**Politecnico di Milano, A.Y. 2016/2017**
**M.Sc. Degree Programme in Computer Science and Engineering**
**Software Engineering 2 Project**

# Integration Test Plan Document

Philippe Scorsolini,
Lorenzo Semeria,
Gabriele Vanoni

15th January 2017

# Contents

# 1 Introduction

## 1.1 Purpose and scope

In this Integration Test Plan Document we are going to develop a detailed description of the path we are going to follow to achieve integration testing. Starting right after unit testing and component testing, this phase is the most crucial. Combining different components is not easy and so is testing them. In purticular in this phase we focus on interactions between components, starting from simple ones and arriving to more complex situations. In the document we highlight the rationale behind our testing choices i.e. our strategy and in which way we plan to write tests.

## 1.2 List of abbreviations

- GUI: graphical user interface
- PC: personal computer
- OS: operating system
- API: application programming interface
- IDE: integrated development environment
- DB: database

## 1.3 List of reference documents

- Project description “Assignments AA 2016-2017.pdf”
- Requirement analysis and specification document “rasd.pdf”
- Design document “dd.pdf”
- Arquillan reference guidehttps://docs.jboss.org/author/display/ARQ/Reference+Guide
- Mockito reference guidehttp://static.javadoc.io/org.mockito/mockito-core/2.5.5/org/mockito/Mockito.
- JUnit javaDochttp://junit.org/junit4/javadoc/latest/index.html
- AssertJ main pagehttp://joel-costigliola.github.io/assertj/index.html
- Pact documentation https://github.com/DiUS/pact-jvm/tree/master/pact-jvm-consumer-junit

# 2 Integration Strategy

This section is aimed at introducing the core aspects of the Testing Phase: the Entry Criteria – that must be verified before the testing starts –, the elements that must be integrated, and the chosen testing strategy.

## 2.1 Entry Criteria

Before actually doing integration tests it is important that some criteria are met. This helps ensuring that the results of the test are not skewed by other factors (e.g. a test fails because a function does not behave properly).
These are the conditions that must be met before integration testing:

- Dependencies among modules must all be defined
- All unit testing must have been successfully done
- Input and expected output/behaviour must be specified for each test case
- Parts of the system that are yet to be implemented must be stubbed
- The testing environment (better specified later) must be completed

## 2.2 Elements to be integrated

The elements that must be integrated in this phase are all the modules present in the Component Diagram already shown in the Design Document.

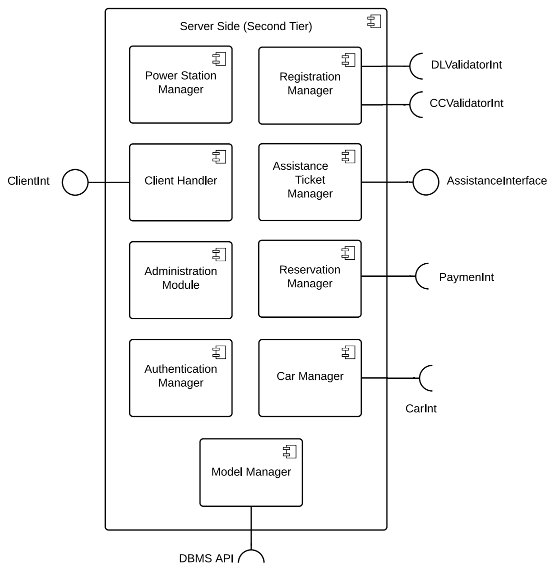


Figure 1: Component Diagram showing all modules.

From the above diagram it is clear that some modules rely or depend on others. These dependendencies are important to structure the testing, in particular the order in which the modules must be tested in. This aspect is better analyzed in other sections of this document.

