myTaxyService

Integration Test Plan Document

Version 1.0

Matteo Farè, Federico Feresini, Thierry Ebali Essomba

January 20, 2016

Contents

Chapter 1

Introduction

1.1 Purpose

This document provides the testing plan for the integration of all the parts of the MyTaxiService system and has the purpose to lead development, testing and integration of the different components according to the guidelines.

1.2 Scope

The scope of all the tests is to ensure the proper functioning of the entire system. A list off all of them will be presented below, with descriptions of the order of implementation of the various components and interfacing methods of them. The aim is to trace a path of efficient work that allows to optimize the results and the time of realization.

1.3 Definitions and Abbreviations

- RASD: Requirements Specification Analysis Document
- DD: Design Document
- DBMS: Data Base Management System
- DB: Data Base

1.4 Reference Documents

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

Chapter 2

Integration Strategy

2.1 Entry Criteria

The following subsection describes the previous verification to apply before starting the integration test. Validation of the requirements is ensured by unit testing of each module of the system. This is completed and performed before the code is added to the system. Unit testing consists of verifying the interfaces which allows data to properly flow from an object to another and that are proper for storing data. The developing of the unit tests also aims to guarantee a minimal automated testing environment, to ensure that each revision to the code can be tested immediately without loss of time. This should make easier error detection operations and it allows to avoid general errors resulting from a big-bang approach.

2.2 Elements to be Integrated

The components that we defined in the DD (for a more precise description of them refer to Chapter Two of DD) are clearly the different components of the system that need to be integrated with each other. The integration testing involves all the code modules developed. Furthermore the end-user application has to be tested in the various system environments. Below is reported the components list:

- Registration Manager
- Customer Login Manager
- Driver Login Manager
- Customer Notification Manager
- Customer Decision Manager

- Driver Notification Manager
- Driver Decision Manager
- Immediate Ride Request Manager
- Booking Ride Request Manager
- Sharing Ride Request Manager
- Booking Scanner
- Rides Sharing Viewer
- Payment Manager
- Availability Manager
- Position Tracking Manager
- Queues Manager
- Top Queue Extractor
- Users Central DBMS
- Rides Central DBMS

2.3 Integration Testing Strategy

The integration test allows to identify the problems occurring when two units interact. Thanks to a previous unit testing on each single module of the system, probably each observed problem is caused by the interface which manages the communication between the modules. This approach makes the integration testing analysis easier. The chosen integration testing strategy is the bottom-up approach. It firstly requires the integration of the units at lower levels. This strategy, starting form the lowest level units, provides a reduction of needed stubs. Moreover the bottom-up provides a good strategy for testing modules based on a hierarchical project structure. The disadvantage consists in the necessity of a high number of drivers which complicates the management of integration testing. The general idea is to start the integration from components of lower level, such as the DBMS, with the components that feed or receive data from database, and then proceed with the integration of the components of the highest level, close to the end user. In practice, after the testing of the individual modules, we have to make sure that the different DBMS to interact properly with the modules above them. They has to populate the different Databases with several fake values, possibly received in input from a appropriate driver (for examples

the Central Users DBMS has to receives input from a driver that simulates the functions of generation of a string containing user data) and also they needs a driver that gives them a data request to ensure the function of supply to the overlying modules the required data. While the bottom-up strategy provides us a simple way to go back over the flow of data along the subcomponents of 'MVC model, inside them, and mainly in the model, there is the need to identify subgroups of components to be tested together, and later expand the coverage of the testing going to integrate between them an increasing number of subgroups.

2.4 Sequence of Component

2.4.1 Software Integration Sequence

The development and the subsequent integration of components will follow the steps below:

1. The first step involves the development of different DBMS. The system, being provided with databases that contain different data formats, will require the appropriate DBMS to access, manipulate the content and extract the necessary information. the User's Central DB will be managed by a DBMS that can receive a string containing the data for each user and stores it in the database, to provide answers to queries from the Log Manager and ensure the properties of consistency and data integrity. For the Rides Central DB there will be an appropriate DBMS, able to recognize the type of ride relative to each string received as input. It also has to be integrated with Rides Sharing Viewer and Booking Scanning, that initially can be simulated by the use of appropriate stubs.

