Politecnico di Milano

A.A. 2016-2017

Software Engineering 2: “PowerEnjoy”

Design Document

Version 1.4

Prasanth Ravulapalli

Fathima B

Lipika L

December 16th 2016

Contents

Introduction

Purpose

The main purpose of this document is to give more technical details than the RASD about PowerEnjoy application.

This document is addressed to developers and aims to identify:

- The high-level architecture
- The design patterns
- The main components and their interfaces provided one for another
- The Runtime behavior

Scope

The project PowerEnjoy, which is a service based on mobile and web based digital management system for car sharing which exclusively employs electric cars. It targets two set of people:

- Guest users
- Registered users

The system allows the user to register into the system by providing their credentials and payment information using the provided web or mobile app. It also helps the users in finding the locations of the car within a certain distance from the user's specific location.

The app also provides the freedom of cancelling the registration if the users doesn't pick up the car in specified time.

Users may not always be registered in the app; they also can see the available cars just by entering the app. This kind of users are considered as guest users.

The system includes various kinds of services and benefits accordingly.

The main purpose of the system is to be more efficient and reliable than the current apps in the same domain which tries to offer better services to the client.

Definitions, acronyms, abbreviations

- Guest user: he is one of the clients of the application. Each should attach the following information whenever he tries to access the application data.
    - Name
    - Phone number
    - GPS location

- Registered User: he is another client of the application who has access to most of the part of the app. He can reserve/ cancel the cars. He will be charged accordingly for the drive he had made with the reserved car. He should provide the following details to the application for better service.
    - GPS location
    - Payment details
    - Sharing mode
    - Time of reservation
    - Name
    - Phone number
    - Email ID
- GPS: Global Positioning System, it provides the exact location of the respective object.
- Electric Cars: those are the cars which work using the electric energy rather than the normal fuels. These are eco-friendly devices.
- Locking of the car: When the car is placed in the parking zone and if there are no people inside the car, the car moves to the locked mode.
- Unlocking of the car: When the user is close to the car, car receives the signal and unlocks the car so that user can use it for his drive.
- Car sharing: It is possible for the user to share his drive with other users who may join on the way of the main user's path.
- Ride: A tour when the car gets unlocked and user starts the drive till the cars gets locked when users leaves the car is called a ride.
- Discount: A portion of amount that has been deducted from the user's ride as a token of appreciation.
- Penalty: A portion of amount need to be extra paid by the user in case of violation of rules.
- User request: it is the request from the user to reserve a car close to his location for his drive.
- Reserving a car: it means the car has been blocked and no one else other than the respective user who reserved the car can use it. If the user fails to arrive the car with in an hour, the reservation will be cancelled and the car will be available open for reservation.
- Request queue: all the requests by the user should be processed in a first in first out approach without maintaining any priority. First request should be provided with the resource only then the next.
- Matching ride: If two persons (X and Y) are matching the ride if they meet the following conditions:
    - Starting point and ending point of Y are close to the starting point and ending point of X. And starting point of Y is in between start and end points of X.
    - There should be enough vacancy in the car.
    - Location of X should not overtake location of Y before matching.
- Wait down charge: This is the charge applied on the user whenever the user keeps the car on unlock mode and still the user gets charged for owning the car for that amount of time of unlock.

Reference Documents

RASD document for the same project.

Sample Design document deliverable discussed.pdf

Design documents slides provided by the teacher.

https://www.tutorialspoint.com/struts_2/basic_mvc_architecture.htm To know about MVC architecture.

Document Structure

Introduction

This section introduces the design document. It contains a justification of his utility and indications on which parts are covered in this document that are not covered by RASD.

Architecture Design

This section is divided into different parts:

- Overview: This sections explains the division in tiers of our application.
- High level components and their interaction: This sections gives a global view of the components of the application and how they communicate.
- Component view: This sections gives a more detailed view of the components of the applications.
- Deploying view: This section shows the components that must be deployed to have the application running correctly.
- Runtime view: Sequence diagrams are represented in this section to show the course of the different tasks of our application.
- Component interfaces: The interfaces between the components are presented in this section.
- Other design decisions

Algorithms Design

This section describes the most critical parts via some algorithms. Pseudo code is used to hide unnecessary implementation details to focus on the most important parts.

User Interface Design

This section presents mockups and user experience explained via UX and BCE diagrams.

Requirements Traceability

This section aims to explain how the decisions taken in the RASD are linked to design elements.

Architectural design

Overview

The PowerEnjoy has a MVC architecture.

