POLITECNICO
MILANO 1863

Computer Science and Engineering

SOFTWARE ENGINEERING 2

A.A. 2015 / 2016

MyTaxiService

ITPD

Integration **T**est **P**lan **D**ocument

January 21st, 2016

REFERENCE PROFESSOR:

Mirandola Raffaela

AUTHORS:

Polidori Lucia

Regondi Chiara

Table of contents

1 Introduction

1.1 Revision history

This is version 1.0 of our Integration Test Plan Document.
Other potential revisions of the document will be listed in this paragraph.

1.2 Purpose and Scope

The Integration Testing is that phase in the software testing process in which individual software modules are combined, integrated and tested as a group. It’s the middle step between the unit testing (performed to test sections of code and individual modules) and the system testing (performed on the complete integrated system, in order to test both functional and non-functional requirements).
The purpose of integration testing is to verify functional requirements, performance and reliability of the main important design items (groups of units), by exercising interfaces and modules interactions. So, the test cases are built to test whether all the components contained in each group interact correctly.

This Integration Test Plan Document describes the plan for testing the integration of the components that characterize our “MyTaxiService” application. The purpose of this document is to illustrate how our plan to accomplish the integration test has been developed. It describes how the interfaces between the components we have previously designed in our Design Document are tested.

1.3 List of definitions and abbreviations

We define here a small glossary that can be useful in order to better understand the following document (if needed, it’s possible to refer to glossaries previously defined in the RASD and in the DD).

Data System: component located on the Database Server used to manage all the relevant data and information of the application. It contains all the application data in a relational database, providing all the necessary functions in order to store and retrieve data from the database when required.

DD: the *Design Document*, a document containing the description of the design and the architecture of our My Taxi Service system.

Driver Communication System: a server component used to manage all the drivers notifications and communication process.

Guest Manager: a server component used to manage all the guests of My Taxi Service, enabling all the actions that a visitor can perform.

Log In System: a server component used to manage the log in process, verifying the inputs inserted and successfully completing the authentication.

My Taxi Service: a web application for managing online taxi requests and reservations and guaranteeing a fair management of taxi queues.

Payments System: a server component used to manage all the payments processes and all the payments associated to each request or reservation.

Profile Managing System: a server component that provides the basic functions to manage user's personal profiles.

Queue Manager: a server component used to manage all the taxis queues of the city.

RASD: the *Requirements Analysis and Specification Document*, a document containing the description of the requirements and specifications analysis phase of our software development process.

Registered User Manager: a server component used to manage all the registered users of the system.

Request System: a server component used to manage all the taxi requests.

Reservation System: a server component used to manage all the taxi reservations.

Sign Up System: a server component used to manage the registration process.

Taxi Manager: a server component used to manage all the taxi drivers.

Unit Testing: a software testing method used to test individual units of source code, single modules, functions and procedures.

User Notification System: a server component used to manage the whole notification process.

User System: a server component used to manage all the functionalities related to a registered user.

1.4 Reference documents

We wrote this document referencing the project description provided by our professor, the RASD and the Design Document we have developed in the previous months and some explicative examples of other available ITP documents.

