



POLITECNICO DI MILANO

SOFTWARE ENGINEERING 2 PROJECT WORK

PowerEnJoy Project: Integration Test Plan Document

Version: 1.0

Authors:

Jacopo FANTIN (mat. 878723)

Francesco LARGHI (mat. 876928)

Professors:

Elisabetta DI NITTO

Luca MOTTOLA

January 16, 2017

Contents

1	Introduction	1
1.1	Revision History	1
1.2	Purpose and Scope	1
1.3	List of Definitions and Abbreviations	2
1.4	List of Reference Documents	2
2	Integration Strategy	3
2.1	Entry Criteria	3
2.2	Elements to be Integrated	3
2.3	Integration Testing Strategy	3
2.4	Sequence of Component/Function Integration	4
2.4.1	Software Integration Sequence	4
2.4.2	Subsystem Integration Sequence	8
3	Individual Steps and Test Description	9
3.1	Guest Controller	10
3.2	PowerGridStation Controller	10
3.3	Car Controller	11
3.4	User Controller	13
3.5	Payment Controller - User Controller	14
3.6	Map controller - User Controller	15
3.7	Reservation Controller - User Controller - Car Controller	16
3.8	Ride Controller - Reservation Controller	18
3.9	Ride Controller - Payment Controller	19
4	Performance analysis	21
5	Tools and Test Equipment Required	23
6	Program Stubs and Test Data Required	25
6.0.1	Program Stubs	25
6.0.2	Test Data	25
7	Appendix	27
7.1	Used Tools	27
7.2	Working Hours	27

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 which 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:** Requirements 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 for the virtual execution 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, i.e. that do not require interaction with external components for the test purpose, 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

This section will run through the ordered sequence of software components that is going to be employed for the integration of the modules of interest for this first approach to the software testing. The component at the arrow tail is the module which is to be taken for granted as already working, whether because it's an external component or because it's already been tested, while the component at the arrow head is the module we're willing to test.

2.4.1 Software Integration Sequence

As we specified, following the bottom-up sequence, we will first test the interaction operations between the database and the Java Persistence API. This means that the data populating the database will be used to create the tables (classes) of the JPA component, and basic operation will be performed on these data to check whether the two components are synchronized.



Figure 2.4.1: Integration of the database with JPA.

We then move on integrating each module with the JPA component, that corresponds to verify that communication between each single module and the interface to the database works properly.

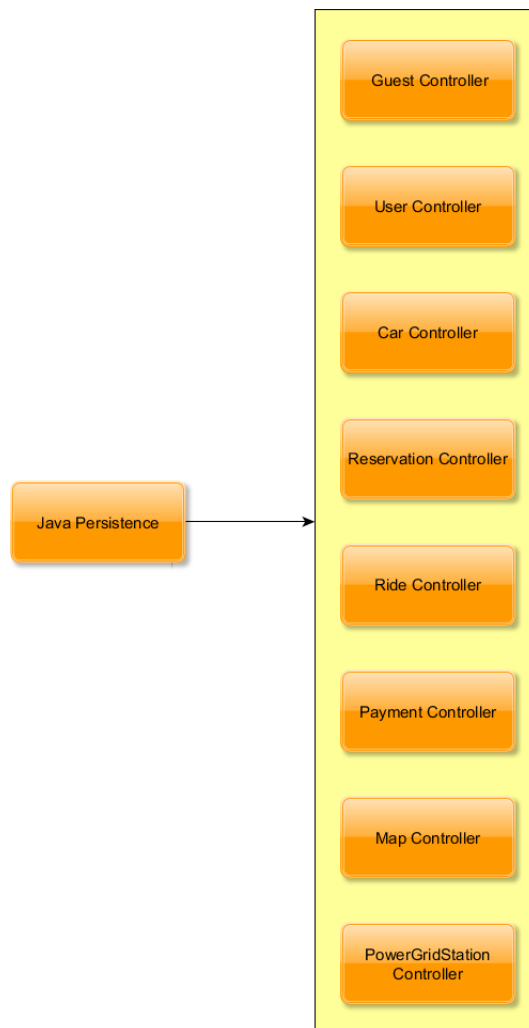


Figure 2.4.2: The Business Tier components integrated with the JPA component.

