Cascianelli Silvia

Baldi Roberto

INTEGRATION TEST PLAN

V 1.0

MyTaxiService

CONTENTS

1 INTRODUCTION

1.1 REVISION HISTORY

Document Title	Integration test plan
Authors	S.Cascianelli, R.Baldi
Version	1.0
Document Status	internally accepted

Version	Date	Author(s)	Summary
0.1	13-01-2016	S.Cascianelli	Document Creation
0.2	15-01-2016	S.Cascianelli, R.Baldi	Draft
0.3	17-01-2016	S.Cascianelli, R.Baldi	Second Draft
1.0	20-01-2016	S.Cascianelli, R.Baldi	Internally accepted

1.2 PURPOSE AND SCOPE

This document defines the strategies and the global plan for testing the integration of the constituent components of MyTaxiService project, described in the Design Document. In effect, the Integration Test Plan arranges and design the phase in software testing in which individual software modules are combined in larger aggregates and tested to exercise their mutual interactions and to verify the functions performed through their reciprocal interfaces. Since this phase of testing occurs actually after unit testing and before system testing, the Integration Test Plan supposes to take as its input modules that have been already unit tested; furthermore it aims at assembling progressively the components of interest, submitting the so-obtained grouping to suitable tests, up to deliver as output the completely integrated system, ready for system testing.

As described also in the other reference documents, the project we will focus on is called MyTaxiService and it concerns the design of a system which is able to manage conveniently and efficiently the taxi service of a large city. Hence this project aims principally at simplifying the access of passengers to the service by exploiting the Net, but it also guarantees a fair management of taxi queues. To be more precise, MyTaxiService grants a potential passenger the possibility of requesting a taxy either through a web application or a mobile one and it assures the applicant to be informed about the code of the incoming taxi and the waiting time, provided that the system succeeds in allocating a taxi for the request. Moreover MyTaxiService provides also the opportunity to demand for a reservation, as long as the booking occurs at least two hours before the expected ride; in this case the passenger must specify origin and destination of the ride as well as the meeting time during the booking process. In this way, the system will proceeds automatically to allocate a taxi for the reservation 10 minutes before the scheduled time; at that point, the applicant receives a confirmation with the code of the incoming taxi and any information concerning an eventual delay. However, if there are not available taxis the allocation procedure fails also for an already booked reservation; in this eventuality the negative outcome associated with the reservation is reported to the applicant through an email or a notification on the app, to apologize for the disruption. Furthermore, essentially MyTaxiService arranges every available taxi in a specific queue associated with the zone of the city the taxi is located at that moment. Thus, the system can always forward an incoming request to the first taxy of the appropriate queue, according with the zone the origin of the ride belongs to and by using the special functionalities of the mobile app dedicated to drivers. Whether the chosen taxi driver does not accept the sent request MyTaxiService records a penalty for him and puts his taxi at the bottom of the local queue; at the same time the systems proceeds by forwarding the request to the following taxi, which has just became the first of the queue. If none among the first three chosen drivers accepts the request the system forces the fourth one with a mandatory service call. In fact MyTaxiService is designed to be efficient and so it cannot let a request missed unless there are not available taxi both in the local queue and in all the adjacent zones. In fact, if an incoming request refers to a zone in which the queue of available taxis is momentarily empty, MyTaxiService provides a taxi by forwarding the request to the longer queue among the ones of the adjacent zones. To conclude this argument, it has to be noted that each queue is scrolled cyclically and so a driver could receive the same request more than once and finally should be forced to accept it. Finally, as far as taxi driver are concerned, each of them must use the mobile app, logging in at the beginning of each shift; actually through the app a driver can notify the availability every time a ride has been concluded and the system can place it in the opportune queue depending on its actual GPS position. Besides of this a taxi driver has to use the mobile app to accept an incoming call, which the system has conveniently sent to him, and also to signalize the occurrence of an urgent problem. Hence, in general, MyTaxiService works in order to handle efficiently the incoming taxi requests. All the requirements, assumptions and constraints as well as the design choices and solutions that allow the system to fulfill all its objectives have been presented both in the Requirement Analysis and Specification Document and in the Design Document.

1.3 List of Definitions and Abbreviations

Taxi driver: a kind of user that, through the mobile app, can access his/her private area and utilize special functionalities of MyTaxiService, usable only by this type of user.

