****

2015-2016

**Integration Test Plan**

Morenzetti Sergio 791558.

Marcucci Antonio Pio 854473.

Released date: 21/01/2016

***Table of content :***

*1. Introduction*

*2. Integration Strategy*

*3. Individual Steps and Test Description*

*4. Tools and Test Equipment Required*

*5.Program Drivers*

***1. Introduction***

* 1. Revision History

This is the first version of this document.

* 1. Purpose and Scope

The document to be produced consists in the creation of an Integration Test Plan Document (ITPD) for the myTaxiService problem.

The ITPD aims at describing the plan to accomplish the integration test.

More in detail, it purpose is to describe the plans for testing the integration of the created components and the interfaces in our system.

It takes the architectural description of the software system as a starting point.

This document explains to the development team what to test, in which sequence, which tools are needed for testing (if any), which stubs/ drivers/oracles need to be developed.

* 1. List of Definitions and Abbreviations
* ITPD: Integration Test Plan Document
* RASD: Requirements Analysis and Specification Document
* DD: Design Document
* Junit: tool for unit testing
* Arquillian: tool for integration testing
* In: integration number n
  1. List of Reference Documents
* The description of the project: Assignments 1 and 2
* The RASD Document
* The DD Document
* Junit (tool)
* Arquillian (tool)

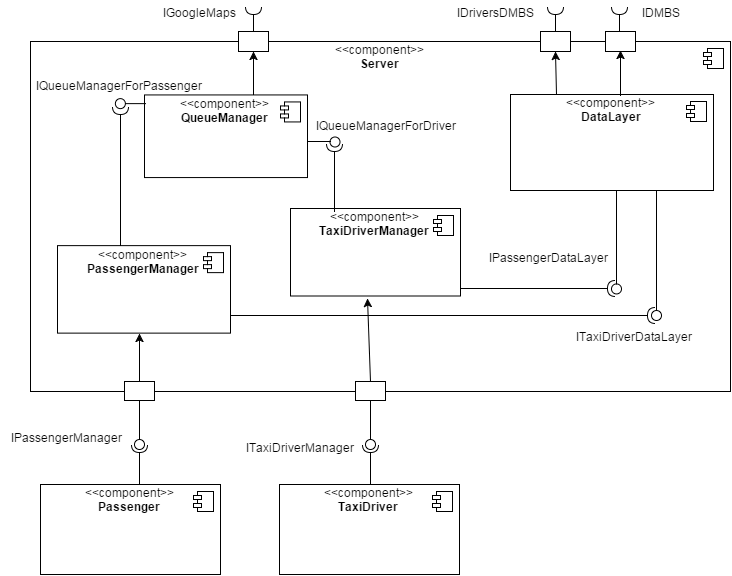
***2. Integration Strategy***

2.1 Entry Critreria

Before the Integration test, it is suggested that all the main system components and their modules are unit tested, as they are created, in order to be sure of having a solid starting point. Furthermore The RASD and the DD documents have to be completed, because lot of information exploited in those documents are mandatory for the execution of a good and consistent Integration test.

2.2 Elements to be Integrated

The diagram below is taken from the DD document and represents the set of the components and interfaces of myTaxiService application:



From the component diagram we have inferred the following dependence schema:

Passenger

TaxiDriver

TaxiDriverManager

PassengerManager

DataLayer

QueueManager

Note: The components pointed by the arrow provides functionalities requested by the one which is pointing

This schema represents how our application components should be seen divided in three levels, where we can say, that starting from the top, that a component can be seen as “input” for the lower level components; while starting from the bottom, a component can be seen as a service provider for the upper components (meaning that this component is the one which implements those functions and method that are called by the upper component) .

2.3 Integration Testing Strategy

For testing we choose a structural orientation : the Bottom up approach.

This means that integration testing starts at the bottom level, and the project will be   
built-up from it. We assume that this strategy should goes along with unit testing (right after).

We choose this approach instead of top-down , sandwich etc. following this thought : firstly we should integrate the “logic part” of our software, the one which refers the management of the queues and the relationship between the taxis and the city zones and so on; secondly the “data storing side”, related to the connection with the db; then we must integrate the system with the “drivers side” in order to be able, as last step, to communicate with customers, that means integrate the “passenger side”.

We have thought that this order should be appropriate for our application.

2.4 Sequence of Component Integration

In this section we have identified the sequence in which the software components will be integrated within the subsystem (which in our case is only one).

The following schema represents the hierarchical structure of our project.

Passanger

TaxiDriver

TaxiDriverManager

PassangerManager

DataLayer

QueueManager

According to the bottom up strategy we will show now the different steps to follow in order to accomplish the integration test:

Note: If it has the red edges, than it’s a driver for the system.

