POLITECNICO
DI MILANO

Computer Science and Engineering
A.A. 2016/2017
Software Engineering 2 Project:
Power&Joy

Integration Test Plan Document

November 11, 2016

Prof. Luca Mottola
Joshua Nicolay Ortiz Osorio Mat: 806568
Michelangelo Medori Matr: 878025

Index

1 Introduction

1.1 Revision History

Version	Date	Author(s)	Summary
1.0	10/01/2017	Joshua Nicolay Ortiz Osorio and Michelangelo Medori	initial release
2.0	14/01/2017	Joshua Nicolay Ortiz Osorio and Michelangelo Medori	completed sections 5 and 6, small changes to section 3

1.2 Purpose and Scope

This is the Integration Testing Plan Document for the Power&Joy car sharing management system. The purpose of this document is to describe in a detailed way the organisation of the integration test activity, which involves all the components that are going to be assembled to build the system
In the following sections we are going to provide:

- In section 2.1 we describe the criteria that must be met by the project status before integration testing of the outlined elements may begin
- In section 2.2 we provide a list of the subsystems and their subcomponents involved in the integration testing activity
- In section 2.3 we provide a description of the integration testing approach
- The sequence in which components and subsystems will be integrated is described in sections 2.4.1 and 2.4.2
- Section 3 contains a description of the planned testing activities for each integration step, including their input data and the expected output
- A list of all the tools that will have to be employed during the testing activities, together with a description of the operational environment in which the tests will be executed can be found in section 4 and 5

1.3 List of Definitions and Abreviations

1.3.1 Definitions

- Subcomponent: each of the low level components realising the functionalities of a subsystem.
- Subsystem: a high-level functional unit of the system.

1.3.2 Acronyms

- RASD : Requirements Analysis and Specifications Document
- DD: Design Document
- ITPD: Integration Test Plan Document
- DBMS: Data Base Management System
- GPS: Global Positioning System
- UI: User Interface
- Notification system: a combination of software and hardware that provides a means of delivering a message to a set of recipients
- User : someone who registered into the system and can can access the system in order to reserve and ride a car
- Operator: Someone who provides assistance to users, works in one of the stations
- Station: place where operators work
- CarId: Unique identifier for every car
- ReservationId: Unique identifier for every Reservation
- RideId: Unique identifier for every Ride
- UserId: Unique identifier for every User
- OperatorId: Unique identifier for every Operator

1.4 List of Reference Documents

- Assignment AA 2016-2017
- Requirements analysis and Specifications Document (RASD)
- design Document (DD)
- Integration Test Plan Example.pdf
- Integration testing example document.pdf

2 Integration Strategy

2.1 Entry Criteria

Before the Integration test can start, there are several conditions about the current state of the project workflow that should be verified. First of all the RASD and DD need to be fully completed and available. Furthermore, the test should only start if the components to be integrated have reached the completion levels described below:

- 100% for the Data Access Utility
- At least 90% for the Car sharing management system
- At least 75% for the System Administration and Account Management subsystems
- At least 60% for the client applications

These percentages are to be considered at the beginning of the integration test and take into account also the order in which the components need to be integrated and the time needed for the integration test. For any component, the minimum percentage that allows it to be considered for integration is 90%

2.2 Elements to be Integrated

In this paragraph we are going to describe the structure of the Power&Joy system in terms of its subsystems and the subcomponents they are made of. The subsystems are the following:

- Car sharing management system
- System administration
- Account Management system
- Data Access Utilities
- DBMS
- Mapping Service
- Notification system
- Remote services interface

The Car sharing management system is composed by the following lower level subcomponents:

- Reservation management
- Ride Management

- Location management
- Car management
- Issues management

These subcomponents strongly rely on each other, and their precise integration is necessary to build all the functionalities the Car sharing management system.
The account management and system administration subsystems, are two collections of subcomponents that do not interact with each other, but their functionalities fall under the same field. The subcomponents constituting these subsystems are the following: For the Administration subsystem:

- The operator management
- The Station location
- The service statistics
- Discounts&penalities management
- Payment management

The account management subsystem takes care both of the users and the operators accounts, registrations and logins, and it is made by the following subcomponents::

- The Login management
- The password retrieval management
- The registration management
- The settings management

Once the lower level subcomponents have successfully been integrated, we can think about integrating the resulting high level subsystems together in order to have the Power&Joy System. The data access utilities is an atomic high level component , while all the other high level components (DBMS, mapping service, remote services interface and the notification system) already exist and are just integrated in order to provide the system with more functionalities.

