POLITECNICO
MILANO 1863

Requirements Analysis and Specification Document

Version: 0.1

Reda Aissaoui, Jinling Xing, Lidong Zhang

November 13, 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 for their 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 delete a reservation, 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)

5. 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), //TODO ADD MORE IF NEEDED

6. 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.

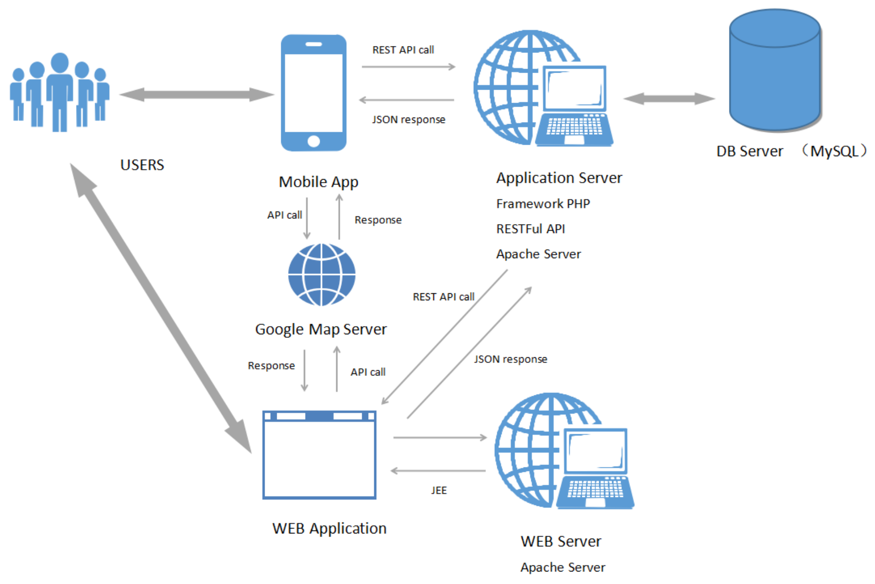

Figure 1: Application architecture

7. 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.

8. 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

1. User requirements

- The user should be able to register in the system.
  - The user can register using his phone number, and system do not need complex registration.
  - The user must be able to register to the system by providing their credentials and payment infomation.
  - The user will receive a password to access the system.
- The user should be able to modify his information
  - The user can modify his destination before or after they check-in.
  - The user can change his personal information.
  - The user can change his payment information.
- The user should see the cars around him or around an address.
  - The user must be able to open his GPS to get his or her location.
  - 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 inputted by users.
- The user should be able to reserve a car.
  - The user should provide his location and his destination when he request a reservation.
  - The system must be able to check the origin location and the destination of a reservation.
  - The system must be able to transfer the request to appropriate car drivers.
- The user should be able to delete a reservation.
  - The user can cancelled a reservation before check-in.
  - The system should noticed the driver about this cancelled reservation.
  - The operator should monitor this information.
- The user should be able to unlock and check-in the car.
  - The user that reaches a reserved car must be able to tell the system he is nearby using GPS.
  - When the system accepts the information that the user reaches the reserved car, the car will be unlocked and the user can check-in the car.
- The user should be able to see how much the ride cost him.
  - The system must be able to provide the cost information of the ride after calculate the cost.
- The user should be able to check his rides history.
  - The operator can monitor the rides history.
  - The system must be able to save the user rides history in the

    databases.
- The user should be able to enable economy mode.
  - The user must bind his credit card and payment information before the first ride.
  - The system can save the user economy mode.

2. System Requirements

- The system should be able to locate all the cars.
  - The system must be able to detect the car's position according to the car's GPS.
- Retrieve the real-time car variables.
  - The system can check the car variables if its GPS available in the real-time.
- Calculate the price of the ride depending on the distance, time, left charge in the battery and number of passengers.
  - The system must estimate the distance between the origin location and the destination.
  - The system must calculate the time between the user check-in and the service finish.
- The system must using a fixed fee for each passengers, and then multiple the total fee of all the passengers and reduce the price of a sharing discount percentage.
- The system must be able to check the battery empty and the parking areas to be recharged, so the system should apply a discount on the last ride.

3. Operator requirements

- Verify the driving license and identity of the drivers
  - when the car drivers registered, the operator should check the upload driving license and identity of the drivers.
- Verify the damaged and faulty cars.
  - If the car damaged or it's the faulty car, the operator must be able to verify it before the car driver login the system.
- Monitor the position of the cars.
  - The operator can monitor the car's position.

2. Non-functional Requirements

1. Mock-ups

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 // TODO ADD REFERENCE TO 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

