Requirements Analysis and Specifications Document

13 novembre 2016

POLITECNICO
DI MILANO

Version 1.0

- Davide Alessandro Cottica (mat. 806868)
- Stefano Antonino Badalucco (mat. 809322)

Indice

1 Introduction

1.1 Description of the given problem

PowerEnjoy is an innovative service about electric car-sharing, and was born to satisfy growing users' needs about travelling in crowded metropolis (in our case, the city of Milan).
It will be developed as a mobile application and website.
The system will allow registered user to reserve a specific electric car.
This vehicle has to be used by him within one hour from reservation. The overall wage of the service will be decreased, from specific PowerEnjoy bill linked to user, at the end of the travel.
The main purpose of the system is helping users that have particular requests, such as moving in specific places not covered by public transportation or in target time slots where is difficult to find another way of transport. Besides, stimulating electric car usage contribute to lower the pollution level of the city.

1.2 Goals

- [G1] Users will be able to create a profile in the system.
- [G2] Users will be able to log into the system with the credentials provided.
- [G3] Users can reserve a car from the website or the mobile application.
- [G4] System will provide an updated map showing all the available cars in the area.
- [G5] System could show user the available cars from a given address.
- [G6] System will show user all the safe area and the power grid stations.
- [G7] When the user with the specific reservation is near the car, system will provide him a way to unlock this car.
- [G8] User will be able to monitor the amount of money he is going to spend by a screen located in the car.
- [G9] System will detect if there is another passenger by weight sensor located under the seats.
- [G10] System will be able to find cars tracking GPS signal every time.
- [G11] System will be able to accept or refuse a reservation by looking at car reservation status.
- [G12] Cars will be provided with GPS navigators.

1.3 Domain properties

We suppose that these properties hold in the analyzed world :

- All cars must be registered in the database.
- All user willing to use the web application are registered in the database.
- A user must be logged in to see the available cars.
- Car GPS cannot be switched off.
- GPS signal always reveal car position with a gap error of 5 (five) meters from real position.
- A client reserve a car within 1 (one) hour from usage.
- A car cannot be reserved from 2 (two) users or more at the same time.
- A client cannot reserve 2 (two) cars at the same time.
- A car has only one driver.
- Once unlocked, car's door could be open at most in 10(ten) seconds from the time that unlock action has been performed.
- The user who reserves a car is legally responsible for the usage of the vehicle, even if he/she isn't the driver itself.
- Money counter cannot be switched off.
- Car ID is unique and unambiguous and identify a specific car in the database.
- User ID is unique and unambiguous and identify a registered user in the database.
- A car with empty battery cannot be reserved by users.
- All the user logged into the system are registered too.

1.4 Glossary

- Client: he is the actor of the service. At the time of registration he has to insert the following information: Name, Phone number, e-mail, payment information. Then when the password is generated and sent to the client, he has to login to the system with his name and his password.
- PowerEnjoy Car: is the car that will be driven by the client. This car is an electric car that is propelled by electric motors, using electrical energy stored in rechargeable batteries.
- Ride: it starts when car starts and finishes when the engine is turned off by the client. All the rides have a cost that depends on the length and the discount applied.
- Reservation: is the ability to reserve a car for an hour before the time of the ride, so when a reservation is done the car is locked for the time of the reservation and no one can use the reserved car except the person who made it.
- System: is the entity we will create for the management of the activities.
- Safe Area: area where the car's driver has to leave the car when he has finished to use it; the set of safe areas for parking cars is predefined by the management system
- Power Grid Station: is a Safe Area where the car could be put in charge.
- Push Notification: it is a notification sent to a smartphone using the installed related application. For example a push notification could warn user that reservation is expiring.
- Push service: it is a service that allows to send push notifications with own API.
- Low Battery: a car has low battery when its battery is under 10
- API: application programming interface; is a set of subroutine definitions, protocols and tools for building software and applications
- Car sharing: when there are more than two passengers in the car the system will apply a discount on the ride.
- Discount: the discount is applied on the cost of a ride relying on certain factors that will be explained better in future.
- Task: a task is an action or a unit of execution done in an automatically way by the server. An example could be "send a push notification to the client when the reservation is expiring "
- Light car's failure: a scratch on the car body or something comparable that is not capable to prejudice client's ride.

- Heavy car's failure: a car damage that prejudices client's ride (i.e. flat tyre).

1.5 Text assumptions

- We should develop a mobile application that must be able to get the client position through a GPS or ask the position directly to the client.
- The clients of the service are all registered into the system, because this makes easier him/her identification and the payments are faster as the cost of the ride is taken directly from user account.
- We assume that if the reservation is not used the client has to pay a fee.
- We assume that we have a fixed global price per minute.
- We assume that the password of each account is generated randomly by the system at the moment of the registration and it is sent to the client by email.
- We assume that the client could change the email directly through the settings of his account.
- We assume that all the cars are of the same model and each car could carry four passengers.
- We assume that the client could put his phone number instead of his username in the login form.
- We assume that cannot exist two reservations at the same time on the same car.
- All the PowerEnjoy cars are registered into the system.
- Registered users must be able to find the locations of available cars within a certain distance from their current location or from a specified address.

1.6 Constrains

1.6.1 Regulatory policies

The system must ask users about permission to share GPS position and about the processing of sensitive data in respect of federal or government law of user's local country. Users have to accept the terms of agreement offered at the time of contract definition. We remind that GPS position is mandatory for the correct functioning of the application.

1.6.2 Hardware limitations

- Mobile App
  - At least 3G connection and GPS chip integration available on the smartphone.
  - Enough space for app installation.
- Website
  - Computer capable of internet connection.

1.6.3 Parallel operation

System performs concurrent operations in a multi-thread optic, granting multiple users use the service at the same time.

1.7 Proposed system

System architecture will be client-server type. AJAX will grant elevated real-time responsivity of the website/mobile app. An MVC pattern subdivide efficiently work among different layers. In particular (as visible in Fig.) we decided to implement a tier-3 architecture with two firewalls to filter webserver calls when trying to query the central database.

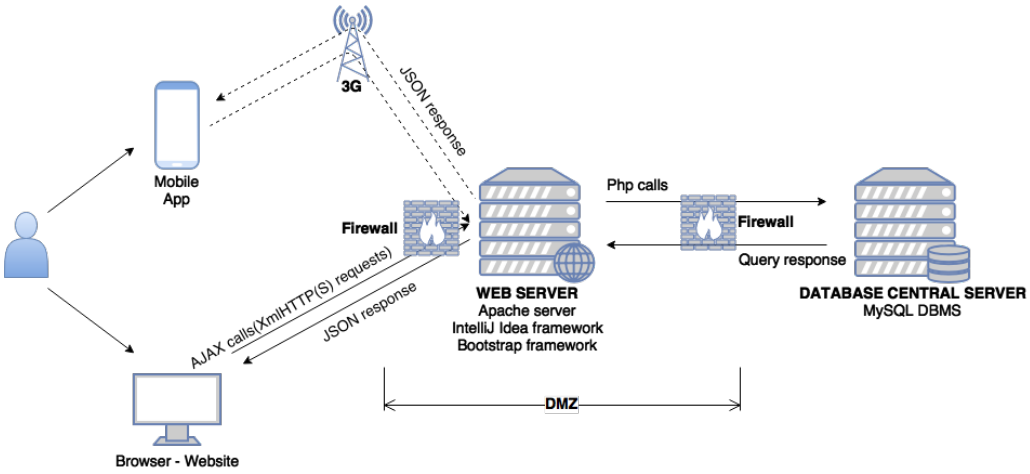


Figura 1: System architecture.

1.8 Identifying stakeholders

Our main stakeholder is a general city governor because system could adapt to various cities' configuration. In particular, this application will be firstly developed for the city of Milan in Italy. We aim to increase furthermore system potential in order to reach at least a regional coverage.

2 Actor identifynig

We have only one actor - the client itself. He needs to register to the system to access its functionality, because we need some information (among these: payment method, email) that must be stored in database.

3 Requirements

3.1 Functional requirements

We will now proceed to explode every goal stated above, minding all the assumptions and respecting domain properties:

- [G1] Users will be able to create a profile in the system:
  - System checks if the string submitted is a valid email address.
  - System checks if the user is trying to register him/herself with an email already stored in the database.
  - System sends an email of confirmation to specified address in order to report the registration.
- [G2] Users will be able to log into the system with the credentials provided:
  - System detects if the password is correct checking from database.
  - System grants access only if the combination username-password is correct.
- [G3] Users can reserve a car from the website or the mobile application:
  - System detects whether the user is using a mobile device or a computer and provides the correct view.
  - System tags reserved car for an hour from request by client and updates the map.
  - System is able to show the user both reservation status and expiration conditions.
- [G4] System will provide an updated map showing all the available cars in the area:
  - Car position is tracked using GPS localization.
  - Available cars are marked with a green tick on a corner. On the other side reserved or unavailable cars are marked with a red cross on the same corner.
  - If users tap or click on a car on the map, the system is able to tell him/her about car status in words.
  - System will disable reservation button of the specified car if it has an active reservation from another user or the user itself.
- [G5] System could show user the available cars from a given address:
  - System shows all cars situated in an affordable walking radius from specified address.
  - Cars are displayed as stated in G4.

