Requirement Analysis and Specification Document

Version 2.0
7 February 2017

Authors:
Patrizia Porati, Tommaso Sardelli

Contents

1 Introduction

1.1 Purpose

This is the Requirement Analysis and Specification Document (RASD from now on). The aim of this document is to show the functional and non-functional requirements of the system-to-be, based on several important aspects: the needs expressed by the stakeholders, the constraints which it is subject to, the typical scenarios that will happen after its deployment. The targeted audience is mainly made of software engineers and developers who have to actually develop the service here described. We want to make clear from the beginning that in this document we are not going to discuss what will be implemented or how. We are just going to collect and analyze all the customer requirements and to provide a general idea of how the product should look in the end and how the users should interact with it.

1.2 Scope

The task we are asked to complete is the definition of a software system to manage a car sharing service composed by electric cars. This is going to be a brand new system without any legacy software or data to deal with. The idea is that users can register to our platform using an internet connected device (computer, smartphone, etc.) and then they are able to look for available cars in a certain area. One of the available cars can be reserved and picked up in not more then one hour; at that point the user is billed for the time he's using the car. Also, the system can apply discounts or penalties. Finally, administrators shall interact with the platform in order to manage issues about cars, safe areas and users. They could also appoint new administrators.

1.3 Stakeholders

The main stakeholder is the **PowerEnJoy** company, whose aim is to provide a service that is profitable for them but at the same time is useful for all the people living in the city and helps reduce the pollution thanks to the electric engines.

1.4 Definitions

- **Agents**
  - **Guest**: A person who is not registered to the platform. He can either register or browse the public website
  - **RegisteredUser**: A registered person that has full access to the platform.

  - **User**: See “RegisteredUser”.
  - **Passenger**: A person that is taken by a RegisteredUser as his passenger during a ride. It doesn’t matter if such person is registered or not.
  - **Administrator**: A person who works for the “PowerEnJoy” and interact with the platform in order to manage issues about cars, safe areas and users.
  - **Admin**: See “Administrator”.

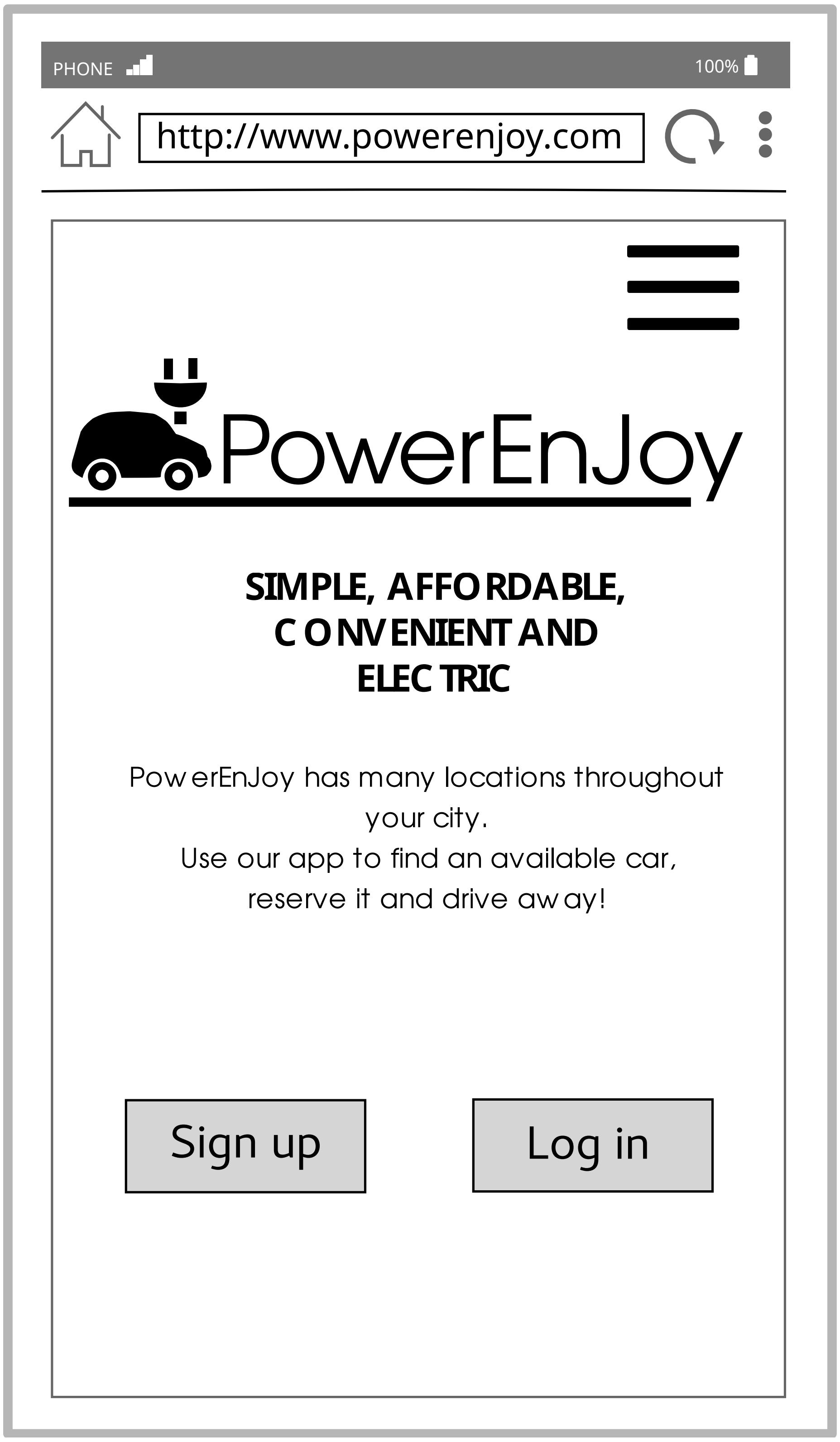
- **Car**: An electric car owned by PowerEnJoy.
- **Reservation**: A one hour lasting booking of a car performed by a single user.
- **Ride**: A ride is what follows a reservation when the user picks up the car in time. It begins with the car unlocking, it ends when the car is parked and locked and it keeps track of the user who drove the car and the time of car usage.
- **SafeArea**: A PowerEnJoy parking slot where the User can leave the car at the end of the ride.
- **PowerGridStation**: A special SafeArea with an electric outlet to charge the car battery.

1.5 Product perspective

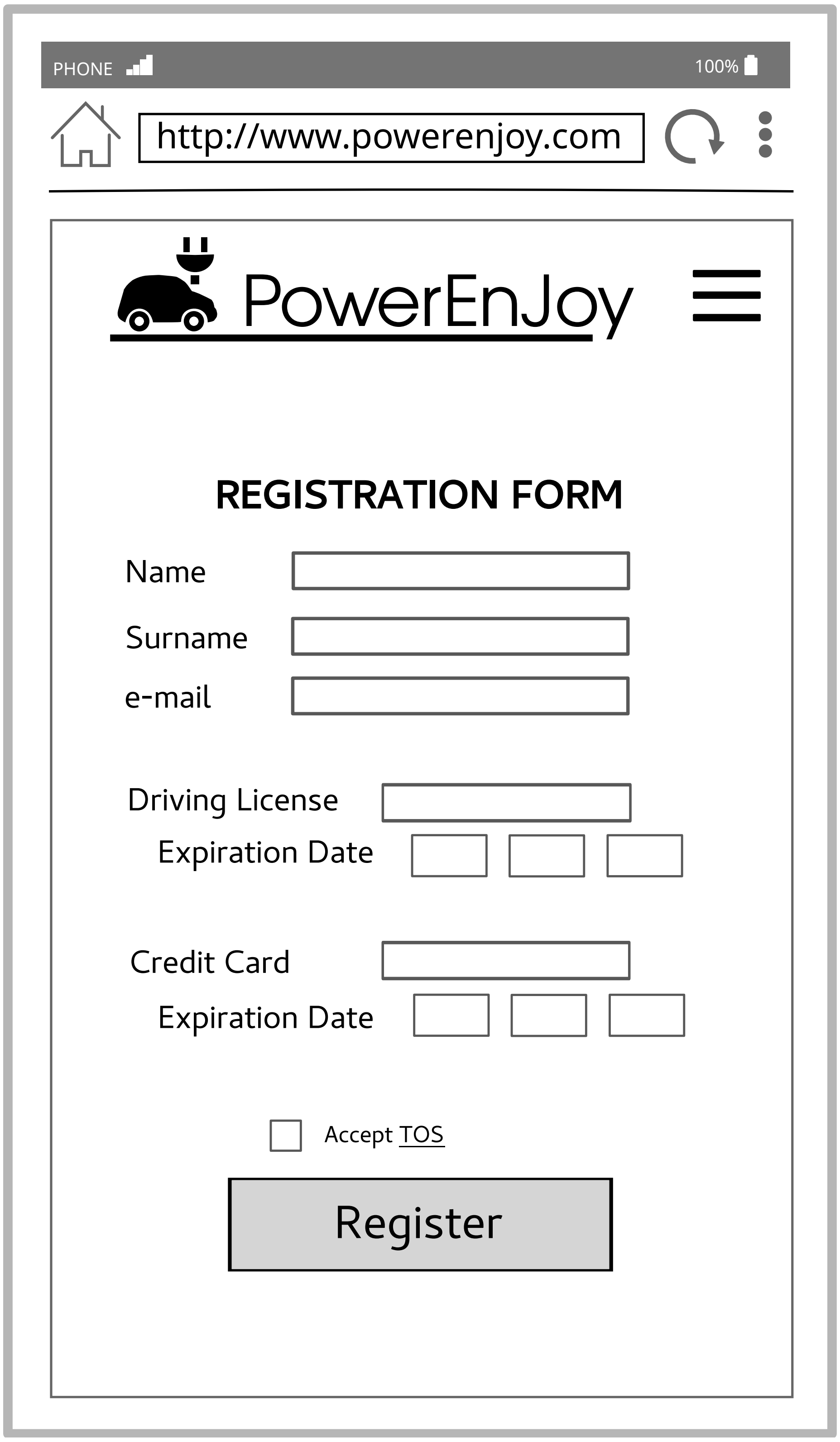
As we said in the introduction of this document, we are not going to discuss the possibile implementations that could be adopted to solve the customer needings. Nontheless we think is important to provide some sketches showing how we think the user experience should be. Specifically, we want users to be able to access the platform from a mobile device and to easly look for and reserve cars near the their location. For such purpose we show some mockups of a possibile application interface.

1.5.1 User interface

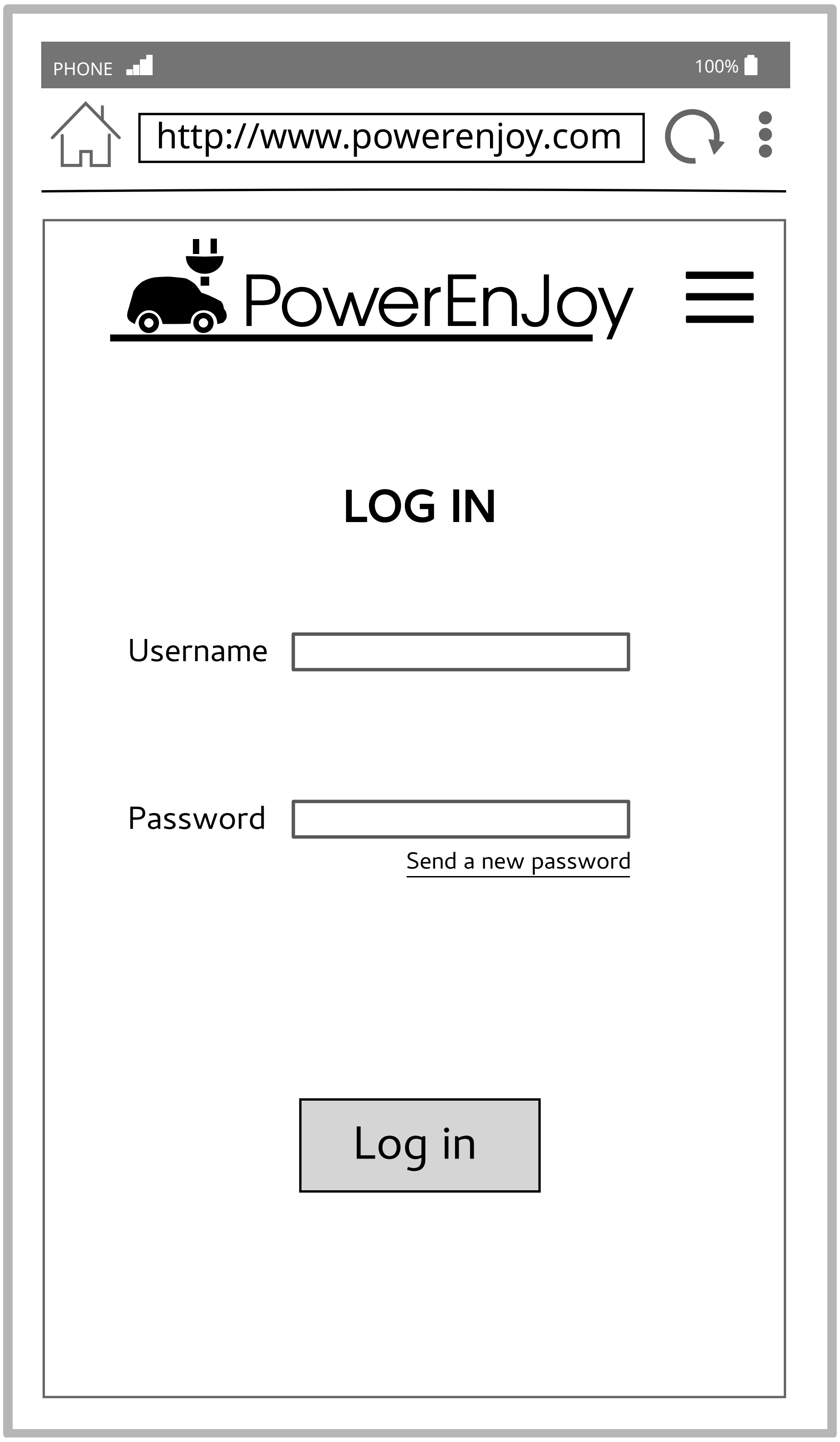
We are going to use a webapp interface as a sample of how the interaction should happen. Additionally we will show an example of the car screen displaying some useful informations.


PHONE
100%
http://www.powerenjoy.com
PowerEnJoy
SIMPLE, AFFORDABLE, CONVENIENT AND ELECTRIC
PowerEnJoy has many locations throughout your city.
Use our app to find an available car, reserve it and drive away!
Sign up
Log in

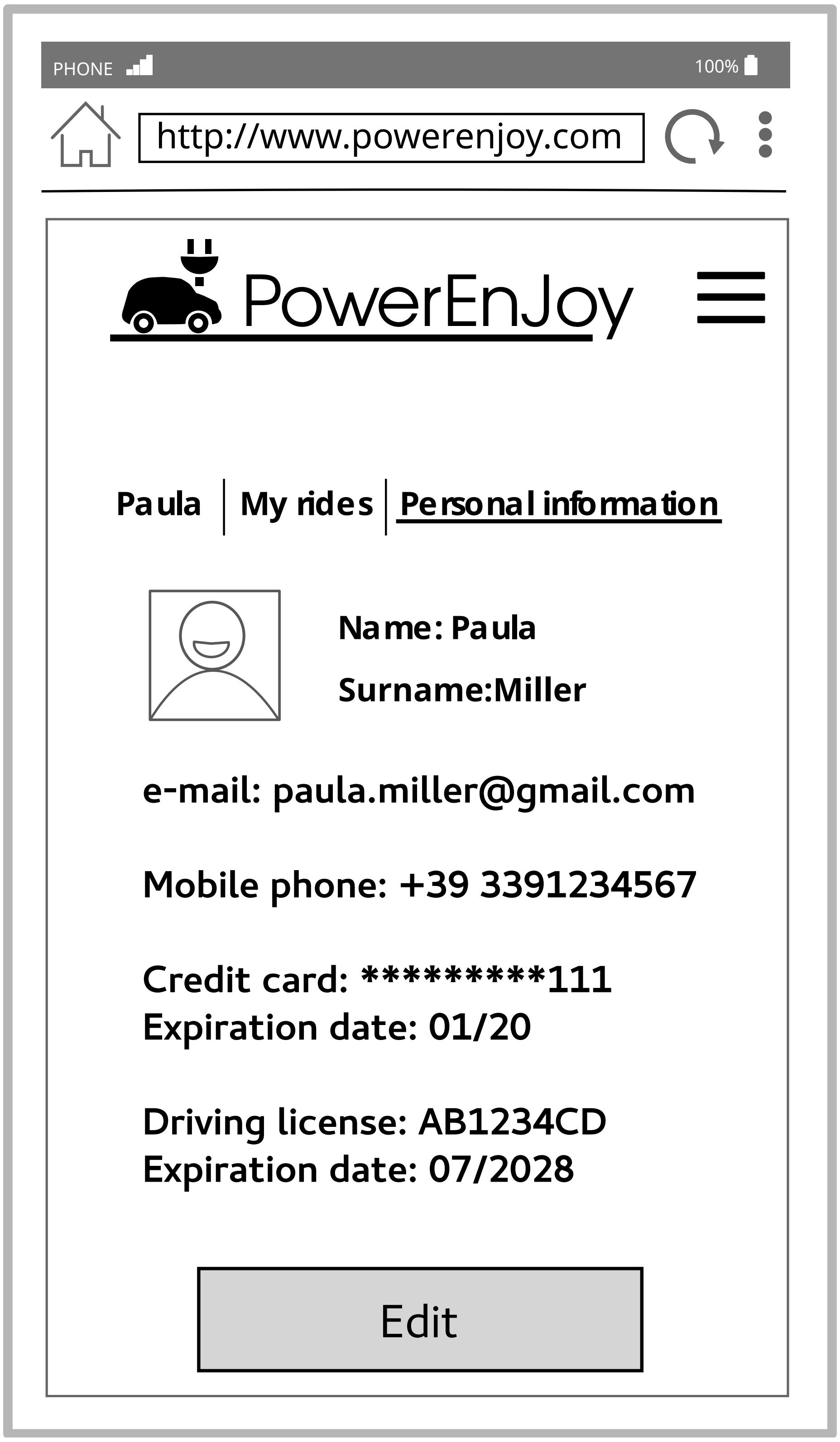
Mockup 1: The home page.



