POLITECNICO
MILANO 1863

Design Document

Version: 1.0

Reda Aissaoui, Jinling Xing, Lidong Zhang

December 11, 2016

Content

1. Introduction
    1. Purpose
    2. Scope
    3. Definition, acronyms, abbreviations
    4. Document structure
2. Architecture design
    1. Overview
    2. High level components and their interaction
    3. Component view
    4. Deploying view
    5. Run-time view
    6. Component interfaces
    7. Selected architectural styles and pattern
    8. Other design decisions
3. Algorithm design
    1. Bill process
    2. Reservation process
    3. Money saving process
4. User interface design
    1. Mock-ups
    2. UX flow chart
5. Requirements traceability
6. Reference
    1. Bibliography
    2. Used tools
7. Hours worked

1. Introduction

1.1. Purpose

In this document, a detailed design is presented. It is base on the RASD document and provides a more detailed view of the architecture of PowerEnjoy's application

The RASD presented a general view of the system and what are the functions that the system should execute. This document aims to present the how of the implementation the system including components, run-time processes, deployment, user experience and algorithm design.

1.2. Scope

The project PowerEnjoy, which is a service based on mobile application(based on Android) and web application. The system allows user to reserve a electric car via mobile app or web app. When user wants to reserve a car, the GPS function will locate the user's current position and the system will filter available cars which are close to user's current position. Also users can apply for the money-saving option when they are ready to finish using the car. And the system will show the steps for users to get a discount once their situation qualify the requirement.

The main purpose of the system to make people's life more convenient by making cars available through an app. All this while protecting the environment through carpooling and electric vehicles.

1.3. Definition, acronyms, abbreviations

- **RASD**: Requirements Analysis and Specifications Document.
- **DD**: Design Document.
- **API**: Application Programming Interface: it is a common way to communicate with another system or service.
- **GUI**: Graphical User Interface.
- **MVC**: Model View Controller is a design pattern used for GUIs.
- **REST**: Representational state transfer (REST), it is a structure style of software.
- **DBMS**: Database Management System.
- **CRUD**: The usual Create, Read, Update and Delete that a user or system can do.
- **HTTP**: Hyper Text Transfer Protocol is the main protocol used for the world wide web.
- **URI**: Uniform Resource Identifier which is the set of characters that identify the location of a resource

- **JSON**: JavaScript Object Notation is a lightweight data-interchange format.
- **UX**: User experience diagram.
- **GPS**: Global Positioning System.

1.4. Document structure

- Introduction: This part briefly introduces the purpose of this document. And it also states the definition of some special words to help reader understand this document.

- Architecture Design: This section contains 8 parts:
- Overview: this section explains the architecture of the PowerEnjoy system
- High level components and their interaction: In this part, we give the communication mechanism of each component.
- Component view: this part shows all the controllers and models in this application and how they work together to assure the system work together.
- Deploying view: this section shows how each component of the system is deployed and in which device.
- Run-time view: this section contains the sequence diagrams of different components and how they work together in order to execute functions.
- Component interface: this part shows the different interfaces between components
- Selected architectural styles and patterns: the application involves different architectures and motivations behind the choice.
- Algorithm design: a description of the main algorithms.
- User Interface design: in this part, we involve what we have done in the RASD
- Requirements Traceability: this section shows how this document fulfills the goals stated before in the RASD.

2. Architecture design

2.1. Overview

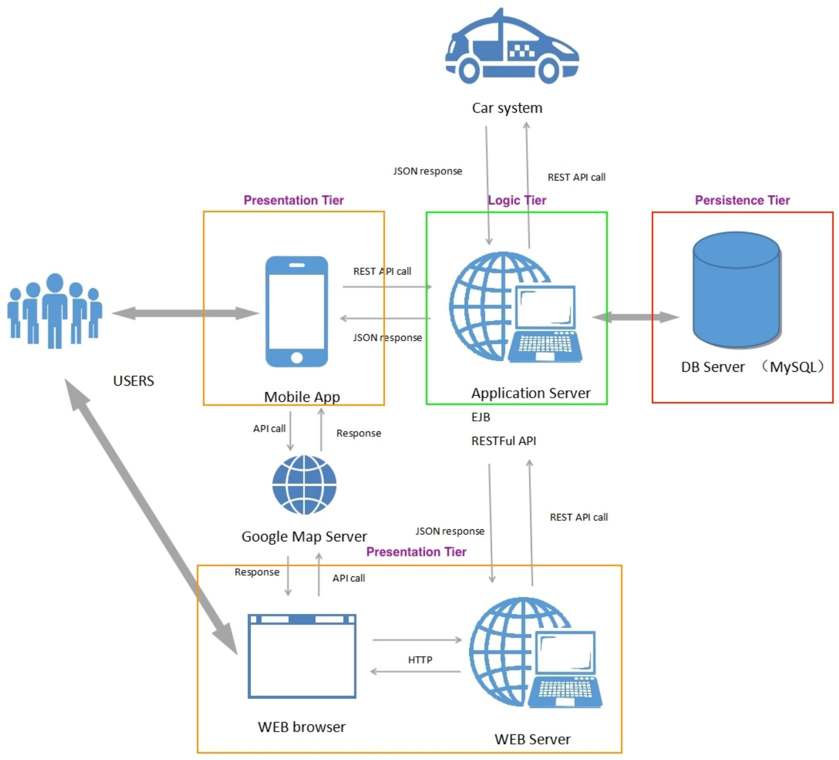


Figure 1: Application Architecture

As stated in the RASD document, we decided to have a 3-tier architecture. The figure above highlights the 3 tiers of our application:

- **Presentation tier**: Composed of he mobile application, web browser and web server, this tier takes care of formatting the data for user viewing.
- **Logic tier**: This tier holds the business logic of the application. It is composed of the application server.
- **Persistence tier**: This tier persists the data used in our application. It is composed of a DBMS.

