Politecnico di Milano

Software Engineering 2

Integration Test Plan Document

Bressan G., de Santis S., Di Marco P.

POWER ENJOY

Professor Elisabetta Di Nitto

Academic Year 2016-2017

Contents

1 Introduction

1.1 Revision History

Version	Date	Authors	Summary
1.0	15/01/2017	Gabriele Bressan, Simone de Santis, Pietro Di Marco	Initial release

1.2 Purpose and Scope

This is the Integration Test Plan Document (ITPD) for the Power EnJoy system. The purpose of this document is to test the integration of the different components (already described Design Document) in order to avoid unexpected behaviors.
Integration test has to also ensure that the components of the Power EnJoy system work consistently with the requirements and guidelines specified in the RASD document. In this document will be discuss some points:

- Integration strategy
- Individual steps and test description
- Tools used and test equipment required
- Program stubs and test data required

2 Definitions, Acronyms, Abbreviations

- RASD: Requirement Analysis and Specification Document.
- DD: Design Document.
- ITBP: Integration test Plan Document
- DBMS: Database Management System.
- DB: Database of Power EnJoy.
- Java EE: Java Enterprise Edition.
- Subsystem: The biggest (high level) part of Power EnJoy system that contains one or more component.
- Component: Low level part of Power EnJoy system contained into Subsystem.
- GM: General Motors.
- Google Maps: Free service that allow to show the map information.

3 Reference Document

- The Integration Test Plan Example document: Integration Test Plan Example.pdf (taxi driver project)
- Power EnJoy Requirement Analysis and Specification Document: Power EnJoy RASD.pdf
- Power EnJoy Design Document: Power EnJoy DD.pdf

4 Integration Strategy

4.1 Entry Criteria

In this section are described all necessary prerequisites to have a correct point of view of the components into the system and how they interface each other and what are the functionalities that we want to implements.
Before start with Integration testing, the **Requirements Analysis Specification Document** and **Design Document** must have fully written and updated, and this is a crucial point to have a clear image of the entire system.

4.2 Elements to be Integrated

In this section are described all the components that need to be integrated together. Our system is based on the integration of different high-level components, each one dedicated to different functionalities.
Main high level components of Power EnJoy system are: User Manager, Ride Manager, Park Manager, Payment Manager, Communication Interface (it shows and handles the communication between different components), DBMS, Map Service and GM API.

At lowest level **User Manager** is composed by:

- User Registration
- Login
- Password Recovery
- User account manager

User Manager is stricted related to **History manager** which is a single component.
At lowest level **Power Enjoy ride manager** is composed by:

- Ride manager
- Reservation manager
- Vehicle manager

At lowest level **Park manager** is composed by:

- Power Station Manager
- Parking manager

At lowest level **Payment Handler Manager** is composed by:

- Payment Manager
- Discount Manager

The communication between mobile App, web App and iPad system is handled by the **Communication Interface**, a subsystem component that make indistinguishable the different devices used.

Administran Manageris a component that let the Administrator to manage and control the entire system. It interacts with the Data Access Utilities and the Communication Interface and does not require the interaction with any other component.

Map service is an external component that allow, once send the vehicles, parks and power station position (using Google API), to show on the maps the precise location of them.
Power EnJoy system is composed also from components that not require test because come from external entity.
These components are:

- DBMS
- Map service (Google Maps)
- General Motors API
- Payment system

The different components are implemented in the client side or in the server one depending on their functionality and also on the architectural choices already described in the design document.
Client side

- Mobile Application
- Web Application
- iPad Application

Server side

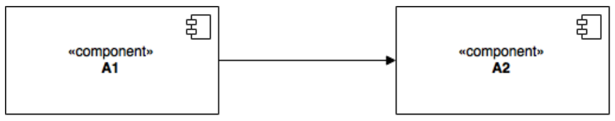
- DBMS
- Data Utilities
- User Manager
- Power Enjoy ride manager
- Park manager
- Payment manager

4.3 Integration Testing Strategy

In order to test properly all the functionalities of the system, we decide to test the entire system using bottom up approach. **Bottom up** approach starts from the bottom (individual modules are tested first) and gradually integrating together the components of the same module that work properly. At the end of this strategy we will have the system totally integrated where we will to apply simple changes.
In this way it is possible to test certain specific critical components deeply, this let us to have a more clear view of the entire system and interaction within the “not work modules” that could interfere with test result.
Bottom up way has some advantages like that test condition are easier than top down approach and a better observation of the test results. So using this strategy allow us to find and fix some problems and bugs of critical modules and assure us that the easier functionality work correctly.
In Power EnJoy system there are some components that came from external company, so these don’t need to be tested and these are: **DBMS (Database Management System), Map services (Google Maps), General Motors API and Payment system**.

5 Sequence of Component/Function Integration

In this part of the project we will explain and describe as all the components are integrated between them. We use a notation in which an arrow links two different components. The arrow that start from component A1 to component A2 means that A1 is necessary for A2.
This is an example:

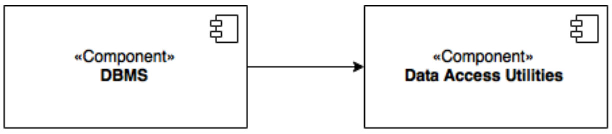


5.1 Software Integration Sequence

In the following paragraph we will describe how the low level component are integrated to create high level components.

