# Integration Test Plan Document

January 21, 2016

POLITECNICO
MILANO 1863

Daniele Grattarola (Mat. 853101)

Ilyas Inajjar (Mat. 790009)

Andrea Lui (Mat. 850680)

# Contents

# 1 Introduction

## 1.1 Purpose and Scope

This is the Integration Test Plan Document for the MyTaxiService platform project. The main purpose of this document is to define the integration testing that must be carried out during and after the development phase of the platform.

The description of the testing process will include a high level specific of the tests to be executed, the testing strategy and an overview of the tools to be used.

This document is intended for stakeholders, software engineers and developers in charge of the testing implementation.

This document will be limited to integration testing and will ignore all aspects of unit testing, by considering them already conducted.

## 1.2 Definitions and Abbreviations

Throughout this document, the definitions specified in the previous documents will be used without further explanation (refer to section 1 of the RASD and SDD documents).

The following acronyms will also be used in place of the extended form:

- **SDD**: Software Design Document
- **RASD**: Requirement Analysis and Specification Document
- **ITPD**: Integration Plan Testing Document

as well as the other abbreviations defined in this same section of the other documents.

## 1.3 Reference Documents

As stated before, this document will assume that the following documents have been read or are accessible to the reader:

- RASD
- SDD

A typical use of the aforementioned abbreviation would be in the form “element Xx, sec. x.x.x” (e.g. req. 1, sec. 1.3 - if this section contained a numbered

requirement with index 1). One last observation is to be done regarding the use of the singular they, which will be used to refer to single persons throughout the whole document.

# 2 Integration Strategy

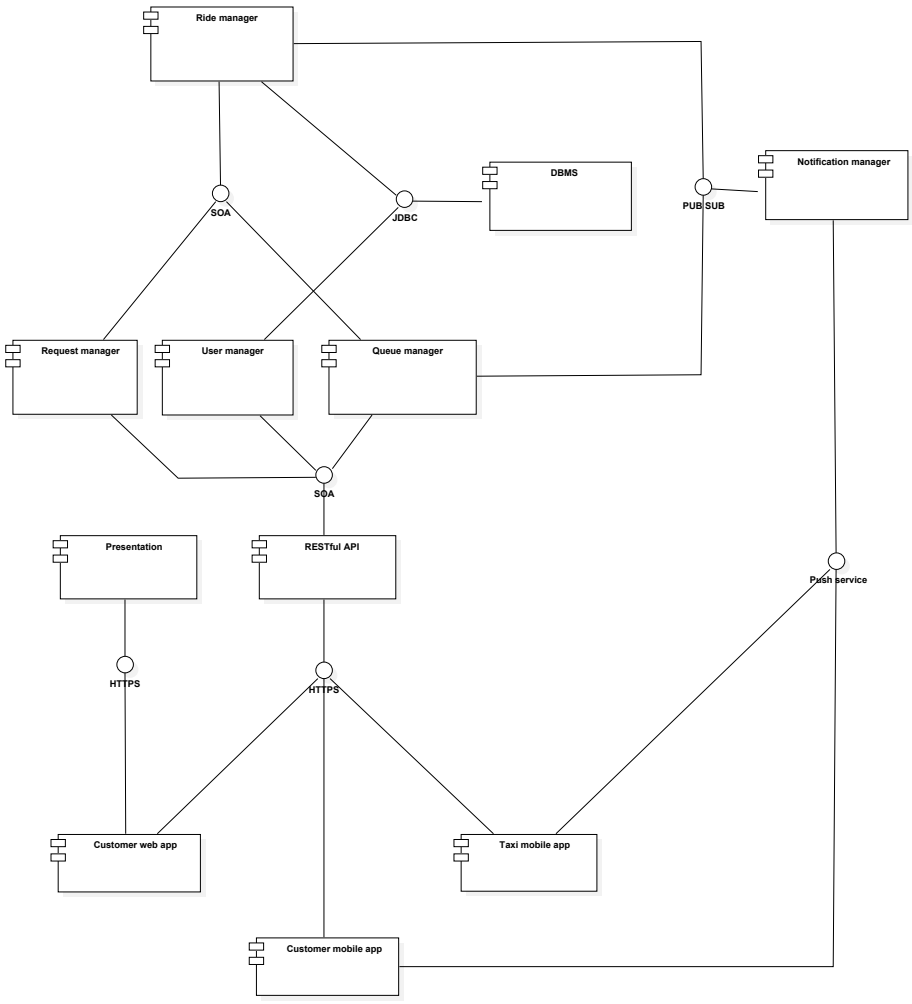
## 2.1 Entry Criteria

For integration testing to be conducted, it is necessary that the main components involved in the integration have been extensively unit tested, with particular regards to those methods that are directly involved in the communication between modules.