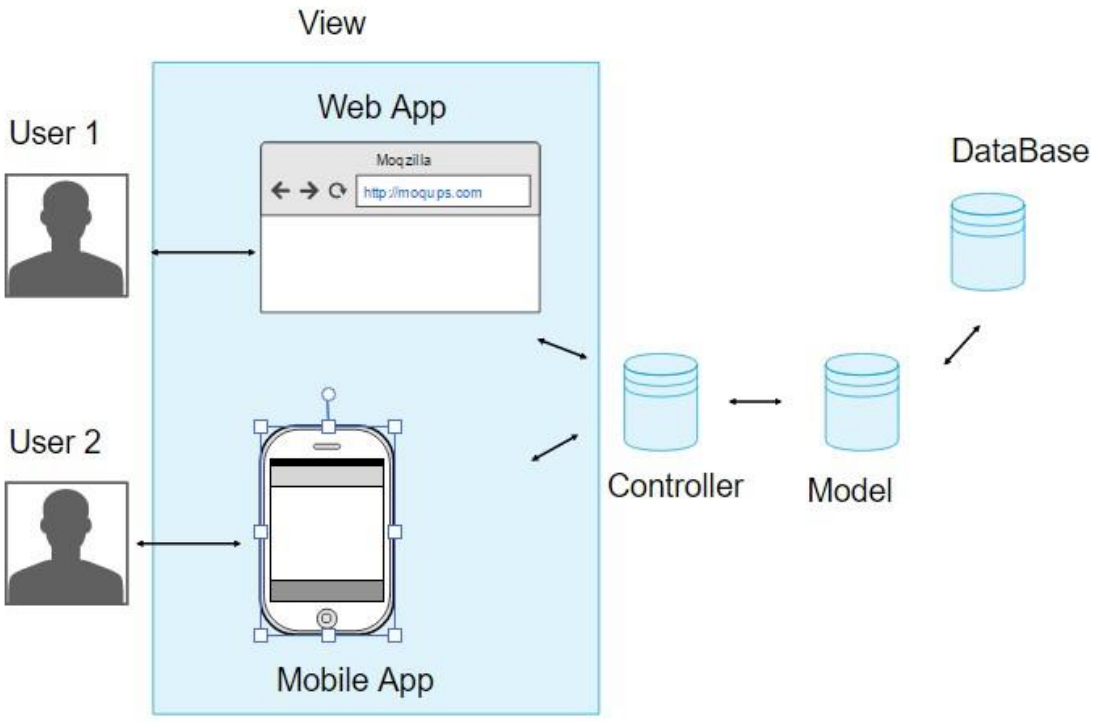

Figure 1: Model MVC architecture view for PowerEnjoy

Model View Controller or MVC as it is popularly called, is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts:

- Model - The lowest level of the pattern which is responsible for maintaining data.
- View - This is responsible for displaying all or a portion of the data to the user.
- Controller - Software Code that controls the interactions between the Model and View.

MVC is popular as it isolates the application logic from the user interface layer and supports separation of concerns. Here the Controller receives all requests for the application and then

works with the Model to prepare any data needed by the View. The View then uses the data prepared by the Controller to generate a final presentable response. The MVC abstraction can be graphically represented as follows.

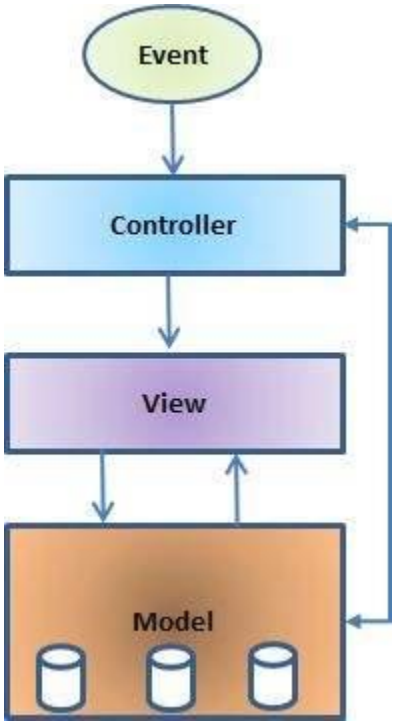

Figure 2: MVC framework

The model

The model is responsible for managing the data of the application. It responds to the request from the view and it also responds to instructions from the controller to update itself.

The view

A presentation of data in a format, triggered by a controller's decision to present the data. They are script based templating systems like JSP, ASP, PHP and very easy to integrate with AJAX technology.

The controller

The controller is responsible for responding to user input and perform interactions on the data model objects. The controller receives the input, it validates the input and then performs the business operation that modifies the state of the data model.

High level components and their interaction

The high-level components architecture is composed of five different components.

- Actions
- Interceptors
- Value Stack
- Results
- View Technologies

The above diagram depicts the Model, View, and Controller to the PowerEnjoy high level architecture. The controller is implemented with a dispatch servlet filter as well as interceptors, the model is implemented with actions, and the view as a combination of result types and results. The value stack provides common thread, linking, and enabling integration between the other components.

Apart from the above components, there will be a lot of information that relates to configuration. Configuration for the web application, as well as configuration for actions, interceptors, results, etc.

Based on the above diagram, the user's request life cycle is as follows:

- User sends a request to the server for requesting for some resource (i.e web pages, mobile application).
- The FilterDispatcher looks at the request and then determines the appropriate Action.
- Configured interceptors' functionalities applies such as validation, file upload etc.
- Selected action is executed to perform the requested operation.
- Again, configured interceptors are applied to do any post-processing if required.
- Finally, the result is prepared by the view and returns the result to the user.

