Politecnico di Milano

Software Engineering 2 Project Work

PowerEnJoy Project: Integration Test Plan Document

Version: 1.0

Autohrs:

Jacopo Fantin (mat. 878723)

Francesco Larghi (mat. 876928)

Professors:

Elisabetta Di Nitto

Luca Mottola

January 15, 2017

Contents

1. Introduction

1.1 Revision History

Version	Date	Author(s)	Summary
1.0	15/01/2017	Francesco Larghi - Jacopo Fantin	First release

1.2 Purpose and Scope

This is the Integration Testing Plan Document for PowerEnJoy, the project we are documenting. Integration Testing is a key activity to test that all the different subsystems interoperate consistently with the requirements they are supposed to fulfill and without exhibiting unexpected behaviors.
There is an ideal specific order in witch different components should be tested, in order to guarantee the consistence of the whole system.

The purpose of this document is to outline, in a clear and comprehensive way, the main aspects concerning the organization of the integration testing activity for all the components in the Business Tier. Firstly we will specify the criteria that must be met by the project before testing, then our integration testing strategy and the sequence for a correct integration.
For each component it will be specified also the expected behaviour of each critical function after a specific input. Finally, you can find some considerations on the performances of the system, tools and equipment required and other stuff.

1.3 List of Definitions and Abbreviations

- **DD:** Design Document.
- **RASD:** Reqirements Analysis Specifications Document.
- **DB:** DataBase.
- **DBMS:** DataBase Management System.
- **Component:** Software element that implements functionalities.
- **Java EE:** Java Enterprise Edition 7, a widely used computing platform for web based software like PowerEnJoy
- **EJB:** Enterprise Java Beans. Component in the Business Tier for the Application Logic.
- **JDBC:** Java Database Connectivity, Java API to connect to DataBases.
- **JPA:** Java Persistence API.
- **AVDM:** Android Virtual Device Manager, tool fot the virtual excecution of Android operating system.

1.4 List of Reference Documents

– Our PowerEnjoy Requirements Analysis Specifications Document

– Our PowerEnjoy Design Document

– The specification document: Assignments AA'16-'17.pdf

– Examples of Integration Testing Plan Document available

2. Integration Strategy

2.1 Entry Criteria

Beginning with specifying the procedure for our test phase, we define the conditions that have to be satisfied before the integration starts.

- Documents such as RASD and SDD must have been completed, so that a full idea about the project is given to the testers.
- The Database of the system must be initialized with some first data, as much as suffices for checking the correctness of the basic operations on each table.
- The classes have to be implemented at least with the main methods; anyway, the code should have been written for the most part.
- Each software component is required to be unit tested before the integration phase starts.

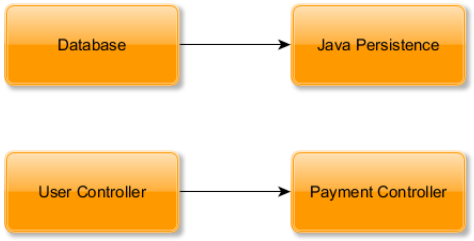
2.2 Elements to be Integrated

For the first version of our integration process, we consider only one part of the system components we specified in the Software Design Document, namely the components that are comprised in the Business Tier. We chose to limit the modules to be tested to those in this software layer, at first, as they represent the actual functionalities of the PowerEnJoy service (see the Software Design Document for more details about the single components), thus they constitute a reasonable starting point for the testing.

2.3 Integration Testing Strategy

Integration will be based on a bottom-up strategy, within a Critical-Module-First in case of modules laying on the same layer.

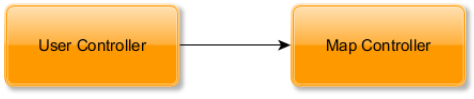
The bottom-up approach ensure us to start integrating components that don't need any other component to be tested, to then go on with integrating those who require the effectiveness of others. The Critical-Module-First will come in handy to solve dependencies soon during the process and preventing the process itself to be blocked by errors on critical components.



