Contents

1 Introduction

1.1 Purpose of this document

The purpose of the RASD (Requirements Analysis and Specification Document) is to give a de- tailed description, analysis and specification of the requirements for the PowerEnjoy system. This document will explain the goals derived from stakeholders' expectations, the characteristics of the application domain and the assumptions made to solve ambiguity and incompleteness. Start- ing from goals and domain properties, requirements will be formulated according to a specific systematic methodology and then specified using both informal and formal notations. However this document should not be considered the final draft for the software specifications since in the following phases several fixing may be necessary.

This document is primarily intended to be proposed to the stakeholders for their approval, to the analysts and programmers for the development of the project and to the testing team the validation of the first version of the software.

1.2 Scope

PowerEnjoy(PE) is an application intended to manage the car-sharing system stakeholders have. The usage of the system must be simple to use for the user in order to make the service competitive.

The system offers a normal car-sharing service: reservation of a car can be done through web browser or using a specific mobile app, but to use the car the users need a smartphone with GPS and internet connection.

The main differnce between PowerEnjoy and other car-sharing relies on the fact that our service is environmen friendly. Our cars are all battery-only powered and user of our system could claim to be friend of the environment.

1.3 Actors of the system

PE will interact with different kind of users to achieve his goals.

- Non registered user: a person that has not registered . These user can do 2 actions: register (and get the credential to login) and login with their credential.
- Registered user: a person that has full access to the functions of the system.
- Cars: they interact with the system providing informations on their state.

1.4 Requirement Engineering

1.4.1 Goals

The following goals have been identified using the available documentation and the laboratory lesson:

G1 Non-registered user must be able to register and sig-in using their credential

G2 User must be able to see available cars

G3 User must be able to reserve a car

G4 User must be able to unlock/get access to car once he is near it

G5 The system should incentivize virtous behaviour

2 Overview of the system

2.1 Product subsystems

PowerEnjoy (PE) software can be decomposed into subsystems interacting between each other. Those subsystems have to be considered abstract as they do not reflect the architecture of the system to be developed.

1. PowerEnjoy web app (PEW): web portal through which users can access services the system offers. Using PEW the user have to manually specify where he/she is. A registered user can see cars and make reservation.
2. PowerEnjoy mobile app (PEM): this application is designed for smartphones. PEM can retrieve GPS position of the user using the GPS sensor of the phone(if available) and offers the same service as PEW. The unlock of the service can only be done using PEM. PEM offer a system that can recognize license plate through camera.
3. PowerEnjoy backend (PEB): it is the heart of the application. It manages all functions the system offer; it is where the business logic of the application is implemented.

2.2 Constraints

The following constrain have to be met

- A car must be parked where there is internet connection
- Users must have a smartphone
- User must have downloaded PEM before make a reservation
- The user must reserve a car before he use (unlock) it
- A user can reserve at most one car at time
- Cars know safe and unsafe areas

2.3 Domain assumption

After a talk with the stakeholders it is possible to assume that the following assertion always hold.

D1 Users are real people (they have provided true informations during registration)

D2 Users have a driving license

D3 GPS always provide the right position

D4 Each car is uniquely identified by a car license plate

D5 Reserved cars cannot be moved by their position (until user unlock it)

D6 It exist an external system that manage payment(ex. paypal)

D7 The stakeholder company has a system to pick up cars and move them. This system must not be integrated into the to-be-developed system

D8 Battery charge indicator always provide true values and can estimate on average how many kilometers the car can run

D9 Internet connection is reliable

3 Requirements

3.1 External interface requirements

In this sections it is provided

3.2 Service interface

The system to be developed need to interact with the external services

1. Gps system of smartphones
2. Gps system of cars
3. Mobile phone camera
4. External maps (ex. Google Maps) to see the position of cars
5. Online payment systems(paypal, credit card ...)
6. email server

3.3 Comunication interface

The system to be developed need to comunicate trought internet

3.4 Technological requirements

The cars that will be used in PowerEnjoy carsharing system must be provided with the following technological subsystems:

1. The cars need to have a GPS system
2. The cars need to have a internet connection
3. The cars need to know how many passengers are on board
4. The cars can be locked/unlocked remotely

3.5 Functional requirement

3.5.1 [G1] Non-registered user must be able to register and log-in using their credential

1. System shall provide a registration form
2. System shall save data provided by users
3. System shall verify payment details provided by the user at registration (using API of the external payment service)
4. System shall return to user login password via email
5. System shall provide a log-in form

3.5.2 [G2] User must be able to see available cars

1. System shall interaface with gps of mobile phone in case the user want to use his/her position
2. System shall retrive available car position(do not show reserved car; do not show cars in use by other users)
3. System shall show position of the user and of other cars in a map on the device used by the user. The system only show available cars within a certain distance from the user position(the default distance is 500m)
   (a) The system shall allow user to specify the distance from a given position

3.5.3 [G3] User must be able to reserve a car

1. System shall allow user to reserve a car
    - (a) System save the reservation and generate an unique id
    - (b) After reservation the system apply a timer to the reservation
        - i. If the user does not pick up the car within a hour from reservation the system charge a 1euros fee
2. System shall notify the user when a reservation has been inserted in the system
3. System shall stop a reservation if the user already have a pending reservation or is using a car

3.5.4 [G4]User must be able to unlock a car if reserved by himself

1. System shall be able to use the position of the user
    - (a) PEM shall be able to send position to the backend
2. System shall unlock the car remotely when a user and his reserved car are less than 10m away

3.5.5 System have to be updated by car

1. Car need a system to communicate (via internet) to the remote system
    - (a) Cars have to provide:
        - i. Power charge
        - ii. estimation of km it can run
        - iii. GPS coordinates
        - iv. Money charge
        - v. Position is safe/unsafe
    - (b) Informations at point a must be provided during the ride and when the car is locked