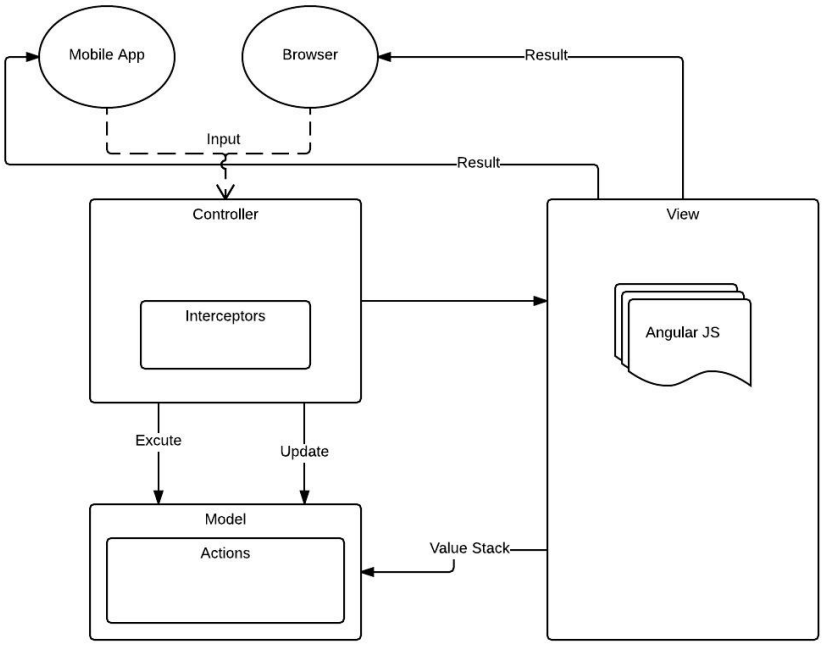

Figure 3: High level components

Component view

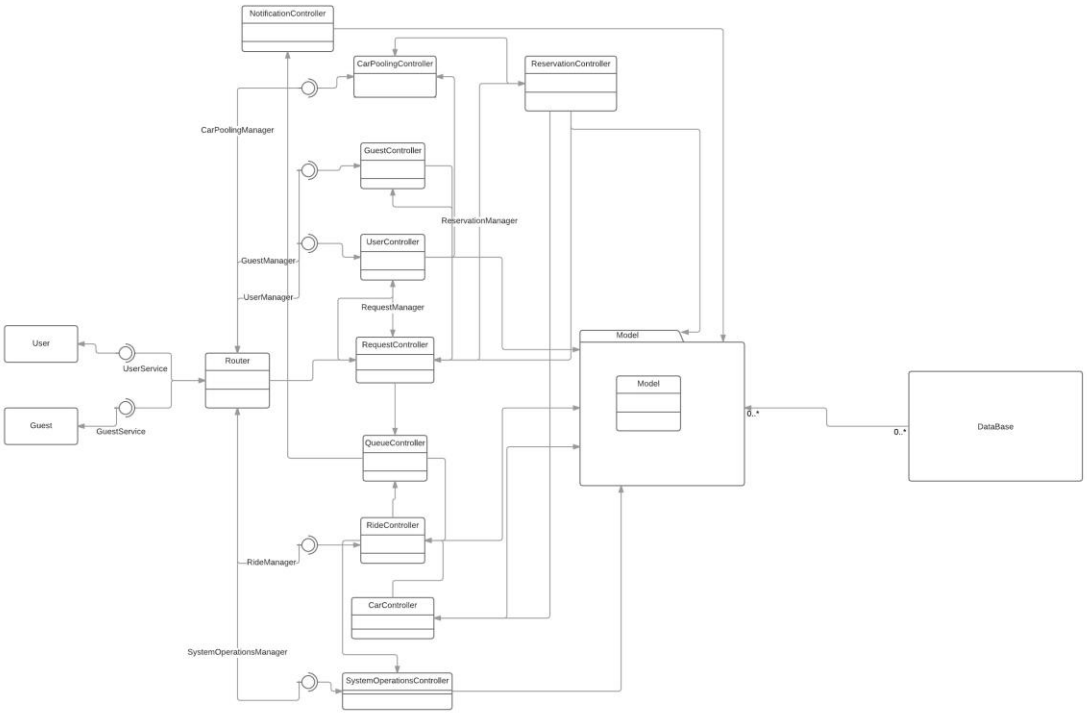

Figure 4: Component View

- NotificationController: manages all notifications related operations like sending confirmation mails or SMS
- CarController: manages cars
- UserController: manages user related operations
- GuestController: manages guest related operations
- RequestController: manages all the requests
- QueueController: manages process queue of requests
- RideController: manages the rides
- CarPoolingController: manages pooling and assigning of the cars
- SystemOperationsController: manages system related operations like unlocking the car, calculating the cost of the ride, some internal requests
- Router: route requests to the respective controller
- Model: communicates between the respective requests and the data
- User: the user's device
- Guest: the guest's device
- DB: the database used to store persistent data

