Design Document

Jia Hongyan	10470662
Lang Shuangqing	10517902

Contents

Chapter 1
Introduction

1.1 Purpose

We will implement *PowerEnJoy*, which is a car-sharing service based on a mobile application system, with two different targets parts:

- system
- users

The system allows users who have registered to rent a car via web or mobile APP. Besides, the system accepts the users' demand and give a route to the car by GPS. The car will be unlocked when user nearby, the car is available and the users may enter. And the users can drive it to the destination and pay the bill.
The system includes extra services and functionalities, that is to incentivize the virtuous behaviors of the users by discount and penalty, according to the quantity of passenger, battery states and special parking area.

The main purpose of the system is to offer a better car-sharing service for the users.

1.2 Scope

This document aims for offering design details in the car-sharing software design phase, whose scope includes the system's forming components, the attributes, methods and events for each component/class, the illustration for external interface between components, data user interface design and design patterns, etc.
The component has to be sample, independent and maintainable in order to increase the reusability for the modules. The components should be independent and isolated from each other. The database has to follow ACID principle. The design architecture should be defined clear and easy to extended.

1.3 Definitions, Acronyms, Abbreviations

Name	**Definition**
RASD	Requirements Analysis and Specification Document
DD	Design Document
IOS	IPhone Operation System
APP	mobile application, here implement on IOS, Android system
ACID	Atomicity, Consistency, Integrity and Durability
DB	database or database layer
OO	Object-Oriented method
SOAP	Simple Object Access Protocol
API	Application Program Interface
DBMS	Data Base Management System
UI	User Interface
NETWORK	3G, 4G and Wi-Fi connection

Table 1.1 Definitions, Acronyms, Abbreviations

1.4 Reference Documents

This document refers to the following documents:

- *Slides for design part*
- *RASD and Design Document assignment - Software Engineering 2 project*
- *Template for the design document - Software Engineering2*
- *Project rules of the Software Engineering 2 project*
- *Software engineering: principles and practice (Vol. 3). Van Vliet, H., Van Vliet, H., & Van Vliet, J. C. (1993).*

1.5 Document Structure

This document is structured in the next parts:

1: Introduction. This section provides general information about the DD document and the system to be developed.

2: Architectural Design. This section shows the main components of the systems with their sub-components and their relationships. This section will also focus on design choices, styles, patterns and paradigms.

3: Algorithm Design. This section will present and discuss in detail the algorithms designed for the system we design two algorithms for car sharing system.
4: User Interface Design. This section shows how the user interface will look like and behave, by means of concept graphics and UX modeling.
5: Requirements Traceability. This section shows how the requirements in the RASD are satisfied by the design choices of the DD.
6. Effort Spent. This section shows the hours of work of this document.
7. References. This section gives the reference materials we used.
8. Used Tool. This section concludes all the tool we use to develop this document.
9. **Change Log.** This section shows all the changlog versions of this DD document.

Chapter 2

Architecture Design

2.1 Overview

This chapter provides a comprehensive view over the system components, both at the physical and at the logical level.

The system will be described starting with high-level components (section 2.2). This high-level design will be thoroughly dissected and detailed in section 2.3. Section 2.4 will put the attention on the deployment on physical tiers, and section 2.5 will describe the dynamic behavior of the software. Section 2.6 will focus on the interface between different components of the system. The design choices and patterns used in the design will be presented and discussed in section 2.7.

2.2 High level components and their interaction

The main high-level components are structured in 4 layers, shown in Figure 2.1. A more detailed figure of layers, with JEE components, is showed in Figure 2.2. Each layer is described as follows:

1. Database

This layer is responsible for the data storage and data access, following ACID principle.

2. Application server

This layer includes the policies, algorithms and calculation, which is the logic part of the system.

3. Web server

This presentation layer provides web server..

4. Client

This layer contains two distinguished types of client:

- ➢ Mobile APP Client
- ➢ Browser Client

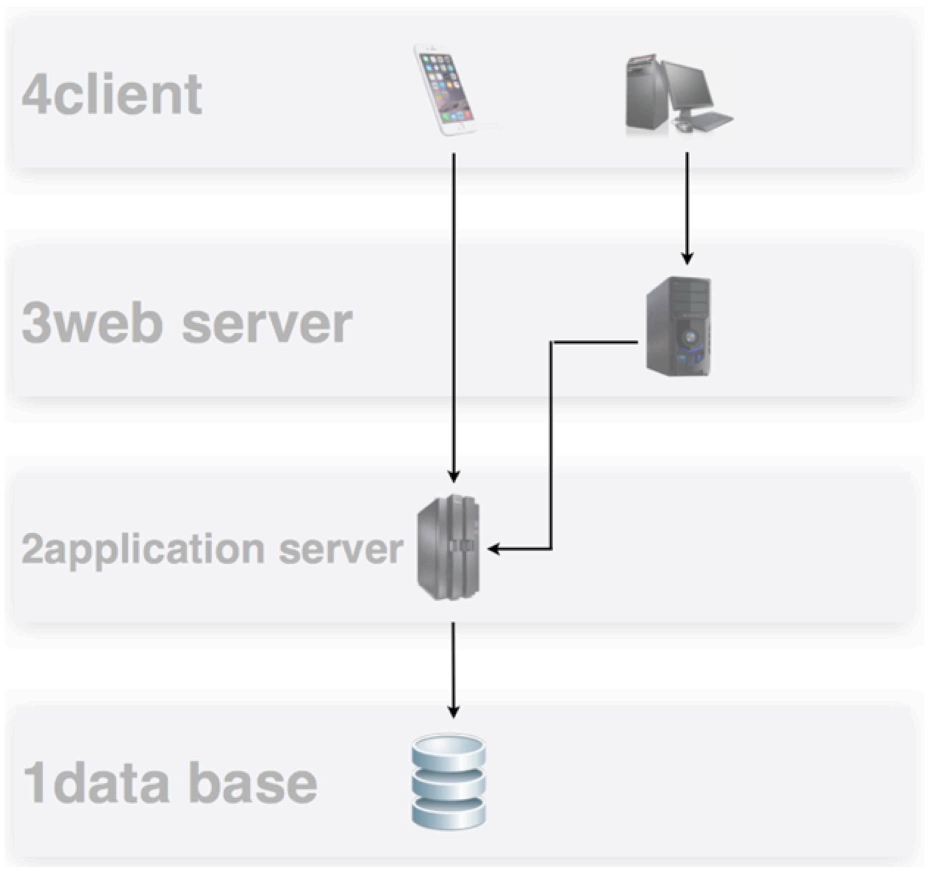


Figure 2.1. Layers of the system.

2.3 Component view

Component view here mainly focus on the application server part. The application server is implemented in the business logic tier using Java EE; it runs on GlassFish Server.
Session Beans used for the application server are shown in Figure 2.3. And the details for each session bean is described as following:

System Manager
This bean manages all the system control features of our application, namely: system parameter, data initial (car, user, safe area), personal date check, including driving license and bank payment, data backup.

Car manager
This bean handles mainly the physical status of cars, namely: battery, GPS, and sensors. Sensors here is what measures the the weight on the car (allows to check the correctness number of passengers.) And in our application will interact with Google map to provide GPS service.

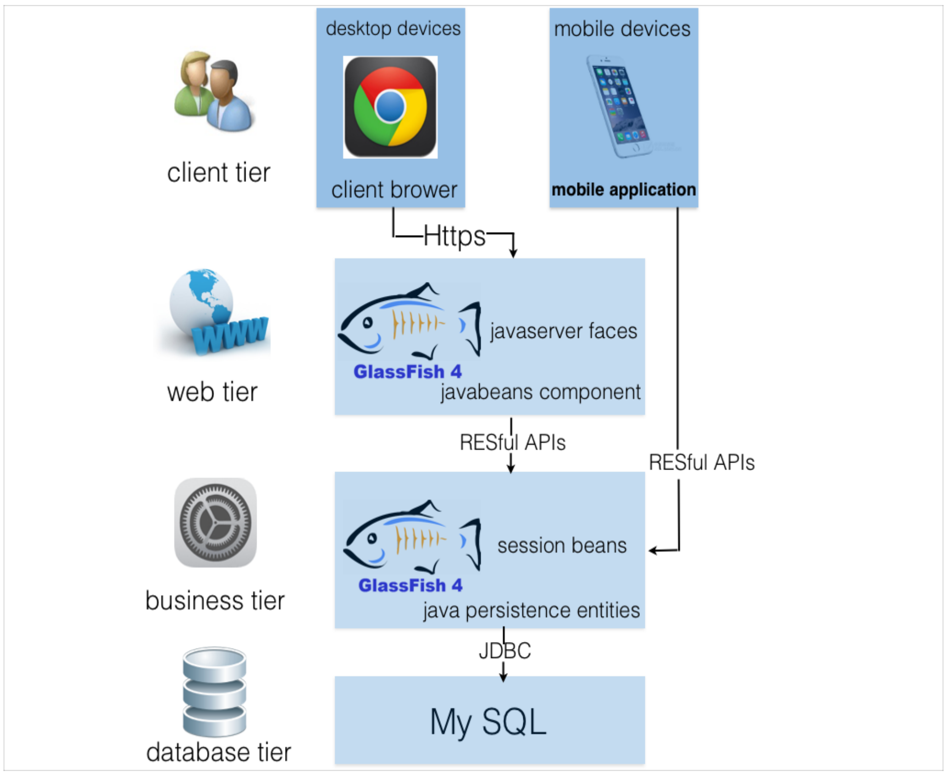


Figure 2.2. Layers of system in details

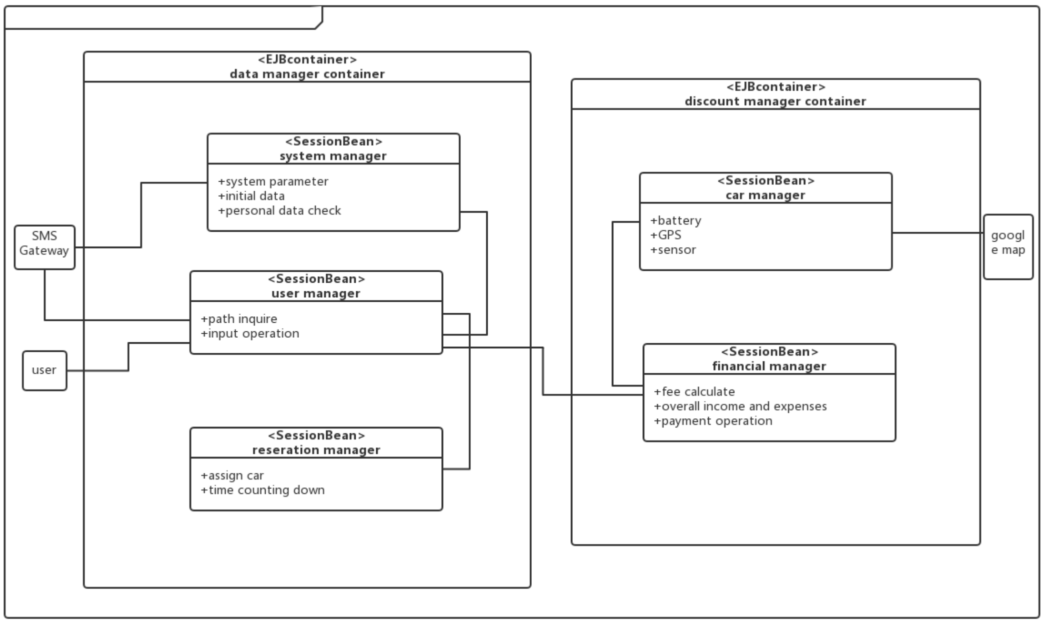


Figure 2.3 The session beans used to implement the business logic.

User manager
This bean deals with all the user operation features, including path inquire, login operation, input management (like personal information or selection of number of passenger, payment information).

