Software Engineering 2: PowerEnjoy Integration Test Plan Document

Andrea Pace, Lorenzo Petrangeli, Tommaso Paulon

February 1, 2017

Contents

1 Introduction

1.1 Revision History

v1.1 better specification of I8 integration test

1.2 Purpose and Scope

This document will define a way to accomplish the integration test of the PowerEnjoy project.
We will start by defining an integration strategy and how the components are going to be integrated.
Then we will show a series of test cases for the most important functionalities of the system and the tools to be used.

1.3 List of Reference Documents

- Design Document of the PowerEnjoy project
- Requirements Analysis and Specification Document of the PowerEnjoy project
- Assignments AA 2016-2017

2 Integration Strategy

2.1 Entry Criteria

This section highlights the conditions to be met before the integration test phase.

It is required that all the components described in the Design Document have been tested through unit tests. This way we can focus on the interactions between components.

2.2 Elements to be Integrated

The component of our system are described in the Design Document. We can classify them referring to the MVC pattern:

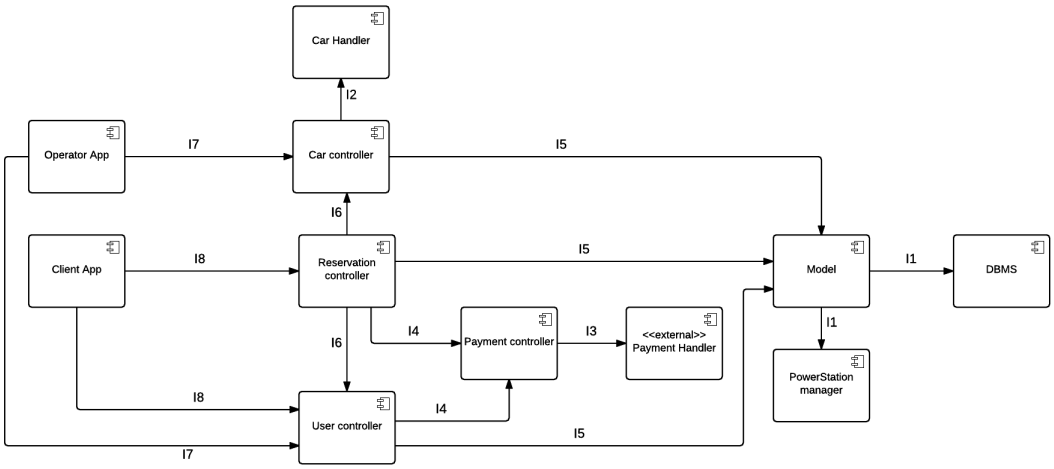
- Model: Database, Model component
- Controller: Reservation controller, Car controller, User controller, Payment controller, PowerStation manager, Car handler
- View: Client app, Operator app, Web browser

2.3 Integration Testing Strategy

Since we assume that every component has been tested, a bottom-up approach will be used: tests will be easier to write and results will be easier to observe. This approach will also require some drivers in order to properly call methods of components' interfaces.

Components with less dependencies will be tested first and we will proceed from the server to the client.

2.4 Sequence of Component/Function Integration



The order of integration will be chosen in order to minimize the number of stubs and driver required. A component will be tested as soon as its dependencies have been integrated, according to our strategy.

2.4.1 Software Integration Sequence

Our system is simple enough to be seen as a unique subsystem composed of the whole set of components described in the Design Document. We start from the server components that can be seen as the bottom of the system. Eventually the client-side components will be the tested. The reason to do so is that the server components are required in order to have a working client, while the server can be tested using an appropriate driver which simulates the client API calls.
The are actually two cycles: the Car Handler can send to the Car Controller the data required to calculate the final cost or a notification of problems without been called by the Car Controller. The same happens between the Car Controller and the Reservation Controller in the same process. This interations will be tested after the opposite ones so that the order of integration is preserved.

N.	Component	Integrates with
I1	Model	DBMS, PowerStation manager

I2	Car controller	Car Handler
I3	Payment Controller	Payment Handler
I4	Reservation Controller, User controller	Payment Controller
I5	Car controller, Reservation Controller, User controller	Model
I6	Reservation controller	Car Controller, User Controller
I7	Operator App	Car Controller, User Controller
I8	Client App	Reservation Controller, User Controller

3 Individual Steps and Test Description

This chapter describes the test cases that will be executed. The notation IxTy means that we are showing test number y with respect to the more general integration test case x.

