Integration Test Plan Document
Version 1.0

Losio Davide Francesco, Luchetti Mauro, Mosca Paolo

January 16, 2016

Contents

1 — Introduction

1.1 Revision History

- January 2016 - Integration Test Plan Document: Version 1.0

1.2 Purpose and Scope

This document provides the testing plan for the integration of the whole MTS system. Components implementation order, interfaces between them and tests that must be executed are stated here. This document should be used as a reference of the testing methodologies used by the MTS team. The scope is the creation of a workflow sequence of operation that allows the testers to work optimizing the available time.

1.3 List of Definitions and Abbreviations

- **MTS**: **M**y**T**axi**S**ervice
- **RASD: R**equirements **S**pecification **A**nalysis **D**ocument
- **DD: D**esign **D**ocument

1.4 List of Reference Documents

- **MyTaxyService RASD**
  - November/6/2015-
- **MyTaxyService DD**
  - December/4/2015-
- The project description
- The documentation of any tool you plan to use for testing

2 — Integration Strategy

2.1 Entry Criteria

Before the beginning of the procedures stated in this document, it is mandatory to develop suitable unit tests for each class. Furthermore, this tests, have to be implemented during the class development in order to ensure their correction and to certify that the requirements of the class functionality are respected. 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. Classes that haven't been tested cannot be included in the integration testing, This in order to simplify error-finding procedures in addition to trying to avoid some sort of big-bang approach. Not completed components, either with respect to the code or the testing, will be integrated and tested incrementally.

2.2 Elements to be Integrated

Elements to be integrated are those individuated in DD document (look at Reference, at point Two). For the sake of legibility they are reported here:

- Queue Handler
- Requests Handler
- Shareable Ride Finder
- Mapping Features Handler
- DBMS and Log-In Handler
- Ban Handler
- Passenger Communicator
- Driver Communicator
- Passenger-views
- Driver-views

The integration testing involves all the code modules developed, moreover, the end-user application has to be tested in the various system environments stated in DD document at the Scope chapter, these are:

- Unix based OS.
- Windows systems.
- Android.
- iOS.
- Windows phone.

2.3 Integration Strategy

The integration strategy will follow a mixed-approach, also known as sandwich approach.

Model integration will start with the completion of DBMS and Log-In Handler, Mapping Features Handler and the Requests Handler. These components can be easily tested in their integration with the the Queue Handler, that will be initilially implemented as a stub component and than integrated in the final realization.

Indeed, since these components require from the Queue handler only some simple return values, this values can be retrieved by very simple stubs class (of about one return method).

This approach also allows the functions incremental integration, in fact the Shareable Ride Finder, which is a component bounded to an "extra" functionality, will be implemented in a second time. Since it isn't an essential part but an additional functionality, we have no need to use stubs to fulfill his functions.

Actually, the Model part of the system can be viewed as a "sub-component", in an ideal hierarchy, with the controller and the view on top. These two parts can be mocked up easily, to test an embryonic version of the model, which is the core part of the entire system; this has to be developed in first place.

Here we are using a bottom-up approach instead of testing the component of the model we have preferred a top-down approach. This definitely justify the general sandwich approach as a proper one.

A little note has to be made on database component testing: this will be done populating the DBMS table with some fake values, and actually it couldn't be unit tested as a normal class. Instead, the class interacting with him will be tested to ensure his correct implementation.

The main feature that this strategy guarantees is that it perfectly matches the incremental approach needed by the developing of the system.

2.4 Software Integration Sequence

The integration will follow the path specified below:

- In a first time, model components will be developed and tested along side their completion.

    1. The first components to be developed and tested will be the DBMS & Log-In Handler, Mapping Features Handler and Requests Handler, as these are the main components of the entire system. Interactions among the related components will be simulated by the realization of simple and quick stubs.

       The development procedure will be split up among three different developing-team. This will optimize the project timeliness since that works will going on in a parallel way.

       The integration between Mapping Features Handler 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. A first integration between Mapping Features Handler and DBMS & Log-In Handler should be done instead.

       The Request Handler will be also provided of a stub component, that will play the role of a simple Shareable Request Finder.

       This procedure will be even applied for the realization of a ban handler stub, which will also be implemented as a driver for the Request Handler.
    2. After that point one is successfully completed, Queue Handler will be developed and tested. When it correctly meets all the Entry Criteria stated above, it will take place of the stub used at point 1, then, the integration will be tested again.

       This approach ensures to clearly separate integration issues coming from the Mapping Features Handler, DBMS & Log-In Handler and Requests Handle, or the newly integrated component.
    3. At least, the Ban Handler will be developed and tested exploiting the already existent stub and driver.
    4. Functionalities of the whole Model "Subsystem" 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.

- In an optic of an incremental approach, Controllers and Views components will be still developed and tested one by one.

