POLITECNICO DI MILANO
Computer Science and Engineering
Project of Software Engineering 2

myTaxiService

Integration Test Plan Document
Ver. 1.0
Release date: January 21, 2016

Authors: Simone Rosmini (853949)
Vincenzo Viscusi (858689)
Matteo Zambelli (776162)

Reference Professor: Mirandola Raffaela

TABLE OF CONTENT

1. INTRODUCTION

1.1 REVISION HISTORY

- 14/01/16 - File creation and start of work.
- 21/01/16 - Version 1.0 of the document delivered.

1.2 PURPOSE AND SCOPE

This document describes the testing plan for the integration of all the parts of the myTaxiService project and has the purpose to lead the testing and integration phases of all components.
The scope of the tests is to ensure the proper functioning of the entire system.
This document is meant to be read by all the testing team components.

1.3 DEFINITIONS AND ABBREVIATIONS

RASD: Requirements Specification Analysis Document

DD: Design Document

1.4 REFERENCE DOCUMENTS

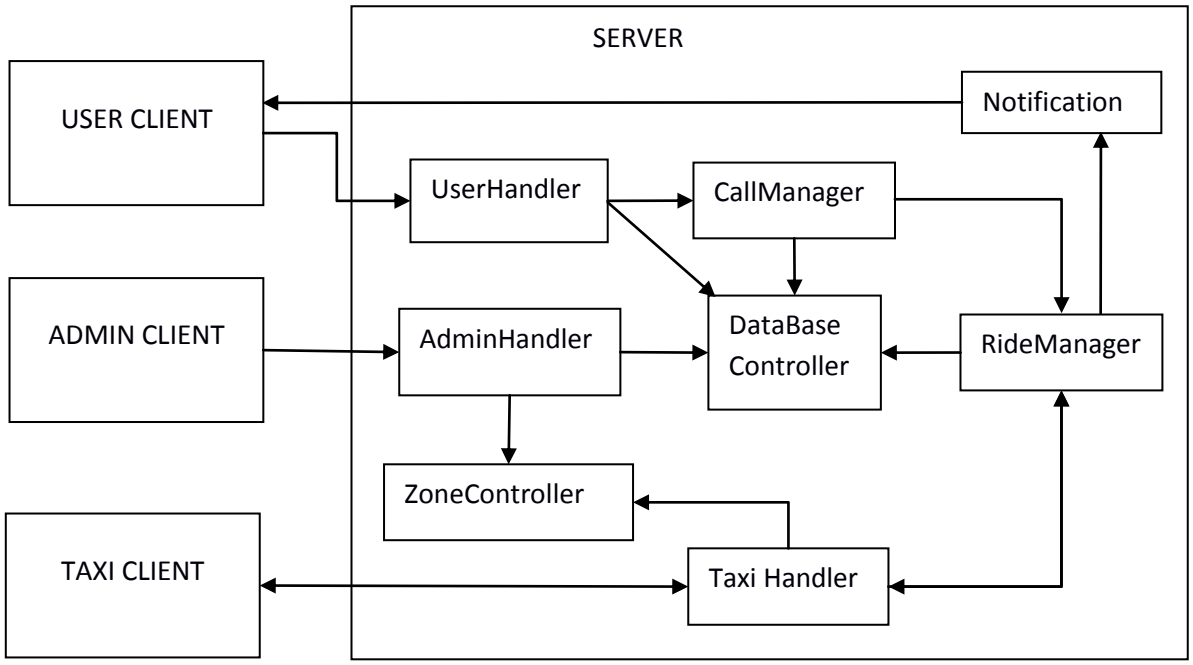
- Assignment 4 - Integration Test Plan
- myTaxyService RASD
- myTaxyService DD

2. INTEGRATION STRATEGY

2.1 ENTRY CRITERIA

Before the beginning of the integration testing it's better to have a Unit Testing for almost all the modules of the system, and if it's not possible to test all the modules, at least the most critical mast be tested. The Unit Testing mast be done first in a structural way in which the methods are tested in white-box and then completed with a functional test, by testing the various functions in black-box. For both the white-box and the black box testing it's necessary to have a good inspection document, previously made, to be sure to have enough documentation and enough comments in the code to better understand what the various part of the code and the various methods actually do. For the white-box testing the preferred coverage criterion is the edge-coverage better if a condition-coverage, and where feasible the critical functions should be tested using the path-coverage criterion. However the various tests mast be done using critical test cases for example by testing the boundary conditions. Finally we assume that the external components (Google Api) work correctly and don't need to be tested.

