TaxiService

Integration Test Plan Document

Authors: Jacopo Silvestri 773082 Simone Penati 850448
Reference Professor: Mirandola Raffaela
Release Date February 2016

POLITECNICO DI MILANO

0. Index

1. Introduction

This document aims at describing how the integration testing should proceed, based on the previous documents, the RASD and DD, and the specification for our project.
The TaxiService's ITPD will contain a description of the strategy we ask to implement during the integration testing, with a detailed representation of individual test steps and an precise enumeration of all things required.

1.1 Revision History
First Draft - 08/01/16
Completion and Final Revision - 16/01/16

1.2 Purpose And Scope
The Purpose of this document is to show how to test the integration between components, software and subsystems, and how their interfaces should behave with respect to one another if the testing is successful.
TaxiService is a mobile/web based application allowing people to call a taxi, be it for a request or a reservation. It will enable the city to greatly enhance its current public transportation service.

1.3 List of Definitions and Abbreviations

- I.T.P.D.: Integration Test Plan Document. This Document is the ITPD.
- R.A.S.D..: Requirement Analysis Specification Document. A Document complementary to this one in describing the service.
- D.D.: Design Document. A Document complementary to this one in describing the service.
- J.E.E.: Java Enterprise Edition. The Java platform to implement this service with.
- D.B.M.S.: Database Management System.
- C.O.T.S.: Commercial-Off-The-Shelf. A third party component of any kind used inside a proprietary project.

1.4 List Of Reference Documents
The following documents had been used for the creation of the I.T.P.D.:

- J.E.E. Documentation
- TaxiService's Design Document
- Microsoft Developer Network Guidances and Documentation
- TaxiService's R.A.S.D.
- Lecture notes on Verification and Validation

2 Integration Strategy

The strategy will be a mixed one. We intend to use a Bottom Up approach, testing the subsystem to be integrated inside each component, and then a Client/Server strategy to test the integration between the logical software components. *(See schemas at section 3.2.1 and 3.3 of the Design Document for the subsystem and component models).*

2.1 Entry Criteria

In order to proceed to the integration phase, all the functions that belong to non-C.O.T.S. components of the TaxiService software should have been tested by unit testing (possibly using Mockito Software; see Tools chapter below); a formal inspection and an in-depth analysis must have also taken place. The software has to be complete.
Obviously, the R.A.S. Document and the Design Document also have to be completed.

2.2 Elements To Be Integrated

We proceed then in verify the actual interaction between the following logical modules:

- Access Control: which has to manage the accessibility of our service by any User (both TaxiDrivers and Customers).
- DBMS User Data: which will contain all the User's (both TaxiDrivers and Customers) info.
- Customer: the thin client app that all the User(Customer) will have.
- Customer: the thin client app that all the User(Taxi Driver) will have.
- Thick Server: contains the business logic of our service.

We will *not* test:

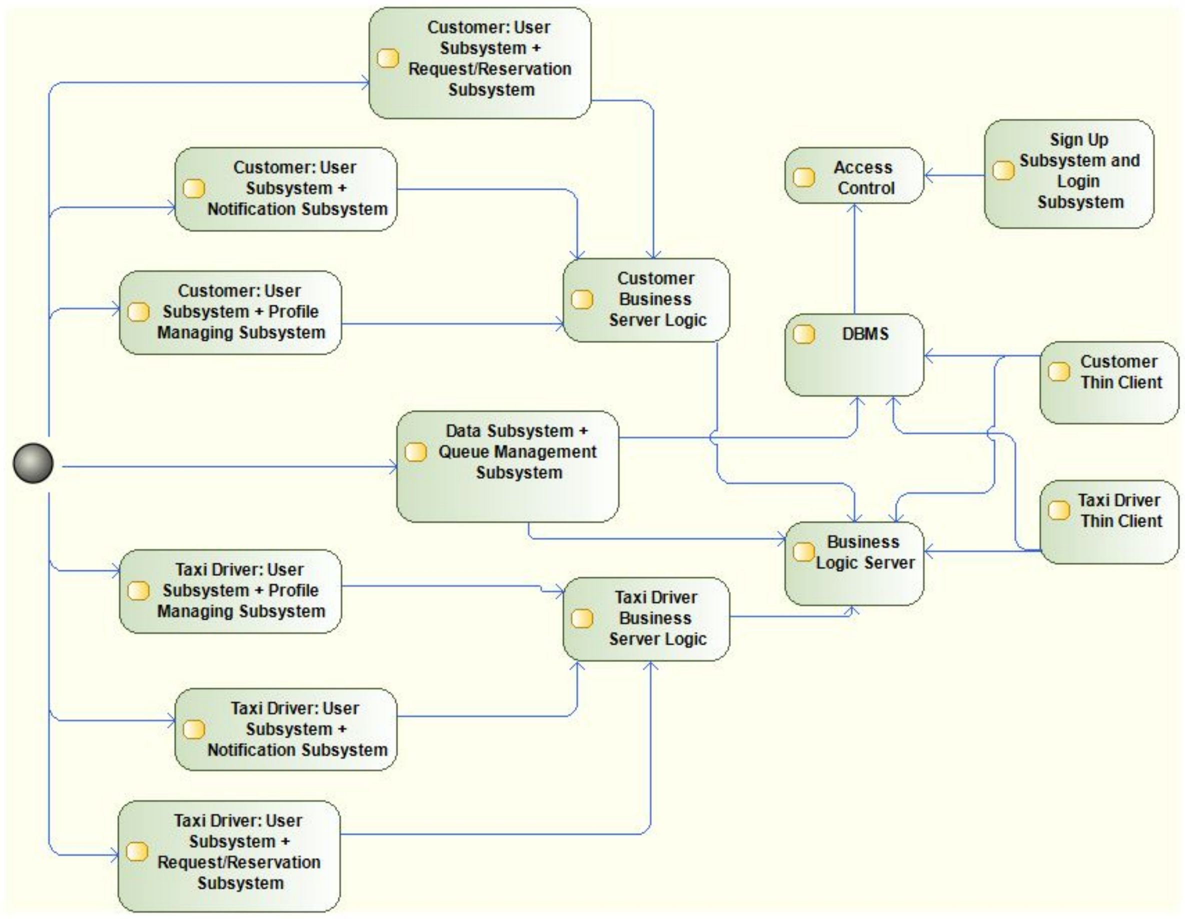
- The State Taxi Licence DBMS: it is offered by the government.
- Identity Manager: has to confirm the client's credentials when inserted and it is C.O.T.S.

2.3 Integration Testing Strategy

As mentioned above the strategy will be a mixture of the bottom-up and client-server approach, based on the structure presented in the design document. In the following subsection, we will present a general schema of the integration, and subsequently define the steps and integration sequences.

2.3.1 Integration Test Plan Schema

The integration schema shown below identifies in the leftmost half the Bottom Up integration procedure, with the subsystem being integrated from the leaves to the upper levels, to form the main components; then, proceeding with a Client/Server integration, we test that the separated components work as expected, each with his own interfaces.


Customer: User Subsystem + Request/Reservation Subsystem
Customer: User Subsystem + Notification Subsystem
Customer: User Subsystem + Profile Managing Subsystem
Customer Business Server Logic
Access Control
Sign Up Subsystem and Login Subsystem
DBMS
Customer Thin Client
Data Subsystem + Queue Management Subsystem
Business Logic Server
Taxi Driver Thin Client
Taxi Driver: User Subsystem + Profile Managing Subsystem
Taxi Driver Business Server Logic
Taxi Driver: User Subsystem + Notification Subsystem
Taxi Driver: User Subsystem + Request/Reservation Subsystem

2.4 Sequence Of Component/Function Integration

The Integration test will adopt a compartmented structure, divided as such:

FIRST PART - BOTTOM UP ***(Subsystem Integration Sequence)***

1. **Customer Business Logic - Bottom Up Integration**
    a. Profile Managing Subsystem with the User Subsystem
    b. Request/Reservation Subsystem with the User Subsystem
    c. Notification Subsystem with the User Subsystem
2. **Taxi Driver Business Logic - Bottom Up Integration**
    a. Profile Managing Subsystem with the User Subsystem
    b. Request/Reservation Subsystem with the User Subsystem
    c. Notification Subsystem with the User Subsystem
3. **DBMS and Access Control - Bottom Up Integration**
    a. Queue Management Subsystem with the Data Subsystem
    b. SignUp Subsystem with the LogIn Subsystem to form Access Control
4. **Business Logic - Bottom Up Integration**
    a. Customer Business Logic with the Taxi Driver Business Logic