2 Integration Strategy

2.1 Entry criteria

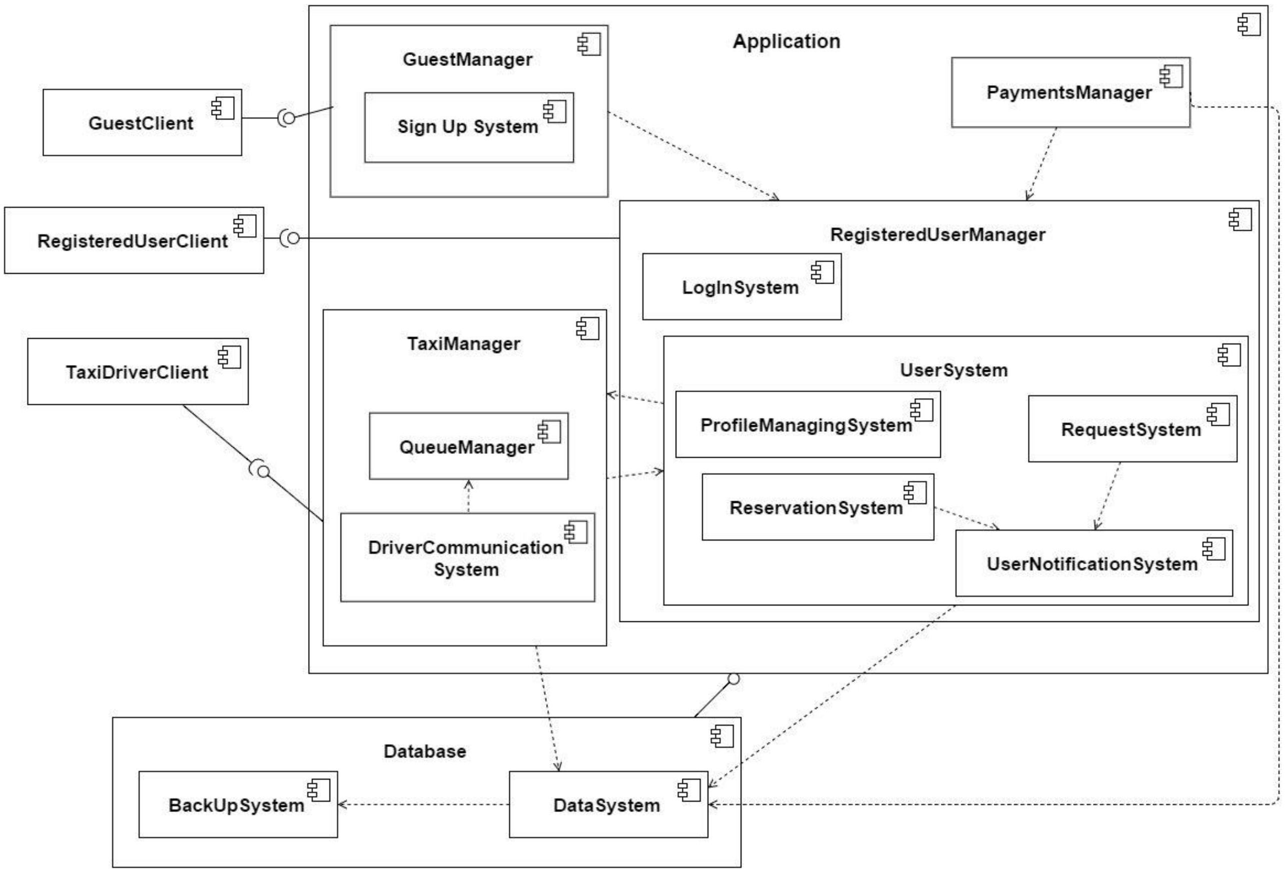
Before starting to describe our integration test plan in details, we are going to illustrate in this paragraph which are the criteria that must be met before the integration testing of specific elements may begin.

- ✓ There are no missing feature or elements in the system.
- ✓ The product satisfies all the functional and non-functional requirements specified in the *Requirements Analysis and Specification Document.*
- ✓ The architecture has already been selected and describe in detail, together with all the components of the system, in the *Design Document*.
- ✓ All the functions and individual modules have already been tested through a unit testing process and all unit test bugs have been fixed.