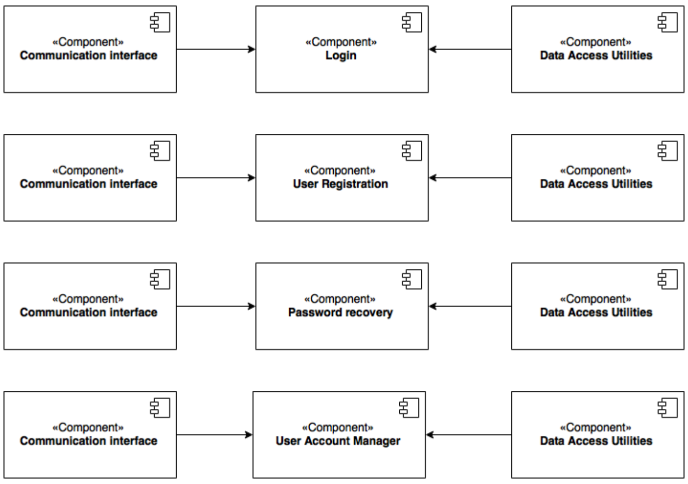
5.2 Data Access Utilities

During the test phase there must be two component integrated between them and they must be 100% completed. These component have a fundamental importance because have got all Power EnJoy data used to manage the system and statistics purpose and they are **Data Access Utilities** and **DBMS**.



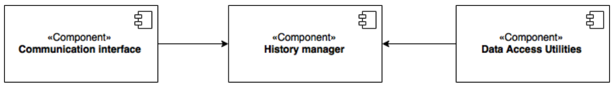
5.3 User Manager

User manager component is composed by three different components: **User registration, User login and User password recovery**. All of these subcomponents needs to be integrated with data access utilities to take, check and modify some information and at the same time they need to show the result of operation to the user using **communication interface**.



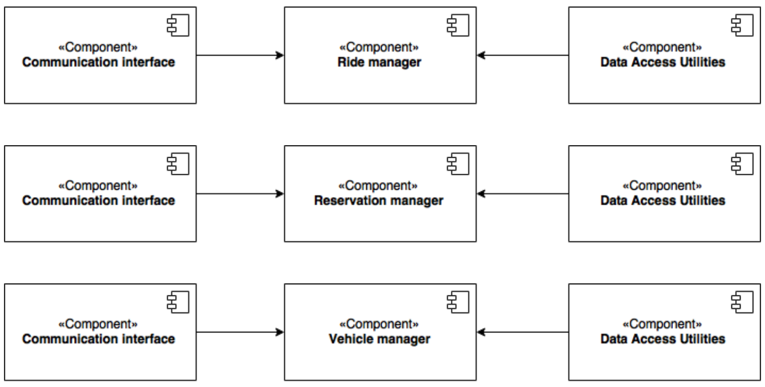
5.4 History manager

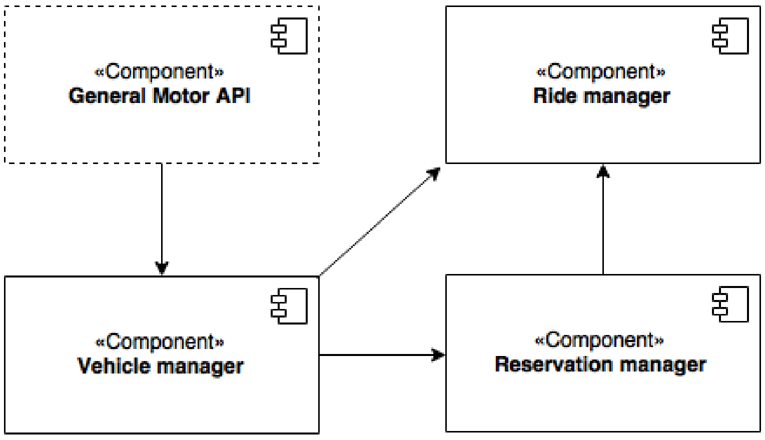
History manager is a single component with only purpose to show to the user his history trips and for this reason it needs to take trips information from the **Data Access Utilities** and show them to the user using **Communication Interface**.



5.5 Power Enjoy Manager

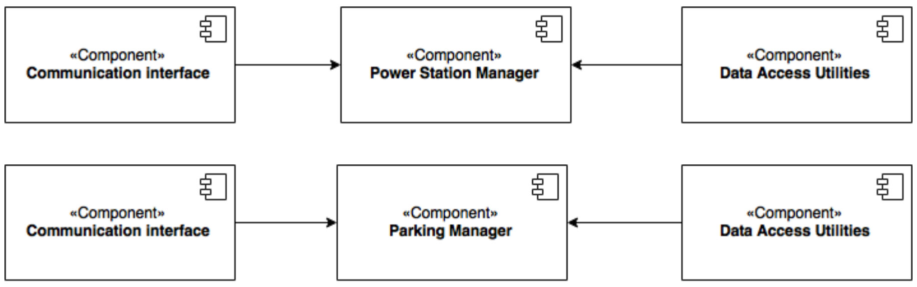
Power Enjoy Manager is the subsystem that handles the operations strictly related to the ride, so it is composed by: **Ride manager, Reservation manager, Vehicle manager.**
These components are integrated with the data Access Utilities and the Communication Interface and they need these two component to work properly, moreover the map service component is necessary to locate vehicles for reservation and during the ride (for using GPS Navigator).


«Component» Communication interface
«Component» Ride manager
«Component» Data Access Utilities
«Component» Communication interface
«Component» Reservation manager
«Component» Data Access Utilities
«Component» Communication interface
«Component» Vehicle manager
«Component» Data Access Utilities


«Component» General Motor API
«Component» Ride manager
«Component» Vehicle manager
«Component» Reservation manager

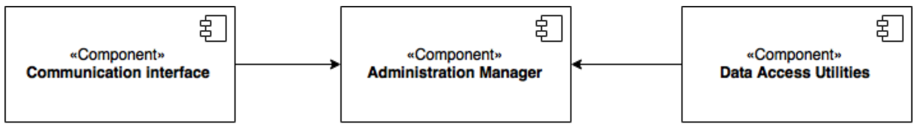
5.6 Park Manager

Park Manager is the subsystem that handles the operations strictly related to the ride, so it is composed by Power Station Manager and Parking Manager.
Power station manager is the component of Park Manager dedicated to manage the power station uses and it needs to read and memorize the information about the status of the singular power station tower.
Parking Manager is the component of Park Manager dedicated to manage the safe park zone and also Parking Manager needs to use Data utilities to read and write information about park status.
Power Station manager and Parking manager both don't need to use Map services because their position is already saved into database during installation phase and their position don't change continuously.


