myTaxiService

Integration test plan document

Belluschi Marco 791878, Cerri Stefano 849945, Di Febbo Francesco 852389

January 21, 2016

Contents

Chapter 1

Introduction

1.1 Revision history

In the following are listed the differences between versions:

1. Document creation

1.2 Purpose and scope

1.2.1 Purpose

This is *Integration Test Plan Document* (**ITPD**) for *myTaxiService*. It describes the process, approach and goals for testing the integration of all interfaces between the various components of the system, according to all design decisions made and described in the DD.

1.2.2 Scope

As more widely explained both in the RASD and in the DD, myTaxiService is a taxi service for a large city. The main goals of the system are:

- simplify the access of passengers to the service;
- guarantee a fair management of taxi queues.

See the two aforementioned documents for further explanations.

1.3 List of definitions and abbreviations

Definitions

- Server Database: data layer
- Server Application: application layer
- Client: client layer
- Mobile App: *myTaxiService* mobile application, in Client
- Web App: *myTaxiService* web application, in Client
- System: the union of software and hardware to be developed and implemented

Acronyms

- RASD: Requirements Analysis and Specification Document
- DD: Design Document
- API: application programming interface
- DBMS: DataBase Management System

1.4 List of reference documents

- Software Engineering 2 Project AA 2015/2016: Project Description And Rules
- Software Engineering 2 Project AA 2015/2016: Assignment 4 - integration test plan
- myTaxiService's Requirement Analysis and Specification Document (RASD)
- myTaxiService's Design Document (DD)

Chapter 2

Integration strategy

2.1 Entry criteria

The main entry condition for this phase is that each of the system components low level functions has been previously subjected to a unit test process. The documentation of all classes and functions must be written in JavaDoc. In particular, the public interfaces of each class and module should be well specified; where necessary with the formal specification language JML.

2.2 Elements to be integrated

myTaxiService, as shown in Design Document, is a three-tier system:

- DBMS
- myTaxiService server
- Client application

Each tier is composed by interacting modules:

1. Application (web/mobile)
2. Data manager
3. Account manager
4. Ride manager
5. Taxi queues
6. Map services
7. Notification

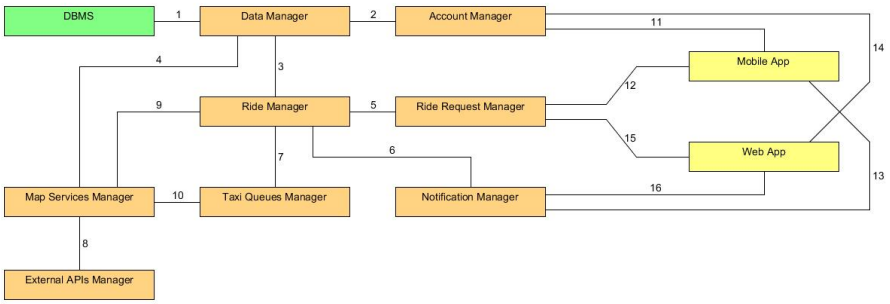
2.3 Integration testing strategy

The integration testing strategy, conducted in this project, is a *bottom-up* approach. This strategy tests the lower level components and start testing a way upwards to higher level components. The advantage of this strategy is that it's easier to maintain code, smaller modules have unit tests and there is a clearer structure of how to do things. The disadvantage is that when releasing a prototype it's impossible to see a working prototype until nearly all the program has been completed so that may take a long time before this happens. In early development, testing tools as Mockito and Arquillian (described in Chapter 4) allow us to test components which depend on incomplete ones through stubs and drivers (Chapter 5).

2.4 Sequence of component/function integration

2.4.1 Software integration sequence

The following diagram illustrates the integration sequence of the various components, following the integration testing strategy described above. This means that in each subsystem, components are integrated starting from the most independent to the less independent, in order to prompt the chosen approach and improving modularity.



2.4.2 Subsystem integration sequence

The following diagram illustrates the integration sequence of the various subsystems, following the integration testing strategy described above. In particular, the Server Database is integrated before the Client, because the former does not need an actual functioning system in order to be tested efficiently, contrary to the latter.