Moreover, the following documents must be delivered before integration testing can begin:

- ♦ Requirements Analysis and Specification Document
- ♦ Design Document (it can be developed simultaneously with this ITP document)

2.2 Elements to be integrated

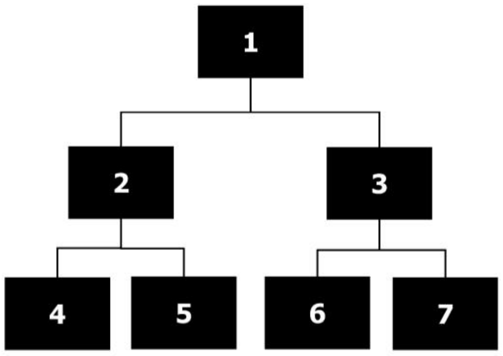


The picture reported above illustrates the components that form our “MyTaxiService” system and that need to be integrated.
The arrows represent the interactions between each pair of components that have to be tested, together with all the interfaces defined between subsystems.

2.3 Integration testing strategy

The integration testing approach that we have decided to use is the ***Bottom-up*** approach.
According to this testing technique, the components at the lowest level are tested first, so that the testing of higher component is facilitated.
This procedure stops only when the component at the top of the hierarchy is tested.

For example, if we have the following components hierarchy:



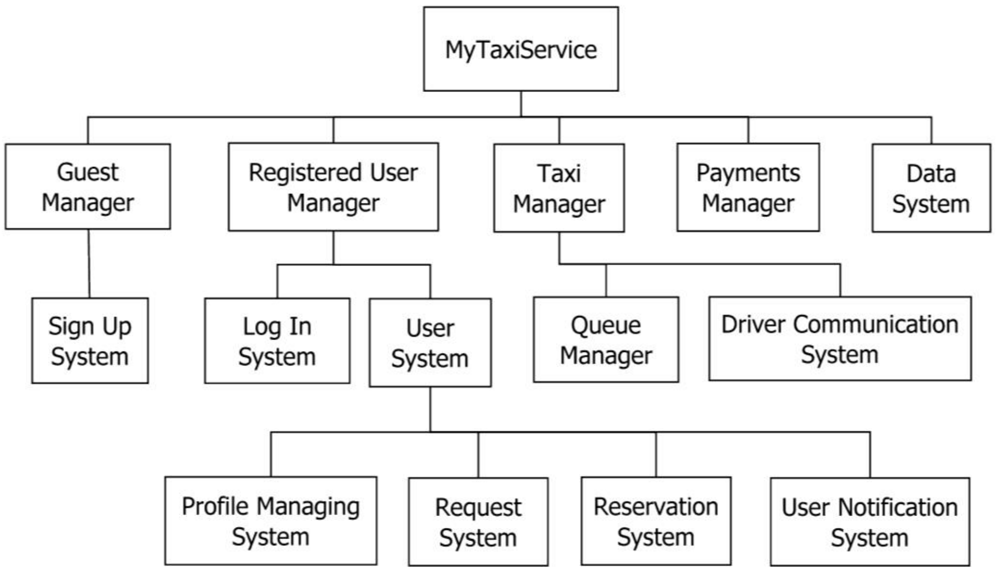
The order of integration using a bottom-up approach will be: (4,2), (5,2), (6,3), (7,3), (2,1), (3,1).
Each component at lower hierarchy is tested individually and then the components that rely upon these components are tested.

So, in this approach the testing is performed starting from the sub-modules to the main module; if the upper module is not developed, a specific component, called “*driver*”, is used to simulate this module, initializing all the needed parameters and non-local variables.

We have decided to use this approach because of the advantages that it can bring to us: in particular we think that, creating test conditions and observing test results is easier following this approach rather than another one, like, for example, a top-down technique.