«Component» Communication interface
«Component» Power Station Manager
«Component» Data Access Utilities
«Component» Communication interface
«Component» Parking Manager
«Component» Data Access Utilities

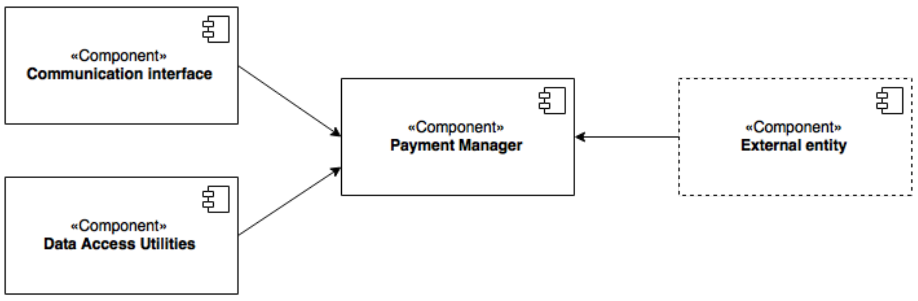
5.7 Administration Manager

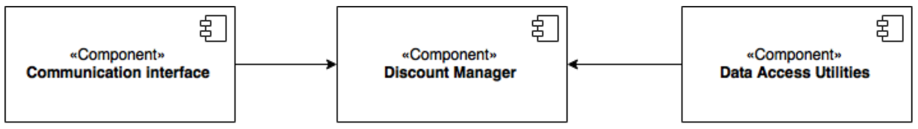
Administration Manager Administration manager is the subsystem that include the Administration Manager component. Administration Manager component is related to Data Access Utilities when the administrator need to modify some data and it is related to Communication interface to show the result of the query.

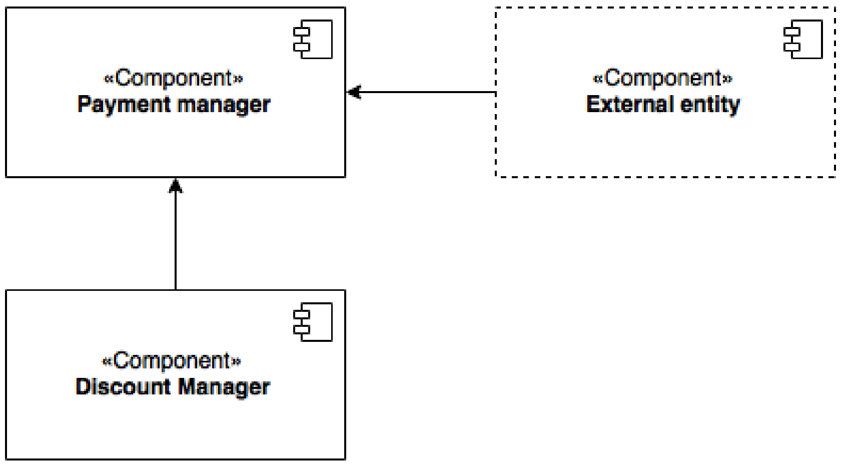


5.8 Payment Handler Manager

Payment Handler Manager is the subsystem that includes the Payment Manager and the Discount Manager that handle the payments of the users.
Like other components they require the Communication Interface and the Data Access Utility to work properly, in addition an external component(third part) handles the effective payment of the user.

