

# **Test Plan Document**

Michele Madaschi      Lidia Moioli      Luca Martinazzi

January 21, 2016

# Contents

<b>Introduction</b>	<b>2</b>
1.1 Revision History . . . . .	2
1.2 Purpose and Scope . . . . .	2
1.3 List of Definitions and Abbreviations . . . . .	2
1.4 List of Reference Documents . . . . .	2
<b>Integration strategy</b>	<b>3</b>
2.1 Entry criteria . . . . .	3
2.2 Elements to be integrated . . . . .	3
2.3 Integration testing strategy . . . . .	4
2.4 Sequence of component/Function integration . . . . .	4
2.4.1 Software integration sequence . . . . .	4
2.4.2 Subsystem integration sequence . . . . .	5
<b>Individual steps and Test description</b>	<b>8</b>
3.1 Integration test case I-1 . . . . .	8
3.2 Integration test case I-2 . . . . .	8
3.3 Integration test case I-3 . . . . .	8
3.4 Integration test case I-4 . . . . .	8
3.5 Integration test case I-5 . . . . .	9
3.6 Integration test case I-6 . . . . .	9
3.7 Integration test case I-7 . . . . .	9
3.8 Integration test case I-8 . . . . .	9
<b>Tools and testing equipment required</b>	<b>10</b>

# Introduction

## 1.1 Revision History

First version (1.0) of the ITPD document.

## 1.2 Purpose and Scope

This document aims to describe, specify and analyze the integration test strategy for *My Taxi Service*, in terms of the components/classes to integrate and the typology of testing, while also providing a general schedule for the whole process; all is done accordingly to what was established in the previous assignments.

## 1.3 List of Definitions and Abbreviations

- RASD: Requirements Analysis and System Design
- DD: Design Document
- ITPD: Integration Test Plan Document

## 1.4 List of Reference Documents

- The project description.
- Our RASD document.
- Our DD document.

# Integration strategy

## 2.1 Entry criteria

In order to start an integration test, two constraints must be satisfied: the major classes must be covered by, at least, 60 percent of unit tests, while for the others a value of 30 percent is sufficient.

## 2.2 Elements to be integrated

In our document, “element” is used as synonym of “class”; the following list describes the classes that need to undergo an integration test, in order to be sure that our application will behave correctly.

<b>Integration Test: Ridesmanager</b>	
Ridesmanager → Ride, Sharedride	in order to store information about the activated rides
Ridesmanager → Taxiqueue	in order to take information of available taxis in case of taxi request.
Ridesmanager → Controller	in order to exchange information about user's( and also guest's) requests

<b>Integration Test: Controller</b>	
Controller → User	in order to create an ad-hoc Controller and to retrieve information about users
Controller → Servernetworkinterface	in order to communicate with the corresponding client side

<b>Integration Test: Servernetworkinterface</b>	
Servernetworkinterface → Clientmessage	in order to read client's messages
Servernetworkinterface → Servermessage	in order to send messages to the client

---

**Integration Test: Activity**

---

Activity → Action	in order to provides the allowed actions
Activity → Userinterface	in order to provide the set of items this class needs to show

---

**Integration Test: Action**

---

Action → Clientnetworkinterface	in order to send requests to the server
---------------------------------	---

---

**Integration Test: Userinterface**

---

Userinterface → Clientnetworkinterface	in order to show the right Activity according to the server message
--	---

---

**Integration Test: Clientnetworkinterface**

---

Clientnetworkinterface → Clientmessage	in order to send messages to the server
Clientnetworkinterface → Servermessage	in order to read server's messages

---

## 2.3 Integration testing strategy

In this section we will explain how we planned the integration test in order to build, as soon as possible, a running application with few working features; this will allow us to promptly show our progress to the customer and also, in case of a delay in the development, to launch a working application, although with missing requirements. In order to reach our goal we decided to apply a *bottom-up* method during the integration test phase, and a *top-down* one for the unit tests.

The first working version of our application will include major classes; in this milestone, there are no users, but only a guest that has the possibility to access all the already implemented features.

The second version will add multiple other users with the related constraints, as explained in the previous documents (see RASD and Design Document). From the second version onward, the application could be released, even if only few features are already implemented.

The next versions will include other features that allow us to meet all the missing requirements.