3.5.6 [G5]The system should incentivize virtuous behavior

Within this section there will be collected all functional requirements related to virtuous behavior

1. The system shall collect the following informations after each ride:
    - (a) number of passengers on board
    - (b) battery charge
2. The system shall calculate the discount according to virtuous functionality

   (a) if more than a discount is applicable to the user, the system apply the maximum discount between all applicable discount (not the cumulative discount)

3.5.7 Others functional requirements

1. System shall notify user about how much he has to pay after each ride.
   (a) The notification contains an invoice with the ammount of money to pay and the applied discount(if any)
2. If the transaction cannot be done due to insufficient money, the application will not offer other services until the user pay the debt

3.6 Scenarios

In order to clarify the expected functionality of the system, in this section we propose some possible scenarios. In each scenarios we try to show different functionalities of the system

3.6.1 Scenario 1

Marco is a trader. When he arrived at the office it was sunny, but now it is raining. Before leaving the office, he use his computer to register in the PowerEnjoy car-sharing application. After he has provided his personal details and payment information, the system save his data and returns his password. He then login into the system and manually specify his current address. In a few seconds the system shows him a map with a label in the center, the position specified by Marco, and the position of cars far 500m from that label. He reserve the nearest car, which is 50m away. After closing the web application, he exit the office and go to the car. When he see the car, he use the PowerEnjoy mobile app to unlock the car. The remote system use his position and compare it with the position of the car and decide to unlock the car as they are less than 10m away. Marco then jump into the car and goes home. He left the car near home, in a safe area, and exit the car. The system locks the car and after some seconds receive from the car the battery charge, which is more than 50%. The system applies a discount of 20% and than charges Marco. Marce receive a notification on his smartphones with the ammount to pay

3.6.2 Scenario 2

Anna went out for a dinner with her friends. When the night is over, she and 2 other friends need a car to come back home. She is already registered in PowerEnjoy sevice, so she login using the mobile app. The phone send her coordinates to the system, and the backend replies sending the map with his position and the car nearby(500m of radius). Unfortunatelly, only few car are near and all of them are already reserved. So she extend the map and find a car

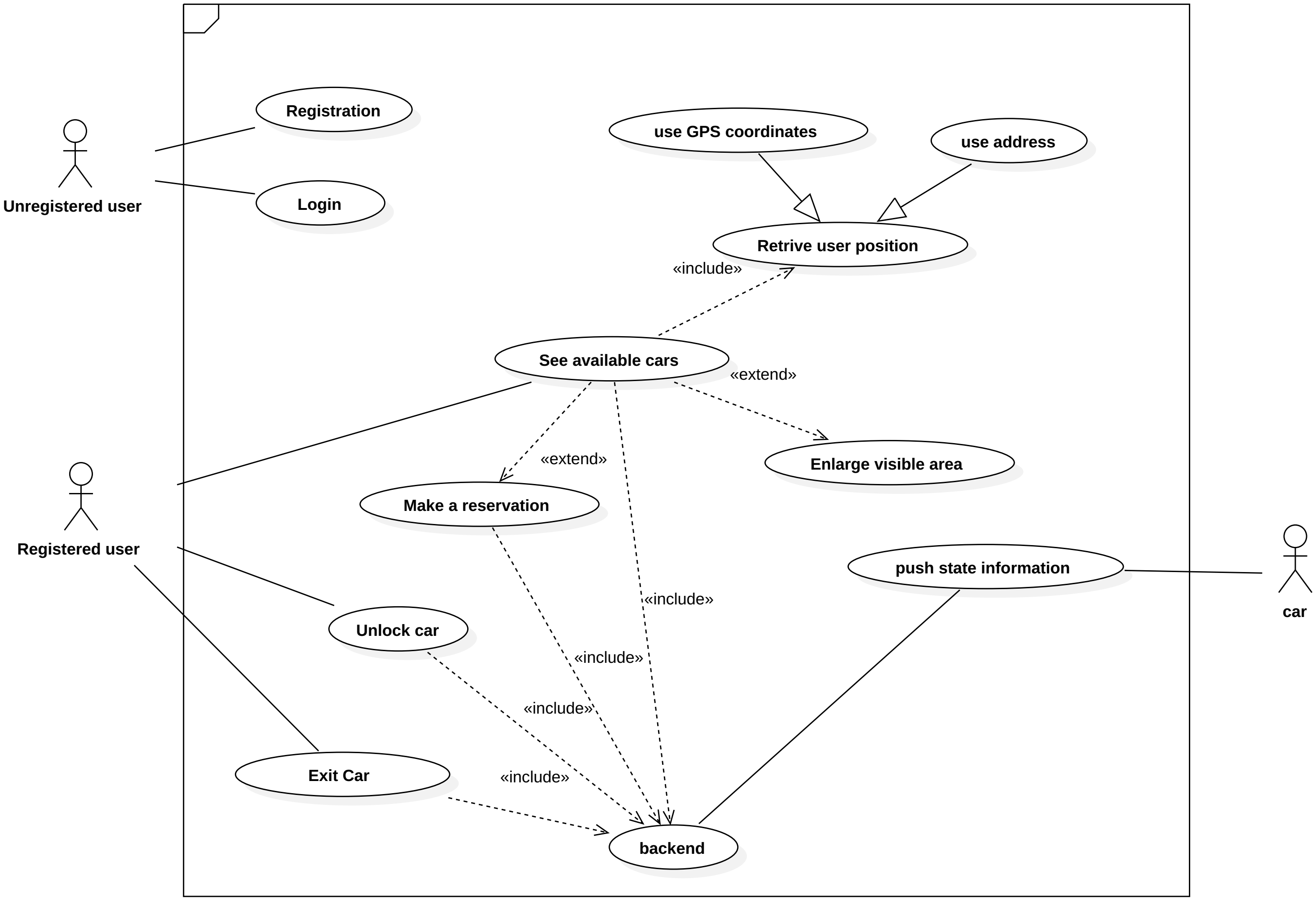
700m away. When she is near, the system free the car only after she requires the car to be unlocked. So Anna bring her friends home first and then drive to his house. She get out and unlock the car. The system than applies a discount of 10% due to passengers discount. The system then charges Anna.