2.3 Integration Testing Strategy

The strategy we are going to use to perform the integration tests is mainly the bottom-up approach, that consists on integrating those components which do not functionally depend on other components of the system first, and proceeding with all the other components, as soon as the ones they depend on are already been successfully integrated.
This approach is very reliable, since the final system is going to be built from subsystems which have already been tested and knowing that they work properly. This implies that in case errors occur, they only affect the current step of the integration, and not the integration already performed.
It is appropriate and efficient that the components to be integrated are actually developed with a bottom-up approach as well, which means that the integration tests can be performed as soon as the components they involve are fully developed. The fact that many components are often functionally independent from each other, also lets us choose the ones to integrate first, enabling us to follow a critical-first approach as well, which means that we can start integrating the highest risk subsystems, that actually are the ones that take care of the main functionalities of the system (i.e. its management subsystem, which is composed by the reservation management, ride management, issues management, location management and car management subcomponents, as described above) before concentrating with all the other components, that take care of minor functionalities.
We also have some components that are commercial components, and thus come already fully developed and working and can be used in a bottom up approach already at the beginning of the integration test. These components are DBMS, mapping service, remote service interface and notification system.

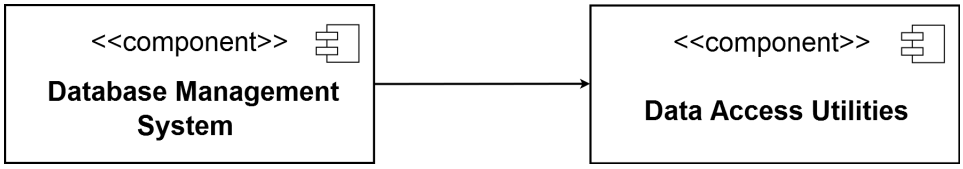
2.4 Sequence of Component/Function Integration

In this section we?re going to describe the order of integration (and integration testing) of the various components and subsystems of Power&Joy system. As a notation, given C1 and C2 two components/subcomponent/subsystems, an arrow from C1 to C2 means that C2 requires C1's subcomponents to have been fully implemented, integrated and tested and C1 to properly work in order for C2 to function .

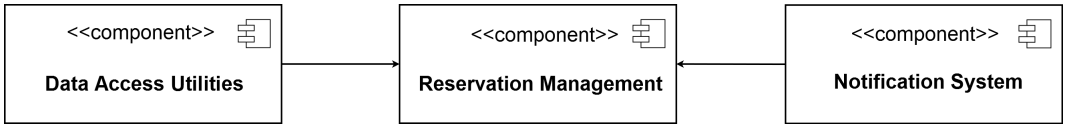
2.4.1 Software Integration Sequence

Following the bottom up approach, we are going to describe in which order the subcomponents are to be integrated and tested.

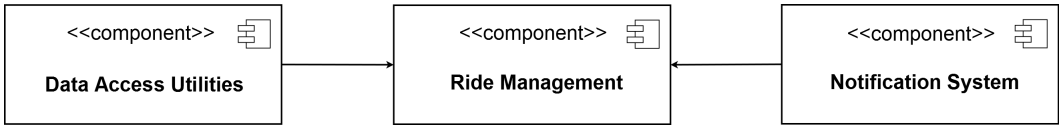
Data Access Utilities
The first two elements to be integrated are the Data Access Utilities and the Database Management System components, since every other component relies on Data Access Utilities to perform queries on the underlying database.



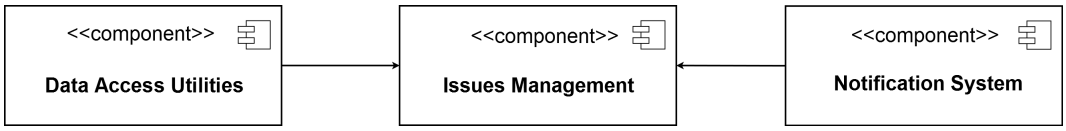
Car sharing management system
Now we are going to describe how to integrate the subcomponents of the Car sharing management system, which is the core of the Power&Joy System. First we want the Reservation management subcomponent to be integrated with the Data Access Utilities and the Notification System:



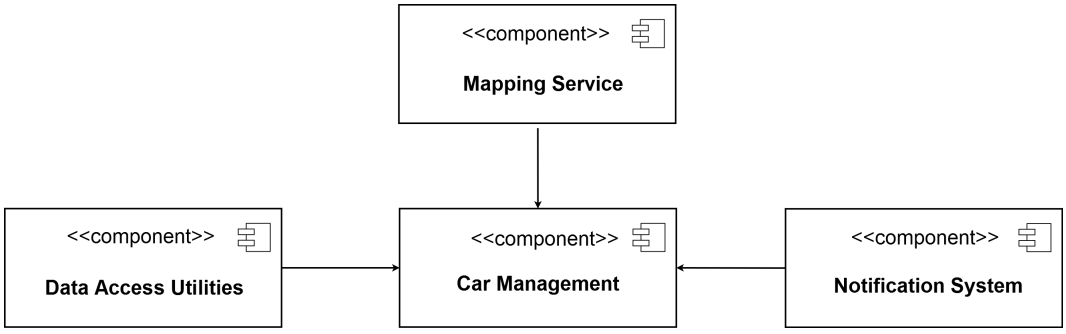
Then we need the Ride Management subcomponent to be integrated with the Data Access utilities and the Notification System:



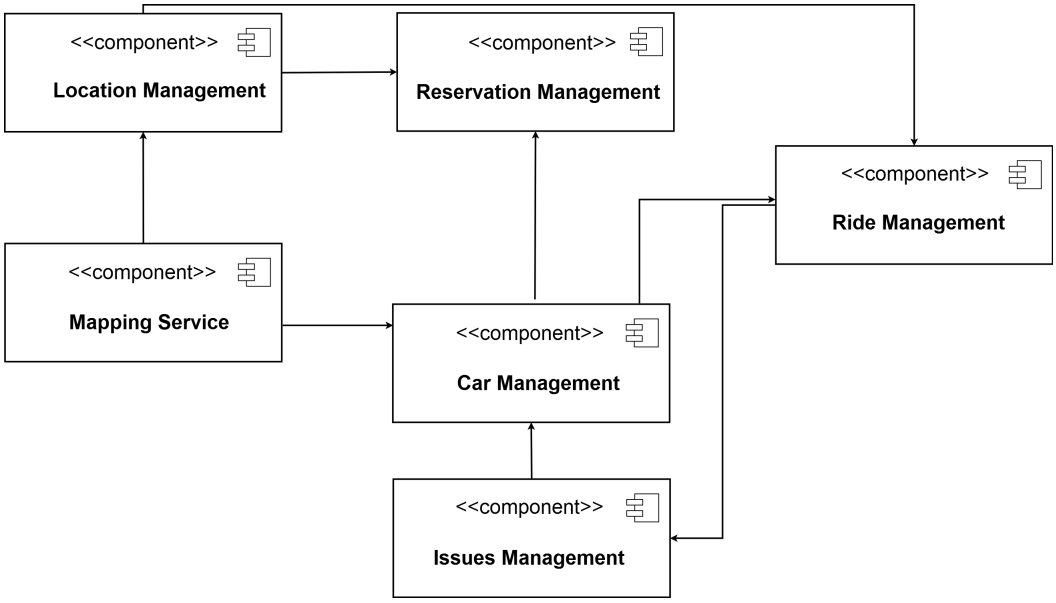
The same is for the Issues manager, to be integrated with the Data Access Utilities and the Notification System:



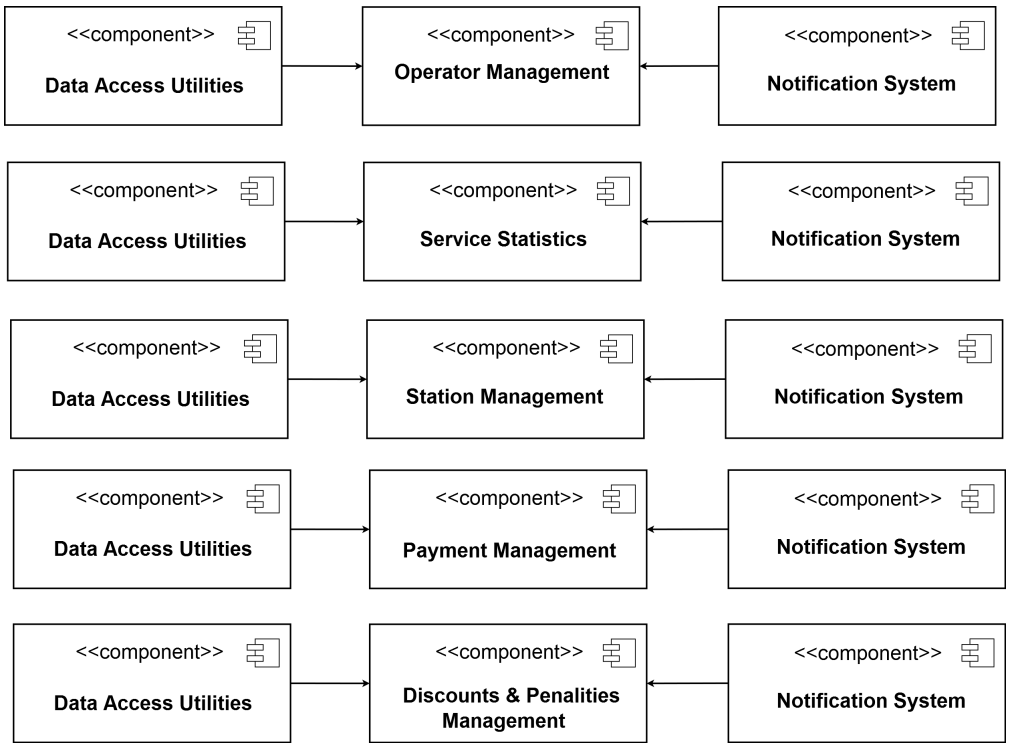
Finally, we need to integrate the car Management subcomponent with the notification Service, Data Access utilities and Mapping Service:



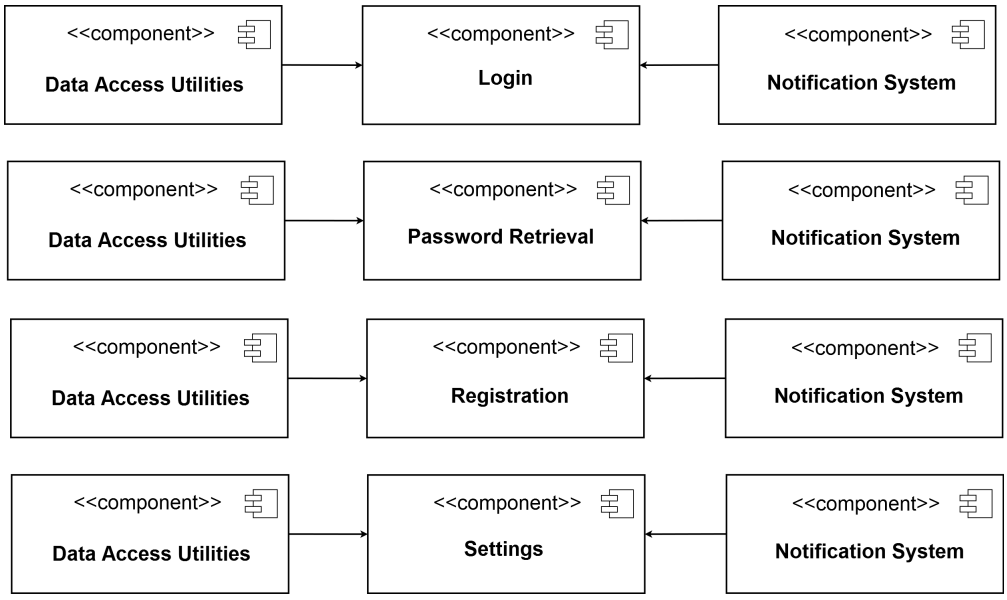
Now we are ready to integrate together the Reservation Management, Ride Management, Location Management and Car Management together in order to build the Car sharing subsystem:



System Administration
Following the critical-first strategy, the next step is to integrate the System Administration, which takes care of many important functions of the system , with the Notification System and the Data Access utilities. The System Administration is just a collection of subcomponents which are loosely correlated to each other, so they can be integrated with the rest of the system one by one in any order. The System Administration is just a wrapper for the methods of all its subcomponents. The way the components are integrated is showed below:

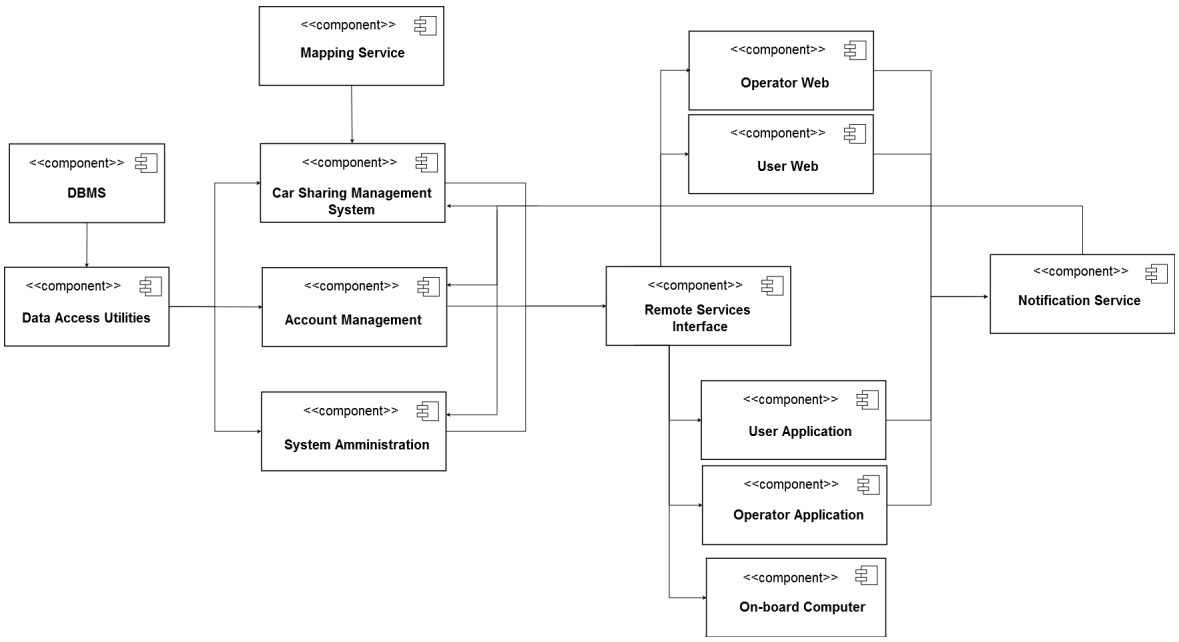


Account Management
The next step consists in integrating the subcomponents of the Account Management subsystem. The subcomponents implementing this subsystem are not dependent from each other, thus they can be integrated independently with the rest of the system, in particular with the Notification system and the Data Access utilities. Since these components are not really related to each other (each one of them takes care of a particular function related to the account management), the Account Management subsystem is just a wrapper for their methods (just like the System Administration). The way the components are integrated is showed below:



2.4.2 Subsystem Integration Sequence

In the following diagram we provide a detailed view of how the various high-level subsystems we build up to now have to be integrated together to create the full Power&Joy system.



3 Individual Steps and Test Description

3.1 Car sharing management subsystem

3.1.1 Reservation Management, Data Access Utilities

insertReservation(reservation)

input	*effect*
Null parameter	NullArgumentException
Reservation with an Id already existent in the database	InvalidArgumentValueException
Valid Argument	A tuple containing reservation data is inserted into the database

deleteReservation(reservation)

input	*effect*
Null Parameter	NullArgumentException
reservation with an inexistent id	nvalidArgumentValueException
Valid Argument	The tuple containing reservation data is deleted from the database

getReservationList()

input	*effect*
Nothing	The list of all the reservation on going

3.1.2 Ride Management, Data Access Utilities

endRide(ride)

input	*effect*
Null Parameter	NullArgumentException
RideId not found into the database	InvalidArgumentValueException
Valid Argument	the tuple in the rideList is deleted

insertRide(ride)

input	*effect*
Null parameter	NullArgumentException
Array with NULL fields	InvalidArgumentValueException
Array with empty fields	InvalidArgumentValueException
All the parameters inside ride tuple are valid	Entry is inserted inserted into the database

getRideList()

input	*effect*
Nothing	Returns a list with all the on-going rides

getPassengers(ride)

input	*effect*
Null parameter	NullArgumentException
The rideId is not found in the database	InvalidArgumentValueException
Valid Argument	Returns the number of passengers inside the car when the ride ends

3.1.3 Issue Management, Data Access Utilities

getCarStatus(carId)

input	*effect*
Null parameter	NullArgumentException
The carId is not found in the database	InvalidArgumentValueException
The carId parameter is valid	Returns the status of the car corresponding to the Id inserted

getAccidentCarList()

input	*effect*
Nothing	Returns a list with all the cars implicated in an accident

getLowChargeCarList()

input	*effect*
Nothing	Returns a list with all the cars with no more than 20% of battery charge

getAssistanceNeededCarList()

input	*effect*
Nothing	Returns a list with all the cars that need assistance. It differs from accident because ?assistance needed? status is marked when an user find issues inside/outside the car when he reserve it

getIssuedCarList()

input	*effect*
Nothing	Returns a list with all the cars and their status

3.1.4 Car Management, Data Access Utilities

changeCarStatus(carId, newStatus)

input	*effect*
Null parameter	NullArgumentException
nvalid Status (the possible status are noIssues, LowCharge, Accident, AssistanceNeeded)	InvalidArgumentValueException
The new status is equal to the previous one	Status of the car does not change
Valid Argument	The status of the car is updated with new entry

getBatteryCharge(carId)