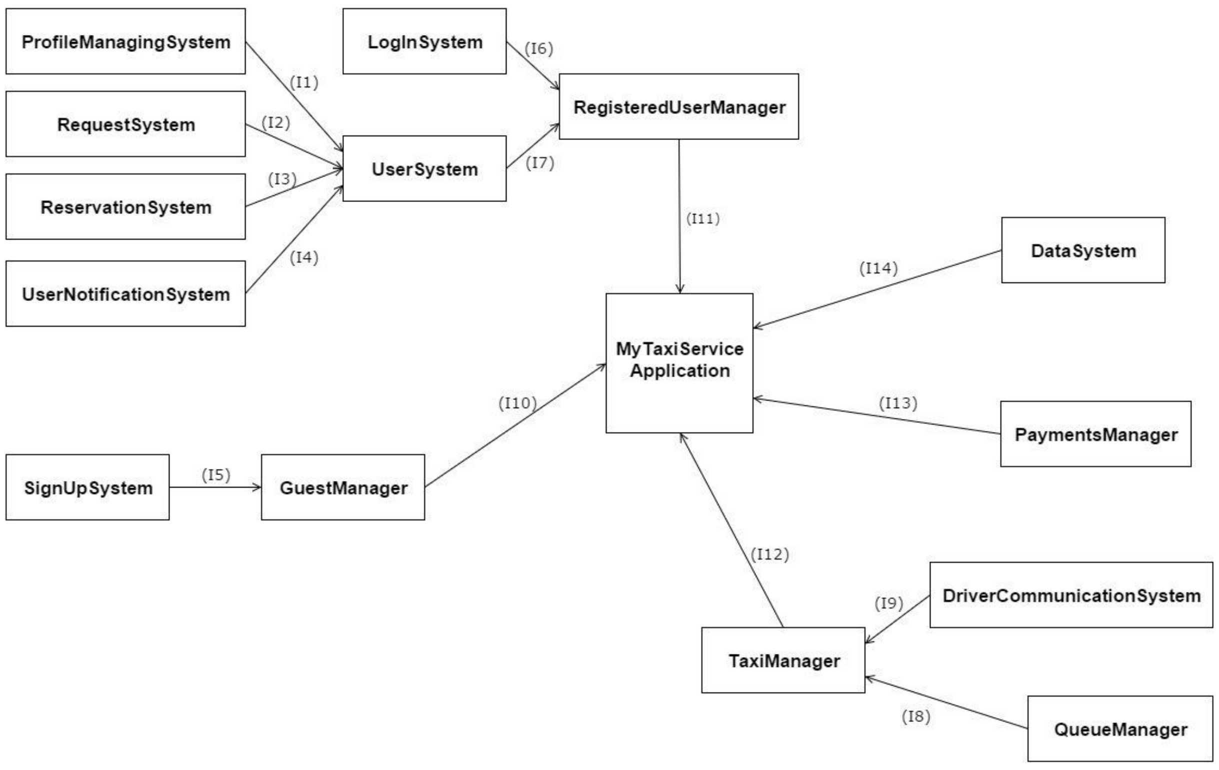
2.4 Sequence of component integration

The components of our systems will be integrated according to the bottom-up approach we have decided to adopt.



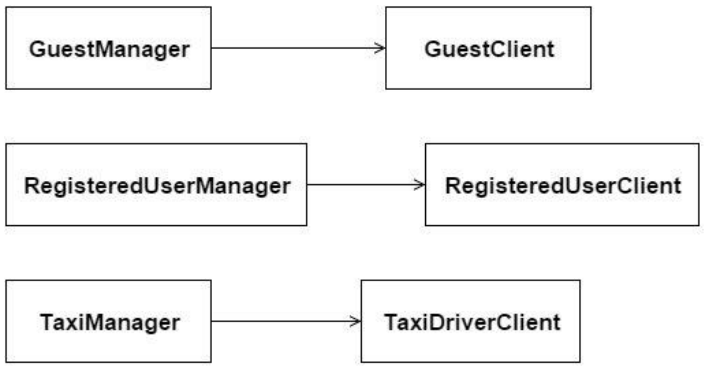
In order to make everything more understandable, we have reported above the hierarchical schema describing the components of “MyTaxiService”.
As we have already explained in the previous paragraph, the bottom-up approach guarantees that testing takes place from the bottom of the control flow upwards.
So, first of all, it’s necessary to test the components at the different levels of abstraction, according to the hierarchy defined.
In our case, the sequence in which the software components will be integrated within each subsystems is the following one:

1) Profile Managing System → User System
2) Request System → User System
3) Reservation System → User System
4) User Notification System → User System
5) Sign Up System → Guest Manager
6) Log In System → Registered User System
7) User System → Registered User System
8) Queue Manager → Taxi Manager
9) Driver Communication System → Taxi Manager
10) Guest Manager → Application (My Taxi Service)
11) Registered User Manager → Application (My Taxi Service)
12) Taxi Manager → Application (My Taxi Service)
13) Payments Manager → Application (My Taxi Service)
14) Data Systems → Application (My Taxi Service)



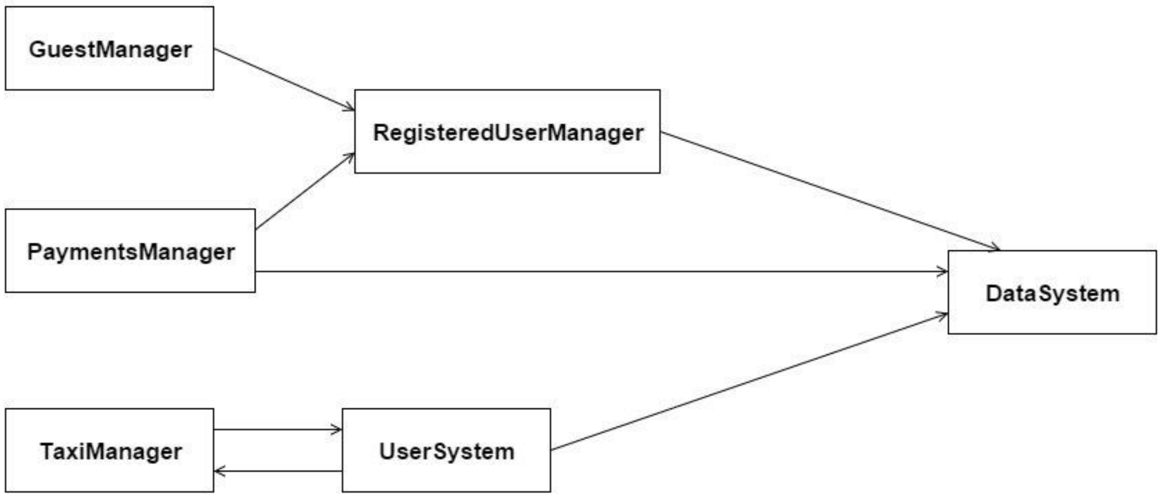
As integration testing aims at exercising both module interactions and interfaces, the integration of the following subsystems needs to be managed too, testing the interfaces between each pair of subsystems:

- Guest Manager → Guest Client
- Registered User Manager → Registered User Client
- Taxi Manager → Taxi Driver Client



Finally, the internal interactions among components in the same subsystem, plus potential communications between system's components on the same level of the hierarchy, need to be exercised:

- Guest Manager → Registered User Manager
- Payments Manager → Registered User Manager
- Taxi Manager → User System
- User System → Taxi Manager
- Registered User Manager → Data System
- Taxi Manager → Data System
- Payments Manager → Data System



3 Individual Steps and Test Description

In this chapter we are going to describe the integration steps and the tests that we have defined in the previous chapter.
As for the integration at different levels of abstraction, according to the bottom-up approach, we have decided to define a test case for each integration between two components.
We describe what are the requirements needed to perform valid tests, from the component at the bottom of our component view, to the ones on the top levels.

For example, in the first test case (I1), we integrate the “Profile Managing System” with the “User System”. A typical Profile Managing System input is created and the objective is to check whether in the User System the correct methods are called, in order to produce an output or to properly analyze and use the input received.
Obviously, as the top components have not been ready yet, we need some drivers that simulate the connection with the other components needed to test this specific integration: Registered User Client Driver and Data System Driver (because the information about the user is stored in the database).
If all the method called are correct, the test is successful.

According to this procedure, we test the integration of the other components.