## 2.3 Integration Strategy

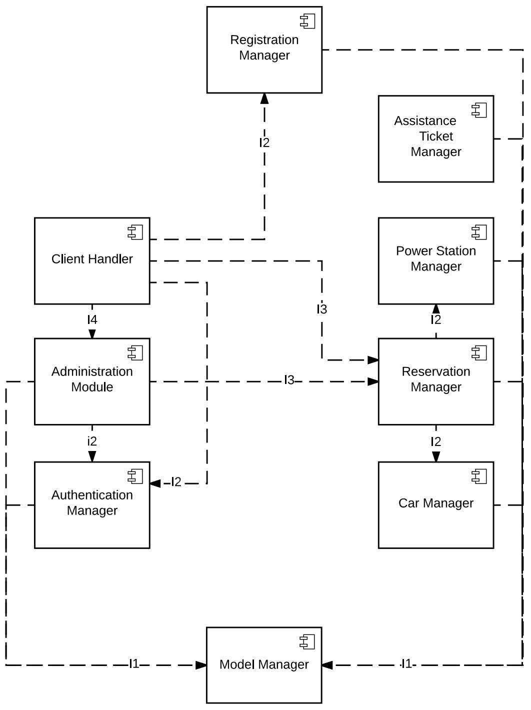
The Integration Strategy defines the order and the approach to testing. For this project, we have decided to use a bottom-up approach. Going bottom up means testing the components with the least dependencies first, thus avoiding the creation of any stub. On the other hand, this forces to create many drivers. Choosing this approach creates an order among components since those with no dependencies will be tested first, then gradually – as more components are tested – those that rely on them will be analyzed. Although ideally only drivers are created, it may become necessary to create some stubs or interfaces in order to break dependency cicles. Choosing a bottom up approach has many advantages:

- It is easier to create test cases.
- It is easier to check the behaviour of "simpler" (less integrated) components.
- Since many components rely on the functions of the "bottom" modules, it makes sense to ensure that these behave properly as soon as possible to avoid skewing other test's results.
- Core components are tested early and developers will have more time to fix the issues.
- It is possible to test meaningful system functions earlier since at an intermediate stage all"system" modules are tested.

Of course, this approach also has drawbacks:

- Drivers must be created: writing and maintaining drivers can be a difficult task.
- Solving an error at the top is likely to be very hard and such and error would be identified only near the end of the testing phase.
- A "sample" system will not be available until the end of the testing phase. It may be useful to have a sample early in order to show a draft to our client.
- It is not possible to observe the behaviour of top level system functions until the top level components are in place.

## 2.4 Sequence of Component/Function Integration



Given this dependencies directed acyclic graph and chosen the bottom-up approach we had to select an integration order that allowed us to minimize the number of stubs needed (reducing them to zero) and we did so by taking into account the number of outgoing dependencies of each component, selecting at each step the ones that had the minimum dependencies.
In the following table we formalize how we selected at each step the components that could be integrated due to the fact that their dependencies had already been all tested, in fact the number of dependecies has to be intended as the number of outgoing arrows in the dependencies graph, updated at each step removing the components and the arrows previously selected.

| Integration step | Model manager | Auth. manager | Admin. manager | Client handler | Registr. manager | Assistance ticket manager | Power station manager | Reserv. manager | Car manager |
|---|---|---|---|---|---|---|---|---|---|
| I1 | 0 | 1 | 3 | 4 | 1 | 1 | 1 | 3 | 1 |
| I2 | | 0 | 2 | 4 | 0 | 0 | 0 | 2 | 0 |
| I3 | | | 1 | 2 | | | | 0 | |
| I4 | | | 0 | 1 | | | | | |
| | | | | 0 | | | | | |

The following table summerize which target components will be tested at each step and which driver shall be implemented in order to test them.