input	*effect*
Null parameter	NullArgumentException
The carId does not exist in the database	InvalidArgumentValueException
Valid Argument	Returns the percentage of the battery

insertCar(carId)

input	*effect*
Null parameter	NullArgumentException
Valid Argument	The car is uploaded into the database

deleteCar(carId)

input	*effect*
Null parameter	NullArgumentException
Valid Argument	The car is deleted from the database

isAvailableCar(carId)

input	*effect*
Null parameter	NullArgumentException
A car with an id inexistent into the database	InvalidArgumentValueException
Car available	returns true
Car unavailable	returns false

getAvailableCarList()

input	*effect*
Nothing	Returns the list of all the car available when the method is called

reportLowCharger(carId, userId)

input	*effect*
Null parameter	NullArgumentException
Valid Paramters	In the end of the ride the method ?changeStatusCar? is called with LowCharge as parameter

3.1.5 Reservation Management, Issues Management

reportAssistanceNeeded(carId, userId)

input	*effect*
Null parameter	NullArgumentException
Valid Parameters	The user reports a problem inside/outside the car and the system calls the method ?changeStatusCar? with ?assistanceNeeded? as parameter

3.1.6 Car Manegement, Location Management

isSafeArea(position)

input	*effect*
Null parameter	NullArgumentException
A position whose coordinates are invalid	InvalidArgumentValueException
A position outside the safe area	Returns false
A position inside the safe area	Returns True

getPosition(carId)

input	*effect*
Null parameter	NullArgumentException
The carId is not found in the database	InvalidArgumentValueException
CarId exist in the database	Return the current position of the car

3.1.7 Reservation Manegement, Location Management

getPosition()

input	*effect*
Null parameter	NullArgumentException
Valid Parameter	Return the current position of the car

3.1.8 Ride Management, Location Management

getPosition()

input	*effect*
Null parameter	NullArgumentException
Valid Parameter	Return the current position of the car

3.1.9 Reservation Management, Car Management

unlockReservedCar(userId)

input	*effect*
Null parameter	NullArgumentException
The userId (contactless card) and the Id of the user that reserve the car are not the same	InvalidArgumentValueException, the system do not unlock the doors
The userId (contactless card) and the Id of the user that reserve the car match	The system unlocks the doors of the car

3.1.10 Ride Management, Issues Management

reportAccident(carId, userId)

input	*effect*
Null parameter	NullArgumentException
Valid Parameters	The user reports the accident and the system calls the method ?changeStatusCar? with ?Accident? as parameter to mark car as unavailable

3.2 Administration subsystem

3.2.1 Operator Management, Data Access Utilities

insertOperator(operatorId)

input	*effect*
Null parameter	NullArgumentException
Valid Parameter	The operator data is inserted into the database

deleteOperator(operatorId)

input	*effect*
Null parameter	NullArgumentException
alid Parameter	The operator data is deleted from the database

operate(carId)

input	*effect*
Null parameter	NullArgumentException
The parameter has as car status "noIssues"	InvalidArgumentValueException
Valid argument, the car has at least one issue	The system marks that car as unavailable and the operator have to resolve the issue(s) on that car. The system adds the car to the list of cars that are being repaired

IssueSolved(carId)

input	*effect*
Null parameter	NullArgumentException
The car is not present inside the list of the car that are being repaired	InvalidArgumentValueException
Valid Argument	The car is repaired, its status get back to ?noIssues? and it is available to be reserved again

getoperatorList()

input	*effect*
Nothing	The list of all operators in the database

getStation(operator)

input	*effect*
Null parameter	NullArgumentException
Valid Argument	Returns the name of the station where the operator actually works

3.2.2 Station Management, Data Access Utilities

getStationPosition(stationId)

input	*effect*
Null parameter	NullArgumentException
The Station is not found in the database	InvalidArgumentValueException
The station is found in the database	Returns the position of the station

getOperatorsList(stationId)

input	*effect*
Null parameter	NullArgumentException
The Station is not found in the database	InvalidArgumentValueException
The station is found in the database	Returns a list of all the operators that work in that station

insertStation(station)

input	*effect*
Null parameter	NullArgumentException
Invalid or null arguments or the station already exists in the database	InvalidArgumentValueException
valid argument	The station data are uploaded into the database

deleteStation()

input	*effect*
Null parameter	NullArgumentException
The station is not found in the database	InvalidArgumentValueException
The station is found in the database	The station data of that station are deleted

3.2.3 Payment Management, Data Access Utilities

SendRidePaymentInformation(price, creditCardNumber)

input	*effect*
Null parameter	NullArgumentException
Price is negative	InvalidArgumentValueException
Valid Argument	All the information are sent to the Payment System in order to pay the ride