    1. Controllers, either for the driver or the passenger side, will be firstly developed an tested. This allows the developers to clearly know what are the inputs requested by the Model to perform his functions. They actually will be the result of the integration, and huge amplification of the drivers used in the model testing.

       At the end of this process, when the Controllers definitely meet the entry criteria and one at time, they will take the drivers place, and will pass thorough the integration test.
    2. View will be developed upon the Controllers integration testing completion. As well as latter, they will be the result of the drivers integration and amplification.

       At the end of this process, when the views definitely meet the entry criteria and one at time, they will take take the drivers place, and will pass thorough the integration test.

- Reached this point, the entire system will going thorough a general integration system test, that will cover:

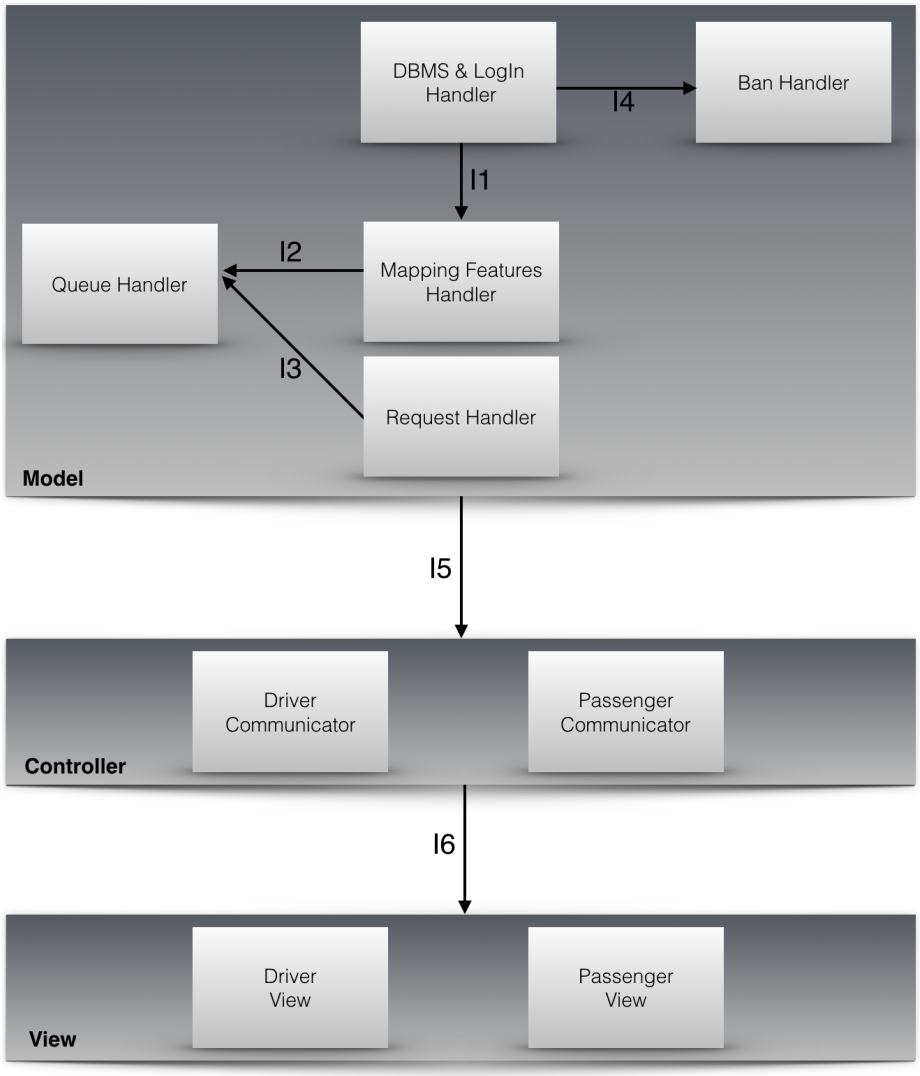
    - Whole System Testing.
    - Performance testing.
    - Ultimate features testing, exploiting the main scenarios in which the system has to work.

- At this time, after passing all integration and acceptance testing, a first finite product version will be released for beta testing purpose, among a bunch of selected test-user. This version will be spoiled of the extra feature provided by the final one. Those are Shareable ride function and MTS API. This will allow a more flexible timetable for the releasing date. Indeed, if the develop process encounters some delays, it will be possible to evaluate a first basic-release spoiled of the "extra" features listed above. From time to time the last functionalities will be added when ready.

- Meanwhile Shareable Ride Finder component will be developed, tested and integrated.

- API functionalities will be implemented and tested at least with some fake implementation of their use.