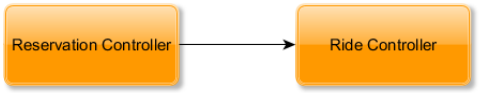
2.4 Sequence of Component/Function Integration

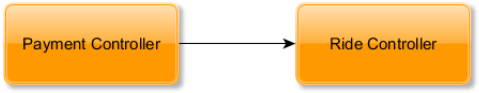
2.5 Software Integration Sequence

2.6 Subsystem Integration Sequence




User Controller
Reservation Controller
Car Controller


Reservation Controller
Ride Controller


Payment Controller
Ride Controller

3. Individual Steps and Test Description

In this chapter we'll provide a detailed description of the tests to be performed on integrated components. We will present possible inputs and their effects in the components' functions, to secure the correctness of the behaviour with respect to invalid input. Then it will be possible to test if each of the component involved works as expected or if not. We consider JPA as a standard and external component, so it doesn't need to be specified right now. We assume that all of these components have to be tested with JPA.

As we said before, we have decided to test only th components of the Business Tier because it's the most critical level of our system, in which the most of the logic and functionalities are implemented. In every section the first component is the one that have to be tested, the eventual second and third one are the components that must have already been tested before because they are integrated in it.
Integration testing should be done in the following order.

3.1 Guest Controller

loginRequest(userID, password)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments, but wrong credentials	Returns False
Valid arguments and credentials	Returns True and create a session for the User

registrationRequest(userID, email, password, name, creditCardNumber, drivingLicence)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments	Returns True

3.2 PowerGridStation Controller

changeStatus(powerGridStationID, status)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments	Returns True

3.3 Car Controller

updatePosition(vehiclePlate, position)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments	Returns True and update in the DB position data

updateStatus(vehiclePlate, status)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments	Returns True and change in DB the car status (Available, Unavailable, Reserved, Used)

updateBattery(vehiclePlate, value)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments not(0<value<100)	Raises a InvalidArgumentException
Valid arguments	Returns True and change the value in DB

unlockCar(vehiclePlate)	
Input	*Effect*
Null	Raises a NullArgumentException
Valid reserved Car	Returns True

lockCar(vehiclePlate)	
Input	*Effect*
Null	Raises a NullArgumentException
Valid reserved Car	Returns True

plugInto(vehiclePlate, powerGridStationID)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments	Returns True and change the value in DB

checkPlugged(vehiclePlate)	
Input	*Effect*
Null	Raises a NullArgumentException
Valid arguments but not plugged	Returns False
Valid arguments and plugged	Returns True

3.4 User Controller

logoutRequest(userID)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments and credentials	Returns True and cancel the session of the User

updateData(userID, email, password, name, creditCardNumber, drivingLicence)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments	Returns True

3.5 Payment Controller - User Controller

newPayment(moneyAmount, userID)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments	Returns entity Payment created

paymentRequest(payment)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments but something goes wrong in the payment procedure	Raises a InvalidPaymentException
Valid arguments and successful payment	Returns true

3.6 Map controller - User Controller

showNearUserCars(userID)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments but invalid position	Raises a InvalidPositionException
Valid arguments and user position	Launch an algorythm to find all near available cars and relative positions. Returns a Car List

showNearAdressCars(position)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments but invalid position	Raises a InvalidPositionException
Valid arguments and position	Launch an algorythm to find all near available cars and relative positions. Returns a Car List

showNearPowerGridStations(position)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments but invalid position	Raises a InvalidPositionException
Valid arguments and position	Launch an algorythm to find all near available PowerGridStations and relative positions. Returns a PowerGridStation List

3.7 Reservation Controller - User Controller - Car Controller

newReservation(userID, vehiclePlate)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments but invalid position	Raises a InvalidPositionException
Valid arguments but Car is not Available	Raises a NotAvailableCarException
Valid arguments but User has still another Reservation	Raises a NotAvailableReservationException
Valid arguments and successfull	Create a new Reservation and insert it in the DB. Return the reservDate to start the 1 hour timer.