2. The following components that will be implemented are the Registration Manager and Login Manager (they can be implemented in parallel). Registration Manager needs a driver to give him input a valid registration request, which could lead to the next generation of an input to the DBMS. It will also need to develop a stub to simulate the sending by the component to the user interface of the answer as the epilogue of the operation. The same pattern of development and testing will be followed by the Login Manager. it should be noted while the Registration Manager is to use the functions to access and write data on the DBMS, the Log Manager provide the necessary information to the DBMS for the formulation of a query, whose response it will be returned.

3. Customer Model: Here can be identified three subgroups to manage and develop in parallel.

(a) Immediate Ride Request Manager

(b) Booking Ride Request Manager – Booking Scanner

(c) Sharing Ride Request Manager – Rides Sharing Viewer

The interactions between the different subgroups and with the controller will initially be tested with the use of stubs, while the details relating to the various types of ride requests will be provided using suitable drivers.

4. Driver Model: Here can be identified two subgroups, one of the highest level and a lower one. The development of the two subgroups is to be performed following the bottom up approach.

   (a) Availability Manager – Position Tracking Manager

   (b) Queues Manager – Top Queue Extractor

   The first group receives input directly from the driver interface, therefore will be need appropriate drivers for the passage of acceptable values. The integration between Position Tracking Manager and the Google Maps API has to be taken in account and carried out. Since this SOA service is already on the market, there is no need of drivers or stubs to test this integration. As this subgroup is developed later, it will not be necessary stubs for the functions of interaction with the other subgroup. As regards the second group, the Queues Manager needs of the stubs for the functions of updating the code in relation to decisions of the drivers and their availability, while the Top Queue Extractor needs one stubs for communication with the controller.

   Customer model and Driver model proceed in their development in parallel. Functionalities of the whole Model will be tested through the using of view and controller drivers. This will give an overview on the main issues to solve in the controllers and views implementation.

5. Controllers and Views components will be developed and tested one by one after the development of the whole model.This allows to know clearly what are the inputs requested by the model. They will be the result of the integration and a deep amplication of the drivers used in the model testing. At the end of this process, when the Controllers denitely meet the entry criteria and one at time, they will take the drivers place, and will pass thorough the integration test. View will be developed upon the Controllers integration testing completion. As well as latter, they will be the result of the drivers integration and application. At the end of this process they will take take the drivers place, and will pass thorough the integration test.

6. On completion of these phases, the entire system will going thorough a general integration system test, that will cover:
   (a) Whole System Testing.
   (b) Performance testing.
   (c) Ultimate features testing, exploiting the main scenarios in which the system has to work
7. Release of a beta version of the product for a first phase of testing by the users.

2.4.2 Subsystem Integration Sequence

In agreement with what has been stated previously the subsystems that make up the application will be integrated according to the following order:

1. Model functionalities
2. Controllers
3. Views

Chapter 3

Individual Steps and Test Description

Here is provided the description of the test cases we think are necessary in order to develop the system.

3.1 Integration Test T1

ID	T1
Components Involved	Registration Manager, Login Manager, Central DBMS.
Description	A fake user tries to register and to log in the system.
Expected Result	The personal data of the user have to be checked and saved into the DB. When he wants to log in the system, his data are retrieved from the DB and his personal page is shown.
Failure Condition	The system displays an error message instead of the user personal page.

3.2 Integration Test T2

ID	T2
Components Involved	Immediate Ride Request Manager, Queue Manager, Top Queue Extractor
Precondition	A set of queue is filled up with fake available drivers.
Description	A request for an immediate ride is simulated, in order to test if the system is able to choose the right queue and to send the request to the first driver in it.
Expected Result	The first driver in the queue of the zone chosen by a customer receives the ride request.
Failure Conditions	The Queue Manager doesn't select the right queue. The Top Queue Extractor doesn't send the request to the first driver.