2.2. High level components and their interaction

The figure above describes the high level components and their interaction. It is based on the application architecture presented in the RASD Document in

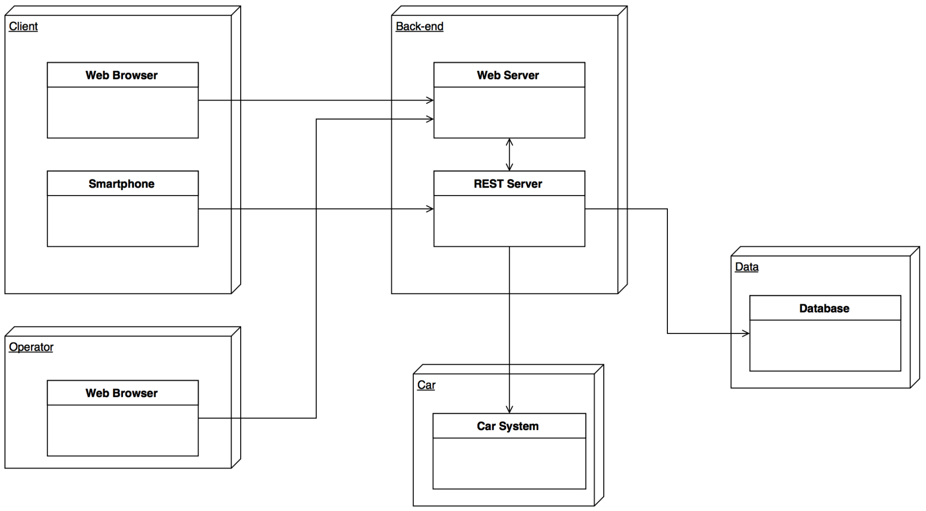


Figure 2: High Level Components

the Section 5. The REST Server is the main component in our application as it is the central point between the GUI (Web and mobile), the database and the cars. It holds the business logic of our system.

Web browser

Any common web browser can be used by the client or the operator to access the application. The user and the operator have different access levels. The user can register, manage his account and reserve a car through the website. The operator, on the other hand, have more rights in the website. He can check the localization of all the cars despite their status. He can also do all CRUD operations on the cars and users. The operator can as well validate the users when he checks their driving license.

Smartphone

The mobile application is designed uniquely for the user. As for the website, it can be used to register, manage account and reserve a car. The mobile application is also used to unlock the car a user reserved using the position of the device. At the end of a ride, the user can check the bill on the mobile application.

Web Server

The web server is the component that takes part of formatting and serving pages to the client/operator. No business logic is held in this component.

Application Server

This server holds all the application's business logic. It is the core of the application. It interfaces with other components through a REST API. The mobile application communicates directly with this server to operate. The web browser in the other hand do not have a direct access to this server. It is the duty of the web server to communicate get date from the REST Server, format it and the follow it to the web browser.

Database

The database is used to persist data generated and used by the application. For security and integrity reasons, the database is only connected the REST Server. Other components are indirectly connected through the latter.

Car Component

It is the abstraction of the on-board computer. It take care of gathering all car variables (battery levels, location, passenger, locking system…) and sending them to the application server that uses them for operations.

2.3.Component view

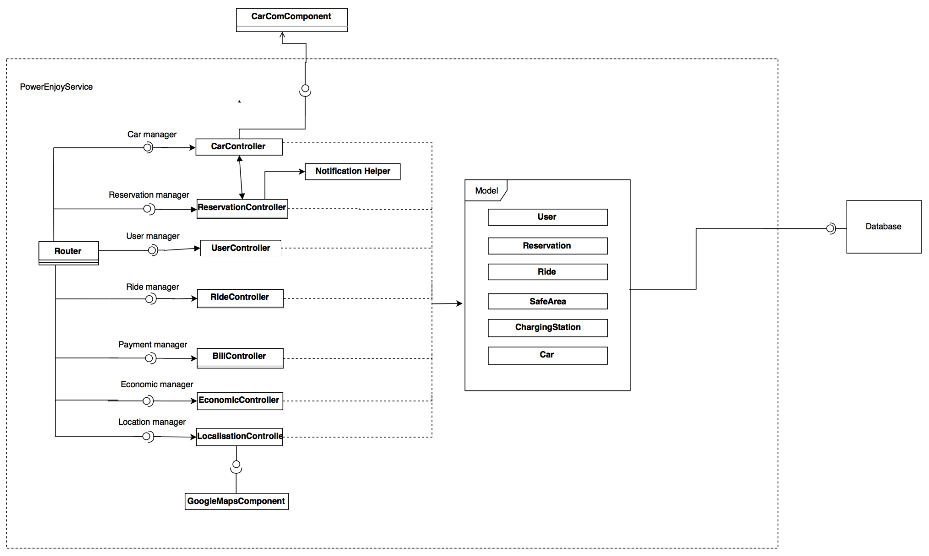


Figure 3: Component View Diagram

- Notification Helper : Manage notifications, noticing the user that they are already close to the car
- Ride Controller : manage rides
- Reservation Controller : manage reservation
- Bill Controller : manage payment method and bills
- Economic Controller : manage money saving request
- Car Controller : manage the status and availability of cars
- Router : route the request to related controller
- Clients : mobile application based on Android and web application (In browser)
- User Controller : manage user, access log in or sign in operations
- Database : store persistent data
- Localization Controller : manage location