3.1 Integration test I1

Test Case Identifier	I1T1
Test Item(s)	Model -> DBMS
Input Specification	Typical calls to the DBMS APIs: insertions/deletions and queries
Output Specification	The DBMS should respond to insertions/deletions by updating the database if no integrity constraint is violated and to queries by returning the result of the query
Environmental Needs	A complete implementation of the database structure and APIs
Test Description	Some queries and requests of insertion/deletions will be sent through the DBMS APIs and compared to the expected outputs

Test Case Identifier	I1T2
Test Item(s)	Model-> PowerStation
Input Specification	Request of a list of the free plugs in each power station
Output Specification	The power station should a list with the actual number of free plugs in each power station
Environmental Needs	Deployment of the power station OS and manager and full implementation of the RESTful APIs
Test Description	For this test we need a stub of the power station manager, otherwise we must perform the test with real cars connected to the power stations and manually check if the number provided via the RESTful API are correct. We check that the output is equal to what is generated by the stub

3.2 Integration test I2

Test Case Identifier	I2T1
Test Item(s)	Car Controller-> Car Handler
Input Specification	The Car Controller should send lock and unlock requests to the Car Handler
Output Specification	The Car Handler should respond by locking or unlocking the car
Environmental Needs	Full imlementation of the RESTful APIs and deployment of the Car Handler onto the car
Test Description	Since we assume that the Car Handler component works we simply check that the right methods are called, without physically checking if the car really locks or unlocks

Test Case Identifier	I2T2
Test Item(s)	Car Handler -> Car Controller
Input Specification	Some set of data used to calculate the final cost of the travel
Output Specification	The data received by the Car Component should be compared with the one expected
Environmental Needs	I2T1 succeded
Test Description	A driver simulating the Car Handler is required in order to break the cycle: the final amount of money returned to the Car Controller will be compared to the one generated by the driver.

3.3 Integration test I3

Test Case Identifier	I3T1
Test Item(s)	Payment Controller->Payment Handler
Input Specification	The Payment Controller should send payment requests, both correct(real payment info) and incorrect
Output Specification	The payment handler should respond correctly with a confirmation or a notification of error.
Environmental Needs	None, since the APIs are provided by the Payment Handler
Test Description	We assume that the Payment Handler has something like a sandbox that allows to perform fake transactions. This way we can make as many payment requests as we want

3.4 Integration test I4

Test Case Identifier	I4T1
Test Item(s)	User Controller -> Payment Controller Reservation Controller -> Payment Controller
Input Specification	Payment requests are sent to the Payment Controller
Output Specification	The payment should be accepted if the informations are correct
Environmental Needs	I3T1 succeded
Test Description	Both components can send requests to the payment controller

3.5 Integration test I5

Test Case Identifier	I5T1
Test Item(s)	Car Controller -> Model
Input Specification	The Car Controller should send requests about the current state of a car and requests to set the state of a car
Output Specification	The database should provide the requested informations or store the new state of a car according to the request
Environmental Needs	I1T1 succeded
Test Description	

Test Case Identifier	I5T2
Test Item(s)	User Controller -> Model
Input Specification	Requests about insertions of data about a client: personal data, payment info, debts(also deletion for this one). Requests about the verification of data provided by the client: password. Request about clients' current reservations
Output Specification	The database should provide the requested informations or store/delete the new information about a client if they don't violate any integrity constraint
Environmental Needs	I1T1 succeded
Test Description	The User Controller should send all the type of requests listed above. Some of them should violate some constraint, for example a request of reservation from a client who has already reserved a car

Test Case Identifier	I5T3
Test Item(s)	Reservation Controller -> Model
Input Specification	Request about the number of free plugs in each power station
Output Specification	The number of free plugs in each power station must be correct
Environmental Needs	I1T1 succeded
Test Description	

3.6 Integration test I6

Test Case Identifier	I6T1
Test Item(s)	Reservation Controller -> Car controller
Input Specification	Request of reservation or unlocking
Output Specification	If all the conditions for a reservation are met the car must be set to reserved and the reservation must be related to the client in the database. An appropriate countdown should start. In case of unlocking the car should be unlocked
Environmental Needs	I2T1 and I5T1 succeded
Test Description	Some reservation inputs or a simple unlocking request are given to the car controller. Since these inputs are verified by the User Controller according to the Design Document we don't expect errors.