For having a better overview of the Integration Sequence procedure, a simple scheme follows:



2.5 Subsystem Integration Sequence

As result of the integration sequence, the subsystems taking part in the application will be integrated in this order:

1. Model core functionalities.
2. Controllers.
3. Views.
4. Extra functionalities and API.

3 — Individual Steps and Test Description

3.1 Integration Test 1

Test Case Identifier	I1
Test Items	DBMS & LogIn Handler – Mapping Features Handler
Other Stubs	None
Input Specification	Method calls from DBMS & LogIn Handler
Output Specification	Check that the correct methods are called in Mapping Features Handler
Tests applied	The tests done must verify that the component: • calls the correct method for locating a driver basing on his position when his availability state changes

3.2 Integration Test 2

Test Case Identifier	I2
Test Items	DBMS & LogIn Handler – Queue Handler stub
Other Stubs	None
Input Specification	Method calls from DBMS & LogIn Handler
Output Specification	Check that the correct methods are called in Queue Handler stub
Tests applied	The tests done must verify that the component: • calls the correct method for adding a driver in the queue related to his position when his availability becomes "true" • calls the correct method for deleting a driver from the queue related to his position when his availability becomes "false"

3.3 Integration Test 3

Test Case Identifier	I3
Test Items	Request Handler – Queue Handler stub
Other Stubs	Shareable Request Finder
Input Specification	Method calls from Request Handler
Output Specification	Check that the correct methods are called in Queue Handler stub
Tests applied	The tests done must verify that the component: • calls the correct method for extracting a driver from the queue related to the ingoing request starting position • calls the correct method for adding a driver in the queue related to the ingoing request starting position if the user doesn't confirm or if the timeout expires • calls the correct method for creating a shereabled request if the ingoing one asks for it

3.4 Integration Test 4

Test Case Identifier	I4
Test Items	DBMS & LogIn Handler – Ban Handler Stub
Other Stubs	None
Input Specification	Method calls from Queue Handler
Output Specification	Check that the correct methods are called in Ban Handler
Tests applied	The tests done must verify that the component: • calls the correct method for adding a new banned user • calls the correct method for checking whether a user passed as parameter is banned or not

3.5 Integration Test 5

Test Case Identifier	I5
Test Items	Model component – Controller
Other Stubs	None
Input Specification	Method calls from Model components
Output Specification	Check that the correct methods are called in Controller
Tests applied	The tests done must verify that: • The Request Handler component calls the correct method for sending a request to a given driver • The Request Handler component correctly receives an ingoing request from a passenger • The Mapping Features Handler component calls the correct method for sending the map informations to a given driver • The Mapping Features Handler component calls the correct method for sending the map informations to a given passenger • The DBMS & LogIn Handler component correctly receives LoginData informations from the Passenger Communicator • The DBMS & LogIn Handler component correctly receives LoginData informations from the Driver Communicator • The DBMS & LogIn Handler component correctly answers to passenger user sending the access permission to the related page • The DBMS & LogIn Handler component correctly answers to driver user sending the access permission the related page • The DBMS & LogIn Handler component refuses the access to a banned user

3.6 Integration Test 6

Test Case Identifier	I6
Test Items	Controller subsystem – View subsystem
Other Stubs	None
Input Specification	Method calls from Controller components
Output Specification	Check that the correct methods are called in View
Tests applied	The tests done must verify that: • The Driver Communicator component calls the correct method for showing the request information to a given driver view • The Passenger Communicator component correctly receives an ingoing request from a passenger view • The Driver Communicator component calls the correct method for showing the map and a possible route to a given driver view, basing on the information provided by the model • The Passenger Communicator component calls the correct method for showing the map to a given passenger view, basing on the information provided by the model • The Passenger Communicator component calls the correct method for showing the user related page on his device • The Driver Communicator component calls the correct method for showing the driver related page on his device • The Passenger Communicator component calls the correct method for showing the ban informations to a banned user

4 — Tools and Test Equipment Required

The following tools will be used in order to make tests according to those presented during the lectures. Manual testing will be also used when no other methods will be avaiable.

- Starting by considering Mockito, it will be used either where will be necessary to isolate dependencies (i.e during Requests Handler's building)or to test Mapping Features Handler's interaction with external service (Google APIs); moreover, Mockito is used to make possible the usage of a not implemented yet class. this tool allows even us to stub a method whenever understanding if a object method answer reply in the foresee manners (i.e "When an immediate request is noticed print 'OK' ").
- Arquillian might be used for set up the right interaction among the components especially in order to understand if the database works well.
- JMeter can be used to have consistency and fault tolerance tested. It provides some functionalities in order to simulate a fulfilled form or an information overload, this can be useful close to the end, either an entire "Booking action" need to be tested or whereas the system can work well under strain or a big number of incoming request may cause a failure.

5 — Program Stubs and Test Data Required

Based on the testing strategy and test design, identify any program stubs or special test data required for each integration step.