3.6.3 Scenario 3

Alice works in a office. At lunch time she has to go home. Before leaving the office, she reserve a car near the office. Unfortunaltelly, her boss ask her to remain at work as the group need to work hard on project, thereby she has reserved a car that she will never use. One hour after the reservation, the system mark the reserved car as free and charge 1 euro to Alice.

3.7 Use Case

Model1::UseCaseDiagram1



3.7.1 Registration

Name: Registration of a user

Actors: Unregistered user

Entry Conditions: No entry condition

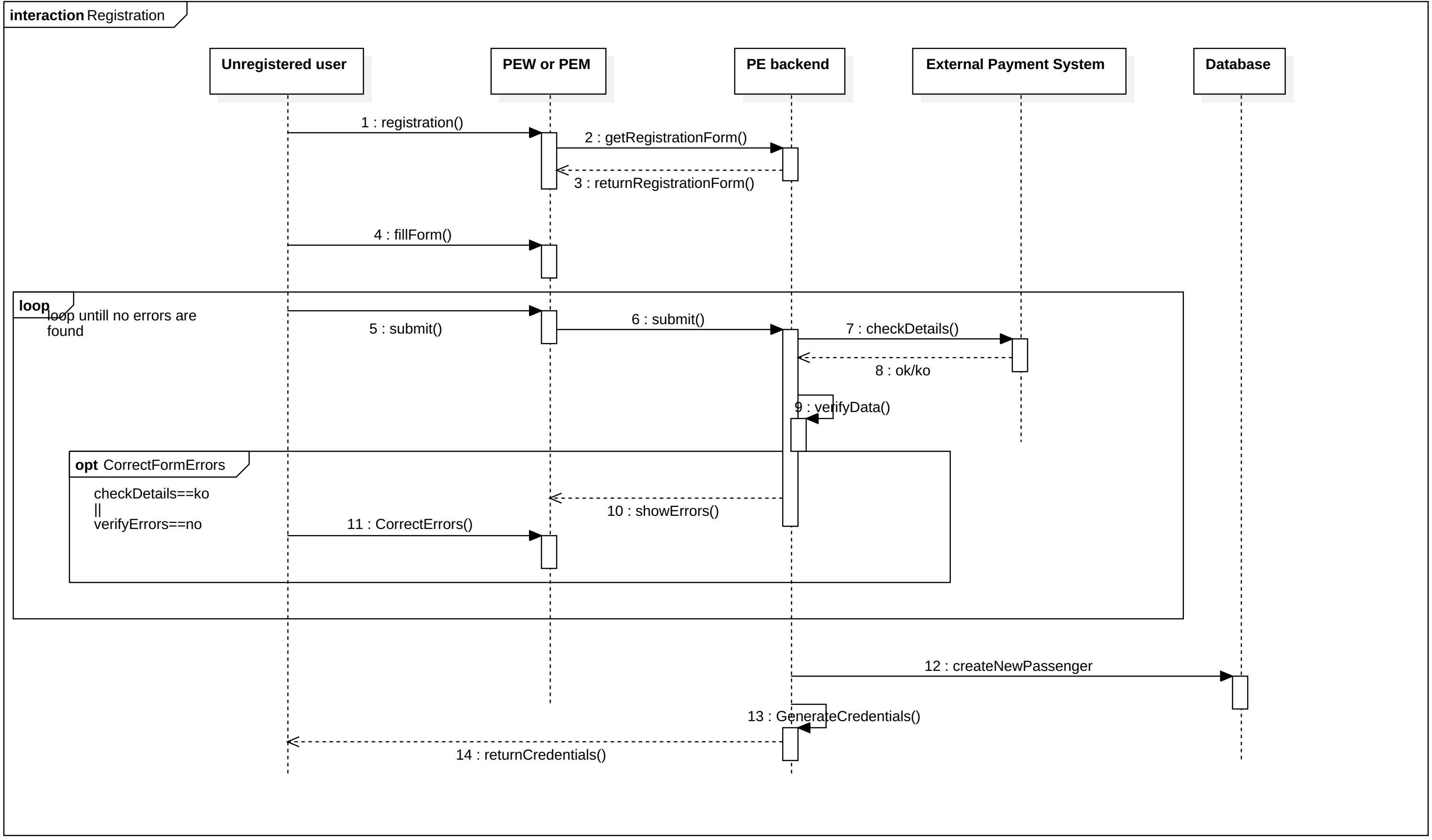
Flow of events:

1. Unregistered user open the registration page
2. Unregistered user fills the form with the required information: firstname, lastname, email, username, address and payment details
3. User submitts the form to the PE backed system
4. Backend checks user data and if the payment details are valid
5. Backend save data of the user in the database
6. Backend generates a password for the user
7. Backend send back to user the password via mail

Exception:

- Informations provided by the user are not valid: the system show a message to the user and ask to insert valid data
- The procedure is interrupted before termination: the system does not insert the new user in the database