Test Case Identifier	I1T1
Test Items	***Profile Managing System → User System***
Input Specification	Create a typical Profile Managing System input
Output Specification	Check if the correct methods are called in the User System
Environmental Needs	User System driver, Registered User Client driver, Data System driver

Test Case Identifier	I1T2
Test Items	***Request System → User System***
Input Specification	Create a typical Request System input
Output Specification	Check if the correct methods are called in the User System
Environmental Needs	User System driver, Registered User Client driver, Data System driver

Test Case Identifier	I1T3
Test Items	***Reservation System → User System***
Input Specification	Create a typical Reservation System input
Output Specification	Check if the correct methods are called in the User System
Environmental Needs	User System driver, Registered User Client driver, Data System driver

Test Case Identifier	I1T4
Test Items	***User Notification System → User System***
Input Specification	Create a typical User Notification System input
Output Specification	Check if the correct methods are called in the User System
Environmental Needs	User System driver, Registered User Client driver, Data System driver

Test Case Identifier	I2T1
Test Items	***Sign Up System → Guest Manager***
Input Specification	Create a typical Sign Up System input
Output Specification	Check if the correct methods are called in the Guest Manager
Environmental Needs	Guest Manager driver, Guest Client driver

Test Case Identifier	I3T1
Test Items	***Log In System → Registered User Manager***
Input Specification	Create a typical Log In System input
Output Specification	Check if the correct methods are called in the Registered User Manager
Environmental Needs	Registered User Manager driver, Registered User Client driver

Test Case Identifier	I3T2
Test Items	***User System → Registered User Manager***
Input Specification	Create a typical User System input
Output Specification	Check if the correct methods are called in the Registered User Manager
Environmental Needs	I1 succeeded

Test Case Identifier	I4T1
Test Items	***Queue Manager → Taxi Manager***
Input Specification	Create a typical Queue Manager input
Output Specification	Check if the correct methods are called in the Taxi Manager
Environmental Needs	Taxi Driver Client driver

Test Case Identifier	I4T2
Test Items	***Driver Communication System → Taxi Manager***
Input Specification	Create a typical Driver Communication System input
Output Specification	Check if the correct methods are called in the Taxi Manager
Environmental Needs	Taxi Manager driver, Taxi Driver Client driver

Test Case Identifier	I5T1
Test Items	***Guest Manager → Application***
Input Specification	Create a typical Guest Manager input
Output Specification	Check if the correct methods are called in MyTaxiService
Environmental Needs	I2 succeeded

Test Case Identifier	I5T2
Test Items	***Registered User Manager → Application***
Input Specification	Create a typical Registered User Manager input
Output Specification	Check if the correct methods are called in MyTaxiService
Environmental Needs	I3 succeeded

Test Case Identifier	I5T3
Test Items	***Taxi Manager → Application***
Input Specification	Create a typical Taxi Manager input
Output Specification	Check if the correct methods are called in MyTaxiService
Environmental Needs	I4 succeeded

Test Case Identifier	I5T4
Test Items	***Payments Manager → Application***
Input Specification	Create a typical Payments Manager input
Output Specification	Check if the correct methods are called in MyTaxiService
Environmental Needs	Database driver, Application driver

Test Case Identifier	I5T5
Test Items	***Data System → Application***
Input Specification	Create a typical Data System input
Output Specification	Check if the correct methods are called in MyTaxiService
Environmental Needs	Application driver

After the previous integration steps, the interfaces between each client and each manager component located in the server must be tested.
The integration testing steps are similar to the ones described before: a specific typical client input must be created and it's necessary to check if the correct methods are called in the corresponding manager component.
Guest Manager, Registered User Manager and Taxi Manager are tested according different procedures that verify whether each software is able to handle the respective client's input and to properly answer to it with the requested information.
Similarly, Guest Client, Registered User Client and Taxi Driver Client have to be able to handle input coming from the server and output to it specific requests or information.