| Step | Test case | Driver | Target |
|---|---|---|---|
| I1 | T1 | Authentication Manager | Model Manager |
| I1 | T2 | Administration Module | Model Manager |
| I1 | T3 | Car Manager | Model Manager |
| I1 | T4 | Reservation Manager | Model Manager |
| I1 | T5 | Power Station Manager | Model Manager |
| I1 | T6 | Assistance Ticket Manager | Model Manager |
| I1 | T7 | Registration Manager | Model Manager |
| I2 | T1 | Client Handler | Authentication Manager |
| I2 | T2 | Client Handler | Registration Manager |
| I2 | T3 | Administration Module | Authentication Manager |
| I3 | T1 | Client Handler | Reservation Manager |
| I4 | T1 | Client Handler | Administration Module |

We haven't taken into account during the Integration testing phase components we won't be in charge of developing because taken as "off the shelf" solutions, these components has to be set up as first thing in order to give the developed components the infrastracture to comunicate (message broker) and go through their life cycle.

# 3 Individual steps and test description

In this character the previously defined integration order will be formalized and the necessary information to accomplish it will be given. Every n-th step will need the (n-1)-th integration step to be already been executed in order to be accomplished, instead the different parts of a single step could be performed in any order or in parallel as soon as the target components and the needed driver have been developed. The drivers has the objective to test the target's public interfaces and to check that they behave as intended on given inputs and have to be substituted by the real components when they are developed.

## 3.1 Integration test step I1

### 3.1.1 Test case 1

| | |
|---|---|
| Test case identifier | I1T1 |
| Test items (Driver → Target) | Authentication Manager → Model Manager |
| Purpose | To check if all the datas in the Model Manager can be reached correctly and persistently by the Authentication Manager and their manipulation is dealt as intended. Particular care should be put into ensuring that passwords are checked correctly since they will likely be salted and hashed. |
| Input specification | Users' and Operators' authentication data modification and queries have to be performed in order to check their correctness. |
| Output specification | Positive acknowledgment and successive consistent retrieval of the same data has to be returned. |
| Environmental needs | The model manager has to be populated previously with consistent fake data regarding Users and Operators. |

### 3.1.2 Test case 2

| | |
|---|---|
| Test case identifier | I1T2 |
| Test items (Driver → Target) | Administration Module → Model Manager |
| Purpose | To check if the datas regarding Cars, Power Stations and Users can be correctly and persistently accessed and manipulated by the Administration Module. |
| Input specification | Users', data modification and queries about their current status has to be performed to check the correct behaviour of the Model Manager and the persistence of its actions. |
| Output specification | Positive acknowledgment and consistent successive data retrieval of the same data has to be returned. |
| Environmental needs | The model manager has to be populated previously with consistent fake data regarding Users, Cars and Power Station. |

### 3.1.3 Test case 3

| | |
|---|---|
| Test case identifier | I1T3 |
| Test items (Driver → Target) | Car Manager → Model Manager |
| Purpose | To check if the datas regarding Cars can be correctly and persistently accessed and manipulated by the Car Manager. |
| Input specification | Cars' data modification and queries about their current status has to be performed to check the correct behaviour of the Model Manager and the persistence of its actions. |
| Output specification | Positive acknowledgment and consistent successive data retrieval of the same data has to be returned. |
| Environmental needs | The model manager has to be populated previously with consistent fake data regarding Cars. |

### 3.1.4 Test case 4

| | |
|---|---|
| Test case identifier | I1T4 |
| Test items (Driver → Target) | Reservation Manager → Model Manager |
| Purpose | To check if the datas regarding Reservations of Users can be correctly and persistently accessed and manipulated by the Car Manager. |
| Input specification | Reservations' data modification and queries about their current status has to be performed to check the correct behaviour of the Model Manager and the persistence of its actions. |
| Output specification | Positive acknowledgment and consistent successive data retrieval of the same data has to be returned. |
| Environmental needs | The model manager has to be populated previously with consistent fake data regarding Reservations. |

