POLITECNICO DI MILANO

A.A. 2015-2016

SOFTWARE ENGINEERING 2: “myTaxiService”

Integration Test Plan Document (ITPD)

Andrijana Mirchevska (838622)

Marija Mavcheva (838647)

21 January 2016

Table of Contents

1 Introduction

1.1 Revision History

Document Title: Integration Test Plan Document (ITPD)
Version: 1.0
Date: 21.01.2016
Summary: Final version

1.2 Purpose and Scope

The aim of this document is to describe the plan for testing, integration testing and verifying that the system development meets the specifications of the Requirement Document and Design document. Starting point of this document is the architectural description of the software system which is discussed in the Design Document.

The project is about developing an application that will enable fast and optimized taxi services in the city. The application will allow users to register and then sign in into the app for using its services. Also taxi drivers can register and sign into the myTaxiService application with the purpose to manage their availability and duties.

1.3 List of Definitions and Abbreviations

MakeReservation	Passenger request a vehicle at least 2 hours before the ride
Request	Passenger request an immediate ride
RideConfirmation	Receive a confirmation about the confirmed ride with all the information about the particular ride
ReservationConfirmation	Notification that the reservation is successfully completed
ReportDriver	Passenger reports driver in case of any irregularities
ReportUser	Taxi driver reports user (particular passenger) in case of any irregularities during the ride
Guest	Not registered person that visits the app
User	A person that is already registered and signed in as user
TaxiDriver	A person that is already registered and signed in as driver
API	Application Programming Interface
GPS	Global Positioning System

1.4 List of Reference Documents

- Assignment 4 - Integration Test Plan Document
- RASD - myTaxiSevice RASD – final
- DD - DD Final Version
- Integration Test Plan Example

2 Integration Strategy

2.1 Entry Criteria

The entry criteria for the integration testing are the functions of the components that need to be Unity tested. Those are:

Guest Manager:

- register()
- logIn()

User Manager:

- requestTaxi()
- makeReservation()
- cancelReservation()
- manageProfile()
- reportDriver()

TaxiDriver Manager:

- confirmDeclineCall()
- manageRequests()
- reportUser()
- manageProfile()

Administrator Manager:

- viewReports()
- banUser()

Controller Manager:

1. *Request Manager*
    - createRequest()
    - provideTaxi()
    - sendConfirmation()
    - findZone()
    - findDriver()

2. *Reservation Manager*
    - createReservation()
    - provideTaxi()
    - sendConfirmation()
    - findZone()
    - findDriver()

3. *Zone Manager*
    - findZone()
    - findAvailableDriver()
    - enqueDriver()
    - dequeueDriver()
    - takeDriverFromQueue()

2.2 Elements to be integrated

The figures below show from which components is constructed MyTaxiService system. We derived these pictures from the Chapter 3: Component View from Design Document. The arrows on the pictures show the integration testing.