3.3 Integration Test T3

ID	T3
Components Involved	Booking Ride Request Manager, Booking Scanner, Immediate Ride Request Manager, Rides Central DBMS.
Description	The data of a booked ride are saved in the DB, then the Booking Scanner checks in the DB if the ride is imminent. When this conditions becomes true, it sends the information to the Immediate Ride Request Manager.
Expected Result	The Booking Scanner calls the methods of the Immediate Ride Request Manager.
Failure Condition	The Booking Scanner doesn't activate the functions of Immediate Ride Request Manager.

3.4 Integration Test T4

ID	T4
Components Involved	Sharing Ride Request Manager, Sharing Rides Viewer
Preconditions	Some fake shared ride are created and saved in the DB.
Description	A fake user asks for the list of the available shared rides that are starting from his position.
Expected Result	The Sharing Ride Request Manager sends the requested ride data to the Sharing Rides Viewer, that scan the DB and finds the list of all the available shared rides for that customer.
Failure Condition	The list present some wrong ride or other available rides are missing.

3.5 Integration Test T5

ID	T5
Components Involved	Availability Manager, Position Tracking Manager, Queues Manager.
Description	A mock driver change his availability status.
Expected Result	When the availability is set to "true" the Position Tracking Manager finds the driver new position, calls the methods of the Queues Manager that push him at the bottom of the right queue.
Failure Condition	The driver is put in the wrong queue.

3.6 Integration Test T6

ID	T6
Components Involved	Driver Decision Manager, Payment Manager
Description	A customer accepts a ride.
Expected Result	The Payment Manager has to redirect the customer to the external system in order to pay the ride.
Failure Condition	The customer isn't redirected to the external system.

3.7 Integration Test T7

ID	T7
Components Involved	Driver Decision Manager, Driver Notification Manager, Queues Manager.
Precondition	A fake customer has requested an immediate ride.
Description	The Driver Notification Manager sends to a driver the ride request. The Driver Decision Manager sends back the driver decision.
Expected Result	If the driver rejects the ride, the Queues Manager has to find a new driver from the queue. Otherwise the customer has to receive the informations of the available ride.
Failure Condition	The information flow is lost through the components.

3.8 Integration test T8

ID	T8
Components Involved	Customer Notification Manager, Customer Decision Manager
Preconditions	A customer has asked for an immediate ride, and a driver has already accepted it.
Description	The Customer Notification Manager sends to the customer the informations of the available ride. When he accepts or rejects it the Customer Decision Manager sends the decision to the driver.
Expected Result	The customer receives correctly the ride informations. The driver receives the customer decision.
Failure Condition	The informations don't arrive to the users.

Chapter 4

Tools and Test Equipment Required

4.1 Test Management

Test management process is used to ensure an high quality software and hardware product. It helps the development and maintenance of the product during the course of project and makes the developers sure that there are few coding faults. The tool used to perform the Test Management is the following:

- QAComplete - It's a tool to manage, plan, organize and execute all test cases. It also use reports to make the test coverage more understandable.

4.2 Functional Testing

Functional Testing is a techniques of testing that ensures the system performs correctly according to design specifications. It tests core applications functions, installations, text inputs and verifies if the application is fully functional. The tools used to perform the Functional Testing are the following ones:

- 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 ad designed, without need to setup the entire application. Moreover it provides a set of tools that make it easier to write test drivers for the code.

4.3 Load Testing and Stress Testing

Load testing is a test about the performance of the system. It applies ordinary stress to the system to see if it can perform ad intended under normal conditions. It ensure that a function or a system can handle what it's designed to handle. Stress Testing measure the strength of the system by applying a big stress to the system. The tool used to perform the Load Testing and Stress Testing is the following:

- JMeter - A GUI desktop application designed to load test functional behaviour and measure performance. It ha a rich graphical interface and can be used to simulate heavy load on server, network or object to test its strength or to analyse overall performance under different load types.

4.4 Other Tools

This subsection contains a list of tools used during the development of the system:

- OpenSSL - It provides a robust toolkit for the Transport Layer Security (TLS) and a Secure Sockets Layer (SSL) protocols. The library implements cryptographic algorithm to provide a secure communication.
- Vertabelo - It is a visual database design tool available through a web browser. It also offers SQL generation, team work support and import/export in XML format.