Secondary methods of the tested components need not be complete in order for integration testing to be conducted, but must behave as expected from a black-box point of view.

## 2.2 Elements to be Integrated

As described in the SDD, the platform was conceived as a traditional client-server architecture with inner communications happening between the modules of the back-end and direct communication happening back and forth on two separate channels between the back-end and the client-side apps.




As can be deducted by the diagram, integration between the different high level entities of the platform (back-end and client-side) happens through the RESTful API and the Notification Manager, which will be referred to as a "communication-related" entity, from now on.

Since the client-side applications are fairly less complex than the back-end counterpart, they are considered as a single module with no internal interactions. This applies also to the web application, which is described in the component diagram in two parts (app and presentation layer), but will be considered equal to the other two apps, and treated as a single client-side "entity".

With these considerations in mind, the integration testing that must be conducted is between the following pairs:

- Client-side apps - RESTful API
- RESTful API - Request manager
- RESTful API - User manager
- RESTful API - Queue manager
- Request manager - Ride manager
- Queue manger - Ride manager
- Ride manager - DBMS
- User manager - DBMS
- Ride manager - Notification manager
- Queue manager - Notification manager
- Notification manager - Client-side mobile apps

Refer to sec. 2.2 of the SDD for a specific description of the components, the high level functions that the components implement and the interactions between components.

## 2.3 Integration Testing Strategy

Due to the complex structure of the back-end, with multiple inner integration between components, a functional grouping testing strategy is the natural choice to enable the separate development of the various parts of the platform, namely the client-side apps and the back-end.

In a more specific way, integration tests must be conducted at first between the modules of the back-end, followed by tests between the communication-related modules (RESTful API and Notification Manager) and the respective connected components.

To be more specific, the integration must happen in the following order:

1. Back-end integration
2. Client-side integration
3. Communication modules integration

4. High level integration (Back-end with communication, client-side with communication)

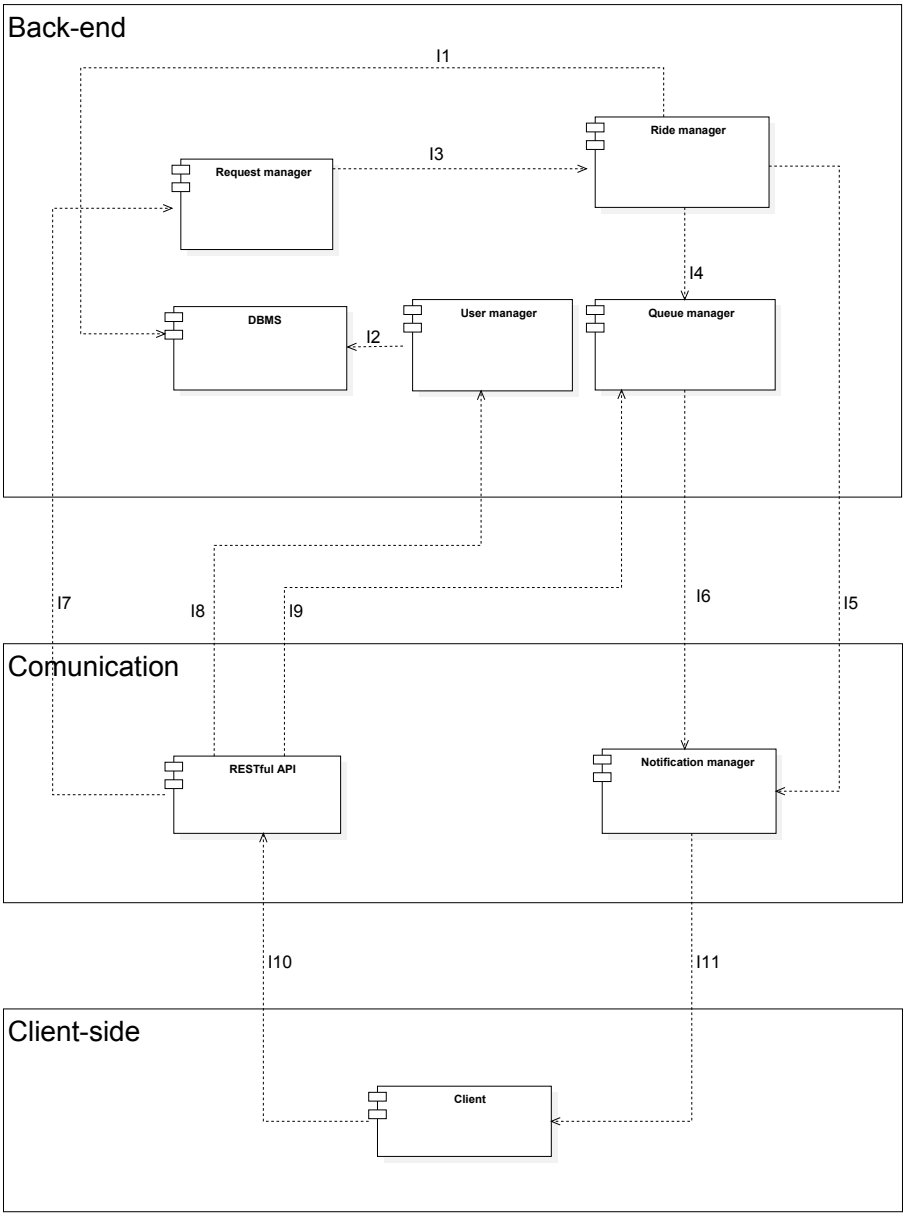
As stated before, the client-side applications and the communication modules can be considered as monolithic components, which do not need integration in order to be functional. Point 2 and 3 of the above list are therefore considered purely for completeness's sake, and do not correspond to any actual tests to be conducted.