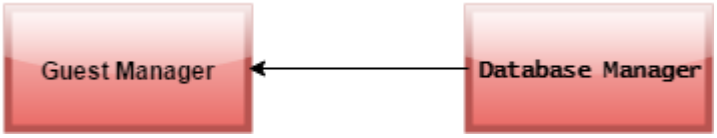

Figure 2.1 Guest Component

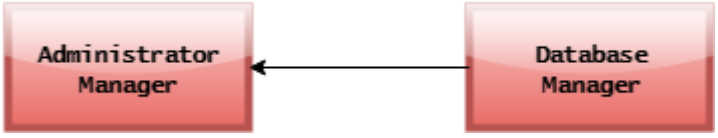

Figure 2.2 Administrator Component

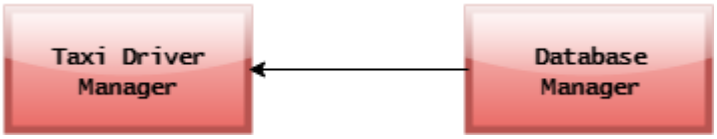

Figure 2.3 Taxi Driver Component

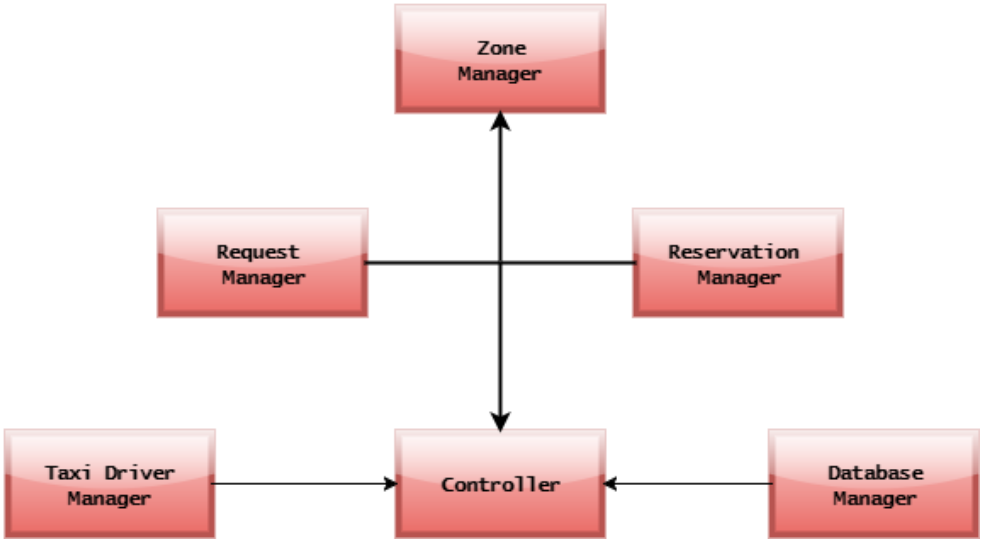

Figure 2.4 Controller Component

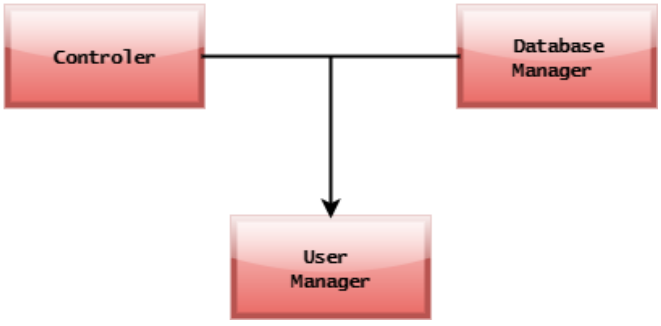

Figure 2.5 User Component

2.3 Integration Testing Strategy

We decided to use bottom up integration testing strategy because in that way we will always start at the bottom of the hierarchy. This means that the critical modules will be generally built and tested first. Therefore any errors or mistakes in these forms of modules are find out earlier in the process. Each component at lower hierarchy will be tested individually; then the higher level components that rely upon the lower ones, will be tested.

Figure 2.6 Integration Test of MyTaxiService System

2.4 Sequence of Component/Function Integration

2.4.1 Software Integration Sequence

2.4.1.1 Integration Test of Guest Component

The integration test of Guest Component corresponds to the figure 2.1 of this document.
IntegrationTest1: Database Manager ⟹ Guest Manager

2.4.1.2 Integration Test of Administrator Component

The integration test of Administrator Component corresponds to the figure 2.2 of this document.
IntegrationTest2: Database Manager ⟹ Administrator Manager

2.4.1.3 Integration Test of Taxi Driver Component

The integration test of Taxi Driver Component corresponds to the figure 2.3 of this document.
IntegrationTest3: Database Manager ⟹ Taxi Driver Manager

2.4.1.4 Integration Test of Controller Component

The integration test of Controller Component corresponds to the figure 2.4 of this document.
IntegrationTest4: Zone Manager ⟹ Request Manger, Reservation Manager
IntegrationTest5: Request Manger, Reservation Manager, Taxi Driver Manager, Database Manager ⟹ Controller

2.4.1.5 Integration Test of User Component

The integration test of User Component corresponds to the figure 2.5 of this document.
IntegrationTest6: Database Manager ⟹ User Manager

2.4.2 Subsystem Integration Sequence

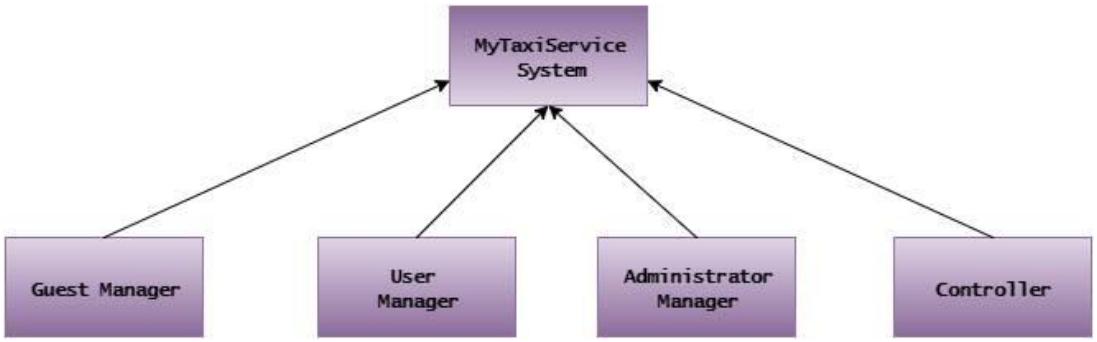

Figure 2.7 MyTaxiService Subsystem Integration

2.4.2.1 Integration Test of MyTaxiService system

The integration test of MyTaxiService system corresponds to the figure 2.6 of this document.
IntegrationTest7: Guest Manger, User Manager, Administrator Manager, Controller ⟹ myTaxiService system

3 Individual Steps and Test Description

3.1 Integration Test Cases

3.1.1 Integration Test Case I1

Test Case Identifier: I1T1
Test Item(s): Database Manager ➡ Guest Manager
Input Specification: Create typical Database input as a database connection
Output Specification: Check if the correct functions are called in the Guest Manager
Environmental Needs: None