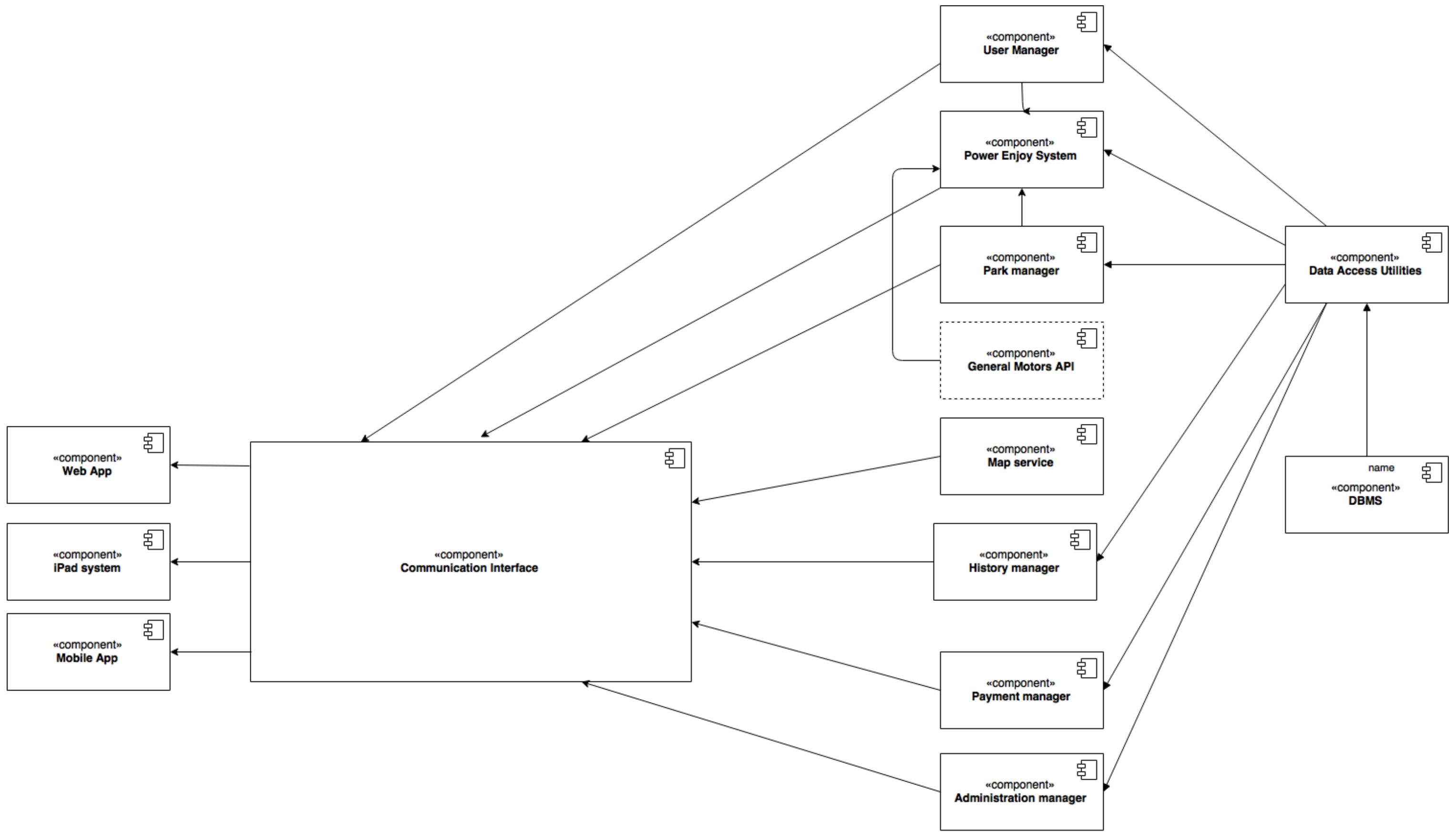

«Component» Communication interface
«Component» Data Access Utilities
«Component» Payment Manager
«Component» External entity


«Component» Communication interface
«Component» Discount Manager
«Component» Data Access Utilities


«Component» Payment manager
«Component» External entity
«Component» Discount Manager

5.9 Subsystem Integration Sequence

The graph below give a general overview of how the subsystem are integrated to create The Power EnJoy System.


«component» User Manager
«component» Power Enjoy System
«component» Park manager
«component» General Motors API
«component» Map service
«component» History manager
«component» Payment manager
«component» Administration manager
«component» Data Access Utilities
name
«component» DBMS
«component» Web App
«component» iPad system
«component» Mobile App
«component» Communication Interface

6 Individual Steps and Test Description

In this part of the project, for each pair of the components already described in the previous paragraphs,we will explain what are the methods used for integration of them explaining what are the input values and what is the return value. The component described are identified by the <caller; called> notation and there is a table that contains the list of methods that the <caller> component invokes on the <called> component and a brief description of them.

6.1 Administration Manager

Administrator can access and modify all relevant data of the Power EnJoy system and check if all components works correctly, because it's given to him permission to write, modify and delete.

6.1.1 Administration Manager, Data Access Utilities

addNewPowerStation(ID,Latitude,Longitude)	
Input A null parameters	Effect A NullArgumentException returns
Correct Data	Add a New PowerStation.
Incorrect Data	An InvalidPowerStationError returns

deleteUser(user)	
Input	Effect
A null parameters	A NullArgumentException returns
Inexistent user	An InvalidCredentialsError returns
Valid User.	User is dropped.

addNewSafePark(ID,Latitude,Longitude)	
Input	Effect
A null parameters	A NullArgumentException returns
Correct id, latitude, longitude	Add a New SafePark.

addNewVehicle(LicencePlate)	
Input	Effect
A null parameters	A NullArgumentException returns
Correct Licence plate	Add a new Vehicle.

6.2 User Manager

6.2.1 Login, Data Access Utilities

loginRequest(mail,password)	
Input	Effect
A null parameters	A NullArgumentException returns
Inexistent user	An InvalidCredentialsError returns
An invalid combination of mail and password	An InvalidCredentialsError returns
Valid combination of mail and password (the user exists)	User page returns

6.2.2 User Registration, Data Access Utilities

createNewUser(registationFormData)	
Input	Effect
A null parameters	A NullArgumentException returns
An invalid registration form data	A WrongFormDataError returns
A valid registration form data	User successfully inserted into database

checkUserAlreadyExist(data)	
Input	Effect
A null parameter	A NullArgumentException returns
An already existing registration form data (user already exists)	A UserAlreadyExistsError returns
A valid registration form data	A boolean False returned

6.2.3 User Account Manager, Data Access Utilities

removeAccount(mail,password)	
Input	Effect
A null parameters	A NullArgumentException returns
Inexistent user	An InvalidCredentialsError returns
An invalid combination of mail and password	An InvalidCredentialsError returns
Valid combination of mail and password (the user exists)	User removed from database

modifyAcount(mail, modifyData)	
Input	Effect
A null parameters	A NullArgumentException returns
Inexistent mail user	An InvalidMailError returns
Invalid modify data	WrongFormDataError returns
Correct mail and modify data	User data succesfull changed

changePassword(user,newPassword)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid or not existing user	A InvalidUserException returns
Invalid new password format	A WrongPassowrdFormatException returns
Correct user and new password	User sets new password

6.2.4 Password recovery, Data Access Utilities

passwordRecovery(mail, pinSentInMail, newPassword)	
Input	Effect
A null parameters	A NullArgumentException returns
Not correct/valid mail	A WrongUserMailException returns
Invalid pin sent in mail	A WrongPinException returns
Invalid new password format	A WrongPassowrdFormatException returns
Correct combination of mail, pin sent in mail and new password	User sets new password