Collaboration1::Interaction1::Registration



3.7.2 Login

Name: Login of an already registered user

Actors: Unregistered user / registered user

Entry Conditions: user has alredy been registered in the database

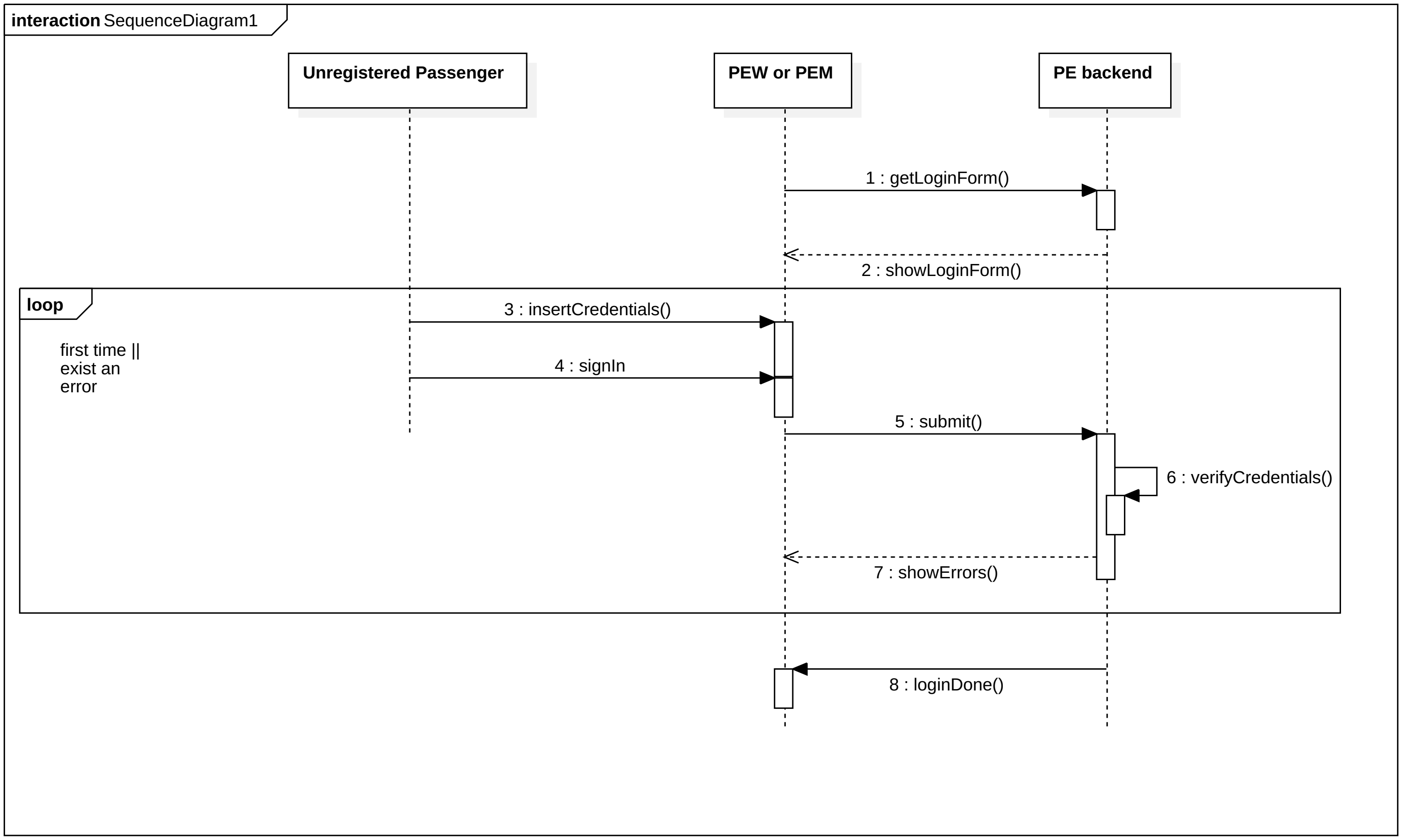
Flow of events:

1. User open the login page
2. User input his pesonal credentials
3. System checks credentials provided by user
4. User get the status of registered user

Exception:

- Credentials provided by the user do not match with saved credentials : show an error on user interface

Collaboration1::Interaction1::SequenceDiagram1



3.7.3 Reservation(web or mobile)

Name: Reservation of a car

Actors: Registered user

Entry Conditions: User is on the search page

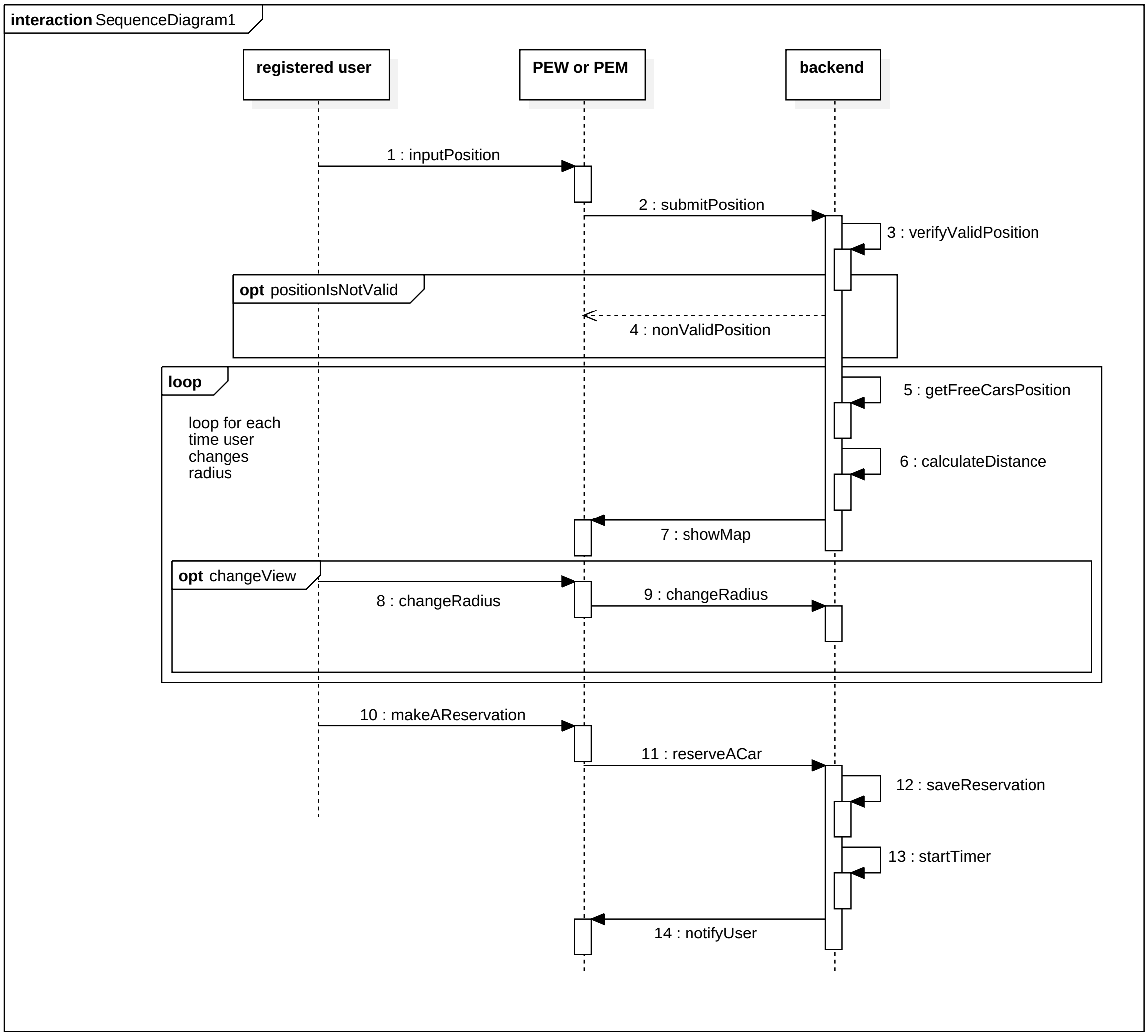
Flow of events:

1. User input a position
    (a) If the user is using PEW he has to manually input an address
    (b) If the user is using PEM he can choose either to input an address or to use GPS coordination provided by phone
2. System get the user position and check in the database the position of free cars
3. System extract from free cars those that are within a distance of 500m from user position
4. System return a map where it is showed user and cars position
    (a) User can enlarge/restrict the radius
5. User select a car
6. User click on “Reserve”
7. System is notified on the reservation and start a timer
8. User is notified that the reservation has been registered by the system

Exception:

- Position is not correct/system can not locate the position: an error is showed on the user interface

Collaboration1::Interaction1::SequenceDiagram1



3.7.4 pick-up

Name: Pick up a reserved car

Actors: Registered user, car

Entry Conditions:

- User is near his reserved car
- The user already have a reservation

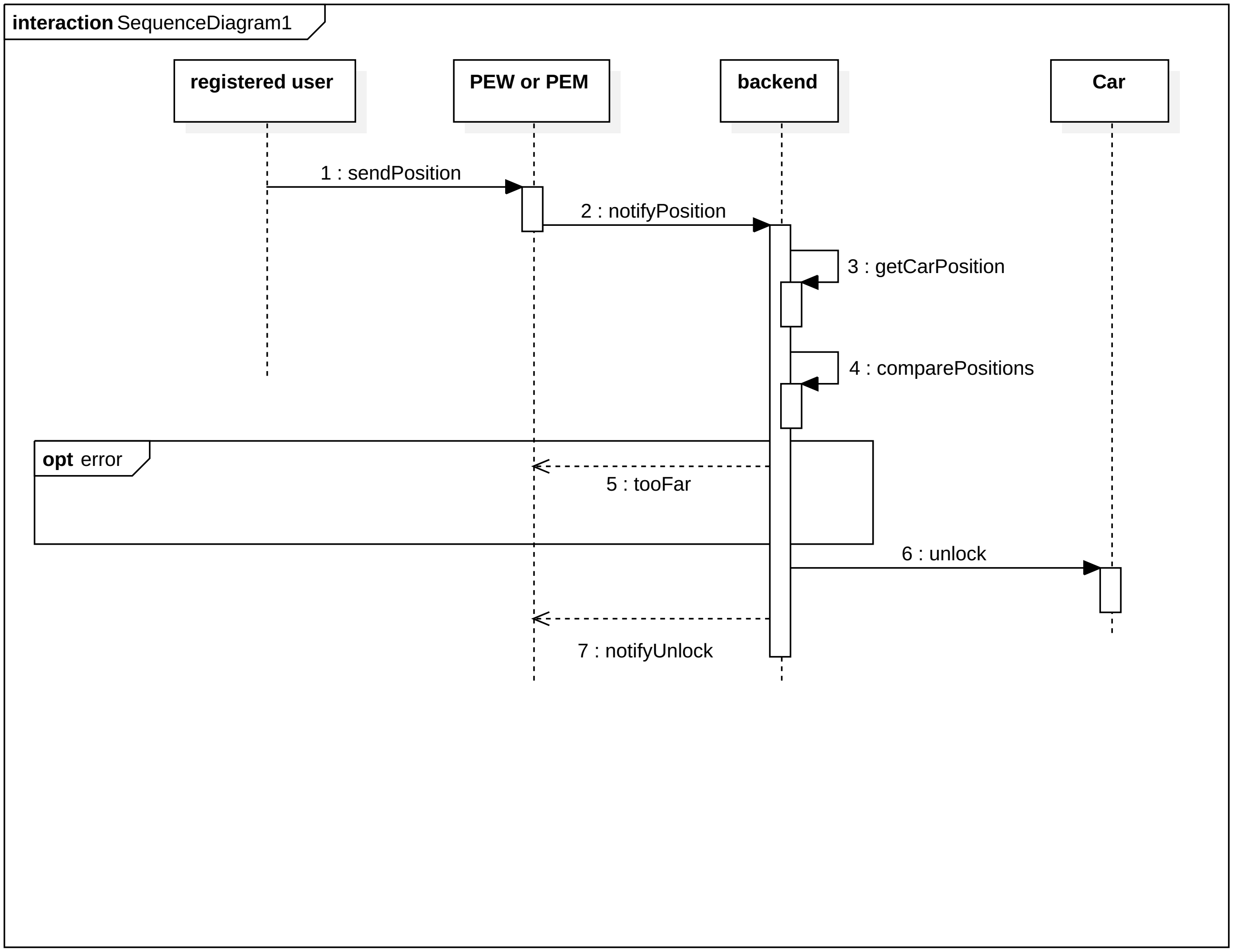
Flow of events:

1. The user allert the backend(using PEM) that he is near the car
   (a) PEM, while contacting the backend, send the GPS coordinates of the user
2. The system compare GPS coordinates of the user with GPS coordinates of the reserved car
   (a) Disctance should be less than 10/15 m
3. The system unlock the car

Exception:

- distance is more than 10/15m: the system aller the user on PEM

Collaboration1::Interaction1::SequenceDiagram1



3.7.5 no-pick up

Name:

Actors: Registered user

Entry Conditions: The user already have a reservation

Flow of events:

1. Timeout of the timer
2. The system charge the user of 1 euros

3.7.6 the ride

Name:

Actors: Registered user, car

Entry Conditions: The user has picked up a car

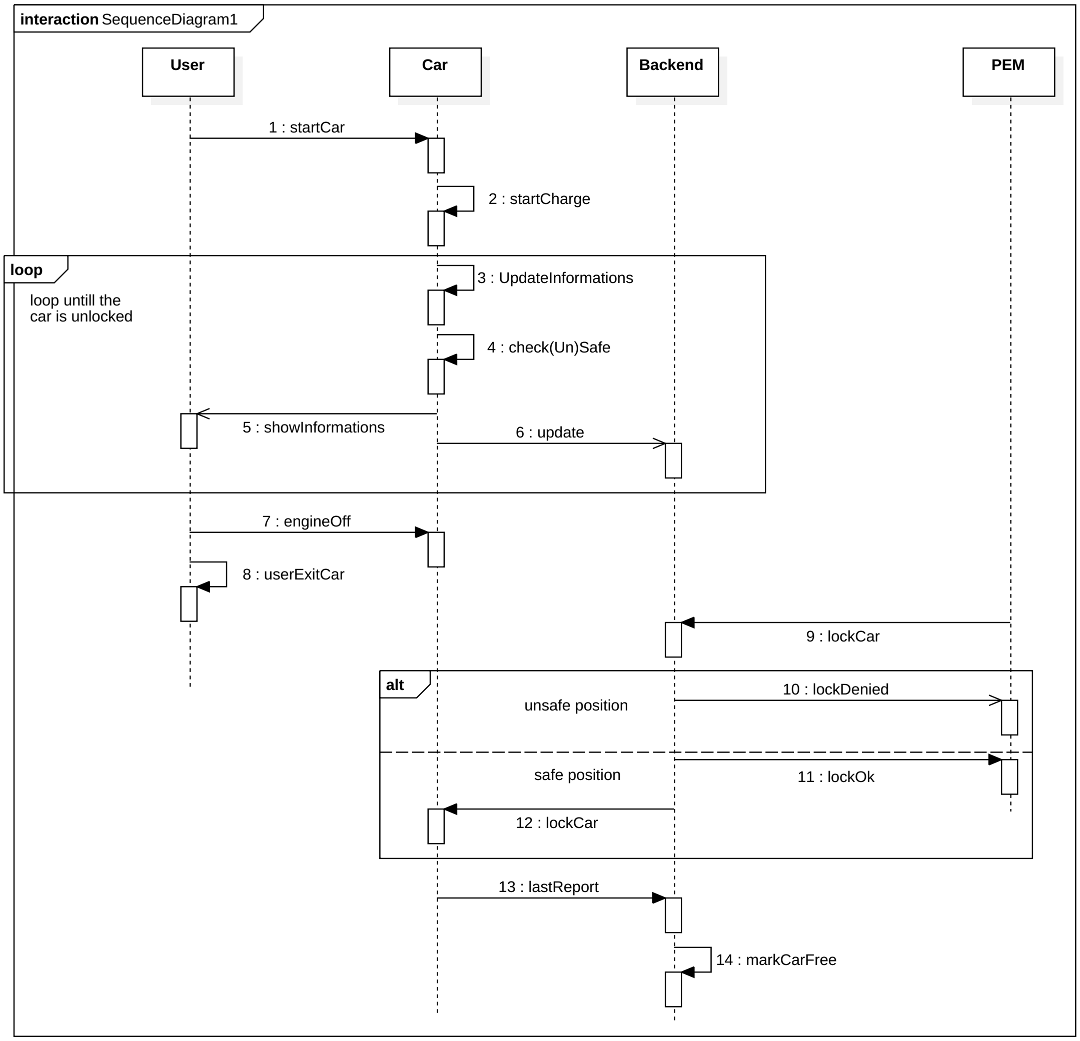
Flow of events:

1. The user starts the car
2. The car starts charging the user
   (a) the charge is showed on the monitor
3. The car start updating the backend:
   (a) GPS position
   (b) Charge indicator
   (c) Estimation of kms the car can run
   (d) Money charge
   (e) Number of passengers
4. The car always signal to the user (through the screen) if he is in a safe or unsafe area.
5. The user park and turn off the engine
6. The user exit the car
7. The user (using PEM) signal to backend to lock the car
8. System locks the car
9. The car sends a last report to the backend
10. System marks the car as free

Exception:

- The user tries to left the car in unsafe area:
  - Backend do not allow lock and show it on PEM

