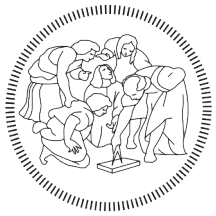

POLITECNICO
MILANO 1863

Requirements Analysis and Specification Document

Version: 1.1

Reda Aissaoui, Jinling Xing, Lidong Zhang

December 8, 2016

Content

1. Introduction

1. Description

The aim of this project is to develop an application for an electric car sharing service company, PowerEnjoy. The company has a fleet of electric cars available all around the city. The city is equipped, by PowerEnjoy, with charging stations for the electric cars. The objective of the use of the electric cars is to offer a good transportation mean that is environment-friendly. In this context, PowerEnjoy wants to motivate its users to carpool. For this reason, discounts are offered to users that have many passengers with them. Discounts are also offered to users that help recharging cars or distribute them evenly in the city. This helps to keep a good service level for the car sharing company.

The application is the true companion of the user as it enables him to reserve cars, find them, check-in and see information about the rides that his makes. Any major person holding a driving license can register with PowerEnjoy using their mobile application or website.

2. Goals

1. Users

- G[1] Allows users to register in the PowerEnjoy application
- G[2] Allows registered users to login using their credentials
- G[3] Allows users to modify their information.
- G[4] Allows users to see the cars around him or around an address on the application.
- G[5] Allows users to reserve cars up to one hour in advance.
- G[6] Allows users to cancel a reservation.
- G[7] Allows users to unlock and check-in the reserved car.
- G[8] Allows users to see how much the previous ride cost along with more ride information.
- G[9] Allows users to check their rides history.
- G[10] Allows users to the user should be able to enable economy mode.
- G[11] Allows users to see where to park the car in order to get discount.

2.System

- G[12] Allows systems to keep real-time data about the car variables.
- G[13] Reservations should time-out if the user doesn't check-in the car.
- G[14] System should calculate the price of the ride depending on the time, left charge in the battery and number of passengers.

3.Operator

- G[15] Allows the operator to validate the identity and driving license of the user after checking them personally.
- G[16] Allows the operator to verify the damaged and faulty cars.
- G[17] Allows the operator can monitor the position of the cars.

3. Domain assumptions

The following always holds in the environment where the application will be deployed.

- The number of cars available in the city are sufficient for the expected demand.
- The number and locations of charging stations are well chosen to suit the density of the city.
- The battery range is sufficient enough to get between two points of the city.
- Every car is equipped a GPS which allows the system to locate its position accurately.
- The GPS of cars cannot be switched off by users.
- Every user has a smartphone always connected to the Internet.
- The position of the user is calculated by his smartphone and is accurate enough to be able to say that he/she is close to the car (Around 5m accuracy).
- Every user has a valid payment to use the cars.
- Every car is not damaged before users reserve it.
- Every user can only reserve/use one car at a time.

- The car is always connected to the management system through 4G/3G.
- Every user registers their account with real identity information that is verified by the operator.
- Every user only registers one account.
- Users rent a car only for their personal use or with friends, but the driver of the car can only be the user who rents the car.
- Cars will be serviced at least once a month to guarantee that all the cars are working.
- Cars report their variables to the system on a real-time basis. This way the information available in the database is always accurate and up-to-date.
- The users having a valid status in the database have a valid payment information.
- We assume that the user can cancel a reservation to make the car available to other users, but will pay the 1€ fee.
- We assume that the city is fully covered with 3G/4G network.
- We assume that the car have a system that exposes an API to check the status of the car (location, battery level, is charging, number of passengers)

4. Glossary

In this section, we define the frequently used words in order to avoid ambiguity. These are the most important concepts used in the documentation of the project.

Client: The physical person that rents the electric car from PowerEnjoy using his smartphone.

Passengers: One or many persons that may be with the client during the ride.

Operator: The PowerEnjoy's employee that supervises the operations and validates driving licenses.

Car: The electric car that is connected to the Internet through 3G/4G. The car has an onboard computer that senses the ignition, battery levels, number of passengers and location and sends them to the application server.

Ride: A travel in the car by the clients and optionally passengers. A ride starts at the moment the client ignites the engine and stops one he leaves the car.

Battery level: The amount of energy left in the car's batteries. 100% being a full capacity battery and 0% and empty battery. The battery level is increased while charging and decreased while the car is traveling.

Charging station: Locations where the cars can be charged by plugging them to the power grid.

Safe areas: Areas in the map defined by PowerEnjoy's management. The clients should take the cars back to these areas at the end of the ride.

Discount: A reduction (expressed in percentages) removed from the total price of the ride.

Car availability: The car have three availability statuses: Available (A), Booked(B), In a ride (R) or Out of service (O).

Available: It is the status when the car is not booked by a user and it is ready to be used (Charged battery, no mechanical problems…).

Booked: It is the status when the car is booked by a user. A car cannot be in the "Booked" status for more than one hour after the user has reserved it.

In a ride: It is the status when the car is being driven by the user.

Out of service: It the exceptional status of when a car has damage or needs maintenance, thus not available for the users.

Car status: The set of variables that describes the status of the car, this includes but is not limited to: battery level, position, mechanical problems, availability (Free, booked, in a ride).

Unverified account: An account that was not verified (Photo ID and driving license) by an operator.