6.3 History Manager

6.3.1 History manager, Data Access Utilities

getHistoricalTripList(user)	
Input	Effect
A null parameter	A NullArgumentException returns
An invalid/inexistent user	A InvalidUserException returns
Correct user	Return a list of user's trips

addUserTrip(user,trip)	
Input	Effect
A null parameters	A NullArgumentException returns
An invalid/inexistent user	A InvalidUserException returns
An invalid trip	A WrongTripException returns
Correct user and trip	A new user's trip added to database

removeUserTrip(user,trip)	
Input	Effect
A null parameters	A NullArgumentException returns
An invalid/inexistent user	A InvalidUserException returns
An invalid trip	A WrongTripException returns
Correct user and trip	A user's trip deleted from database

6.4 Power EnJoy Manager

6.4.1 Reservation Manager, Data Access Utilities

addReservation(reservation)	
Input	Effect
A null parameter	A NullArgumentException returns
Invalid reservation	A InvalidReservation
Correct reservation	A new user's reservation is added to the database

setTimeExceeded(reservation)	
Input	Effect
A null parameter	A NullArgumentException returns
Reservation already exceeded	A InvalidReservation
Correct reservation	Reservation setted to exceeded into the database

getReservation(user)	
Input	Effect
A null parameter	A NullArgumentException returns
Invalid User	A InvalidUser
Correct User	A list of user's reservation returns.

changePassword(user,newPassword)	
Input	Effect

getCarPosition(vehicle)	
Input	Effect
A null parameter	A NullArgumentException returns
Invalid vehicle	A InvalidVehicleException returns
Correct vehicle	The position of the vehicle returned

openCar(vehicle)	
Input	Effect
Invalid vehicle	An InvalidVehicleException returns
Valid vehicle	Opens the vehicle

deleteReservation(reservation)	
Input	Effect
A null parameter	A NullArgumentException returns
An Invalid reservation	An InvalidReservationException returns
Valid reservation	Delete a specified reservation from database

6.4.2 Vehicle Manager, Data Access Utilities

getVehicleStatus(vehicle)	
Input	Effect
Invalid parameter	An InvalidVehicleException returns
Valid vehicle	Get the status of vehicle

setVehicleStatus(vehicle,status)	
Input	Effect
A null parameters	A NullArgumentException returns
An invalid status	An InvalidStatusException returns
Invalid vehicle	An InvalidVehicleException returns
Correct vehicle and status	Vehicle set with new status

getVehiclePosition(vehicle)	
Input	Effect
A null parameter	A NullArgumentException returns
Invalid vehicle	An InvalidVehicleException returns
A valid vehicle	Get vehicle position

setVehiclePosition(vehicle,position)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid vehicle	An InvalidVehicleException returns
Invalid position	An InvalidPosition Exception return
A valid vehicle and position	Set the vehicle position

getAvailableVehicles()	
Input	Effect
Nothing	Return a list of available vehicles

setGuestsNumber(vehicle,number)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid vehicle	An InvalidVehicleException returns
Valid vehicle and number	Sent the number of car's guest into database

setBatteryLevel(vehicle,number)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid vehicle	An InvalidVehicleException returns
Valid vehicle and number	Set the car battery level percentage into database

getBatteryLevel(vehicle)	
Input	Effect
A null parameter	A NullArgumentException returns
Invalid vehicle	An InvalidVehicleException returns
Valid vehicle	Get the battery level of specified vehicle

setPlugState(vehicle,status)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid vehicle	An InvalidVehicleException returns
Invalid status	An InvalidStatusException returns
Valid vehicle and status	Car plug state is set into database

6.4.3 Ride Manager, Data Access Utilities

addRide(ride)	
Input	Effect
A null parameter	A NullArgumentException returns
An invalid ride	A NotValidRideException returns
Valid ride	A ride is added into database

setStartTime(Time)	
Input	Effect
A null parameter	A NullArgumentException returns
An invalid time	A InvalidTimeException returns
Valid Time	set start Time.

setStopTime(ride)	
Input	Effect
A null parameter	A NullArgumentException returns
A invalid ride	A InvalidRideException returns
Valid ride	Set stop time

setDate(ride)	
Input	Effect
A null parameter	A NullArgumentException returns
An invalid ride	A InvalidRideException returns
Valid Ride	Set Date

setDistance(ride)	
Input	Effect
A null parameter	A NullArgumentException returns
An invalid ride	A InvalidRideException returns
Valid Ride	Set Distance

setEndPointPosition(ride)	
Input	Effect
A null parameter	A NullArgumentException returns
An invalid ride	A InvalidRideException returns
Valid Ride	Set end position

getEndPointPosition(ride)	
Input	Effect
A null parameter	A NullArgumentException returns
An invalid Ride	A InvalidRideException returns
Valid Ride	Get a end position of specified ride

getGuestNumber(ride)	
Input	Effect
A null parameter	A NullArgumentException returns
An invalid Ride	A InvalidRideException returns
Valid Ride6Get a car guests number of a specified ride	

setRideStatus(status)	
Input	Effect
A null parameter	A NullArgumentException returns
An invalid status	An InvalidStatusException returns
Valid status	Ride status set into database

setStartPosition(Position)	
Input	Effect
A null parameter	A NullArgumentException returns
An invalid Position	An InvalidPositionException returns
Valid Position	Set Ride Position into database

getStartPosition(ride)	
Input	Effect
A null parameter	A NullArgumentException returns
An invalid ride	A InvalidRideException returns
Valid ride	get the start postion of selected ride

getRideStatus(ride)	
Input	Effect
A null parameter	A NullArgumentException returns
An invalid ride	A InvalidRideException returns
Valid ride	get the selected ride status

6.4.4 Reservation Manager, Vehicle Manager

setVehicleStatus(status)	
Input	Effect
A null parameter	A NullArgumentException returns
An invalid status	An InvalidStatusException returns
A valid status	A vehicle status is set

getCarsAvailable()	
Input	Effect
Nothing	Return available cars.