3.2.4 Discounts&Penalities Management, Data Access Utilities

apply(percentage)

input	*effect*
Null parameter	NullArgumentException
Valid Argument	The ride cost is modified adding the percentage of the ride

3.3 Account Management subsystem

3.3.1 Login Management, Data Access Utilities

checkPassword(userId, password)

input	*effect*
Null parameter	NullArgumentException
userId is inexistent in the database	InvalidArgumentValueException
Not valid combination of userId and password	InvalidCredentialsException
Valid combination of parameters	Returns user home page

3.3.2 Password Retrieval, Data Access Utilities

forgotPassword(userid, email)

input	*effect*
Null parameter	NullArgumentException
userId is inexistent in the database	InvalidArgumentValueException
Not valid combination of userId and email	InvalidCredentialsException
Valid combination of parameters	An email is sent to the user in order to change his password

3.3.3 Registration, Data Access Utilities

insertUser(user)

input	*effect*
Null parameter	NullArgumentException
Valid user	The user data is inserted into the database

deleteUser(userId)

input	*effect*
Null parameter	NullArgumentException
Valid user	The user data is deleted from the database

3.3.4 Settings, Data Access Utilities

updateAccount(userId, userData)

input	*effect*
Null parameter	NullArgumentException
UserId not found in the database or not Valid Data	InvalidArgumentValueException
Valid user and data	The user data is updated in the database

3.4 Integration among subsystems

3.4.1 Remote Services Interface, Car Sharing Subsystem

reserveCar(carId, userId)

input	*effect*
Null parameter	NullArgumentException
not valid userId	InvalidArgumentValueException
Not valid carId	InvalidArgumentValueException
Valid userId and car	the system reserves the car for the user

startRide(userId, carId)

input	*effect*
Null parameter	NullArgumentException
Not valid userId	InvalidArgumentValueException
Not valid carId	InvalidArgumentValueException
Not valid reservation associated to the user	InvalidArgumentValueException
Valid user and car identifiers, valid reservation	the reservation is cancelled and a new ride is created

endRide(rideId)

input	*effect*
Null parameter	NullArgumentException
Not valid rideId	InvalidArgumentValueException
Valid rideId	The system ends the ride

3.4.2 Remote Services Interface, Account Manager

insertUser(user)

input	*effect*
Null parameter	NullArgumentException
not valid/ already existent user	InvalidArgumentValueException
Valid user	The user data is inserted into the database and the registration confirmation is sent to him

deleteUser(userId)

input	*effect*
Null parameter	NullArgumentException
The userId does not exist in the database	InvalidArgumentValueException
Th userId is found in the database	The user data is removed from the database

3.4.3 Remote Services Interface, Administration Subsystem

insertoperator(operator)

input	*effect*
Null parameter	NullArgumentException
not valid/ already existent operator	InvalidArgumentValueException
Valid operator	The operator data is inserted into the database and the registration confirmation is sent to him

deleteOperator(operatorId)

input	*effect*
Null parameter	NullArgumentException
The operatorId does not exist in the database	InvalidArgumentValueException
The operatorid is found in the database	The operator data is removed from the database

4 Tools and Test Equipment Required

4.1 Tools

In this section we are going to describe the automated testing tools needed to accurately test all the components of the Power&Joy system. As concerns the business logics inside the JEE environment, we are going to use mainly three tools:

Junit framework : is a tool mostly used for unit testing, but still valid for some integration testing activities such as verifying that invoked methods return correct objects, that the right exceptions are raised when certain selected parameters are sent as inputs to them , and other issues due to the interaction among the components.

Mockito : is a tool really useful in order to provide functional dependencies abstraction, by means of "mocking" not yet implemented classes, parts of code and external components, in order to test the interaction with other components of the system and provide predictable results.

Arquillan framework : this tool is used to define containers (e.g. .jar files) to check that the the interaction between components and their surrounding environment happens correctly, that the connections with the database correctly work and that the right components are injected during the execution of each functionality of the system.

We also need some tool to analyse and test the performance of the system. To do that we use Jmeter, which is a tool built in Java that can be run on any environment, and provides functionalities to simulate the usage of any function provided by the system (such as logging, filling forms, etc.) from any number of users, thanks to the creation of threads. The results of the tests are available to be analysed in terms of throughput provided against the number of simulated requests.
In addition to the tools described above, it could be necessary to also have a significant amount of manual testing work in order to have a complete coverage.