Passenger: the other kind of user, that can utilize MyTaxiService both through the mobile app or the web application; in particular these users are interested in exploiting the application to look for a taxi easily and quickly.

Local queue: This term refers to the organization of the city in different zone to cover with the taxi service; in effect a local queue is supposed to be the ordered set of available taxi associated to a specific zone; in particular this relation is based on the GPS current position of each available taxi.

Request: This term indicates that a passenger is requiring a ride because he/she needs a lift at the current moment

Reservation: This term indicates a ride that a passenger want to book for a certain moment successive with respect to the sending time. Precisely the required time must be at least two hours later the booking.

Unsuccessful call: This terms indicates a call forwarded by the system to a chosen taxi driver, who simply does not answer and consequently does not accept the ride associated with that call.

- DD: Design Document.
- RASD: Requirements Analysis and Specification Document.
- DB: Data Base.
- DBMS: Data Base Management System.
- API: Application Programming Interface.
- JEE: Java Enterprise Edition.
- ITP Integration Test Plan

1.4 LIST OF REFERENCE DOCUMENTS

- Specification Document: MyTaxiService Project AA 2015-2016.pdf.
- MyTaxiService Project: Requirement Analysis and Specification Document
- MyTaxiService Project: Design Document
- DOCUMENTATION OF TOOLS