5. System architecture

Our system contains mobile application, WEB application and server. We will implement a client-server architecture based on common REST API and MVC design pattern, so with just one server application we manage both web application and mobile application. The following diagram describes the architecture of the proposed system.

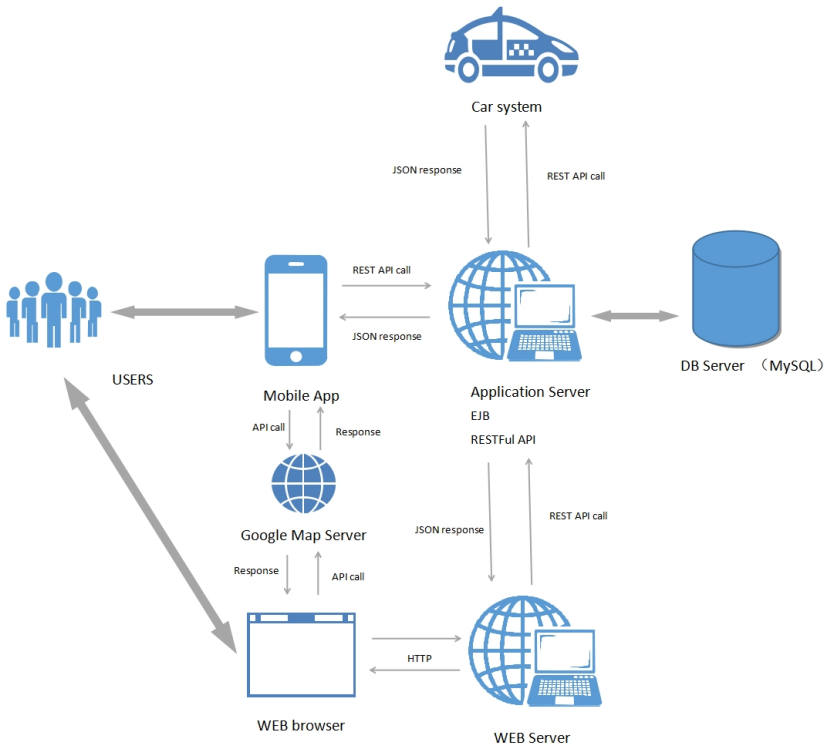

Figure 1: Application architecture

6. Stakeholders

The stakeholder of our application is a company that wants to introduce the car sharing service to a city. The company is sponsored by the government and some private companies as it improves the carbon footprint of the country.

7. Reference documents

- Assignments AA 2016-2017
- Sample Document: RASD sample from Oct. 20 lecture

- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements

2. Actors

The following actors will be using the application:

Visitor

The person that visits the website or the mobile application without being registered. His access to the application is limited.

User

The person that rents the electric cars using the application.The client has a smartphone connected to the Internet and has the mobile application installed in his device. The user has access from both the mobile application and web interface.

Operator

The employee that supervises the operations and verifies the driving licenses.We consider that the employees of PowerEnjoy are all operators. This access grants the user the ability to manage (CRUD) the cars and users. They are the supervisors of the cars fleet. The operator is the agent that takes care if the maintenance of the cars. He can see all the cars variables.

3. Functional and Non-functional Requirements

1. Functional Requirements

- **G[1]** Allows visitors to register in the PowerEnjoy application
  - The visitor is prompted to register in the website or mobile application.
  - The visitor registers to the system by providing their credentials and payment information.
  - The user will receive a password to access the system.
  - The user is prompted to confirm his identity with an operator.
- **G[2]** Allows registered users to login using their credentials
  - The user enters his username and password in the login page.
  - The access is granted to user after verification of his credentials.
  - The access is reject in case of false credentials, wrong payment information or unverified accounts.
- **G[3]** Allows users to modify their information.
  - The user can modify his destination before or after they check-in.
  - The user can change his personal information (except identity and driving license).
  - The user can change his payment information.
- **G[4]** Allows users to see the cars around him or around an address on the application.
  - The system must be able to provide cars locations available according to users GPS location.
  - The system must be able to provide cars locations available according to addresses input by users.
- **G[5]** Allows users to reserve cars up to one hour in advance.
  - The users chooses the car that suits his need.
  - The car is reserved for the given user and cannot be reserved by other users.
  - The reservation is held for one hour after the reservation time.
- **G[6]** Allows users to cancel a reservation.
  - The user choses to cancel the reservation.
  - The user is noticed about the fee of cancelation.
  - The car is available again after the cancellation of the reservation.
- **G[7]** Allows users to unlock and check-in the reserved car.
  - The user that reaches a reserved car must be able to tell the system he is nearby using GPS.
  - The user is notified that he is close and clicks a button to unlock the car.
  - The user can enter the car and ignite the engine.
- **G[8]** Allows users to see how much the previous ride cost along with more ride information.
  - The user can chose a previous ride.
  - The user is shown information about the chosen ride.