Mockup 2: The registration page for a guest willing to become a customer.



Mockup 3: The login screen for a customer.



Name: Paula

Surname:Miller

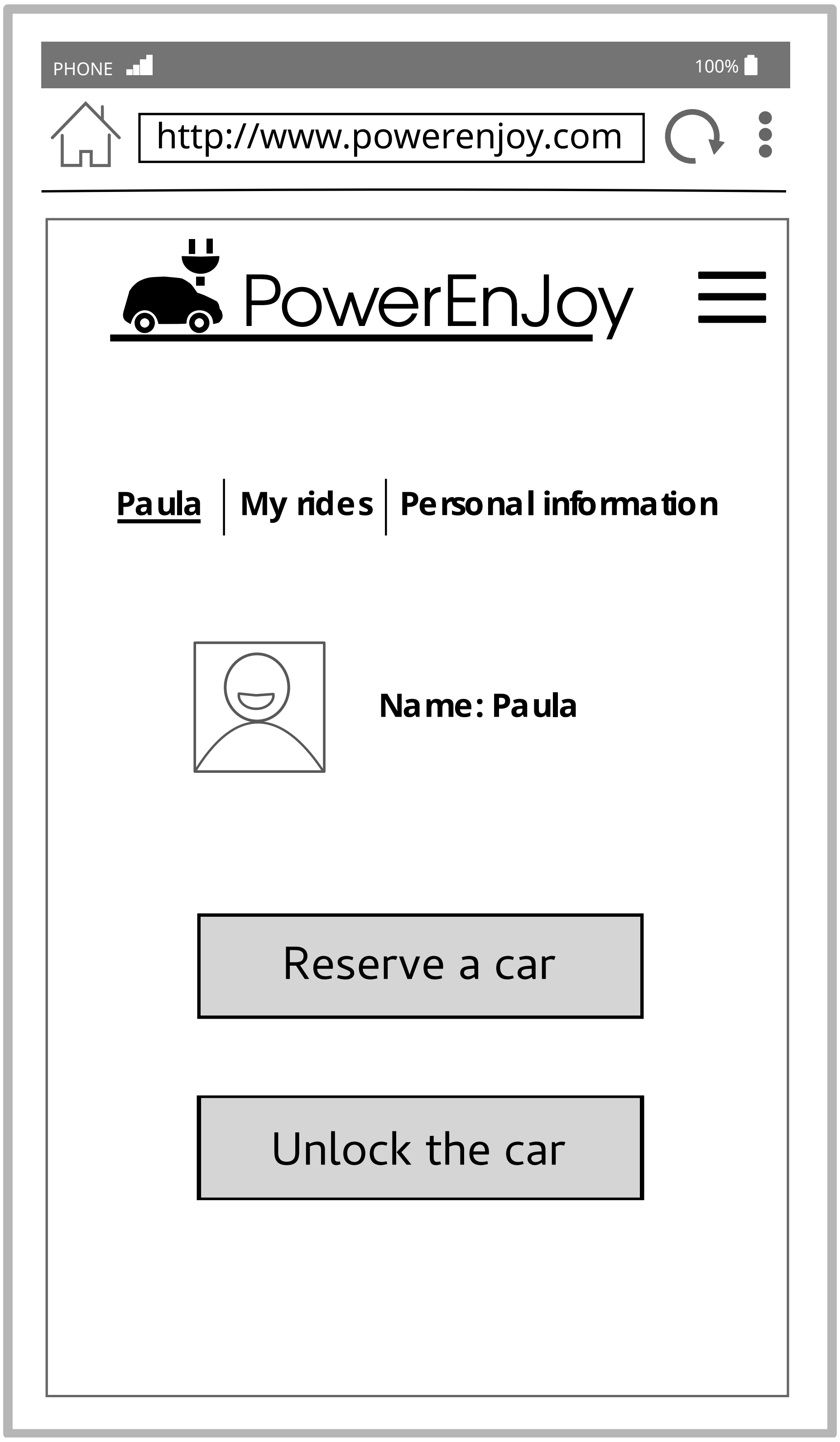
e-mail: paula.miller@gmail.com

Mobile phone: +39 3391234567

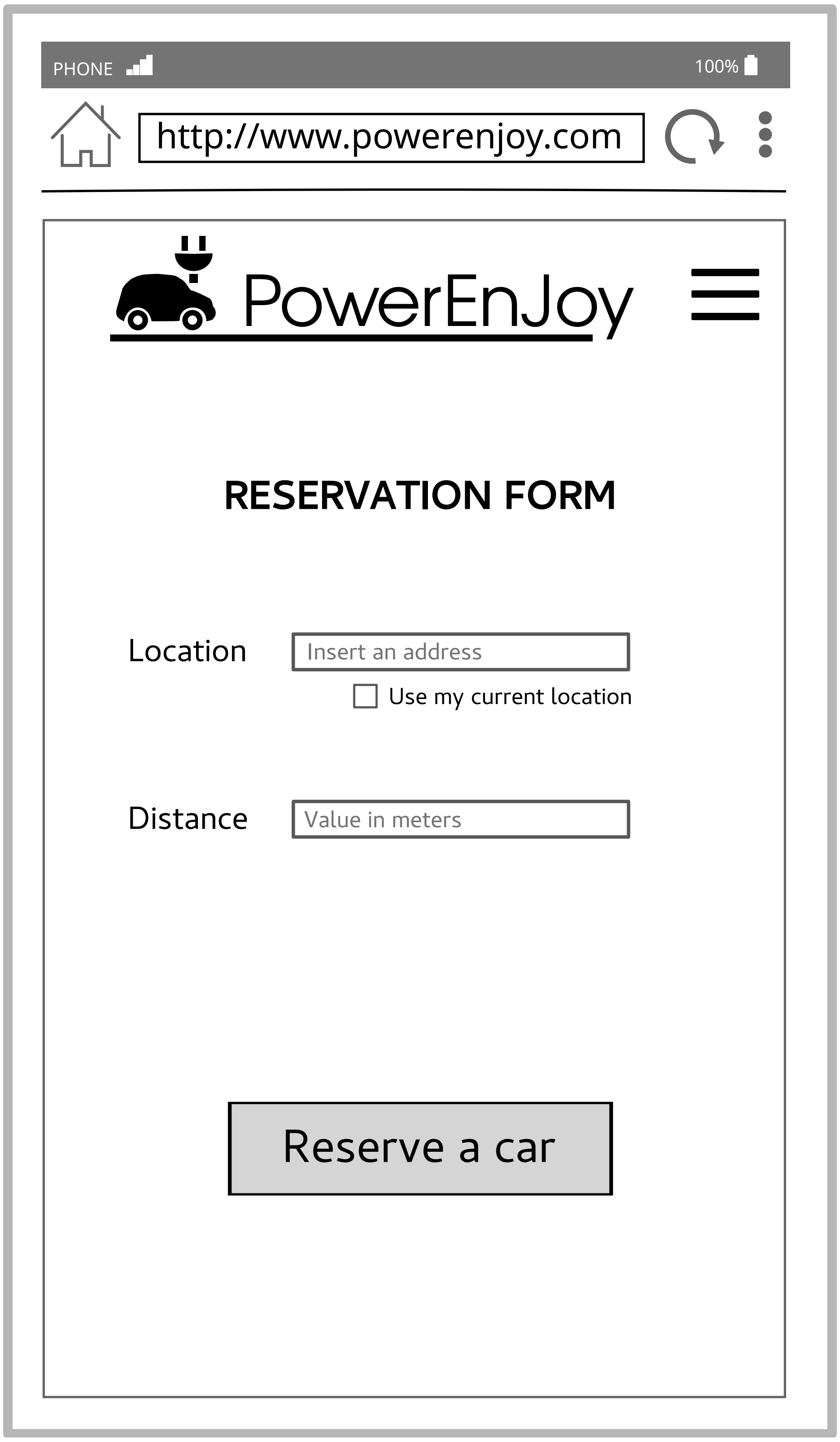
Credit card: **********111
Expiration date: 01/20

Driving license: AB1234CD
Expiration date: 07/2028

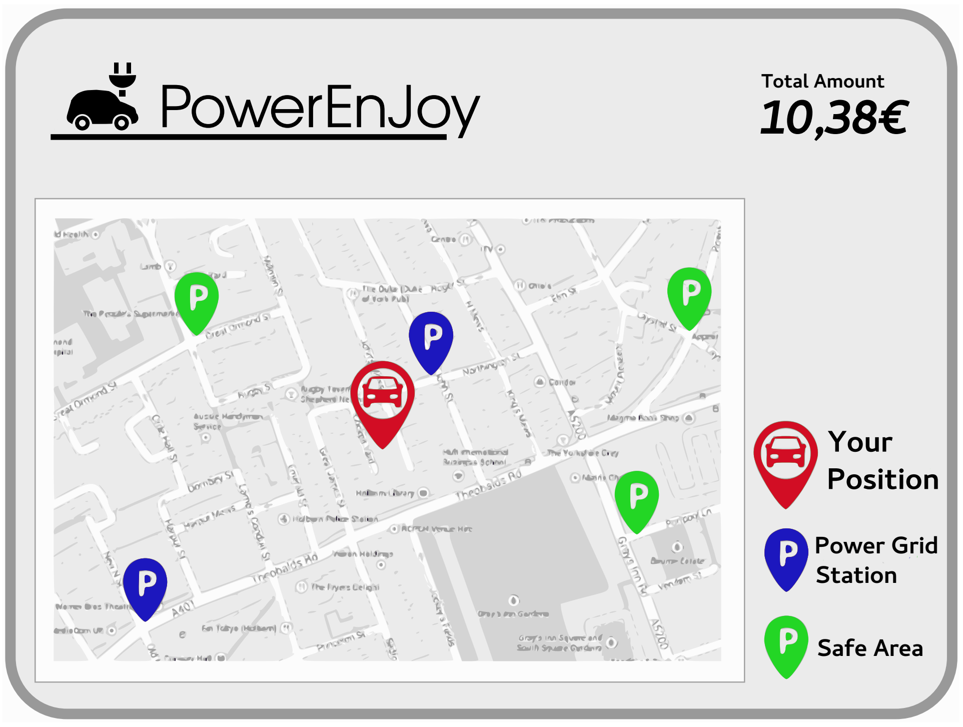
Mockup 4: A page displaying the user information.



Mockup 5: Personal profile page.



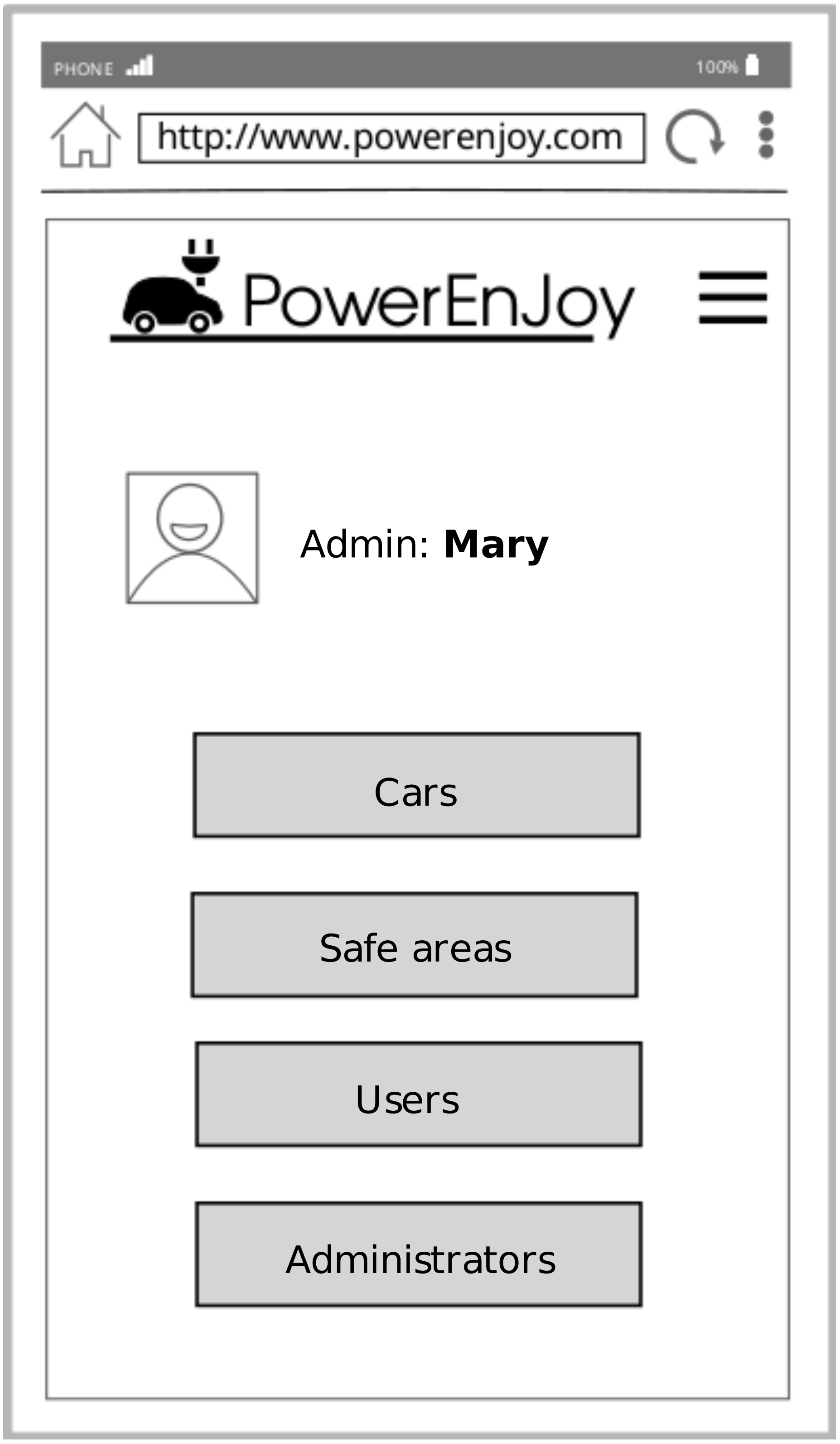
Mockup 6: Insertion of information to perform a car research.



Mockup 7: The car screen displaying information about nearby safe areas and power grid stations.

1.5.2 Administrator interface

Here is a sample of the webapp interface of an admin. The administrator can manage cars, safe areas, users and administrators.



Mockup 8: Administrator page.

2 Requirements

2.1 Geneal Assumptions

This are some geneal assumptions that we considered to be always valid when defining our model. They are listed in this generic list beacause they are essential to fulfill the majority of the goals described in the following section. For this reason we preferred to list all of them here rather then repeat them multiple times.

- All the cars and the users' devices are equipped with a GPS system
- The GPS system provides accurate and correct informations
- The devices used by the users have a mobile internet connection
- Every car is connected to a network (internet or local) and we can send or retrieve informations in any moment
- We can control some physical devices of the car remotely, like the locking system

2.2 Functional Requirements

Goal 1: Guests must be able to register to the platform receiving back a login password.

> **Requirement 1**: The system shall validate any input by the guest.
>
> **Requirement 2**: The system shall send a login password to the user who has just signed up in less than 5 minutes.

Goal 2: Registered users must be able to search available cars.

> **Requirement 1**: The system shall allow users to choose a maximum radius for the car research.
>
> **Requirement 2**: The system shall allow the user to provide an address or the current position as the center of the research area.
>
> **Requirement 3**: Before the reservation, the system shall check that the user's status is 'active'.
>
> **Domain Assumption 1**: The user position is provided by the user's device GPS system.

Goal 3: Registered users must be able to reserve a single car among all the available cars.

Requirement 1: The system shall verify that the user is not reserving more than one car at a time.

Requirement 2: The system shall verify that the user can select a car only among the list of cars marked as “available” in the search radius.

Requirement 3: The system shall change the status of the car from “available” to “reserved” once the car is selected.

Requirement 4: The system shall keep track of the time elapsed as soon as the reservation is completed.

Requirement 5: Before the reservation, the system shall check that the user’s status is ‘active’.

Goal 4: Car reservation expires after one hour and a fee is charged to the user.

Requirement 1: The system shall change the status of the car from “reserved” to “available” if the car is not picked up within one hour from the reservation.

Requirement 2: The system shall impose a charge of 1€ to the user for the reservation expiration.

Domain Assumption 1: Payments are withdrew automatically from the user’s credit card without any user interaction.

Domain Assumption 2: The system doesn’t manage payments directly since we rely on an external service who is exposing some specific APIs.

Domain Assumption 3: We consider that the user has picked up the car when the car is unlocked.

Goal 5: The user must be able to unlock the reserved car.

Requirement 1: The system shall not unlock if the distance between the user and the car is greater than the defined disance.

Requirement 2: The system shall change the status of the car from “reserved” to “in use” when the car is unlocked.

