Politecnico di Milano

A.A. 2015-2016

Software Engineering 2 Project: myTaxiService

Integration Test Plan Document

Damiano Binaghi, Giovanni Zuretti

Contents

1 Introduction

1.1 Revision History

- First version released on the 17th of January 2016.

1.2 Purpose and Scope

The purpose of this document is to explain and plan how the integration among the software modules of our application should be done. We will proceed by giving an overview of the whole system, identifying the subsystems that must be integrated, and for each subsystem we will also provide the software modules which compose them and how the integration among those software modules should be done. We will give a description of what integration tests should achieve and which are the tools that could help in reaching those achievements.

1.3 List of Definitions and Abbreviations

- ITPD: Integration Test Plan Document
- RASD: Requirement Analysis and Specification Document
- DD: Design Document
- DBMS: DataBase Manager System
- DB: DataBase
- JEE: Java Enterprise Edition
- OS: Operative System

1.4 Reference Documents

We are now going to state which are the reference document needed for building the ITPD:

- Specification Document: Assignment 1 and 2.pdf.
- RASD: RASD-binaghi-zuretti.pdf
- DD: DesignDocument-binaghi-zuretti.pdf

2 Integration Strategy

2.1 Entry Criteria

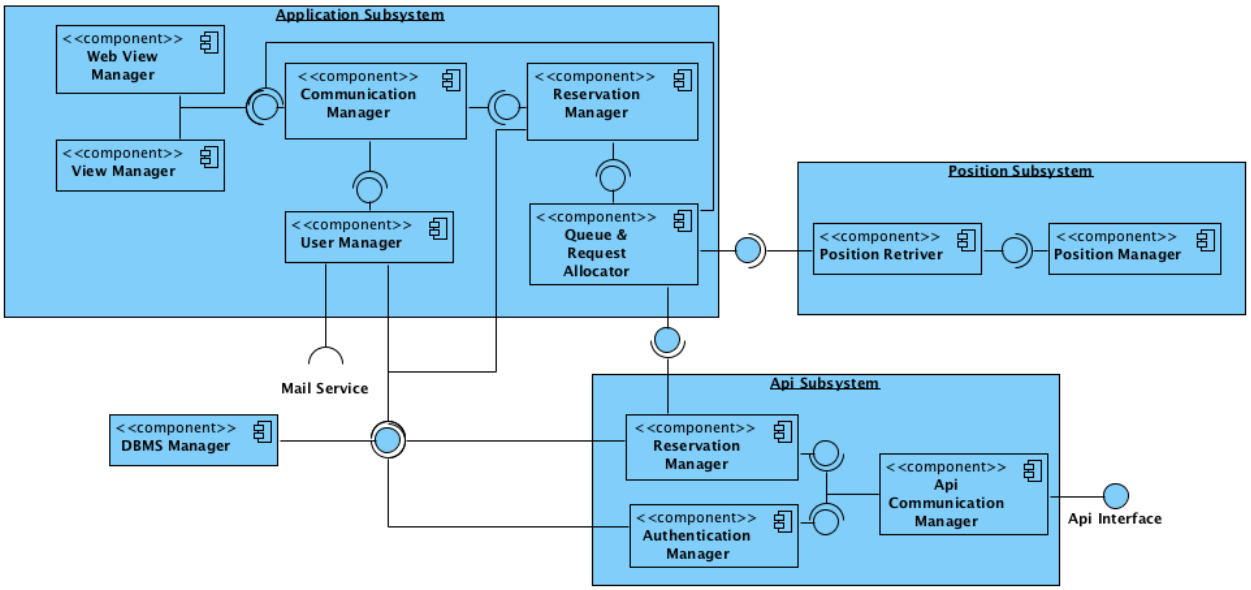
We are now going to desctibe the criteria tha any specific component needs to match before starting integration testing on such component.

- The software component, which offers the interface that needs to be tested, must be complete, which means that no code for such component is missing.
- The software component, which offers the interface that needs to be tested, must be correct, which means that all the unit tests for sich component were done and carry out successfully.
- The RASD must be complete and updated.
- The DD must be completed and updated.
- Those who provides the test cases must not be those who programmed the software modules.

2.2 Elements to be integrated

We are now going to provide a view of what are the elements that need to be integrated. Such view is built starting from the component view described in the DD. We have identified 4 subsystems:

- The Application subsystem, which includes the modules taking care of the major functionalities of our application.
- The API subsystem, which includes the software modules needed to provide the API service.
- The TaxiPosition subsystem, which includes the modules needed for managing the communication between the mobile application of the taxi dirver, communicationg the position of the taxi, and the application server.
- The DBMS Manager, which is the software module that manages the communication between the DBMS and the application. We modeled this module as a subsystem because it is independent from the other software modules of our application.