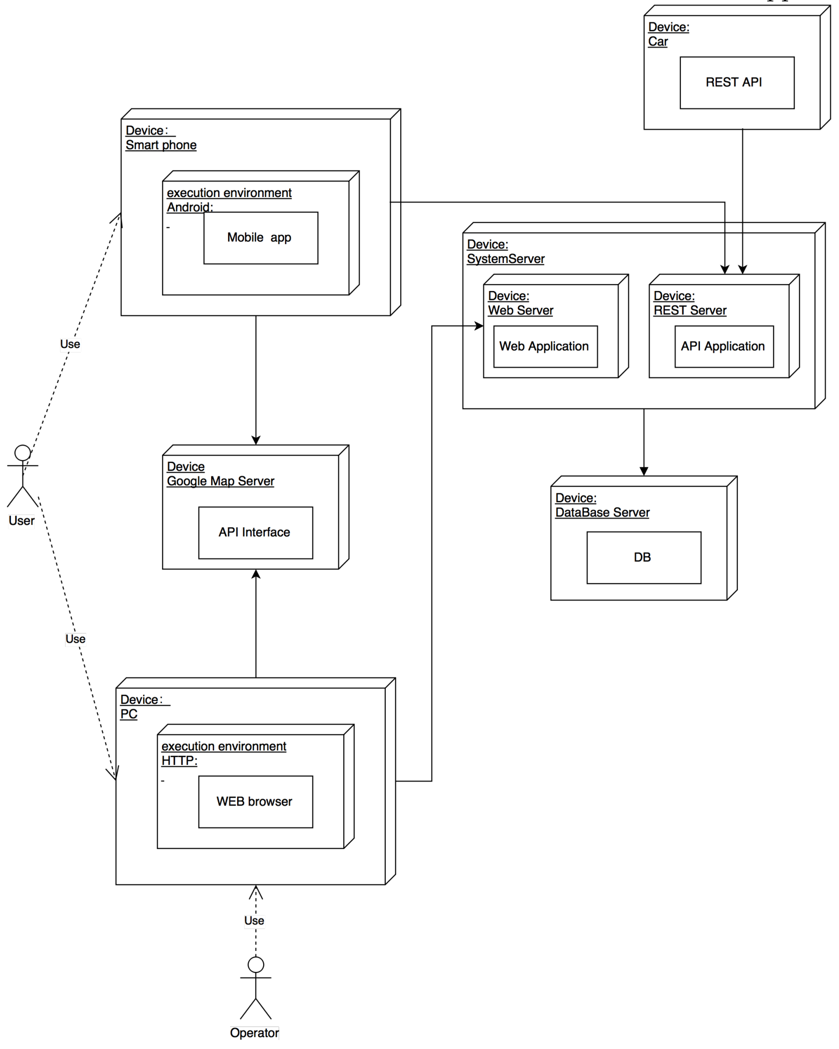
The system exposes a REST API with many public endpoints and resources, most of them require a proper authentication and authorization to be used.

The router will dispatch the request to the relevant controllers. Then each controller will do further process as the user required.

For example, when a reservation request comes it will arrive to the router first then it will be dispatched to reservation controller by router then the controller will dispose the request.

2.4. Deploying view

Deploying view gives the correct design of components and when all the components work, the deploying view should make sure the running view correctly. The user can use personal computer or smart phone to make a reservation, both of computer and smart phone use google map to fix location. In the system, we set system server to save the related information and the application server used the REST API socket. It also has a database server to support system.