Financial manager
This bean handles all financial management of our system, mainly fee calculation, payment operation and overall financial management. Fee calculation sums up the fee for each ride, which is basic fee with the penalty or discount fee. And this bean will interact with banks outside out system for user payment operation. In addition, function called overall financial management is to manage the cost and profit. Because for sure, the total cost and profit is a key interest for the car sharing business.

Reservation manager
This bean is in charge of manage the reservation of out system, including assign one car for one reservation and time-counting down control. Time-counting down refers to the time limit of each reservation, which is after one hour, the reservation will be expired.

2.4 Deployment view

Our device with software interaction is physically distributed among multiple processors, and *deployment diagram* in Figure 2.3 is used to describe how the components of our software system are mapped to hardware components. It also simulates the static deployment realization of our system.

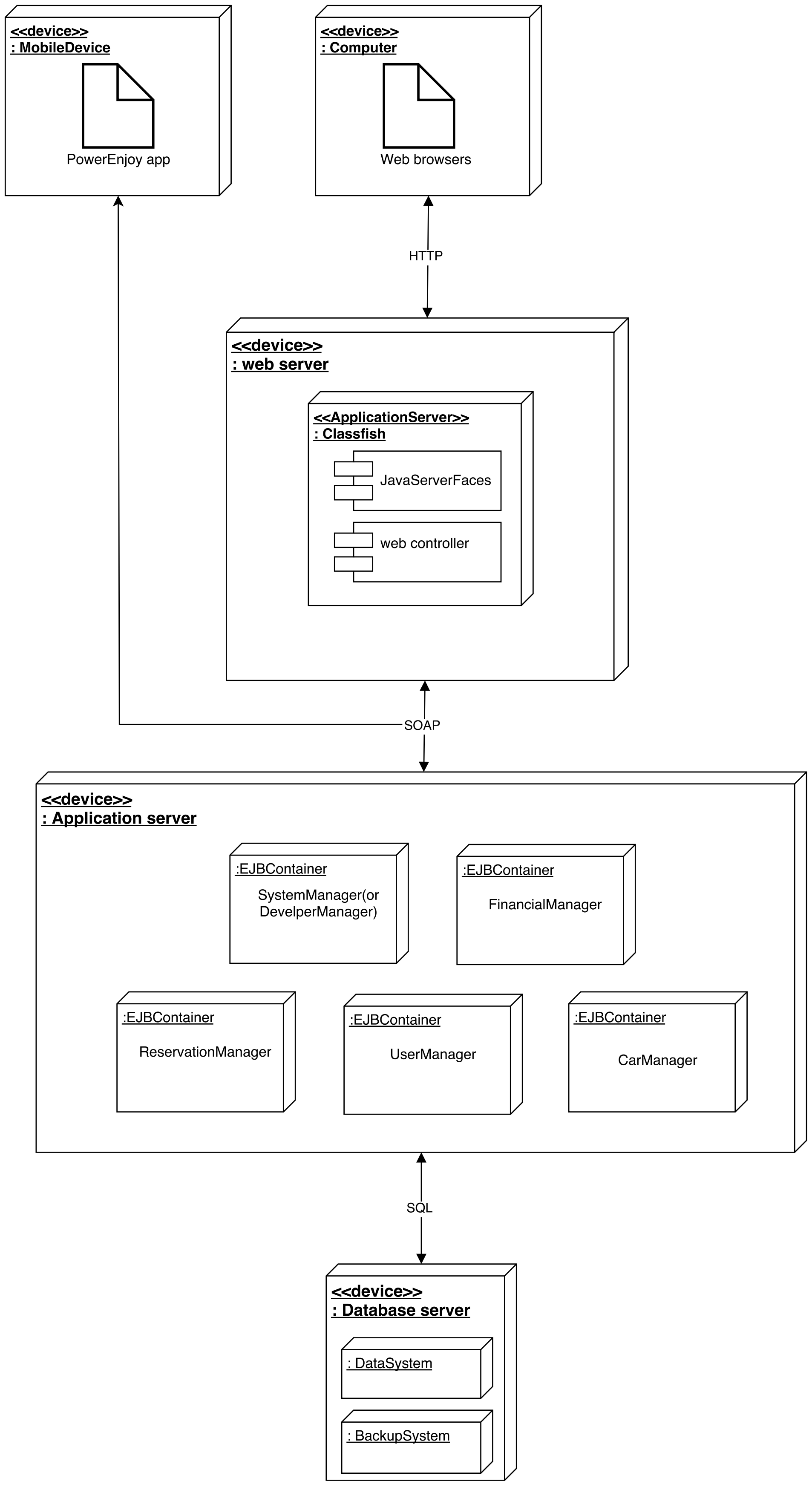


Figure 2.3 Deployment diagram

2.5 Runtime view

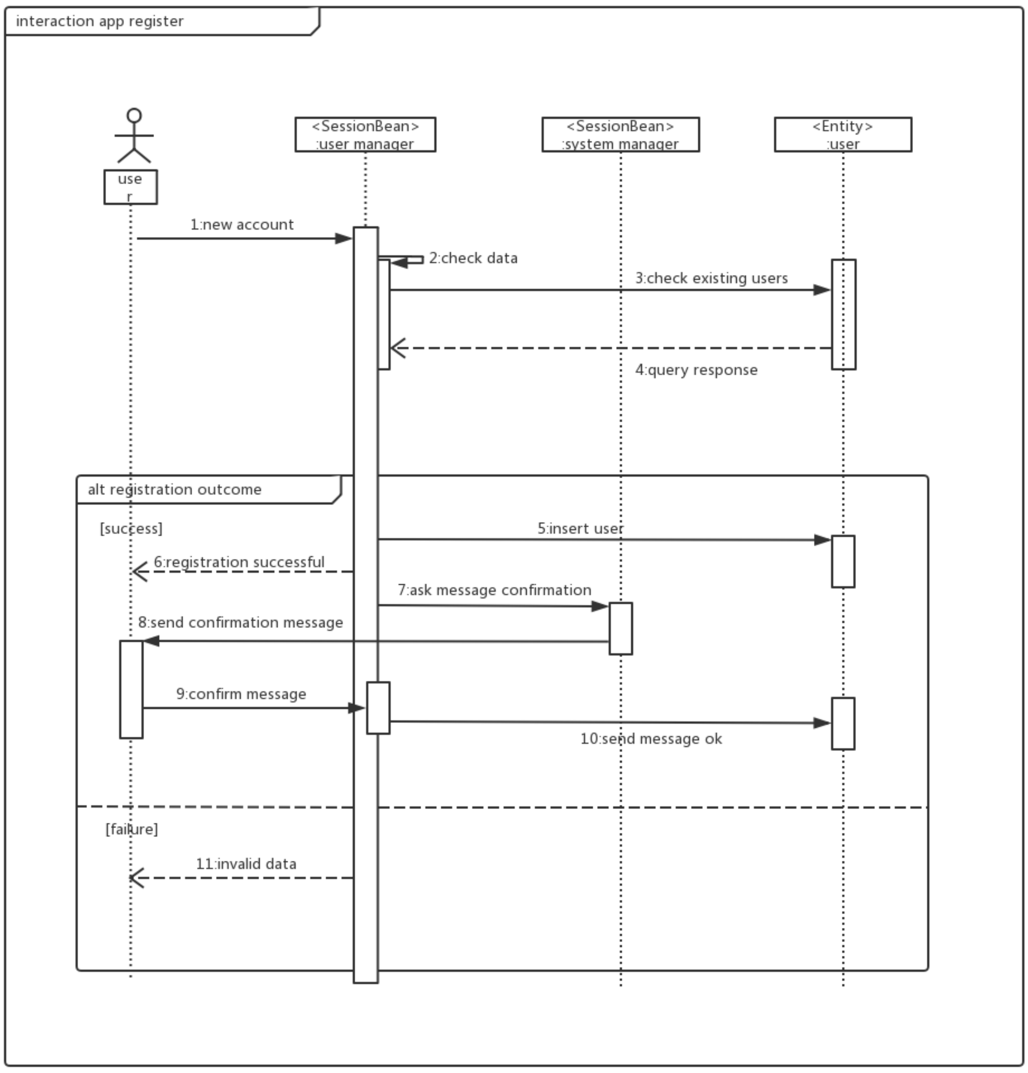


Figure 2.4. Sequence diagram of the user register

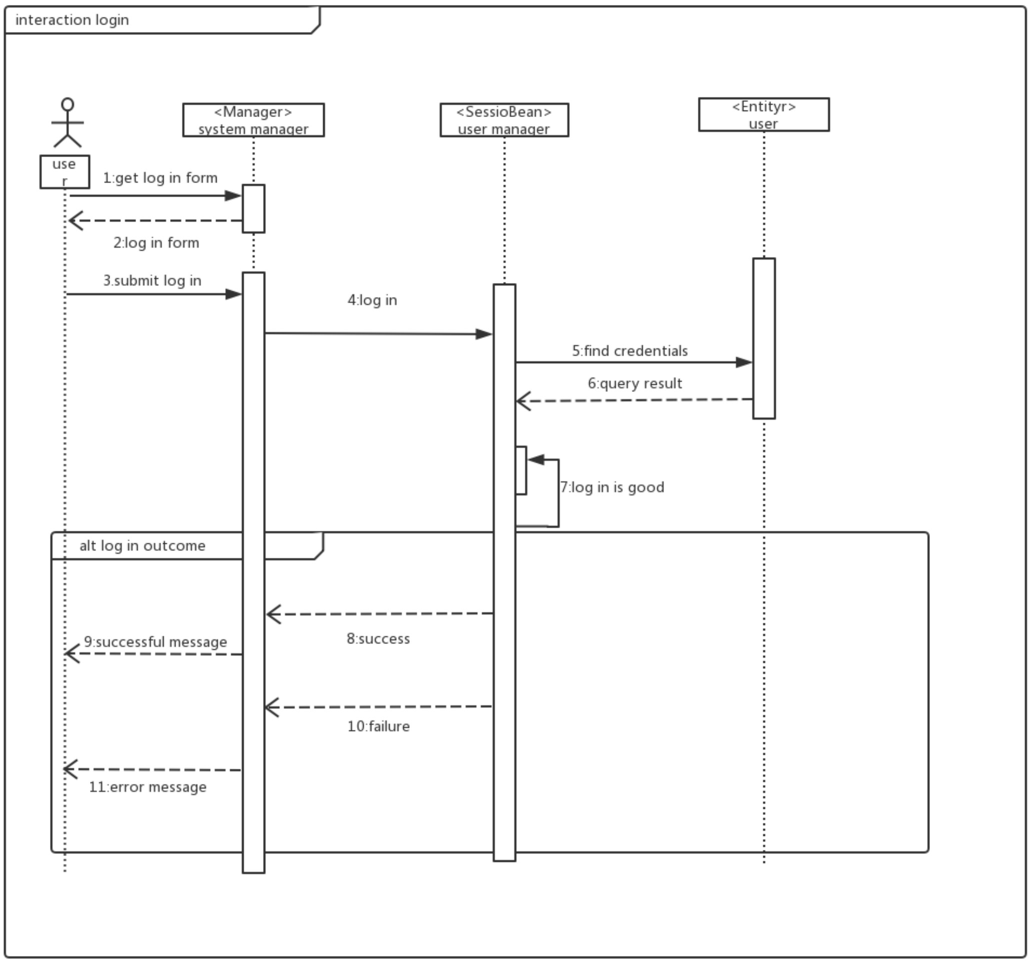


Figure 2.5. Sequence diagram of user login.