Requirement 3: If the car isn’t picked up within ten minutes, the system has to lock the car again and the ride ends.

Goal 6: The user is charged on a per minute basis from the time when the ride begins.

> **Requirement 1**: The system shall keep track of the time elapsed from the car unlock.
>
> **Requirement 2**: The system shall update the total cost of the ride on a per minute basis, using the current elapsed time.
>
> **Domain Assumption 1**: Payments are withdrew automatically from the user's credit card without any user interaction.
>
> **Domain Assumption 2**: The system doesn't manage payments directly since we rely on an external service who is exposing some specific APIs.
>
> **Domain Assumption 3**: The ride begins when the car is unlocked.

Goal 7: The user is notified of the current charges throgh the car display.

> **Requirement 1**: During the ride, the system shall send the current charges to the car on a per minute basis.
>
> **Domain Assumption 1**: Every car has a display.
>
> **Domain Assumption 2**: The car actually shows the information received from the system.

Goal 8: At the end of the ride the car is locked automatically and the user is charged.

> **Requirement 1**: The system shall lock the car automatically when the car is turned off and there are no more passengers inside.
>
> **Requirement 2**: The system shall wait five minutes before charging the user with the final cost in order to consider possible discounts or additional fees.
>
> **Requirement 3**: The system shall send a notification to the user with information about the ride details and the final cost.
>
> **Domain Assumption 1**: The ride ends when the car is locked.
>
> **Domain Assumption 2**: Payments are withdrew automatically from the user's credit card without any user interaction.

Domain Assumption 3: The system doesn't manage payments directly since we rely on an external service who is exposing some specific APIs.

Domain Assumption 4: The car is equipped with sensors to the detect if there are passengers on the seats.

Goal 9: Suspend the account for insolvent users and redirect them to the customer service.

Requirement 1: The user shall suspend the account for insolvent users.

Requirement 2: The system shall send a notification to the insolvent user telling him/her to get in touch with the customer service.

Domain Assumption 1: The information about insolvent users is provided by the external payment system.

Goal 10: Discourage parking outside of safe areas by charging 80% more on the ride balance and if that happens, mark the car as unavailable.

Requirement 1: The system shall know in advance what are the safe areas and their precise location.

Requirement 2: The system shall charge the user by 80% more if the car is parked outside of a safe area.

Requirement 3: The system shall change the status of the car to "unavailable" if the car is parked outside of a safe area.

Domain Assumption 1: Unavailable cars are managed by a "PowerEnJoy" employee.

Goal 11: The car display provides information about the location of SafeAreas and PowerGridStations.

Requirement 1: The system shall send information to the car about the nearest SafeAreas and PowerGridStations based on the current position of the car.

Domain Assumption 1: The car display integrates a GPS navigator and shows the information received from the system about the nearest SafeAreas and PowerGridStations.

Goal 12: Cars with less then 20% of battery left are marked as unavailable.

> **Requirement 1**: The system shall change the status of to car the "unavailable" if the car is turned off and the battery level is below 20%.
>
> **Domain Assumption 1**: The battery level detected by the car is accurate.
>
> **Domain Assumption 2**: The car is able to send updated information to the system with respect to the battery level.
>
> **Domain Assumption 3**: Unavailable cars are managed by a "PowerEnJoy" employee.

Goal 13: Apply a 10% discount if there are more than 2 passengers.

> **Requirement 1**: The system shall apply a 10% discount on the last ride if the car reports the presence of more than 2 passengers.
>
> **Domain Assumption 1**: The car is equipped with sensors to detect if there are passengers on the seats.

Goal 14: Apply a 20% discount if the car is left with no more than 50% of battery empty.

> **Requirement 1**: The system shall apply a discount of 20% on the last ride if the battery level at the end of the ride is above 50%.
>
> **Domain Assumption 1**: The battery level detected by the car is accurate.
>
> **Domain Assumption 2**: The car is able to send updated information to the system with respect to the battery level.

Goal 15: Apply a 30% discount if the car is plugged to the power grid at the end of the ride.

> **Requirement 1**: The system shall detect if the car has been plugged to a power grid within five minutes from the moment when the car is turned off.
>
> **Requirement 2**: The system shall apply a 30% discount if the car is plugged to the power grid within that time frame.
>
> **Domain Assumption 1**: The car is able to detect if the power cord is plugged in and inform the system about it.

Goal 16: Charge an additional fee if the car is left more than 3km far from the nearest PowerGridStations or with less then 20% of battery left.

> **Requirement 1**: The system shall detect if the car is left in a location that is more than 3km far from the nearest PowerGridStation.
>
> **Requirement 2**: The system shall detect if the battery level is less than 20%.
>
> **Requirement 3**: The system shall apply a 30% additional fee on the cost if these events occur.
>
> **Requirement 4**: The system shall change the status of the car to "unavailable".
>
> **Domain Assumption 1**: The battery level detected by the car is accurate.
>
> **Domain Assumption 2**: The car is able to send updated information to the system with respect to the battery level.

Goal 17: An admin must be able to add and delete SafeAreas and PowerGridStations.

> **Requirement 1**: The system shall verify that the safe area that the admin wants to add isn't already inserted.
>
> **Domain Assumption 1**: In the safe area that the admin wants to delete from the system there are no cars.

Goal 18: An administrator must be able to manage cars: add a new car and delete an existing one, show and change the status and the details of a selected car.

> **Requirement 1**: The system shall verify that the car that the admin wants to add isn't already inserted.
>
> **Requirement 2**: An administrator must be able to get the list of cars whose status is "unavailable" in order to fix the car problem.
>
> **Requirement 3**: The system shall verify that the status of the car that the admin wants to delete or modify isn't 'reserved' or 'in use'.

Goal 19: An admin must be able to manage users: add a new user and delete an existing one; show and update details, payment information and the status of a selected user.

> **Requirement 1**: The system shall verify that the user that the administrator wants to add isn't already registered.
>
> **Requirement 2**: The system shall verify that the inserted or modified information is correct.
>
> **Requirement 3**: The system shall send a notification to the user if his/her information is updated.

Goal 20: An administrator must be able to add new administrators providing an email, a name and a passowrd.

> **Requirement 1**: The system shall verify that the new admin isn't already registered.
>
> **Requirement 2**: The system shall verify that the inserted email is valid.

3 Scenarios Identification

3.1 Registration

Partecipating actor: Guest (Tommaso)

Flow of events: Tommaso has seen an advertisement of “PowerEnJoy”, a new car-sharing service that exclusively employs electric cars. So he decides to sign up, because he thinks that this service could be useful for him. He accesses the site, clicks on the “*Sign up*” button and fills out the form with the information needed for the registration (name, surname, address, e-mail, mobile phone number, driving license number, credit card number), checks the checkbox “*I have read and agree to PowerEnJoy Terms of Use and Privacy Policy.*” and he confirms. However he forgets to insert the credit card number, so an error message is displayed and Tommaso is brought back to the page where the registration form is shown. Tommaso inserts all data and then clicks on “Confirm”. The system confirms his registration, redirects Tommaso on the login page and sends him an e-mail containing his password.

3.2 Login

Partecipating actor: Registered user (Tommaso)

Flow of events: Tommaso has received the e-mail containing his credentials and he wants to access his personal profile page. He access the homepage of the website and clicks on the “*Log in*” button. Tommaso inserts his email and his password and the login page is refreshed to the user personal profile page.

3.3 Recover the password

Partecipating actor: Registered user (Patrizia)

Flow of events: Patrizia needs to reserve a car, but she is a bit careless and she doesn’t remember the password. She accesses the site and clicks on “*Log in*”. She tries to insert three different passwords, but every time an error message is displayed. So she clicks on “*Send a new password*”: the system sends her a new password by e-mail and shows a confirmation message. After two minutes she receives the e-mail. Patrizia tries the login again: she inserts her email and the new password received and confirms. Now she’s correctly authenticated.

3.4 Reserve a car

Partecipating actor: Registered user (Patrizia)

Flow of events: Patrizia needs a car to arrive on time to a meeting on the other side of the city. So she decides to use the web application "PowerEnJoy". This isn't the first time she uses this service, so she has already registered. She accesses the site and clicks on "*Log in*" button. She inserts her credentials and confirms. After the authentication is completed, she is redirected on her personal profile page, where she can decide to visualize her personal information or the list of her previous reservations and rides. She clicks on the button "*Reserve a car*" in order to complete her reservation. At this point she has to choose among two options: use her current location or specify an address. She selects the first one because she needs a car as near as possible to the place where she is. She inserts 100 meters as maximum distance for the research and clicks on "*Search cars*". The system searches available cars within 100 meters from her current location. Unluckily there are no available cars in the specified area, so an apologize message is displayed and she is redirected to the previous page (reservation page). She ticks "*Use my current position*" again and insert 500 meters as new maximum distance value and clicks on "*Search cars*". This time, the system shows a list of available cars specifying for each of them the license plate number and the location. She selects the best car for her purposes and clicks on the "*Reserve*" button. A confirmation message appears. Now Patrizia walks quickly to the car and when she is next to the car, she opens the web application on her mobile phone, logs in and clicks on "*Unlock the reserved car*". The system checks her location and, because of she's at less than 3 meters of distance, unlocks the car. She gets in the car and carefully drives to her appointment.

3.5 Unlock the reserved car

Partecipating actor: Registered user (Patrizia)

Flow of events: Patrizia has already reserved a car and while she's walking to it she accesses the homepage of the "PowerEnJoy", logs in writing her credentials and she's redirected to the profile page. From her personal profile page she clicks on "*Unlock the reserved car*", but an error message is displayed: "*ERROR: you are too far from the car to unlock it! Please go next to the reserved car.*". So Patrizia goes on walking and when she's at less than 3 meters from the car, she tries again: accesses the homepage, logs in and clicks on "*Unlock the reserved car*". The system checks her location and unlocks the car. Patrizia gets in the car and goes to work.

3.6 End the ride

Partecipating actor: Registered user (Tommaso)

Flow of events: Tommaso is driving a PowerEnJoy car and his girlfriend, who is sitting next to him, is looking outside the window. In the seats behind there are two Tommaso's friends, that are singing. They are going to a birthday party. When they are close to the place of the party, Tommaso looks to the screen of the car, where there are displayed the cost of the ride within this moment and a map. In the map are highlighted the current location of the car, the safe areas (PowerEnJoy parking areas) and the power grid stations (PowerEnJoy parking area where the car can be re-charged). Tommaso notices that there is only a power grid station at 5 minutes walking from the party place. He could enter in the power grid station or park in the car parking in front of the party place, which is not owned by PowerEnJoy. Tommaso knows that if he parks in the power grid station he receives a 30% discount on the cost of the ride, otherwise he will pay 80% more. So he decides to park in the power grid station, although the protests of his friends. As soon as Tommaso and his friends get off the car, it locks, Tommaso takes care of plugging the car into the power grid and they go to the party. Five minutes later, Tommaso receives an e-mail with the confirmation of payment success and the details of his last ride: the amount is 12,60€, but the system applies a discount of 10% of 12,60€ (1,26€) because he takes more than one passenger onto the car and a discount of 30% of 12,60€ (3,78€) because he parks in a power grid station and he plugs the car into the power grid. So he has a total discount of 5,04€ and he pays 7,56€.

3.7 Expired reservation

Partecipating actor: Registered user (Patrizia)

Flow of events: Patrizia is at the shopping center and she decides to return back home using PowerEnJoy car-sharing service. She needs to buy few things more so she believes to finish her shopping in less than an hour. Patrizia opens the PowerEnJoy web application with her mobile phone, logs in, reserves a car near the shopping center and she goes on with her shopping. Then she searches for the last item, but she doesn't find it until asking the clerk. Without realizing she has lost a lot of time. After founding all she needs, Patrizia gets in line to pay. There is a lot of people and the queue is long. When Patrizia manages to pay, she realizes that more than an hour has passed and her reservation has expired. Unfortunately she wastes 1€. She opens again the PowerEnJoy web application and makes a new reservation, then she reaches the selected car, unlocks it and returns home.

3.8 Payment not successful

Partecipating actor: Registered user (Tommaso)

Flow of events: Tommaso used the PowerEnJoy car-sharing service to go to the airport. But this month Tommaso has bought a lot of things and he exceeded the maximum usage limit of his credit card, so at the end of the ride, the automatic payment fails. Tommaso's account is suspended by the system and he receives an e-mail telling him to get in touch with the customer care. Until he doesn't call the customer care and solve the payment problem, Tommaso couldn't interact with the platform anymore. So Tommaso phones the customer care, explains his problem and pays his debt using another payment method. Finally, the administrator can update Tommaso's payment information and change the status from 'insolvent' to 'active'.

3.9 Update personal information

Partecipating actor: Registered user (Patrizia)

Flow of events: Few days ago Patrizia move to a new house, so she has to change her personal information on the web application "PowerEnJoy". She accesses the homepage and logs in with her credentials. She authenticates correctly, so she is on her profile page. She clicks on "*Personal information*" and she's redirected on the page where she can visualize her information. Then she clicks on "*Edit*" and replace the old address with the new one and verifies that also the other fields are correct. Now she has to click on "*Save changes*" in order to store the update.

3.10 Show the details of the ride

Partecipating actor: Registered user (Tommaso)

Flow of events: Last week Tommaso went to a football match with his friends using a "PowerEnJoy" car. Now he wants to know how much they spent for the ride in order to divide the cost with his friends. So he accesses the homepage and clicks on "*Log in*", he inserts the correct credentials and he authenticates correclty. In the personal profile page he clicks on "*My rides*" at the top of the page and he is redirected to a new page where he can see the list of all his reservations and rides. Clicking on the selected ride, Tommaso can see all the details about the ride (cost, discounts, starting point, number of passengers . . .). Now he knows the total cost of the ride and his friends can repay him.

3.11 Add new cars

Partecipating actor: Administrator (Adam)

Flow of events: The "PowerEnJoy" bought some new cars in order to improve its car sharing. So Adam, one of the company administrators, has to add these cars to the system. He accesses the webapp, inserts his credentials, clicks on the "*Cars*" button and then on "*Add a new car*". At this point, he fills the form with the details, like the licence plate, of the first car and sets the status to 'available'. The system checks and sets the battery level and the position using the information provided from the sensors on each car. Then a confirmation message is shown and the administrator can add the second car in the same way and so on until all the bought cars are added. Finally, Adam logs out.

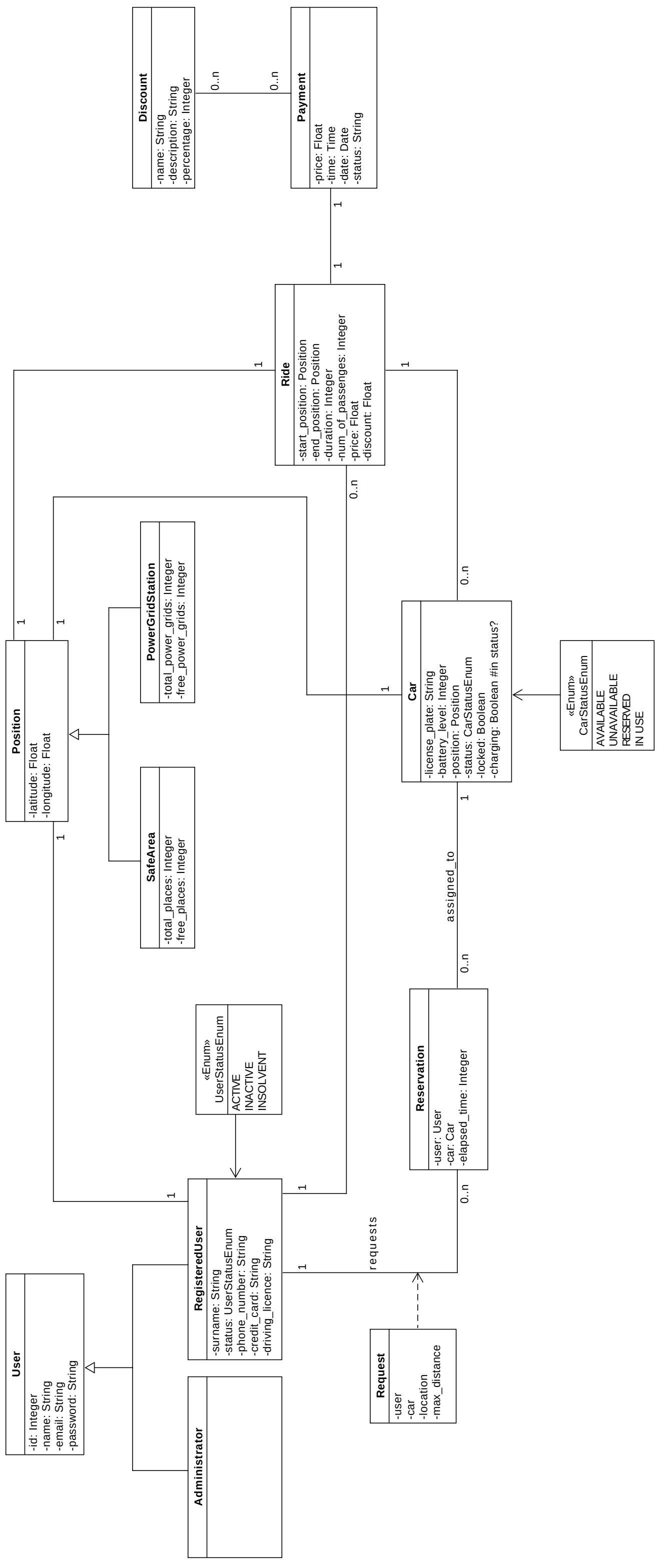
3.12 Delete a safe area

Partecipating actor: Administrator (Adam)

Flow of events: The "PowerEnJoy" sold the power grid station situated near the gym because very few people used to park there. So Adam, the new employee of the company, has to update the safe area list deleting the sold power grid station. He accesses the webapp, logs in and clicks on "Safe areas": a list of all the safe areas and power grid stations appears, Adam select the sold one and clicks on "Delete". His task is completed and he can log out.