Collaboration1::Interaction1::SequenceDiagram1



3.7.7 Charge the user

Name: How to charge the user

Actors: car, payment system

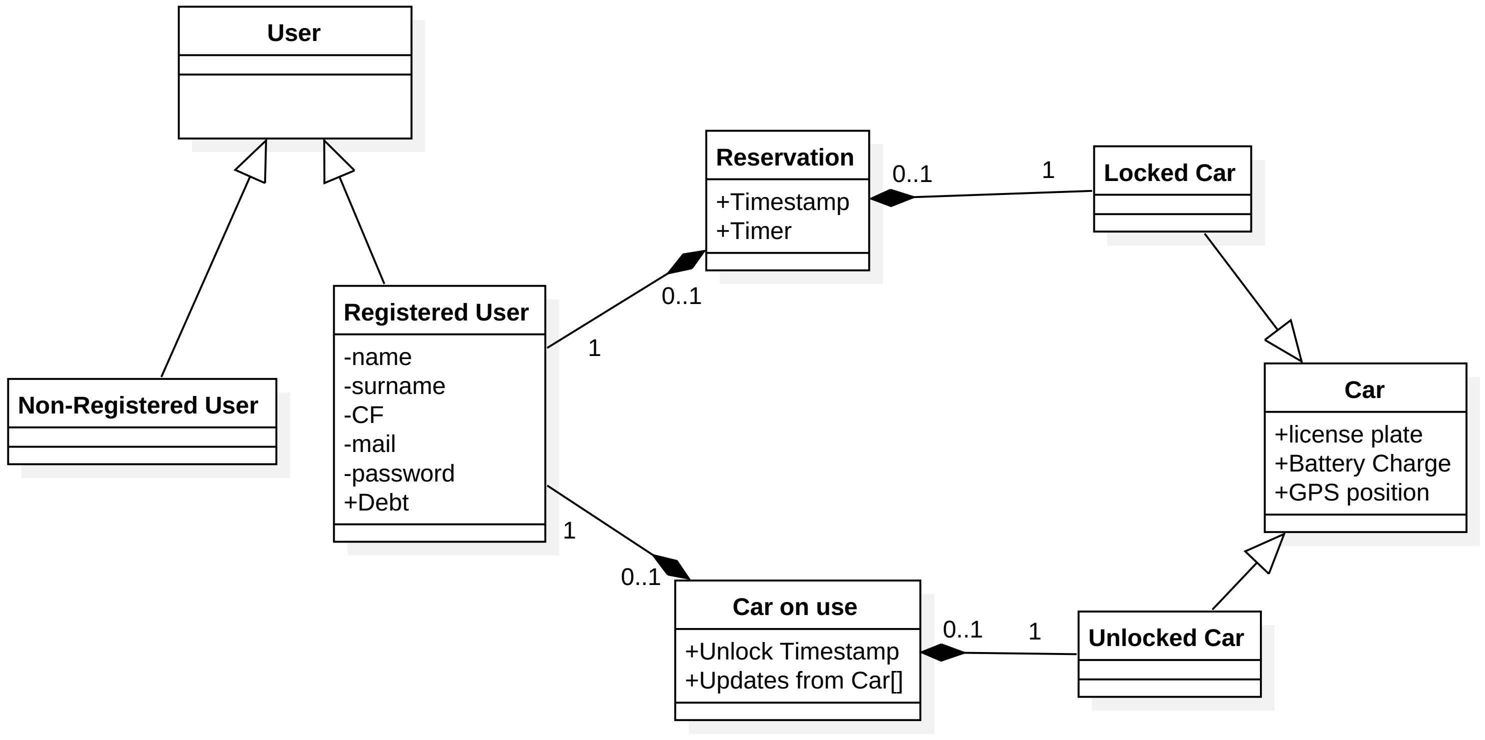
Entry Conditions: The user have ended the ride, the car has been locked

Flow of events:

1. The system get last informations from the car
2. The system evaluate all the informations has been received from the car along the ride (if there has been 2 or more passengers on the car the system must know this for the next point)
3. The system evaluate all the discount the user is elegible for
4. The system calculate the bill
5. The system applies the maximum discount between those available
6. The system charge the user

3.8 Class Diagram

Model1::ClassDiagram1



3.9 Non functional requirements

Since PE is intended to offer a service to customers and in order to guarantee a high quality of service, according to stakeholders' expectations we identified the following non functional requirements

3.9.1 Performance

- The response time of PEM and PEW must be less than 2 sec in 99% of times (from user input submission to user notification of the action)
- Each car must send informations to the system each 10 sec when it is used by users
- After each ride, when the car is locked, the car must send a final report to the system in less than a minute
- The system shall support at least 1000 parallel connections

3.9.2 Reliability

- All data of the system is saved in a transactional database (reliability is assured by the database itself)
- A backup of data should be performed every week to prevent loss of data

3.9.3 Security

- All sensible information must be protected with encryption when exchanged on internet
- A user can change only his personal information
- Only administrators can access all data of the system

3.9.4 Availability

- PE system must be available 24 hours per day, all days

3.9.5 Portability

- PEM shall be developed for iOS, Android and Windows phone

3.9.6 Documentation

- A user guide must be available online
- All documentation produced during the design and development of PE shall be available to personnel

3.9.7 User interface

- The interface (PEM, PEW) must be easy to use and intuitive. No previous knowledge or experience is required to use the service.
- The interface must detect whether a robot is using the application and stop it