6.4.5 Reservation Manager, Ride Manager

addRide(reservation)	
Input	Effect
A null parameter	A NullArgumentException returns
An invalid reservation	An InvalidReservationException returns
Valid reservation	Return nothing

6.4.6 Ride Manager, Vehicle Manager

getVehiclePosition()	
Input	Effect
Nothing	Return vehicle position

getDistance()	
Input	Effect
Nothing	Get the covered distance

getGuestNumber()	
Input	Effect
Nothing	Get the guests number into the car

6.4.7 Vehicle Manager, Ride Manager

rideStatus(status)	
Input	Effect
A null parameter	A NullArgumentException returns
An invalid status	An InvalidStatusException returns
Valid Status	Ride status is setted to the parameter received

6.5 Park Manager

6.5.1 Parking Manager, Data Access Utilities

setParkStatus(park,status)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid park	An InvalidParkException returns
Invalid status	An InvalidStatusException returns
Valid park and status	Park status is set

getParkStatus(park)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid park	An InvalidParkException returns
Valid park	A park status returns

getParkList()	
Input	Effect
Nothing	Return list of Parks.

6.5.2 Power Station Manager, Data Access Utilities

getParkWithPower()	
Input	Effect
Nothing	Return list of Park With Power Station.

setPowerStationStatus(status)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid status	An InvalidStatuException returns
Valid status	Power station status is set

6.6 Payment Handle Manager

6.6.1 Payment Manager, Data Access Utilities

addPaymentRecord(payment)	
Input	Effect
A null parameters	A NullArgumentException returns
An invalid payment	An InvalidPaymentException returns
A valid payment	A payment is added into database

getTimeTrip()	
Input	Effect
Nothing	Return the time of trip

6.6.2 Discount Manager, Data Access Utilities

getDiscountInformation(reservation)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid reservation	An InvalidReservationException returns
A valid reservation	Return information about specified reservation

6.6.3 Discount Manager, Payment Manager

setDiscount(discount, reservation)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid discount	An InvalidDiscountException returns
Invalid reservation	An InvalidReservationException returns
Valid discount and reservation	Set the discount value for payment of specified reservation

6.6.4 Payment Manager, Discount Manager

checkDiscount(reservation)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid reservation	An InvalidReservationException returns
Valid reservation	Return a boolean value that indicate if exists or not a discount for a specific reservation

6.7 Integration between subsystem

6.7.1 Power EnJoy System, Park Manager

setParkStatus(position,status)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid position	An InvalidPositionException returns returns
Invalid status	An InvalidStatusException returns
Valid position and status	Set the park status

setPowerStationStatus(position,status)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid position	An InvalidPositionException returns returns
Invalid status	An InvalidStatusException returns
Valid position and status	Set the power station status identified by position

6.7.2 Power Enjoy System, General Motor API

getVehicleInformation(vehicle)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid vehicle	An InvalidVehicleException returns
Valid vehicle	Get information about specified vehicle

disableIgnition(vehicle)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid vehicle	An InvalidVehicleException returns
Valid vehicle	Disable ignition of specified vehicle

lockDoor(vehicle)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid vehicle	An InvalidVehicleException returns
Valid vehicle	Lock the doors of specified vehicle

openCar(vehicle)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid vehicle	An InvalidVehicleException returns
Valid vehicle	Unlock the doors of specified vehicle

start(vehicle)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid vehicle	An InvalidVehicleException returns
Valid vehicle	Start ignition of specified vehicle

getBatteryLevel(vehicle)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid vehicle	An InvalidVehicleException returns
Valid vehicle	Get battery level percentage of specified vehicle

checkCarPlug(vehicle)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid vehicle	An InvalidVehicleException returns
Valid vehicle	Check if specified vehicle is connected with power station

6.7.3 Communication Interface, User Manager

checkRequest(mail,password)	
Input	Effect
A null parameters	A NullArgumentException returns
An invalid mail	A NotUserExistException returns
An invalid password	A InvalidPasswordException returns
Valid match between mail and password	Return a boolean value that indicates if an account exists

requestNewAccount(registrationFormData)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid registration form data	WrongFormDataError returns
Valid registration form data	A new account is created

removeAccount(mail,password)	
Input	Effect
A null parameters	A NullArgumentException returns
An invalid mail	A NotUserExistException returns
An invalid password	A InvalidPasswordException returns
Valid match between mail and password	A specified account is deleted

modifyAccount(mail,modifyData)	
Input	Effect
A null parameters	A NullArgumentException returns
An invalid mail	A NotUserExistException returns
An invalid modify data	A WrongDataInsertException returns
Valid mail and modify data	A specified account data is modified

This method is used when user wants to change his password

changePassword(user,newPassword)	
Input	Effect
A null parameters	A NullArgumentException returns
An invalid user	A NotUserExistException returns
An invalid new password format	A WrongPasswordFormatException returns
Valid user and new password	A password is changed

This method is used when user loses the password or when he forgot it

passwordRecovery(mail,pinSentInMail,newPassword)	
Input	Effect
A null parameters	A NullArgumentException returns
An invalid mail	A NotUserExistException returns
Invalid pin sent in mail	A WrongPinException returns
Valid user and new password	A password is changed
Valid mail, pin and new password format	A temporary password is sent in user email

6.7.4 Communication Interface, History Manager

getHistoricalTripList(user)	
Input	Effect
A null parameters	A NullArgumentException returns
An invalid user	A NotUserExistException returns
Valid user	Return a trip list of specific user returns

6.7.5 Communication Interface, Park Manager

getParks()	
Input	Effect
Nothing	Return a list of parks

6.7.6 Communication Interface, Map Service

addInformationOnMap(parks,vehicles,powerStations)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid parks	An InvalidParksException returns
Invalid vehicles	An InvalidVehiclesException returns
Invalid power station	An InvalidPowerStationException returns

getMap()	
Input	Effect
Nothing	Return a map with vehicles, parks and power stations

6.7.7 Communication Interface, Payment Manager

addPayment(reservation)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid reservation	An InvalidReservationException returns
Valid reservation	Return a boolean value that indicates if payment is done or not

6.7.8 Communication Interface, Administration Manager

The Communication Interface in this case can perform every invocation of every method in the system.