### 3.1.5 Test case 5

| | |
|---|---|
| Test case identifier | I1T5 |
| Test items (Driver → Target) | Power Station Manager → Model Manager |
| Purpose | Check that the Power Station Manager can correctly update and have access to the data concerning Power Stations (available and total plugs, current cars etc) |
| Input specification | Try to add or remove cars from a Power Station as well as retrieve its status (current usage, available and total plugs). Checks must assert that every change is persistent as well as verifying that the information retrieved is always correct and up-to-date. |
| Output specification | All actions have the expected effects on the database and the edits are visible. |
| Environmental needs | The model manager has to be populated with fake but consistent data about Power Station. Ideally one or more fake Power Stations are inserted and will allow to check all possible scenarios: insertion in an empty or partially full station, retrieval of status in empty, half full and full station. |

### 3.1.6 Test case 6

| | |
|---|---|
| Test case identifier | I1T6 |
| Test items (Driver → Target) | Assistance Ticket Manager → Model Manager |
| Purpose | Check if a ticket can be correctly created, updated (modified) and deleted. Any of these actions must be persistent. Retrieval of tickets (all active tickets, all resolved tickets, a specific ticket, ...) must return the correct data. |
| Input specification | Insertion, modification and deletion of tickets that are already in the database and that are created by the test itself. It is important to check that every action has the desired effects on the database in any scenario. |
| Output specification | The database is modified as intended and all modifications are persistent and can be retrieved correctly. |
| Environmental needs | The database must be populated with fake tickets, some of which must be resolved while others should be marked as pending. |

### 3.1.7 Test case 7

| Test case identifier | I1T7 |
|---|---|
| Test items (Driver → Target) | Registration Manager → Model Manager |
| Purpose | Check that any registration results in the creation or update of all tables and fields concerning the newly created user. All fields must contain the correct information, particular care should be put in checking that passwords are correctly stored and retrieved since they are likely salted and hashed. Deletion and modification of current user's data must be possible and work as intended. |
| Input specification | All the data needed to fill all fields of a new user (creation), the data to update and their respective fields (update), the user to be deleted (deletion). |
| Output specification | All user's data can be retrieved from the database and is correct, or the user is no longer in the database in case of deletion. |
| Environmental needs | The model manager has to be populated previously with consistent fake users. |

## 3.2 Integration test step I2

### 3.2.1 Test case 1

| Test case identifier | I2T1 |
|---|---|
| Test items (Driver → Target) | Client Handler → Authentication Manager |
| Purpose | To check if trying to login (authenticate) works as intended, correctly granting access if and only if the correct combination of user and password is inserted. The access should correctly grant the appropriate privileges (user, admin, operator). |
| Input specification | Username and password. |
| Output specification | Positive answer and access confirmation if the correct combination of username and password is given, an error otherwise. |
| Environmental needs | Some fake users must be already present in the database. |

### 3.2.2 Test case 2

| | |
|---|---|
| Test case identifier | I2T2 |
| Test items (Driver → Target) | Client Handler → Registration Manager |
| Purpose | To check if it is possible to register new users and if the component correctly interacts with the database, e.g. not permitting the creation of already registred users. |
| Input specification | Creation of a new user. |
| Output specification | Positive acknowledgment and consistent successive data retrieval of the same data has to be returned in case of success, otherwise an error message. |
| Environmental needs | Some fake users must be already present in the database. |

### 3.2.3 Test case 3

| | |
|---|---|
| Test case identifier | I2T3 |
| Test items (Driver → Target) | Administration Module → Authentication Manager |
| Purpose | To check if it is possible for administrators to have privileges that they need to have. |
| Input specification | Execute some queries that only administrors can do, after being logged as administrators. |
| Output specification | Correct query result consistent database if modified. |
| Environmental needs | Administrator account existent in the database. |

## 3.3 Integration test step I3

### 3.3.1 Test case 1

| | |
|---|---|
| Test case identifier | I3T1 |
| Test items (Driver → Target) | Client Handler → Reservation Manager |
| Purpose | To check if it is possible to access registration features e.g. creating and deleting. |
| Input specification | Execute some calls that create, delete, modify registration. |
| Output specification | Consistent state of the database according to the functions called. |
| Environmental needs | The database must be filled with some data in order to create, delete or modify reservations. |