Chapter 5

Program Stubs and Test Data Required

In this section are described stubs and data test that have been used in the different phases of the integration.

5.1 DBMS Integration

- 1

    1. Stubs: not used
    2. In this initial phase databases, in this instance Users Central DB, Queues DB and Rides Central DB, are integrated with their DBMS, therefore it will be needed a series of inputs to verify the correct management by the DBMS. As described in the previous section, tools like Aquilian allow you to check that DB works well.

- 2

    1. Stubs: User Interface Stub, Customer Interface Stub, Driver Interface Stub
    2. In this phase the Users Central DBMS must be integrated with the components that manage the user requests for registration or login (User Registration Manager, Customer Login Manager, Driver Login Manager). The test data required consist in a set of fake registrations or login requests that must be properly managed by them before they require database operations. Stubs listed above are also necessary to test the response functions of the components to the user interfaces. These stubs must return a value with meaning to describe the ending of the operation.

5.2 Customer Model Integration

- 3

    1. Stubs: not used.
    2. As in the previous phase, also in this phase there is the integration between the model's components and Rides Central DBMS. Being tested only the functions of interaction with the DBMS are not necessary stubs, while the data test necessary are limited to simulations of requests for the different types of rides. The components under consideration (Immediate Ride Request Manager, Booking Ride Request Manager and Sharing Ride Request Manager) have to control the input and decide whether to save the request or not.

- 4

    1. Stubs: Immediate Ride Request Stub
    2. In this phase Booking Scanner has to be integrated with the DBMS. The test data They are provided by the database, previously populated with fictitious values while the Immediate Ride Request stub is necessary to test the transition of the set of rides identified by the component to the manager of immediate rides.

- 5

    1. Stubs: Customer Interface Stub
    2. In this phase Rides Sharing Viewer has to be integrated with the DBMS. The data are provided by the database in this case too and stub listed above is necessary to test the transition of ride's set, that matches with the criteria provided by the Sharing Ride Request Manager, to the Customer Interface.

- 6

    1. Stubs: Queue Manager Stub, Customer Notification Manager Stub, Customer Interface Stub
    2. This stage involve the integration of the different sub-modules of the customer model. As in previous case the test data are represented by a set of fictitious ride requests that the components in question should properly handle. Queue Manager Stub is necessary to test the the proper functioning of Immediate Ride Request Manager forwarding function of requests. Notification Manager Stub serves to test that the Booking Ride Request Manager is able to send a response regarding the outcome of the reservation, while

Customer Interface Stub is necessary for the functions previously described. Driver Model Integration

- 7

  1. Stubs: Driver Notification Manager
  2. In this phase the lower sub-module of driver model has to be integrated with the Queues DBMS and with the Driver Controller. The DB is already filled with fictitious values and the Queues Manager has to update the data in relation to the input it receives. Test data can be a possible change of availability on the part of a driver or the arrival of a new request to be assigned (simulated with a driver). Top Queue Extractor receives input from the Queues Manager (request and driver code) and to test his forwarding function of the request to the correct driver is necessary a Driver Notification Manager Stub to obtain a resume value of the operation.

- 8

  1. Stubs: not used
  2. In this phase the higher sub-module of driver model has to be integrated with the lower lever. The test data consist in a set of changes of availability connected to a set of position provided by Google API in random way (however, limited to the city of interest). Using a real API is not necessary to create a stub for this function.

5.3 Customer Controller Integration

- 9

    1. Stubs: Customer Interface Stub
    2. In this phase the Customer Controller (Customer Notification Manager & Customer Decision Manager) must be integrated with the model. Test data are a set of customer decisions about a step of request process. A Customer Interface Stub is also necessary to test the notification function of Controller.

5.4 Driver Controller Integration

- 10

    1. Stubs: Driver Interface Stub
    2. As in the previous test, here the Driver Controller must be integrated with the model. The test data are the same of the previous one and also the use of the stubs.

5.5 Views

- 11

    1. Stubs: not used.
    2. In this final stage the different views must be integrated with the underlying system. No stubs are used because all the system is already integrated, and the test data are provided directly by the views.