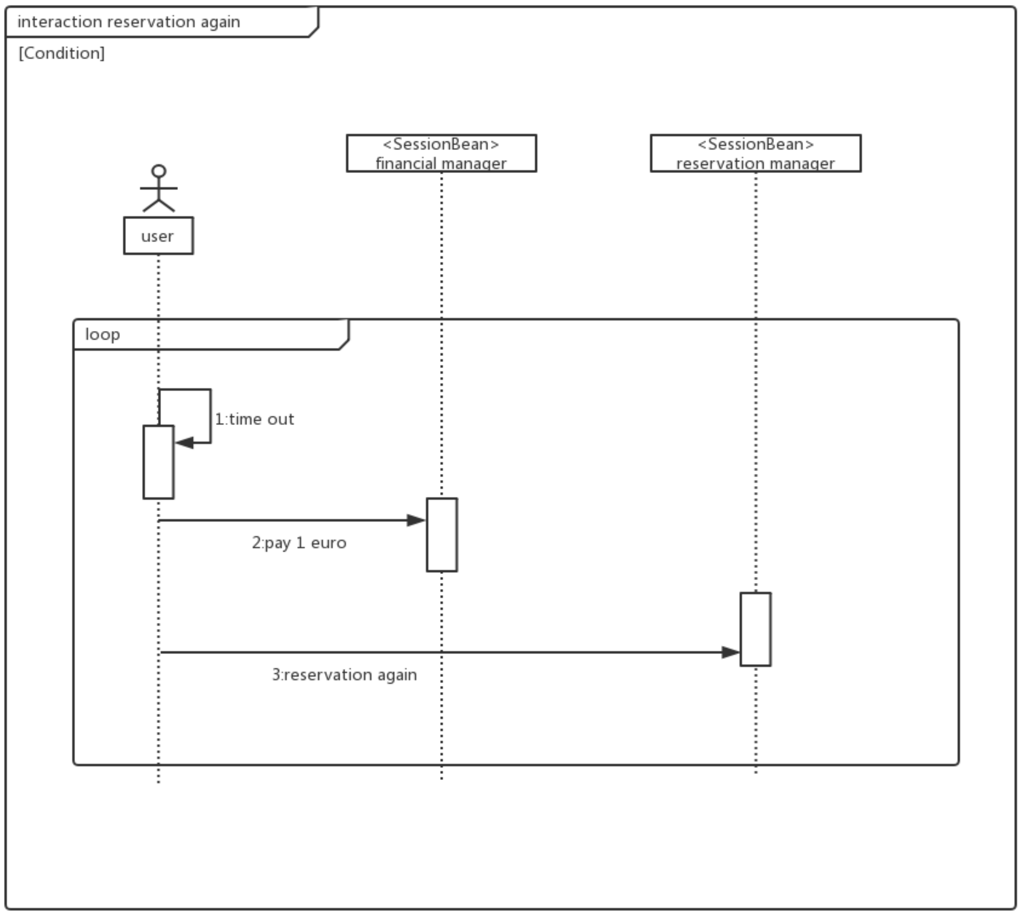


Figure 2.6 Sequence diagram of user re-reservation after an expired one.

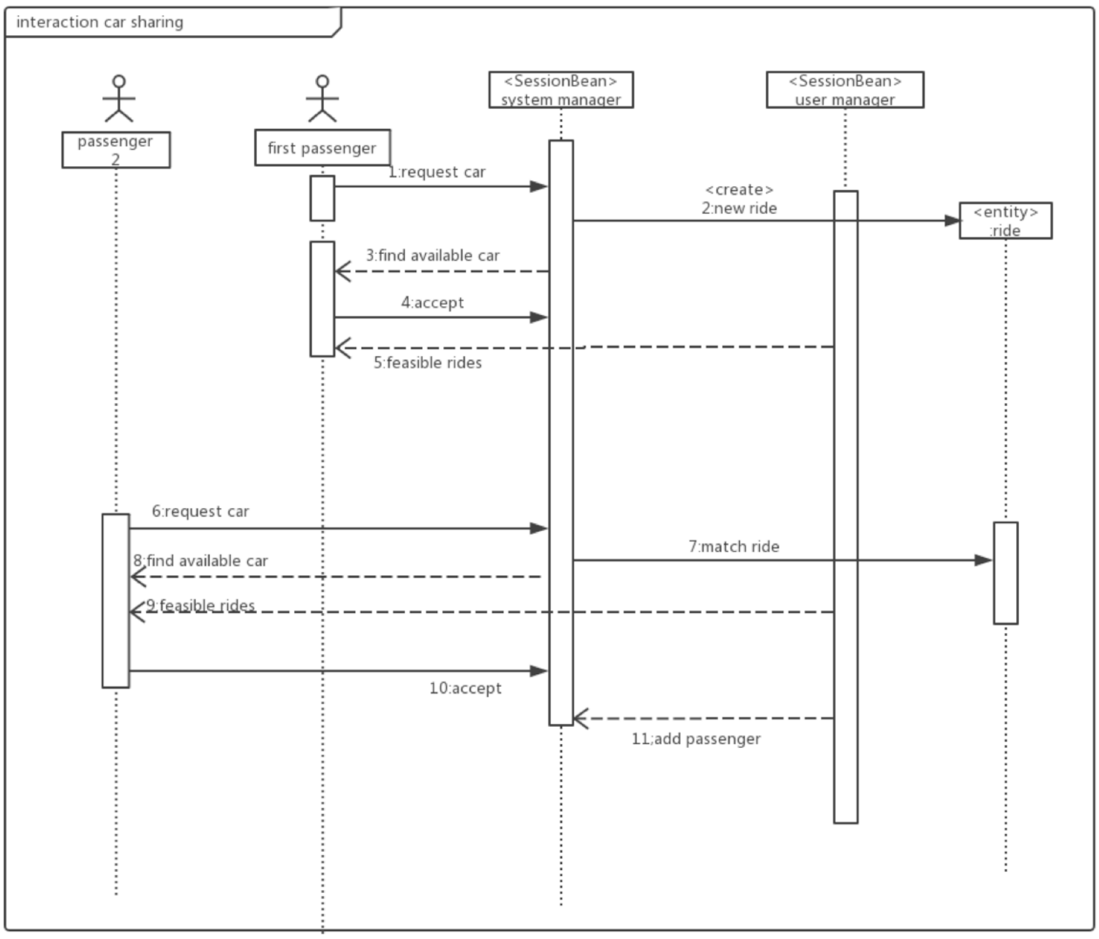


Figure 2.7 Sequence diagram of user searching for car-sharing ride.

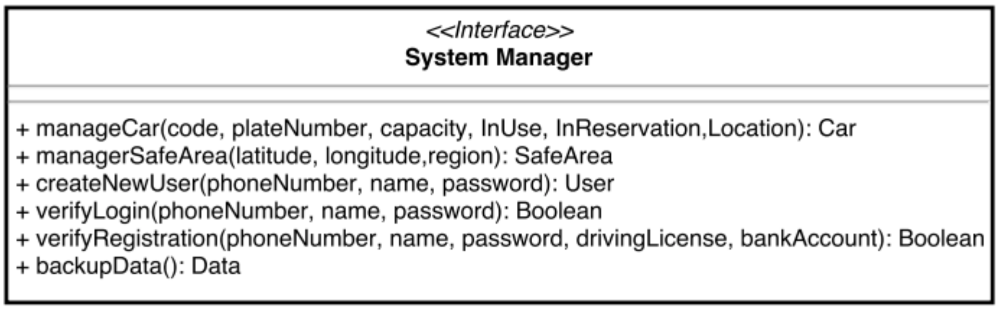
2.6. Component interface

The overall configuration of component interfaces is showed in Figure 2.8. we explain functions offered by each Beans of the Business Tier in details here:

System Manager

This system manager bean should have these methods:

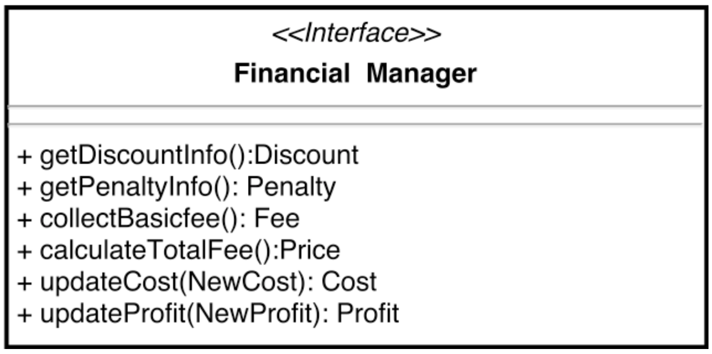
- ✓ Data initial: *createNewUser* to add in database, *manageCar* and *manageSafeArea*.
- ✓ Personal date check: *verifyLogin* and *verifyRegistration*. The two function *verifyLogin* and *verifyRegistration* are in Boolean data form. Registration information contains driving license and check bank account as data input.
- ✓ Data Backup: *backupData* and store them in database.



<<Interface>>
System Manager
+ manageCar(code, plateNumber, capacity, InUse, InReservation,Location): Car
+ managerSafeArea(latitude, longitude,region): SafeArea
+ createNewUser(phoneNumber, name, password): User
+ verifyLogin(phoneNumber, name, password): Boolean
+ verifyRegistration(phoneNumber, name, password, drivingLicense, bankAccount): Boolean
+ backupData(): Data

Financial manager

- ✓ This finacial manager bean should have these methods as follows:
- ✓ Fee calculation: *getDiscountInfo, getPenaltyInfo and collectBasicFee* to *caculateTotalFee.*
- ✓ Overall financial management: *updateCost* and *updateProfit* for in Database for the car sharing business.

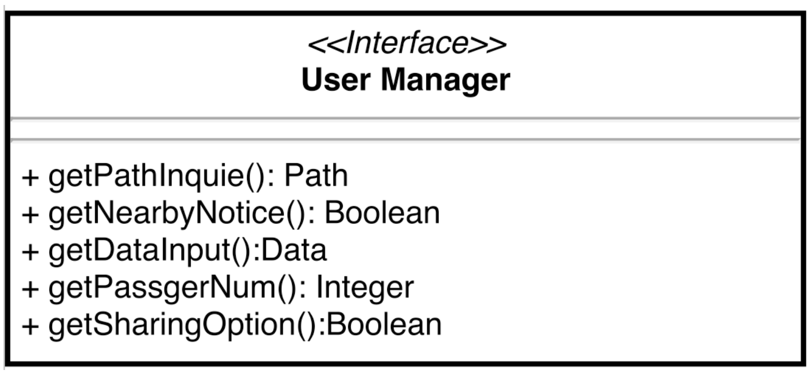


<<Interface>>
Financial Manager
+ getDiscountInfo():Discount
+ getPenaltyInfo(): Penalty
+ collectBasicfee(): Fee
+ calculateTotalFee():Price
+ updateCost(NewCost): Cost
+ updateProfit(NewProfit): Profit

User manager

This user manager bean should have these methods:

- ✓ *GetPathInquire* to retrieve the path inquire from users and store the destination in database which relates to search the safe area.
- ✓ UserInput: *getDataInput* to acquire data, *getNearbyNotice* to unlock the car, *getSharingOption* allow user to enable the car-sharing function, and *getPassengerNum* may allow user to get discount later.

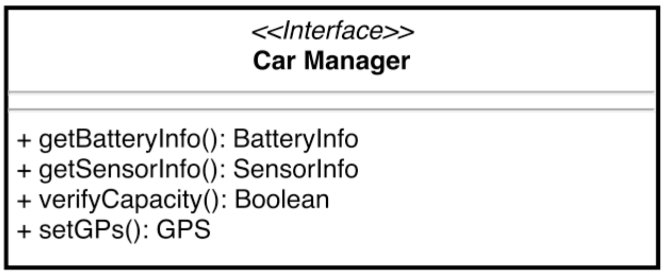


<<Interface>>
User Manager
+ getPathInquie(): Path
+ getNearbyNotice(): Boolean
+ getDataInput():Data
+ getPassgerNum(): Integer
+ getSharingOption():Boolean

Car manager

This car manager bean should have these methods:

- ✓ Battery: *getBatteryInfo* to obtain the battery state of the car
- ✓ Sensor: *getSensorInfo* and *verifyCapacityInfo.* The sensor measures the weights of the car. First, getSensorInfo to retrieve the weights of the car to get the current numbers of passenger. Then, verifyCapacity by comparing the number of passengers which users input with the sensor's reading.
- ✓ GPS: *setGPS* to guide road.