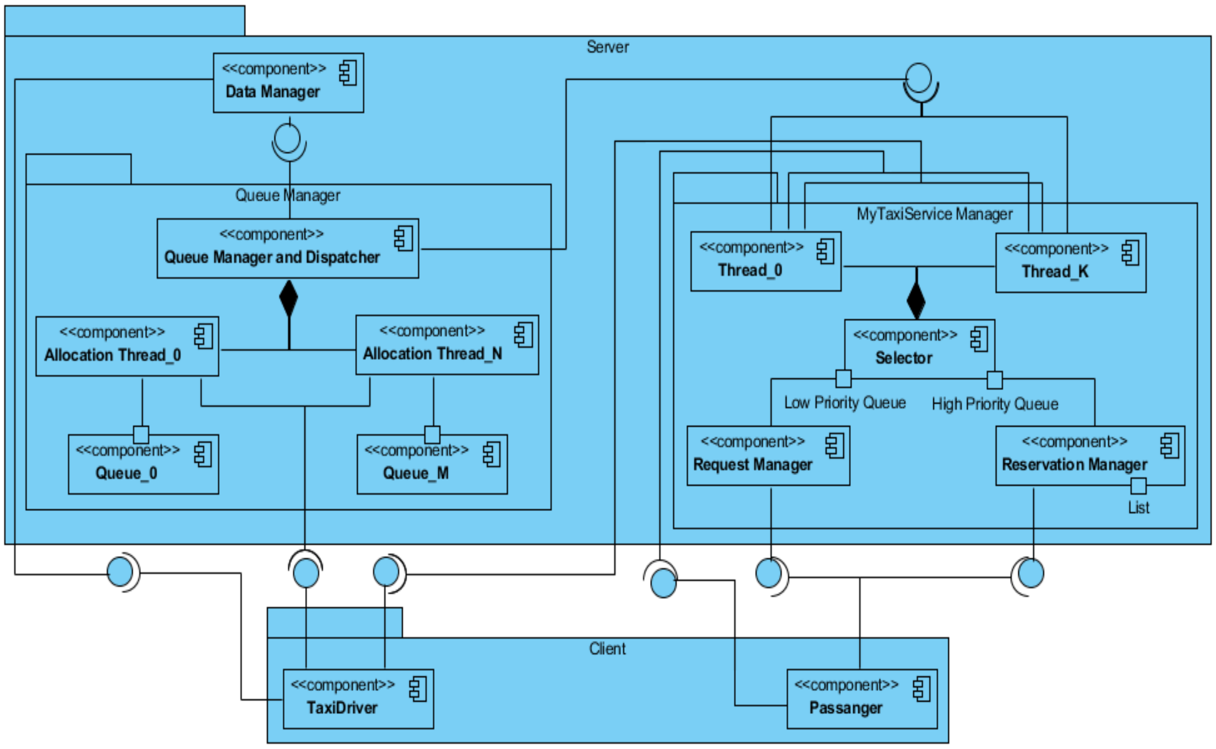
2 INTEGRATION STRATEGY

2.1 ENTRY CRITERIA

Prior to introducing all the modules of MyTaxiService design that will be integrated and tested according with the plan provided in this document, it is important to clarify which are the preconditions to satisfy before starting this phase of testing. In effect, since all the tests suggested in this Integration Test Plan have been thought in order to check the interactions among modules and assess they behave as expected, the essential prerequisite is assuring that all the functions of the system have been widely unit tested. Hence, the proposed plan can be eventually carry out if and only if the implementation code developed for MyTaxiService project has already been extensively tested in terms of single units, components and functionalities. Moreover it is reasonable to consider that the system has been also subject to different kind and stages of static analysis, as manually inspections concerning the examination of specification, the design, the provided documentation and even the code itself.

2.2 ELEMENTS TO BE INTEGRATED

The modules to be integrated and tested are the components that we have presented in the Design Document of MyTaxiservice project. These constituent parts are showed in the following picture, extracted from the Design Document indeed.



Meaningful Subsystem	Components or subcomponents
Client	• Taxi Driver • Passenger
MyTaxiService Manager	• Request Manager • Reservation Manager • Selector • Each Thread taking care of a single request/reservation
Queue Manager	• Queue Manager and Dispatcher • Each Allocation Thread
Data Manager	

2.3 Integration Testing Strategy

The proposed strategy for the Integration tests combines different kind of approaches. First of all a bottom up tactic based on the hierarchical structure of MyTaxiService project. But at the same time, the offered Integration Test Plan suggests also to use two other different functional approaches. One of these takes into account all the modules involved in the operations related to the meaningful threads, that we have already described in the Design Document as the principle responsible for the handling and allocation of cab requests or reservations. The other functional strategy instead, is based on grouping components with respect to their collaboration into the accomplishment of some specific activities. Anyway, all the adopted choice for module integration are listed in the next paragraph, whereas the following section of the document provides detailed description of the suggested test in order to clarify their goals as well as the chosen approaches.

2.4 Sequence of Component/Function Integration

2.4.1 Software Integration Sequence

In this first paragraph we have provided a list of tests involving different components within a single subsystem; hence we have supposed that their global interactions have not been already verified during the unit test phase.

ID	INTEGRATION TEST	SUBSYSTEM
1.1	• Request Manager • Reservation Manager • Selector	MyTaxiService Manager

1.2	• Request Manager • Selector • Handler Thread	MyTaxiServiceManager
1.3	• Reservatiom Manager • Selector • Handler Thread	MyTaxiServiceManager
1.4	• Queue Manager and Dispatcher • Allocation Thread	Queue Manager

2.4.2 Subsystem Integration Sequence

In the following table we have specified the tests concerning the integration of different subsystems or of portions belonging to different subsystems.

ID	INTEGRATION TEST	SUBSYSTEM
2.1	• Passenger • Request Manager	Client, MyTaxiService Manager
2.2	• Passenger • Reservation Manager	Client, MyTaxiService Manager
2.3	• Handler Thread • Queue Manager and Dispatcher • Allocation Thread	MyTaxiService Manager, Queue Manager
2.4	• Data Manager • Queue Manager and Dispatcher	Data Manager, Queue Manager

2.5	• Data Manager • Taxi Driver	Data Manager, Client
2.6	• Taxi Driver • Allocation Thread	Client, Queue Manager
2.7	• Taxi Driver • Handler Thread	Client, MyTaxiService Manager
2.8	• Handler Thread • Taxi Driver	MyTaxiService Manager, Client
2.9	• Passenger • Handler Thread	Client, MyTaxiService Manager

3 Individual Steps and Test Description

Test identifier	1.1
Test items	Request Manager, Reservation Manager → Selector
Purpose	This test verifies that each request or reservation, created respectively by the Request Manager or the Reservation Manager, is correctly inserted in one of the two priority queues, accessible to the Selector.
Environment needs	A driver is needed to ask the Manager to create a request/reservation as a Passenger had asked for it

Test identifier	1.2
Test items	Request Manager → Selector → Handler Thread
Purpose	The goal of this test is to control that the Handler Thread (associated with a request through the Selector) contains all the information about the required ride; this information must be the same that the Request Manager has previously collected and used to create the request itself.
Environment needs	It needs that the test 1.1 has successfully ended, in such a way that a request is already in the proper queue.