## 3.4 Integration test step I4

### 3.4.1 Test case 1

| | |
|---|---|
| Test case identifier | I4T1 |
| Test items (Driver → Target) | Client Handler → Administration Module |
| Purpose | To check if it is possible to access administration functionalities, such as editing user profiles or car data. |
| Input specification | Execute some administration calls. |
| Output specification | Consistent state of the database according to the functions called. |
| Environmental needs | The database must be filled with some data in order to be modified by administrators. At least an administrator has to be present. |

## 3.5 End to End integration test (E2E)

In order to test the whole system against a real world scenario, that will guarantee the core business functionalities to be fully functioning, we have decided to add a further integration step on the deployed system. This step shall simulate the entire process of renting a car, following these steps:

1. the registration of a new user with valid data.
2. the login of the newly created user.
3. the choice of a specific car.
4. the actual rental of that car. (unlocking, locking it back after the the end of the rental).
5. the proportional payment for the rental calculation.
6. the payment, we'll accept the test as successfull when the request will be received by the fake user's banking account.

If any of the above steps doesn't succeed, the test has to be considered failed. The car chosen has to be faken for obvious reasons and to grant the testing system to check that the request has really been forwarded correctly, giving a coherent successful acknowledgment to the Car Manager. The client that will effectively perform the test has to be created ad hoc to allow the full automation of the process. Considering that the bank account and the user data have to be accepted by the system, they have to be created for this purpose (the driving licence can be accepted automatically only in this case). After the run of the test all modifications to the system has to be undone in both, success and unsuccess, cases. To perform this test all the aforementioned tests have to be successfully been run.

# 4 Tools and test equipment required

## 4.1 Tools

Testing would be impossible without the use of proper tools. There are on the market many different frameworks that achieve the task of automatizing testing. We briefly describe those that we are going to use during the integration test phase. For further informations and technical details refer to the official web site of each tool.

### 4.1.1 JUnit

JUnit is a testing framework for Java source code. Integrated in most Java IDEs, it allows automatic unit testing. Since other testing frameworks heavily rely on it, it is part of our integration test suite.

### 4.1.2 AssertJ

AssertJ provides a rich set of assertions, that can make more readable JUnit tests. In fact test shouldn't be obscure since different developers have to work together with them.

### 4.1.3 Mockito

Mockito is mocking framework. With its API it is easy to write unit tests that rely on other classes, by just mocking them i.e. creating an empty box which can be filled with any behaviour. In case of integration testing interfaces can be mocked so to isolate some components from the others. In this way it is possible to disambiguate sources of errors.

### 4.1.4 Arquillian

Arquillian provides a component model for integration tests, which includes dependency injection and container life cycle management. It is built on top of JUnit. It makes quite easy testing interactions with database, and interoperability between the programming model and the different available containers.

### 4.1.5 Pact JVM

Pact is a testing tool compliant created to support the Consumer Driven Contract testing. A Contract is a collection of agreements between a client (Consumer) and an API (Provider) that describes the interactions that can take place between them. Consumer Driven Contracts is a pattern that drives the development of the Provider from its Consumers point of view. Pact is a testing tool that guarantees those Contracts are satisfied.
We think that this testing approach is orthogonal to classic integration testing and so that code that passes both is more trusty. Furthermore this approach forces developers to write code in a better way, following closely requirements and architectural decisions, encouraging best practices.

### 4.1.6 Manual testing

There are some software components or functionalities that cannot be tested in automatic way. In those cases the only way of testing them is by manually stressing them. GUI is an example of component that can be tested only manually and in particular it's usability.

After a huge number of tests the developer decides if it is compliant with the specification of both functional and non functional requirements. One possibility to enhance the reliability of this approach is to distribute the artifact to some people required to leave a feedback after having spent some time with the software. Manual testing will be used also for IoT integration, in particular for power grid and on car software.