3.1.2 Integration Test Case I2

Test Case Identifier: I2T1
Test Item(s): Database Manager ➡ Administrator Manager
Input Specification: Create typical Database input as a Database connection
Output Specification: Check if the correct functions are called in the Administrator Manager
Environmental Needs: None

3.1.3 Integration Test Case I3

Test Case Identifier: I3T1
Test Item(s): Database Manager➡ Taxi Driver Manager
Input Specification: Create typical Database input as a Database connection
Output Specification: Check if the correct functions are called in the Taxi Driver Manager
Environmental Needs: None

3.1.4 Integration Test Case I4

Test Case Identifier: I4T1
Test Item(s): Zone Manager ➡ Request Manger
Input Specification: Create typical Zone Manager input
Output Specification: Check if the correct functions are called in the RequestManger
Environmental Needs: Driver must receive a request first

Test Case Identifier: I4T2
Test Item(s): Zone Manager ⟹ Reservation Manager
Input Specification: Create typical Zone Manager input
Output Specification: Check if the correct functions are called in the Reservation Manger
Environmental Needs: Driver must receive a reservation first

3.1.5 Integration Test Case I5

Test Case Identifier: I5T1
Test Item(s): Taxi Driver Manager ⟹ Controller
Input Specification: Create typical Taxi Driver Manager input
Output Specification: Check if the correct functions are called in the Controller
Environmental Needs: Require I3 to be successful

Test Case Identifier: I5T2
Test Item(s): Request Manger ⟹ Controller
Input Specification: Create typical Request Manger input
Output Specification: Check if the correct functions are called in the Controller
Environmental Needs: Require I4.1 to be successful

Test Case Identifier: I5T3
Test Item(s): Reservation Manger ⟹ Controller
Input Specification: Create typical Reservation Manger input
Output Specification: Check if the correct functions are called in the Controller
Environmental Needs: Require I4.2 to be successful

Test Case Identifier: I5T4
Test Item(s): Database Manger ⟹ Controller
Input Specification: Create typical Database Manger input
Output Specification: Check if the correct functions are called in the Controller
Environmental Needs: None

3.1.6 Integration Test Case I6

Test Case Identifier: I6T1
Test Item(s): Database Manager ⟹ User Manager
Input Specification: Create typical Database Manger input
Output Specification: Check if the correct functions are called in the User Manager
Environmental Needs: None

3.2 Integration Test Procedures

3.2.1 Integration Test Procedure TP1

Test Procedure Identifier: TP1
Purpose: This test procedure verifies wether the **Guest Component**:

- can handle guest input
- can return correct information to the user

Procedure Steps: Execute I1

3.2.2 Integration Test Procedure TP2

Test Procedure Identifier: TP2
Purpose: This test procedure verifies wether the **Administrator Component**:

- can handle administrator input
- can return correct information to the administrator

Procedure Steps: Execute I2

3.2.3 Integration Test Procedure TP3

Test Procedure Identifier: TP3
Purpose: This test procedure verifies wether the **Taxi Driver Component**:

- can handle taxi driver input
- can return correct information to the taxi driver

Procedure Steps: Execute I3

3.2.4 Integration Test Procedure TP4

Test Procedure Identifier: TP4
Purpose: This test procedure verifies wether the **Controller Component**:

- can handle taxi driver input
- can handle user input
- can return correct information to the taxi driver
- can return correct information to the user

Procedure Steps: Execute I6 after I3-I5

3.2.5 Integration Test Procedure TP5

Test Procedure Identifier: TP5
Purpose: This test procedure verifies wether the **User Component**:

- can handle user input
- can output requested information to the controller
- can return correct information to the user

Procedure Steps: Execute I7 after I6

4 Tools and Test Equipment Required

In order to have a more effective test, we decided to combine together the following tests:

- Manual Test
- Automatic Test

4.1 Manual

The team should deal with user story both in client and server side, following the requirements specified in the RASD. The integration testing will be done manually by the same team member who made it, and after this initial testing, again it will be tested by other member of the team to check if all the requirements are completed.

4.2 Automatic Test

In this part the team should take care of writing code to automatically test each user story. There are several possible scenarios, which is why automatic testing will save us time in testing, because it can be run at any time and each member can check reports.

For automatic testing we use a few parts of “Cucumber” and “Watir-Webdriver”. “Ruby” is the programming language use for it.

4.3 Program Stubs and Test Data Required

Specifications needed to perform the integration steps described in Chapter 3 are:

- ***Test database***: First, sample data should be inserted into the Database Component, in order to execute the test cases.
- ***External Google API stub:*** It is needed to replace the external GoogleMaps and GooglePlaces API system. This stub should provide sample data needed to the TaxiDriverManager component in order to correctly perform navigation, UserManager component to be able to pick his destination address and to be able to see available taxi vehicles near him on the map.
- ***External Mailing Service API stub:*** it is needed to replace the external mailing system. This stub should provide sample data needed to GuestManager when performing signUp procedure.

5 References

- Slides of the Software Engineering 2 course (Beep platform)
- Top-Down and Bottom-Up Approach for Model-Based Testing of Product Lines, Stephan Weißleder & Hartmut Lackner