4 UML modeling

4.1 Class Diagram

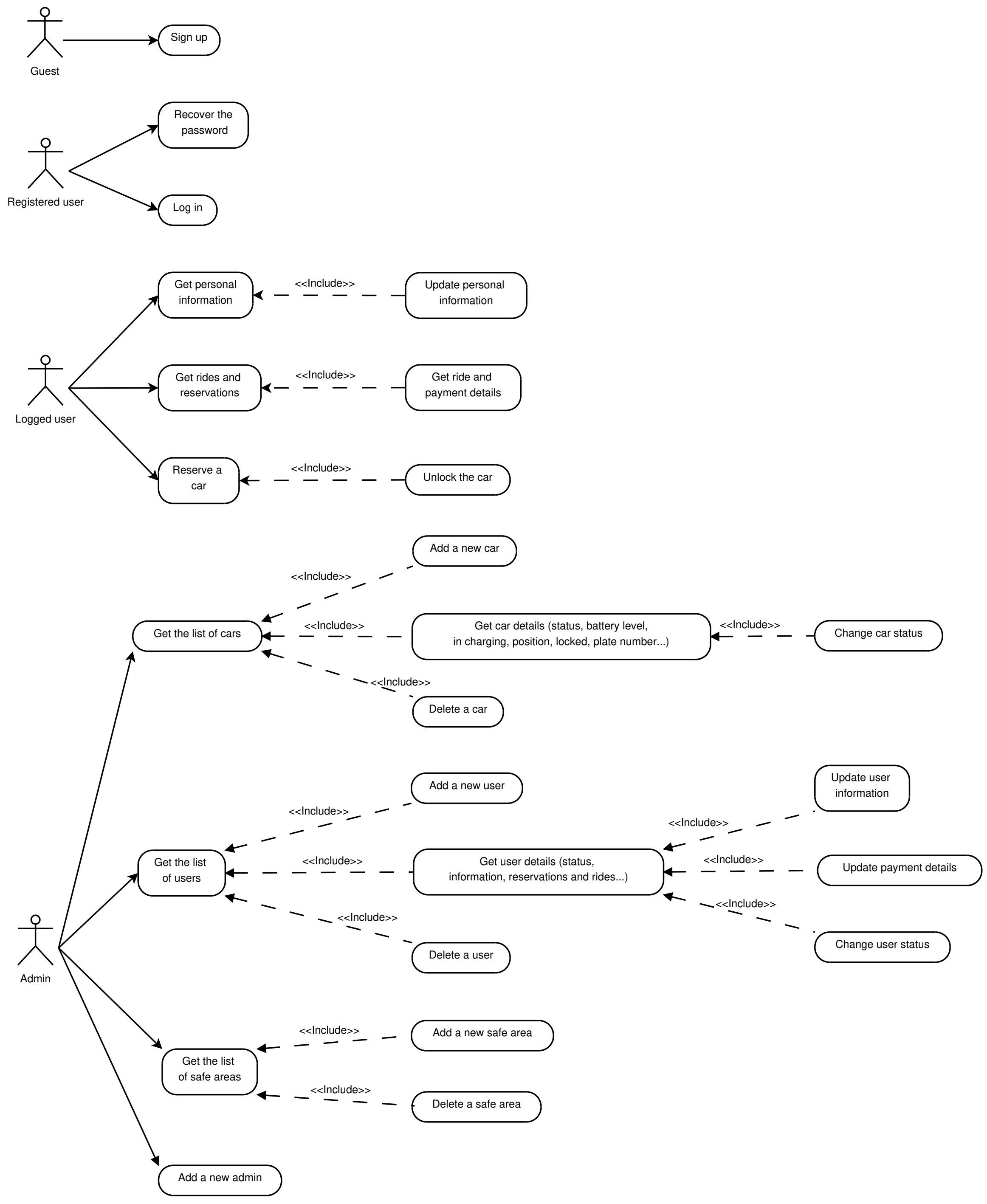


Discount
-name: String
-description: String
-percentage: Integer
Payment
-price: Float
-time: Time
-date: Date
-status: String
0..n
0..n
1
1
Ride
-start_position: Position
-end_position: Position
-duration: Integer
-num_of_passenges: Integer
-price: Float
-discount: Float
1
1
0..n
Position
-latitude: Float
-longitude: Float
1
1
1
PowerGridStation
-total_power_grids: Integer
-free_power_grids: Integer
SafeArea
-total_places: Integer
-free_places: Integer
Car
-license_plate: String
-battery_level: Integer
-position: Position
-status: CarStatusEnum
-locked: Boolean
-charging: Boolean #in status?
0..n
1
1
«Enum»
CarStatusEnum
AVAILABLE
UNAVAILABLE
RESERVED
IN USE
assigned_to
0..n
Reservation
-user: User
-car: Car
-elapsed_time: Integer
0..n
«Enum»
UserStatusEnum
ACTIVE
INACTIVE
INSOLVENT
RegisteredUser
-surname: String
-status: UserStatusEnum
-phone_number: String
-credit_card: String
-driving_licence: String
1
1
1
requests
User
-id: Integer
-name: String
-email: String
-password: String
Administrator
Request
-user
-car
-location
-max_distance

4.2 Actors identification

- **Guest:** a guest, as we already mentioned in the glossary section, is someone who hasn't sign up. He/she can only visit the homepage and sign up.
- **Registered user:** a registered user, or simply a user, is a person that has already registered in the system. He/she has a profile with his/her personal information and, after logging in, he/she has the capability of using all the services that the application provides.
- **Logged user:** a logged user is a registered user that has already logged in. When a user is logged, he/she can:
  - visualize his/her profile page, personal information and previous rides and reservations;
  - update personal information;
  - search and reserve available cars;
  - unlock the reserved car.
- **Administrator:** An administrator, or simply an admin, is a "PowerEnJoy" employee whose task is to manage cars, safe areas, users and administrators. An administrator can:
  - get the list of all users;
  - add and delete users;
  - get the details of a selected user, update his/her information, change his/her status and modify payment details;
  - get the list of all cars;
  - get the list of all cars whose status is "unavailable";
  - add and delete cars;
  - get the details of a selected car and change its status;
  - get the list of all safe areas and power grid stations;
  - add and delete a safe area or a power grid station;
  - add a new administrator.

4.3 Use Cases



Guest
Sign up
Registered user
Recover the password
Log in
Logged user
Get personal information
<<Include>>
Update personal information
Get rides and reservations
<<Include>>
Get ride and payment details
Reserve a car
<<Include>>
Unlock the car
Admin
Get the list of cars
Add a new car
<<Include>>
Get car details (status, battery level, in charging, position, locked, plate number...)
<<Include>>
Change car status
<<Include>>
Delete a car
<<Include>>
Get the list of users
Add a new user
<<Include>>
Get user details (status, information, reservations and rides...)
<<Include>>
Update user information
<<Include>>
Update payment details
<<Include>>
Change user status
<<Include>>
Delete a user
<<Include>>
Get the list of safe areas
Add a new safe area
<<Include>>
Delete a safe area
<<Include>>
Add a new admin

In this paragraph we are going to identify and describe the most important use cases of "PowerEnjoy" web application. Based on the scenarios defined in the previous chapter, we can derive some significant use cases:

1. Registration
2. Login
3. Recover the password
4. Reserve a car
5. Unlock the reserved car
6. End the ride
7. Expired reservation
8. Payment not successful
9. Update personal information
10. Show the details of the ride
11. Add new cars
12. Delete a safe area

4.3.1 Registration

Partecipating actors: Guest: a guest is whoever visits the website

Entry condition: This use case starts when the guest access the homepage of the web application and clicks on "*Sign up*".

Flow of events:

- The guest clicks on "*Sign up*"
- The guest fills out the form entering all required information:
  - name
  - surname
  - address
  - e-mail
  - mobile phone number
  - driving license number
  - credit card number

- The guest checks the checkbox "*I have read and agree to PowerEnJoy Terms of Use and Privacy Policy.*"
- The guest clicks on "*Confirm*"
- The system verifies that the user isn't already registered
- The system verifies that the inserted information is valid
- The system stores the new data in the user's database
- The system sends the user an e-mail containing his/her credentials (username and password)
- The system displays a confirmation message, informing the new user that the registration has been successfully completed
- The system shows the new user the log in page

Exit condition: This use case terminates when the registration is successfully completed and the new user receives the mail with his/her credentials.

Exceptions:

- **The guest is already a registered user:** if this exception occurs, the system displays the error message: "*ERROR: you are already registered!*" and the application goes back to the page where the registration form is shown.
- **The inserted information isn't valid:** if this exception occurs, the system displays the error message: "*ERROR: inserted information is not valid!*" and the application goes back to the page where the registration form is shown.
- **The user doesn't fill all the fields in the registration form:** if this exception occurs, the system displays the error message: "*ERROR: all the fields has to be filled!*" and the application goes back to the page where the registration form is shown.

4.3.2 Login

Partecipating actors: Registered user: a registered user is a guest that has already signed up.

Entry condition: This use case starts when the registered user, that has already received the e-mail with his/her credentials, clicks on "*Log in*" from the homepage of the website.

Flow of events:

- The registered user clicks on "*Log in*"
- The registered user enters his/her email
- The registered user enters his/her password
- The registered user clicks on "*Confirm*"
- The system checks the inserted email
- The system checks the inserted password
- The system shows the user's personal profile page

Exit condition: This use case terminates when the login is successfully completed and the new user access his/her personal area.

Exceptions:

- **The inserted email is incorrect:** if this exception occurs, the system displays the error message: "*ERROR: the inserted email is wrong!*" and the application goes back to the log in page.
- **The inserted password is incorrect:** if this exception occurs, the system displays the error message: "*ERROR: the inserted password is wrong!*" and the application goes back to the log in page

4.3.3 Recover the password

Partecipating actors: Registered user: a registered user is a guest that has already signed up.

Entry condition: This use case starts when the user forgot the password and clicks on "*Send a new password*" from the login page.

Flow of events:

- The registered user is on the login page and clicks on "*Send a new password*"
- The system sends the user an e-mail with the new password
- The system displays a confirmation message
- The system displays the login page
- The user receives the e-mail with the new password

Exit condition: This use case terminates when the user receives the new password by e-mail.

4.3.4 Reserve a car

Partecipating actors: Registered user: a registered user is a guest that has already signed up.

Entry condition: This use case starts when, after logging in, the registered user clicks on "*Reserve a car*" from his/her personal profile page.

Flow of events:

- The registered user clicks on "*Reserve a car*"
- The system checks that the status of the user is 'active'
- The system shows the reservation form
- The user selects where to search the car: choosing if use his/her current location or a specified address (in this case he/she also has to write an address)
- The user selects a maximum distance for the car research
- The user clicks on "*Search cars*"
- The system searches available cars within the maximum distance indicated from the given location
- The system shows the list of available cars
- The user selects one of the cars in the list
- The user clicks on "*Reserve*"
- The system changes the status of the car from "*available*" to "*reserved*".
- The system starts counting elapsed time from the reservation.
- The system displays a confirmation message containing the details of the reservation (time of the reservation, license plate, position, distance from the given location . . .)
- The system displays the user's personal profile page

Exit condition: This use case terminates when the confirmation message is shown and the user is redirected on his/her personal profile page.

Exceptions:

- **The user doesn't select a location for the car research:** if this exception occurs, the system displays the error message: "*ERROR: you have to select a location for the car research!*" and the application goes back to the page where the reservation form is shown.
- **The user writes an inexistent address:** if this exception occurs, the system displays the error message: "*ERROR: the inserted address doesn't exist!*" and the application goes back to the page where the reservation form is shown.
- **The user writes an invalid value for the maximum distance:** if this exception occurs, the system displays the error message: "*ERROR: the inserted distance isn't valid!*" and the application goes back to the page where the reservation form is shown.
- **There aren't cars in the selected area:** if this exception occurs, the system displays the error message: "*ERROR: there are no cars in the selected area! Please change the maximum distance or the selected location.*" and the application goes back to the page where the reservation form is shown.
- **The status of the user isn't 'active':** if this exception occurs, the system displays the error message: "*ERROR: your status is not active, there is a problem with your account! Contact the customer care.*" and the application goes back to the personal profile page.

4.3.5 Unlock the reserved car

Partecipating actors: Registered user: a registered user is a guest that has already signed up.

Entry condition: This use case starts when a registered user who has already reserved a car wants to unlock it, so logs in the application.

Flow of events:

- The system displays the user's personal profile page
- The user clicks on "*Unlock the reserved car*"
- The system checks the distance between the user and the reserved car
- The system change the status of the car from "*reserved*" to "*in use*"
- The system unlocks the car
- The user gets on the car in less than ten minutes

Exit condition: This use case terminates when the system unlocks the car and the user gets in.

Exceptions:

- **The user is at more than 3 meters from the car:** if this exception occurs, the system displays the error message: "*ERROR: you are too far from the car to unlock it! Please go next to the reserved car.*" and the application shows the user's personal profile page.
- **The user doesn't gets on the car in less than ten minutes:** if this exception occurs, the system locks the car and the ride is considered ended.

4.3.6 End the ride

Partecipating actors: Registered user: a registered user is a guest that has already signed up.

Entry condition: This use case starts when the car is parked and the user and all eventual passengers exit the car.

Flow of events:

- The user parks and turns off the car
- The user and eventual passengers exit the car
- The system locks the car
- The system waits five minutes
- The system checks the location
- The system checks the level of the battery
- The system checks if the power grid is plugged
- The system calculates the total to be paid applying possible discounts
- The system charges the user the cost of the ride
- The system changes the status of the car from "*in use*" to "*available*" or "*unavailable*", based on car conditions
- The system sends the user an e-mail containing all the details of the ride (total cost, discounts, fees, duration, starting location, ending location. . .)
- The user receives the e-mail

Exit condition: This use case terminates when the user receives the e-mail containing the details of his last ride.

4.3.7 Expired reservation

Partecipating actors: Registered user: a registered user is a guest that has already signed up.

Entry condition: This use case starts when an hour has passed from when the user reserved a car and he/she hasn't already unlocked the car.

Flow of events:

- After an hour from the reservation, the system changes the status of the car from "*reserved*" to "*available*"
- The system imposes a charge of 1€ to the user

Exit condition: This use case terminates when the status of the car is changed into "*available*"

4.3.8 Payment not successful

Partecipating actors: Registered user: a registered user is a guest that has already signed up.

Entry condition: This use case starts when the automatic payment fails.

Flow of events:

- The automatic payment fails and the external service notifies the system.
- The system change the status of the user's account from "*active*" to "*insolvent*"
- The system sends the user an e-mail telling to get in touch with the costumer care
- The user receive the e-mail

Exit condition: This use case terminates when the user receive the e-mail telling to get in touch with the costumer care.

4.3.9 Update personal information

Partecipating actors: Registered user: a registered user is a guest that has already signed up.

Entry condition: This use case starts when a registered user is on his/her personal profile page and clicks on "*Personal information*".

Flow of events:

- The user clicks on "*Personal information*"
- The system shows the page where the user can visualize his/her personal information
- The user clicks on "*Edit*"
- The system shows an input form where the user can make changes
- The user updates the information he/she wants in the specific input form
- The user clicks on "*Save changes*"
- The system verifies that the updated information is valid
- The system stores the new data in the user's database
- The system shows the page where the user can visualize his/her personal information

Exit condition: This use case terminates when the new data are stored in the user's database and the personal information page is shown.

Exceptions:

- **The inserted information isn't valid:** if this exception occurs, the system displays the error message: "*ERROR: inserted information is not valid!*" and the application goes back to the page where the user can make changes.

4.3.10 Show the details of the ride

Partecipating actors: Registered user: a registered user is a guest that has already signed up.

Entry condition: This use case starts when a registered user clicks on "*My rides*" from his/her personal profile page.

Flow of events:

- The registered user clicks on "*My rides*"
- The system shows the page where the user can visualize the list of his/her rides and reservations
- The user clicks on the selected ride or reservation
- The system shows the specific page of the selected ride (or reservation) containing all the ride details

Exit condition: This use case terminates when the system visualize the details of the ride (or reservation) selected by the user.

Exceptions:

- **The user is just registered to the application and he/she has never reserved nor used a "PowerEnJoy" electric car:** if this exception occurs, the system show an empty list.

4.3.11 Add new cars

Partecipating actors: Admin: an admin is a person whose task is to manage cars, safe areas, users and other administratos.

Entry condition: This use case starts when an admin logs in.

Flow of events:

- The admin logs in with his/her credentials
- The system shows the administrators page
- The administrator clicks on the button "*Cars*"
- The system shows the list of all "PowerEnJoy" cars
- The administrator clicks on "*Add a new car*"
- The system shows the form for the registration of a new car
- The administrator fills the form with electric car details (plate number)
- The system checks and sets the information provided by sensors on the car (position, battery level...)
- The system change the status of the car to "available"
- The system adds the car to the database and show a confirmation message

Exit condition: This use case terminates when the system shows the confirmation message.

4.3.12 Delete a safe area

Partecipating actors: Admin: an admin is a person whose task is to manage cars, safe areas, users and other administratos.

Entry condition: This use case starts when an admin logs in.