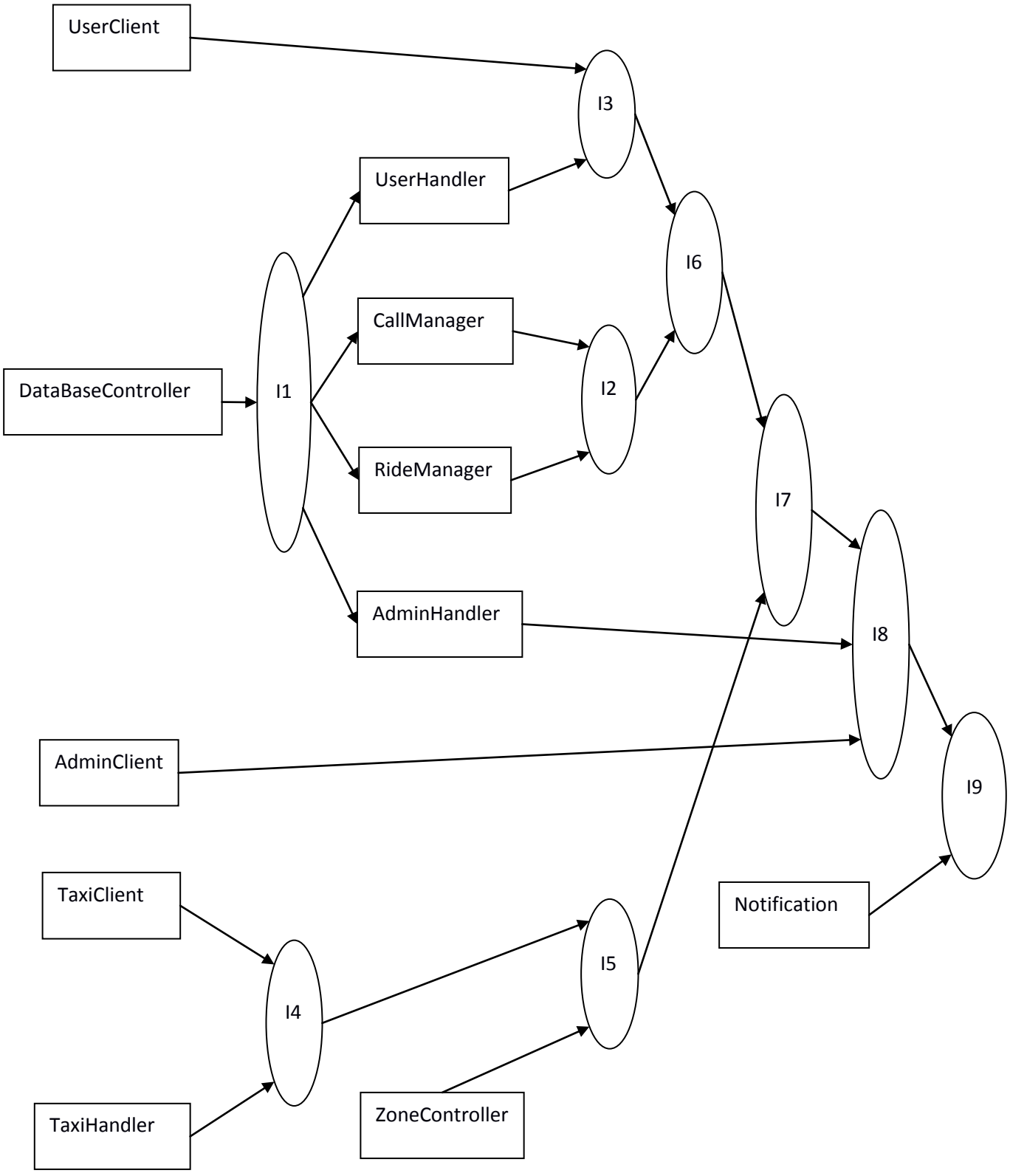
2.2 ELEMENTS TO BE INTEGRATED



2.3 INTEGRATION TESTING STRATEGY

The basic strategy that we have decided to use for the integration testing is the “Sandwich” strategy. Nevertheless, the sequence of components integration is set with respect to the functional role of the modules but still following the basic guide lines of the “Sandwich” strategy. In this way we have tried to write a testing plan that best fits with the architecture of our system, described in the Design Document. The main reason for choosing the “Sandwich” approach as basic strategy is its flexibility. In fact it allows us to extract the best benefits of both the top-down and the bottom-up strategies, in particular we can anticipate the testing of the critical modules concerning the application logic of the system and testing in parallel the client interfaces trying at the same time to decrease the number of stubs and drivers needed for the integration testing.

2.4 SEQUENCE OF COMPONENTS/FUNCTION INTEGRATION



3. INDIVIDUAL STEPS AND TEST DESCRIPTIONS

3.1 Integration test case I1

Test case Identifier	I1A
Test Item(s)	UserHandler → DataBaseController
Input Specification	Input coming from the UserHandler
Output Specification	Check if the correct read/write operations are performed on the DataBase
Environmental Needs	UserHandler Driver

Test case Identifier	I1B
Test Item(s)	RideManager → DataBaseController
Input Specification	Input coming from the RideManager
Output Specification	Check if the correct read/write operations are performed on the DataBase
Environmental Needs	RideManager Driver

Test case Identifier	I1C
Test Item(s)	CallManager → DataBaseController
Input Specification	Input coming from the CallManager
Output Specification	Check if the correct read/write operations are performed on the DataBase