4.2 Test equipment

In this section we are going to describe the testing environment in which all the integration tests are performed, and the characteristics of the devices that have to be used, both as concerns clients and backend infrastructure. For what concerns the mobile side of the testing environment, the following devices are required:

- At least a smartphone and a tablet running Android 4.0 or higher
- At least a smartphone and a tablet running iOS 8.0
- at Least a smartphone running Windows Phone

These devices will be used to test both the native mobile applications and the mobile versions of the web applications. Regarding the desktop web applications, they will be tested using a set of normal desktop and notebook computers. There are no specific requirements on display resolution, operating system and processing power. As for the backend testing, the business logic components should be de- ployed on a cloud infrastructure that closely mimics the one that will be used in the operating environment. In particular the testing cloud infrastructure needs to run the same operating system, the same Java Enterprise Application Server, Notification System and Remote Services Interface middleware and the same DBMS. There are any available cloud infrastructures on the market; as a guideline we assume we are going to use the Red Hat OpenShift cloud infrastructure, which provide all the tools we need such as:

- The Red Had Enterprise Linux distribution
- The Java Enterprise Edition runtime
- he GlassFish Java Application Server
- The GlassFish Message Broker
- The Apache Web Server as an HTTP load balancer
- The Oracle Database Management System.

5 Program Stubs and Test Data Required

5.1 Program stubs and Drivers

In this section we are going to describe the drivers needed to perform the integration tests described above. This is due to the choice of bottom-up approach strategy used to perform all the tests, as mentioned in the integration strategy section (2.3). These drivers are going to perform the actual methods invocations of all the subcomponents, according to the Junit framework.

Data Access Drtiver: this testing module will invoke the methods exposed by the Data Access Utilities component in order to test its interaction with the DBMS

Ride Management Driver: this testing module will invoke the methods exposed by the Ride Management Subcomponent in order to test its interaction with the Location Management, Car Management , Data Access utilities, and the Notification system

Reservation Management Driver: this testing module will invoke the methods exposed by the Reservation Management Subcomponent in order to test its interaction with the Location Management, Car Management, Data Access Utilities, Notification System

Car Management Driver: this testing module will invoke the methods exposed by the Car Management Subcomponent in order to test its interaction with the Data Access Utilities, Notification System, Mapping Service, Issues Management

Issues Management Driver: this testing module will invoke the methods exposed by the Issues Management Subcomponent in order to test its interaction with the Ride Management, Data Access utilities, Notification system

Location Management Driver: this testing module will invoke the methods exposed by the Location Management Subcomponent in order to test its interaction with the Mapping Service

Car Sharing Driver this testing module will invoke the methods exposed by the Location Management Subcomponent in order to test its interaction with the Mapping Service, DataAccess Utilities, Notification System

Account Management Driver: this testing module will invoke the methods exposed by the Account Management subsystem to test its interactions with the Data Access Utilities and the Notification System components

System Administration Driver: this testing module will invoke the methods exposed by the System Administration subsystem to test its interactions with the Data Access Utilities and the Notification System components.

The bottom-up approach in principle r does not require any stub for the integration testing. Anyway, since we start developing the system from the backend components, we are going to use some of them to simulate the functionalities (the requests) from the clients in order to make sure that the interactions between them and the core of the system correctly work.

5.2 Test Data

in this section we give some guidelines over the data needed to perform all the tests described in this document:

- A list of both valid and invalid User and Operator instances to test the Registration and Login subcomponents of the Account Manager Subsystem. The set should contain instances exhibiting the following problems:
  - Null / empty fields
  - null objects
  - Personal details not compliant with the regular format (age, fiscal code, ecc.)
  - Driving license not compliant with the legal format
- A list of both valid and invalid Cars instances to test the Car Management subcomponent. The set should contain instances exhibiting the following problems:
  - Null / empty fields
  - null objects
  - Invalid CarId
  - invalid position
- A list of both valid and invalid Reservations instances to test the Reservation management subcomponent. The set should contain instances exhibiting the following problems:
  - Null / empty fields
  - null objects
  - Invalid ReservationId
  - Invalid /already existent User/car identifiers
- A list of both valid and invalid Rides instances to test the Ride management subcomponent. The set should contain instances exhibiting the following problems:
  - Null / empty fields

  - null objects
  - Invalid RideId
  - Invalid User/length/final price/passengers parameters

- A list of both valid and invalid Issues instances to test the Issues management subcomponent. The set should contain instances exhibiting the following problems:
  - Null / empty fields
  - null objects
  - Invalid Issue identifier
  - Invalid User/Ride/Reservation identifiers

6 Effort Spent

To redact this document, we spent 30 hours per person.