## 2.4 Sequence of Integration

As discussed before, the chosen functional grouping strategy for testing is due to the high modularity with which the back-end is designed. The integration testing (which of course immediately precedes the actual integration of the software modules) must therefore start from the testing of the back-end modules and their different interactions in the various use cases. The test can then focus on the integration between the client-side and back-end high-level entities and the communication-related entities.

From a practical point of view, the integration of the software will occur as follows:

1. Test and integrate the back-end modules
2. Test and integrate the back-end and the client-side applications with the dedicated communication modules.




The first tests to be conducted are:

| ID | Test | Section |
|---|---|---|
| I1 | Ride manager - DBMS | 3.1.1 |
| I2 | User manager - DBMS | 3.1.2 |
| I3 | Request manager - Ride manager | 3.1.3 |
| I4 | Ride manager - Queue manager | 3.1.4 |

The testing team can then proceed to test the communication-related interactions. These can be conducted in any desired order, as they are equally important and on the same level of the architecture. The integration tests of this second phase are:

| ID | Test | Section |
|---|---|---|
| I5 | Ride manager - Notification manager | 3.2.1 |
| I6 | Queue manager - Notification manager | 3.2.2 |
| I7 | RESTful API - Request manager | 3.2.3 |
| I8 | RESTful API - User manager | 3.2.4 |
| I9 | RESTful API - Queue manager | 3.2.5 |
| I10 | Client-side apps - RESTful API | 3.2.6 |
| I11 | Notification manager - Client-side mobile apps | 3.2.7 |

# 3 Test description

## 3.1 Stage 1: back-end integration

### 3.1.1 Integration test case I1

| | |
|---|---|
| **Test case identifier** | I1T1 |
| **Test item(s)** | Ride Manager -> DBMS |
| **Input specification** | Create a Ride Manager Object that writes a sample Ride object to DBMS |
| **Output specification** | Check the successful of the operation |
| **Environmental needs** | DBMS running |

| **Test case identifier** | I1T2 |
|---|---|
| **Test item(s)** | Ride Manager -> DBMS |
| **Input specification** | Create a Ride Manager Object that writes a sample Report object to DBMS |
| **Output specification** | Check the successful of the operation |
| **Environmental needs** | DBMS running |

### 3.1.2 Integration test case I2

| **Test case identifier** | I2T1 |
|---|---|
| **Test item(s)** | User Manager -> DBMS |
| **Input specification** | Create a User Manager that writes User Information into the DBMS |
| **Output specification** | Check the successful of the operation |
| **Environmental needs** | DBMS running |

| **Test case identifier** | I2T2 |
|---|---|
| **Test item(s)** | User Manager -> DBMS |
| **Input specification** | Create a User Manager that update User Information into the DBMS |
| **Output specification** | Check the successful of the operation |
| **Environmental needs** | DBMS running, Existing User Information into DBMS |

| **Test case identifier** | I2T3 |
|---|---|
| **Test item(s)** | User Manager -> DBMS |
| **Input specification** | Create a User Manager that requieres User Information from the DBMS |
| **Output specification** | Check the correctness of the data obtained |
| **Environmental needs** | DBMS running, Existing User Information into DBMS |