SECOND PART - CLIENT/SERVER ***(Software Component Integration Sequence)***

1. **Interface Check - Access Control**
    a. Customer Thin Client with Access Control Server
    b. Taxi Driver Thin Client with Access Control Server
2. **Interface Check - DBMS**
    a. Business Logic Server with DBMS
3. **Interface Check - Business Logic**
    a. Customer Thin Client with Business Logic Server
    b. TaxiDriver Thin Client with Business Logic Server

Integration between Access Control, the DBMS and the COTS Identity Management System cannot be tested. If, however, we can assume a standard functionality for the IMS, we can use some specifically designed drivers and stubs to simulate the interaction between the components.

2.4.1 Software Integration Sequence

The Software Integration Sequence will convey that all units for each subsystem work correctly, making each subsystem functional as a whole; it will be considered as an intermediate step between unit and integration testing, given the testing plan of paragraph 2.4.

As for the Software Components Integration Sequence, it constitutes the second and final part of the testing plan.

2.4.2 Subsystem Integration Sequence

The Bottom Up steps (paragraph 2.4, first part, 1.a to 3.b) represent the Subsystem Integration Sequence, which is key to the hybrid method adopted.

3 Individual Steps And Test Description

3.1 Integration Test I1

Test Case Identifier	1.1.A
Test Item(s)	Profile Managing Subsystem ⇒ User Subsystem - Customer
Input Specification	Typical Profile Info
Output Specification	The functions of the Profile Managing Subsystem return the correct value
Environmental Needs	User Test Driver

3.2 Integration Test I2

Test Case Identifier	1.1.B
Test Item(s)	Request/Reservation Subsystem ⇒ User Subsystem - Customer
Input Specification	Generic Request, generic Reservation
Output Specification	All the correct functions of the R/R Subsystem are called
Environmental Needs	User Test Driver

3.3 Integration Test I3

Test Case Identifier	1.1.C
Test Item(s)	Notification Subsystem ⇒ User Subsystem - Customer
Input Specification	Typical Notification
Output Specification	The functions of Notification return the correct value
Environmental Needs	User Test Driver

3.4 Integration Test I4

Test Case Identifier	1.2.A
Test Item(s)	Profile Managing Subsystem ⇒ User Subsystem - Taxi Driver
Input Specification	Typical Profile Info
Output Specification	The functions of the Profile Managing Subsystem return the correct value
Environmental Needs	User Test Driver

3.5 Integration Test I5

Test Case Identifier	1.2.B
Test Item(s)	Request/Reservation Subsystem ⇒ User Subsystem - Taxi Driver
Input Specification	Generic Request, generic Reservation
Output Specification	All the correct functions of the R/R Subsystem are called
Environmental Needs	User Test Driver

3.6 Integration Test I6

Test Case Identifier	1.2.C
Test Item(s)	Notification Subsystem ⇒ User Subsystem - Taxi Driver
Input Specification	Typical Notification
Output Specification	The functions of Notification return the correct value
Environmental Needs	User Test Driver

3.7 Integration Test I7

Test Case Identifier	1.3.A
Test Item(s)	Queue Management Subsystem ⇒ Data Subsystem
Input Specification	Queue Mockup data
Output Specification	All the proper functions are accessed and return correct values
Environmental Needs	Data Management Driver

3.8a Integration Test I8a

Test Case Identifier	1.3.B.1
Test Item(s)	SignUp Subsystem ⇒ Access Control Logic
Input Specification	Typical SignUp tentative
Output Specification	The correct functions in Access Control and SignUp are called, and return the correct values.
Environmental Needs	Access Control Driver

3.8b Integration Test I8b

Test Case Identifier	1.3.B.2
Test Item(s)	LogIn Subsystem ⇒ Access Control Logic
Input Specification	Typical LogIn tentative
Output Specification	The correct functions in Access Control and LogIn are called, and return the correct values.
Environmental Needs	Access Control Driver, Test 1.3.B.1

3.9 Integration Test I9

Test Case Identifier	1.4.A
Test Item(s)	Customer Business Logic, Taxi Driver Business Logic ⇒ Business Logic
Input Specification	Typical User Input
Output Specification	The correct functions are accessed in both Business Logics
Environmental Needs	1.1.A, 1.1.B, 1.1.C, 1.2.A, 1.2.B, 1.2.C