- **G[9]** Allows users to check their rides history.
  - The system must be able to save the user rides history in the databases.
- **G[10]** Allows users to the user should be able to enable economy mode.
  - Before making a reservation, the user can enable the economy mode.
- **G[11]** Allows users to see where to park the car in order to get discount.
  - The user enters the address of his destination.
  - The system calculates the best place to park the car to have an even distribution.
  - The user is suggested the place to have a discount.
- **G[12]** Allows systems to keep real-time data about the car variables.
  - Through the car interface, the system keeps the database in an up-to-date state of the car variables.
- **G[13]** Reservations should time-out if the user doesn't check-in the car.
  - The system checks all the current reservations to check if they timed out or not.
  - If a reservation times out, the user is charged and the car is available.
- **G[14]** System should calculate the price of the ride depending on the time, left charge in the battery and number of passengers.
  - Taking in consideration the time of the ride, the battery level and the number of passengers, the system calculates the cost of the ride.
- **G[15]** Allows the operator to validate the identity and driving license of the user after checking them personally.
  - The operator looks for the newly registered user.
  - If the provided photo ID and driving license are valid, the user is marked as valid.
- **G[16]** Allows the operator to verify the damaged and faulty cars.
  - The operator can have a complete overview of the fleet and the status of each car on a map.
- **G[17]** Allows the operator can monitor the position of the cars.
  - The operator can see where each car is.

2. Non-functional Requirements

1. Mock-ups

In the following, we present an overview of how the graphical user interface of the application will look for both the mobile and web applications.