Finally, the integration goes on with the remaining modules, which need other components to be tested.

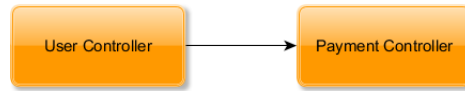


Figure 2.4.3: Integration of the Payment Controller with User Controller.



Figure 2.4.4: Integration of the Map Controller with User Controller.

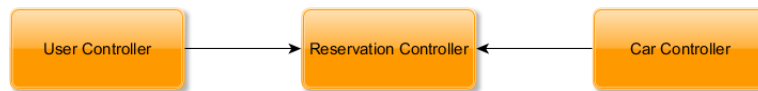


Figure 2.4.5: Integration of the Reservation Controller with User and Car Controllers.



Figure 2.4.6: Integration of the Ride Controller with Reservation Controller.



Figure 2.4.7: Integration of the Ride Controller with Payment Controller.

2.4.2 Subsystem Integration Sequence

The following scheme shows all the components we considered for the test phase and the relations exposed in the previous subsection.

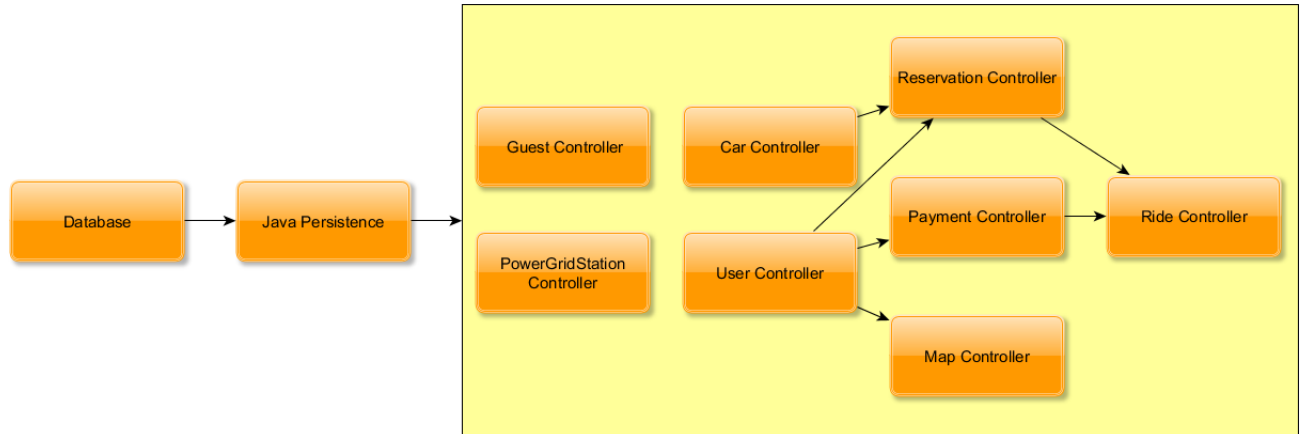


Figure 2.4.8: Integration sequence for the modules considered in our test phase.

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 the 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

<i>loginRequest(userID, password)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullPointerException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments, but wrong credentials	Returns False
Valid arguments and credentials	Returns True and creates a session for the User

<i>registrationRequest(userID, email, password, name, creditCardNumber, drivingLicence)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullPointerException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments	Returns True

3.2 PowerGridStation Controller

<i>changeStatus(powerGridStationID, status)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullPointerException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments	Returns True

3.3 Car Controller

<i>updatePosition(vehiclePlate, position)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments	Returns True and updates in the DB position data

<i>updateStatus(vehiclePlate, status)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments	Returns True and changes in DB the car status (Available, Unavailable, Reserved, Used)

<i>updateBattery(vehiclePlate, value)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullArgumentException
Invalid arguments not($0 < \text{value} < 100$)	Raises a InvalidArgumentException
Valid arguments	Returns True and changes the value in DB

<i>unlockCar(vehiclePlate)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullArgumentException
Valid reserved Car	Returns True

<i>lockCar(vehiclePlate)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullArgumentException
Valid reserved Car	Returns True

<i>plugInto(vehiclePlate, powerGridStationID)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments	Returns True and changes the value in DB