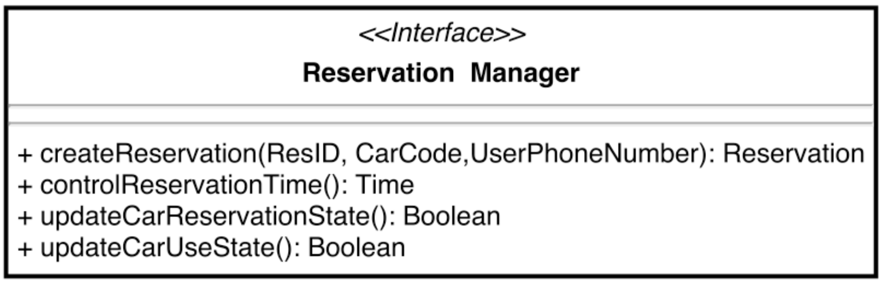


<<Interface>>
Car Manager
+ getBatteryInfo(): BatteryInfo
+ getSensorInfo(): SensorInfo
+ verifyCapacity(): Boolean
+ setGPs(): GPS

Reservation manager

This reservation manager bean should have these methods:

- ✓ *CreateRes:* search the car and relate the car, reservation and user together.
- ✓ *ControlReservationTime:* coun down time for one-hour reservation time limit.
- ✓ *UpdateCarRevervationState:* to update the car reservation state to be in reservation or not.
- ✓ *UpdataCarUseState:* to update the car use state to be in use or not.


<<Interface>>
Reservation Manager
+ createReservation(ResID, CarCode,UserPhoneNumber): Reservation
+ controlReservationTime(): Time
+ updateCarReservationState(): Boolean
+ updateCarUseState(): Boolean

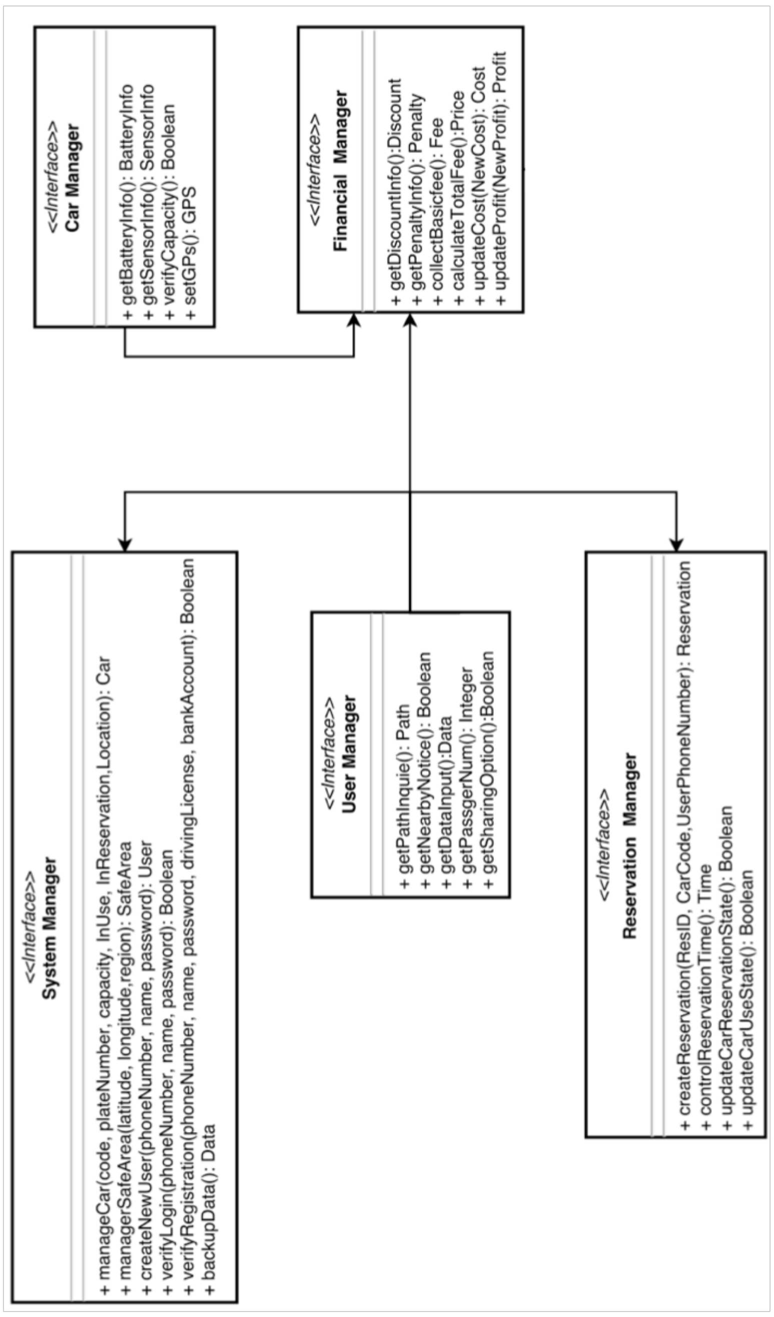

<<Interface>>
Car Manager
+ getBatteryInfo(): BatteryInfo
+ getSensorInfo(): SensorInfo
+ verifyCapacity(): Boolean
+ setGPs(): GPS
<<Interface>>
Financial Manager
+ getDiscountInfo():Discount
+ getPenaltyInfo(): Penalty
+ collectBasicfee(): Fee
+ calculateTotalFee():Price
+ updateCost(NewCost): Cost
+ updateProfit(NewProfit): Profit
<<Interface>>
System Manager
+ manageCar(code, plateNumber, capacity, InUse, InReservation,Location): Car
+ managerSafeArea(latitude, longitude,region): SafeArea
+ createNewUser(phoneNumber, name, password): User
+ verifyLogin(phoneNumber, name, password): Boolean
+ verifyRegistration(phoneNumber, name, password, drivingLicense, bankAccount): Boolean
+ backupData(): Data
<<Interface>>
User Manager
+ getPathInquie(): Path
+ getNearbyNotice(): Boolean
+ getDataInput():Data
+ getPassgerNum(): Integer
+ getSharingOption():Boolean
<<Interface>>
Reservation Manager
+ createReservation(ResID, CarCode,UserPhoneNumber): Reservation
+ controlReservationTime(): Time
+ updateCarReservationState(): Boolean
+ updateCarUseState(): Boolean

Figure 2.8 Component Interfaces

2.7 Selected architectural styles and patterns

Layered style, 3 tiers, is chosen as our architecture styles, and showed in Figure 2.9.

2.7.1 Architecture styles

We want to separate functionality with layers. In addition, we want that Server provides service and client requests and use service. The communication between Service and Client is only though interface.

- UI layer (mobile app/user's browser)
- Application-Logic layer (web server/application server)
- Database layer

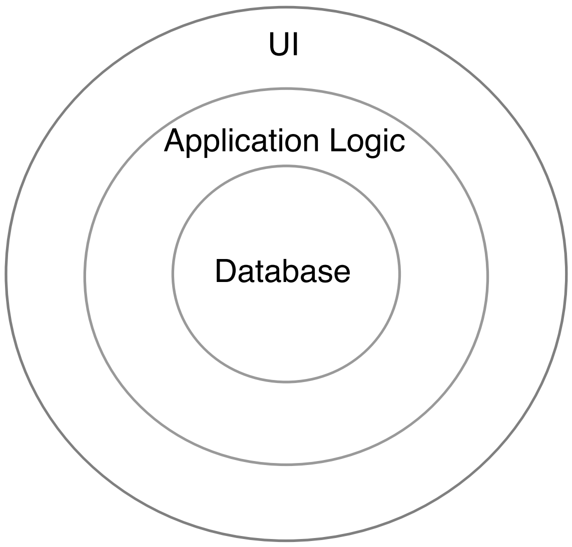


Figure 2.9 Layered style of our system

2.7.2 Patterns

Model-View-Controller

Model-View-Controller design pattern is the main pattern thought our system. First, data and how to represent data are two important parts in our front-end design. And by using model-view-controller pattern, data and the presentation layer can be separated from each other. Second, Controller is self-contained, and is independent from Model and View, which makes it is convenient to change the data layer and business rules without interfering controllers. This property also brings in good flexibility and the compatibility.

<u>Thin Client</u>

The thin client is implemented to let the application run on low-resources devices. The data of thin client is processed on server instead of hard desk. The advantage of the thin client is that the management cost and hardware cost are low, and data protection is easy.

<u>Client-Server</u>

Our web page used the client-server pattern. For example, when a client using our website to make a reservation, the client PC and web browser is a client, and PowerEnjoy Database, application, computers is a server. When client requests a reservation, the server searches the Database and composes the information to a web page to the client. Server invokes to provide one of a set of services Client uses the provided services and initiates communication. They are accessible only through interface.

2.8 Other design decisions

2.8.1 Password

- Password Requirement: The system will have some specific requirements of the password which users use to register, like an acceptable password must be longer than 8 characters and less than 20 characters, at least with 1 capitalized letter and no special symbol inside, which is less likely to be in a dictionary and, in general, requires more time to discover.
- Password Notification: When users register, a notification with sentence, that is "an acceptable password must be longer than 8 characters and less than 20 characters, at least with 1 capitalized letter and no special symbol inside", will be under the password inserting field, which reminds the users to put one feasible password.
- Password Hash: when user registered, with hash function SHA-512, the password will be hashed before stored. when user login, the password will be hashed then compared with the one stored with the same user name.

2.8.2 Driving licenses check

The drivers are divided into two types, European user and non-European user:

- For European drivers, system can check and confirm the driving licenses based on a third party organization, like the website http://europa.eu/youreurope/driving-licence.
- For non-European user, they can call the Customer Services, then after checked by our worker their driving license will be checked and marked as available.

2.8.3 UI language

So far, the UI (User Interface) will be displayed only in English language, more languages can be added according to the popularity and development of this software product in the future.

2.8.4 Map

Google map will be the default map in our system because of its popularity and good performance.

Chapter 3

Algorithm design

3.1 Shared car

This is the algorithm that is responsible to manage the Shared Rides. Here there are some general information about the 3 steps algorithm:

First, there is **Algorithm 1, Compatibility Check** to check the potential reservations satisfied the conditions that can be merged together.

Second, **Algorithm2, Merge Car** to create a new element to merge the car together, and copy the information to the new element.

Third, **Algorithm 3**, **Insert New Element** creates a list with elements of possible reservations in List2, ordered with descending order of expire time. And the List2 is illustrated in Figure 3.1 as follows.