## 4.2 Test equipment

The environment needed for testing the server side of our application is nothing but the development environment.
Client side testing instead needs a more accurate description.

**Web interface** Customers should be able to browse our web app from every browser installed on every type of device. So both the PC version and mobile one should be tested in different browsers on different operating systems on different devices, in particular regarding screen size.
Just to have an idea these combinations should cover the majority of the situations.

| Device type | Operating system | Browser |
|---|---|---|
| PC | Windows | Windows Explorer |
| | | Microsoft Edge |
| | | Google Chrome |
| | | Mozilla Firefox |
| | Mac Os | Safari |
| | | Google Chrome |
| | | Mozilla Firefox |
| Tablet | Android High Resolution | Stock browser |
| | | Google Chrome |
| | Android Low Resolution | Stock browser |
| | | Google Chrome |
| | iOS | Safari |
| | | Google Chrome |
| SmartPhone | Android High Resolution | Stock browser |
| | | Google Chrome |
| | Android Low Resolution | Stock browser |
| | | Google Chrome |
| | iOS | Safari |
| | | Google Chrome |

**Mobile app** The mobile app should be compatible with every recent device. Test should cover both smartphones and tablets with different versions of the OS and different screen resolutions. Same devices of the previous section can be used to test compliance to their screen resolutions and concerning OS versions, last three major releases should be taken into account.

# 5 Program Stubs and Test Data

## 5.1 Program Drivers

As mentioned we are adopting a bottom-up approach to the components integration and testing, so we'll need a number of drivers to access the external interfaces offered by the tested components, before actually implementing the real consumers of the APIs.

| Driver | Test(s) Involved | Description |
|---|---|---|
| Authentication Manager | I1T1 | Should be able to test the methods of the Model Manager to access and manipulate authentication data. |
| Administration Module | I1T2, I2T3 | Should allow to test the part of the APIs offered by the Model Manager to access and manipulate Users' data. Furthermore it has to call authentication APIs. |
| Car Manager | I1T3 | Should allow to test the APIs offered by the Model Manager to access and manipulate cars' data. |
| Reservation Manager | I1T4 | Should allow to test the storing and fetching of reservations by the ModelManager. |
| Power Station Manager | I1T5 | Should allow to test the system capability to retrieve and update a Power Station's status (available and total plugs, cars parked, ...). |
| Assistance Ticket Manager | I1T6 | Should allow to test that tickets can be correctly created, edited and deleted. |
| Registration Manager | I1T7 | Should check that the registration works properly: all input fields are correctly saved using the Model Manager |
| Client Handler | I2T1, I2T2, I3T1, I4T1 | It has to be able to allow existing users to login and visitors to register. Moreover it must be possible to make, cancel and edit reservations as well as using the administraion APIs. |
| Client | E2E | Has to perform all the action needed to perform the E2E test as specified communicating with the external interface of the Client Handler. |

As mentioned in the section 3.5 the only stub needed will be the Car for the E2E test.

## 5.2 Test Data

Some of our integration tests require that some data are present in the database. In the following table needed data are specified for each integration test.

| **Integration test** | **Data to fill in the DB** |
|---|---|
| I1T1 | Users and operators |
| I1T2 | Users, cars and power stations |
| I1T3 | Cars |
| I1T4 | Reservations, users, cars, power stations |
| I1T5 | Power stations and cars |
| I1T6 | Tickets, users, operators |
| I1T7 | Users |
| I2T1 | Users |
| I2T2 | Users |
| I2T3 | Operators |
| I3T1 | Reservations, users, cars, power stations |
| I4T1 | Operators, reservations, users, cars, power stations |

## 6 Effort spent

| Component | Time spent (in hour) |
|---|---|
| Philippe Scorsolini | 10 |
| Lorenzo Semeria | 12 |
| Gabriele Vanoni | 15 |