2.5. Run-time view

Login process

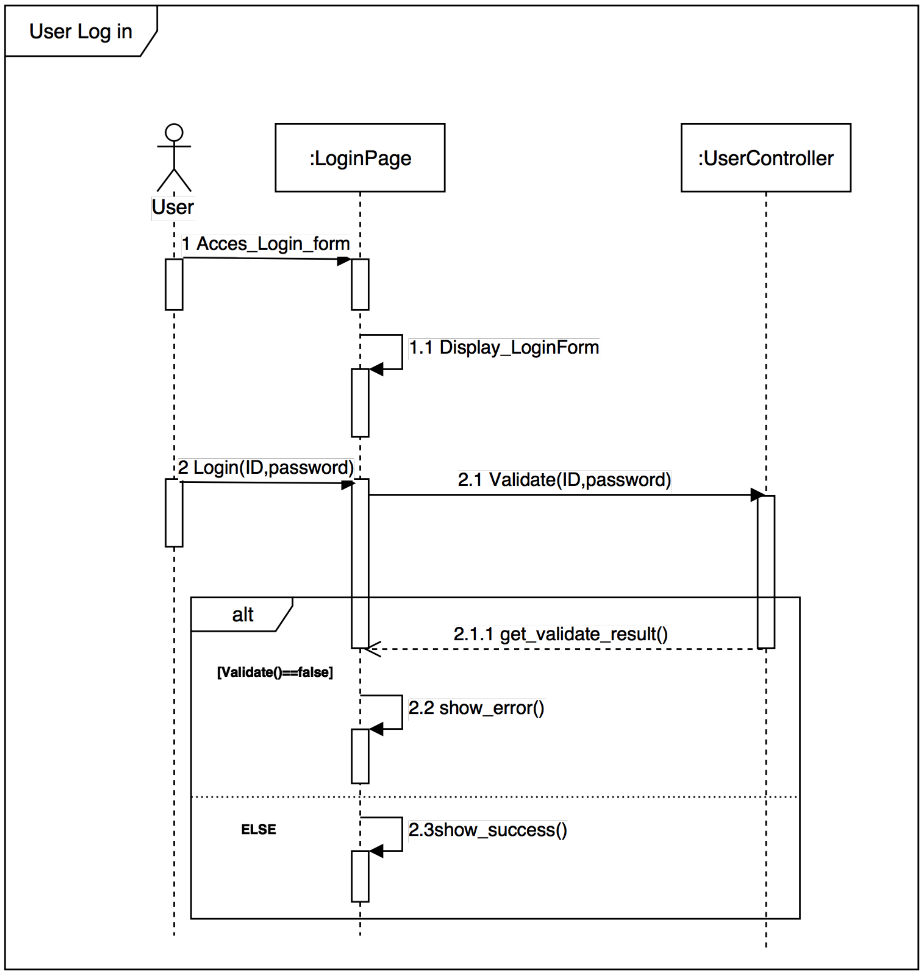


Figure 4: Login Sequence Diagram

In this sequence diagram it is shown that users have to input their login information to the App when they want to use the system. The login request is sent with these information to the system as parameter. First these information will be sent to the UserController which will check these in the database. If users' information(username) is found in the database and the password matches the username then the UserController returns login_success message to the Mobile application so that user can login into the system. Otherwise, the system shows error messages.

Reservation process

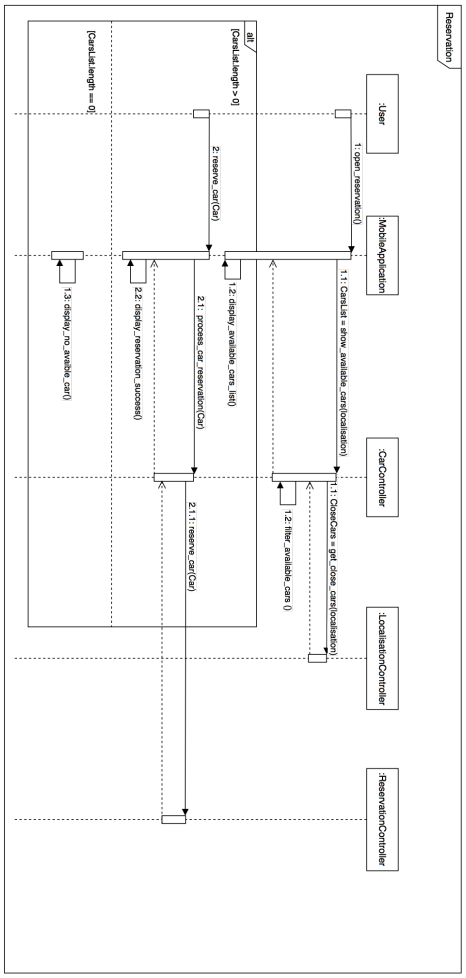

Reservation
:User
:MobileApplication
:CarController
:LocalisationController
:ReservationController
1: open_reservation()
1.1: CarsList = show_available_cars(localisation)
1.1: CloseCars = get_close_cars(localisation)
1.2: filter_available_cars ()
1.2: display_available_cars_list()
alt
[CarsList.length > 0]
2: reserve_car(Car)
2.1: process_car_reservation(Car)
2.1.1: reserve_car(Car)
2.2: display_reservation_success()
[CarsList.length == 0]
1.3: display_no_avaible_car()

Figure 5: Reservation Sequence Diagram

The user starts the process, in the mobile application, by searching for cars close to him or around an address. The mobile application sends a request to the Car Controller (through the router). The location is passed as a parameter in the call. The location comes from the GPS module of the smartphone or is manually entered by the user. The Car Controller calls the Localization Controller to get all the cars close to the location. Upon return, the list is filtered and then displayed to the user, if it is not empty. The user can choose a car to reserve, it is then added as a reservation and marked as unavailable. To conclude the operation, a success message is displayed to the user.

Billing process

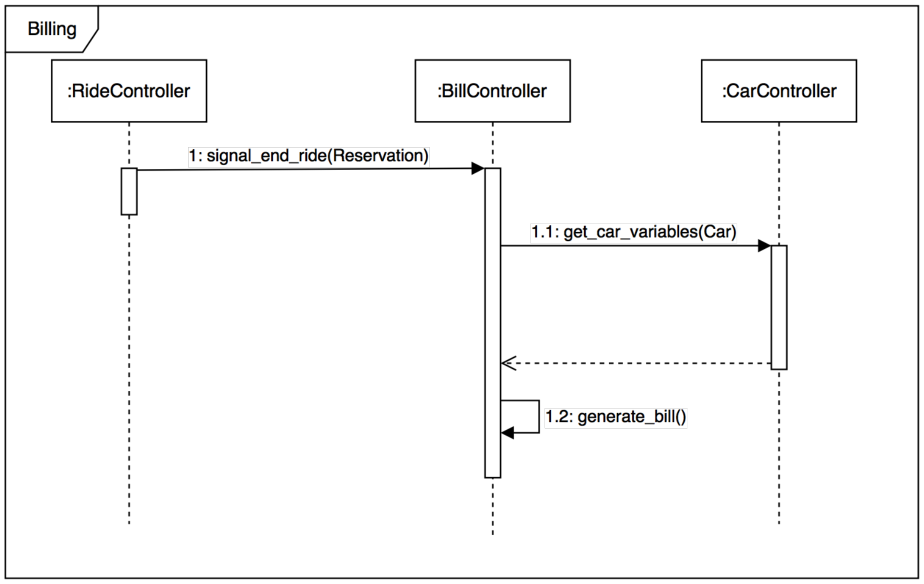


Figure 6: Billing Sequence Diagram

The billing process is started at the end of a ride. The ride controller signals the bill controller that a ride has ended. The reservation relative to that ride is passed as a parameter as it contains many variables that are used in calculating the bill.

