

386 //
    ------------------------------------------------------------------
     FUN
    ------------------------------------------------------------------

388 fun Int.applyDiscount[discount : Int] : one Int{
        //  this-amount *discount   /100
390     div[sub[this, mul[this, discount]],100]
    }
392
    fun Position.squaredDistance[p : Position]: one Int {
394     mul[p.lat-this.lat, p.lat-this.lat]+mul[p.lon-this.lon,p.
            lon-this.lon]
    }
396
    // Returns the Position of the nearest Charging Station
398 fun nearestCS[p : Position] : one Position {
        {pCS : ChargingStation.area •
400         ∀ p2 :  ChargingStation.area • pCS.squaredDistance[p] ≤
                p2.squaredDistance[p]
        }
402 }

404 //--- PRED ---

406 pred getTenPercentDiscount[r : Ride]{
        (#r.passengers) ≥ 2
408 }

410 pred getTwentyPercentDiscount[r : Ride]{
        r.car.battery = MoreThanHalf
412 }

414 pred getThirtyPercentDiscount[r : Ride]{
        r.car.status = Charging
416 }

418 pred getThirtyPercentMore[r : Ride]{
        r.car.battery = ChargeNeeded or  nearestCS[r.car.pos].
            squaredDistance[r.car.pos] > 9000000
420 }

422 pred getTwentyPercentDiscountMSO[r : Ride]{
```

```
        r.MSO ≠ none and r.car.pos in r.MSO.area and r.car in r.MSO
            .plugs.pluggedCar
424 }

426

    pred show(){
428     /*#(Company.cars) >1
        #(Company.users) >1
430     #(Riding) >1
        #(Company.stations) > 1
432     #(Company.parkingAreas) > 1
        //#(P_InUse) = 0
434     #(P_Available) > 0
        #(P_Reserved) > 0*/
436     #(Plug.pluggedCar) > 1

438 }

440 run show for 4
```

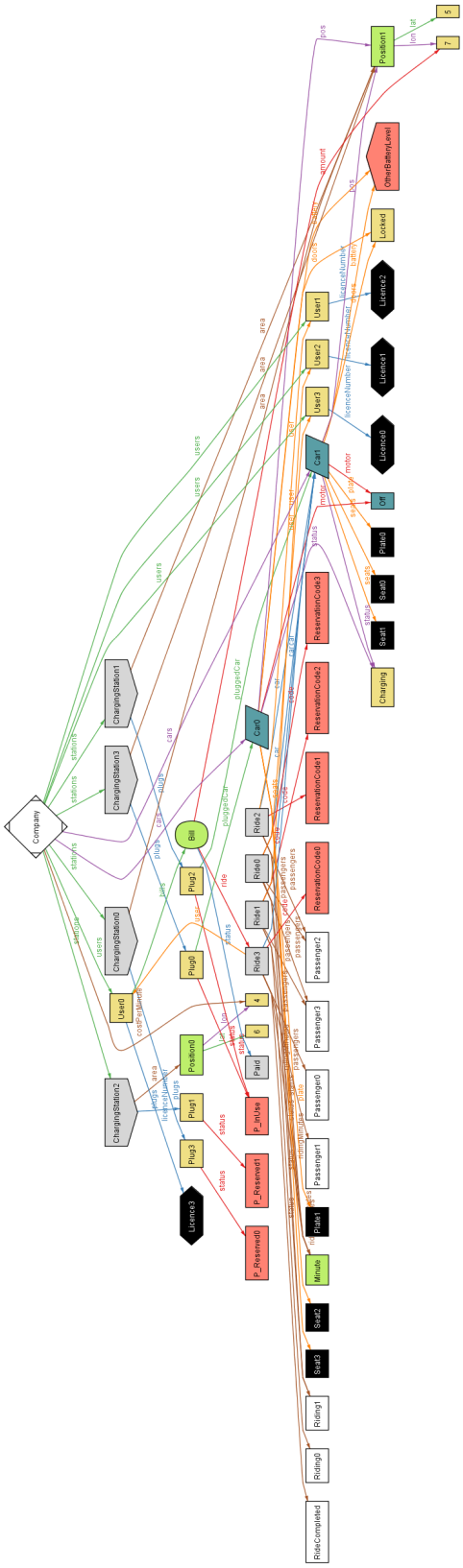


Figure 13: The world we generated with Alloy

4.2 Software and tools used

- Balsamiq for mockups.
- Astah professional for UML diagrams.
- GitHub for version control.
- TEXStudio as a LATEXeditor.
- Google Docs for the draft.

4.3 Hours of work

- Pietro Ferretti: 45 hours
- Nicole Gervasoni: 45 hours
- Danilo Labanca: 45 hours

5 Revisions

5.1 Changelog

- RASD v1.0, published on November 13, 2016
- RASD v1.1, published on December 11, 2016
  - Added goal G21
  - Added use case UC.7
  - Added GraphHopper API
  - Improved english grammar