Politecnico di Milano

Scuola di Ingegneria Industriale e dell'Informazione

M.Sc. in Computer Science and Engineering

myTaxiService

Integration Test Plan Document

Authors:

Angelo Gallarello

Edoardo Longo

Giacomo Locci

January 20, 2016

Version 1.0

Contents

List of Figures

1 Introduction

1.1 Revision History

Version	Changes
1.0	Creation of the document

1.2 Purpose and Scope

This document represents the *Integration Test Plan* and it is the intended to be read by the development team. The purpose of this document is to provide a guideline to follow in order to accomplish the integration test.

1.3 List of Definitions and Abbreviations

Here is a list of the acronyms, abbreviations and terms that this document will use and are worth of an expalanation:

- Each integration test has a unique identifier with the following syntax: $\mathrm{I}[0-9]^{+}$
- Each test case has a unique identifier with the following syntax: $\mathrm{I}[0-9]^{+}\mathrm{T}[0-9]^{+}$.

1.4 List of Reference Documents

Here we list all the documents that we have used in order to produce this document:

- RASD (*Requirement Analysis and Specification Document*) of the *myTaxiService* system.
- DD (*Design Document*) of the *myTaxiService* system.
- Assignment 4 - integration test plan.
- Assignments 1 and 2 (RASD and DD).

2 Integration Strategy

2.1 Entry Criteria

Iorder to perform the integration tests the following task must be completed:

- Design and check Unit Tests.
- Design and implements driver/stub classes and method.
- Design and create test data.

2.2 Elements to be Integrated

All the components, with their respective modules, described in section 3 of the DD will have to be tested. In detail:

- **Server Component**:
  - Queue Manager
  - Ride Manager
  - Actor Manager
  - Position Manager
  - Request Manager
  - Database Interface
- **User Client**
  - Main Activity
  - Signup Activity
- **Driver Client**
  - Driving Request Fragment
  - Driving Activity

2.3 Integration Testing Strategy

This plan will follow a bottom-up approach. Provided that the unit testing is completed, this strategy enables the integration to almost always use the actual code that will be deployed, allowing the testing team to hardly ever create stubs and mocks, and reduce the use of dummy code. Of course this is not always possible and section *Program Stubs and Test Data Required* will specify when and what stubs are needed.

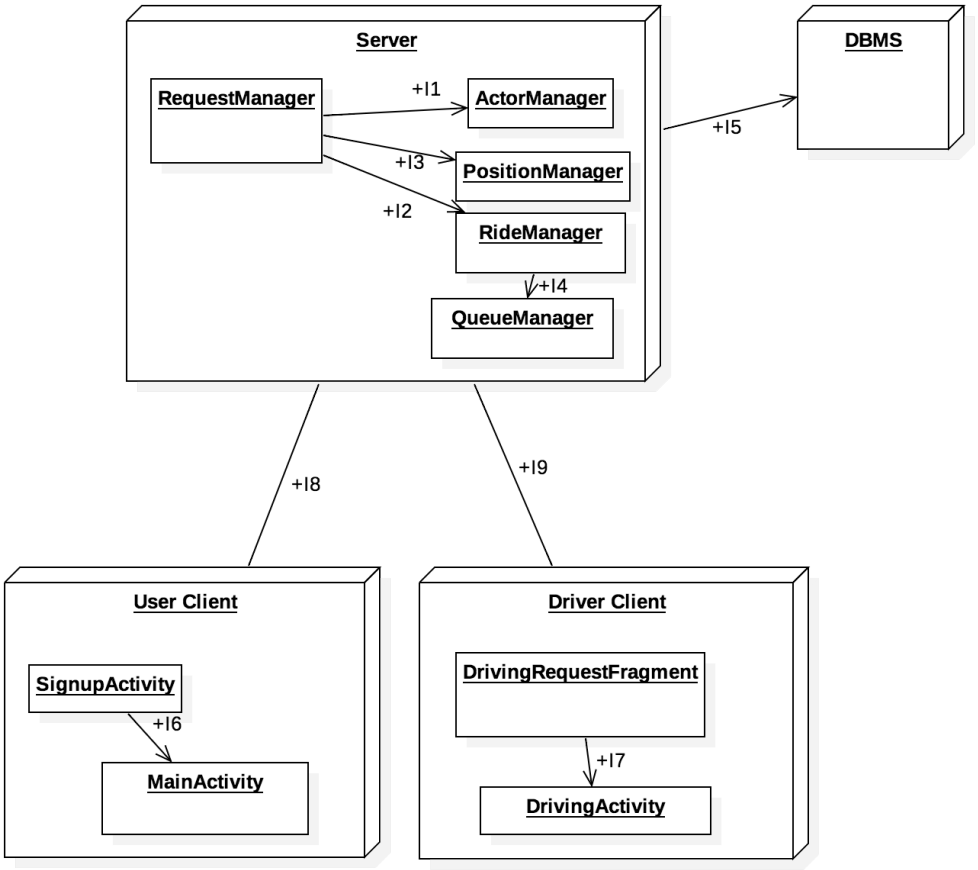
2.4 Sequence of Components

The integration plan will follow this order:

1. Server Component
2. User Client
3. Driver Client
4. Client-Server

The first three are needed in order to test the internal submodules. Once the first three test have taken place then it is possible to test the *Client-Server* interaction, with both the *User Client* and the *Driver Client*.