Figure 2: Mobile - Welcome activity

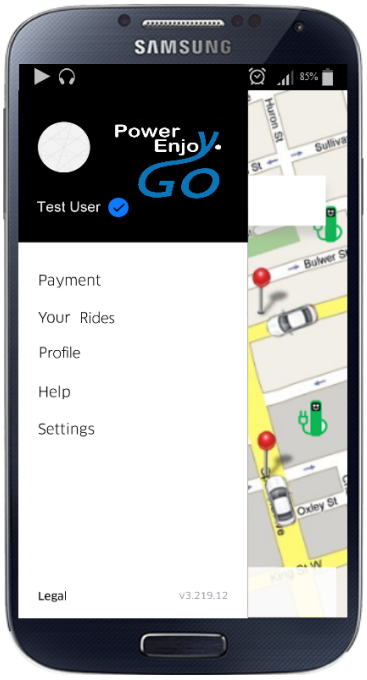

Figure 3: Mobile - User's personal space

Figure 4: Mobile - Main activity

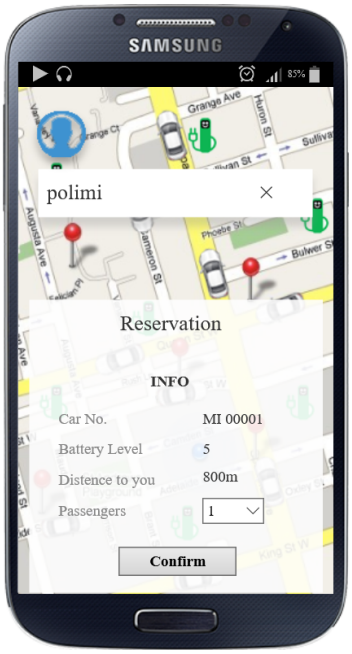

Figure 5: Mobile - Reservation activity

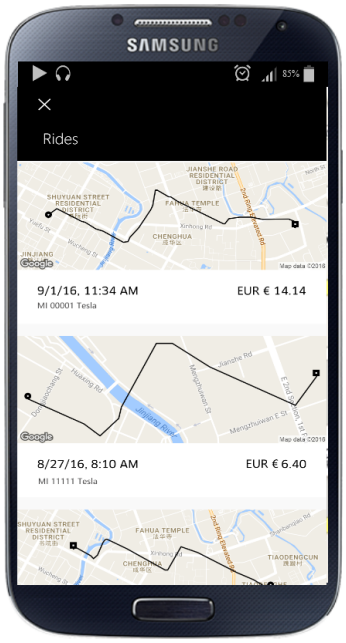

Figure 6: Mobile - History

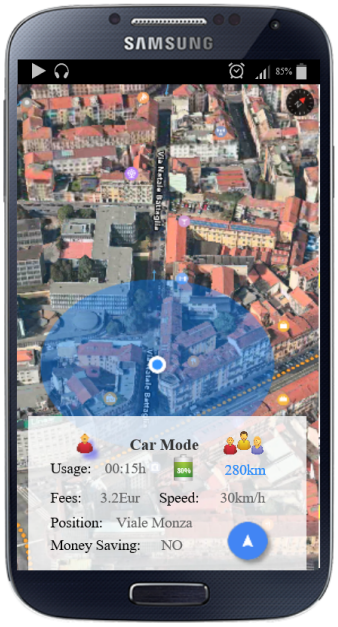

Figure 7: Mobile - Car mode

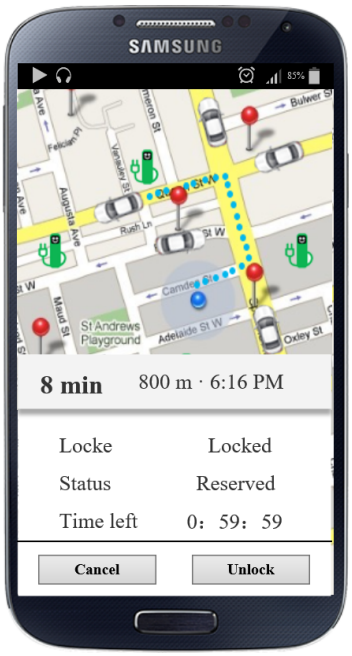

Figure 8: Mobile - Route to the car

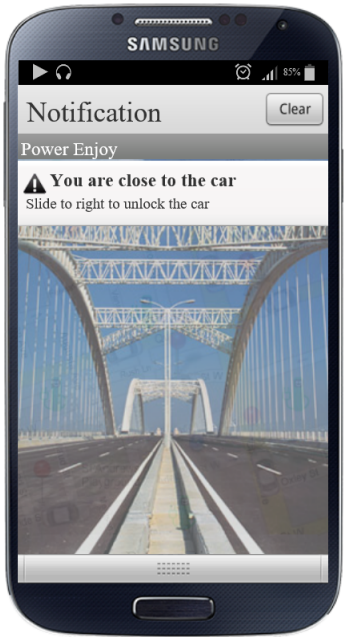

Figure 9: Mobile - Notification to unlock the car

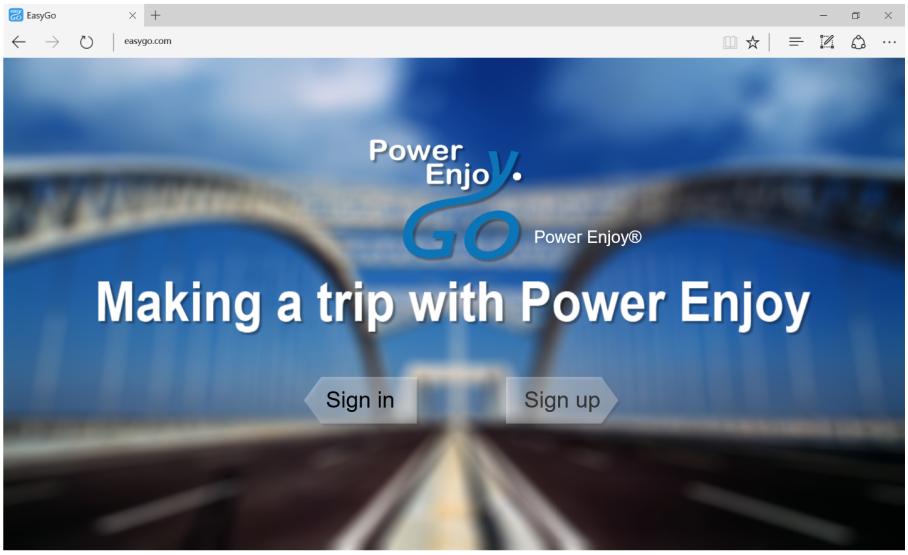

Figure 10: Web - Login page

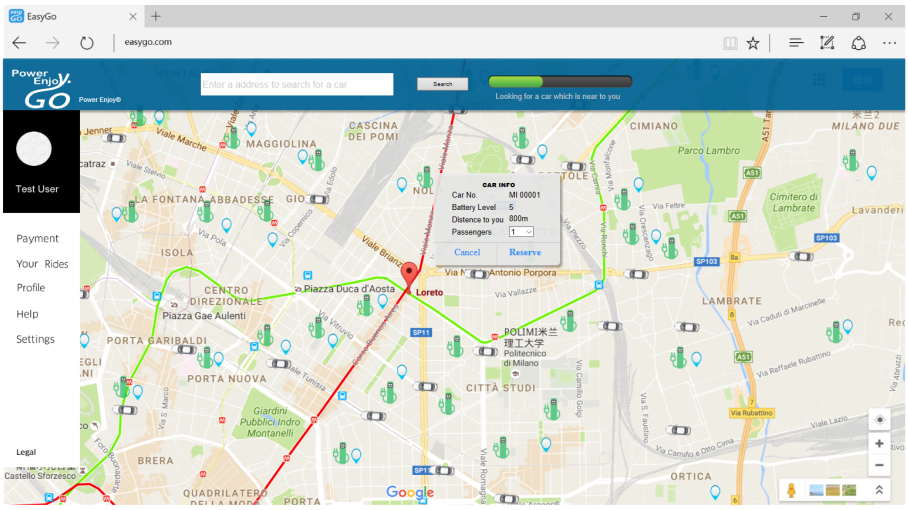

Figure 11: Web - Main page

2. System Quality

- **Performance:** The users will rely on the application to get a car to move around. For this reason, we have to ensure that the application is very reactive and quick.
- **Scalability:** The application should respond properly to the increase of users, usually during commuting hours.
- **Extensibility:** The application should be easily extensible in order to support other platforms for example Windows phones or interface with other applications. For this reason, RESTful web services will be privileged for communication between the application nodes (User's phone, car, server).
- **Privacy and security:** Given the fact that the application holds sensitive information about the users, it should ensure the confidentiality of the information. This will be enforced by the use of SSL for network communications. In addition to that, the passwords should be encrypted using a high-security encryption.
- **Maintainability:** To facilitate the addition of features, the application should be easily maintainable. A well-written code and a complete documentation will be used in order to enforce this point.

3. Technology Enablers

As detailed in *1.6 System Architecture*, our application will follow the 3-tier

client-server application. The application will be composed in this way:

- *Presentation layer:* An Android mobile application and a web application should be used as a graphical user interface.
- *Application layer:* A JEE application running on a Glassfish server will take care of running the business logic of the application.
- *Data layer*: A MySQL server should be used in order to persist all the data that is needed for running the application.

4. Scenarios

Scenario 1

Maria discovers PowerEnjoy through her social media. She is really interested because she occasionally needs a car but she don't want to invest in one. Using the mobile PowerEnjoy application, she registers herself by entering her information and payment details. She went to one of the PowerEnjoy offices to get her account validated by showing her driving license. The account is validated instantly and she is now ready to take her first ride with a PowerEnjoy car.