3.10 Integration Test I10

Test Case Identifier	2.1.A
Test Item(s)	Customer Thin Client ⇒ Access Control Logic
Input Specification	A generic Customer interaction with Access Control
Output Specification	Access Control manages correctly the Client interaction
Environmental Needs	Test 1.3.B.2

3.11 Integration Test I11

Test Case Identifier	2.1.B
Test Item(s)	Taxi Driver Thin Client ⇒ Access Control Logic
Input Specification	A generic Taxi Driver interaction with Access Control
Output Specification	Access Control manages correctly the Client interaction
Environmental Needs	Test 1.3.B.2

3.12 Integration Test I12

Test Case Identifier	2.2.A
Test Item(s)	Business Logic ⇒ DBMS
Input Specification	A typical query
Output Specification	DBMS handles correctly the query
Environmental Needs	Test 1.4.A

3.13 Integration Test I13

Test Case Identifier	2.3.A
Test Item(s)	Customer Thin Client ⇒ Business Logic
Input Specification	A generic Customer interaction with Business Logic
Output Specification	Business Logic manages correctly the Client interaction
Environmental Needs	Test 1.4.A, 2.2.A

3.14 Integration Test I14

Test Case Identifier	2.3.B
Test Item(s)	Taxi Driver Thin Client ⇒ Business Logic
Input Specification	A generic Taxi Driver interaction with Business Logic
Output Specification	Business Logic manages correctly the Client interaction
Environmental Needs	Test 1.4.A, 2.2.A

4 Tools And Test Equipment Required

We have chosen these tool in order to test our software at best with respect to its own structure and functionalities. First off, we will recognize the importance in using the JMETER Software tool, as it is designed to load test functional behaviour on our TaxiService's functions and measure its performance. Also, since it was originally designed for web application, it is well suited for our project. The main goal with it is to use it along with the other tests to conduct a multiple stress test, simulating a heavy load on our thick server module to check its strength and overall durability.

As mentioned in the Entry Criteria paragraph it is best to have all the code unit-tested with the Mockito tool, as it provides great accuracy when covering the smaller testable functionalities of our project.

Finally we will use Arquillian in order to check our system interaction strength while executing tests against the TaxiService's containers. This will serve us to test the quality of the work executed by the central DBMS as well as the correctness of all component injection.

5 Program Stubs And Test Data Required

Given that our approach is based on the Bottom-Up integration strategy we find little need for stubs. Drivers on the other side are fundamental to the success of this validation test, as many of our tests requires at least one. In particular the drivers needed are:

- User Test Driver
    - It is present in the following tests: 1.1.A, 1.1.B, 1.1.C, 1.2.A, 1.2.B, 1.2.C;
    - It should emulate functionality of User (super component of both TaxiDrivers and Customer);
    - It has to mock system calls and interactions with the Profile Managing Subsystem, Notification Subsystem and Request/Reservation Subsystem when tests simulate Taxi Calls, Profile operations and Notification sending.
    - It has to distinguish the case between TaxiDrivers' operations and Customer's.

- Data Management Driver
    - It is present in the following tests: 1.3.A.
    - It has to simulate the managing function for the taxi queue, returning mockup values. It also should simulate the behavior of the system when interactions with the database happen.

- Access Control Driver
    - It is present in the following tests: 1.3.B.1, 1.3.B.2.
    - It has to simulate the login function for the TaxiService's client, returning mock values which would be inputs of an User.
    - It has to distinguish the case between TaxiDrivers' operations and Customer's.

The amount of data necessary to the integration testing phase should be enough to run the drivers' simulated execution. The data we will need are:

- Data about typical profile info.
- A generic Request.
- A generic Reservation.
- A typical notification.
- A Generic Database Query.
- Some fake Map and Taxi Data in order to mock the creation of a queue.
- Access Data to emulate a Login/Sign In tentative.

In this document we assumed that the Identity Management System, as COTS, was already tested and provided standard API in order for us to integrate that functionality in the software; therefore, we did not consider it in the integration testing. However, since this is an ideal assumption in an ideal situation, it is safe to assume that a program stub, simulating the response of the IMS to external input (interactions with Access Control and the DBMS, as of paragraphs 2.4 of this document and 3.3 of the DD), could and should be implemented to test specifically that these third party functionalities integrate well and smoothly with our software.