Check-in car process

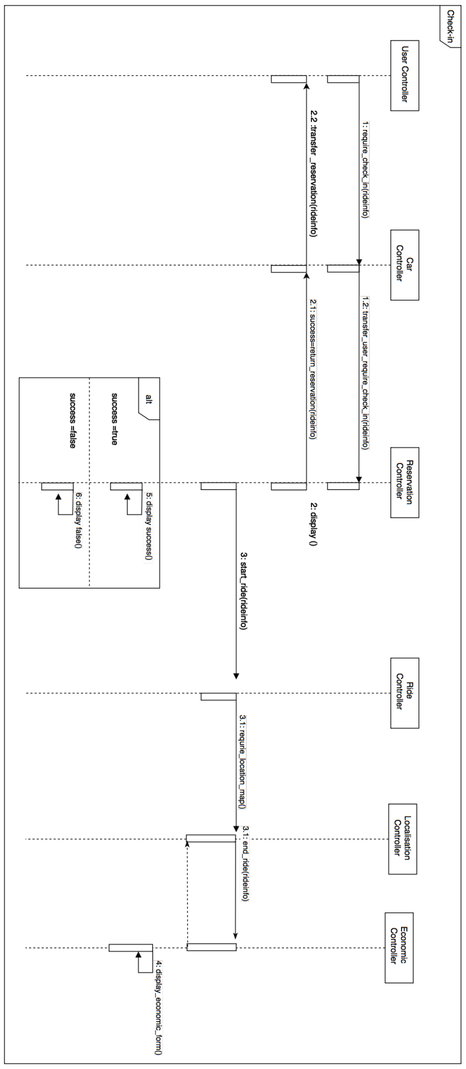


Figure 7: Check-in sequence diagram

After reservation, the user need to ask for a check-in process. The user controller send a required message to the car controller and the car controller transfer the ride information to the reservation controller. If this process success, the car controller will transfer the success information to the user and display the ride information on the device of user. Meanwhile, the ride controller received the ride information to start ride.

Check-out car process

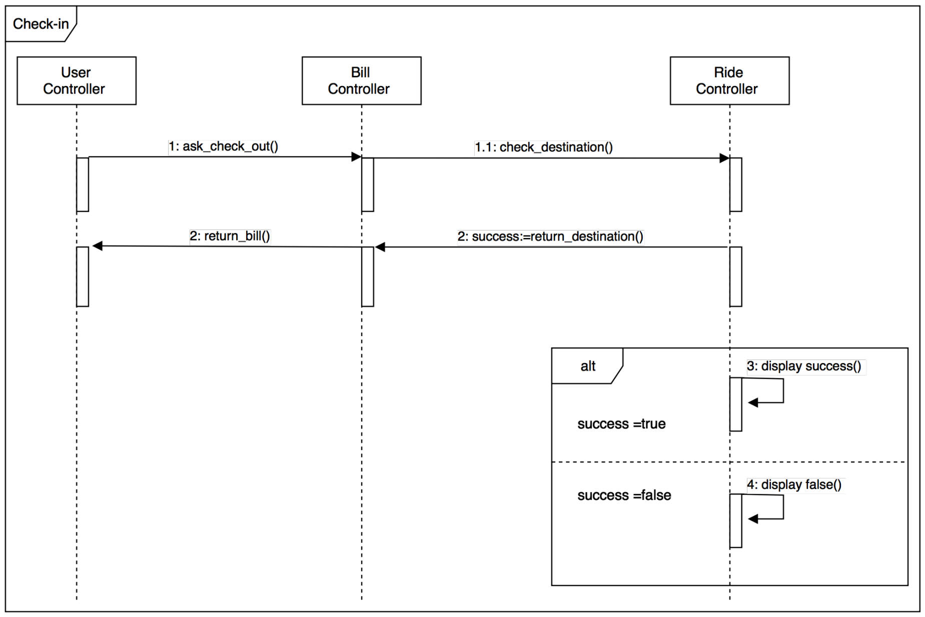


Figure 8: Check-out sequence diagram

When the ride process finished, the user ask for check out and send a check-out request to bill controller. The bill controller transfer the request to the ride controller to get the destination information and the bill controller calculate the bill of the certain ride and then return the bill information to the user.