Test Case Identifier	I6T2
Test Item(s)	Reservation Controller -> Car controller
Input Specification	Request of reservation canceled or expired
Output Specification	In both cases the car should be set to available and the client should not be related to the car in the database. The car should also lock itself when there is nobody inside.
Environmental Needs	I5T1 succeded
Test Description	These request will be sent to the Car Controller and the state of the database will be compared to the expected output. Methods concerning the request of car locking are expected to be called

Test Case Identifier	I6T3
Test Item(s)	Car Controller -> Reservation Controller
Input Specification	All the data needed to calculate the final cost of the travel
Output Specification	The payment should be performed, the car should be set to available and not related to the client anymore.
Environmental Needs	I4T1 and I2T2 succeded
Test Description	A driver simulating the Car Controller is required in order to break this cyclic relation: the driver will submit to the Reservation Controller data about the travel and the final amount of money submitted to the Payment Handler will be compared to the expected result

Test Case Identifier	I6T4
Test Item(s)	Reservation Controller -> User Controller
Input Specification	Request to retrieve clients' data or to store informations(debts)
Output Specification	The requested listed above should be satisfied
Environmental Needs	I5T2
Test Description	

3.7 Integration test I7

Test Case Identifier	I7T1
Test Item(s)	Operator App -> Car Controller
Input Specification	Calls to the REST API requesting to modify the state of a car
Output Specification	The indicated car should be set to unavailable/available in the database.
Environmental Needs	I5T1
Test Description	

3.8 Integration test I8

This section describes the integration of the client App with the server. The functionality to be tested are exactly the same
listed in the RASD and described in the DD. They are:

- Login
- Registration
- Reservation
- Car Unlocking

- Delete reservation
- Change payment info
- Money saving option
- Estinguish debt

At this point of the integration test this last phase is pretty straightforward compared to the complexity of the interations between server components: a driver that substitutes the client makes proper API calls and the output must be compared with the expected result.

- Login: if the informations provided are correct the user must be able to perform some operations: clients can perform the operation already discussed and operators can change cars' state
- Registration: if the informations provided are consistent and do not overlap with previous data, the new user must be stored in the database
- Reservation: if the client associated with the request has no pending debt and no other reservations the selected car must be set to reserved in the database and the other user should be notified. To do so we need multiple client drivers logged in that must receive the notification about the new reservation.
- Car Unlocking: if the user associated with the request has really reserved a car and his position is close enough, the car must be set to reached in the database and actually unlocked
- Delete reservation: if the client associated with the requested matches the one who has reserved a car in the database and the car state is not "in use", the reservation is canceled and the client is charged of a fee. Therefore the client driver should receive a notification of debt and the debt itself must be stored in the database.
- Change payment info: we have to check if the new payment informations are consistent and if so they must be stored in the database at the end of the procedure
- Money saving options: if the car is not in the "in use" state the returned power station must be the one correctly calculated with the procedure highlighted in the design document
- Estinguish debt: if the payment information are correct and appropriate to estinguish the debt, it should be deleted from the database. This test requires the payment handler's sandbox to be provided with a certain number of payment informations. The test will be performed by submitting informations present or absent in the sandbox or correct information with unsifficient balance. The results should be the ones expected

3.9 Other tests

For what concerns the web application it is enough to create various HTTP requests of the functionalities provided by the application website, send them from the client browser and compare the results with the expected ones.

4 Program Stubs and Test Data Required

According to the bottom-up strategy we need several drivers to perform integration tests. We need a driver for each integration test, which has to be provided with the logic for the right method/API calls and the expected outputs. Some of the most relevant drivers are:

- Client driver: required in order to test the server components when the client is not fully developed. It is also necessary to test the APIs between the client and the server
- Car Handler and Car Controller driver: simulates the situations when a car notifies the system that the engine has been turned on or when the data concerning the payment are sent to the server

For what concerns test data we need the database structure shown in the Design Document(ER diagram) to be fully implemented and some user, car, power station and operator data stored in the database.

5 Tools and Test Equipment Required

We plan to use the following tools in order to accomplish the integration tests.

5.1 Arquillian

Aquillian is a test framework that we plan to use for integration testing. It allows the tests to be deployed on a server and it is extremely useful if the target of the tests is a JEE server.

5.2 JUnit

JUnit is normally used for unit tests, which are supposed to be produced before the integration tests. It can be used anyway with Arquillian for integration tests.

5.3 Mockito

Mockito is often used as a unit test tool to create mocks. Stubs are more general than mocks but Mockito can be useful along with JUnit to perform integration tests.

6 Hours of work

- Andrea Pace: 25 hours
- Lorenzo Petrangeli: 20 hours
- Tommaso Paulon: 15 hours