We would like to underline that the interface between Queue & Request Allocator and the Reervation Manager component is the same for both the API subsystem and the Application subsystem. For sake of semplicity in the drawing we replicated it, but it is actually only one single interface.

2.3 Integration Testing Strategy

We will use a bottom-up approach. We chose such approach because we thought was the best for the type and the complexity of the application we have designed. We thought, indeed, that the thread and the critical modules approach were too complex for our system which is rather simple. A sandwich approach could be a little confusing, while we chose bottom-up over top-down because we have more confidence with it and because it is easier to test "leaves" components in such away. Furthermore, the development of many stubs (required in top-down analysis) may require a lot of time and is more complex than building drivers for the bottom-up analysis

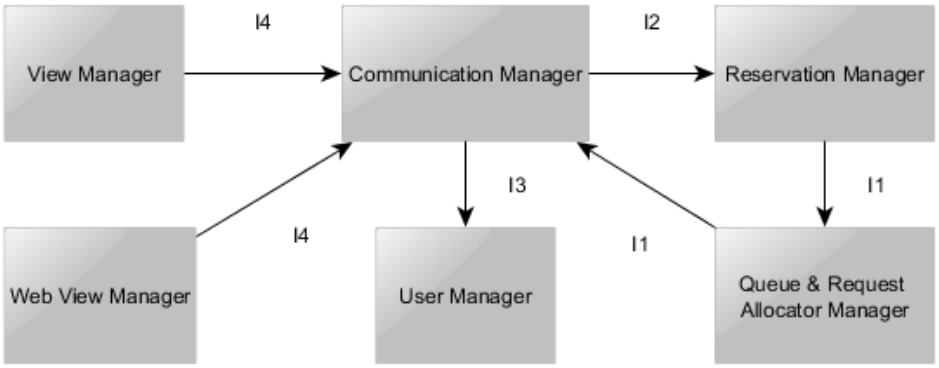
2.4 Sequence of Component Integration

We are now going to identify integration sequences of software module within each subsytem and then between the subsystems themselves.

2.4.1 Software Integration Sequence

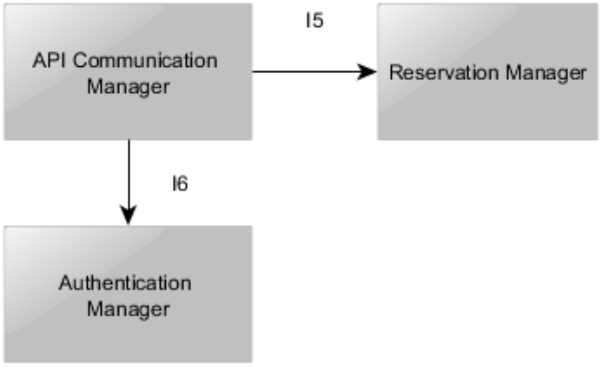
In order to be clear, $\rightarrow$ goes from the component that uses the interface to the component that offers it.

Integration Tests of the Application subsystem:



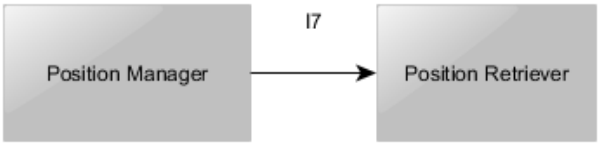
ID	Integration Test
I1	Reservation Manager → Queue & Request Allocator Manager → Communication Manager
I2	Communication Manager → Reservation Manager
I3	Communication Manager → User Manager
I4	View Manager, Web View Manager → Communication Manager

Integration Tests of the API subsystem:



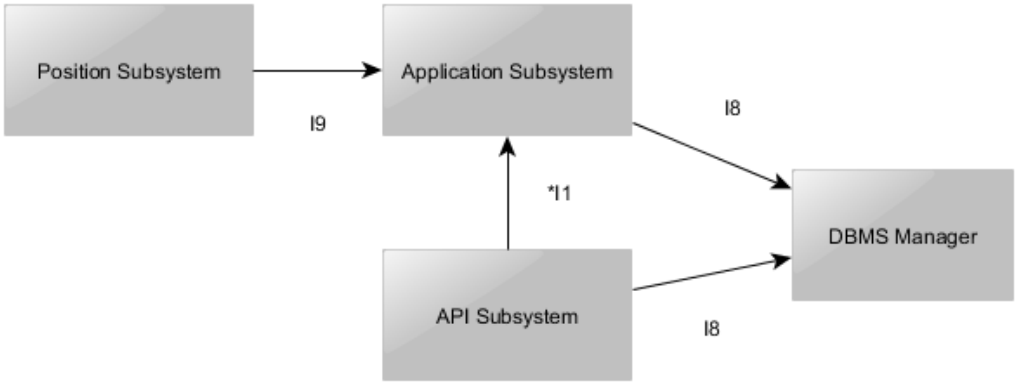
ID	Integration Test
I5	API Communication Manager → Reservation Manager
I6	API Communication Manager → Authentication Manager

Integration Tests of the Position subsystem:



ID	Integration Test
I7	Position Manager → Position Retriever

2.4.2 Subsystems Integration Sequence



ID	Integration Test
*I1	API Subsystem → Application Subsystem
I8	Application Subsystem, API Subsystem → DBMS Manager
I9	Position Subsystem → Application Subsystem

*I1 has such id, because the interface that must be tested in this step is the one already tested in test case I1. Thus, there will be no need to test it any further.

3 Individual Steps and Test Description

3.1 Integration test cases

3.1.1 Integration test case I1

Test Items: Reservation Manager → Queue & Request Allocator Manager → Communication Manager
Input Specification: Create multiple instance of an instant-booking request by different users in different zones. Create also different type of answers (accept or refuse) by taxi drivers.
Output Specification: Check that a taxi driver is assigned to the requests.
Objects and Methods: It willl be used the method "newRequest (user, position)" in the Queue & Request Allocator Manager, an the chain of calls for assigning a taxi driver between the Queue & Request Allocator Manager and the Communication Manager**.**
Enviromental Needs: Request Manager Driver, which creates the instant-booking requests, and Communication Manager Stub, which receives the communications of assignation of a driver and relplies to them. Furthemore, the Queue & Request Allocator must be initialized with different queues and with different taxi drivers in each of them.

3.1.2 Integration test case I2

Test Items: Communication Manager → Reservation Manager
Input Specification: Instances of requests for instant or advance booking.
Output Specification: Confirmation of the creation of the reservation instance in the system.
Objects and Methods: It will be used the method "createInstantRequest(user, position)", which will cause the activation of the chains of calls seen in test case I1, and it will also be calle the method "createAdvanceRequest(user, position, date, time)" which will need a stub of the DBMS Manager component in order to be completed succesfully.
Enviromental Needs: I1 succeded, Communication Manager Driver, which makes requests for a new booking instance, and a stub of the DBMS Manager which simulate the creation of a trigger in case of advance-booking request. The DBMS is simulated in Main Memory. It is simple to realize such mechanism and it allows us to exclude the DB from the testing.

3.1.3 Integration test case I3

Test Items: Communication Manager → User Manager
Input Specification: Creation of new users, simulation of login procedure, requests for modification of user profile information.
Output Specification: Confirmation of account creation, success/reject login, user information modified succesfully.

Objects and Methods: The methods "RegistrateNewUser(data)", "confirmMailUser(user)" and "modifyUser(data)" will be used.
Enviromental Needs: Communication Manager Driver, which will call the aforementioned methods. Creation of a stub of the DBMS Manager which replies in postive or negative way to the password checking on the DB or with succesful/unsuccesful result for creation and modification of the user. Once again, the DBMS is simulated in Main Memory. It is simple to realize such mechanism and it allows us to exclude the DB from the testing.

3.1.4 Integration test case I4

Test Items: View Manager, Web View Manager → Communication Manager
Input Specification: Coverage of all requests coming from the view.
Output Specification: Right aswer messagge for each type of request.
Enviromental Needs: Test 1, 2 and 3 succeded and View Driver.

3.1.5 Integration test case I5

Test Items: API Communication Manager → Reservation Manager
Input Specification: See test case I2.
Output Specification: See test case I2.
Enviromental Needs: API Communication Manager Driver which creates the reservation requests.

3.1.6 Integration test case I6

Test Items: API Communication Manager → Authentication Manager
Input Specification: Authentication data.
Output Specification: Control that token is generated if and only if data are correct.
Enviromental Needs: API Communication Manager Driver which uses the Authentication Manager interface. Stub of the DBMS Manager, as in test case I3

3.1.7 Integration test case I7

Test Items: Position Manager → Position Retriever
Input Specification: Incoming GPS positions
Output Specification: Request of position updating if the position of the taxi has changed .
Enviromental Needs: Position Manager Driver.