Money saving process

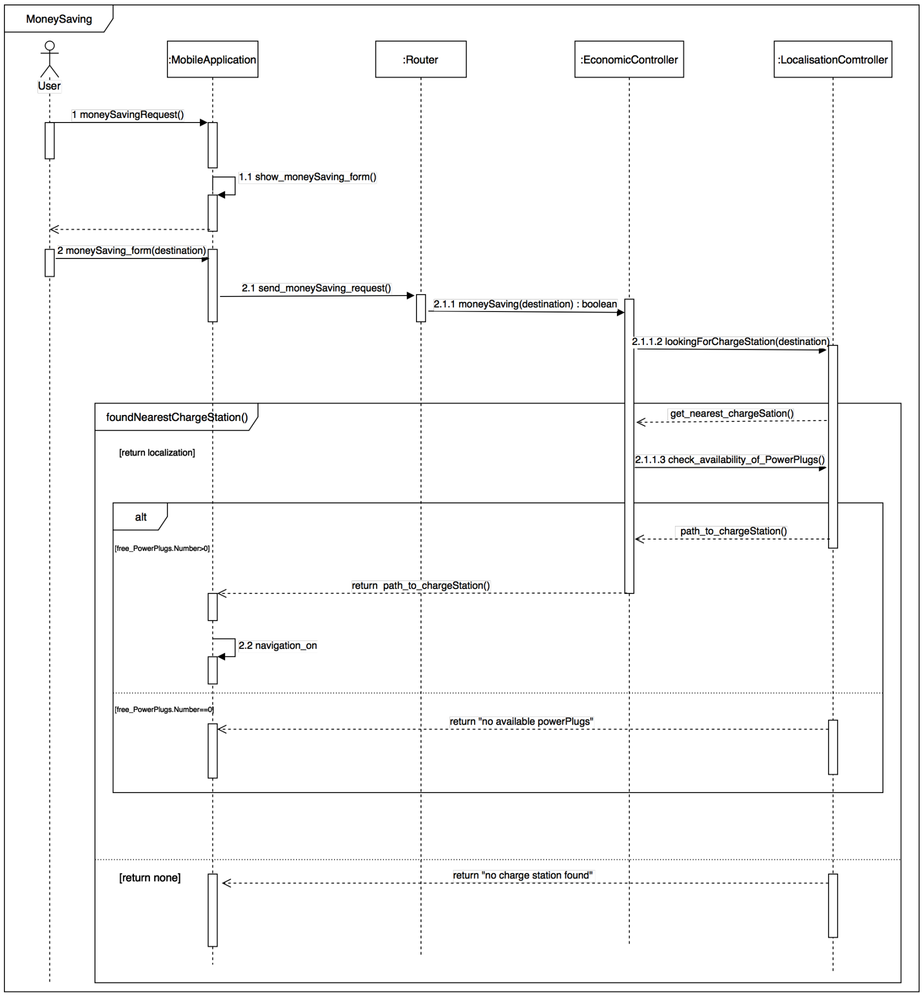

MoneySaving
User
:MobileApplication
:Router
:EconomicController
:LocalisationComtroller
1 moneySavingRequest()
1.1 show_moneySaving_form()
2 moneySaving_form(destination)
2.1 send_moneySaving_request()
2.1.1 moneySaving(destination) : boolean
2.1.1.2 lookingForChargeStation(destination)
foundNearestChargeStation()
get_nearest_chargeSation()
[return localization]
2.1.1.3 check_availability_of_PowerPlugs()
alt
path_to_chargeStation()
[free_PowerPlugs.Number>0]
return path_to_chargeStation()
2.2 navigation_on
[free_PowerPlugs.Number==0]
return "no available powerPlugs"
[return none]
return "no charge station found"

Figure 9: Money saving sequence diagram

In this sequence diagram it can be seen that if users ask for Money Saving option they will be asked to input a destination by the system. The request is then sent with the filled information as parameter to the system. And the system will determine whether the request meets the requirement.

2.6. Component interfaces

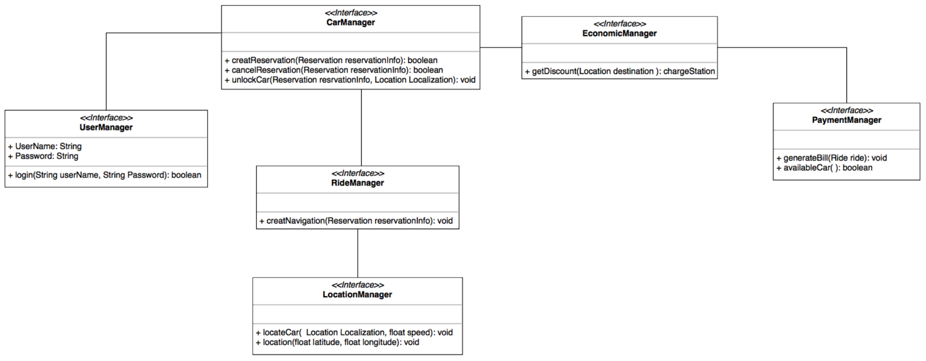

<<Interface>>
CarManager
+ creatReservation(Reservation reservationInfo): boolean
+ cancelReservation(Reservation reservationInfo): boolean
+ unlockCar(Reservation resrvationInfo, Location Localization): void
<<Interface>>
EconomicManager
+ getDiscount(Location destination): chargeStation
<<Interface>>
UserManager
+ UserName: String
+ Password: String
+ login(String userName, String Password): boolean
<<Interface>>
PaymentManager
+ generateBill(Ride ride): void
+ availableCar( ): boolean
<<Interface>>
RideManager
+ creatNavigation(Reservation reservationInfo): void
<<Interface>>
LocationManager
+ locateCar(Location Localization, float speed): void
+ location(float latitude, float longitude): void

Figure 10: Component interfaces diagram

2.7. Selected architectural styles and pattern

Three-tier for application architecture

As stated in the RASD Document, we will be using the **Three-tier client-server architecture**. The presentation tier is composed of the mobile application and the website. The application layer is composed of two parts. The REST Server that exposes the REST API and holds the business logic. It can be consumed by the web server or the mobile application. In addition, it is a security barrier between the client and the database as it prevents direct accesses to the database by the user. The other component of the application layer is the web server. The web server takes care of formatting the data in webpages and communicating with the web browser. The last tier is the data tier which is, in this case, composed of only one database that takes care of persisting the data of the whole application. The fact that the business logic is held at the level of our servers, the client-side of the application is kept as light as possible. Therefore, users can quickly access the application by installing it on their device or browsing the website. It also prevents the direct access to the database from the GUI which increases security.

MVC for web server

Model-**V**iew-**C**ontroller is a design pattern that is commonly used for GUIs. It relies on three objects:

- **Model**: is the logical structure of data used by the application.
- **View**: all the elements that the user can interact with including buttons, text fields …
- **Controller**: connects the model and the view.

The separation of concerns is the main motivation behind using MVC design pattern as it increases the possibility to reuse the components. It is important to note that in this case the controller in the web server does not perform any business logic on the data since the application server is the one to do it. The controller serves as a glue between the Model and the View.