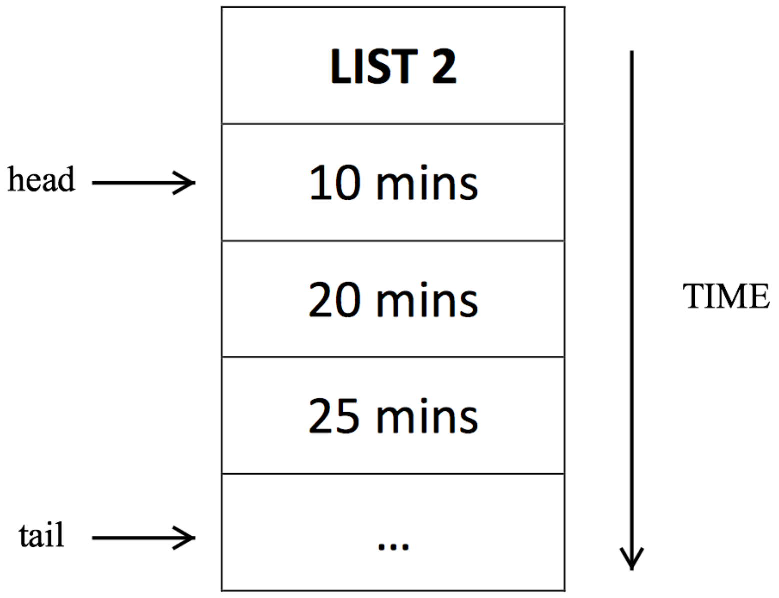


Figure 3.1 structure of List2

Algorithm 1 Compatibility check

```
1.procedure CompatilityCheck
2.   for each reservation r1 in List1(x)   do
3.   if (r1.startposition == x.startposition) &&
(SharingOption==ture) &&(NofP≤ Capacity) &&(r1.expiretime<
r2.expire time+1 hour)          //ckeck if satisfy all car-sharing
condition and expired less than 1 hour than r2 expire time
         then
4.           MergeCar(r1)
5.             break()
6.       end if
7.   end for
8. end procedure
```

Algorithm 2 MergeCar

```
1.procedure MergeCar(r1,r2)
2.   new x     //create a new revervation element x
3.       if r1.expiretime<  r2.expiretime
4.            then x.expiretime←r1.expiretime, copy(r1)
4.            else x.expiretime←r2.expiretime, copy(r2)
5.       end if
6. end procedure
```

Algorithm 3 With Desecding Order, insert new element in List2

```
1.procedure Insert(r1,List2)
2.   for  ∀r in List2
3.          if (L.head==L.tail)   //Empty list
4.               then break
2               else if (r.expiretime>  rl.expiretime)
6.              successor(predecessor(r)) ←x
7.              predecessor(r1) ←ppredecessor(r)
8.              successor(r1)   ←r
9.              predecessor(r)  ←r1
10.        break()
```

```
11.        end if
12.    end for
13. end procedure
```

3.2 Fee calculation

The algorithm is run by the back-end to calculate the fee. The function is giving as follows. Price is the total fee that a user has to pay in one ride.

$$\boldsymbol{Price = B(t)(1 - 10\% x - 20\% y - 30\% z + 80\% w)}$$

And the parameters are defined as follows:

- $\boldsymbol{D}$: distance from nearest grid station.
- $\boldsymbol{Q}$: $\boldsymbol{a\ set\ of\ special\ parking\ areas, where\ car\ can\ be\ charged.}$
- $\boldsymbol{NofP}$: $\boldsymbol{number\ of\ pasengers\ the\ user\ takes\ onto\ the\ car.}$
- $\boldsymbol{PA}$: $\boldsymbol{parking\ area, can\ be\ an\ element\ of\ Q.}$
- $\boldsymbol{Battery}$: $\boldsymbol{percentage\ of\ the\ car's\ battery\ which\ has\ been\ used.}$
- $\boldsymbol{B(t)}$: $\boldsymbol{basic\ price, a\ function\ of\ time.}$

$$\boldsymbol{x = \begin{cases} 1, & NofP \geq 2\ , \\ 0, & otherwise. \end{cases}}$$

$$\boldsymbol{y = \begin{cases} 1, & Battery \leq 50\%, \\ 0, & otherwise. \end{cases}}$$

$$\boldsymbol{z = \begin{cases} 1, & PA \in Q \\ 0, & otherwise. \end{cases}}$$

$$\boldsymbol{w = \begin{cases} 1, & D \geq 3km\ or\ Battery \geq 80\% \\ 0, & otherwise. \end{cases}}$$

Chapter 4

User Interface Design

4.1 UX Diagram

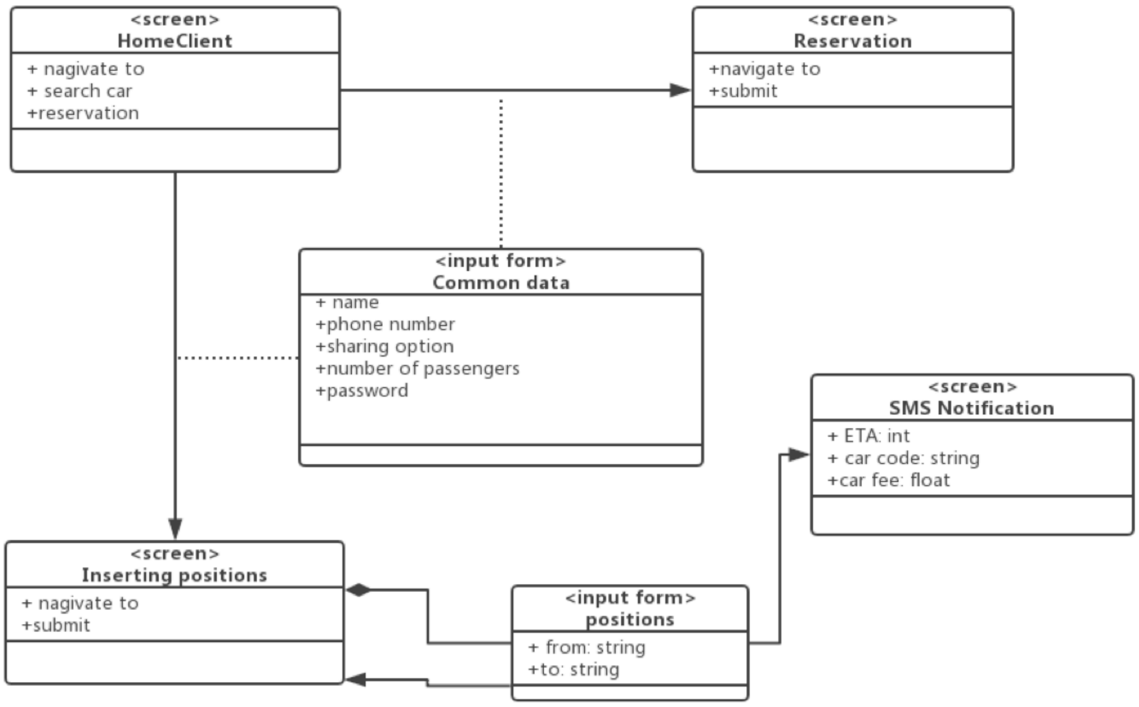


Figure 4.1 UX User Mobile

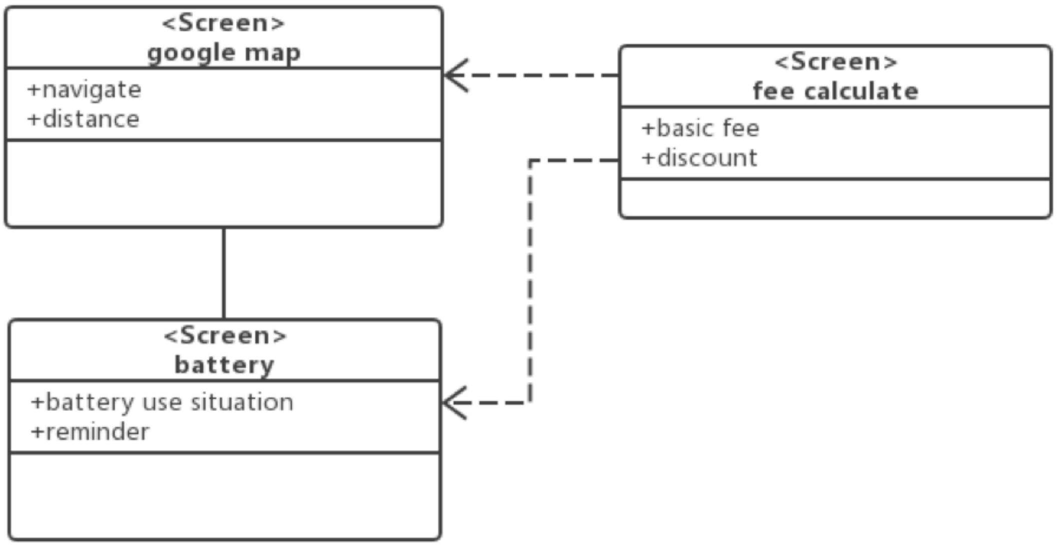


Figure 4.2 UX Car Screen

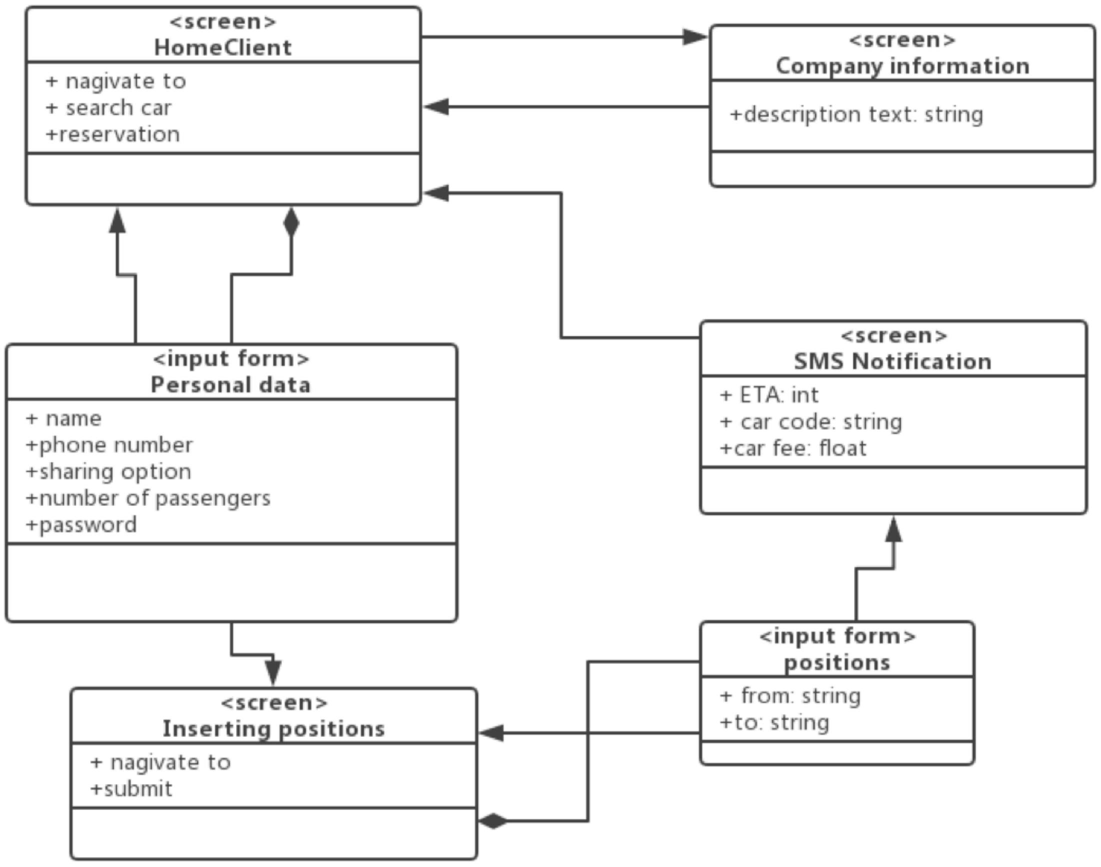
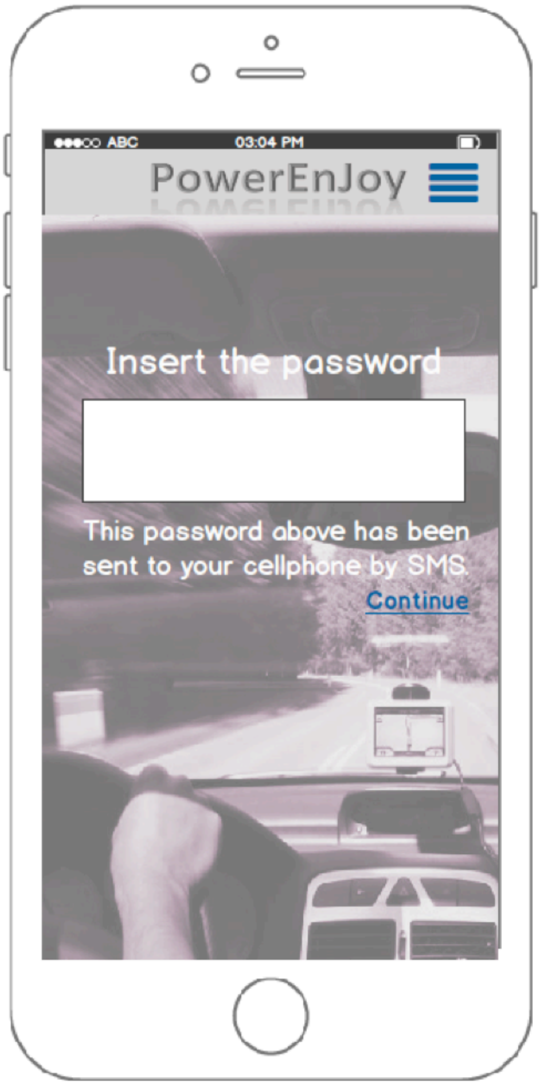