- [G6] System will show user all the safe area and the power grid stations:
  - System affixes different marks on the border of both safe areas and power grid stations, to help user identify them easily.
  - System is able to find out safe areas and power grid stations position sending queries to the database.
- [G7] When the user with the specific reservation is near the car, system will provide him a way to unlock this car:
  - System is able to display to the user a button capable to unlock the car only if he/she is in the proximity of the specified reserved car.
  - System is able to determine if a user is 'near' reserved car comparing GPS car with user one.
  - System finds out if the car has been unlocked and provides to disable the unlock button.
- [G8] User will be able to monitor the amount of money he is going to spend by a screen located in the car:
  - System is able to show any moment to the user the total amount accrued from the beginning of the ride.
  - System updates every minute on a screen the amount the user is going to spend.
  - System stops charging the user when he/she press on the start/stop button of the car inside an area marked as 'safe area'.
  - System automatically decreases the amount from the account linked to user profile at the end of the ride.
- [G9] System will detect if there is another passenger by weight sensor located under the seats:
  - System detects user using a weight sensor with at least 20 kg sensitivity.
  - System acknowledge the presence of a passenger only if the sensor triggers itself.
- [G10] System will be able to find cars tracking GPS signal every time:
  - System is able to know second per second the exact position of a car tracking GPS signal, with a maximum gap of 5 meters.
- [G11] System will be able to accept or refuse a reservation by looking at car reservation status:
  - In particular, the system will grant the reservation to a specified car if not only reserved by another user.

  - System will mark a car as unavailable if someone makes a reservation of it.

- [G12] Cars will be provided with GPS navigators:
  - System will be able to grant the user all the functionality of a normal GPS navigator.
  - System will highlight safe areas and power grid stations on the screen of the GPS navigator.

3.2 Non-functional requirements

3.2.1 Client interface - Mobile

Figura 2: SmartPhone's application Home page and menu .

Figura 3: SmartPhone interface.

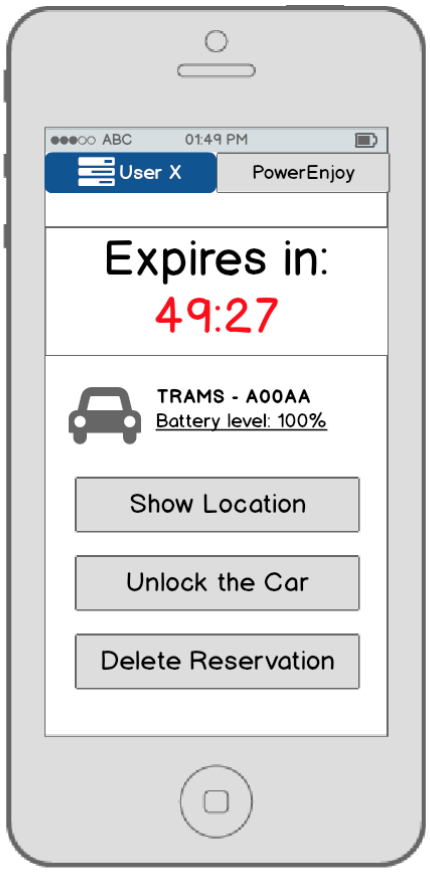


Figura 4: Smartphone interface.

3.2.2 Client interface - Desktop

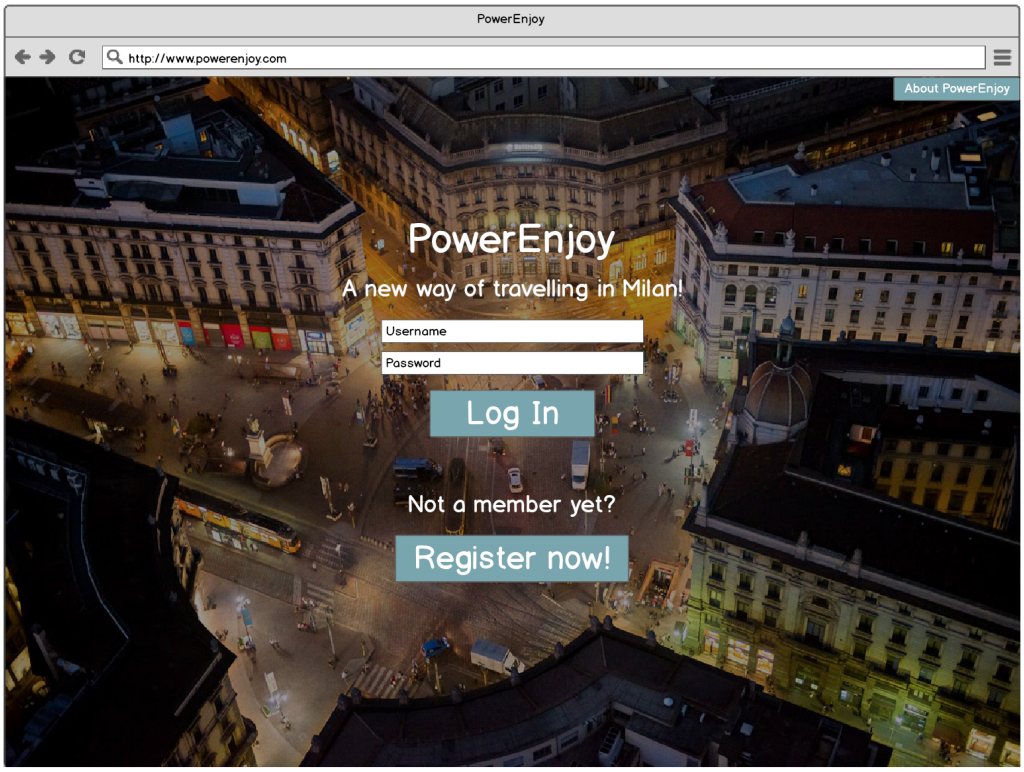


Figura 5: Login page desktop interface.

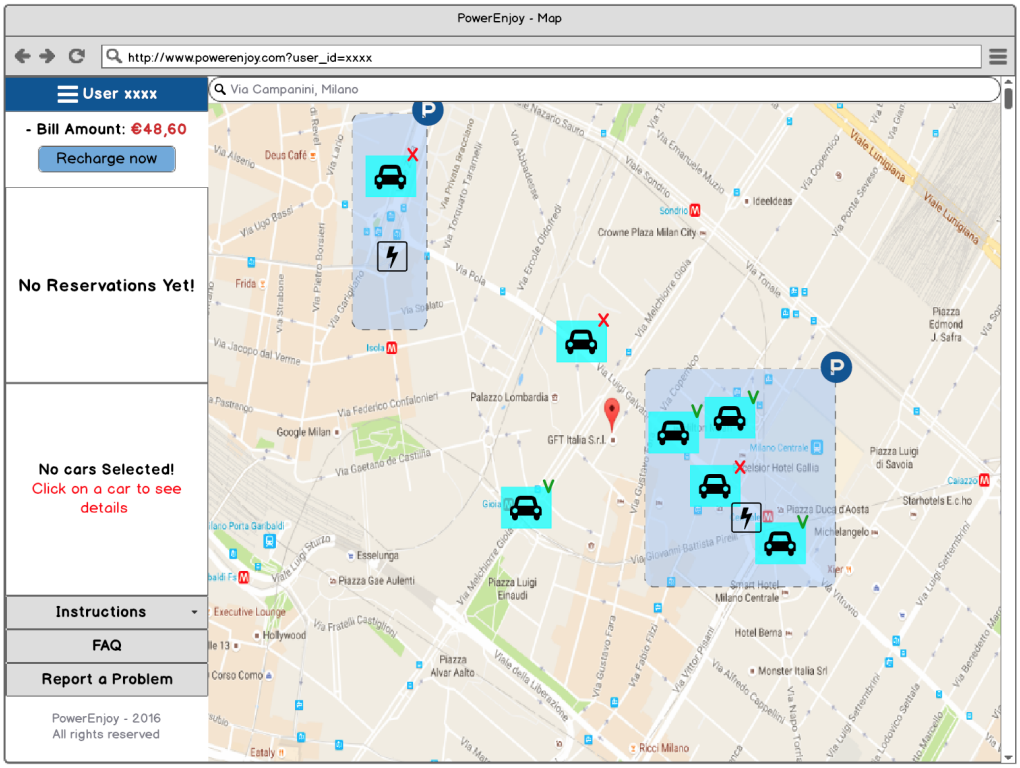


Figura 6: Map page desktop interface.