<i>checkPlugged(vehiclePlate)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullArgumentException
Valid arguments but not plugged	Returns False
Valid arguments and plugged	Returns True

3.4 User Controller

<i>logoutRequest(userID)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments and credentials	Returns True and cancels the session of the User

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

3.5 Payment Controller - User Controller

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

<i>paymentRequest(payment)</i>	
<i>Input</i>	<i>Effect</i>
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

<i>showNearUserCars(userID)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments but invalid position	Raises a InvalidPositionException
Valid arguments and user position	Launches an algorithym to find all near available cars and relative positions. Returns a Car List

<i>showNearAdressCars(position)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments but invalid position	Raises a InvalidPositionException
Valid arguments and position	Launches an algorithym to find all near available cars and relative positions. Returns a Car List

<i>showNearPowerGridStations(position)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments but invalid position	Raises a InvalidPositionException
Valid arguments and position	Launches an algorithym to find all near available PowerGridStations and relative positions. Returns a PowerGridStation List

3.7 Reservation Controller - User Controller - Car Controller

<i>newReservation(userID, vehiclePlate)</i>	
<i>Input</i>	<i>Effect</i>
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	Creates a new Reservation and inserts it in the DB. Returns the reservDate to start the 1 hour timer.

<i>deleteReservation(userID, vehiclePlate, reservDate)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments and successfull	Deletes the Reservation from the DB. Returns True

<i>newTimer(userID, vehiclePlate, reservDate)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments and successfull	Creates a 1 hour timer. If the timer expires, launches deleteReservation. Returns True

<i>stopTimer(userID, vehiclePlate, reservDate)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments and successfull	Stops the timer of the Reservation. Returns True

3.8 Ride Controller - Reservation Controller

<i>newRide(userID, vehiclePlate, reservDate)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullPointerException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments and successfull	Creates a new Ride and inserts it in the DB. Returns rideDate.

<i>updateRideStatus(userID, vehiclePlate, reservDate, rideDate, status)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullPointerException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments and successfull	Updates Ride status and inserts it in the DB. Returns true.

<i>updateRideKm(userID, vehiclePlate, reservDate, rideDate, km)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullPointerException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments and successfull	Updates Ride km and inserts it in the DB. Returns true.

<i>upDiscAndFees(userID, vehiclePlate, reservDate, rideDate, battDisc, passDisc, pwrDisc, fee)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments and successfull	Updates all discounts and fees and inserts it in the DB. Returns true.

<i>finishRide(userID, vehiclePlate, reservationDate, rideDate)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments and successfull	Inserts the finalDate of the Ride in the DB and stops Ride and Reservation. Returns true.

3.9 Ride Controller - Payment Controller

<i>billCalculus(userID, vehiclePlate, reservationDate, rideDate)</i>	
<i>Input</i>	<i>Effect</i>
Null	Raises a NullArgumentException
Invalid arguments	Raises a InvalidArgumentException
Valid arguments and successfull	Waits 10 minutes, then calculates the final bill with discounts and fees, finally requests 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 **Arquillian**. 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 specified, 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

6.1 Program Stubs

The test phase makes use of driver programs to achieve its aim. These drivers are the software that is actually going to invoke the functions arranged in the Individual Steps and Test Description section of this document on the components we decided to test. For this reason, the drivers we need to be implemented are:

- the Guest Driver, which simulates the account creation process through the registration operation and then attempts to log in the system with the account that it has just created;
- the User Driver, which simulates the behaviour of a registered user of PowerEnJoy trying functions like modifying its personal data in the reserved area, searching for cars and power grid stations in the neighborhood, reserving one of the displayed cars and deleting the just performed reservation.

6.2 Data

This paragraph provides the main information that should constitute a problem or an invalid argument for the functions we previously specified, and thus should be included in the test data set in the system database among valid data that should be normally computed. In particular:

- Invalid User information:
 - Null objects
 - Null fields
 - Wrong driving license format
 - Wrong credit card number format
 - Wrong e-mail address format
 - Wrong format for other personal information (name, date of birth, address, ...)

- Invalid Car information:
 - Null objects
 - Null fields
 - Wrong car plate format
- Invalid location information for the Power Grid Stations and the Safe Area Boundaries;
- Invalid status for the Ride entity.

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	6 h