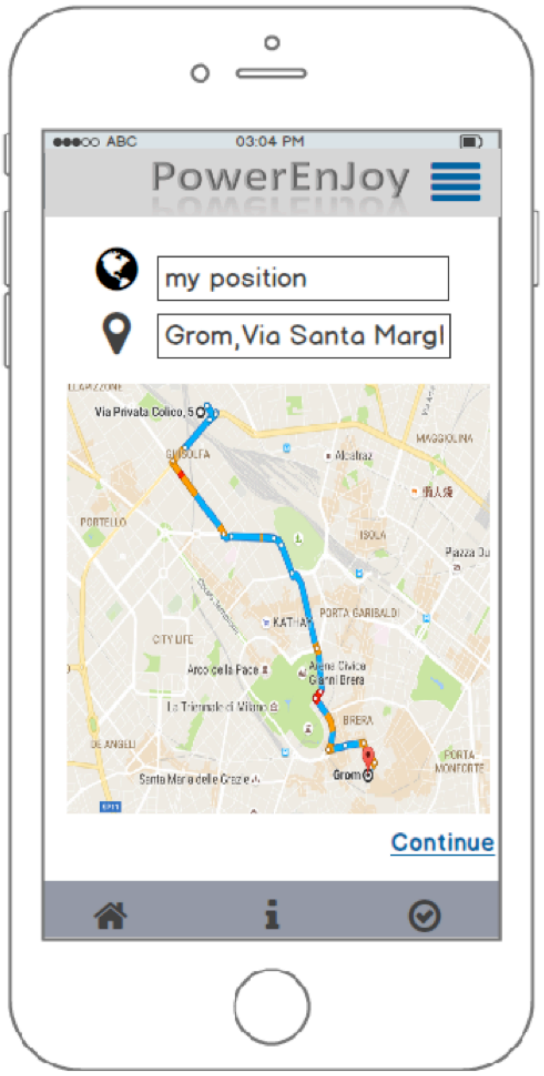
Figure 4.3 UX User Web

4.2 User interface concept

4.2.1 User mobile interface

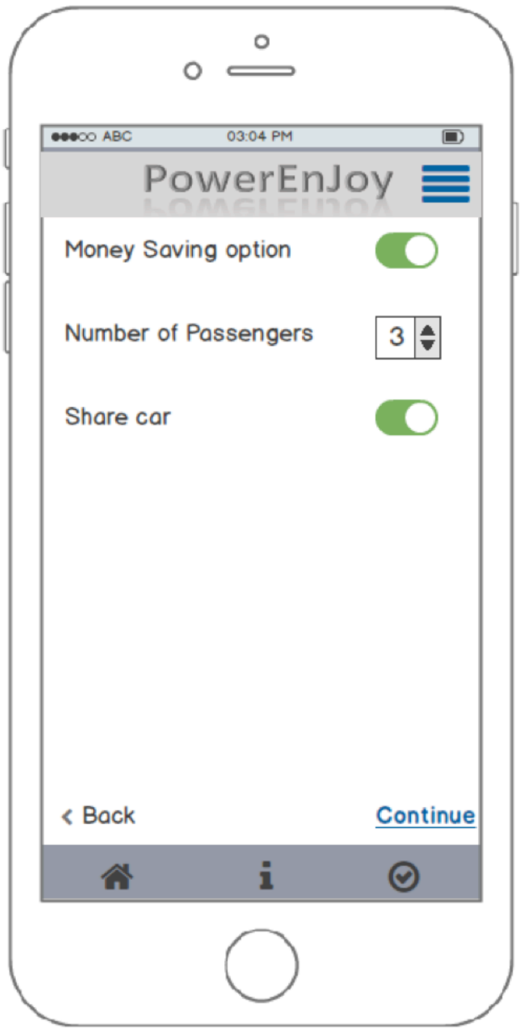


This password above has been sent to your cellphone by SMS

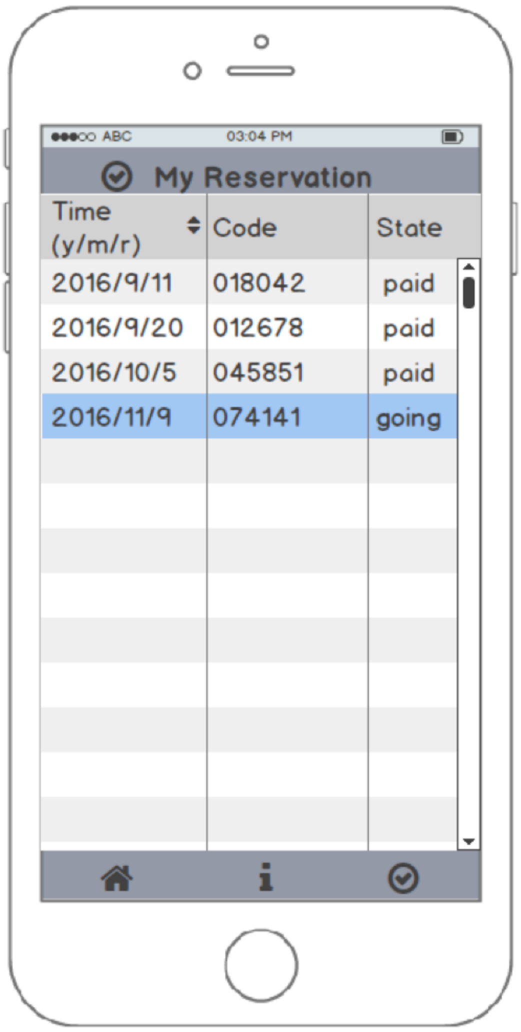


my position

Grom, Via Santa Margl

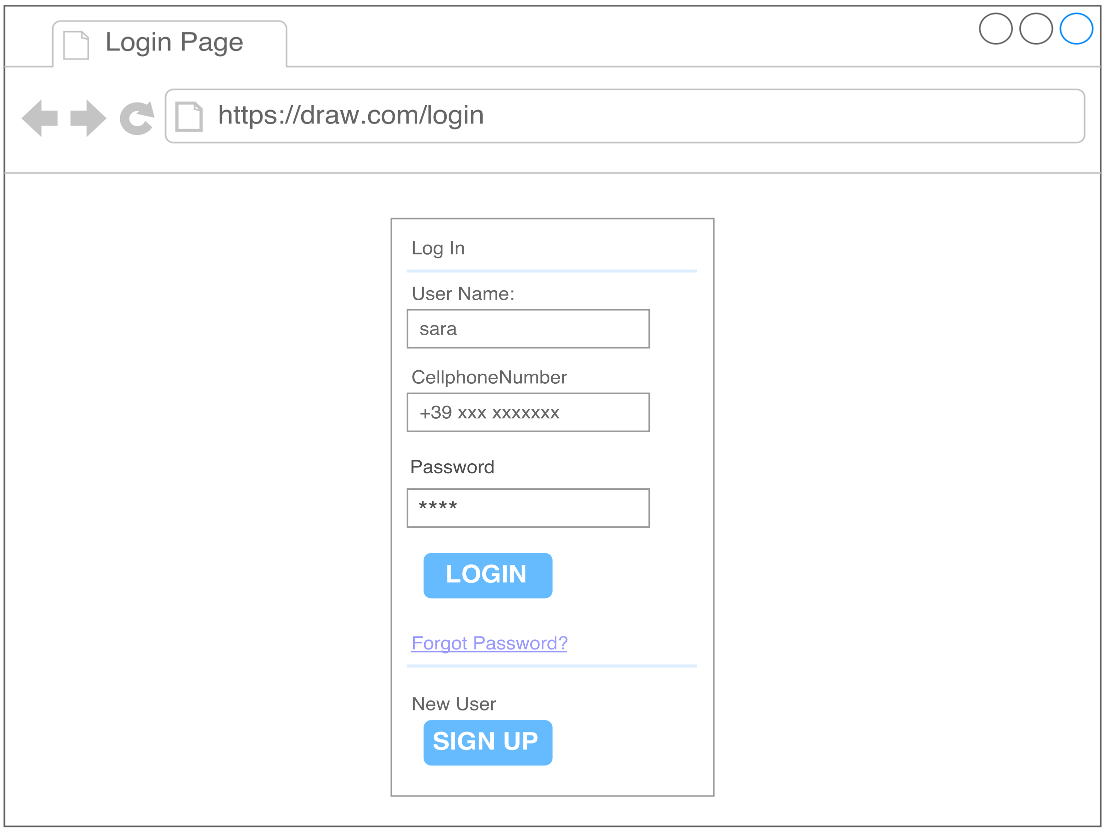


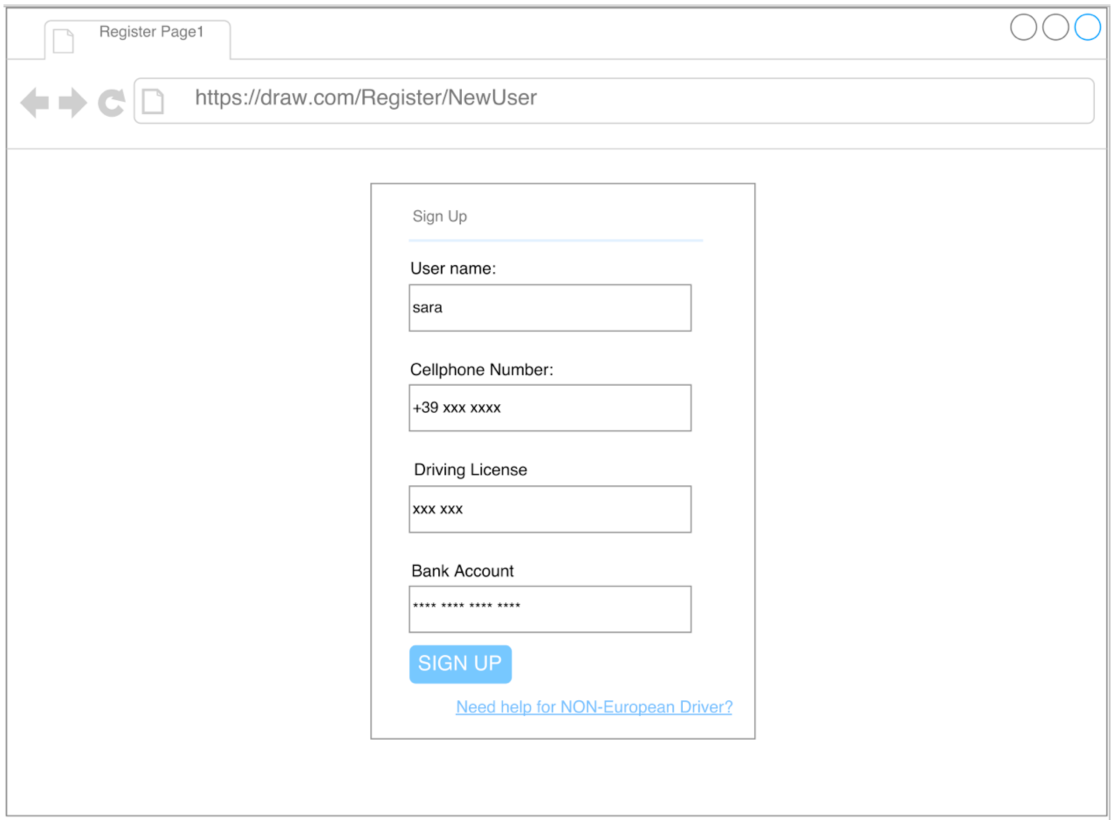
Money Saving option	🟩
Number of Passengers	3
Share car	🟩