3.1.8 Integration test case I8

Test Items: Application Subsystem, API Subsystem → DBMS Manager
Input Specification: User related operation (such as sign up, login, editing of user personal information) and creation of advace booking instances.
Output Specification: Check that the information exchange between DBMS Manager and the two subsystem is correct. Check that the advance booking management is correct and fullfills the time constraints (i.e. trigger in the DB are activated at the right time)
Enviromental Needs: Subsystems were fully and succesfully tested.

3.1.9 Integration test case I9

Test Items: Position Subsystem → Application Subsystem
Input Specification: GPS posidion update coming from different zones and different taxis.
Output Specification: Check that the information about taxi positons are correctly managed and updated.
Enviromental Needs: Subsystems were fully and succesfully tested.

3.2 Test Procedures

3.2.1 Integration test procedure TP1

Purpose: This procedure verifies that

- The system manages a booking request by the user.
- The system correctly communicates with the assigned taxi.
- The system retrieves and generates the correct information about the booking request and that it communicates them to the interested parts.
- The system handles correctly the generation of request from an advance booking instance stored in the DB.

Procedure steps: I1, I2, I4, I8

3.2.2 Integration test procedure TP2

Purpose: This procedure verifies that

- The system manages correctly the registration of a new user.
- The system manages correctly the login procedure.
- The system manages correctly modification of user infromation.

Procedure steps: I3, I4, I8

3.2.3 Integration test procedure TP3

Purpose: This procedure verifies that

- The system manages correctly the updating of the position of the taxis

Procedure steps: I7, I9

3.2.4 Integration test procedure TP4

Purpose: This procedure verifies that

- The system manages correctly the login of external application through APIs.
- The system manages correctly all functionalities provided through APIs.

Procedure steps: I1,I5, I6, I8

4 Tools and Test Equipment Required

Since the web application is going to be developed with JEE, a good strategy for integration testing would be to use the Arquillan testing framework. Integration testing is critical for JEE apllications, for which the behavior of components of the system implemented do not depend only on the components themselves, but also on the interactions among them. As the documentation on Arquillan states, "The mission of the Arquillian project is to provide a simple test harness that developers can use to produce a broad range of integration tests for their Java applications (most likely enterprise applications). A test case may be executed within the container, deployed alongside the code under test, or by coordinating with the container, acting as a client to the deployed code." Thus we think Arquillan would be the best tool to use in the process of automating the integration testing of our application. Such tool requires to be configured with JUnit (tool for unit testing in java), which thus is going to be also needed to carry out integration testing succesfullly.
On top of using tools for automating the integration test phase, we should not forget that manual testing can always be good to test if functionalities of the system are actually working properly. We think that integrating the tests done with Arquillan with manual testing, exspecially for what concerns the front end components, it is fundamental in order to achieve a complete integration testing.
Finally we are going to give a brief list of the equipment require in the testing enviroment:

- Glassfish Server open source edition 4.1
- Oracle database 12c
- Java EE.7
- NetBeans IDE 8.1 or Eclipse 4.5.1
- Any OS which supports JDK 7 and JRE 7
- Latest version of most used browser (such as Google Chrome, Mozilla Firefox, Safari, Internet Explorer) for the testing of the web application
- For the mobile application components that are going to be tested during integration testing, we think they should be tested on the latest version of both iOS and Android.

5 Program Stubs and Test Data Required

Since we have used a bottom-up approach, we need several drivers for testing our compononents. In particular we need:

- Request Manager Driver (Test case I1)
- Communication Manager Driver (Test cases I2 and I3)
- View Driver (Test case I4)
- Position Manager Driver (Test case I7)
- API Communication Manager Driver (Test cases I5 and I6)

Having a dependency cycle between components (Communication Manager, Reservation Manager, Queue & Request Allocator Manager) we need a stub of Communication Manager in test case I1, in order to simulate the procedure of assignation of a taxi river to a request.
Furthermore, in testing the Application Subsystem and the API Subsystem, we need a component which is able to simulate the interaction with the DB, because we think it is too risky and heavy to operate directly on the DB during the testing. For this reason we create a DBMS Manager Stub for test cases I2, I3 and I6, which answer with some preinitialized data stored in main memory.
Finally, in test case I1, the component Queue & Request Allocator manager must be initialized with a set of non-empty queues.

Work Hours
Damiano Binaghi: 15 ore
Giovanni Zuretti: 15 ore