Test identifier	1.3
Test items	Reservation Manager → Selector → Handler Thread
Purpose	This test controls that the Handler Thread (associated with a request through the Selector) contains all the information about the booked ride; this information must be the same that the Reservation Manager has previously collected and used to create the reservation itself. In addition, it must also control that the selection of the reservation of interest and the association with a Handler Thread takes place at the right allocation time, i.e. ten minutes before the arranged time for the ride.
Environment needs	It is necessary that the test 1.1 has been already executed, to be sure to find a reservation in the high priority queue.

Test identifier	1.4
Test items	Queue Manager and Dispatcher → Allocation Thread
Purpose	The aim of the test is to verify that the Allocation Thread works properly with the information received about a ride. Furthermore it controls that the Allocation Thread looks for a Taxi Driver exclusively in the local queues belonging to a certain area; hence it verifies the search is concentrated in the zone of the starting point for the ride or at most in the adjacent zones.
Environment needs	It is necessary to use a driver to play the role of a handler Thread requiring the allocation of a hypothetical ride with all the related data.

Test identifier	2.1
Test items	Passenger → Request Manager
Purpose	First of all the test has to attest that information sent by the client are correctly received by the Request Manager. Moreover, the test controls that the Manager accepts to create a request, as the client has required, if and only if this client of kind Passenger has inserted correct information in the compilation of the proper form; this information must be coherent with what the blank spaces indicate to fill in, and must respect all the constraints the Manager applies to avoid replicated or fake requests.
Environment needs	

Test identifier	2.2
Test items	Passenger → Reservation Manager
Purpose	The test has to confirm that information sent by the client are correctly received by the Reservation Manager. Moreover, the test controls that the Manager accepts to create a reservation if and only if the applicant client of kind Passenger has inserted correct information in the compilation of the proper form; this information must be coherent with what the blank spaces indicate to fill in, and must respect all the constraints the Manager applies to avoid fake or not acceptable reservations (for example, two incompatible reservations from the same applicant or a reservation with unacceptable starting time).
Environment needs	

Test identifier	2.3
Test items	Handler Thread → Queue Manager and Dispatcher → Allocation Thread
Purpose	First of all this test must control that the Queue Manager and Dispatcher instantiates the correct number of Allocation Thread (i.e. one or two) according with the option specified among the information about the ride. Furthermore, this test is very important for assuring that the corresponding Allocation Thread(s) receive(s) correctly all the data about the ride to be allocated. Finally it has to verify that as soon as a Taxi Driver accepts to take care of the ride of interest, the Handler Thread receives the correct information about this Taxi Driver.
Environment needs	The test 2.6 has to be performed before this test to guarantee that the communication between the Taxi Driver and The Allocation Thread is perfectly functioning. Furthermore, before starting the execution of the test, it is necessary to use a driver to launch the handler thread with suitable information about a hypothetical ride.

Test identifier	2.4
Test items	Data Manager → Queue Manager and Dispatcher
Purpose	This test has the objective to prove that all the changes of interest for the Queue Manager and Dispatcher and occurring into the database are properly made known to this Queue Manager and Dispatcher; in this way the local queue data structures are conveniently updated with correct information every time it is necessary. For example this situation can be tested modifying the position of an available Taxi Driver, since this data are treated by the Data Manager but represent also information of interest for the Queue Manager and Dispatcher (that actually must update the involved local queues).
Environment needs	This test requires a stub that simulates the role of a GPS tracker and transmits plausible location information

Test identifier	2.5
Test items	Taxi Driver → Data Manager
Purpose	This family of tests has to control that the Data Manager correctly takes into account and saves in the database the outcomes of the operations performed by the Taxi Driver. Hence, the following actions are tested with suitable tests:

	• Login 2.5.1 • Change state 2.5.2 • Logout 2.5.3 In particular, the 2.5.2 may be also decomposed into different tests if it is necessary to compensate for any lack in the Unit Tests.
Environment needs	