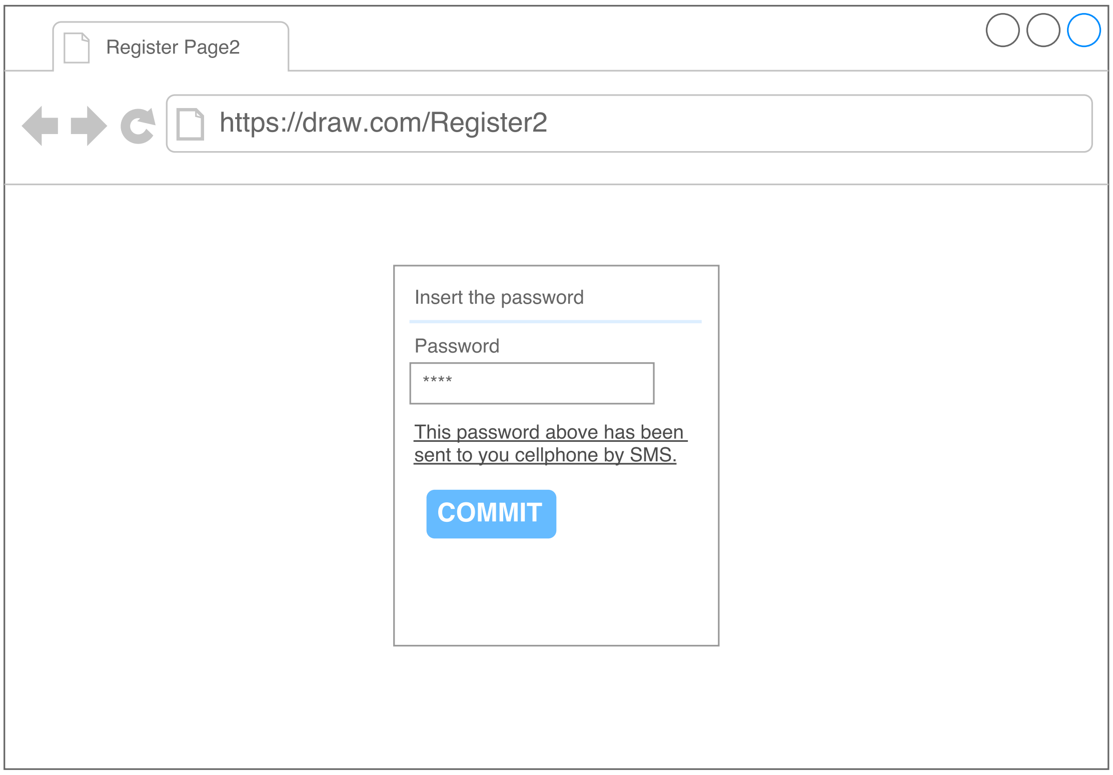


Time (y/m/r)	Code	State
2016/9/11	018042	paid
2016/9/20	012678	paid
2016/10/5	045851	paid
2016/11/9	074141	going

4.2.2 User web interface





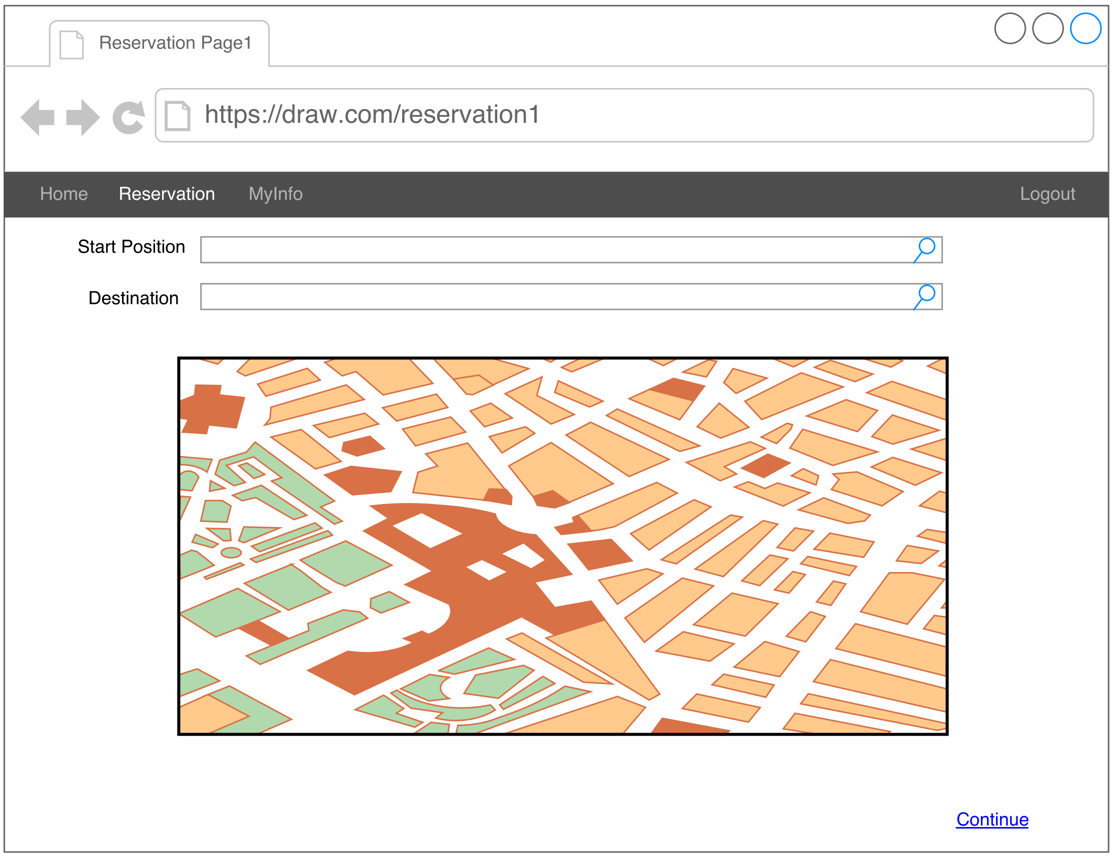


Insert the password

Password

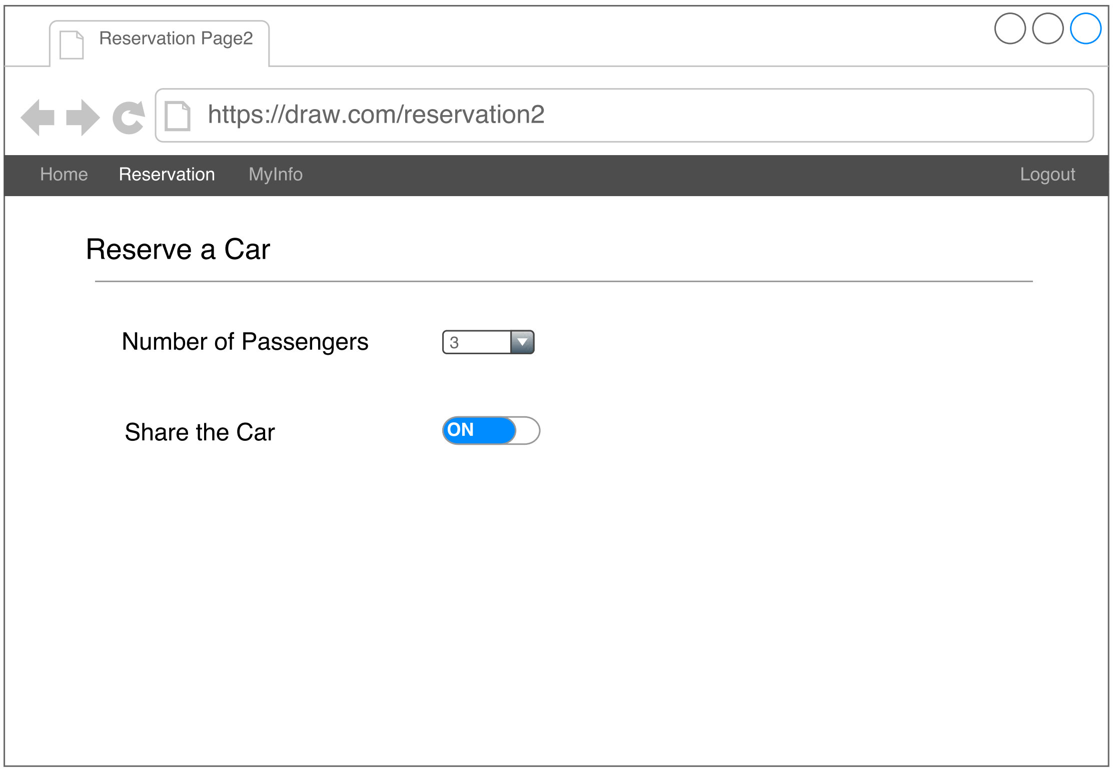
****

This password above has been sent to you cellphone by SMS.