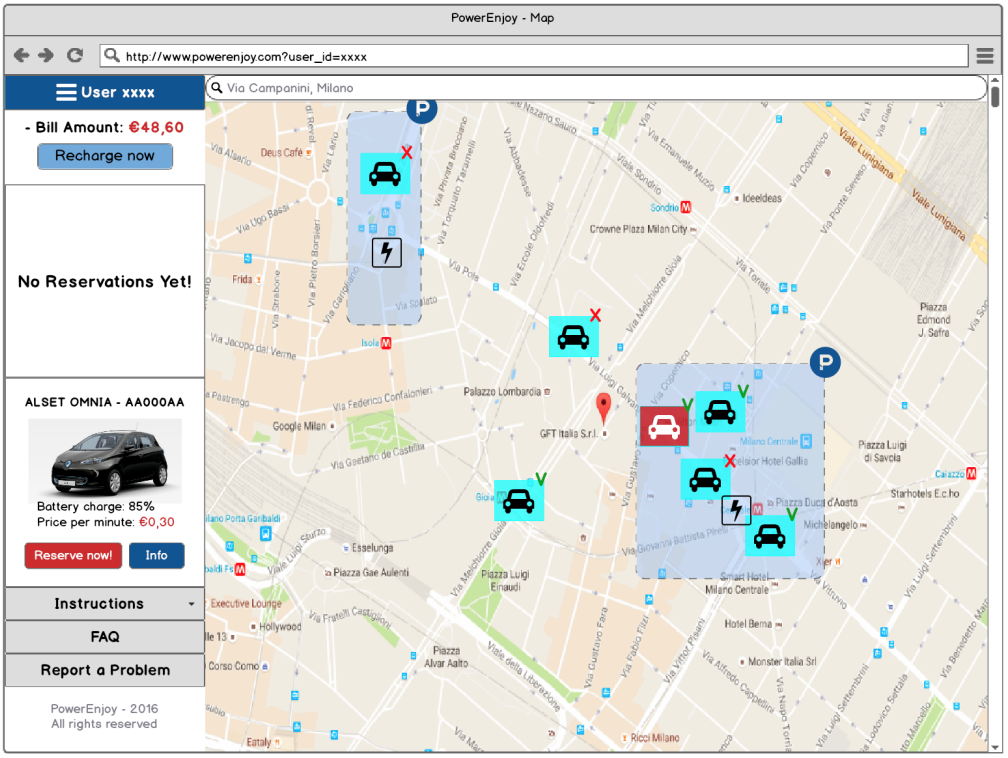


Figura 7: Car selected in desktop interface.

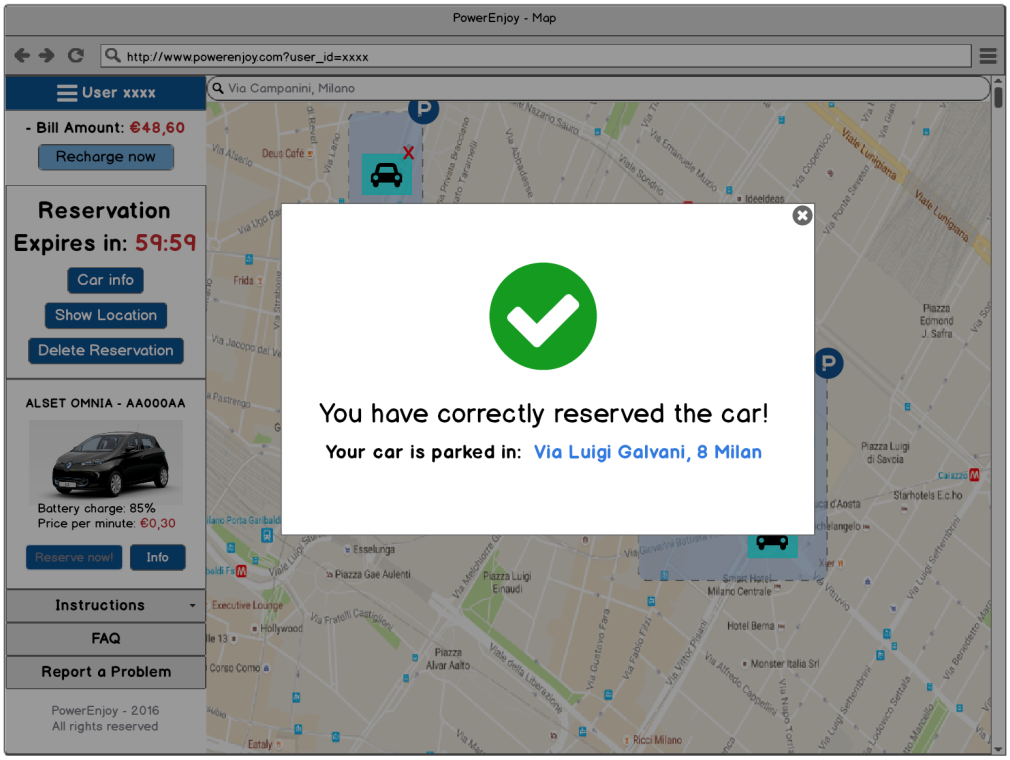


Figura 8: Car reserved in desktop interface.

3.2.3 Architectural consideration

We will build the system as it follows:

- System architecture
  - Apache server with MySQL as DBMS.
  - IntelliJ framework to build PHPs and JS.
  - HTML5 and CSS3 to render responsive web pages, JSON to encode information gathered from DB over HTTP(S).
  - Google Maps integration granted by a Google API Key.
- User requirements
  - Modern browser with javascript and AJAX support enabled (works well with Chrome, Firefox, Safari, Opera).
  - Smartphone with at least Android 4.1/iOS 7.1 operating system installed for Java support and GPS Driver System Software that uses low level API's to communicate with the GPS chip.
  - Internet connection (recommended at least 3G connection type for mobile devices).

4 Scenario identifying

We'll now see some real-world scenarios as everyday situations of people interacting with our system:

1. Stephen from Malnate(VA) wants to book a car in Milan because he has to travel around the city for work. Even if he is aware of the one-hour reservation time limit, he decides to reserve the car when still in Malnate. He opens the app from his smartphone and type via Pirelli, 1 Milan in the 'search for address' section. He finds out a suitable free car in the vicinity of the street, so decides to reserve it. Train is late and the reservation is going to expire, so a notification from the app remembers him that remains only a quarter hour to turn on selected car. Unfortunately, when he reaches target car the hour is expired and 1 euro fee has been charged on his account.

2. Alexander opens the browser and searches for a nearby free car from the PowerEnjoy website. He finds out a car some meters from home, but it is still reserved from another user. The system doesn't let Alexander reserve the car from the app disabling the button. So Alexander book a further car and after some minutes of walk he is in a 10 meters radius from it. The app sends him a notification that the car is now unlockable. So he takes the phone, press on the related button and the car unlocks. Unfortunately, Alexander notices that the car has a flat tyre, so he opens

the app and from the menu press on the 'Report a problem' button. In the section 'heavy car failure', he shortly describe the problem and submits the complaint. A notification informs him that the problem has been submitted with success and reservation for that car has been cancelled.

3. Dave from Milan has successfully reserved a car from the app. When he unlocks the car and press the Start&Stop button, the system starts charging him. He wants to stop for a while so he parks the car, stops the engine and press on the 'Transitory Stop' button. The system detects Dave has moved away from the car and proceeds to lock it. When Dave returns, the 'unlock car' button has been re-enabled in his app, so he can continue his ride. However the system has continued to charge him for the time he was away from the car.

4. Albert Dode has successfully reserved a car from the app. When he unlocks the car and press the Start&Stop button, the system starts charging him. Even if he wants to stop at a certain point but he needs the car for later use, he stops the engine but forgets to press on the 'Transitory Stop' button situated on the screen. So when the system detects Albert far enough from the car, it provides to lock the car permanently as the ride has been considered finish. The system deduct for the amount of the ride and charge the user for an additional fee due to the fact the car wasn't stopped in a safe area.

5. Caroline and Isabel must go to Piacenza (a city in Italy) for work, so Caroline decide to register to our service in order to reserve a car. She successfully reserves a car and enters it with Isabel. The system detects Isabel on passenger side when she sits on the seat and weight sensor triggers itself. Due to this fact, when they both return in Milan on the car and they park it in a safe area, a discount is applied on the total amount subtracted by Caroline's bill.

6. Emmanuel has to go in Locate city (near Milan) to meet some friends of him. He opens the app and searches for nearby cars. When he finds a suitable vehicle he reserves it and decides to have the ride. Once he arrives at the car, he spots some scratch and the sign of a collision on one of the sides. So he opens the app and press on the 'Report a Problem' button. Under the 'Light car failure' section, he provides a short description of the problem, attaching a photo of the event. Due to the fact the car is still operational, he decides to take the ride and meet the friends.

7. Nick is travelling around the city of Milan with one of the PowerEnjoy cars. Current battery level (that is below 30%) pushes a notification on car's screen advising Nick that the car needs as soon as possible to be charged. But shortly after having received the message, Nick is arrived at destination and do not care about plugging the car to the nearest power grid station available. Even if the user is aware about instructions and

terms of use, he decides to pay the additional fee charged on users not caring about plugging low battery car to a power station.

8. Sara opens PowerEnjoy website and decides to take a reservation of an available car. When she finds out a suitable car for her needs, she decides to press on the "Reserve Now!" button on the screen. Unfortunately, at the same time, another user was looking for the same car and pressed the button on his screen approximately at Sara's time, but his request was processed faster. Sara receives a pop-up notification stating this fact and aborting her reservation.