Deploying view

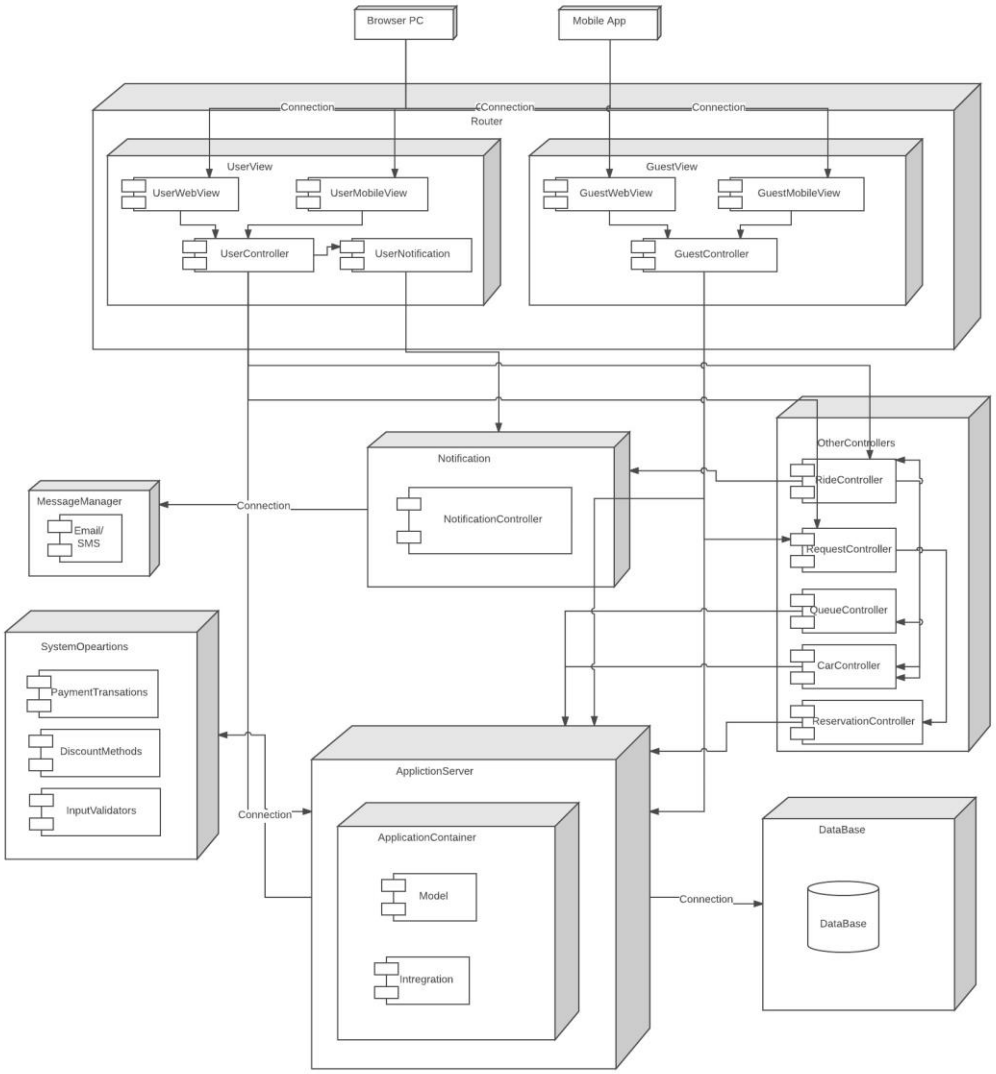

Figure 5: Deployment View

Runtime view

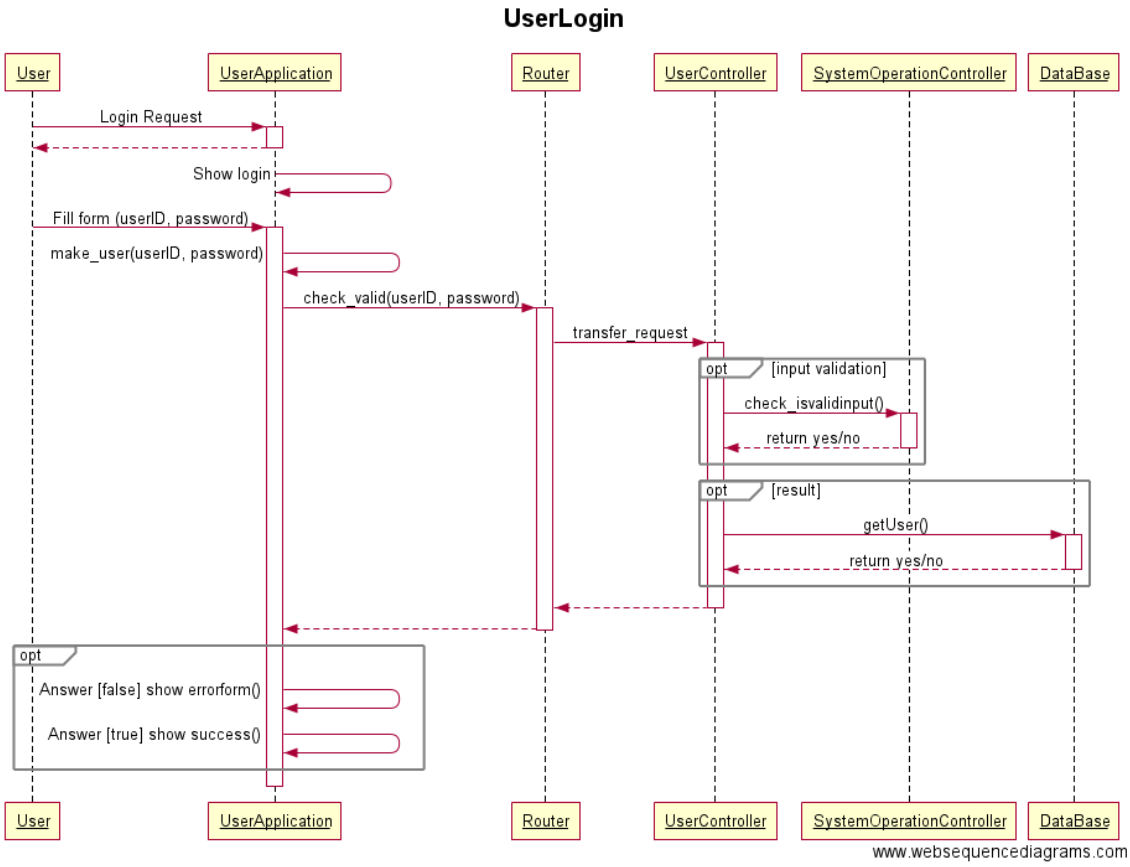

Figure 6: UserLogin

title UserLogin

User->+UserApplication: Login Request

UserApplication-->-User:

UserApplication->UserApplication: Show login

User->+UserApplication: Fill form (userID, password)

UserApplication->UserApplication: make_user(userID, password)

UserApplication->+Router: check_valid(userID, password)

```
Router->+UserController: transfer_request

opt input validation