deleteReservation(userID, vehiclePlate, reservDate)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments and successfull	Delete the Reservation from the DB. Return True

newTimer(userID, vehiclePlate, reservDate)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments and successfull	Create a 1 hour timer. If the timer expires, launch deleteReservation. Return True

stopTimer(userID, vehiclePlate, reservDate)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments and successfull	Stops the timer of the Reservation. Return True

3.8 Ride Controller - Reservation Controller

newRide(userID, vehiclePlate, reservDate)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments and successfull	Create a new Ride and insert it in the DB. Return rideDate.

updateRideStatus(userID, vehiclePlate, reservDate, rideDate, status)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments and successfull	Update Ride status and insert it in the DB. Return true.

updateRideKm(userID, vehiclePlate, reservDate, rideDate, km)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments and successfull	Update Ride km and insert it in the DB. Return true.

upDiscAndFees(userID, vehiclePlate, reservDate, rideDate, battDisc, passDisc, pwrDisc, fee)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments and successfull	Update all discounts and fees and insert it in the DB. Return true.

finishRide(userID, vehiclePlate, reservationDate, rideDate)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments and successfull	Insert the finalDate of the Ride in the DB and stops Ride and Reservation. Return true.

3.9 Ride Controller - Payment Controller

billCalculus(userID, vehiclePlate, reservationDate, rideDate)	
Input	*Effect*
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments and successfull	Wait 10 minutes, then calculate the final bill with discounts and fees, finally request the payment

4. Performance analysis

Of course an actual performance analysis of the PowerEnJoy system will be executed only in the system integration phase, but it could be still useful to perform some preliminary considerations. In particular, it is appropriate to verify that the applications runs for all the target mobile platforms and specifications, that the API are compatible with the highest number possible of devices.
Furthermore, the storage occupation should be reasonably small, we can excpet less than 50MB of memory. However, this number should be reconsidered during the development phase taking into account the improvements into smartphone and tablet technology that may occur meanwhile.
We will specify all the proper softwares and tools to analyze our performances in the next chapter.

5. Tools and Test Equipment Required

We think that in order to test the components of our system we are going to make usage of some automated testing tools. For what concerns the client side running in Android, we will use **Android Studio** not only to develop but also to test our applications thanks to simulation and virtualization of Android operating system through Android Virtual Device Manager (AVDM). Thanks to this powerful tool it will be possible for us to test virtually the application with a lot of different smartphones and tablets without actually have them. Moreover, we could also consider the possibility of performing an analysis of the smartphone market to identify the most common display sizes and resolutions right before starting the integration testing phase. For what regards the desktop web application it will be enough to test it with different desktops and browsers.

Moreover we can also use **JUnit** to test properly Java Classes and we can use it in business logic tier too. The JUnit Framework is not only useful to single unit testing, it's also a valid instrument to verify interactions between different components and their outputs or expected results. For example it is useful to verify the correct raise of Java exception.

Another very useful software is **Arquillan**. It is an integration testing software used to execute test cases against the container in order to check that the interaction between a component and its surrounding execution environment is happening correctly. We are going to use this tool to verify that the right components are injected when dependency injection is specifed, that the connections with the database are properly managed and similar container level tests. It looks just like a JUnit test, but with some more functionalities. Of course this framework is compatible with JEE containers.

We think also to use a tool to measure effective performances like **Jmeter**. It allows us to load test behavior and measure performance. It can be used to simulate a heavy load on a server, network, or object, to test its strength or to analyze overall performance under different load situations. This can be a good way to verify the scalability of our application for a large number of user.

6. Program Stubs and Test Data Required

7. Appendix

7.1 Used Tools

- TeXMaker: to create this pdf document
- yEd Graph Editor: to create graphs

7.2 Working Hours

Last Name	First Name	Total Hours
Larghi	Francesco	10 h
Fantin	Jacopo	? h