Flow of events:

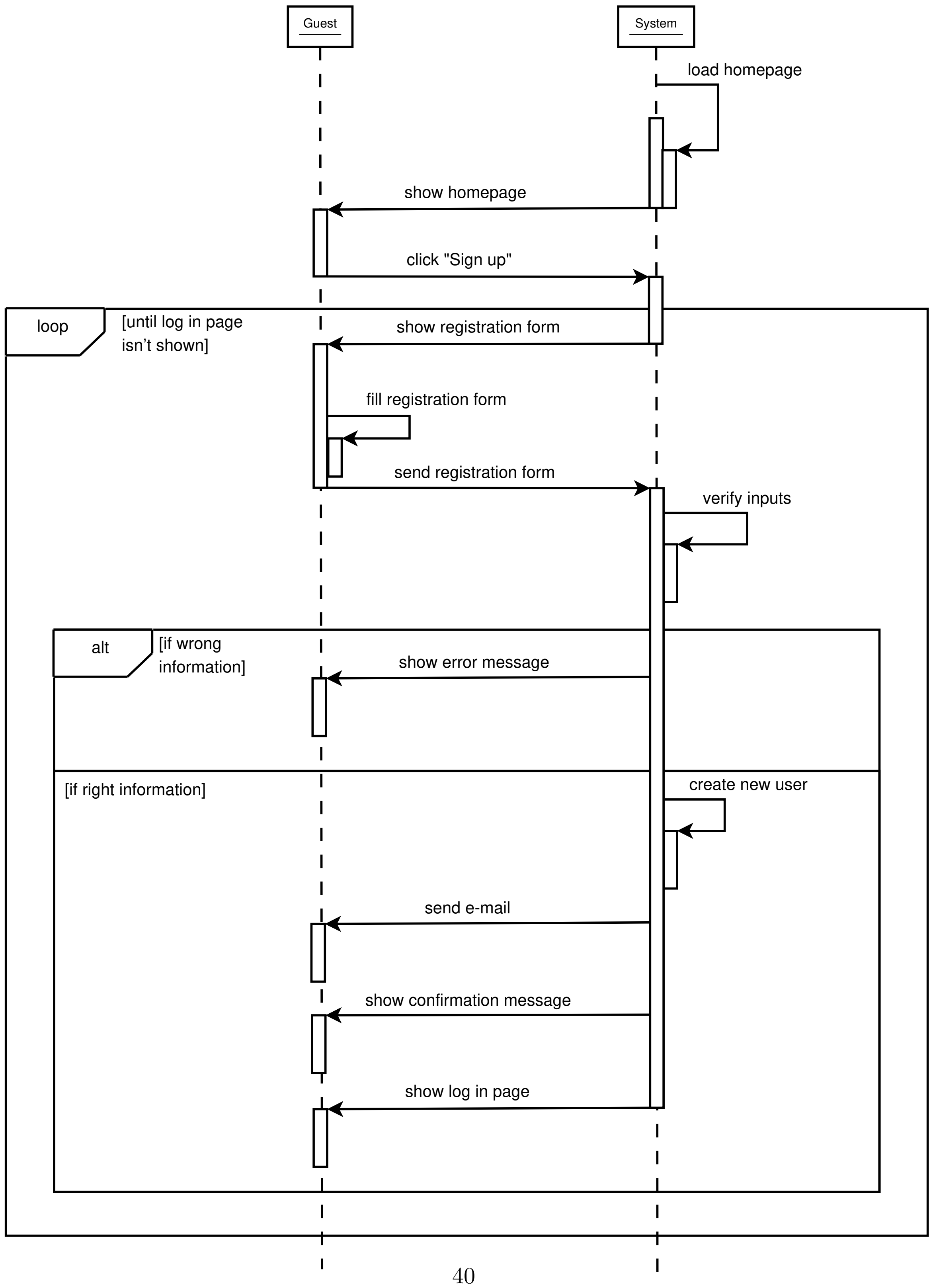
- The admin logs in with his/her credentials
- The system shows the administrators page
- The administrator clicks on the button "*Safe areas*"
- The system shows the list of all "PowerEnJoy" safe areas and power grid stations
- The administrator selects the safe area that has to be deleted and clicks on it
- The system shows the details about the selected safe area
- The administrator clicks on "*Delete*"
- The system deletes the selected safe area from the database and shows a confirmation message

Exit condition: This use case terminates when the system shows the confirmation message.

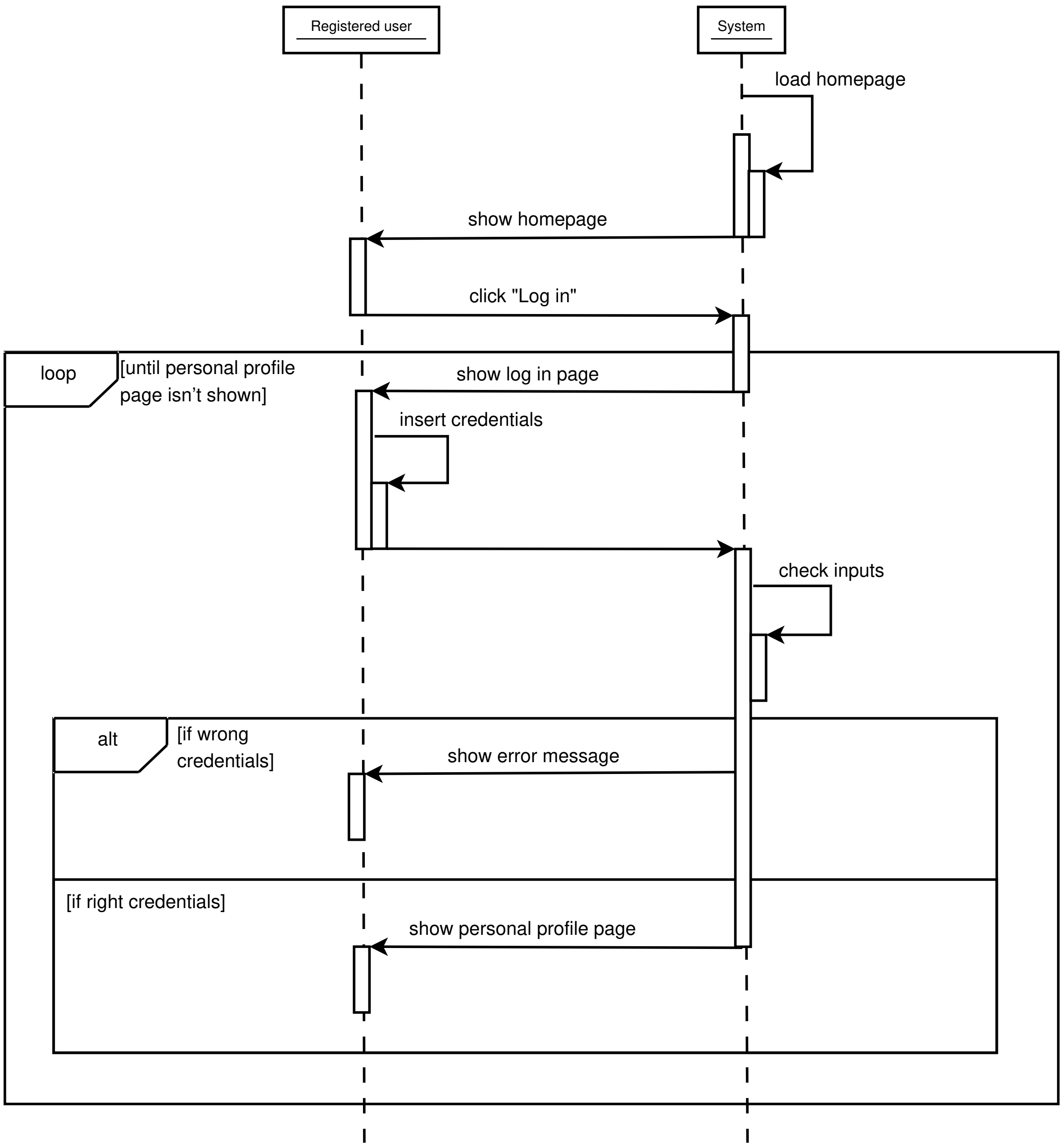
4.4 Sequence diagrams

In this paragraph we are going to show the sequence diagrams associated to some of the use cases in order to explain the interaction between objects.

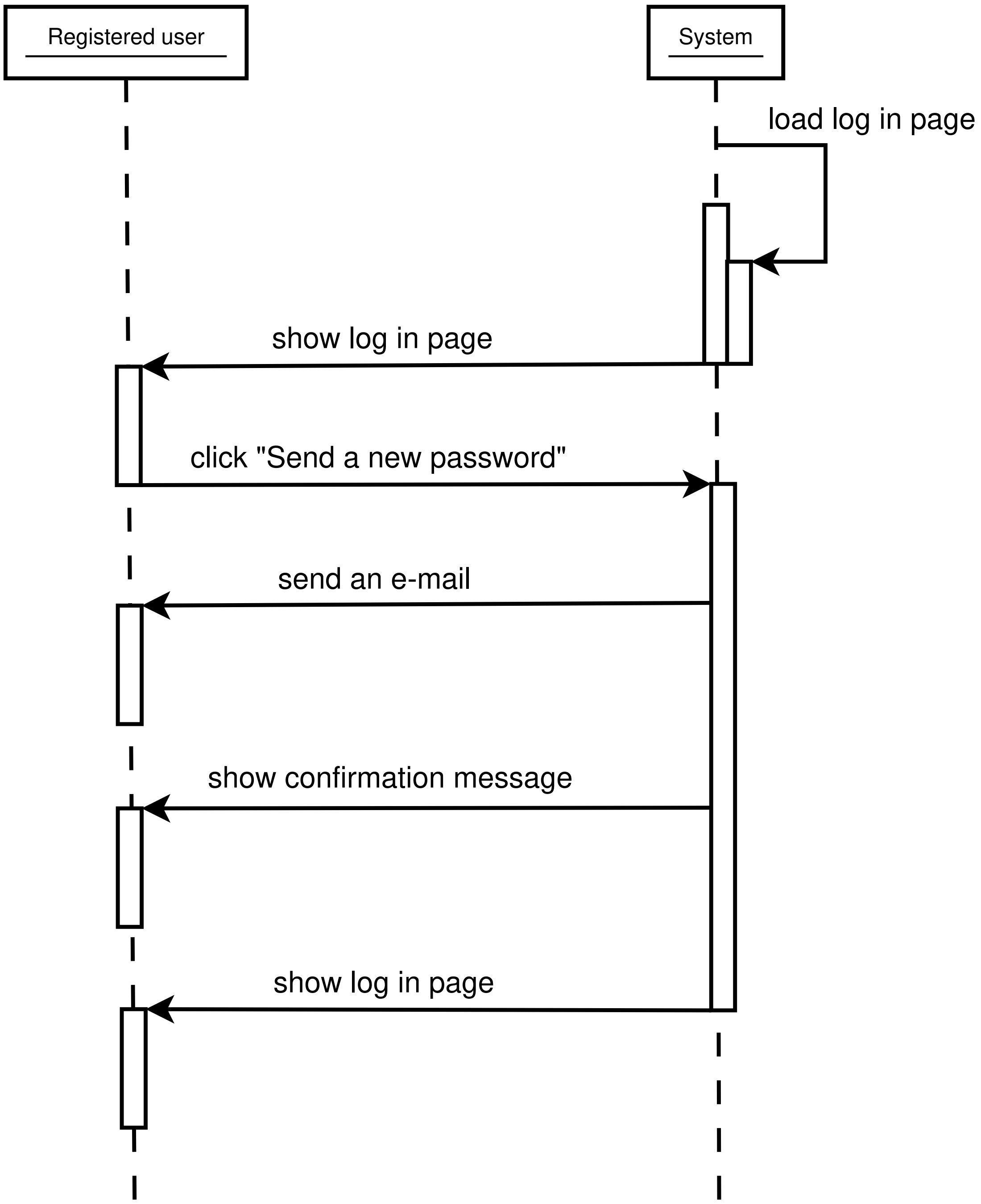
4.4.1 Registration



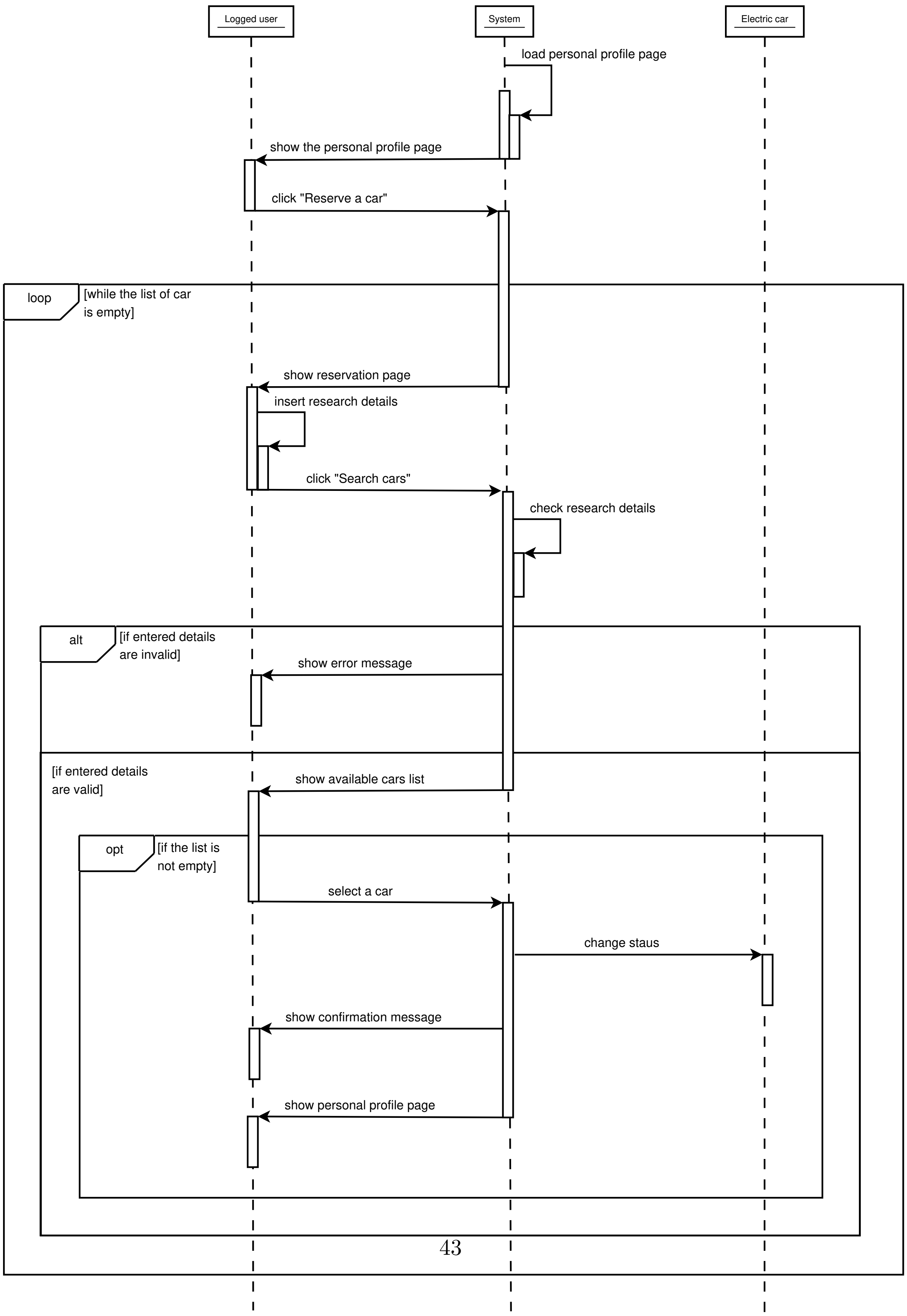
4.4.2 Login


Registered user
System
load homepage
show homepage
click "Log in"
loop
[until personal profile page isn't shown]
show log in page
insert credentials
check inputs
alt
[if wrong credentials]
show error message
[if right credentials]
show personal profile page