REST over HTTP

REpresentational **S**tate **T**ransfer is the architectural style that we decided to use. REST is a lightweight way to make calls between the different machines of a client server application. REST is used in PowerEnjoy in almost all communications between the servers in our system. It relies on the HTTP Methods (GET, POST, PUT and DELETE).

The motivation behind using REST is that it improves portability; our system may be integrated with any client since it is supported by almost any platform.

By being stateless, it reduces the load on the application server since it does not have to keep track of the user sessions.

Concerning security, authentication is required to communicate with the REST API and the data is encrypted using SSL.

The data will be embedded in a JSON format in the body of the HTTP requests.

The following table shows the main REST endpoints that are needed in our application. The Method sections shows the HTTP method used to do the REST call (GET,POST,PUT,DELETE). The endpoint is the URI that should be queried. Arguments are the parameters sent in the body in a JSON format. The response contains the parameters received in the body of the response, formatted in JSON also.

Method	**Endpoint**	**Arguments**	**Response**
POST	/login	**username** Username of the user that wants to connect **password** Password entered by the user	**token** Token to use for further requests **message** Authentication status
POST	/cars/locate	**location** GPS coordinates of user or entered address	**carsList** List of cars available around
GET	/reserve/:id		**message** To signal success or failure
DELETE	/reserve/:id		**message** To signal success or failure
POST	/rides/end	**ride** ride to end	**message** To signal success or failure
POST	/moneysaving	**location** GPS coordinates of user or entered address	**chStation** Charging station to get discount
GET	/bills		**bills** Past bills of user
GET	/cars/:id/unlock		**message** To signal success or failure

The authentication is taken care by REST through the `GET /login` endpoint.

When this user is sent and the credentials are verified, the server sends back a token that should always be included for further requests. The token puts the server into context as it refers uniquely to a single user. Given the fact that REST API is stateless, this is how the server knows which “state” should be used.

3. Algorithm Design

3.1. Billing process

After the user finishes a ride in a PowerEnjoy car, the application has to calculate the amount that should be charged to the user. The amount is calculated is the multiplication of the time (in minutes) spent in the car and the price per minute. After calculating this amount, discounts/penalties may be applied:

- *10% discount* if the driver had other passengers with him.
- *20% discount* if he left the car with at least 50% battery.
- *30% discount* if he parked the car in a charging station and plugged it.
- *30% penalty* if he left the car 3km away from charging stations or with less than 20% battery.

If many discounts or penalties should be applied, they cumulate and are applied on the total ride amount. For example, a client had more than two passengers with him and left the car charging in a station he will benefit of 40% discount (10% + 30%).

```
int PRICE_PER_MIN = config.getPricePerMin();

public Bill caculate_Bill(Reservation r)
{
  Ride ride = r.ride;
  Car  car  = r.car;

  float BaseFee = ride.duration * PRICE_PER_MIN;
  float discount = 0;
  // More than 2 passengers discount
  if (car.passengers >= 2) discount += 0.1 ; // 10% Discount
  // More than  50% battery left
  if (car.battery >= 0.5 ) discount += 0.2; // 20% Discount
  // User recharged car
  if (car.isCharging()) discount += 0.3; // 30% Discount
  // If car is left 3KM away from charging OR 20% battery
  boolean isFar = LocalisationController.isCarFar();

  if (LocalisationController.isCarFar(car) || car.battery <= 0.2)
              discount -= 0.3;

   float ChargedAmount = BaseFee * (1 - discount );

   return new Bill(new Date (), ChargedAmount);
}
```

3.2. Reservation process

When the user make a reservation, we should check the status of user ,the car status and the reservation time. The system allows user to make a reservation 1 hour ahead of the reservation time. In addition, the application allows the user to cancel reservation and it can also monitor the ride information.

```
//Define the reservation time
public ReservationTimeValid(time)
{
  if (time.Nowtime)
  return true;
  else {
          if(time.Nowtime<=1)
          return new time;
          else
          return false;
        }
}
public Reservation (Car c, User u)
{

  if (c.status=="available"&& u.status=="available"&&ReservationTimeValid(time))
  {
    isThisReservation= true;
    User.startRide();
  }
  else if (time.timeNow<=1)// the user made a reservation one hour ahead of the reservation
  return Reservation(Reservation time(), User ID());

}
//Create a new reservation
public createReservation()
{
   Reservation r = new reservation;
   User.Reservation();
}
//Allow user to cancel reservation
public cancelReservation()
{
   if (isThisReservation)
   {
     isThisReservation = false;
     User.cancelReservation();
     User.bill-=1;// if user cancelled the order, system will asked for 1 euro as compensat
   }
}
//Monitor the location and cars nearby
public updateLocation(double[] location)
{
  User.updateLocation(location);
}
```

3.3.Money saving process

When the user use the car they are allowed to request to get the money-saving option. The system will check that whether there is a charging station and the availability of the station nearby the destination, which the user inputted. If the request fills all conditions then the GPS will navigate the user to the charging station where they can get money saving. And the system will keep the charging plugs for the user in a period of time.

```
import Location.Map;
import ChargeStation.dao;
public class moneySaving {

private boolean mNearByChargeStation;
private int mPlugsAvailability;
private String mChargeStationName;
private Location Path;
public moneySavingOptions(boolean NearByChargeStation, int PlugsAvailability ){
    if(NearByChargeStation == true){
      mChargeStationName = ChargeStationName;
      if(PlugsAvailability >= 2){
        //keep the plugs for the user.
        mPlugsAvailability = PlugsAvailability-1;
        // navigate the user to the charging station.
        Path = Map.location.Naviagte(ChargeStationName);
      } else Toast.makeText(this, "No available plugs for charging").show();
// show error
    } else Toast.makeText(this,"No Charge Station nearby the destination").show();//show er

}
//determine whether there is a charging station nearby the destination.
public boolean isNearByChargeStation(Location d ){
    if Map.Location(d) != none {
      return mNearByChargeStation;
      }
   }
// return the name of the charging station.
public void getChargeStationName(boolean NearByChargeStation){
     mChargeStationName = ChargeStationName;
}
//  return the availability of charging plugs in the charge station.
public int getPlugsAvailability(String ChargeStationName){
      return mPlugsAvailability;
  }
}
```

4.User interface design

4.1. Mock-ups

The mock-ups were presented in the RASD Document in section 2.1.