6.7.9 Communication Interface, Power EnJoy System

sendLocation(position)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid position	An InvalidPositionException returns
Valid position	Return nothing

getVehicleInformation(vehicle)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid vehicle	A InvalidVehicleException returns
Valid vehicle	Return a list of vehicle information

reserveVehicle(vehicle,user)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid vehicle	An InvalidVehicleException returns
Invalid user	An InvalidUserException returns
Valid user and vehicle	Return a boolean value that indicates if reservation of specified vehicle is done or not

setPowerStationStatus(position,status)	
Input	Effect
A null parameters	A NullArgumentException returns
Valid Postion and Status	Set status of vehicle to charge, and enable power station.

getPosition(vehicle)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid vehicle	An InvalidVehicleException returns.
Valid vehicle	A position of the vehicle returns.

openCar(vehicle)	
Input	Effect
A null parameters	A NullArgumentException returns
Invalid vehicle	An InvalidVehicleException returns.
Valid vehicle	Unlock vehicle doors.

takeABreak()	
Input	Effect
Nothing	Set Vehicle in break status.

parkCar()	
Input	Effect
Nothing	set Vehicle in status park and all the relative information.

chargeCar()	
Input	Effect
Nothing	Abilitate charging of vehicle.

insertPinCode(pinCode)	
Input	Effect
A null parameter	A NullArgumentException returns
Invalid pinCode	Request to insert new pinCode
Valid pinCode	validate pinCode and unlock vehicle.

resumeDrive()	
Input	Effect
Nothing	Send command in order to resume drive.

6.7.10 Payment Manager, Communication Interface

paymentDone(payment)	
Input	Effect
A Null parameter	A NullArgumentException returns
A valid payment	Show summary of payment information.
Invalid payment	An InvalidPaymentException returns.

7 Performance Analysis

In order to work properly, mobile systems both Android and iOs will support Power EnJoy Mobile App, all the other devices can only access through Web Page with restricted capabilities. This system will support only iOs devices running iOs 8, Android 5.0 devices and Windows Phone 8 devices(assuming that hardware requirements for running correctly operating system will be much enough to run this client app).

All System Mobile App functionalities will be tested in some different android smartphone running Android 5 and Windows Phone 8 Out of the box.

8 Tools and Test Equipment Required

8.1 Tools

In this part of the project we will introduce the tools used during the test phase for Power EnJoy system components. For the choice of the tools we decided to use the tools that integrate better with Java Enterprise Edition, so for this reason we used:

- **JUnit framework:** is one of the most important framework for testing Java code. In according with our bottom-up approach, we used JUnit first of all to test the low level methods and components in such a way to have a solid foundations for more complex functionalities and structures. In other words JUnit has helped us to verify that the interaction between components are producing the expected results and in particular we use it to check if the methods returns wanted results.

- **Arquillian framework:** Power EnJoy is a distributed system in sense that it is composed by database, server part and client part. For these reasons we decided to use Arquillian framework, because allow us to test the entire system inside a remote or embedded container. In other words Arquillian can handle the test of the containers and their integration with JavaBeans.

- **Mockito framework:** is a framework that we use to perform tests about the interaction between different components. Moreover Mockito lets us testing also the values returned after the method invocation depending on the input parameters and the exceptions eventually raised.

8.2 App Performance Analysis

In order to perform performance tests on the user App we used some specific analysis tools depending on the device tested, in particular these kind of tests are useful to determinate the amount of CPU and main memory required by the different OS. Different operating systems in fact handle the App in different ways, our work is to guarantee a reasonable resources usage by all of them, here are reported the main tools that we used:

Operating System	**Tool Name**	**Description**
Android	Systrace	Systrace helps you analyze how the execution of your application fits into the many running systems on an Android device. It puts together system and application thread execution on a common timeline. In order to analyze your app with Systrace, you first collect a trace log of your app, and the system activity. The generated trace allows you to view highly detailed, interactive reports showing everything happening in the system for the traced duration.
Android	Little Eye	Little Eye allows monitoring of all apps built for Android 2.3 or later, including system apps. Once the device is plugged in, Little Eye automatically detects all installed apps on your device. By the app you can select an app and start monitoring.
iOS	OpenGL Driver Monitor	Gathers GPU-related performance data, including data related to VRAM usage, video bus traffic, and hardware stalls among others. You can use this information to identify the cause of temporary slowdowns or sporadic hesitations in your OpenGL application.
iOS	OpenGL Profiler	Creates a runtime profile of your OpenGL-based application. You can view function statistics and the call-trace history of your application's OpenGL calls.
iOS	the full suite of performance analysis tools provided by the Xcode IDE.	
Windows Phone	Application Analysis tool	App Monitoring. With the app monitoring option you can evaluate the most important behaviors of your app that contribute to a good user experience, such as start time and responsiveness.
Windows Phone	Application Analysis tool	Profiling. With the profiling option you can evaluate either execution-related or memory-usage aspects of your app.