Scenario 2

John want to go to a furniture and home appliances store to get some new furniture for his apartment. However, he wants to buy so many things that he can't take them with him in public transportation. He checks PowerEnjoy's website and finds a car right next to the store. He reserves it and he is now sure that he will take all his shopping home without trouble.

Scenario 3

Jessy and his friends like to play football during weekends to destress. The problem is that the football field is out of the reach of public transport. Since Jessy is a PowerEnjoy member, he can reserve a car and drive all his friends to the football field. He will even benefit from a discount because he had three passengers with him.

Scenario 4

Maria is a very concerned about the environment and wants to adopt new habits to protect the environment. She knows that electric cars are very environment-friendly but she cannot afford an electric car. With PowerEnjoy she can easily lookup all the electric cars available on a map and reserve one using her smartphone. By using PowerEnjoy for all her commutes, Maria decreases her carbon footprint.

5. UML Diagrams

1. Class diagram

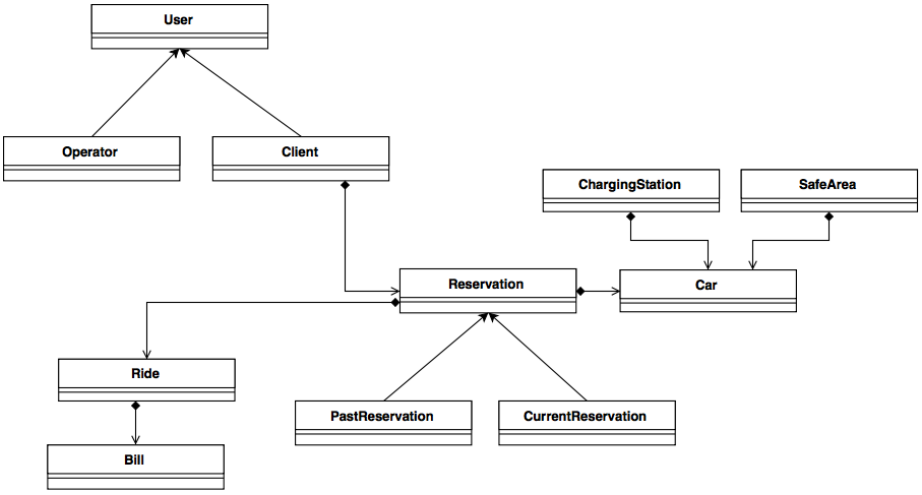

Figure 12: Class diagram

2. Use cases diagrams

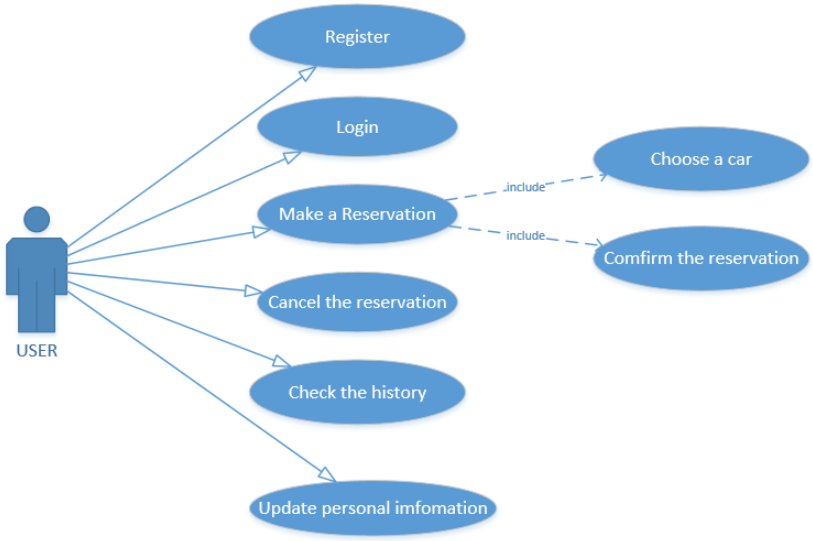

Figure 13: User

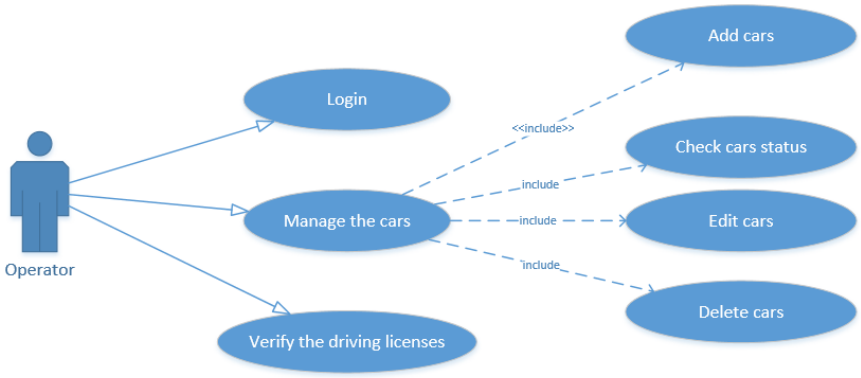

Figure 14: Operator

3. Use cases description

Use cases diagram user

User search the cars

- Name: User requires to take a car
- Actors: User
- Entry requirements:
- User has login.
- User is in the homepage.
- Flow of event:
- User open his GPS to monitor his location or enters an address.
- User can see the cars close to his place along with other information.
- Exit conditions: The system forwards the request to the appropriate car and the use case “User requires to reserve the a car”begins.
- Exceptions:
- User has not open the GPS or not entered and address.
- User is not validated.