9. Valentina wants to reserve a PowerEnjoy car, but she haven't got an account yet. So she press on the 'Register now!' button on the app and inserts name and email address. Then she receives a mail with the password generated automatically provided to enter in the system. After that, she successfully reserve a car, but unfortunately, in half an hour, another appointment occurs and she has to delete the reservation. System warns the user that a 1 euro fee must be paid to complete the operation.

5 UML models

5.1 Use case diagram

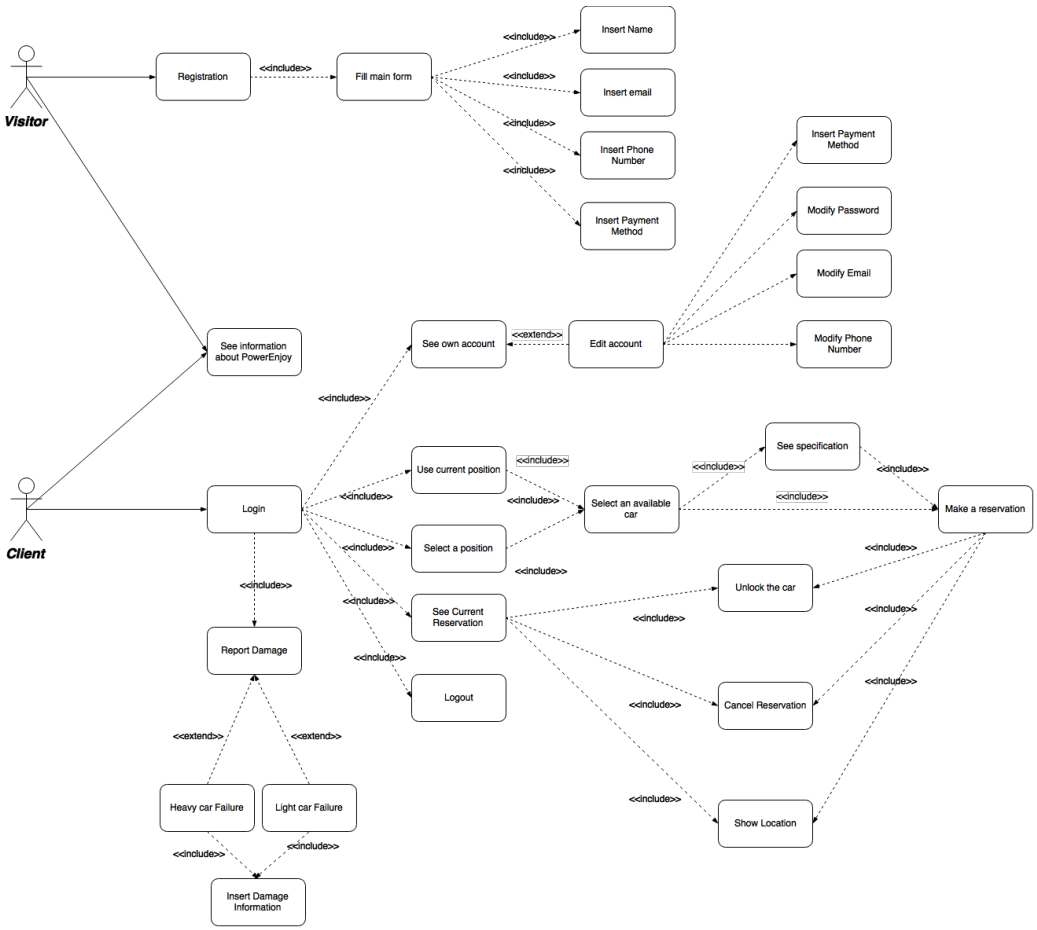


Figura 9: Use case diagram.

5.2 Use case description

In this paragraph we are going to show some use case that can be derived from the scenario and the use case diagram.

Client logs in the application
Name: Client logs in the application.
Actor: Client.
Entry condition: The client must be registered into the system.
Flow of events:

- The client opens the mobile application and reaches the homepage.
- The client clicks on the "Log in" button.
- The client inputs his username and his password.
- The client clicks on the "connect" button.
- The system redirects the client to the main page of the application.

Exit condition: The client is successfully redirected to the main page of the application.
Exception: The password or the username inserted by the client are not correct. The application doesn't redirect the client to the main page but notifies that an error occurs and let him input his username and password again.

Client perform a nearby car's reservation
Name: Client performs a nearby car's reservation.
Actor: Client.
Entry condition:

- The client must be logged into the system.
- The car selected must be free.
- The client has to be on the "map" page.

Flow of events:

- The client selects a car on the map.
- The client clicks on the "Book now" button that will appear over the icon of the car selected.

- The system redirects the client to the reservation page that shows the remaining time of the reservation and some details of the car.

Exit condition: The client is successfully redirected to the reservation page and the car selected will not be more available until reservation expires or the client finishes to use the car.
Exception: The client selects a car that is free but at the same time another client books the car. When the first one clicks "Books now" button the car is no more available. The system notifies that an error occurs and redirects the other client to the "map" page.

Client reserves a car loading an address
Name: Client reserves a car loading an address.
Actor: Client.
Entry condition:

- The client must be logged into the system.
- The address loaded must exist.
- The address loaded must be valid.
- Selected car must be free.
- The client has to be in the "map" page.

Flow of events:

- The client insert the address in the "Type an address" field.
- The client is redirected to an updated "map" page that shows all the cars close to the address.
- The client selects a car in the map.
- The client clicks on the "Reserve now" button.
- The system redirects the client to the reservation page that shows the remaining time of the reservation and some details of the car.

Exit condition: The client is successfully redirected to the reservation page and the car selected will not be more available until client's reservation expires or client stops using the car.
Exception: The client selects a car that is free but at the same time another client books the car. When the first one clicks "Books now" button the car is no more available. The system notifies that an error occurs and redirects the

other client to the "map" page.

Client completes the ride
Name: Client completes the ride.
Actor: Client.
Entry condition:

- The client has started the ride.
- The client is in the car.

Flow of events:

- The client parks the car in a safe zone and turns off the engine.
- The client selects from the car display the option to end the ride.
- The client is notified about the cost of the ride.
- The client leaves the car and the car is blocked.

Exit condition: The system detracts the cost of the ride from the user account and the car is available again.
Exception: The client doesn't have enough money to pay the cost of the ride. The account is blocked until the client charges his account bill or change the payment method.

Client unlocks the car and gets in
Name: Client unlocks the car and gets in.
Actor: Client.
Entry condition:

- The client must be logged into the system.
- The client must have a reservation for the car he wants to unlock.
- The client must be in the reservation page of the car.

Flow of events:

- The client clicks on the button to unlock the car in the reservation page.
- The car is unlocked, the car remains unlocked for a fixed time.
- The client enters the car.

Exit condition: The client successfully unlocks the car and is able to turn on the engine.
Exception:

- The client pushes the button when he is too far, this way the car cannot detect the signal and the car is not unlocked.
- The client wait too much time to enter the car, so the car is locked again.

Client starts a ride
Name: Client starts a ride.
Actor: Client.
Entry condition:

- The client must have a reservation for the car he wants to use.
- The client must have his smartphone with the application installed.
- The client must be in the reservation page of the car and he has unlocked the car.

Flow of events:

- The client enters in the car.
- The client push on the trip computer the button to start the ride.
- The client turns on the engine of the car and the amount starts the count.

Exit condition: The client starts a ride.
Exception: The car is experiencing an issue that could be classified as "Heavy Car Failure" (i.e. mechanical failure) and the user cannot start the ride.

Client does a transitory stop
Name: Client does a transitory stop.
Actor: Client.
Entry condition:

- The client is driving the reserved car.

Flow of events:

- The client parks the car.
- The client push on the trip computer the button to make a transitory stop.

Exit condition: The client exits from the car but the car remains occupied, the ride's count keeps running.
Exception: The client doesn't return from the stop. The ride's count keeps running until the client finish his money, after that the car returns to be free.

Client reservation is expiring
Name: Client reservation is expiring.
Actor: Client.
Entry condition:

- The client has made a reservation for a car.

Flow of events:

- The client makes a reservation for a car.
- The reservation is expiring and 15 minutes before the end of the reservation the system sends a push notification to the client reminding that his reservation is expiring.

Exit condition: The client receives the notification.
Exception: Client's smartphone is no more reachable so it cannot receive the push notification.

Client reservation expires
Name: Client reservation expires.
Actor: Client.
Entry condition:

- The client has made a reservation for a car.

Flow of events:

- The client reservation expires.

Exit condition: The amount of the penalty is subtracted from the user's bill.
Exception: No exception.

Client reports a light car's failure
Name: Client reports a light car's failure.
Actor: Client.
Entry condition:

- The client has made a reservation for a car.

Flow of events:

- The client notices a light car's failure.
- The client reports the problem on the relative page on the application.
- The client starts the ride.

Exit condition: The client reports the problem to the assistance service.
Exception: No exception.

Client reports a heavy car's failure
Name: Client reports a heavy car's failure.
Actor: Client.
Entry condition:

- The client has made a reservation for a car.