Chapter 3

Individual steps and test description

3.1 Individual Step and Test Description

3.1.1 Test Case I1

Test Item(s)	Data Manager $\longleftrightarrow$ DBMS
Input Specification	Queries on the DBMS for the Table User, Passenger, Taxi Driver and Ride
Output Specification	The queries return the expected results
Environmental Needs	Glassfish Server, a test Database
Purpose	Verify that the typical queries to the DBMS works

3.1.2 Test Case I2

Test Item(s)	Data Manager $\longleftrightarrow$ Account Manager
Input Specification	A set of methods calls on Data Manager to retrieve user information
Output Specification	Check that the user information are correct
Environmental Needs	Glassfish Server, a test Database, I1 successful
Purpose	Verify that the user information are retrieved from the Data Manager

3.1.3 Test Case I3

Test Item(s)	Data Manager ⟷ Ride Manager
Input Specification	A set of methods calls on Data Manager to retrieve reserved ride information
Output Specification	Check that the reserved ride information are correct
Environmental Needs	Glassfish Server, a test Database, I1 successful
Purpose	Verify that the reserved ride information are retrieved from the Data Manager

3.1.4 Test Case I4

Test Item(s)	Data Manager ⟷ Map Service Manager
Input Specification	A set of methods calls on Data Manager to retrieve zone information
Output Specification	Check that the zone information are correct
Environmental Needs	Glassfish Server, a test Database, I1 successful
Purpose	Verify that the zone information are retrieved from the Data Manager

3.1.5 Test Case I5

Test Item(s)	Ride Request Manager ⟷ Ride Manager
Input Specification	A typical set of methods calls on Ride Manager in order to create a Ride
Output Specification	Check if the correct Ride is created with the information from Ride Request Manager
Environmental Needs	I2 successful
Purpose	Create a Ride from a typical Ride Request

3.1.6 Test Case I6

Test Item(s)	Ride Manager ⟷ Notification Manager
Input Specification	A set of methods calls in order to create a notification
Output Specification	Check if the correct notification is created
Environmental Needs	I2 and I5 successful
Purpose	Verify that the Notification Manager creates the notification from the Ride Manager

3.1.7 Test Case I7

Test Item(s)	Ride Manager ⟷ Taxi Queues Manager
Input Specification	A set of methods calls on Taxi Queues Manager
Output Specification	Check if an available Taxi Driver is returned
Environmental Needs	I2 and I6 successful
Purpose	Retrieve an available Taxi Driver

3.1.8 Test Case I8

Test Item(s)	Map Services Manager ⟷ External APIs Manager
Input Specification	Create a typical set of methods calls by Map Services Manager on External APIs Manager
Output Specification	Check that all the methods of External APIs Manager produce the expected results
Environmental Needs	N/A
Purpose	Verify that the External APIs Manager works with the Map Services Manager

3.1.9 Test Case I9

Test Item(s)	Ride Manager ⟷ Map Services Manager
Input Specification	A set of methods calls on Map Services Manager
Output Specification	Verify that the returned route is the correct one
Environmental Needs	I5 and I8 successful
Purpose	Compute the optimal route

3.1.10 Test Case I10

Test Item(s)	Taxi Queues Manager ⟷ Map Services Manager
Input Specification	A set of methods calls on Map Services Manager
Output Specification	Verify that the position of the Taxi Driver is correct
Environmental Needs	I2 and I8 successful
Purpose	Retrieve the position of a Taxi Driver in a Taxi Queue

3.1.11 Test Case I11

Test Item(s)	Account Manager $\longleftrightarrow$ Mobile App
Input Specification	Create a typical set of methods calls performed by Mobile App on Account Manager
Output Specification	Check if the methods calls mentioned in Input Specification produce the expected results
Environmental Needs	A device that can run Mobile App
Purpose	Verify if Account Manager can handle correctly Mobile App methods calls