* First Integration (I1) :

PassangerManager driver

QueueManager

TaxiDriverManager driver

* Second Integration (I2) :

TaxiDriverManager driver

PassangerManager driver

DataLayer

* Third Integration (I3) :

TaxiDriver driver

TaxiDriverManager

QueueManager

DataLayer

* Fourth Integration (I4) :

Passenger driver

PassangerManager

QueueManager

DataLayer

The whole sequence can be resumed in the following table:

|  |  |
| --- | --- |
| Integration ID | Components involved in the Integration test |
| I1 | PassangerManger driver, TaxiDriverManager driver🡪QueueManager |
| I2 | PassangerManger driver, TaxiDriverManager driver 🡪 DataLayer |
| I3 | TaxiDriver driver 🡪 TaxiDriverManager (🡪 QueueManager, DataLayer) \* |
| I4 | Passenger driver 🡪 PassengerManager (🡪 QueueuManager, DataLayer) \* |

\*

During I3 and I4 we are focusing on the integration between the upper component and the new considered driver, but QueueManager and DataLayer are also present and indirectly interact in the test; for this reason we put them between brackets.

***3 Individual steps and test description***

For each step of the integration process identified in the previous section, we will describe the type of the tests that will be used to verify that the elements integrated in this step perform as expected.

* Integration test I1 :

|  |  |
| --- | --- |
| Test ID | I1 |
| Involved Components | PassangerManger driver, TaxiDriverManager driver🡪QueueManager |
| Interface(s) | IQueueManagerForPassenger, IQueueManagerForDriver |
| Input Specification | 1)Input from PassengerManager;  2)Input from TaxiDriverManager. |
| Output Specification | 1 and 2)Check if the correct methods are called in the QueueManager and data are managed correctly. |
| Purpose | This test procedure verifies whether the QueueManager component:   * can handle PassangerManager taxi request * can handle the queue updating * can forward the request to TaxiDriverManager * can handle TaxiDriverManager answer * can forward an answer to PassangerManager |
| Environmental Needs | PassangerManger driver, TaxiDriverManager driver |

* Integration test I2 :

|  |  |
| --- | --- |
| Test ID | I2 |
| Involved Components | PassangerManger driver, TaxiDriverManager driver 🡪 DataLayer |
| Interface(s) | IPassengerDataLayer, ITaxiDriverDataLayer |
| Input Specification | 1)Input from PassengerManager;  2)Input from TaxiDriverManager. |
| Output Specification | 1 and 2)Check if the right methods are called in the DataLayer, and if data storage and DB querying are correctly done. |
| Purpose | This test procedure verifies whether the DataLayer component :   * can handle PassangerManager registration * can handle PassengerManager reservation * can handle TaxiDriverManager registration * can handle the DB updating * can forward the query response to TaxiDriverManager * can forward the query response to PassangerManager |
| Environmental Needs | PassangerManger driver, TaxiDriverManager driver |

* Integration test I3 :

|  |  |
| --- | --- |
| Test ID | I3 |
| Involved Components | TaxiDriver driver 🡪 TaxiDriverManager (🡪QueueManager, DataLayer) |
| Interface(s) | ITaxiDriverManager |
| Input Specification | Input from TaxiDriver |
| Output Specification | Check if the right method is called in the TaxiDriverManager and in the other lower components |
| Purpose | This test procedure verifies whether the components :   * can handle TaxiDriver registration * can forward taxi request to TaxiDriver * can receive and elaborate TaxiDriver response * can handle the driver wills received from TaxiDriver (e.g. the request of being inserted in a taxi queue…) |
| Environmental Needs | TaxiDriver driver |

* Integration test I4:

|  |  |
| --- | --- |
| Test ID | I4 |
| Involved Components | Passenger driver 🡪 PassengerManager (🡪QueueuManager,DataLayer) |
| Interface(s) | IPassengerManager |
| Input Specification | Input from Passenger |
| Output Specification | Check if the right method is called in the PassengerManager and in the other lower components |
| Purpose | This test procedure verifies whether the components :   * can handle Passenger registration * can receive and elaborate Passenger request * can send back to Passenger his request response |
| Environmental Needs | Passenger driver |

***4. Tools and Test Equipment Required***

The tools that we identified in order to accomplish the integration test are:

*  which is used for the unit testing process, phase that we mentioned in the Entry Criteria section; we will use it mainly for testing expected results with assertions.

*  which is used to execute integration test. The Arquillian’s purpose is to provide simple testing that developers can use to produce a wide range of integration tests for their java applications. We selected this tool because it is a helpful open source standard for integration testing.

***5. Program drivers***

Based on the testing strategy and test design, we have identified program drivers required for the integration steps:

* PassengerManager (driver) : this is a very important driver because it allows to test QueueManager and DataLayer components.
* TaxiDriverManager (driver) : this is a very important driver because it allows to test QueueManager and DataLayer components.
* Passenger (driver) : it is used in order to be able to test interaction with the PassengerManager component without waiting the Customer part to be completed.
* TaxiDriver (driver) : it is used in order to be able to test interaction with the TaxiDriverManager component without waiting the Customer part to be completed.