Flow of events:

- The client notices a heavy car's failure that prejudice client ride (i.e. flat tyre).
- The assistance service deletes reservation for the car without charging user the cost of the reservation.

- The assistance service removes the potentially broken car from the available cars.

Exit condition: The client reports the problem to the assistance service and his reservation is deleted. The car is removed from the available cars by the assistance service until the problem is solved.
Exception: No exception.

5.3 Class diagram

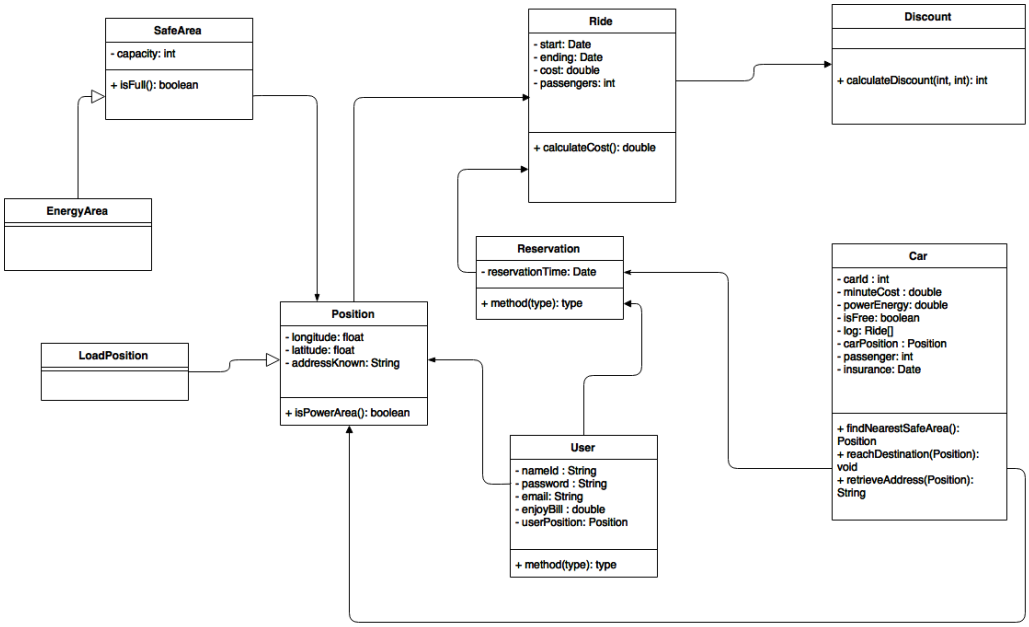
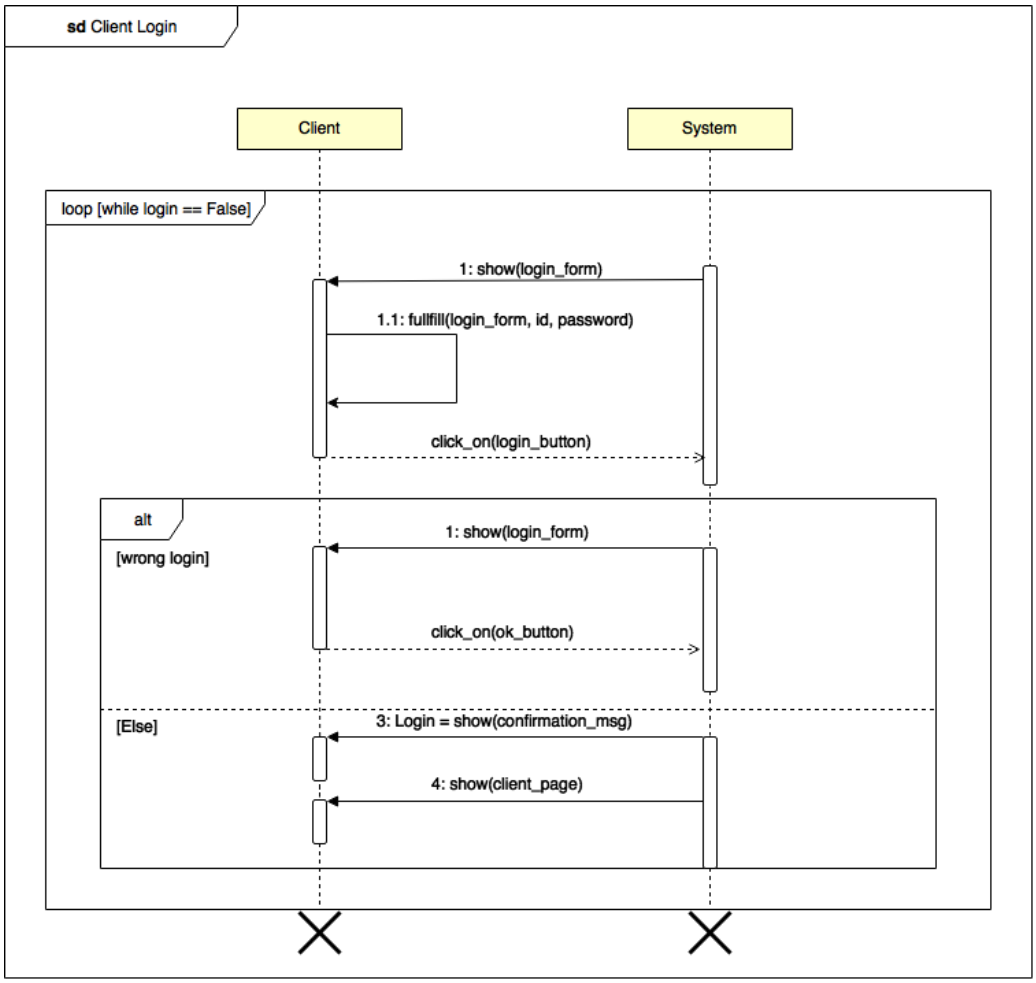
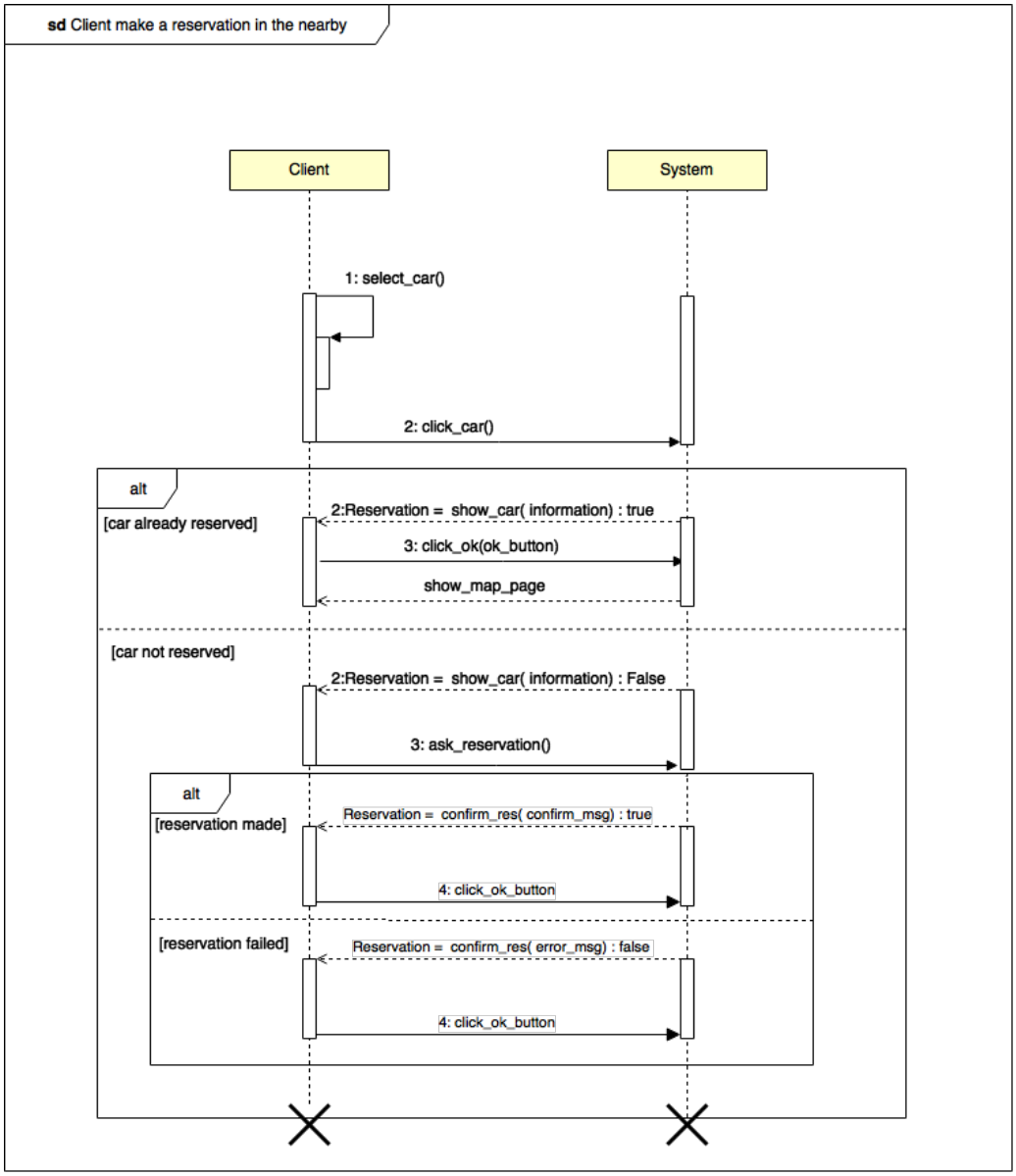
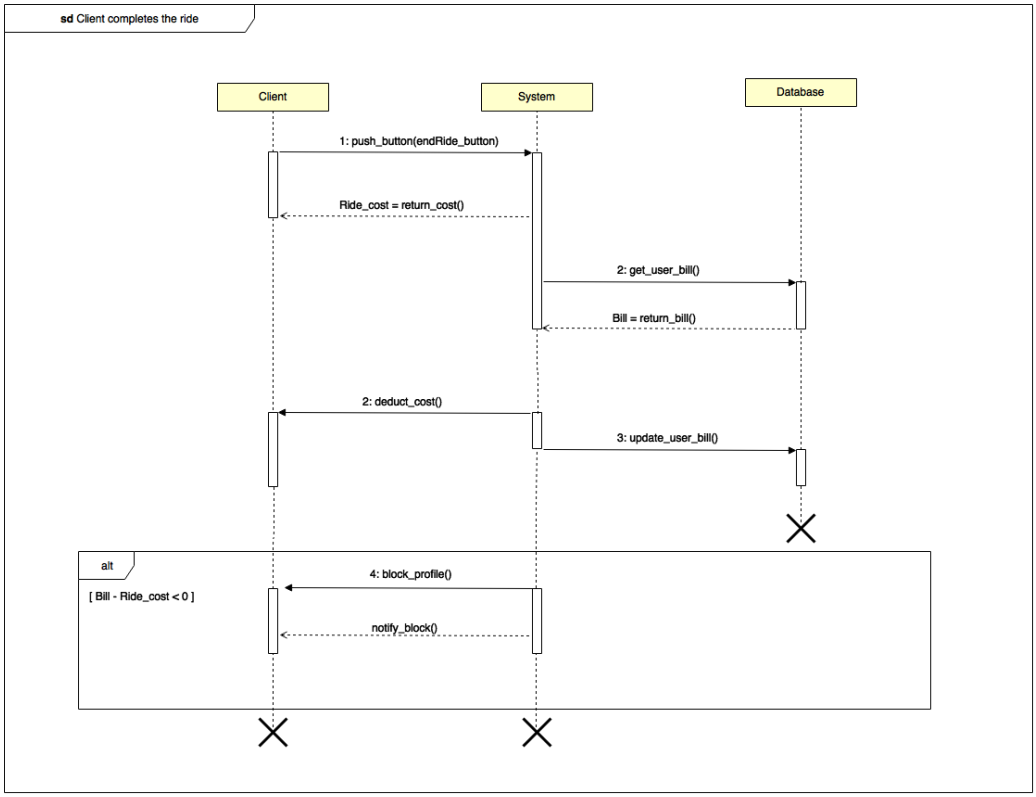