   UserController->+SystemOperationController: check_isvalidinput()
   SystemOperationController-->-UserController: return yes/no
end

opt result
   UserController->+DataBase: getUser()
   DataBase-->-UserController: return yes/no
end

UserController-->-Router:

Router-->-UserApplication:

opt
   UserApplication->UserApplication: Answer [false] show errorform()
   UserApplication->UserApplication: Answer [true] show success()
end
```

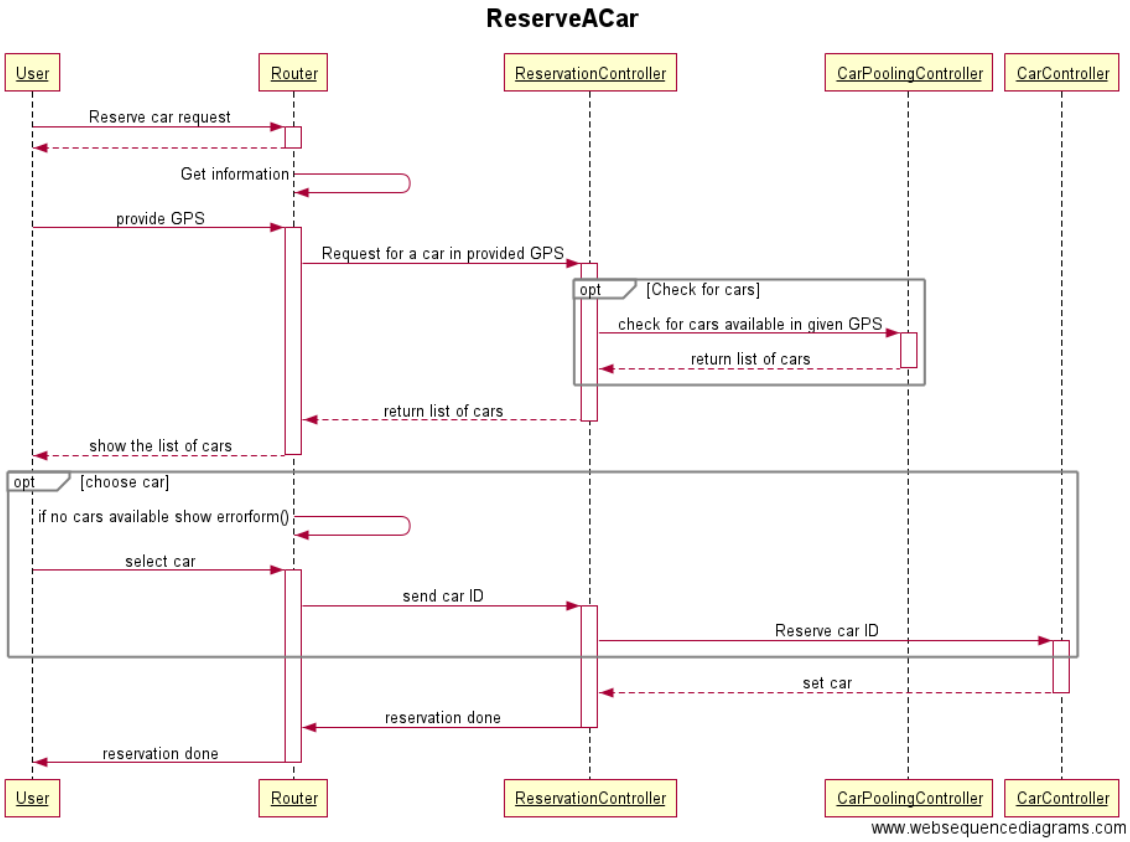

Figure 7: Reserve a car

title ReserveACar

User->+Router: Reserve car request

Router-->-User:

Router->Router: Get information

User->+Router: provide GPS

Router->+ReservationController: Request for a car in provided GPS

opt Check for cars

ReservationController->+CarPoolingController: check for cars available in given GPS

```
    CarPoolingController-->-ReservationController: return list of cars
end

ReservationController-->-Router: return list of cars

Router-->-User: show the list of cars

opt choose car
    Router->Router: if no cars available show errorform()
    User->+Router: select car
    Router->+ReservationController: send car ID
    ReservationController->+CarController: Reserve car ID
end

CarController-->-ReservationController: set car
ReservationController->-Router: reservation done
Router->-User: reservation done
```

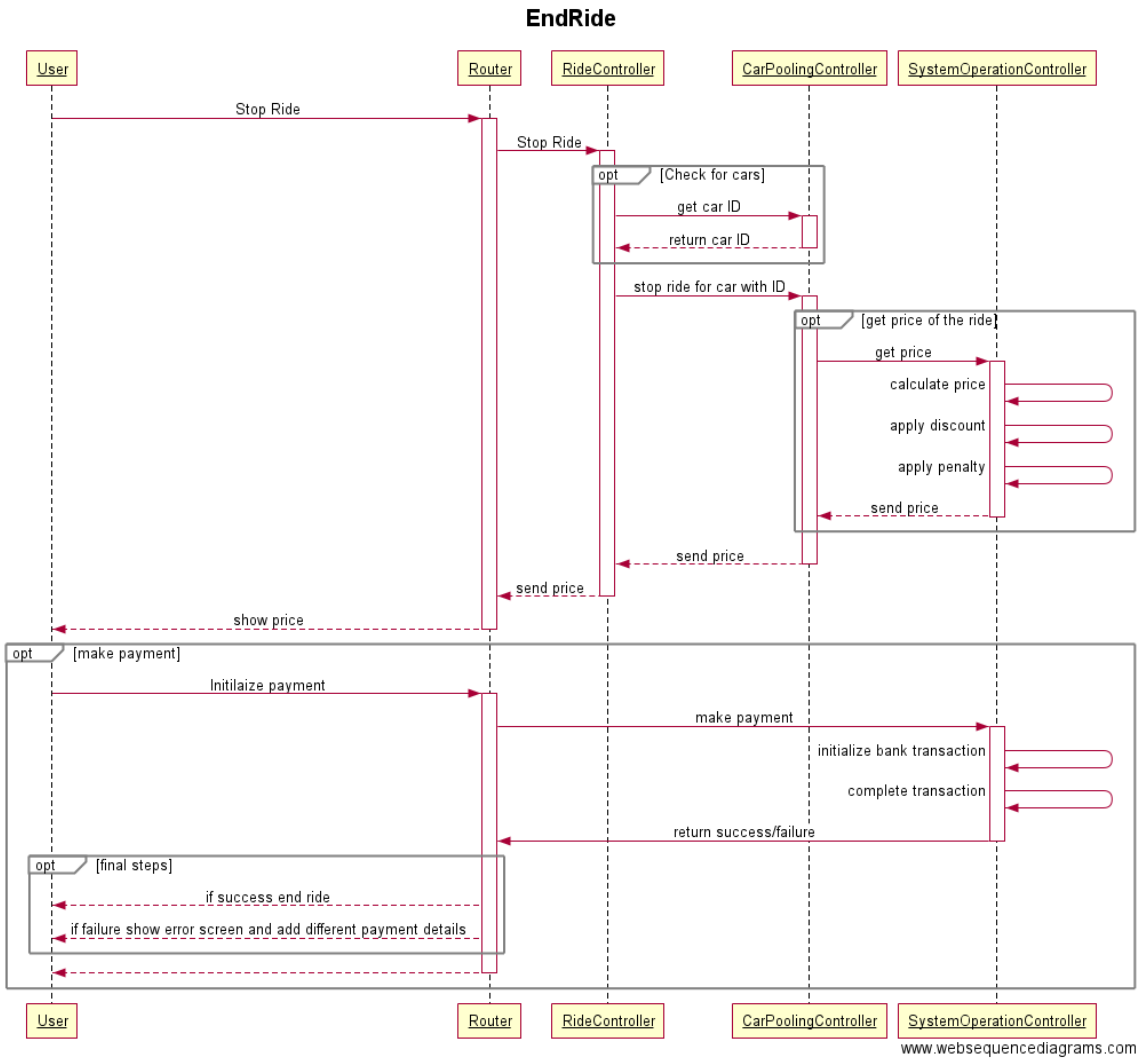

Figure 8: End ride

title EndRide

User->+Router: Stop Ride

Router->+RideController: Stop Ride

opt Check for cars

RideController->+CarPoolingController: get car ID

CarPoolingController-->-RideController: return car ID

end

```
RideController->+CarPoolingController: stop ride for car with ID

opt get price of the ride
   CarPoolingController->+SystemOperationController:get price

   SystemOperationController->SystemOperationController: calculate price
   SystemOperationController->SystemOperationController: apply discount
   SystemOperationController->SystemOperationController: apply penalty

   SystemOperationController-->-CarPoolingController: send price
end

CarPoolingController-->-RideController: send price

RideController-->-Router: send price

Router-->-User: show price

opt make payment
   User->+Router: Initilaize payment

   Router->+SystemOperationController: make payment

   SystemOperationController->SystemOperationController:initialize bank transaction
   SystemOperationController->SystemOperationController:complete transaction

   SystemOperationController->-Router: return success/failure
```

```
    opt final steps
        Router-->User: if success end ride
        Router-->User: if failure show error screen and add different payment details
    end
    Router-->-User:
End
```

Component interfaces

We will present in this part of the document the interfaces relevant for each component identified in the *Component view*. To do so, we describe the required and provided interfaces, and the corresponding operations that can be executed on them. Note that the *Deployment view* induces remote and local interfaces on some of the components, but we assume that both will provide the same operations.

- **WebBrowser**: It allows the displaying and the use of the PowerEnjoy website, thus it encapsulates some display functions and some income functions as well such as login, create account, create request. It needs to communicate with the UserWebView/ GuestWebView that will provide it all the displaying command. The WebBrowser interface is on the one hand responsible of the transmission of the displaying command from the UserWebView/ GuestWebView to the WebBroswer and on the second hand is responsible of the way back transmission of the input events received by the web browser.
- **UserWebView**: It is an intermediary between the web browser and the controller. The UserController interface is in charge to give displaying command from the controller to the UserWebView and to transmit input events in the opposite direction, thus encapsulates some display and income functions.
- **UserController:** It provides displaying orders to the UserWebView that are based on the information received from the Model. It also receives notifications about input event from UserView then sends the appropriate requests to the Model. It owns some result interpretation functions that will create a displaying command from the result of a Model method and an input event interpretation that will from an input event call a model method.