Test identifier	2.6
Test items	Allocation Thread → Taxi Driver
Purpose	This test must assure that the connection from the Allocation Thread to the Taxi Driver is stable; the only interaction to perform is associating a ride (actually a request or a reservation) to a Driver. So, the test must confirm that when the Taxi Driver accepts an assignment the Allocation Thread is informed and stops the search; moreover, the Allocation Thread has to send the correct information about the ride to the Taxi Driver before notifying the positive outcome of the allocation. The test must also consider the case in which the chosen Taxi Driver refuse the assignment (actually ignoring the call); in this situation the test verifies that the Allocation Thread goes on searching and forwarding the assignment to another candidate. Finally the test must also take into account the possibility of a mandatory call that forces the involved Taxi Driver to accept the assigned ride; also in this case the test has to control the right management of the situation.
Environment needs	To launch the Allocation Thread is needed a driver that accomplish this job, simulating the role of the Queue Manager and Dispatcher and providing information about a hypothetical ride.

Test identifier	2.7
Test items	Handler Thread → Taxi Driver
Purpose	This test has to control that, when a just allocated ride is cancelled, the involved Taxi Driver is immediately informed.
Environment needs	The test suppose the Handler Thread has a reference to a Taxi Driver chosen for taking care of a hypothetical ride; for this reason the test must follow a successful execution of the test 2.3. In this way, the test 2.3 assures also that the handler thread has been already launched by a driver, performing the functionality of the Selector. Furthermore this test requires another driver to simulate the role of the involved Passenger that is responsible for calling off the just allocated ride.

Test identifier	2.8
Test items	Taxi Driver → Handler Thread
Purpose	The test needs to control the communication between Taxi Driver and Handler Thread in order to be sure that the Handler Thread correctly receives information about possible problems of the Taxi Driver, that force to give up the ride . In effect, as consequence, the handler Thread will communicate this information to the Passenger and will start a new allocation procedure for the same ride.
Environment needs	The test must follow a successful allocation procedure to minimize the use of drivers; for this reason the test is preceded by the execution of the test 2.3. In this way, the test 2.3 assures also that the handler thread has been already launched by a driver, performing the functionality of the Selector. This test needs that the communication between the Passenger and the Handler Thread has been already properly tested through the test 2.9.

Test identifier	2.9
Test items	Handler Thread → Passenger
Purpose	This test has to control that all the communications from the handler Thread to the Passenger are correctly received without errors. It must test all the kind of communication: • the receipt of a reservation; • the confirmation of a positive outcome for the allocation of a request/reservation; • the negative outcome for the allocation of a request/reservation • a possible notice of a delay; • a possible re-allocation for the ride whether any problem involves the associated Taxi driver.
Environment needs	Also in this case a driver is necessary to launch the Handler Thread and to perform the test in a correct way. Furthermore, the Handler Thread needs correct reference to the involved Passenger, as the tests 1.2 and 1.3 have to verify.

4 TOOLS AND TEST EQUIPMENT REQUIRED

The architecture needed to perform the proposed integration tests can be built in a virtual space; for this reason the physical equipment can be reduced to only few computers. We suggest to exploit Arquillian, an integration testing platform developed for Java EE. In effect, this framework empowers the developers to write integration tests for business objects that are executed inside a container or that interact with the container as a client. In particular, Arquillian grants the opportunity to test cases against a container, which could be remote or embedded. A container may be, for instance, a Java EE Application Server as GlassFish the open-source one we have supposed to use in the Design Document for MyTaxiService project. Hence in our case, Arquillian should permit to test component interactions against GlassFish and all its related APIs, such as for example Java Servlet, JavaServer Pages, Enterprise JavaBeans, Java Persistence API, Contexts and Dependency Injection, Java Message Service, etc. Another positive aspect of Arquillian is that it integrates transparently with familiar testing frameworks, such as JUnit, allowing tests to be launched using existing IDE and Maven test plugin; finally Arquillian has been realized with the goal of making integration testing no more complex than writing basic unit test, hence it is quite simple to be used.

5 PROGRAM STUBS AND TEST DATA REQUIRED

Since this Integration Test Plan aims at providing the outline for component testing that may be done in isolation with the rest of the system, drivers have been widely used to replace the missing software and simulate the interface between the software components in a simple manner. Anyway, their presence and use have been already discussed in the Test Description section, in each case they are necessary to perform correctly a test. Finally it is important to consider that some test cases include also methods that treat data deriving from a GPS tracker. Hence, assuming the interaction between the Data Manager and this device has been properly tested, on the scope of the proposed integration tests the GPS could be virtualized through a stub that is able to provide plausible information.