Environmental Needs	CallManager Driver

Test case Identifier	I1D
Test Item(s)	AdminHandler → DataBaseController
Input Specification	Input coming from the AdminHandler
Output Specification	Check if the correct read/write operations are performed on the DataBase
Environmental Needs	AdminHandler Driver

3.2 Integration test case I2

Test case Identifier	I2
Test Item(s)	CallManager → RideManager
Input Specification	New Call forwarded to the CallManager
Output Specification	-Check if the correct path has been created and associated to the new ride. -Check that the new ride has been correctly added to the Time Schedule. -Check that the correct taxi has been associated to the Ride.
Environmental Needs	TaxiHandler stub + Google Api + I1 succeded

3.3 Integration test case I3

Test case Identifier	I3
Test Item(s)	UserClient→ UserHandler
Input Specification	UserGUI input
Output Specification	Check if the correct functions are called in the UserHandler
Environmental Needs	CallManager Stub

3.4 Integration test case I4

Test case Identifier	I4
Test Item(s)	TaxiClient → TaxiHandler
Input Specification	TaxiClient input
Output Specification	Check if the correct functions are called in the TaxiHandler
Environmental Needs	RideManager Stub

3.5 Integration test case I5

Test case Identifier	I5
Test Item(s)	TaxiHandler → ZoneController
Input Specification	TaxiHandler input coming from the TaxiClient
Output Specification	Check if the taxi is always associated in the correct way to the right zone queue.
Environmental Needs	I4 succeded

3.6 Integration test case I6

Test case Identifier	I6
Test Item(s)	UserHandler → CallManager
Input Specification	UserHandler input coming from the UserClient
Output Specification	Check if the correct operations are performed in the CallManager
Environmental Needs	I2 + I3 Succeded