4 Alloy

ALLOY MODELLING

Model

```
open util/boolean
// The Client Object
sig Client {
password : Int}
{
password > 0
}
//Declaring the uniqueness of the passwords
fact passwordsAreUnique {
all c1,c1 : Client | (c1 != c2) => c1.password != c2.password
}
//Electronic Car Object
sig Car{
seats : Int}{
seats > 0
seats <= 4
}
// Matching the client and the car
fact clientOfCar {
Client.car = Car
}
// Declaring every car will only have 1 client only. (There could be more than 1 passangers they are defined seperately)
fact allCarHaveOneClient {
all c1,c2 : Client | (c1 != c2) => c1.car != c2.car
}
// Position Object
sig Position {
lalitude : Float,
longitude : Float
}
// The object CarPosition in the object Position
abstract sig carPosition extends Position {
car : one Car
}
// The object ParkPosition in the object Position
abstract sig parkPosition extends Position {
park : one Park
}
// The objects of Start and Stop positions of the car in the object Car Position
sig StartPosition extends carPosition{
}
sig StopPosition extends carPosition{
}
// Stop Position for one client is Start Position for another Client
pred CarPosition.sameClient[other: CarPosition] {
this.client = other.client
}
pred CarPosition.complement[other: CarPosition] {
other != this this.sameClient[other] (this in StopPosition) <=> (other in StartPosition)
}
// Declaring the Object Path
sig Path {
positions : seq StopPosition
}{
#positions >= 2
}
// Declaring the uniqueness of the Positions
fact pathPositionsAreUnique {
all p : Path | not p.position.hasDups
}
// Declaring the clients will get in the car and get out of the car meaning the Start and Stop
fact pathClientHasAStartAndAStop {
all p: Path | all i1: p.positions.inds |
one i2: p.positions.inds | i2 != i1 and
let p1 = p.positions[i1] | let p2 = p.positions[i2] |
p1.complement[p2] and not ( one i3: p.positions.inds |
i3 != i2 and i3 != i1 and let p3=p.positions[i3] |
p3.complement[p1] or p3.complement[p2])
}
// Declaring the object of path ends meaning of Stop
fact pathEndWithStop {
all p: Path | p.positions.last in StopPosition
}
// Stating the client will get in and out
assert clientGetInAndOut {
all p : Path | all i1 : p.positions.inds | one i2 : p.positions.inds |
i2 != i1 and let p1 = p.positions.[i1] | let p2 = p.positions.[i2]
p1.complement[p2]
}
assert sameNumberOfStartAndStop {
all p : Path | #(p.positions.elems & StopPosition) = #(p.positions.elems & StartPosition)
}
// Declaring the object Request
abstract sig Request {
path : Path,
passengers : Int
}{
passengers > 0
passengers < 4
}
// The object Single Request in the object Request
sig SingleRequest extends Request {
}
// The object Multi Request in the object Request since the charging will be different with more or equal with 3 passangers (1 of them is obviously client)
sig AtLeastThreePassangersRequest extends Request {
}
// The object Queue
abstract sig Queue {
s : seq univ
}
// The object Car Queue in the object Queue
sig CarQueue extends Queue {
}{
s.elems in Car
}
// The object Request Queue in the object Queue
sig RequestQueue extends Queue {
}{
s.elems in Request
}
// Declaring the cars which are enqueued must be available for the Ride
fact enqueuedCarMustBeAvailable {
all c : CarQueue.s.elems | d.available.isTrue
}
// Declaring the available cars must be enqueued
fact availableCarMustBeEnqueued {
all c : Car | c.available.isTrue <=> c in CarQueue.s.elems
}
// Declaring the uniqueness of the enqueued elements
fact enqueuedElemntsMustBeUnique {
all q : Queue | not q.s.hasDups
}
// Declaring the Object Zone
sig Zone {
cars : one CarQueue,
requests : one RequestQueue,
bounds : seq Position
}{
#bounds > 2
all p : bounds.elems | not p in StopPosition
```

```
not bounds.hasDups
}
// Declaring the one queue in the object Zone
fact oneQueueforZone {
all z1, z2 : Zone | z1 != z2 =>
z1.cars != z2.cars and z1.requests != z2.requests
}
// Declaring that the Zone must have all the Car Queues
fact zoneOwnsAllCarQueues {
Zone.cars = CarQueue
}
// Declaring that the Zone must have all the Request Queues
fact zoneOwnsAllRequestQueues {
Zone.requests = RequestQueue
}
// Declaring the Object Ride
sig Ride {
cars : some Car,
path : Path,
prices : Client -> Int
}
// Declaring the Prices for the process of Ride
fact RideContainsPricesForItsClients {
all r : Ride | r.prices.(Int) = r.path.positions.elems.(client)
}
// The object Car Central that governs the cars
one sig CarCentral {
cars : some Car,
clients : some Client,
zones : some Zone
}
// Declaring the CarCentral owns all of the Cars
fact ownAllCars {
CarCentral.cars = Car
}
// Declaring the CarCentral owns all of the Clients
fact ownAllClients {
CarCentral.cars = Client
}
// Declaring the Car Central owns all of the Zones
fact ownAllZones {
CarCentral.zones = Zone
}
// Showing the results of the Model
pred show() {
#CarCentral = 1
```

```
#Ride = 1
#SingleRequest = 1
#AtLeastThreePassangersRequest = 1
}
```

Used Tools

1. LYX visual editor for LATEX (http://www.lyx.org/) to write this document.
2. Star UML (http://staruml.io/) for use case diagram, class diagram, sequence diagram.

Hours of work

Alessandro Folloni

- 22 october: 3h
- 23 october: 4h
- 2 november: 2h
- 4 november: 6h
- 5 november: 10h
- 6 november: 6h
- 7 november: 2h

Hassancan Sayilan

- Studying PDF and Class Notes
  - 20/10/2016: 1h
  - 25/10/2016: 3h
  - 26/10/2016: 30m
- Goals-Domain Assumptions-Requirements
  - 26/10/2016: 30m
  - 29/10/2016: 1h
  - 30/10/2016: 2h
- Functional Requirements-Scenarios
  - 04/10/2016: 1h
- Checking Alessandro's Work-Fixing Mistakes
  - 05/10/2016: 1h
  - 06/10/2016: 1h
  - 07/10/2016: 1h
- Checking Diagrams-Working on Alloy

- 09/10/2016: 3h
- 11/10/2016: 2h
- 12/10/2016: 5h
- 13/10/2016: 3h