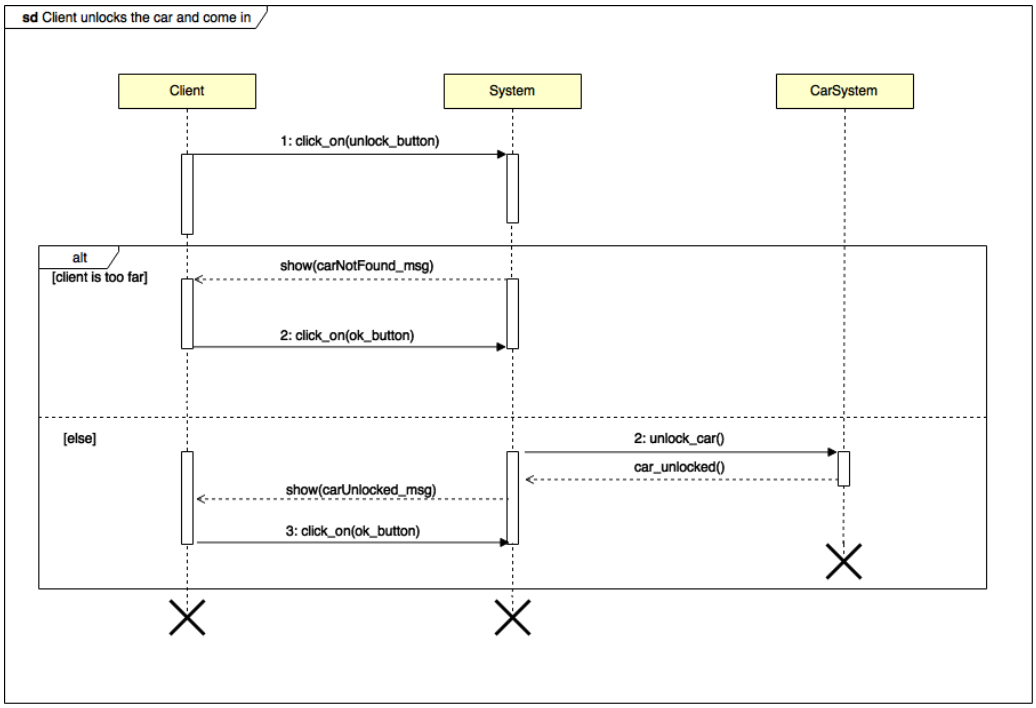
Figura 10: Class diagram.