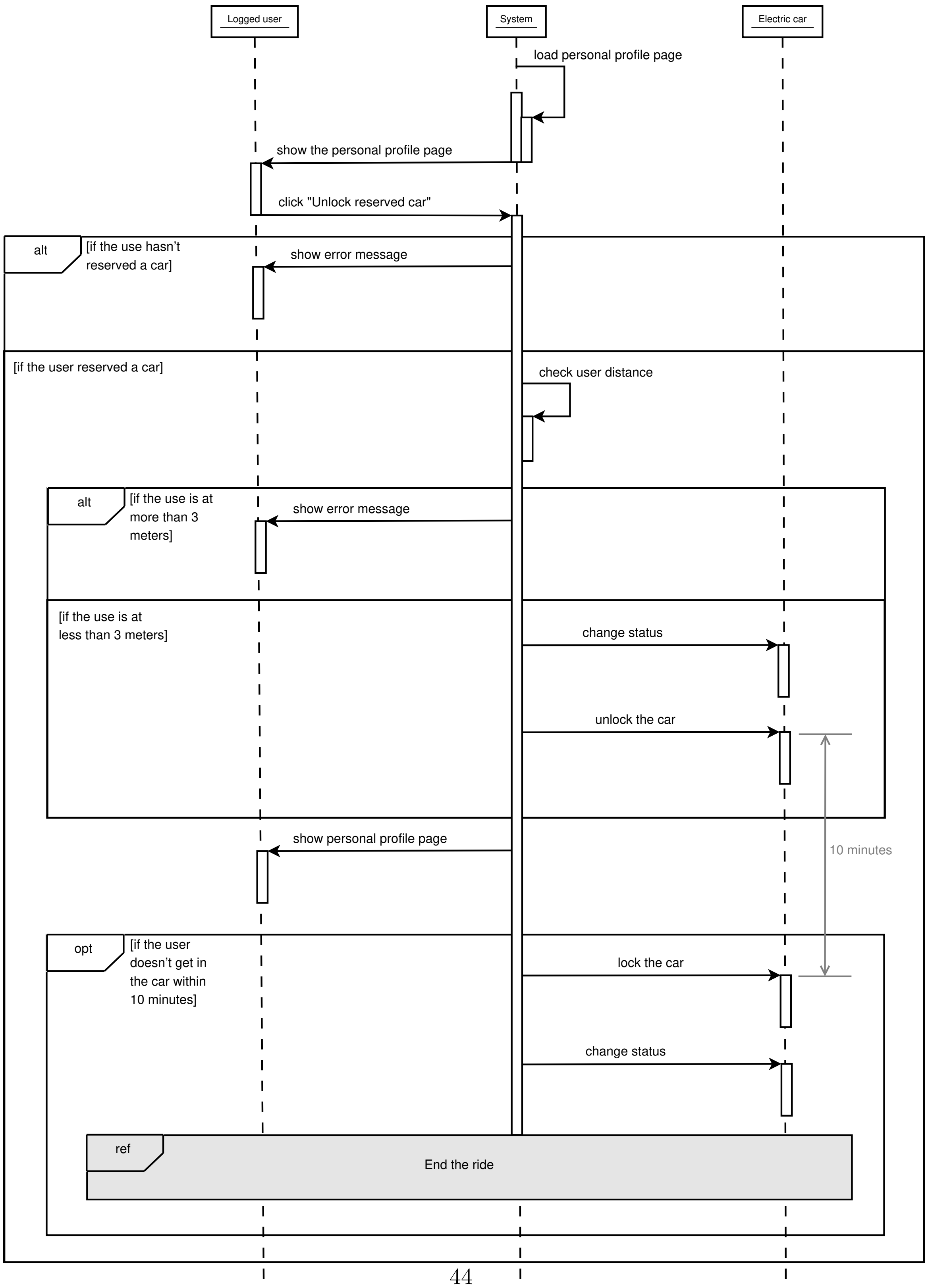
4.4.3 Recover the password



4.4.4 Reserve a car

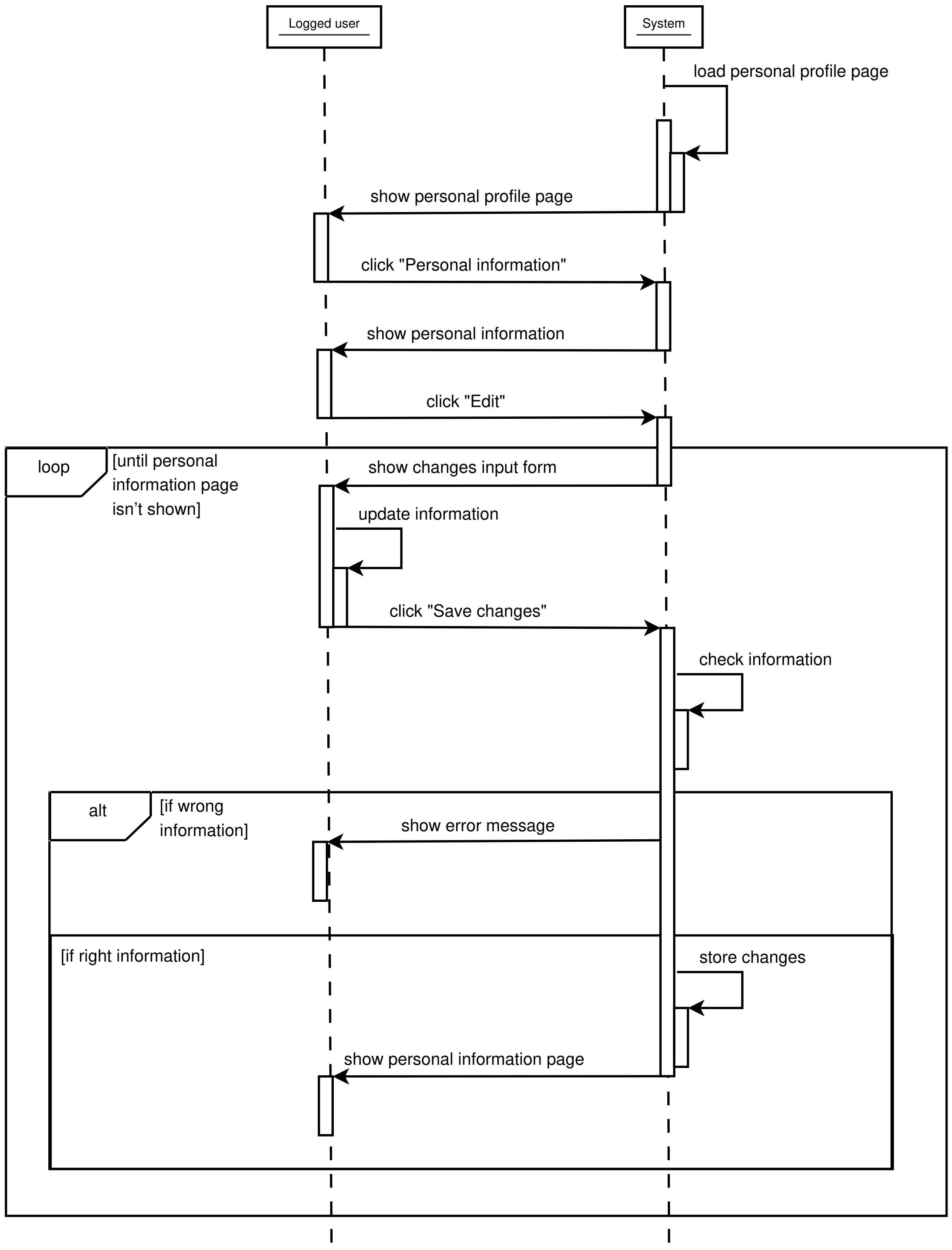

Logged user
System
Electric car
load personal profile page
show the personal profile page
click "Reserve a car"
loop
[while the list of car is empty]
show reservation page
insert research details
click "Search cars"
check research details
alt
[if entered details are invalid]
show error message
[if entered details are valid]
show available cars list
opt
[if the list is not empty]
select a car
change staus
show confirmation message
show personal profile page

4.4.5 Unlock the reserved car

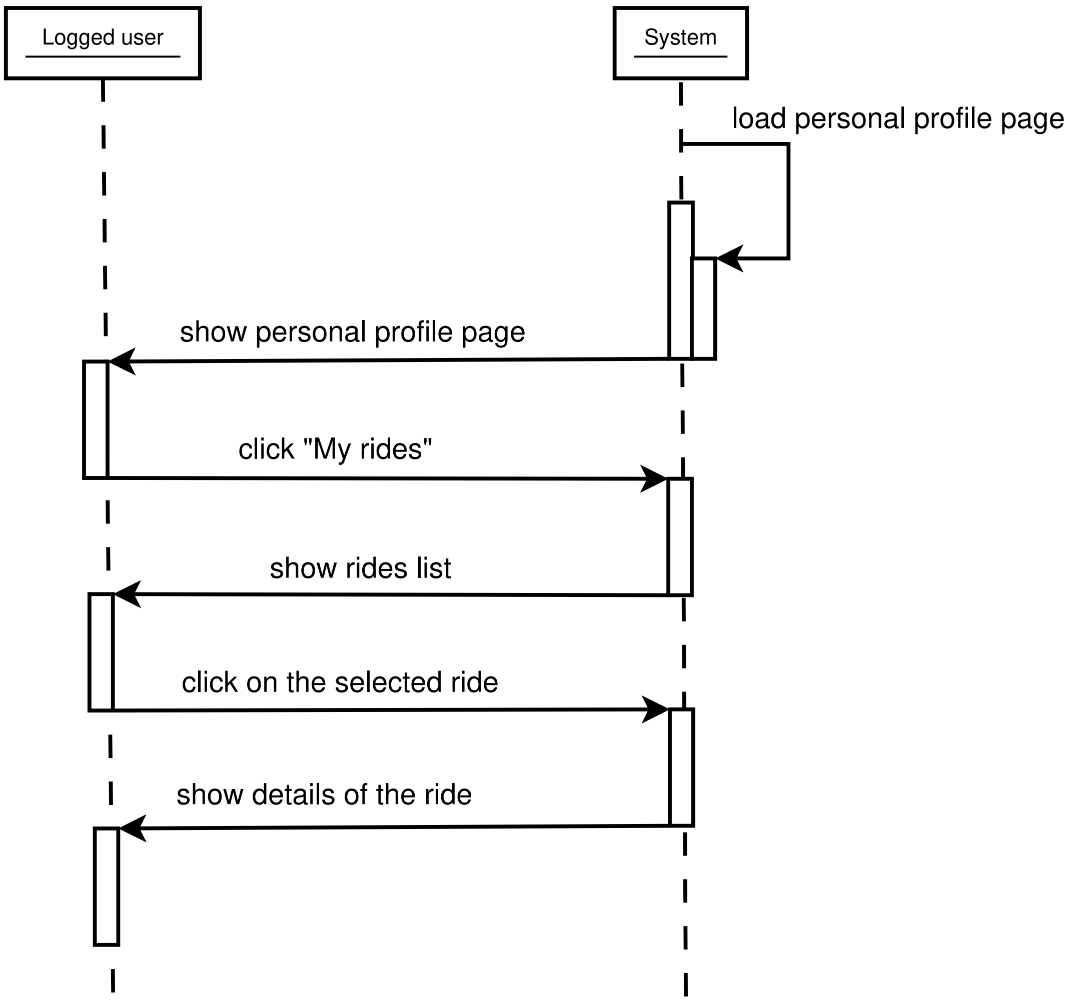


Logged user
System
Electric car
load personal profile page
show the personal profile page
click "Unlock reserved car"
alt
[if the use hasn't reserved a car]
show error message
[if the user reserved a car]
check user distance
alt
[if the use is at more than 3 meters]
show error message
[if the use is at less than 3 meters]
change status
unlock the car
10 minutes
show personal profile page
opt
[if the user doesn't get in the car within 10 minutes]
lock the car
change status
ref
End the ride

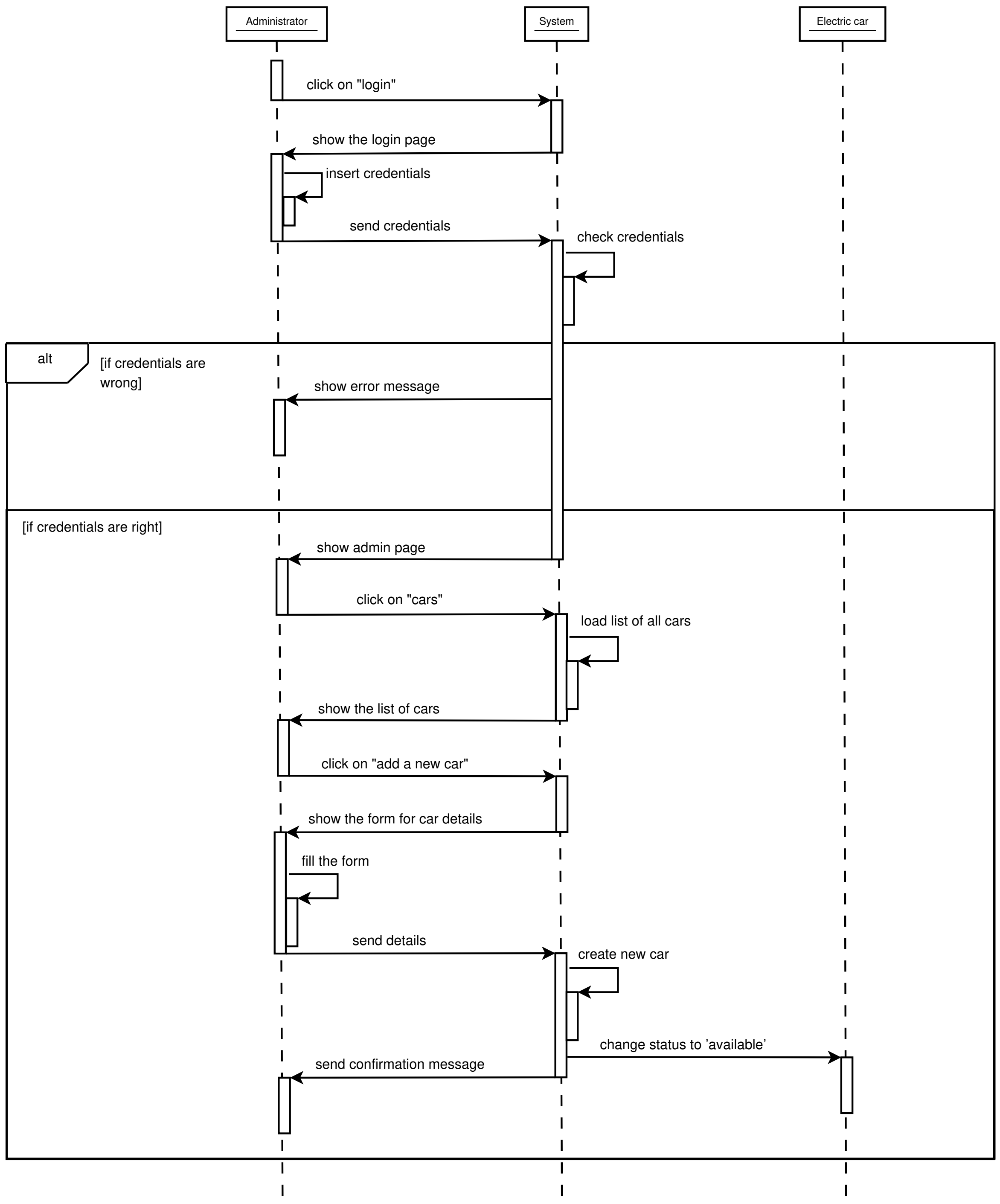
4.4.6 Update personal information



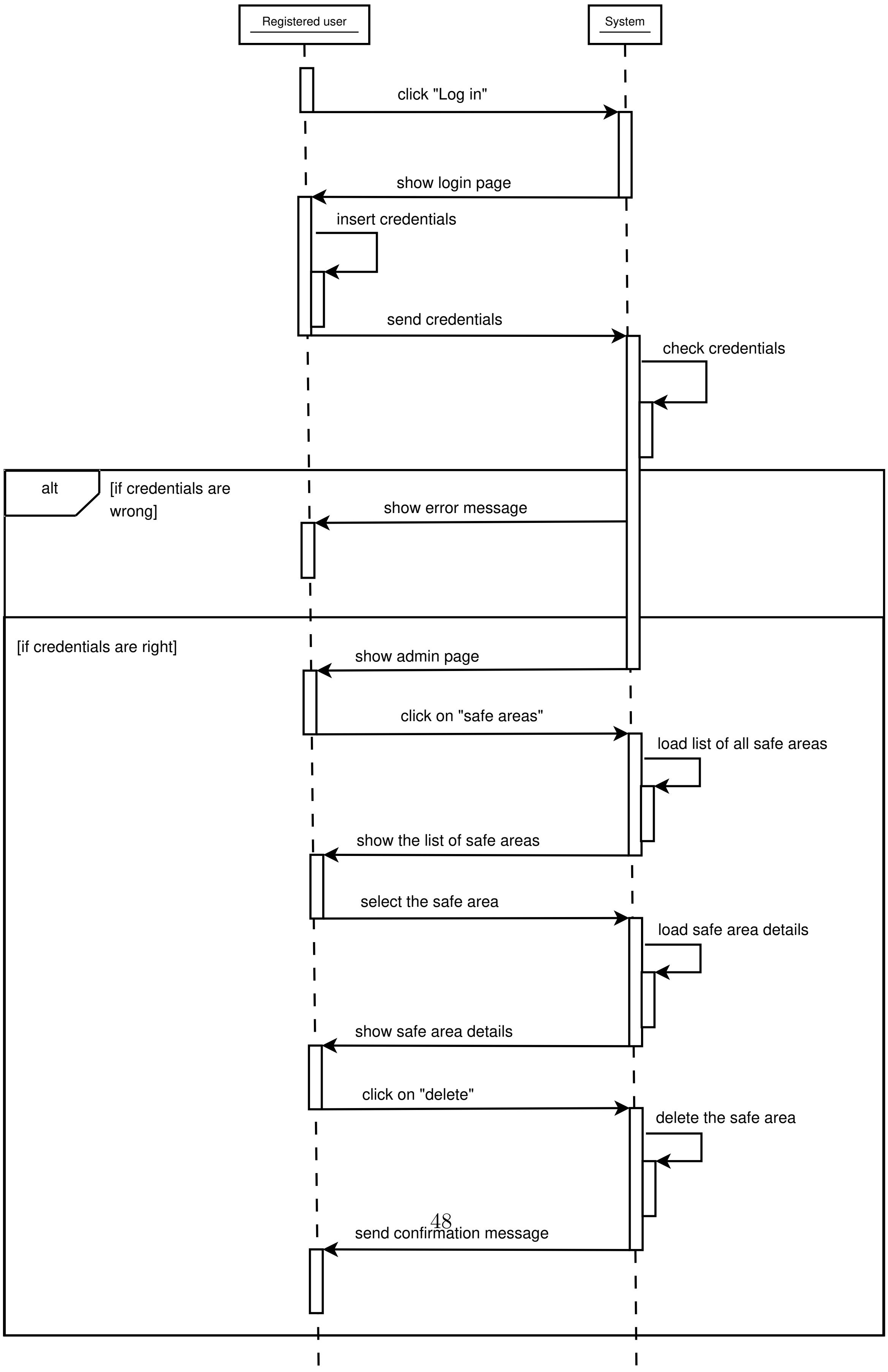
4.4.7 Show the details of the ride


Logged user
System
load personal profile page
show personal profile page
click "My rides"
show rides list
click on the selected ride
show details of the ride