Start Position

Destination

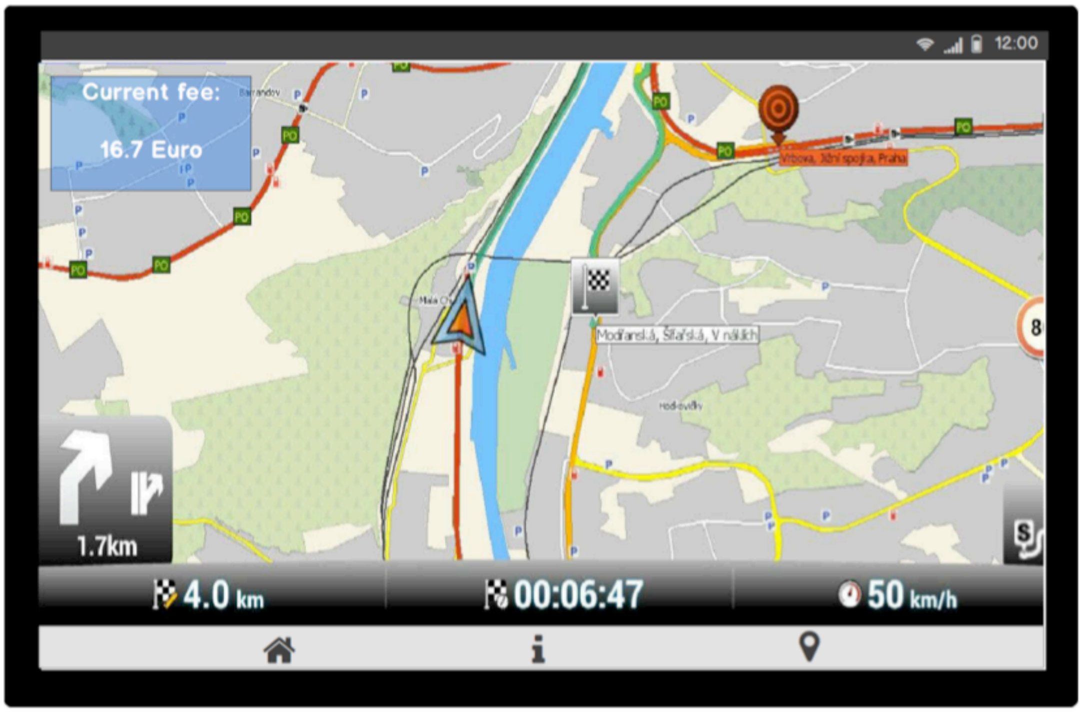


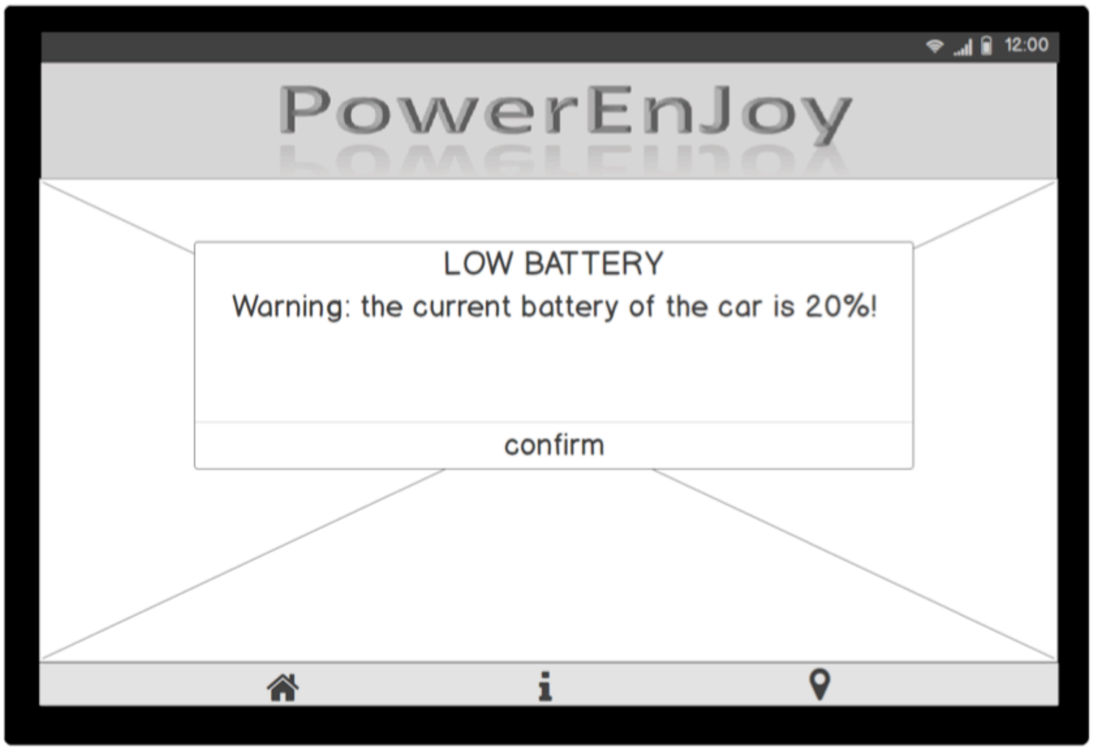
Reserve a Car

Number of Passengers: 3

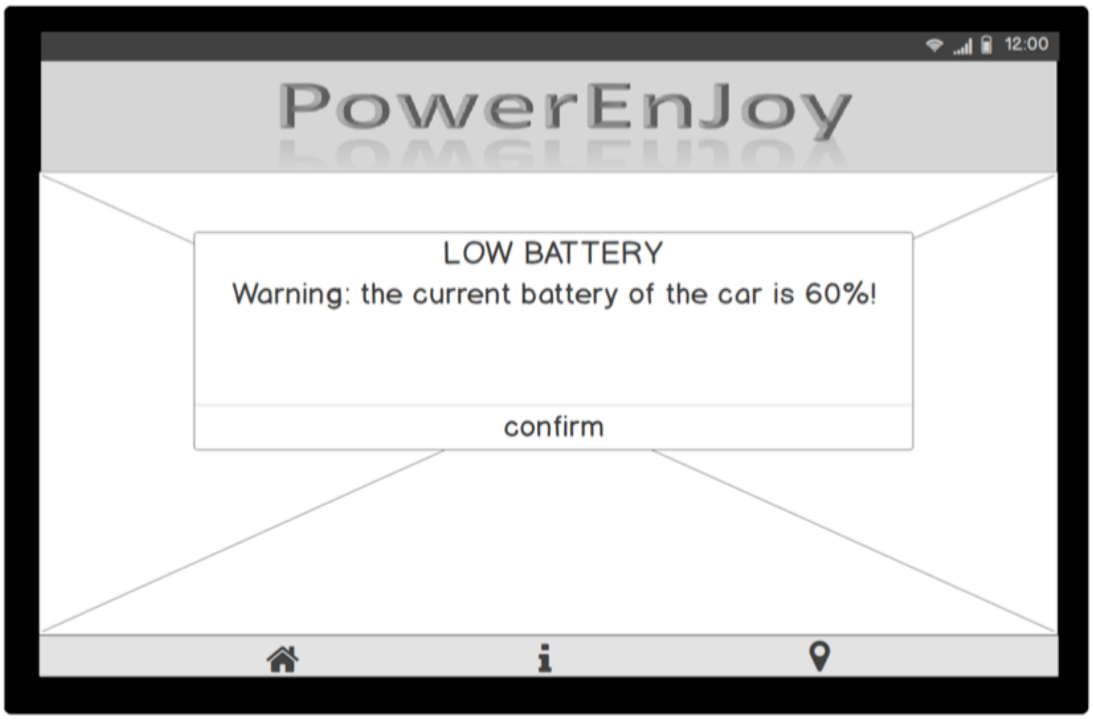
Share the Car: ON

4.2.3 Car Screen interface

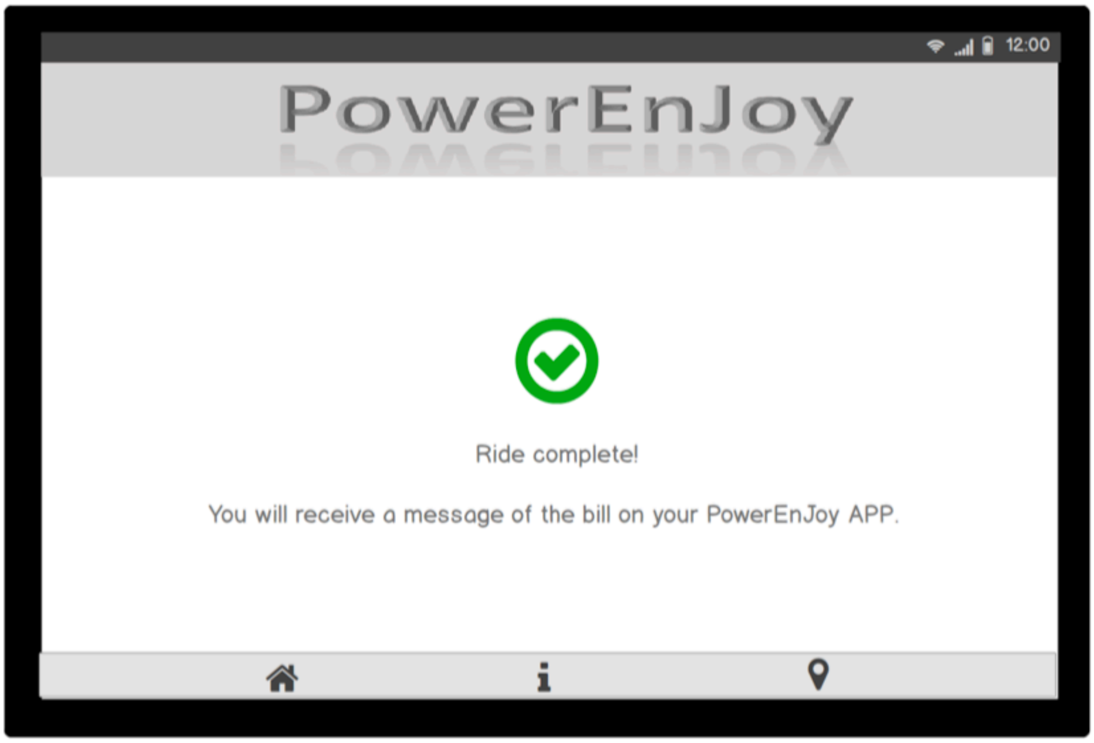




LOW BATTERY
Warning: the current battery of the car is 20%!
confirm



LOW BATTERY
Warning: the current battery of the car is 60%!



Ride complete!

You will receive a message of the bill on your PowerEnJoy APP.

Chapter 5

Requirements traceability

5.1 Functional requirements and components

DD Component	**RASD requirement**
System manager	G.1 System aware of state of car G.7 User login and register
Financial manager	G.3 Calculate fee G.4 System can give user discount/penalty fee G.10 User can make payment
User manager	G.6 User can input destination G.8 Path inquire G.11 User can input number of passengers, open car-sharing option.
Car manager	G.2 Car not overload G.5 Situation of road
Reservation manager	G.9 User reserves the car

5.2 Non-functional requirements

1. Users can see the battery state all the time.
2. Users can find the company information in the web interface.
3. Car system and google can give users the best route.

5.3 Modifications to the RASD

In previous RASD document, users can only access our application via mobile phone APP. Now we add the web front-end. So the client who has desktop can access to our service via websites.

We put company information in web interface. So client can contact us.

Chapter 6

Effort spent (hour of work)

Jia HongYan

• 15/11/16: 2h

• 17/11/16: 2h

• 19/11/16: 2h

• 20/11/16: 2h

• 21/11/16: 1h

• 23/11/16: 1h30

• 25/11/16: 3h

• 28/10/16: 5h

• 1/11/16: 4h

• 3/12/16: 1h30

• 5/12/16: 3h

• 7/12/16: 3h

• 9/12/16: 6h

• 10/12/16: 3h

Lang Shuangqing

• 16/11/16: 2h

• 18/11/16: 2h

• 19/11/16: 3h

• 21/11/16: 2h

• 22/11/16: 4h

• 24/11/16: 2h

• 26/11/16: 3h

• 29/11/16: 4h

· 1/12/16:2h

· 3/12/16:2h

· 4/12/16: 3h

· 7/12/16:4h

· 9/12/16: 2h

·10/12/16: 2h

Chapter 7

Reference Documents

This document refers to the following documents:

- *Design Part 1 - Slides*
- *Design Part2 - Slides*
- *DD document, MyTaxiService 2015-2016 - Chitti Eleonora, De Nicolao Pietro, Delbono Alex*
- *RASD and Design Document Assignment – Software Engineering 2 project*
- *Reasoning on design through an example - Slides*
- *Template for the design document - Software Engineering2*
- *Project rules of the Software Engineering 2 project*
- *Software engineering: principles and practice (Vol. 3). Van Vliet, H., Van Vliet, H., & Van Vliet, J. C. (1993).*

Chapter 8

Used tools

Google drive: a tool for version controller and group cooperation.
ProcessOn: a tool to make sequence diagram part.
Draw.io: a tool for mockup and UML diagrams.
Word: a tool to write down the document.
Github: a tool for submit this document

Chapter 9
Change log

V1.1
-Added the definition of OO in chapter 1.3.
-Drew figure 2.6 again, cancelled user operation.

V1.2
-Modified component view.
-Modified sequence diagram models.
-Added Web front-end.

V1.3
-Modified runtime view.
-Fixed the overall format.
-Added the sequence diagram of user re-reservation after an expired one.