User requires to reserve a car

- Name : User requires to reserve a car
- Actors : User
- Entry requirements:

- User has login.
- User is in the homepage.
- Flow of event:
- User click the “Reserve Car” button to make a reservation.
- User chooses the car to reserve.
- The system redirects the user to an information page about the car that includes many information about the car.
- User decides to take a car and click the take car button.
- User sets the “number of passengers” to finish the share car information.
- Exit conditions:
- The users validates the car to reserve.
- Exceptions:
- The chosen car is not available.
- The number of passengers exceeds the capacity of the car.

User requires to cancel a reservation

- Name: User requires to cancel a reservation
- Actors: User
- Entry requirements:
- User has login.
- User is in the reservation homepage.
- User has made a reservation.
- Flow of event:
- User cancels the reservation.
- Exit conditions:
- The system forwards the request to cars which has already been cancelled.
- The system forwards the page that the reservation cannot be cancelled.
- Exceptions:
- User has taken the car and checked-in.

Use cases diagram operator

Operator requires to login

- Name: Operator requires to login
- Actors: Operator
- Entry requirements:
- There are no entry requirements.
- Flow of event:
- The operator inputs his operate code (his ID) and his password.
- The operator clicks on the log in button.
- The system redirects the operator to the operate page.
- Exit conditions:
- The operator is successfully redirected to the operate page.

- Exceptions:
- The code and password furnished by the operator are not correct.

Operator manages the system

- Name: Operator manages the system
- Actors: Operator
- Entry requirements:
- The operator has login and has got the operation role.
- Flow of event:
- The users checks the information about cars.
- Exit conditions:
- The operator made sure that all the cars are working.
- Exceptions:
- None

Operator verifies information

- Name: Operator verifies information
- Actors: Operator
- Entry requirements:
- The operator has login and has got the operate authentication.
- Flow of event:
- Verify the driving license.
- Verify the identity of the drivers
- Exit conditions:
- Mark the user as valid if the information is verified.
- Exceptions:
- The information provided by the client is false.