3.1.12 Test Case I12

Test Item(s)	Ride Request Manager $\longleftrightarrow$ Mobile App
Input Specification	Create a typical set of methods calls performed by Mobile App on Ride Request Manager
Output Specification	Check if the methods calls mentioned in Input Specification produce the expected results
Environmental Needs	A device that can run Mobile App
Purpose	Verify if Ride Request Manager can handle correctly Mobile App methods calls

3.1.13 Test Case I13

Test Item(s)	Notification Manager $\longleftrightarrow$ Mobile App
Input Specification	Create a typical set of methods calls performed by Notification Manager on Mobile App
Output Specification	Check if the methods calls mentioned in Input Specification produce the expected results
Environmental Needs	A device that can run Mobile App
Purpose	Verify if Mobile App can handle correctly Notification Manager methods calls

3.1.14 Test Case I14

Test Item(s)	Account Manager ⟷ Web App
Input Specification	Create a typical set of methods calls performed by Web App on Account Manager
Output Specification	Check if the methods calls mentioned in Input Specification produce the expected results
Environmental Needs	A device that can run Web App
Purpose	Verify if Account Manager can handle correctly Web App methods calls

3.1.15 Test Case I15

Test Item(s)	Ride Request Manager ⟷ Web App
Input Specification	Create a typical set of methods calls performed by Web App on Ride Request Manager
Output Specification	Check if the methods calls mentioned in Input Specification produce the expected results
Environmental Needs	A device that can run Web App
Purpose	Verify if Ride Request Manager can handle correctly Web App methods calls

3.1.16 Test Case I16

Test Item(s)	Notification Manager ⟷ Web App
Input Specification	Create a typical set of methods calls performed by Notification on Web App
Output Specification	Check if the methods calls mentioned in Input Specification produce the expected results
Environmental Needs	A device that can run Web App
Purpose	Verify if Web App can handle correctly Notification methods calls

3.1.17 Test Case SI1

Test Item(s)	Server Database ⟷ Server Application
Input Specification	Queries on the DBMS for the Table User, Passenger, Taxi Driver and Ride
Output Specification	The queries return the expected results
Environmental Needs	Glassfish Server, a test Database
Purpose	Verify that the typical queries to the DBMS works

3.1.18 Test Case S2

Test Item(s)	Server Application $\longleftrightarrow$ Client
Input Specification	A set of methods calls on both Server Application and Client
Output Specification	Check that the methods calls mentioned in Input Specification produce the expected results
Environmental Needs	Glassfish Server, a test Database
Purpose	Verify that the interaction between Server Application and Client works

Chapter 4

Tools and test equipment required

This section describes the tools required to perform the integration testing plan described previously. Following the validation and verification model, it is possible to perform different tests on components.

- **jUnit**: is a unit testing framework for the Java programming language. Each individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use.
- **Mockito**: is an open source testing framework for Java that allows the creation of test double objects (mock objects) in automated unit tests. It allows to abstract dependencies and have predictable results, and to check that the interaction between the testee and the mock is correct.
- **Arquillian**: a test framework that can be used to perform testing inside a remote or embedded container, or deploy an archive to a container so the test can interact as a remote client.
- **Apache JMeter**: is designed to load test functional behaviour and measure performance of a variety of services, with a focus on web applications.

Chapter 5

Program stubs and test data required

This section describes the specification of stubs and drivers needed to replace the part of software components that still don't exist and test the others. This is necessary to perform the integration steps. DBMS should contain sample data in order to perform proper test cases. In early development, web and mobile applications are replaced by stubs to test the client connection. These stubs should provide sample data to correctly simulate real-life conditions. During development some components may not be available yet for integration test, thus they will be replaced by drivers.

Appendix A

Appendix

A.1 Software and tool used

- TeXstudio http://www.texstudio.org/: to redact and to format this document
- eclipse https://eclipse.org/: to draw diagrams

A.2 Working hours

This is the time spent for redact the document

- Belluschi Marco: 10 hours
- Cerri Stefano: 10 hours
- Di Febbo Francesco: 10 hours