8.3 Test Equipment

8.3.1 Client side

The test phase of the system on client side, was made using different operating system so we test the application on different environments. Thanks to developer tools and sdk installed into PC, we can simulate the application on different OS such (iOS, Android, Windows Phone), and verify the correctness of graphical aspects, the performance and the correct execution of application. One on the most important consideration about Mobile App is about graphical aspect due to the large number of different models of smartphones, because each of them has got different hardware characteristic such display resolution and aspect ratio. To fix the problem about display resolution and size we decide to implement and improve the graphical aspect for these type of devices:

- Android smartphone and tablet with display size from 4" to 12"
- iOS smartphone and iPad with display size from 3.5" to 12.9"
- Windows Phone with display size from 4" to 5.7"

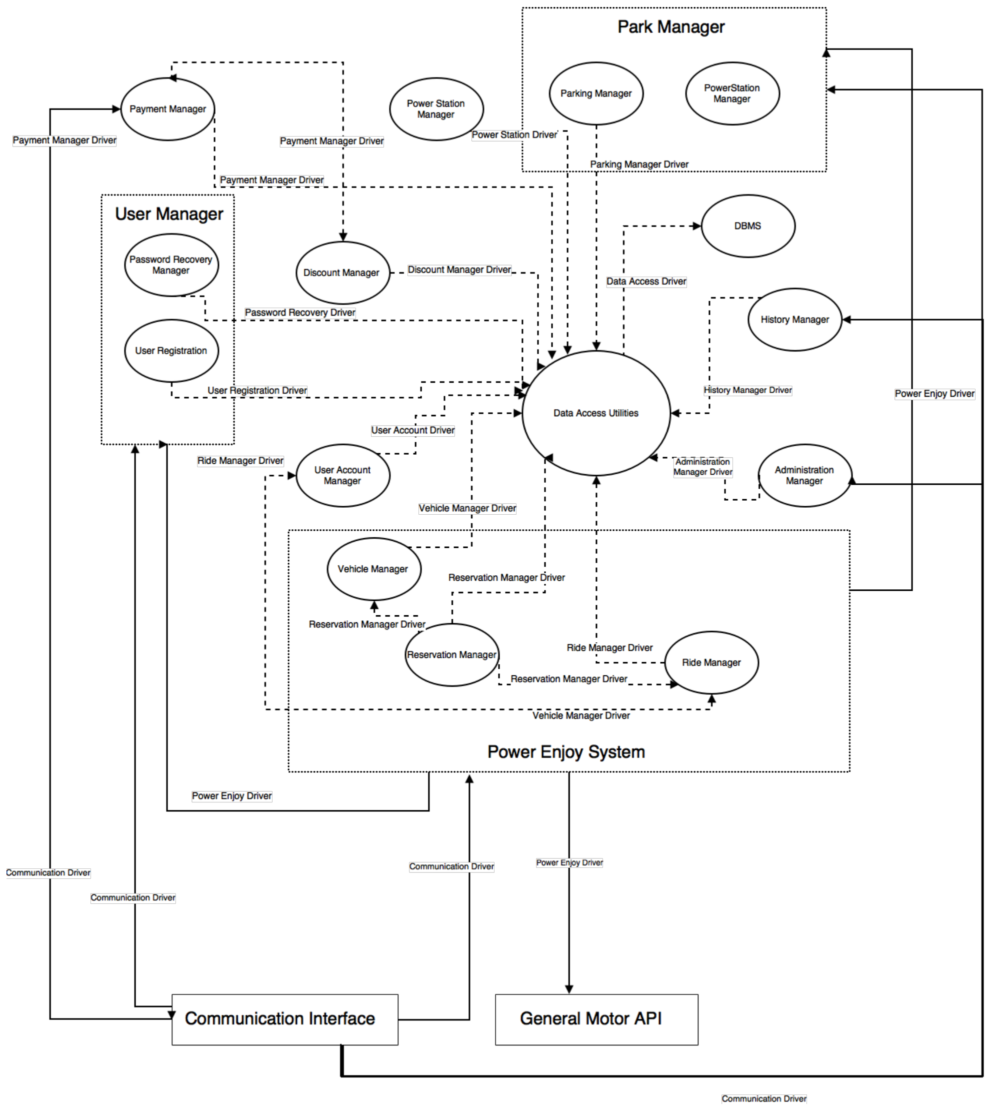
The test part about web application was made also in this case using a Google Chrome component that permit to simulate different resolution, so there is no particular problem about display size. Furthermore the web application was tested also on different operating system that use differents browser such **Safari, Google Chrome, Mozilla Firefox, Internet Explorer, Chromium** and **Microsoft Edge**.

Web Server side will be tested on a common Web Service, running OpenShift software by Red Hat. Configuration will use Glassfish, Java Persistence and mySQL.

9 Required Program Stubs and Test Data

9.1 Program Stubs and Drivers

In order to test all the different component we decided to adopt a bottom-up approach, as we explained before, so now we show the drivers that we developed to complete and perform the testing phase. Every driver is dedicated to test a specific area of our system and the interaction between the different components of the same subsystem.



9.2 Test Data

In order to test the methods already defined we need to have valid and invalid candidates, this list includes the instances of these components:

- **User Registration**, with these problems:
  - Null object
  - Null fields
  - Invalid mobile phone number
  - Invalid email address
  - Invalid ID Card
  - Invalid Licence
- **Ride Manager**, with these problems:
  - Null Object
  - Null fields
  - End Time before Start Time
- **Vehicle Manager**, with these problems:
  - Null Object
  - Null fields
  - Invalid Licence
- **Reservation Manager**, with these problems:
  - Null Object
  - Null fields
  - Invalid Position
  - Invalid Pin Code
- **Payment Manager**, with these problems:
  - Null Object
  - Null fields
  - Negative amount

10 Hours of work

For this part of the project, we spent 40 hours per person.