4. Sequence diagrams

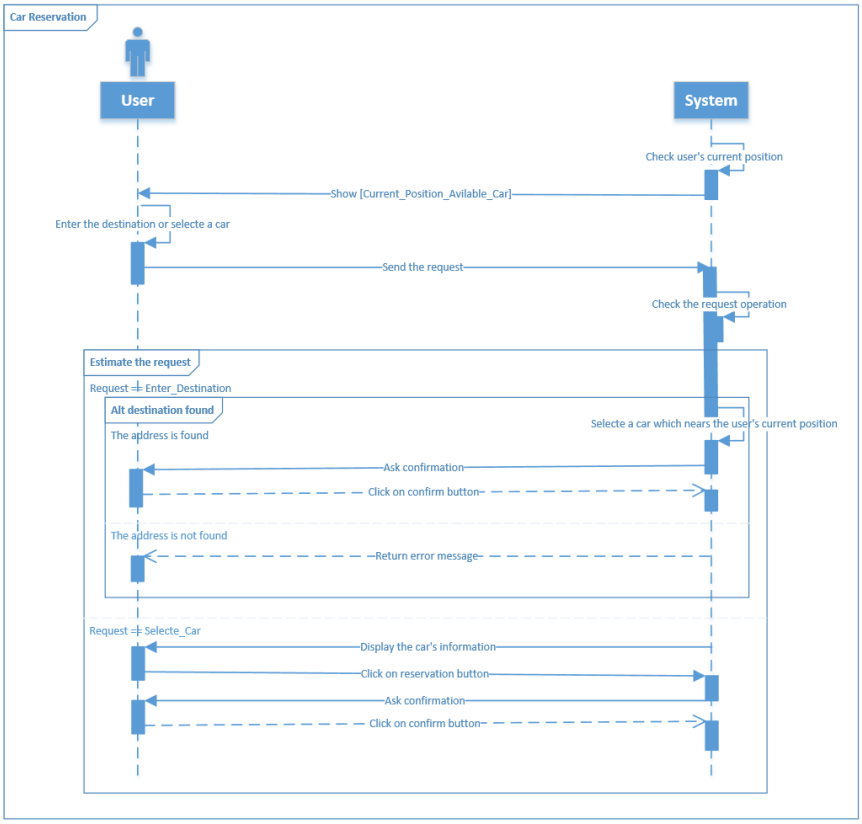

Figure 15: Sequence Diagram - Login

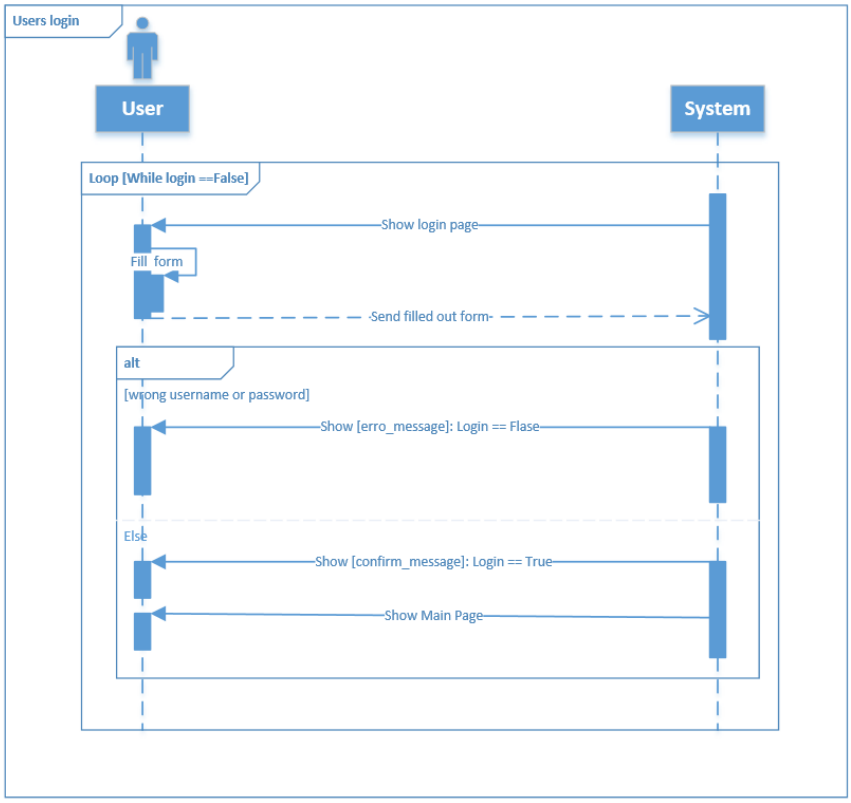

Figure 16: Sequence Diagram - Reservation

5. State diagram

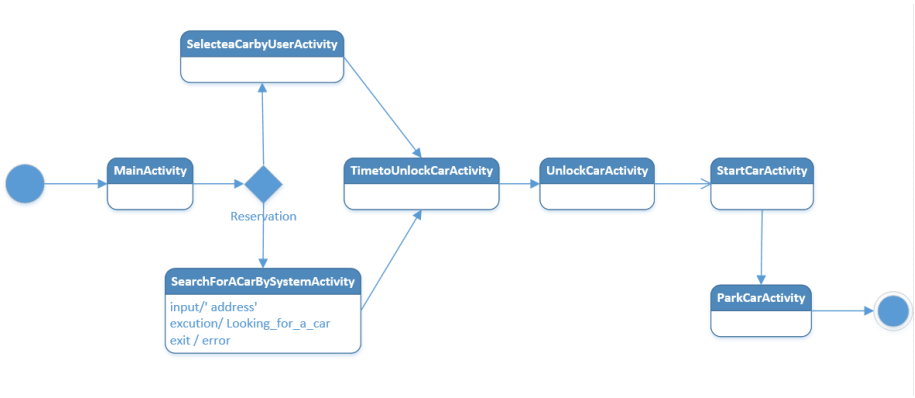

Figure 17: State Diagram

6. Alloy Model and Checking

```
open util/boolean
/*
one sig PowerEnjoy {
                fleet: set Car,
                clients: set Client,
                operators: set Operator,
                chargingStations: set ChargingStation,
                safeAreas: set SafeArea,
                reservations: set Reservation,
                rides: set Ride
}*/

abstract sig  User {
}

sig Client extends User {
                reservations: some Reservation
}

sig Operator extends User {
}

sig Car {
        licensePlate: one LicensePlate,
        batteryLevel: Int,
        position: one Position,
        isLocked: one Bool,
        isCharging: one Bool,
        isOnRide: one Bool
} {
        batteryLevel <= 100
        batteryLevel >= 0
}
// If a car is charging it's not in a ride
fact isOnRideNotCharg {
       all c: Car | (c.isCharging = True) => ( c.isOnRide = False )
}
fact isOnRideNotLock {
       all c: Car | ( c.isOnRide = True ) => ( c.isLocked = False )
}

sig LicensePlate {}
```

```
sig Position {
        latitude: Int, //Float
        longitude: Int //Float
} {
        latitude >= 0
        longitude >=0
}

abstract sig Reservation {
        car: one Car,
        ride: lone Ride,
        bill: one Bill
}

sig CurrentReservation extends Reservation {}
sig PastReservation extends Reservation {}
fact AllReservationHaveClient {
        all r: Reservation | one c: Client | r in c.reservations
}

sig  Ride {
        startPoint: one Position,
        endPoint: one Position,
        passenger: Int,
} {
    startPoint != endPoint
    passenger >= 0
}
// All ride are in a reservation
fact {
        all r: Ride | one res: Reservation | r = res.ride
}

sig  Bill {
}

sig ChargingStation {
        position: one Position,
        numberPlugs: Int,
        pluggedCars: set Car
}

//If car is plugged, status is Charging
fact carPlugged {
     all c: Car | one ch: ChargingStation |  (c.isCharging = True)  => c in ch.pluggedCars// retu
}
```

```
sig SafeArea {
        borders: set Position
} {
    #borders >= 3
}
// License plate is unique
fact licensePlateUnique {
     all c1, c2: Car | (c1 != c2) => c1.licensePlate != c2.licensePlate
}
// A Reservations has a unique client
fact reservationBelongsToOneUser {
     no r: Reservation | some c1, c2: Client  | c1 != c2 and r in c1.reservations and r in c2.rese
}
// A User can only have one current reservation
fact {
     no c: Client | some r1, r2: CurrentReservation | r1 != r2 and r1 in c.reservations and r2 in
}
// Bill has one reservation
fact billUniqueReservation {
        no r1, r2: Reservation | r1 != r2 and r1.bill = r2.bill
}
// To check
assert ReservationIsUnique {
     all disj r, r1: Reservation , c: Client | r in c.reservations and r1 in c.reservations
           implies r != r1
}

pred PowerEnjoy {
    #Client = 3
    #CurrentReservation = 2
    #Car = 2
    #Ride = 2
}

pred addReservation(c: Client, r: CurrentReservation, car: Car) {
        c.reservations = c.reservations + r
}

run addReservation
check ReservationIsUnique
run PowerEnjoy
```