## 2.4 Sequence of component/Function integration

### 2.4.1 Software integration sequence

The software integration sequence is shown in Figure 2.1

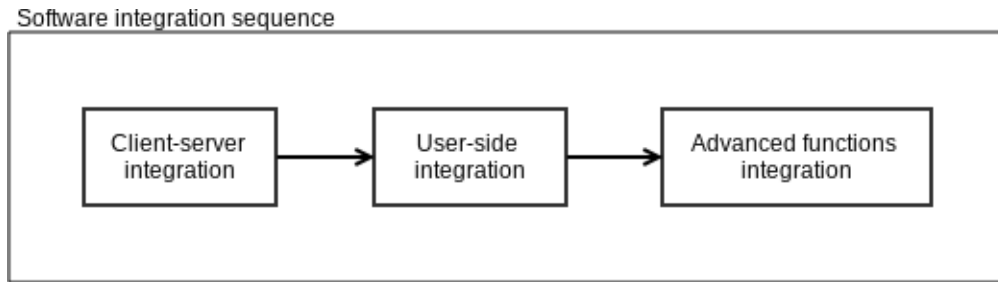


Figure 2.1: Software integration

### 2.4.2 Subsystem integration sequence

The classes presented in the following figures are ordered according to the sequence in which they will be tested, which is: 2.2  $\rightarrow$  2.3  $\rightarrow$  2.4  $\rightarrow$  2.5  $\rightarrow$  2.6

Note: the arrows here represent the ordering of the integration, which may happen to partially match the logical structure of the project; however, those arrows do not aim to describe the inter-class relationships