4.4.8 Add new cars



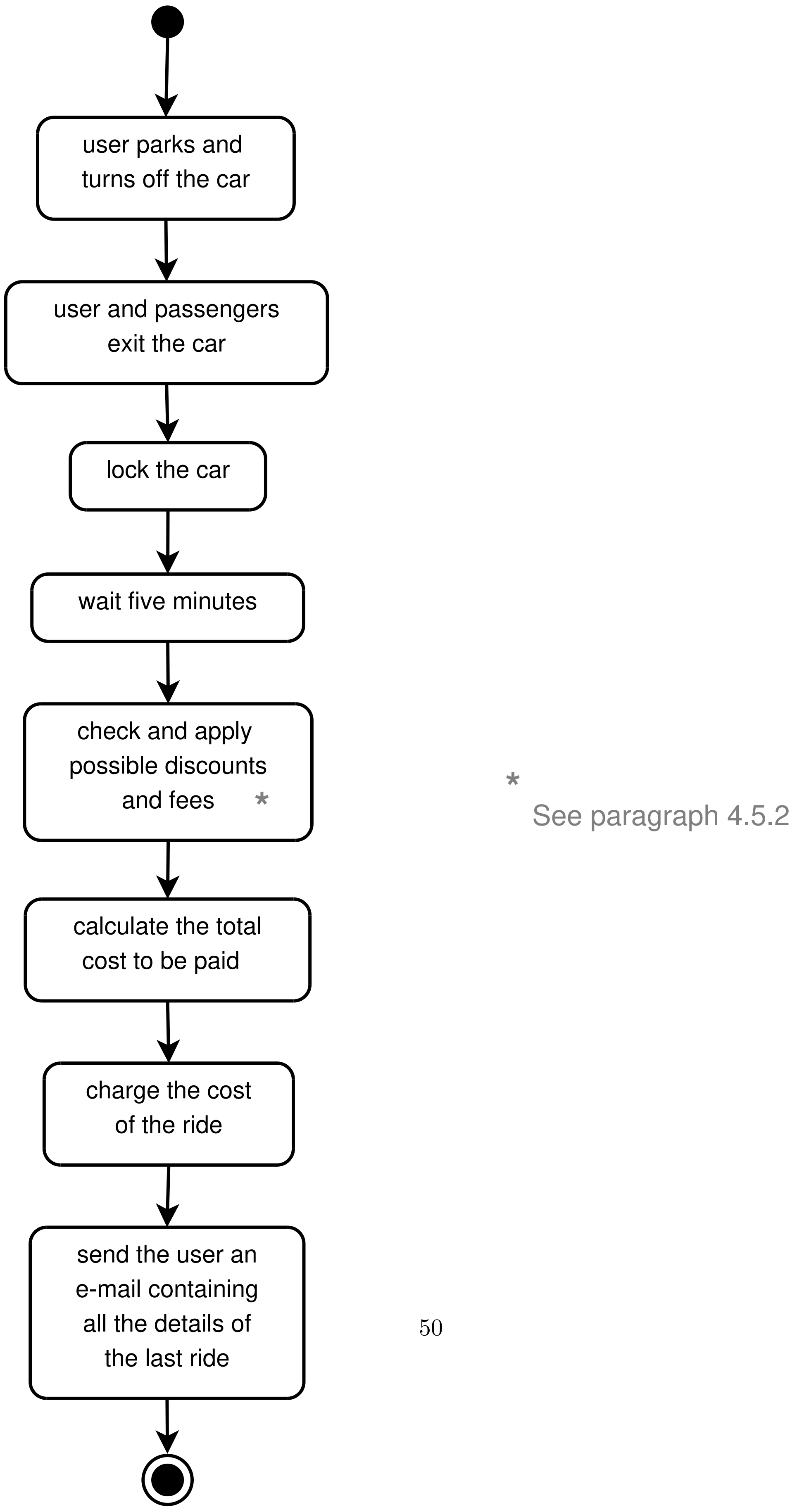
4.4.9 Delete a safe area



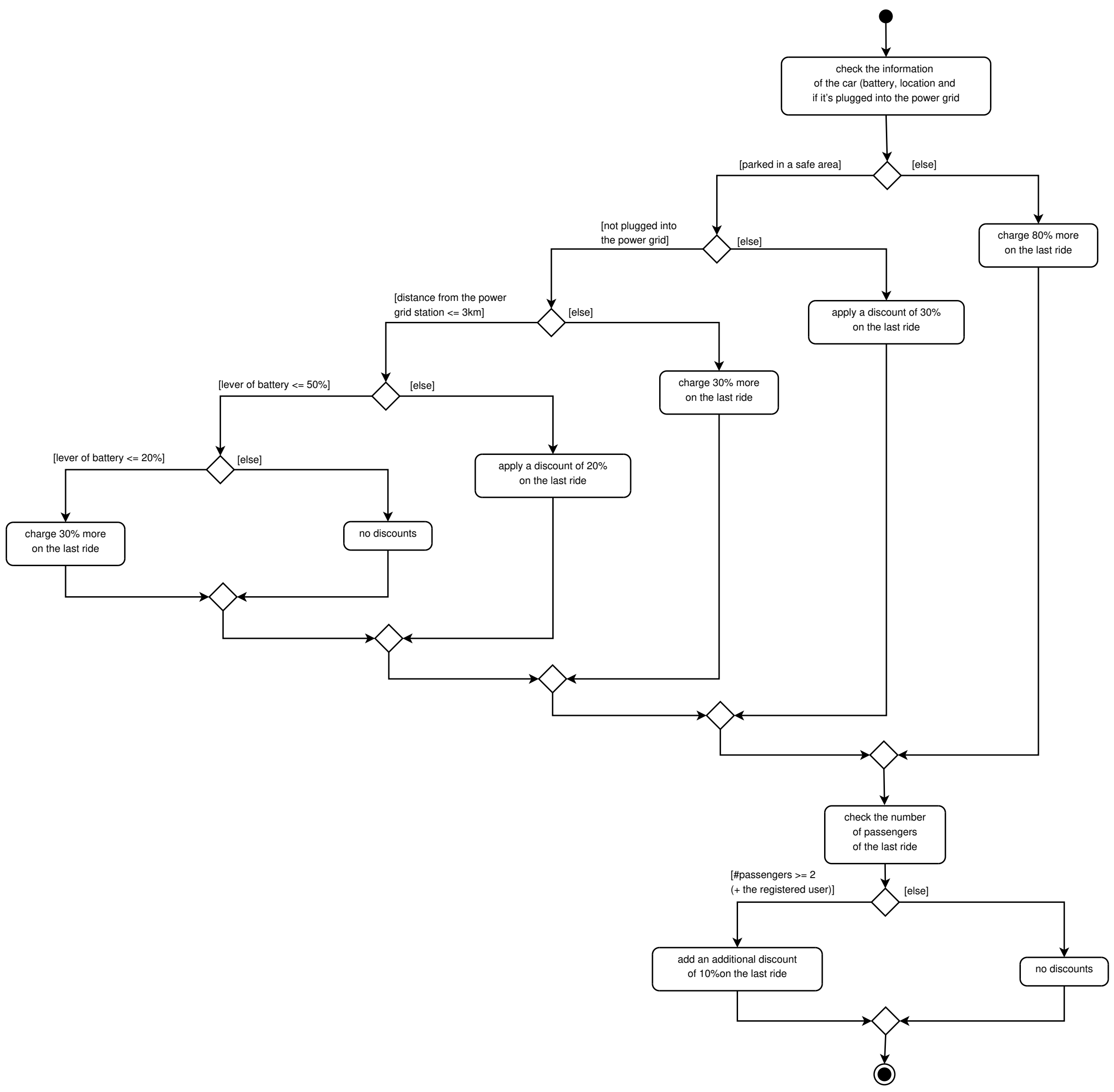
4.5 Activity diagrams

In this paragraph we are going to show the activity diagrams in order to explain the flow of events and activities of some use cases.

4.5.1 End of the ride


user parks and turns off the car
user and passengers exit the car
lock the car
wait five minutes
check and apply possible discounts and fees *
calculate the total cost to be paid
charge the cost of the ride
send the user an e-mail containing all the details of the last ride
* See paragraph 4.5.2

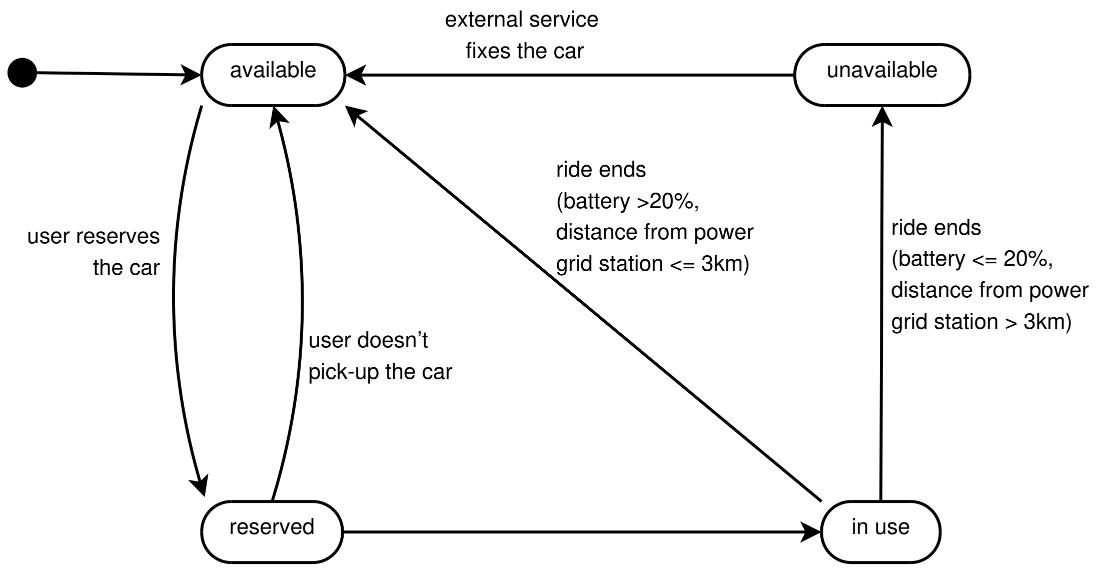
4.5.2 Check and apply possible discounts and fees


check the information of the car (battery, location and if it's plugged into the power grid
[parked in a safe area]
[else]
charge 80% more on the last ride
[not plugged into the power grid]
[else]
apply a discount of 30% on the last ride
[distance from the power grid station <= 3km]
[else]
charge 30% more on the last ride
[lever of battery <= 50%]
[else]
apply a discount of 20% on the last ride
[lever of battery <= 20%]
[else]
charge 30% more on the last ride
no discounts
check the number of passengers of the last ride
[#passengers >= 2 (+ the registered user)]
[else]
add an additional discount of 10%on the last ride
no discounts

4.6 State charts

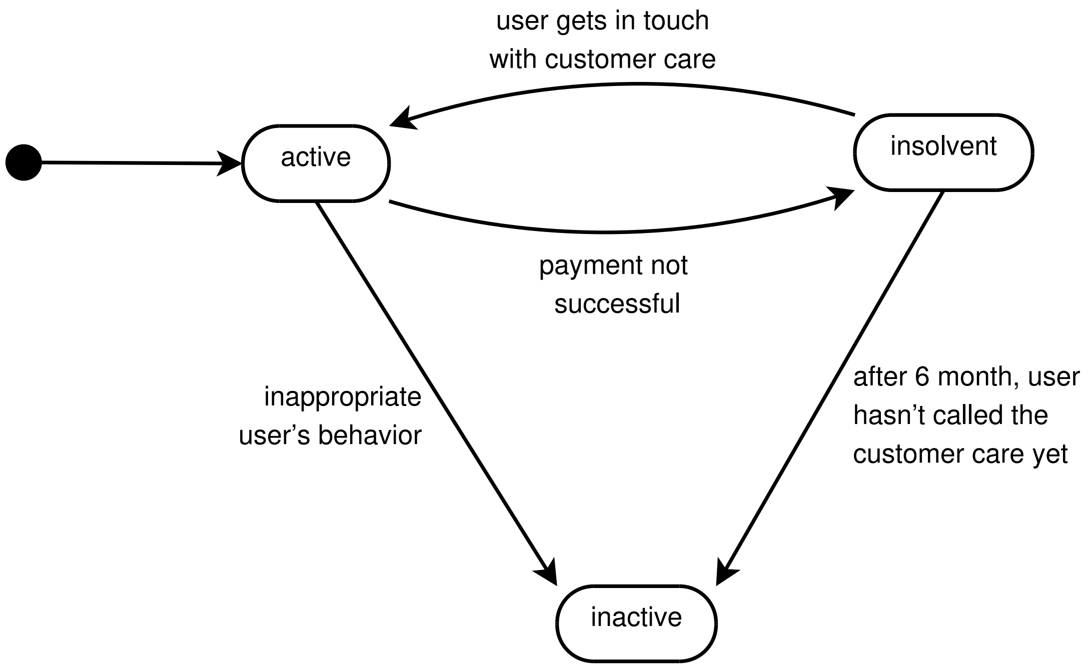
In this paragraph we are going to show two state charts, in order to explain the possible cycle states of cars and of user's accounts.

4.6.1 States of the car



external service fixes the car
available
unavailable
user reserves the car
user doesn't pick-up the car
ride ends (battery >20%, distance from power grid station <= 3km)
ride ends (battery <= 20%, distance from power grid station > 3km)
reserved
in use

4.6.2 States of the user's account



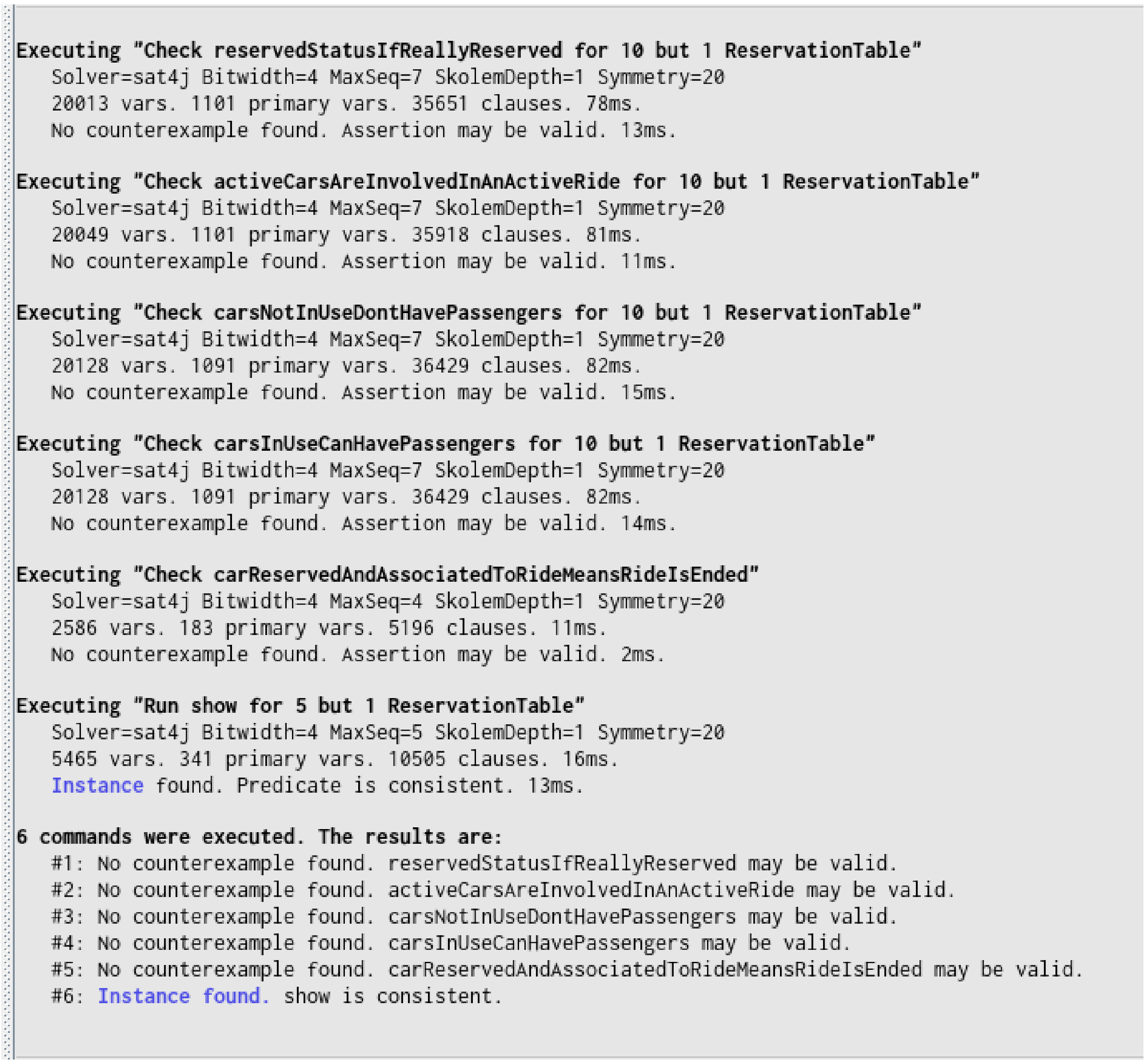
user gets in touch with customer care
active
insolvent
payment not successful
inappropriate user's behavior
after 6 month, user hasn't called the customer care yet
inactive