Figure 1: A graphic representation of the modules and interfaces that will be tested. Each arrow has a label assigned to it that identifies the integration test.



The following tables show the needed integration tests, highligthing their ID used in this document and the paragraph that describes them.

2.4.1 Server Component

ID	Integration Test			Paragraphs
I1	Request Manager	→	Actor Manager	3.1
I2	Request Manager	→	Ride Manager	3.2
I3	Request Manager	→	Position Manager	3.3
I4	Ride Manager	→	Queue Manager	3.4
I5	Database Interface	→	DBMS	3.5

2.4.2 User Client

ID	Integration Test			Paragraphs
I6	Signup Activity	→	Main Activity	3.6

2.4.3 Driver Client

ID	Integration Test			Paragraphs
I7	Driving Request Fragment	→	Driving Activity	3.7

2.4.4 Client-Server

ID	Integration Test			Paragraphs
I8	User Client	→	Server	3.8
I9	Driver Client	→	Server	3.9

3 Individual Steps and Test Description

For each of the integration tests described in section 2.4, one or more detailed test case is defined in the following tables.

3.1 Integration test case I1

Test Case Identifier	I1T1
Test Item(s)	RequestManager →ActorManager
Input Specification	Give a input to manage actors behaviour
Output Specification	Check if the object received by the ActorManager is consistent
Environmental Needs	Server running

3.2 Integration test case I2

Test Case Identifier	I2T1
Test Item(s)	RequestManager →RideManager
Input Specification	Give a input to manage ride behaviour
Output Specification	Check if the ride object generated is consistent
Environmental Needs	Server running

3.3 Integration test case I3

Test Case Identifier	I3T1
Test Item(s)	RequestManager →PositionManager
Input Specification	Stubbed User requests (Signup, Login, Ride)
Output Specification	Check if data received are consistent and the correct function are called
Environmental Needs	Server running and stubbed user request

Test Case Identifier	I3T2
Test Item(s)	RequestManager →PositionManager
Input Specification	Stubbed Driver requests (Login, Ride Management)
Output Specification	Check if data received are consistent and if the correct function are called
Environmental Needs	Server running and stubbed driver request

3.4 Integration test case I4

Test Case Identifier	I4T1
Test Item(s)	RideManager →QueueManager
Input Specification	Ride object data to the QueueManager
Output Specification	The given object has satisfiable and consistent parameters
Environmental Needs	Server running and test data

3.5 Integration test case I5

Test Case Identifier	I5T1
Test Item(s)	DatabaseInterface →DBMS
Input Specification	Typical SQL query
Output Specification	Database tables are update as expected
Environmental Needs	Server and database running with test data

3.6 Integration test case I6

Test Case Identifier	I6T1
Test Item(s)	SignUpActivity →MainActivity *(User Client)*
Input Specification	Data from SignUpActivity (username, home position)
Output Specification	Initialize MainActivity with user's data
Environmental Needs	Mock server response from Sign-Up

3.7 Integration test case I7

Test Case Identifier	I7T1
Test Item(s)	DrivingRequestFragment →DrivingActivity *(Driver Client)*
Input Specification	Data about the request from the fragment (username, home position)
Output Specification	Initialize the DrivingActivity with the correct data for the ride
Environmental Needs	Mock server notification of new ride request

3.8 Integration test case I8

Test Case Identifier	I8T1
Test Item(s)	User Client →Server
Input Specification	User Client test request
Output Specification	Request received correctly by the server
Environmental Needs	Server running and test data

3.9 Integration test case I9

Test Case Identifier	I9T1
Test Item(s)	Driver Client →Server
Input Specification	Driver Client test request
Output Specification	Request received correctly by the server
Environmental Needs	Server running and test data

4 Tools and Test Equipment Required

During the integration test, the team should use the following tools:

- Manual
- Arquillian + JUnit

4.1 Manual

We will use manual testing when the integration testing will need mock data. For instance, test [I2] will need some example data in order to be properly tested, as desccribed is section 5. This is the only reason why the integration will ever require manual testing.

4.2 Arquillian + JUnit

Arquillian, together with JUnit, provides a simple, flexible and pluggable integration testing environment. This tool will be used because it minimizes the burden on the testing team to carry out integration testing by handling all aspects of test execution.

5 Program Stubs and Test Data Required

5.1 Stubs

- In the I3T1 and I3T2 tests a stub that mocks the user request is needed in order to simulate an HTTP request.
- In the I6T1 test a stub that mocks the user data is needed in order to simulate the built up process of the activity.
- In the I7T1 test a stub that mocks notification arrival to the driver client application is needed.

5.2 Test Data

- In the I4T1 some test data are required in order to simulate the queue management of the QueueManager.
- In the I5T1 some test data are required in order to simulate input/output of data from the Database.
- In the I9T1 some test data are required in order to simulate a consisten communication between Client and Server.

Appendices

A Tools

- *Sublime Text 2* as editor
- *LatexTools for Sublime Text 2 + MacTex* to build
- *Trello* for team coordination
- *Git + Git Flow* for version control

B Hours of work

- Angelo Gallarello : 10 hours
- Edoardo Longo : 10 hours
- Giacomo Locci : 10 hours