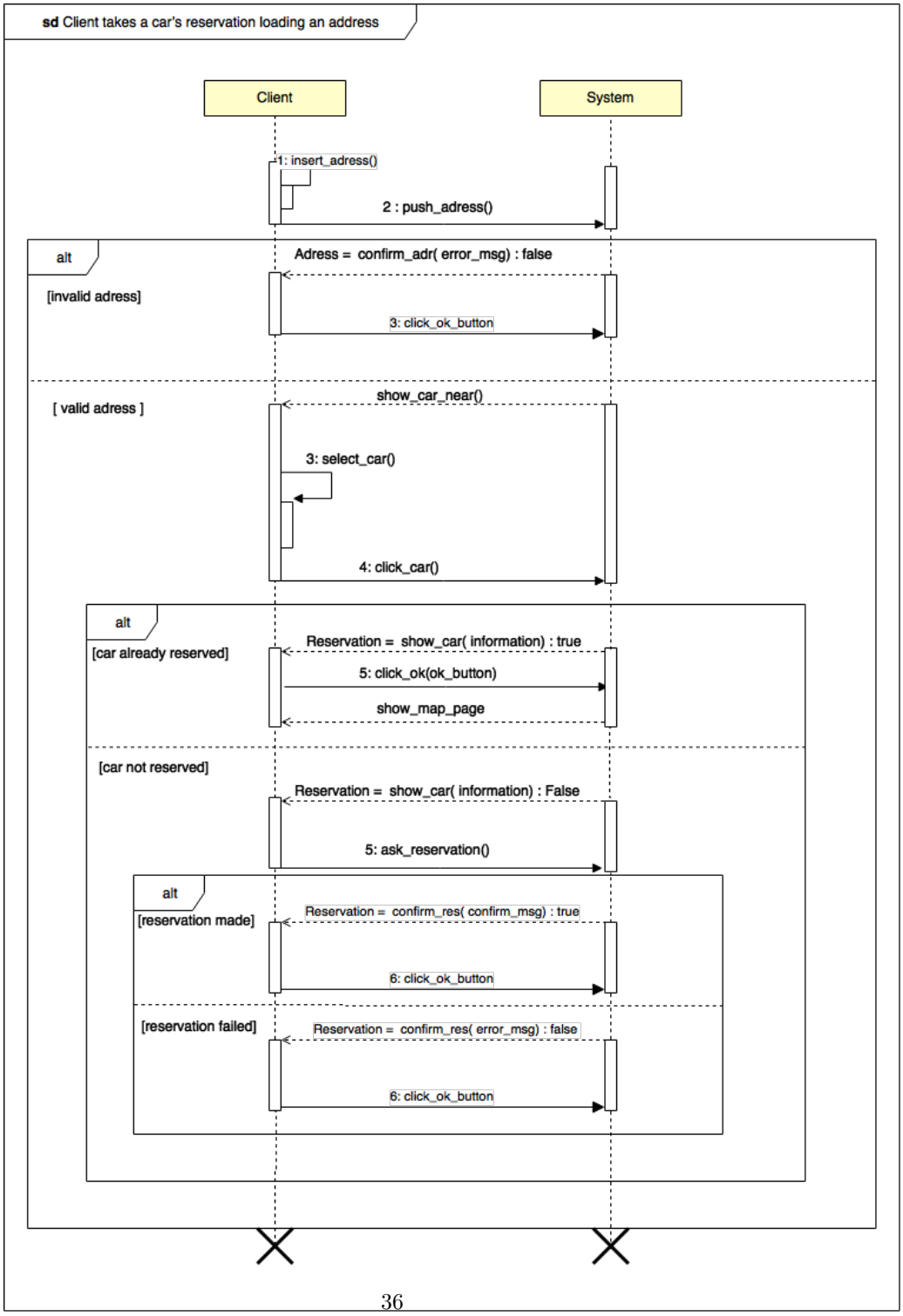
5.4 Sequence diagrams

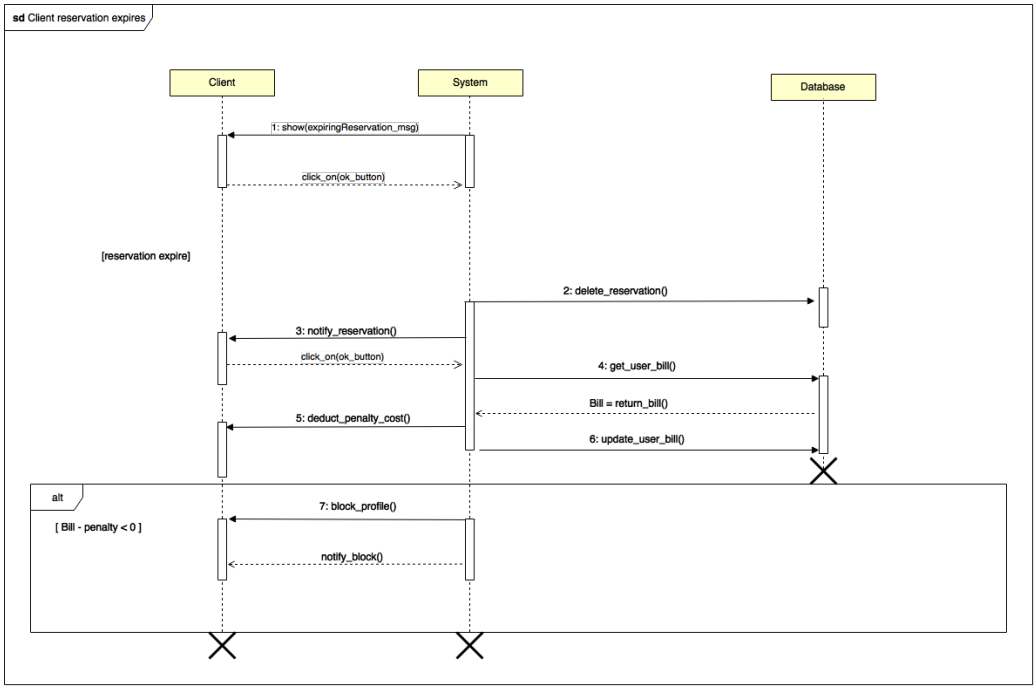


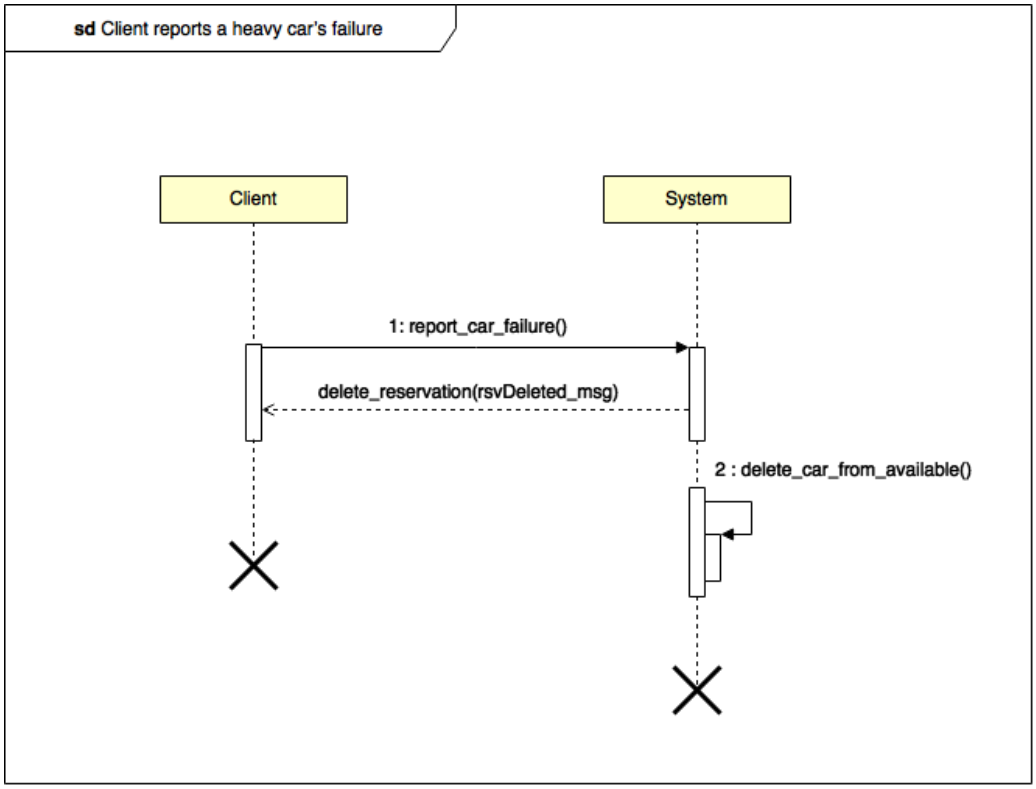

sd Client make a reservation in the nearby
Client
System
1: select_car()
2: click_car()
alt
[car already reserved]
2:Reservation = show_car(information) : true
3: click_ok(ok_button)
show_map_page
[car not reserved]
2:Reservation = show_car(information) : False
3: ask_reservation()
alt
[reservation made]
Reservation = confirm_res(confirm_msg) : true
4: click_ok_button
[reservation failed]
Reservation = confirm_res(error_msg) : false
4: click_ok_button


sd Client completes the ride
Client
System
Database
1: push_button(endRide_button)
Ride_cost = return_cost()
2: get_user_bill()
Bill = return_bill()
2: deduct_cost()
3: update_user_bill()
alt
[Bill - Ride_cost < 0]
4: block_profile()
notify_block()


sd Client unlocks the car and come in
Client
System
CarSystem
1: click_on(unlock_button)
alt
[client is too far]
show(carNotFound_msg)
2: click_on(ok_button)
[else]
2: unlock_car()
car_unlocked()
show(carUnlocked_msg)
3: click_on(ok_button)


sd Client takes a car's reservation loading an address
Client
System
1: insert_adress()
2 : push_adress()
alt
[invalid adress]
Adress = confirm_adr(error_msg) : false
3: click_ok_button
[valid adress]
show_car_near()
3: select_car()
4: click_car()
alt
[car already reserved]
Reservation = show_car(information) : true
5: click_ok(ok_button)
show_map_page
[car not reserved]
Reservation = show_car(information) : False
5: ask_reservation()
alt
[reservation made]
Reservation = confirm_res(confirm_msg) : true
6: click_ok_button
[reservation failed]
Reservation = confirm_res(error_msg) : false
6: click_ok_button


sd Client reservation expires
Client
System
Database
1: show(expiringReservation_msg)
click_on(ok_button)
[reservation expire]
2: delete_reservation()
3: notify_reservation()
click_on(ok_button)
4: get_user_bill()
Bill = return_bill()
5: deduct_penalty_cost()
6: update_user_bill()
alt
[Bill - penalty < 0]
7: block_profile()
notify_block()


sd Client reports a heavy car's failure
Client
System
1: report_car_failure()
delete_reservation(rsvDeleted_msg)
2 : delete_car_from_available()