- **UserMobileView:** It has the same behavior than the UserWebView unless that it doesn't command a WebBrowser but owns some display and income functions to command the hardware directly inner the mobile. The interface between it and the UserController has the same purpose than the one between the WebView and the WebController.
- **GuestMobileView:** It has the same behavior than the UserMobileView unless that it has limited features communicates with the GuestController.
- **GuestWebView:** It has the same behavior than the UserWebView unless that it has limited features communicates with the GuestController.
- **Model:** Its methods are called by the controllers and it sends back to them the results. It includes all the methods described in the document such as editing an account, answering a request. It has an interface with the DataBase to send some write and read requests and receive the results.
- **DataBase:** It communicates only with the Model to receive from it some read or write requests and send back some results. It includes all the data access methods such as

creating an account, editing an account, logging in, getting a passenger instance and so on.

Other design decisions

Those design decisions which are not based on the selection of an architectural pattern are presented and justified here.

As previously stated, this document represents a logical architecture. By doing so, we pretended to focus on the behavior of the system independently of specific communication and implementation issues. This adds portability to the architecture, so that it might be implemented on a broad variety of platforms.

The reservations requests are stored in the DB in-order to free resources and space in the system. Otherwise, the information of such requests should be kept in main memory, for example, and this would unnecessarily consume resources.

Algorithm design

Pseudo code of the payment calculation method

The main steps of the algorithm are:

- Declaration of the constant variables
- Calculation of trips on every part of the ride
- Calculation of the final price based on all discount possibilities available and considering penalties made
- Adding the final price to the user's account

```
//Declaration of all the price constants

pricePerMinute, pricePerKilometer

discountOnTowOrMoreSharedPassengers = 10

discountOnBatteryLeftWithMoreThan50% = 20

discountOnCarParkedInSpecialParkingZone = 30

penaltyOnCarLeftFar= 30


void  calculatePrice(CurrentRide) :

        requestlist = []

        origin= CurrentRide.requests[0].origin //common origin for all the requests

        //Fill the requestlist with tuples (request, distance origin-destination, time taken to finish the ride)

        for request in CurrentRide.requests :

                destination= request.destination

                requestlist.add((request, origin-destination, end time – start time)

                // origin-destination  : calculate the distance between origin and destination => |origin.latitude-destination.latitude|+|origin.longitude-destination.longitude|

        requestlist.sort(1) // sort the requestlist increasingly according to the distance origin-destination

        //Create a list of trips between the destination of the previous request and the destination
```

of the current request

```
trips = []
for e in requestlist :
        request=e[0] // get the first component of e which is the request
        destination= request.destination
        trips.add(MapServer.getItinerary(origin,destination,hour))
        origin=destination
```