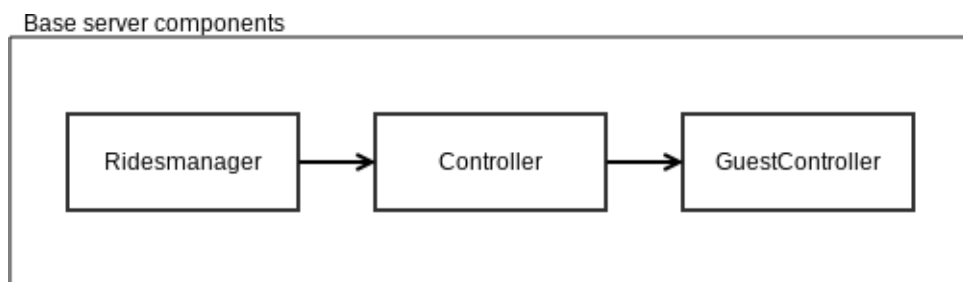


Figure 2.2: Base server components

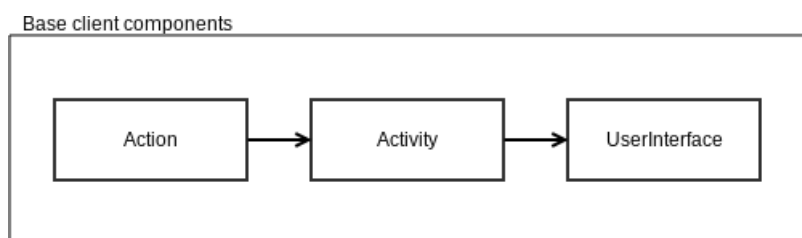


Figure 2.3: Base client components

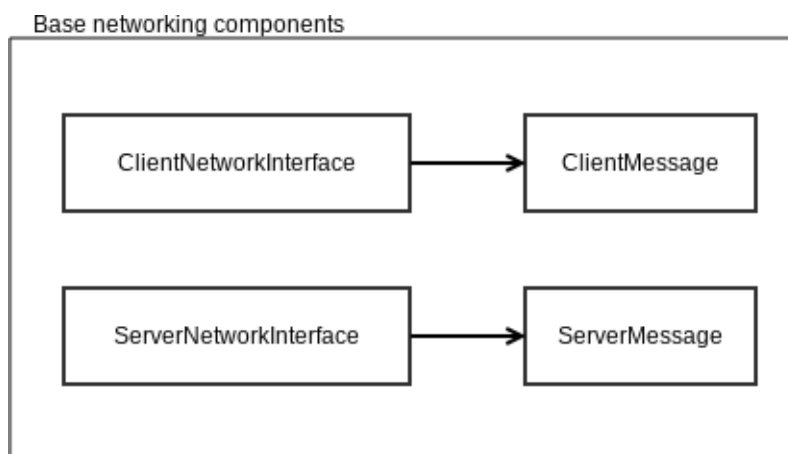


Figure 2.4: Base networking components

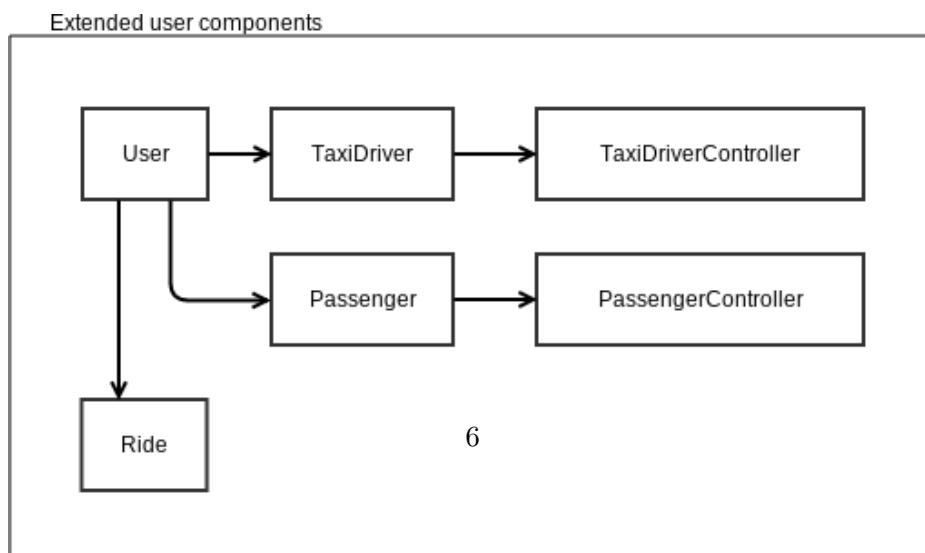


Figure 2.5: Extended client components

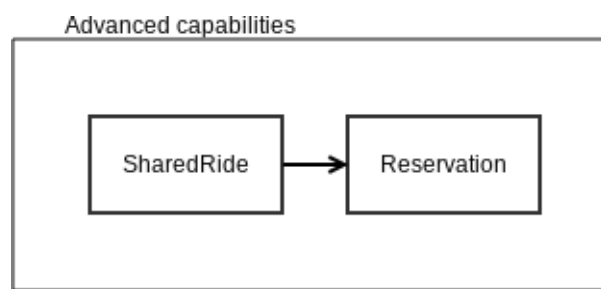


Figure 2.6: Advanced functions



# Individual steps and Test description

## 3.1 Integration test case I-1

<b>Test Case Identifier</b>	I-1-T1
<b>Test Item(s)</b>	Ridesmanager → Controller
<b>Input specification</b>	Create the typical Ridesmanager input
<b>Output specification</b>	Check if the correct methods are called in the Controller
<b>Environmental needs</b>	Ridesmanager driver

## 3.2 Integration test case I-2

<b>Test Case Identifier</b>	I-2-T1
<b>Test Item(s)</b>	Controller → GuestController
<b>Input specification</b>	Create the typical Controller input
<b>Output specification</b>	Check if the correct methods are called in the GuestController
<b>Environmental needs</b>	I-1 succeeded

## 3.3 Integration test case I-3

<b>Test Case Identifier</b>	I-3-T1
<b>Test Item(s)</b>	Activity → Action, TaxirequestAction, TaxiresponseAction
<b>Input specification</b>	Create a generic Activity and the Home
<b>Output specification</b>	Check that the Activities does create the correct Actions
<b>Environmental needs</b>	An Activity driver

## 3.4 Integration test case I-4

<b>Test Case Identifier</b>	I-4-T1
<b>Test Item(s)</b>	ClientNetworkInterface → ClientMessage
<b>Input specification</b>	Invoke various types of network methods
<b>Output specification</b>	Check that the correct ClientMessage(s) are generated
<b>Environmental needs</b>	ClientNetworkInterface driver

### 3.5 Integration test case I-5

<b>Test Case Identifier</b>	I-5-T1
<b>Test Item(s)</b>	ServerNetworkInterface → ServerMessage
<b>Input specification</b>	Invoke various types of network methods
<b>Output specification</b>	Check that the correct ServerMessage(s) are generated
<b>Environmental needs</b>	ServerNetworkInterface driver

### 3.6 Integration test case I-6

<b>Test Case Identifier</b>	I-6-T1
<b>Test Item(s)</b>	User → Ride
<b>Input specification</b>	Add a new Ride to an User
<b>Output specification</b>	Check that the Ride is correctly added
<b>Environmental needs</b>	User driver

### 3.7 Integration test case I-7

<b>Test Case Identifier</b>	I-7-T1
<b>Test Item(s)</b>	Taxidriver → TaxidriverController
<b>Input specification</b>	Add a new Taxidriver to a TaxidriverController
<b>Output specification</b>	Check that the Taxidriver is correctly added
<b>Environmental needs</b>	TaxidriverController driver

### 3.8 Integration test case I-8

<b>Test Case Identifier</b>	I-8-T1
<b>Test Item(s)</b>	Passenger → PassengersController
<b>Input specification</b>	Add a new Passenger to a PassengersController
<b>Output specification</b>	Check that the Passenger is correctly added
<b>Environmental needs</b>	PassengersController driver

# Tools and testing equipment required

Maven : As a platform to manage builds and dependencies

JUnit : For the implementation of unit tests it's an obvious choice, given its integration with the major IDEs and the overall simplicity and familiarity for the developers

Arquillian : For the integration testing phase, we have chosen this tool, since it easily integrates with Maven and JUnit, and presents speed and simplicity as its defining traits