5.5 Activity diagrams

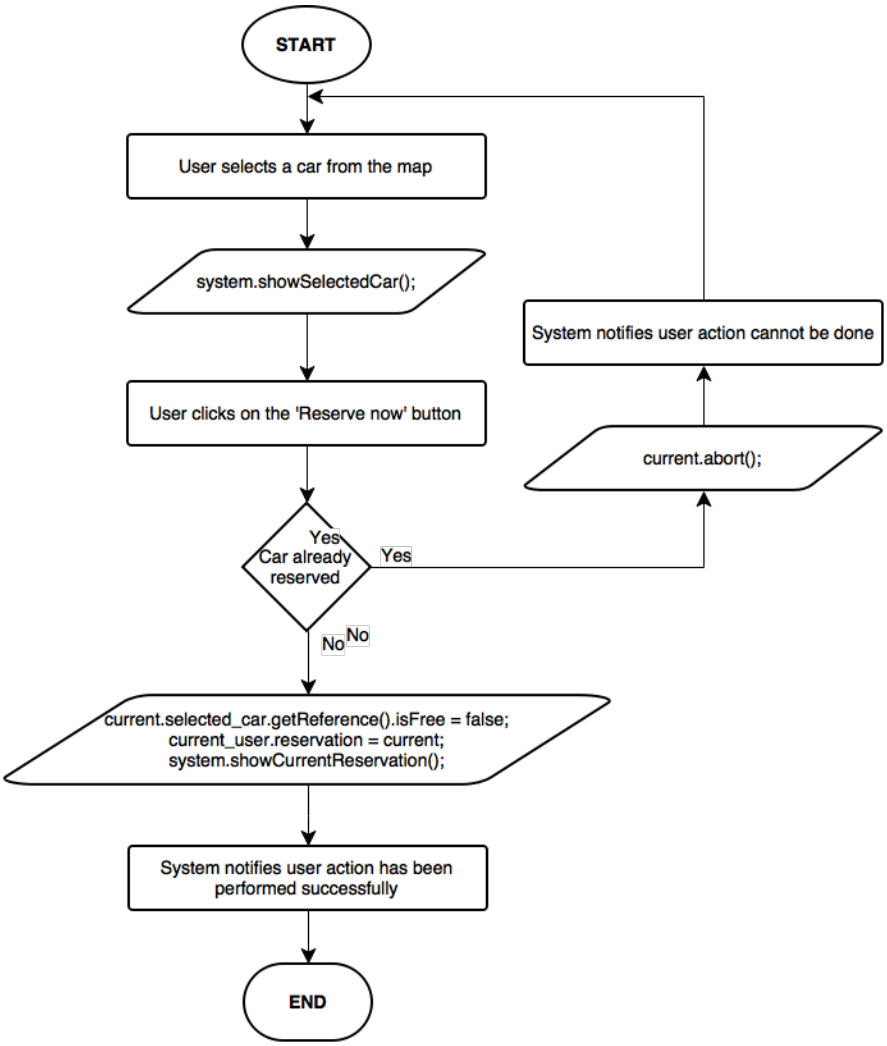


Figura 11: Activity diagram.

5.6 State diagrams

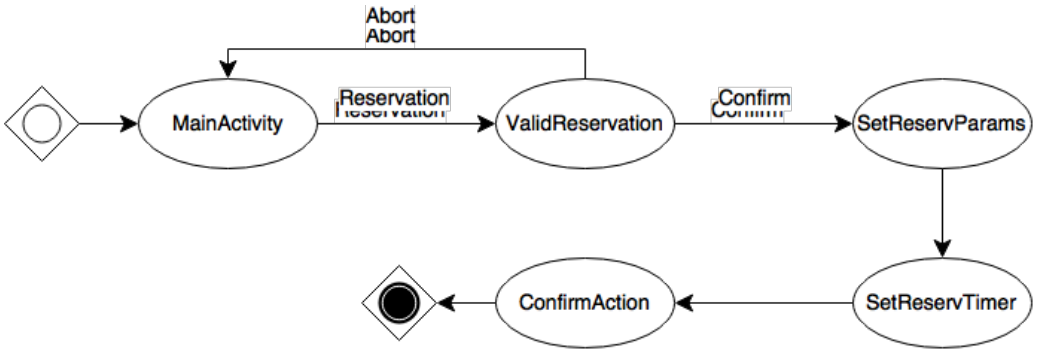


Figura 12: State diagram.

6 Alloy modeling

```
open util/boolean


one sig null {}


sig Car{
        id: one Int,
        minuteCost : Int, // should be double
        powerEnergy : Int,
        isFree: Bool,
        carPosition: one Position,
        carCapacity: Int,

}{
        id > 0
        minuteCost >0
        powerEnergy >=0 && powerEnergy <= 100
}


fact idIsUnique{
        all c1, c2: Car | (c1 != c2) <=> c1.id != c2.id
}


fact carWithoutUserMustBeParked{
        all c: Car|  one s: SafeArea | (c.isFree.isTrue) =>
        ( c in s.carParked )
}




//pendant reservation
sig Reservation{
        carReserved: one Car,
        user: one User,
        reservationTime: one Date,
        startTime: one Date+null,
        endTime: one Date+null,
        totalMinute: one Int,
        passengers: Int,
```

```
        carInSafeArea: Bool, //true if the User leaves the car in an Energy
        cost: Int, // should be double
        discount1: one Discount1,
        discount2: one Discount2,
        discount3: one Discount3,
        discount4: one Discount4,
        totalDiscount: Int, //should be double
        finalCost: Int // should be double
}{
        totalMinute = add[ mul[ ( sub[endTime.hour, startTime.hour] ), 60],
        sub[endTime.minute, startTime.minute] ]
        totalDiscount = mul[ add[add[add[discount1.discountForMinute,
         discount2.discountForMinute], discount3.discountForMinute],
         discount4.discountForMinute], totalMinute]
        cost > totalDiscount
        finalCost = sub[ totalMinute, totalDiscount]
}


fact ifReservationExpiresCostIsOne{
        all r: Reservation|
        sub[r.reservationTime.hour, r.startTime.hour] = 1
        and sub[r.reservationTime.minute, r.startTime.minute] = 0
        => r.cost = 1 and r.startTime = r.endTime
}


fact PassengersLessOrEqualCarCapacity{
        all r: Reservation| one c: Car| r.carReserved = c and
        r.passengers <= sub[c.carCapacity, 1]
}


fact carFreeIfNoReservationsActive{
        all c: Car| c.isFree.isTrue <=> {no r: Reservation|
        r.carReserved = c}
        or {all r: Reservation| one d:Date| r.carReserved = c
         => r.endTime = d }
}


sig Date{
        hour: one Int+null,
        minute: one Int+null,
        day: one Int+null,
        year: one Int+null
```

```
}{
        hour <=24 and hour >=0
        minute >=0 and minute <= 60
        day >= 0 and day <= 365
        year >=0
}


sig User{
        name:one String,
        password:one String,
        email: one String,
        enjoyBill: one Int, // should be double
        userPosition: one Position
}


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


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


pred User.hasActiveReservation[reserv: Reservation]{
        reserv.user = this
        reserv.endTime = null
}


pred Car.hasActiveReservation[reserv: Reservation]{
        reserv.carReserved = this
        reserv.endTime = null
}


fact userCannotReserveTwoCarsSimoultaneously{
        all u: User {no r1,r2: Reservation|
        r1!=r2 and u.hasActiveReservation[r1] and
        u.hasActiveReservation[r2]}
}
```

```
fact noReservationsActiveWithSameCar{
        all c: Car {no r1,r2: Reservation| r1!=r2 and
         c.hasActiveReservation[r1] and
          c.hasActiveReservation[r2] }
}


fact reservedCarActiveIsFalse{
        all c: Car| one r: Reservation|
        c.hasActiveReservation[r] => c.isFree.isFalse
}


sig Position{
        latitude: Int, //should be float
        longitude: Int //should be float
}


sig LoadPosition extends Position{
}


sig DropPosition extends Position{
}


abstract sig Discount{
        discountForMinute: Int
}


//at least two passengers
sig Discount1 extends Discount{
}


fact discount1LessThanTwoPassenger{
        all r: Reservation | (r.passengers < 1)
        => r.discount1.discountForMinute = 0
}


//no more than 50% of the battery empty
sig Discount2 extends Discount{
}
```

```
fact discount2LowBattery{
        all r: Reservation | (r.carReserved.powerEnergy < 50)
         => r.discount2.discountForMinute = 0
}


//car in energyArea
sig Discount3 extends Discount{
}


fact discount3EnergyArea{
        all r: Reservation | r.carInSafeArea.isFalse
        => r.discount3.discountForMinute = 0
}


//car closes to energyArea or with more than 80% of the battery empty,
//this is a negative discount
sig Discount4 extends Discount{
}


fact discount4{
        all r: Reservation | (r.carInSafeArea.isFalse
        and r.carReserved.powerEnergy < 20) =>
        r.discount4.discountForMinute < 0
}


sig SafeArea{
        carParked: some Car,
        position: one Position,
        capacity: Int
}{
        capacity >0
}


fact TheSameCarInOneSafeArea{
        all s1, s2: SafeArea {no c: Car| c in s1.carParked and
         c in s2.carParked}
}
```

```
fact notTooManyCarsParked{
        all p: SafeArea| #p.carParked <= p.capacity
}


sig EnergyArea extends SafeArea{}


pred show(){
        #Reservation = 1
        #User = 1
        #Car = 1

}


run show for 5
```

7 Used tools

The tools we used to create this RASD document are:

- Balsamiq: for mockups
- Github: for version controller
- Draw.io: for diagrams
- Google Drive: to share files
- Alloy Analizer 4.2: for alloy model
- Texpad: to create Latex version of the document

8 Hours of work

8.0.1 Davide Cottica

- 22/10/16: 3h
- 23/10/16: 6h
- 27/10/16: 2h
- 29/10/16: 5h
- 3/10/16: 1h
- 5/10/16: 6h
- 6/10/16: 5h
- 12/10/16: 6h
- 13/10/16: 6h

8.0.2 Stefano Badalucco

- 24/10/16: 2h
- 26/10/16: 3h
- 27/10/16: 3h
- 29/10/16: 3h
- 3/10/16: 2h
- 4/10/16: 4h
- 5/10/16: 5h
- 6/10/16: 4h
- 7/10/16: 3h
- 8/10/16: 2h
- 9/10/16: 1h
- 11/10/16: 3h
- 13/10/16: 5h