// Calculate the price on every request and store it in a list. The calculation is based on the time and distance provided by the Map server threw the trips and the time taken to finish each sub trip. The price is divided per the number of passengers who are sharing the ride for a given trip.

```
price = []
n = trips.length()
f=0
m=n
for trip in trips:
        f=f + (pricePerKilometer*trip.distance/m + pricePerMinute*trip.timetaken/m)
        price.add(pricePerKilometer*trip.distance/m + pricePerMinute*trip.timetaken/m + f )

m=m-1
for i in range(0, n):
        requestlist[i][0].acceptedRequest.price=price[i]

fp = p = price[0]
if(checkDiscountOnTowOrMoreSharedPassengers())
        fp = fp – (p*discountOnTowOrMoreSharedPassengers/100)
if(checkDiscountOnBatteryLeftWithMoreThan50())
        fp = fp – (p*discountOnBatteryLeftWithMoreThan50 /100)
```

```
        if(checkDiscountOnCarParkedInSpecialParkingZone())
                fp = fp – (p*discountOnCarParkedInSpecialParkingZone/100)
        fp = max(fp,0)
        if(checkPenaltyOnCarLeftFar())
                fp = fp + (p*penaltyOnCarLeftFar/100)
requestlist[0][0].acceptedRequest.price = fp
```

User interface design

Mockups

Mockups are included in Requirements and Specification Document in section 3.2.1 and 3.2.2.

Requirements traceability

The design of this project was made aiming to fulfill optimally the requirements and goals specified in the RASD. The reader can find here under the list of these requirements and goals and the designed component of the application which will assure its fulfillment.

- [G1] Allowing the guest users to enter the app.
    - The GuestUserController
    - The Router
    - The Database
- [G2] Allowing the users to find all nearby available cars and their locations.
    - The UserController/ The GuestUserController
    - The Router
    - The QueueManager
    - The RequestManager
- [G3] Allowing the users to register into the application by providing their credentials and payment information.
    - The UserController
    - The Router
    - The Database
- [G4] Allowing registered users to book car in a selected geographical area.
    - The UserController
    - The Router
    - The QueueManager
    - The RequestManager
    - The CarPoolManager
    - The CarController
- [G5] Allowing registered users to reserve only a single car and make it available for up to one hour before they pick it up.
    - The UserController
    - The Router
    - The QueueManager
    - The RequestManager
    - The RideController
    - The CarPoolManager
    - The CarController

- [G6] Allowing admin users to keep track of the cars at current locations.
    - The AdminController
    - The Router
    - The CarPoolManager
    - The RideController
    - The CarController
- [G7] Allowing admin users to send notifications or alerts to some cars accordingly.
    - The AdminController
    - The Router
    - The NotificationManager
    - The CarController
- [G8] Allowing registered users to choose option in sharing other passengers.
    - The UserController
    - The Router
    - The QueueManager
    - The RequestManager
    - The CarPoolManager
- [G9] Allowing registered users to choose on money saving option by letting them enter their destination.
    - The UserController
    - The Router
    - The SystemOperationController
    - The Database
- [G10] Allowing registered users to check the percentage of battery left in the car.
    - The UserController
    - The Router
    - The CarController
- [G11] Allowing registered users to check the price charged till that moment of time.
    - The UserController
    - The Router
    - The CarController
    - The SystemOperationController
- [G12] Allowing registered users to find the safe areas for parking the car after their use.
    - The UserController
    - The Router
    - The SystemOperationController
    - The Database
- [G13] Allowing registered users to send the information of their current location for the car to get unlocked and ready for the drive.
    - The UserController
    - The Router
    - The CarController
    - The RequestManager
- [G14] Allowing registered users to receive a confirmation SMS or an email stating that the car has been booked, location of the car and threshold time to collect the car.

    - The UserController
    - The Router
    - The CarController
    - The CarPoolManager
    - The NotificationManager
- [G15] Allowing registered users to know the estimated price of the ride and optimal statistics.
    - The UserController
    - The Router
    - The CarController
    - The SystemOperationController
- [G16] Allowing registered users to show the nearby charging points.
    - The UserController
    - The Router
    - The SystemOperationController
    - The Database
- [G17] Allowing registered users to unlock the car whenever needed.
    - The UserController
    - The Router
    - The CarController
    - The CarPoolManager
- [G18] Allowing registered users to stop the ride whenever needed.
    - The UserController
    - The Router
    - The CarController
    - The CarPoolManager
    - The QueueManager
    - The RequestManager
    - The SystemOperationController

References

Used Tools

The tools used to create this document are:

- Star UML: for UML models
- Sublime Text: to write code and maintain indentations
- GitHub: Version controlling tool
- Moqups: To generate online mock up designs
- Word: For local edits of the document and create pdf
- https://www.websequencediagrams.com, https://www.lucidchart.com: To generate UML diagrams

Hours of Work

Prasanth R: 36 hrs

Fathima B:

Lipika L:

Change Log:

- V 1.1
  - Created basic structure of the document.
  - Added all initial basic things to the document.
- V 1.2
  - Designed component diagram
  - Worked on requirements traceability
- V 1.3
  - Added all other UML diagrams
  - Added Algorithmic part design
- V 1.4
  - Fixed few things in the document
  - Cleaned up the document for the final submission