Alloy model checking

```
Executing "Check ReservationIsUnique"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  4353 vars. 399 primary vars. 9647 clauses. 228ms.
  No counterexample found. Assertion may be valid. 41ms.
```

Figure 18: Assert

```
Executing "Run addReservation"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  4320 vars. 399 primary vars. 9611 clauses. 94ms.
  Instance found. Predicate is consistent. 84ms.
```

Figure 19: Predicate 1

```
Executing "Run PowerEnjoy"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  4281 vars. 390 primary vars. 9582 clauses. 83ms.
  Instance found. Predicate is consistent. 89ms.
```

Figure 20: Predicate 2

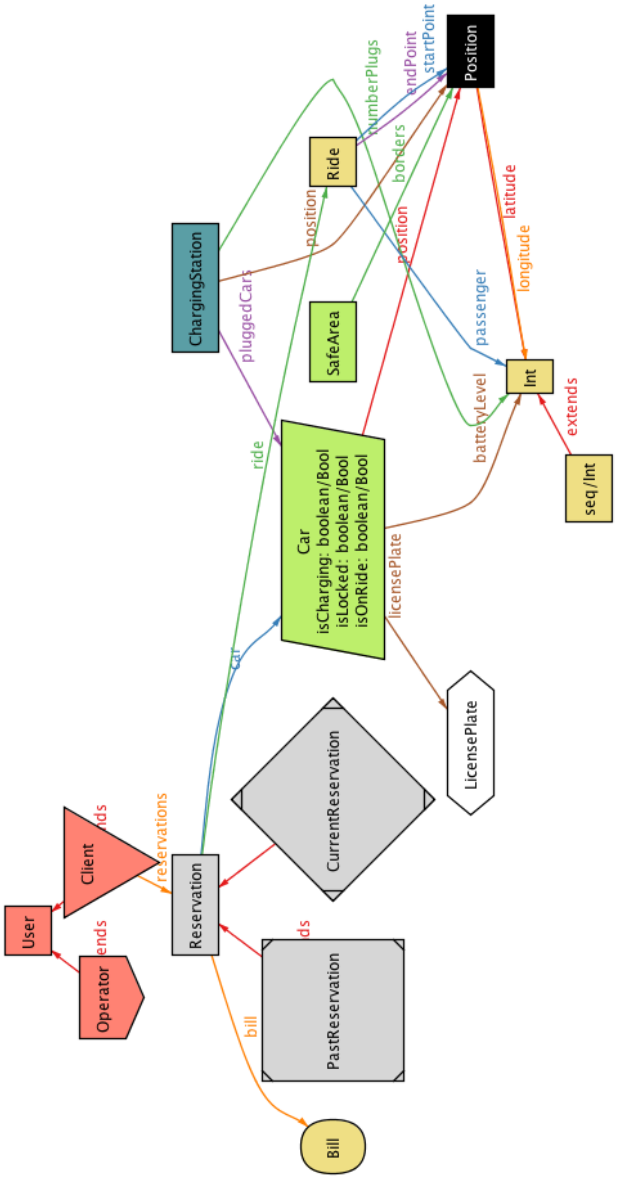

Figure 21: World generated

Used tools

- Atom - Text editor
- Pandoc - PDF Creation
- Github
- Alloy Analyzer 4.2
- Microsoft Visio 2016 - Diagrams
- Axure RP Pro 7.0 - Web and mobile mock-ups

Hours worked

Reda Aissaoui

- 27/10/16: 30m
- 28/10/16: 2h
- 27/10/16: 3h Meeting
- 02/11/16: 3h
- 03/11/16: 3h Meeting
- 07/11/16: 2h
- 08/11/16: 2h
- 09/11/16: 3h
- 10/11/16: 5h Meeting
- 12/11/16: 2h
- 13/11/16: 4h

Jinling Xing

- 27/10/16: 3h
- 28/10/16: 4h
- 03/11/16: 6h
- 06/11/16: 3h
- 08/11/16: 2h
- 10/11/16: 3h
- 13/11/16: 4h

Lidong Zhang

- 26/10/16: 30m
- 27/10/16: 3h group meeting
- 28/10/16: 30m
- 29/10/16: 1h
- 02/11/16: 1.5h
- 03/11/16: 3h group meeting
- 04/11/16: 30m
- 05/11/16: 30m
- 06/11/16: 2h
- 07/11/16: 2h
- 10/11/16: 5h group meeting
- 10/11/16: 2h
- 11/11/16: 2.5h
- 12/11/16: 5h
- 13/11/16: 4h

Changelog

- **1.1** Modified system architecture by adding the car system. *8/12/16*