Test Plan Document

Michele Madaschi Lidia Moioli Luca Martinazzi

January 21, 2016

Contents

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.
- Ou 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.
Major classes are: Userinterface, Activity, Action, Clientnetworkinterface, Servernetworkinterface, Controller, Ridesmanager, User, Ride, Sharedride, Taxiqueue.

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.

Integration Test: Ridesmanager	
Ridesmanager → Ride, Sharedride	in order to store information about the actived 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

Integration Test: Controller	
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

Integration Test: Servernetworkinterface	
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

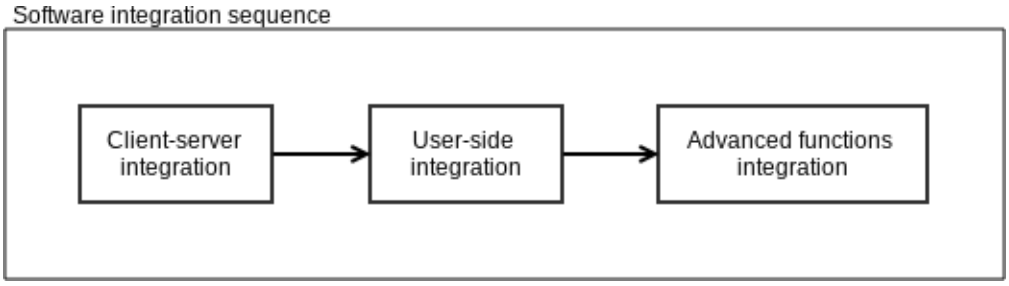

Figure 2.1: Software integration

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 → 2.3 → 2.4 → 2.5 → 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

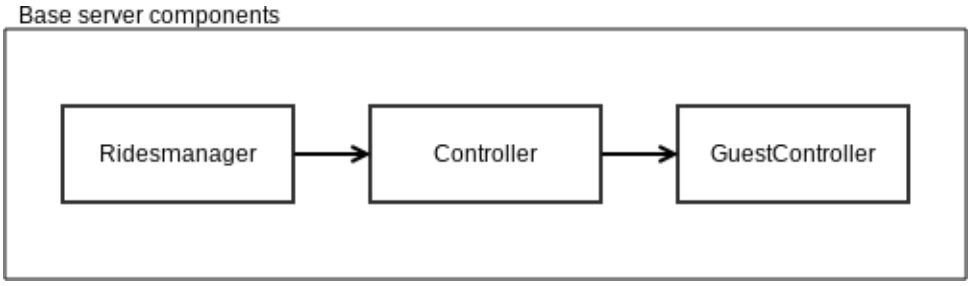

Figure 2.2: Base server components

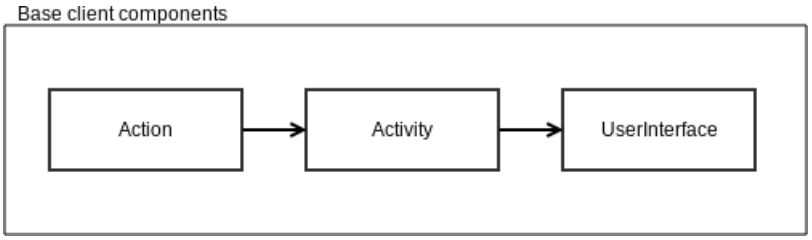

Figure 2.3: Base client components

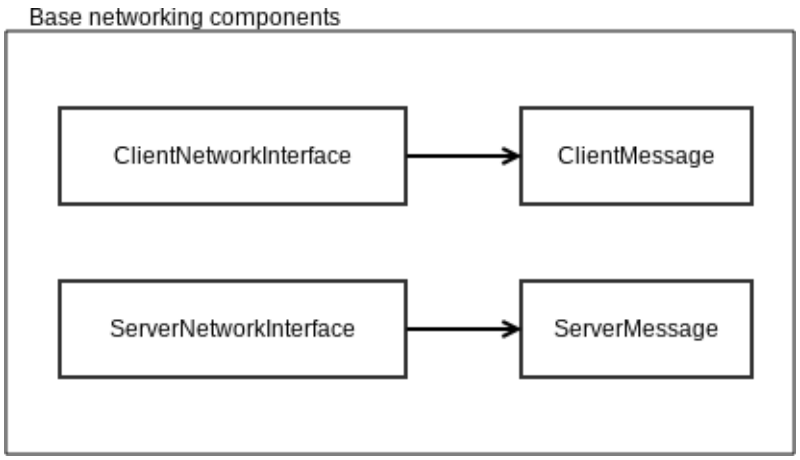

Figure 2.4: Base networking components

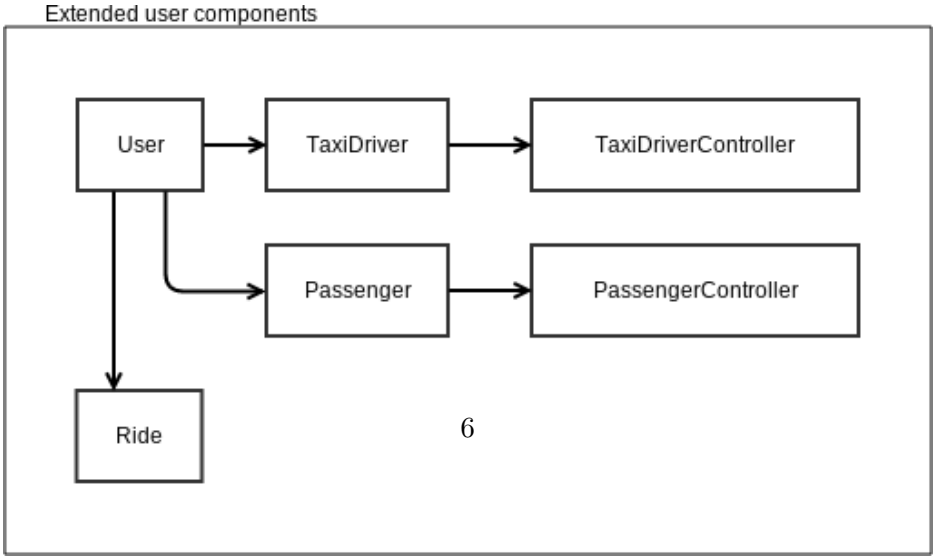

Figure 2.5: Extended client components

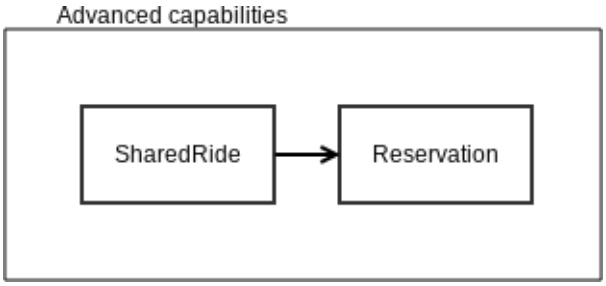

Figure 2.6: Advanced functions

Individual steps and Test description

3.1 Integration test case I-1

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

3.2 Integration test case I-2

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

3.3 Integration test case I-3

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

3.4 Integration test case I-4

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

3.5 Integration test case I-5

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

3.6 Integration test case I-6

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

3.7 Integration test case I-7

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

3.8 Integration test case I-8

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