4.2. UX flow chart

We introduce the UX (User experience) flow chart to show the workflow of functions.

Login process

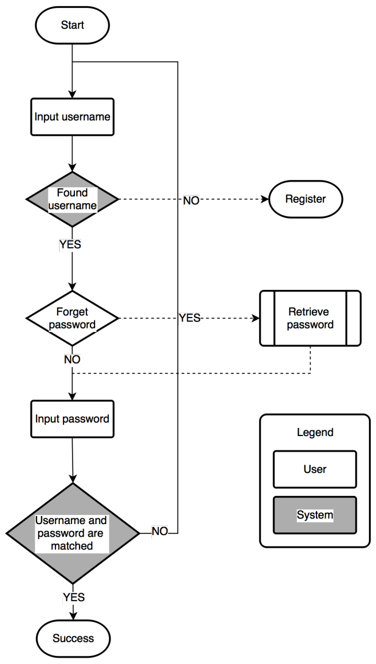


Figure 11: Login UX

Reservation process

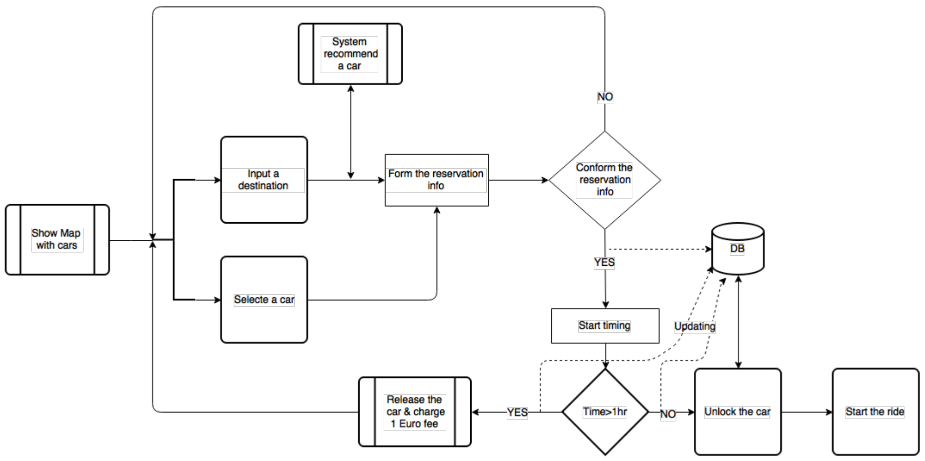


Figure 12: Reservation UX

5.Requirements Traceability

The design document is aiming to explain the goals in the RASD.

G[1] Allows users to register in the PowerEnjoy application

- The User controller

G[2] Allows registered users to login using their credentials

- The User controller

G[3] Allows users to modify their information.

- The User controller

G[4] Allows users to see the cars around him or around an address on the application.

- The user controller
- The localization controller

G[5] Allows users to reserve cars up to one hour in advance.

- The reservation controller

G[6] Allows users to cancel a reservation.

- The reservation controller
- The user controller

G[7] Allows users to unlock and check-in the reserved car.

- The user controller
- The ride controller
- The reservation controller

G[8] Allows users to see how much the previous ride cost along with more ride information.

- The bill controller
- The user controller

G[9] Allows users to check their rides history.

- The user controller

G[10] Allows users to the user should be able to enable economy mode.

- The reservation controller
- The economic controller

G[11] Allows users to see where to park the car in order to get discount.

- The localization controller
- The user controller

G[12] Allows systems to keep real-time data about the car variables.

- The car controller

G[13] Reservations should time-out if the user doesn't check-in the car.

- The car controller
- The user controller
- The reservation controller

G[14] System should calculate the price of the ride depending on the time, left charge in the battery and number of passengers.

- The ride controller
- The bill controller
- The car controller

G[15] Allows the operator to validate the identity and driving license of the user after checking them personally.

- The car controller

G[16] Allows the operator to verify the damaged and faulty cars.

- The car controller

G[17] Allows the operator monitor the position of the cars.

- The car controller
- The localization controller

6.References

6.1. Bibliography

- Sample Design Deliverable Discussed on Nov. 2
- Assignments AA 2016-2017
- IEEE standard on requirement engineering

6.2. Used tools

- Draw.io: for UML,UX,BCE diagrams.
- Atom: for writing document.
- Pandoc: for PDF creating.
- Github: for version controller.

7.Hours Worked

Reda Aissaoui

- 23/11/2016 4h
- 24/11/2016 4h
- 01/12/2016 4h
- 05/12/2016 1h
- 08/12/2016 6h
- 11/12/2016 2h

Xing Jinling

- 23/11/2016 3h
- 24/11/2016 3h
- 01/12/2016 3h
- 07/12/2016 3h
- 08/12/2016 4h
- 11/12/2016 1h

Zhang Lidong

- 20/11/2016 0.5h
- 23/11/2016 3h
- 24/11/2016 4h
- 27/11/2016 2h
- 01/12/2016 3h
- 08/12/2016 3h
- 10/12/2016 4h
- 11/12/2016 2h