### 3.1.3 Integration test case I3

| **Test case identifier** | I3T1 |
|---|---|
| **Test item(s)** | Request Manager -> Ride Manager |
| **Input specification** | Create a Request Manager that provides a Request to be managed to the Ride Manager |
| **Output specification** | Check that the correct methods are called and manage correctly the request to build a ride |
| **Environmental needs** | N/A |

| **Test case identifier** | I3T2 |
|---|---|
| **Test item(s)** | Request Manager -> Ride Manager |
| **Input specification** | Create a Request Manager that submit a Report to the Ride Manager |
| **Output specification** | Check that the correct methods are called and the Report is submit (if there is a associated pending Ride) |
| **Environmental needs** | N/A |

#### 3.1.4 Integration test case I4

| **Test case identifier** | I4T1 |
|---|---|
| **Test item(s)** | Ride Manager -> Queue Manager |
| **Input specification** | Create a Ride Manager that request a Taxi Driver from the Queue Manager |
| **Output specification** | Check that the correct methods are called and the returned object (if any) is correct |
| **Environmental needs** | N/A |

## 3.2 Stage 2: client-server integration

#### 3.2.1 Integration test case I5

| **Test case identifier** | I5T1 |
|---|---|
| **Test item(s)** | Ride Manager -> Notification Manager |
| **Input specification** | Create a Ride Manager and a Client Message to be send by the Notification Manager |
| **Output specification** | Check that the Message is sent to the correct Client |
| **Environmental needs** | Previous stages of testing succeeded |

### 3.2.2 Integration test case I6

| **Test case identifier** | I6T1 |
|---|---|
| **Test item(s)** | Queue Manager -> Notification Manager |
| **Input specification** | Create a Ride Manager and a Ride Request to be send to the Taxi Driver by the Notification Manager |
| **Output specification** | Check that the Request is provided to the correct Taxi Driver |
| **Environmental needs** | Previous stages of testing succeeded |

### 3.2.3 Integration test case I7

| **Test case identifier** | I7T1 |
|---|---|
| **Test item(s)** | RESTful API -> Request Manager |
| **Input specification** | Create a RESTful Interface that should redirect the received request to the Request Manager (in this test we create a typical Request Object) |
| **Output specification** | Check the correct redirect method is called |
| **Environmental needs** | Previous stages of testing succeeded |

| **Test case identifier** | I7T2 |
|---|---|
| **Test item(s)** | RESTful API -> Request Manager |
| **Input specification** | Create a RESTful Interface that should redirect the received request to the Request Manager (in this test we create a typical Report Object) |
| **Output specification** | Check the correct redirect method is called |
| **Environmental needs** | Previous stages of testing succeeded |

### 3.2.4 Integration test case I8

| **Test case identifier** | I8T1 |
|---|---|
| **Test item(s)** | RESTful API -> User Manager |
| **Input specification** | Create a RESTful Interface that should redirect the received User request to the User Manager (in this test we create a typical New User Object) |
| **Output specification** | Check the correct redirect method is called |
| **Environmental needs** | Previous stages of testing succeeded |

| **Test case identifier** | I8T2 |
|---|---|
| **Test item(s)** | RESTful API -> User Manager |
| **Input specification** | Create a RESTful Interface that should redirect the received User request to the User Manager (in this test we create a typical Edit User Object) |
| **Output specification** | Check the correct redirect method is called |
| **Environmental needs** | Previous stages of testing succeeded |

| **Test case identifier** | I8T3 |
|---|---|
| **Test item(s)** | RESTful API -> User Manager |
| **Input specification** | Create a RESTful Interface that should redirect the received User request to the User Manager (in this test we create a typical Request Information Object) |
| **Output specification** | Check the correct redirect method is called |
| **Environmental needs** | Previous stages of testing succeeded |

### 3.2.5 Integration test case I9

| **Test case identifier** | I9T1 |
|---|---|
| **Test item(s)** | RESTful API -> Queue Manager |
| **Input specification** | Create a RESTful Interface that should redirect the received a Taxi Availability change (in this test we create a typical Taxi Object or a Reply to a Notification) |
| **Output specification** | Check the correct redirect method is called |
| **Environmental needs** | Previous stages of testing succeeded |