5 Alloy

```
sig RegisteredUser { position: Position }

sig Car {
    position:       Position,
    passengers:     Int,
    status:         one CarStatus
    }

sig ReservationTable {
    reserve: RegisteredUser -> lone Car
}

sig Ride {
   startPosition:   Position,
   endPosition:     Position,
   user:            one RegisteredUser,
   car:             one Car,
   status:          one RideStatus
}

abstract sig Position {}
sig CurrentPosition, SafeArea, PowerGridStation extends Position {}

/* ------------------ ENUMS ------------------ */

enum CarStatus {
        AVAILABLE,
        UNAVAILABLE,
        RESERVED,
        IN_USE
}

enum RideStatus {
        ACTIVE,
        ENDED
}

/* ------------------ FACTS ------------------ */
// If the car has passangers it means that it's in use
fact onlyInUseCarsCanHavePassengers {
    all c: Car | c.passengers > 0 implies c.status = IN_USE
}

// A user can be associated with just one acttive ride
fact onlyOneActiveRidePerUser {
    no u: RegisteredUser | some r1,r2: Ride | r1.status = ACTIVE and
        ↪ r2.status = ACTIVE
        and r1.user = u and r2.user = u and r1≠r2
}

// Rides should start or ends in PowerGridStations or SafeAreas and
    ↪ not in
// generic CurrentPosition
fact rideStartPoint{
    all r: Ride | all s: Position | s = r.startPosition implies
        s in PowerGridStation or s in SafeArea
}
fact rideEndPoint{
    all r: Ride | all e: Position | e = r.endPosition implies e in
        ↪ PowerGridStation or e in SafeArea
}

// A car is associated with a ride iff is in use
fact carInUseAreRelatedToActiveRides {
    all c: Car | all r:Ride | c.status = IN_USE implies (r.car = c and
        ↪  r.status = ACTIVE)
    all c: Car | all r:Ride | (r.car = c and r.status = ACTIVE)
        ↪ implies c.status = IN_USE
        /*and (r.status = ACTIVE and r.car = c) implies c.status =
            ↪ IN_USE*/
    /*all c: Car, r: Ride | r.status = ACTIVE and c.status = IN_USE
        ↪ implies (r.car = c)*/
    /*no c: Car | some r: Ride | c.status = IN_USE and c not in r.car
        ↪ */
}

// A reserved car is associated to a ReservationTable
fact allReservedCarsAreInTheReservationTable {
    all c: Car | some u: RegisteredUser | c.status = RESERVED implies
        ↪ (c in u.( ReservationTable.reserve ))
}

// A car can be reserved only by one user
fact onlyOneUserReservesCar {
    no c: Car | some u1,u2: RegisteredUser | c in u1.(ReservationTable
        ↪ .reserve)
        and c in u2.(ReservationTable.reserve) and u1≠u2
}

// A reserved car has status 2
fact reservedCarStatus {
    all c: Car, u: RegisteredUser | c in u.(ReservationTable.reserve)
        implies c.status = RESERVED
}

// A car can be driven only by one user
fact onlyOneUserDriveCar {
    no r1, r2: Ride | r1.car = r2.car and r1≠r2
}

// Safe areas and power grid stations are only meant for cars
fact noSafeOrGridForUser {
    no u: RegisteredUser | u.position in SafeArea or u.position in
        ↪ PowerGridStation
}

// Two cars can't have the same position
fact noCarsInTheSamePosition {
   no c1,c2: Car | c1.position = c2.position and c1≠c2
}

// Two users can't have the same position
fact noUsersInTheSamePosition {
   no u1,u2: RegisteredUser | u1.position = u2.position and u1≠u2
}

// A ride exists only if there is a user drving a car
fact noRideWithoutUserAndCar {
    /*no r: Ride | r.     */
}
```

```
// No more than four passengers in a car no less than 0
fact minMaxPassengersInCar {
    no c: Car | c.passengers > 4 or c.passengers < 0
}

// If a car has passengers, then it needs to be in a ride
fact noUnattendedPassengers {
   /*no c:Car | no u:RegisteredUser | c.passengers > 0 and c not in u
       ↪ .(Ride.drive)*/
   no c:Car | some r: Ride | c.passengers > 0 and r.car≠c
    }


/* ------------------ ASSERTIONS ------------------ */
// A1
assert reservedStatusIfReallyReserved {
    no c: Car | some u: RegisteredUser |
        c in u.(ReservationTable.reserve) and c.status ≠ RESERVED
}
check reservedStatusIfReallyReserved for 10 but 1 ReservationTable

// A2
assert activeCarsAreInvolvedInAnActiveRide {
    all r: Ride | all c: Car |
        (r.car = c and r.status = ACTIVE) implies (c.status = IN_USE)
}
check activeCarsAreInvolvedInAnActiveRide for 10 but 1
    ↪ ReservationTable

//A3
assert carsNotInUseDontHavePassengers {
    no c: Car | c.status ≠ IN_USE and c.passengers > 0
}
check carsNotInUseDontHavePassengers for 10 but 1 ReservationTable

//A4
assert carsInUseCanHavePassengers {
    all c: Car | c.passengers > 0 implies ( c.status = IN_USE)
}
check carsInUseCanHavePassengers for 10 but 1 ReservationTable

//A5
assert carReservedAndAssociatedToRideMeansRideIsEnded {
    all c: Car, r: Ride | (r.car = c and c.status = RESERVED) implies
        ↪ r.status = ENDED
}
check carReservedAndAssociatedToRideMeansRideIsEnded


pred show() {}
run show for 5 but 1 ReservationTable
```

5.1 Output



```
Executing "Check reservedStatusIfReallyReserved for 10 but 1 ReservationTable"
   Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
   20013 vars. 1101 primary vars. 35651 clauses. 78ms.
   No counterexample found. Assertion may be valid. 13ms.

Executing "Check activeCarsAreInvolvedInAnActiveRide for 10 but 1 ReservationTable"
   Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
   20049 vars. 1101 primary vars. 35918 clauses. 81ms.
   No counterexample found. Assertion may be valid. 11ms.

Executing "Check carsNotInUseDontHavePassengers for 10 but 1 ReservationTable"
   Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
   20128 vars. 1091 primary vars. 36429 clauses. 82ms.
   No counterexample found. Assertion may be valid. 15ms.

Executing "Check carsInUseCanHavePassengers for 10 but 1 ReservationTable"
   Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
   20128 vars. 1091 primary vars. 36429 clauses. 82ms.
   No counterexample found. Assertion may be valid. 14ms.

Executing "Check carReservedAndAssociatedToRideMeansRideIsEnded"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   2586 vars. 183 primary vars. 5196 clauses. 11ms.
   No counterexample found. Assertion may be valid. 2ms.

Executing "Run show for 5 but 1 ReservationTable"
   Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   5465 vars. 341 primary vars. 10505 clauses. 16ms.
   Instance found. Predicate is consistent. 13ms.

6 commands were executed. The results are:
   #1: No counterexample found. reservedStatusIfReallyReserved may be valid.
   #2: No counterexample found. activeCarsAreInvolvedInAnActiveRide may be valid.
   #3: No counterexample found. carsNotInUseDontHavePassengers may be valid.
   #4: No counterexample found. carsInUseCanHavePassengers may be valid.
   #5: No counterexample found. carReservedAndAssociatedToRideMeansRideIsEnded may be valid.
   #6: Instance found. show is consistent.
```

5.2 Generated Worlds

Here are presented two generated by Alloy according to the model specification. The first one shows a rather simple and general case: there are three cars, one in used by a user and associated to its relative ride, one reseved by another user and the last one available for reservations. The first one on the contarary is focused more on the concept of ride history. We see five rides associated to the same user and to five different cars but only one if this ride is active, meaning that the car is currently in use and the others are just kept for backlog. This is stressed by the fact that some of the cars used in the previous rides are now reserved again.

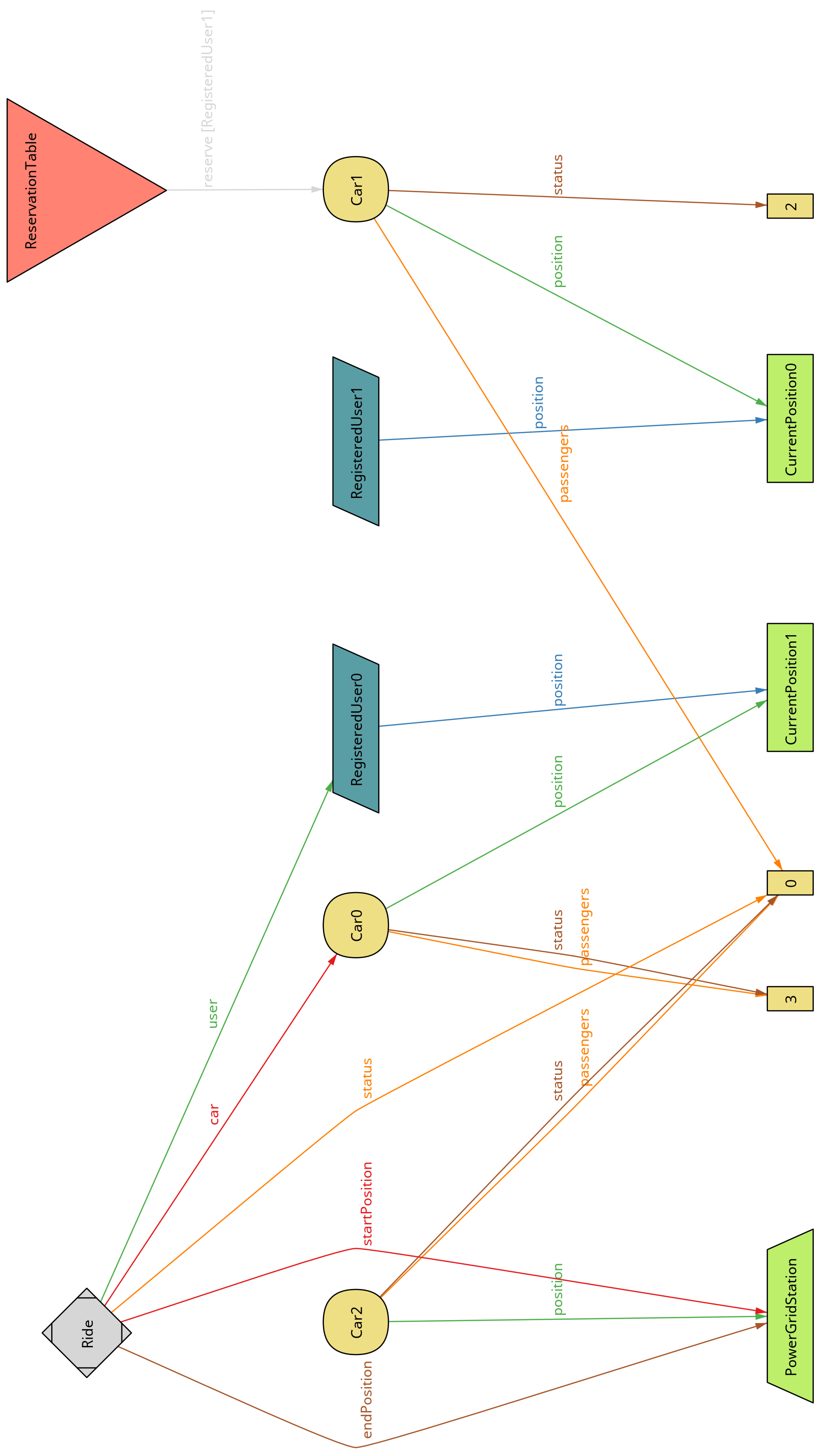


Figure 1: Simple Alloy world.

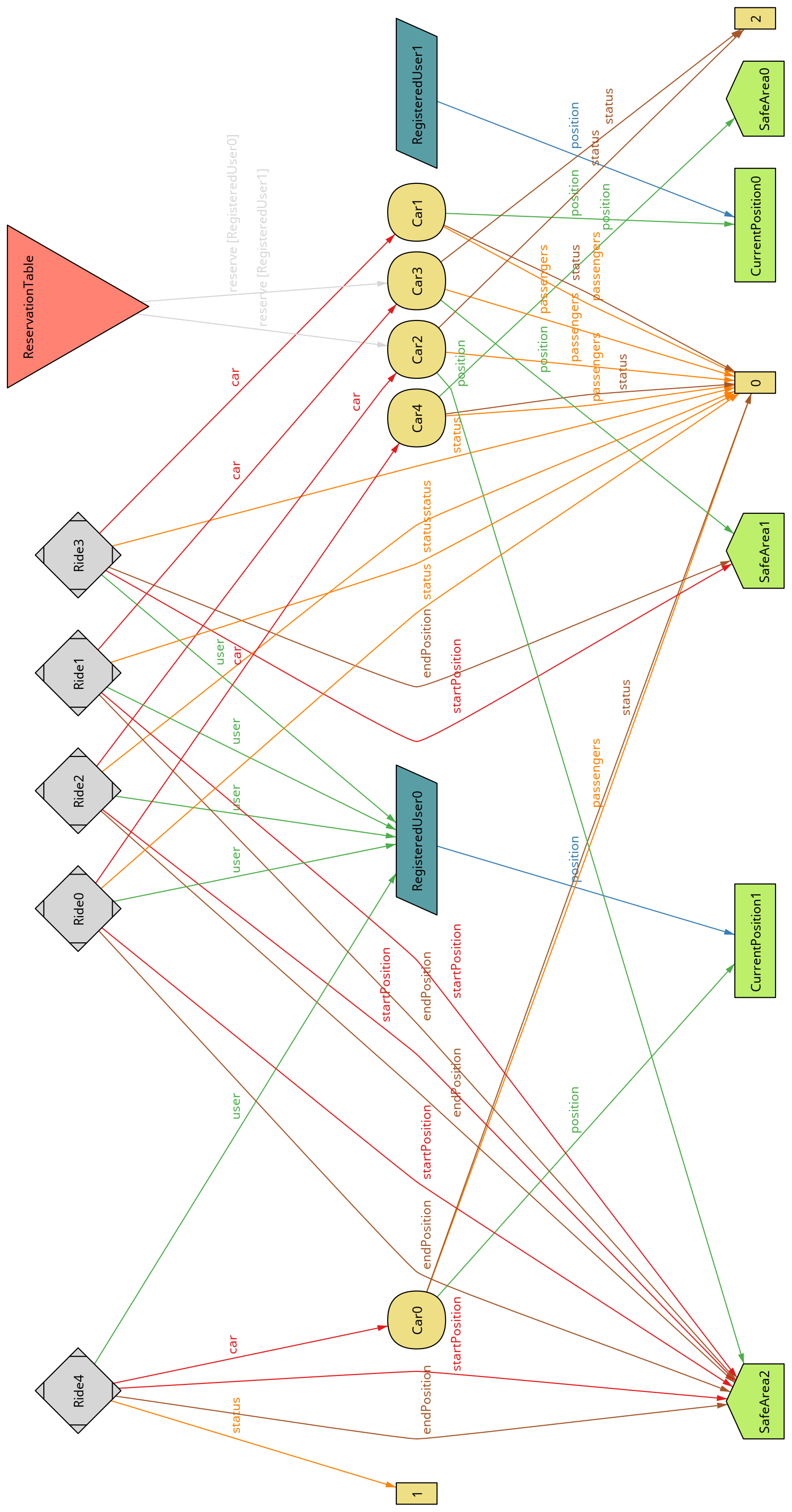


Figure 2: Complex Alloy world with Rides history.

6 Final notes

6.1 Traceability matrix

Raw ID	Goal	Scenario	Use case	UML diagram	Mockup
1	1	3.1	4.3.1	4.4.1	1,2
2	2,3	3.4	4.3.4	4.4.4	4,5
3	4	3.7	4.3.7		
4	5	3.5	4.3.5	4.4.5	5
5	7				7
6	8	3.6	4.3.6	4.5.1	7
7	9	3.8	4.3.8	4.6.2	
8		3.2	4.3.2	4.4.2	1,3
9		3.3	4.3.3	4.4.3	3
10		3.9	4.3.9	4.4.6	1,4
11		3.10	4.3.10	4.4.7	1
12	6,11				7
13	10,13,14,15,16			4.5.2	
14	17	3.12	4.3.12	4.4.9	8
15	18	3.11	4.3.11	4.4.8	8
16	19,20				8

6.2 Used tools

Here is the list of the tools we used to create this document:

- *TeXstudio, Vim (with vimtex plugin) and TeX Live*: to write this doument
- *Dia and Umlet*: for UML models
- *Pencil and Inkscape*: to create mockups
- *Alloy IDE, Vim*: For Alloy modeling
- *Trello and Git*: for working coordination and cooperation

6.3 Working hours

Section	Hours	Author
Introduction	3	Tommaso Sardelli
Mockups	5	Patrizia Porati, Tommaso Sardelli
Specific requirement	17	Patrizia Porati, Tommaso Sardelli
Scenarios identification	11	Patrizia Porati
UML modeling	22	Patrizia Porati, Tommaso Sardelli
Alloy	15	Tommaso Sardelli
Review	9	Patrizia Porati, Tommaso Sardelli
Graphic adjustments	4	Patrizia Porati, Tommaso Sardelli

6.4 Revision history

Version	Date	Authors	Summary
1.0	13/11/2016	Patrizia Porati, Tommaso Sardelli	First version
1.1	11/12/2016	Patrizia Porati, Tommaso Sardelli	Minor fixes
2.0	7/02/2017	Patrizia Porati, Tommaso Sardelli	Add administrator interface