Test Procedure Identifier	**TP1**
Purpose	This test procedure verifies whether the User System: • can handle Profile Managing System inputs • can handle Request System inputs • can handle Reservation System inputs • can handle User Notification System inputs • can output requested information to the Profile Managing System • can output requested information to the Request System • can output requested information to the Reservation System • can output requested information to the User Notification System
Procedure Steps	Execute I1

Test Procedure Identifier	**TP2**
Purpose	This test procedure verifies whether the Guest Manager: • can handle Sign Up System inputs • can handle Guest Client inputs • can output requested information to the Sign Up System • can output requested information to a Guest Client
Procedure Steps	Execute I2

Test Procedure Identifier	**TP3**
Purpose	This test procedure verifies whether the Registered User Manager: • can handle Log In System inputs • can handle User System inputs • can handle Registered User Client inputs • can output requested information to the Log In System • can output requested information to the User System • can output requested information to a Registered User Client
Procedure Steps	Execute I3

Test Procedure Identifier	**TP4**
Purpose	This test procedure verifies whether the Taxi Manager: • can handle Queue Manager inputs • can handle Driver Communication System inputs • can handle Taxi Driver Client inputs • can output requested information to the Queue Manager • can output requested information to the Driver Communication System • can output requested information to a Taxi Driver Client
Procedure Steps	Execute I4

Test Procedure Identifier	**TP5**
Purpose	This test procedure verifies whether the Application: • can handle Guest Manager, Registered User Manager, Taxi Manager, Payments Manager inputs • can output requested information to the Guest Manager, Registered User Manager, Taxi Manager, Payments Manager
Procedure Steps	Execute I5 after I1, I2, I3, I4

Obviously, even the interactions among components in the same subsystem, or the ones between system’s components on the same level of the hierarchy, must follow a similar procedure:
each component has to be able to manage inputs from the other and produce correct answers or retrieve the required information. The integration test is successful if all the correct methods necessary to manage the inputs are called.
Clearly, a precondition to these tests is the successful execution of integration and tests previously described.

4 Tools and Test Equipment Required

Until now, we have developed our project thinking of Java EE as programming language for the implementation.

For this reason, we have decided to use *Arquillian*, the integration testing framework for Java EE, in order to test our components interactions and interfaces.

Arquillian is a testing framework, developed at JBoss.org, that empowers the developer to write integration tests for Java EE application: it tests objects that are executed inside a container or that interact with a container as a client.

The purpose of this tool is allow developers to produce a broad range of integration tests for their Java applications. This tool allows us to understand if the right component is injected and if the interaction with the database is properly working.

We chose this type of tool because of the advantages that it offers: *Arquillian* tries to make integration testing no more complicated than the basic unit testing and it is set to become the first complete solution for testing Java EE applications, especially because it leverages the container rather than an artificial runtime environment.

5 Program Stubs and Test Data Required

During the integration testing, if there are any uncompleted functionalities or components, we can replace them with temporary dummy programs which are called “stubs” or “drivers”.
Stubs are used in the top-down integration testing approach, when the top module is ready to be tested, but the sub-modules are not ready yet.
On the other hand, drivers are the “calling” programs and are the ones used in the bottom-up integration testing approach. A driver is dummy code, used when the sub-modules are ready but the main module is not ready yet: the driver initializes all the parameters and the non-local variables that are needed.
As already explained before, we adopted a bottom-up approach: this means we have to use some drivers to successfully complete the integration.
In particular, in our integration process, we needed the User System driver, Guest Manager driver, Registered User driver, Taxi Manager driver, Registered User Client driver, the Guest Client driver, the Taxi Client driver, the Database driver and the Application driver, as already reported in the environmental needs row of the test cases table in Chapter 3.

6 Used Tools

In this paragraph we are going to list all the tools we have used to create this document:

- *Microsoft Office Word 2010* : to redact and format this document

7 Working Hours

In order to analyze the code, redact and write this document we spent about 15 hours per person.