Class

2. Use cases diagrams

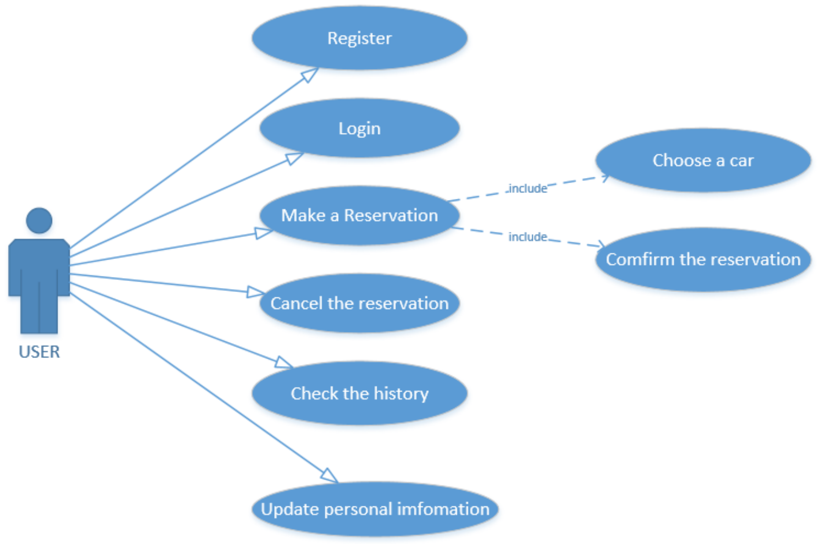

Figure 2: User

Operator

3. Use cases description

Use cases diagram user

User requries to take a car

- Name: User requires to take a car
- Actors: User
- Entry requirements:
- User has login.
- User is in the operation homepage.
- Flow of event:

- User click the button to take a immediately.
- User open his GPS to monitor his location.
- User can see the cars close to his place.
- User decides to take a car and push the take car button.
- User input the destination and passenger numbers.
- Exit conditions: The system forwards the request to the appropriate car and the use case "operator responds to a request" begins.
- Exceptions:
- User has not open the GPS.
- User furnishes invalid data.

User requries to reserve a car

- Name : User requires to reserve a car
- Actors : User
- Entry requirements:
- User has login.
- User is in the operation 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 a form where the user has to give some information like theplace of departure and his destination of the ride.
- 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 system forwards the request to cars which can be reserved and the use case "operator responds to a request" begins.
- Exceptions:
- User has not open the GPS.
- User doesn't input some basic information which is necessary, such as the starting location and destination.

User requries 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 make 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 requries to login

- Name: Operator requries 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 operate authentication.
- Flow of event:
- The operator deal with the information of feedback.
- Monitor the position of the cars.
- Exit conditions:
- The operator is successfully dealed with the feedback report.
- Exceptions:
- The operate process has already time-out.

Operator verefies information

- Name: Operator verefies information
- Actors: Operator
- Entry requirements:
- The operator has login and has got the operate authentication.
- Flow of event:
- Verify the driving license and identity of the drivers.
- Verify the damaged and faulty cars.

- Exit conditions:
- no information needs to be verified.
- Exceptions:
- The operate process has already time-out.

4. Sequence diagrams

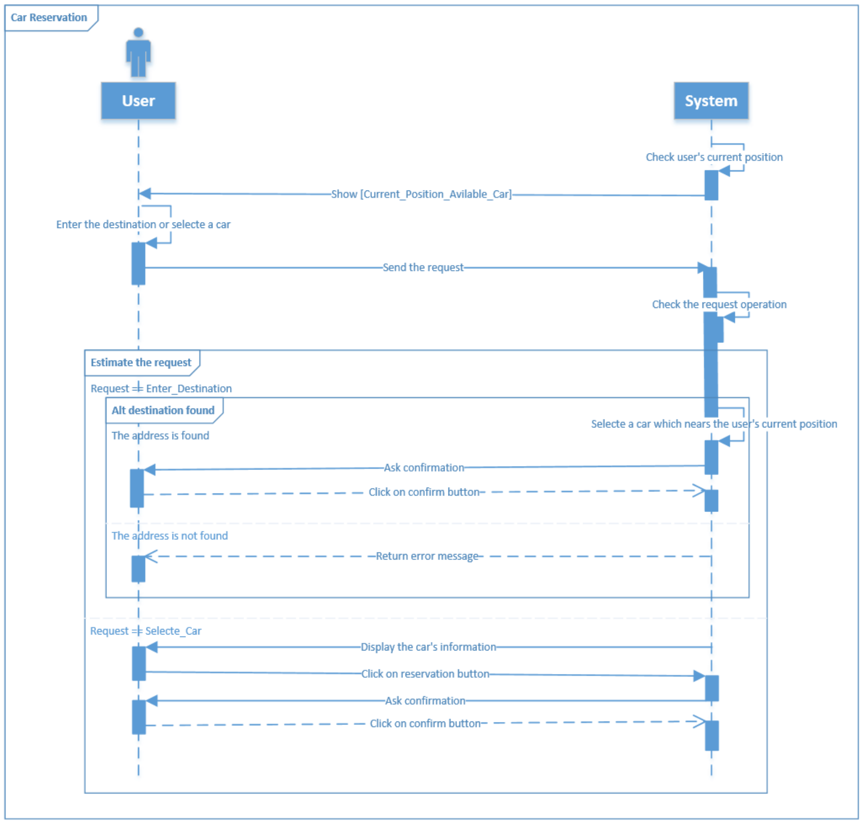
Car Reservation
User
System
Check user's current position
Show [Current_Position_Avilable_Car]
Enter the destination or selecte a car
Send the request
Check the request operation
Estimate the request
Request = Enter_Destination
Alt destination found
The address is found
Selecte a car which nears the user's current position
Ask confirmation
Click on confirm button
The address is not found
Return error message
Request = Selecte_Car
Display the car's information
Click on reservation button
Ask confirmation
Click on confirm button

Figure 3: Sequence diagram_login

Sequence diagram_reservation

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 )
}
/*
fact OneCarOneReserv {
     no c: Car, r1, r2: CurrentReservation  | r1 != r2 and r1.car = c and r2.car = c
}*/

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
}
/*
fact allReservedCarsNotOnRide {
     no c: Car, r1, r2: CurrentReservation | r1 != r2 and c in r1.car and c in r2.car
}*/

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 example {
    #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 example
```

Hours worked