Integration Test Plan Document

Mite Ristovski Dushica Stojkoska

January 21, 2016

Software Engineering 2
Politecnico di Milano

version 1.0

Contents

1 Introduction

1.1 Revision History

Version of this document: 1.0
Last updated: 21 January 2016

1.2 Purpose and Scope

The purpose of this document is to describe how integration testing for the software product MyTaxiService should be performed. Integration testing phase means that we are interested in verifying that every component of the system works as expected with correlation with others components. Moreover, presented are: the elements that should be tested, the strategy for the testing, tools, programs, stubs needed for development of this software product.

The main goal of MyTaxiService is to ease taxi requests and reservations for the registered customers. In order to function properly, MyTaxiService must be able to support remote communication over the Internet, as well as successfully accessing external services like GPS service and performing database transactions. Therefore the scope of this document refers on testing the system in the whole, considering all components together.

1.3 List of Definitions and Abbreviations

- **User**: All registered users that are using the product
- **Request**: Asking for available taxi
- **Reservation**: Booking taxi ride for a specified time and date
- **GPS**: Global positioning system
- **RASD**: Requirement Analysis and Specification Document
- **DD**: Design Document

1.4 List of Reference Documents

- RASD MyTaxiService
- DD MyTaxiService

Tools used

- Sharelatex(LaTeX): For writing this file.

2 Integration Strategy

2.1 Entry Criteria

There are several entry criteria that must be respected before integration testing phase starts:

- RASD and DD are complete and revisited
- Software product satisfies all requirements and design specified in RASD and DD
- Coding part of the modules is completed
- All bugs are fixed
- Components are unit tested
- Required test environment is ready for integration testing

2.2 Elements to be Integrated

Elements to be integrated are:

- User Manager
- Authentication Manager
- Ride Manager
- Event Manager
- Queue Manager
- Database Manager

2.3 Integration Testing Strategy

For testing of the modules we are choosing the bottom-up approach. This means that integration testing starts at the bottom level and later on continues on the higher level. The choice for this approach comes from the entry criteria, which states that unit tests for each component are already done, so we can proceed from the bottom level. The integration tests described in this document are at the component level Section 2.2.

2.4 Subsystem Integration Sequence

Integration Tests of the scheduling subsystem:

ID	Integration Test	Subsection
I1	Ride Manager → Event Manager	3.1
I2	Ride Manager → Queue Manager	3.2
I3	Event Manager → Queue Manager	3.3
I4	Event Manager → Database Manager	3.4

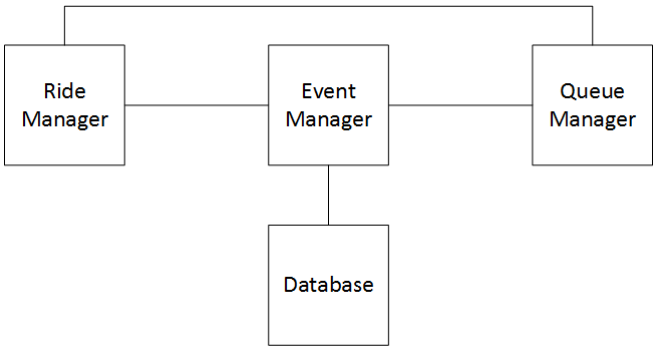

Figure 1: Scheduling subsystem

Integration Tests of the operational subsystem:

ID	Integration Test	Subsection
I5	User Manager → Database Manager	3.5
I6	Authentication Manager → Database Manager	3.6

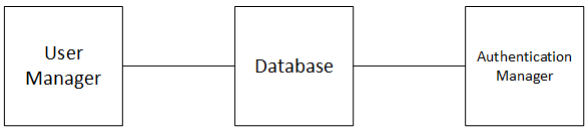

Figure 2: Operational subsystem

2.5 Sequence of Component/Function Integration

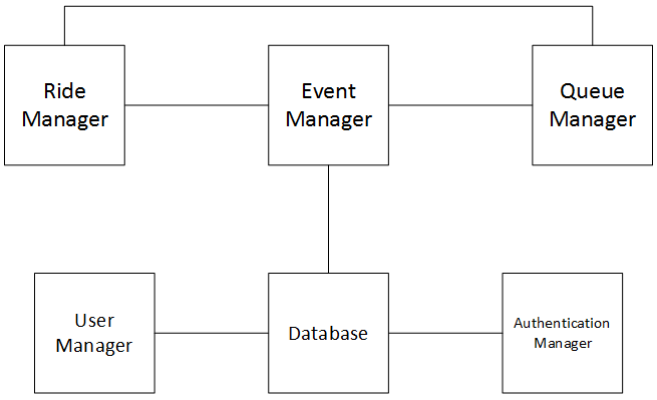

Figure 3: System integration

3 Individual Steps and Test Description

3.1 Integration test case I1

Test Case Identifier	I1T1
Test Item(s)	Ride Manager → Event Manager
Input Specification	Data about the ride and taxi allocation
Output Specification	Dispatch taxi
Environmental Needs	Queue Manager Driver and Authentication succ

3.2 Integration test case I2

Test Case Identifier	I2T1
Test Item(s)	Ride Manager → Queue Manager
Input Specification	Location data
Output Specification	Closest available taxi
Environmental Needs	Mobile app and Authentication Manager driver

3.3 Integration test case I3

Test Case Identifier	I3T1
Test Item(s)	Event Manager → Queue Manager
Input Specification	Allocated taxi
Output Specification	Confirmation of the allocation
Environmental Needs	I1T1 and I2T1 succeeded

3.4 Integration test case I4

Test Case Identifier	I4T1
Test Item(s)	Event Manager → Database Manager
Input Specification	Reservation to store
Output Specification	Reservation key
Environmental Needs	I1T1, I2T1 and I3T1 succeeded

3.5 Integration test case I5

Test Case Identifier	I5T1
Test Item(s)	User Manager → Database Manager
Input Specification	Registration data
Output Specification	Store data securely
Environmental Needs	N/A

3.6 Integration test case I6

Test Case Identifier	I6T1
Test Item(s)	Authentication Manager → Database Manager
Input Specification	User name and password
Output Specification	Confirm authenticity of the given credentials
Environmental Needs	N/A

4 Tools and Test Equipment Required

Here are presented some suggestions for tools that can be used to perform different kinds of testing:

- **JUnit**[1] is the most used framework for writing unit tests in Java. JUnit can be used for testing of the single components, but also can be used together with Mockito and Arquillian to perform integration testing.
- **Arquillian**[2] is a test framework that can be used for testing inside a remote or embedded container, or deploy an archive to a container so the test can interact as a remote client.
- **Mockito**[3] is an open-source test framework that can be used to generate mock objects, stubs and drivers.

5 Program Stubs and Test Data Required

Because we bottom-up approach for testing, there are several drivers that can be identified for testing our components. The following are needed:

1. Authentication Manager driver
2. Queue Manager driver

6 Appendix

6.1 Hours of work

Mite Ristovski: ≈ 13h.

Dushica Stojkoska: ≈ 12h.

[1] http://junit.org/
[2] http://arquillian.org/
[3] http://site.mockito.org/