3.7 Integration test case I7

Test case Identifier	I7
Test Item(s)	RideManager → TaxiHandler
Input Specification	New Call coming from the CallManager
Output Specification	Check if the ride is correctly assigned to the correct taxi. Check that all the operations on the taxis are performed in the correct way with respect to the assigned ride.
Environmental Needs	I5 + I6 Succeded

3.8 Integration test case I8

Test case Identifier	I8
Test Item(s)	AdminClient → AdminHandler → ZoneController
Input Specification	Input coming from the AdminClient
Output Specification	Check if the correct functions are called in the AdminHandler and the correct output is displayed on the Admin Gui.
Environmental Needs	I7 + I1D Succeded

3.9 Integration test case I9

Test case Identifier	I9
Test Item(s)	Notification → UserClient
Input Specification	Notification input
Output Specification	Check if the notification are displayed correctly on the UserClient
Environmental Needs	No environmental needed

4. TOOLS AND TEST EQUIPMENT REQUIRED

For the tests we would use:

- Aquilian : it is used to build in-container tests in order to check components in business application and the interaction with the system.

- JUnit : it is a Java library for testing source code. It ensures that methods in java code work as designed. Moreover it provides a set of tools that make it easier to write test drivers for the code.

- JMeter : it is a GUI desktop application designed to load test functional behavior and measure performance. It has a rich graphical interface and can be used to simulate heavy load on server, network or object to test its strength or to analyze overall performance under different load types.

- Manual testing: for the parts where no program could help.

5. PROGRAM STUBS AND TEST DATA REQUIRED

Integration test step 3.1:

In this step of integration we test the correct working of all the functionality of the DataBaseController so we need four drivers to simulate the calling components: UserHandler Driver, CallManager Driver, RideManager Driver, AdminHandler Driver.
To perform this integration step we need some initial temporary data stored in the database in order to test the correctness of the query.

Integration test step 3.2:

In this step we test the core functionality of the system so we need a Driver to simulate the user (UserHandler Driver) and a stub to simulate the taxi at which will be passed the rides (TaxiHandler Stub).

Integration test step 3.3:

In this step we chek if the the correct functions ae called in the User Handler and the right result is returned back to the user Gui. To do that we need a stub to simulate the behaviour of the CallManager.

Integration test step 3.4:

In This step we chek if the the correct functions ae called in the TaxiHandler and the right result is returned back to the taxi Gui. To do that we need a stub to simulate the behaviour of the RideManager.

Integration test step 3.5:

In this step we test that the correct operations are performed on the queues, so we only need a driver for the RideManager to check that all the requests coming from the RideManager for a taxi in a certain zone causes the correct function calls in the TaxiHandler and the ZoneController. If the step 3.4 is terminated with success we can check without any driver or stub that all the movements of the taxis through the zones and all the requests about the availability of the taxis are correctly reported in the queues of the corresponding zones.

Integration test step 3.6:

In this step we test that all the inputs coming from the cause the correct function calls in the CallManager. Thanks to the Sandwich strategy, if the steps 3.2 and 3.3 are terminated with success we don't need any other stub or driver.

Integration test step 3.7:

In this step we test that all the calling coming from the RideManager produce the right function calls on the TaxiClient and the Zone Controller and that the output returned back in correct.
Thanks to the Sandwich strategy, if the steps 3.5 and 3.6 are terminated with success we don't need any other stub or driver.

Integration test step 3.8:

In this step we test that all the information required from the AdminClient produce the correct function calling in the AdminHandler and gets back the correct output displayed on the Admin Gui. The TaxiHandler uses two components already integrated: DatabaseController and the ZoneController, so we don't need any other driver or stub.

Integration test step 3.9:

In this step we test the correct integration between the notification component and the UserGui component, all the necessary component have been already integrated so we don't need any other driver or stub.

APPENDIX

Software and tools used

- Microsoft Office Word 2013: to write this document.
- Github

Hours of works

- Rosmini Simone: ~ 10 hours.
- Viscusi Vincenzo: ~ 10 hours.
- Zambelli Matteo: ~ 10 hours.