### 3.2.6 Integration test case I10

| **Test case identifier** | I10T1 |
|---|---|
| **Test item(s)** | Client -> RESTful API |
| **Input specification** | Create a Mock Client Application that send throught RESTful API User Informations to be elaborated (Login, Signup, Manage User Information ) |
| **Output specification** | Check that the correct methods are called |
| **Environmental needs** | Previous stages of testing succeeded |

| **Test case identifier** | I10T2 |
|---|---|
| **Test item(s)** | Client -> RESTful API |
| **Input specification** | Create a Mock Client Application that send throught RESTful API a Report against another User |
| **Output specification** | Check that the correct methods are called |
| **Environmental needs** | Previous stages of testing succeeded |

| **Test case identifier** | I10T3 |
|---|---|
| **Test item(s)** | Client -> RESTful API |
| **Input specification** | Create a Mock Customer Application that request, throught RESTful API a Taxi Ride (or a Edit Request) |
| **Output specification** | Check that the correct methods are called |
| **Environmental needs** | Previous stages of testing succeeded |

| **Test case identifier** | I10T4 |
|---|---|
| **Test item(s)** | Client -> RESTful API |
| **Input specification** | Create a Mock Taxi Driver Application that send throught RESTful API a Reply to a Notification |
| **Output specification** | Check that the correct methods are called |
| **Environmental needs** | Previous stages of testing succeeded |

| **Test case identifier** | I10T5 |
|---|---|
| **Test item(s)** | Client -> RESTful API |
| **Input specification** | Create a Mock Taxi Driver Application that send throught RESTful API a Availability Change |
| **Output specification** | Check that the correct methods are called |
| **Environmental needs** | Previous stages of testing succeeded |

#### 3.2.7 Integration test case I11

| | |
|---|---|
| **Test case identifier** | I11T1 |
| **Test item(s)** | Notification Manager -> Client |
| **Input specification** | Create a Notification Manager Object that send a Notification to the Client Application |
| **Output specification** | Check the Notification reached the client |
| **Environmental needs** | Previous stages of testing succeeded |

# 4 Tools and Test Equipment Required

## 4.1 Test equipment

While the final deployment of the platform will need to happen on separate physical machines as described in sec. 2.4. of the DD, for integration testing to happen during development it will be necessary to have available at least a dedicated server able to run the necessary modules at the same time (possibly with separate virtual machines, especially during the second phase of the integration).

## 4.2 Test tools

### 4.2.1 Mockito

Mockito is a test framework usually used to cut out dependencies during unit testing.

It can nevertheless be used to aid the developers during integration, especially during the early stages of development when it may be useful to mock some not-yet-developed components in order to perform some initial integration testing.

### 4.2.2 Arquillian

Arquillian is a test framework that can be used to perform testing inside a remote or embedded container, or deploy an archive to a container so the test can interact as a remote client; Arquillian integrates with other testing frameworks (e.g., JUnit), allowing the use of IDE and Maven test plugins and facilitating the developers' work.

Integration testers should use Arquillian whenever possible, especially during the second phase of the integration.

#### 4.2.3 Manual testing

Manual tests should always be kept as a viable alternative, especially when testing non-standard implementations of algorithms and interfaces. Developers should resort to manual testing whenever the aforementioned tools should result not expressive enough to test specific aspects of the integration.

# 5 Program Stubs and Test Data Required

## 5.1 Program stubs

- In the test suite I10 a code stub that simulates a Client application's behavior is required in order to feed HTTP requests to the RESTful API.
- Code stubs are required each time that the testing involves modules which are not fully developed yet (in their core functions)

## 5.2 Test data

- **I1**: one or more fictitious Ride objects are required, so the necessary data must be provided.
- **I2**: one or more fictitious User objects are required, so the necessary data must be provided.
- **I3**: one or more fictitious Request objects are required, so the necessary data must be provided.
- **I4**: the queues in the Queue manager must be populated with data.
- **I6**: the queues in the Queue manager must be populated with data.
- **I7**: one or more fictitious Request and Report objects are required, so the necessary data must be provided.
- **I8**: one or more fictitious Request objects are required, so the necessary data must be provided.

- **I9**: the queues in the Queue manager must be populated with data.
- **I10**: one or more fictitious Request and Report objects are required, so the necessary data must be provided.

# 6 Additional comments

The production of this document has been a joint effort of all the authors, with a fair distribution of the mansions which caused each member of the group to work on all the parts of the document. The production has been carried out between 13/1/2015 and 21/1/2016 for a total time expense of:

- **Group work**: 4 hours
- **Individual work**:

| | |
|---|---|
| Daniele Grattarola (Mat. 853101) | 6 hours |
| Ilyas Inajjar (Mat. 790009) | 7 hours |
| Andrea Lui (Mat. 850680) | 6 hours |