Politecnico di Milano

A.A. 2015-2016

Software Engineering 2 Project – “myTaxiService”

Prof. Raffaela Mirandola

Integration **T**est **P**lan **D**ocument

Version 1.0

Christian Zichichi (mat. 840565), Luigi Marrocco (mat. 854884)

January 21, 2016

Table of Contents

1. INTRODUCTION

1.1 Revision History

Version	Date	Authors	Summary
1.0	21-01-2016	Christian Zichichi Luigi Marrocco	First Release (Delivery)

1.2 Purpose and Scope

This document describes the plans for testing the integration of the components developed for myTaxiService's system. The purpose of this document is to test the interfaces between the components as described in the relative Design Document. The system is composed of a web application and two mobile applications (one for Taxis' cars, one for Passengers' smartphones) that make the access to the taxi service easy for passengers and, at the same time, guarantee a fair management of taxi queues.

1.3 List of Definitions and Abbreviations

- **Driver:** a routine that simulates a test call from parent component to child component (usually used in bottom-up integration strategy)
- **Stub:** It is a dummy module that provides the response that would be provided by the real sub-element (usually used in top-down integration strategy)
- **JUnit:** a framework to write repeatable unit tests
- **RASD:** Requirements Analysis and Specification Document
- **DD:** Design Document
- **ITPD:** Integration Test Plan Document
- **GUI:** Graphical User Interface
- **REST:** Representational State Transfer (related information on chapter 2.7 of the DD)
- **API:** Application Programming Interface

1.4 List of Reference Documents

This document directly refers to the following documents:

- myTaxiService – **R**equirement **A**nalysis and **S**pecification **D**ocument
- myTaxiService – **D**esign **D**ocument

2. INTEGRATION STRATEGY

2.1 Entry Criteria

All the classes and functions must undergo and pass severe JUnit-based unit tests (done by a reasonable number of testers), which should discover issues and fix bugs. Moreover, all the software has to be accurately inspected in order to avoid mistakes hardly detectable during the integration phase that are not dependent on it. Eventually, the documentation of all classes and functions has to be complete and updated in order to be used as reference for integration testing development and reflect the current state of the project.

2.2 Elements to be Integrated

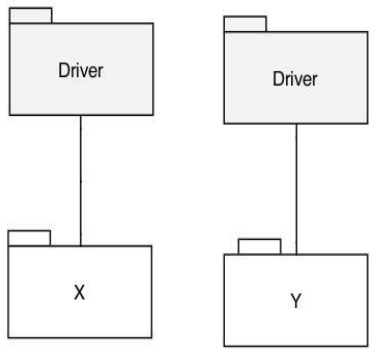
The detailed description of each component's function and the interaction between them is contained in the DD. The general architecture of MyTaxiService can be divided in four main subsystem, each one formed by some components, as follows:

- Application subsystem
  - ZoneManager
  - QueueManager
  - RideManager
  - TaxiManager
  - AccountManager
  - PassengerManager

    - PollingManager
    - Cronjob
- Routing subsystem
    - Dispatcher
    - Security
- Client subsystem
    - PassengerWebGUI
    - PassengerMobileGUI
    - TaxiGUI
- Storage subsystem
    - DataLayer

2.3 Integration Testing Strategy

The elements to be tested consist of the integration of the modules developed in the DD for the myTaxiService project. In order to test them, we chose to adopt the **bottom-up** integration strategy, which consists in integrating a low hierarchy component with its parent. For each module, a routine that simulates a test call from a parent component to a child component is needed (Driver), as shown in the figure below. The integration tests of lower level code modules are described in the corresponding components' unit tests. Unit tests are described in the Unit Test Plan Document, which we did not have to do (we assume it has been already done).

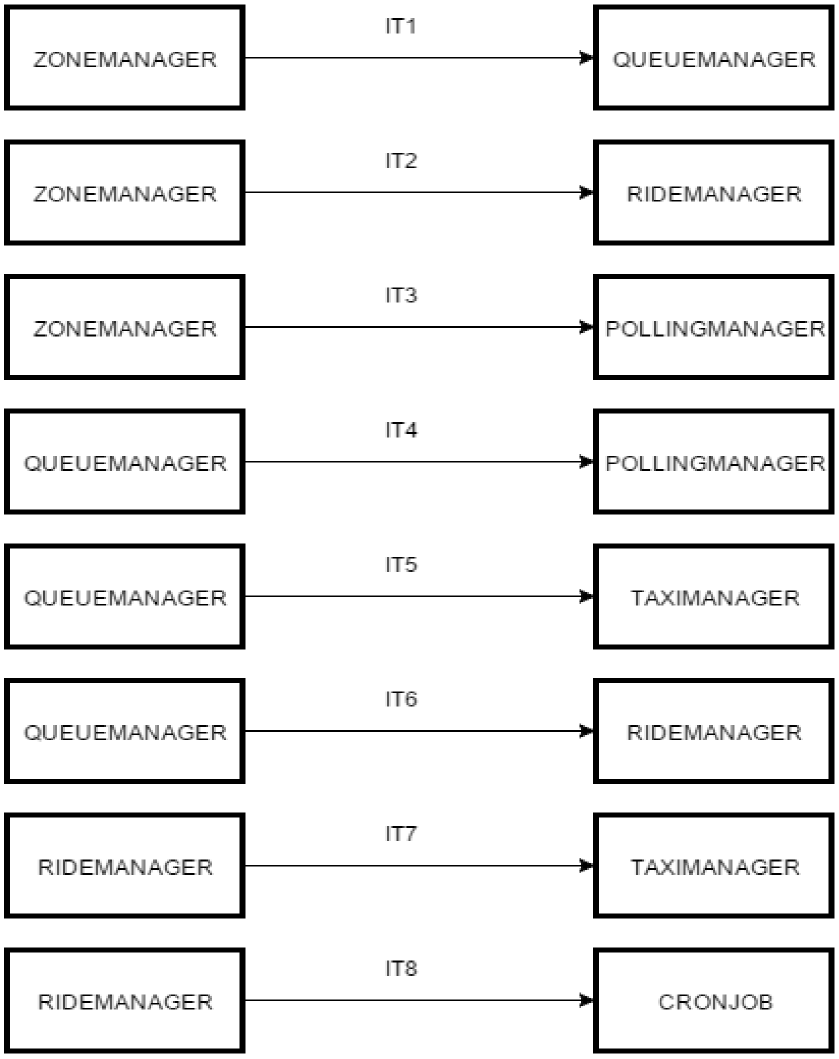


2.4 Sequence of Components/Functions Integrations

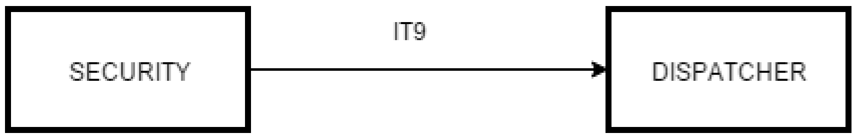
The figure below shows the components that form myTaxiService (derived from the Component Diagram in the Design Document, chapter 2.3). The arrows represent the order of integration, i.e., the integration testing.

2.4.1 Software Integrations Sequence

APPLICATION SUBSYSTEM



ROUTING SUBSYSTEM

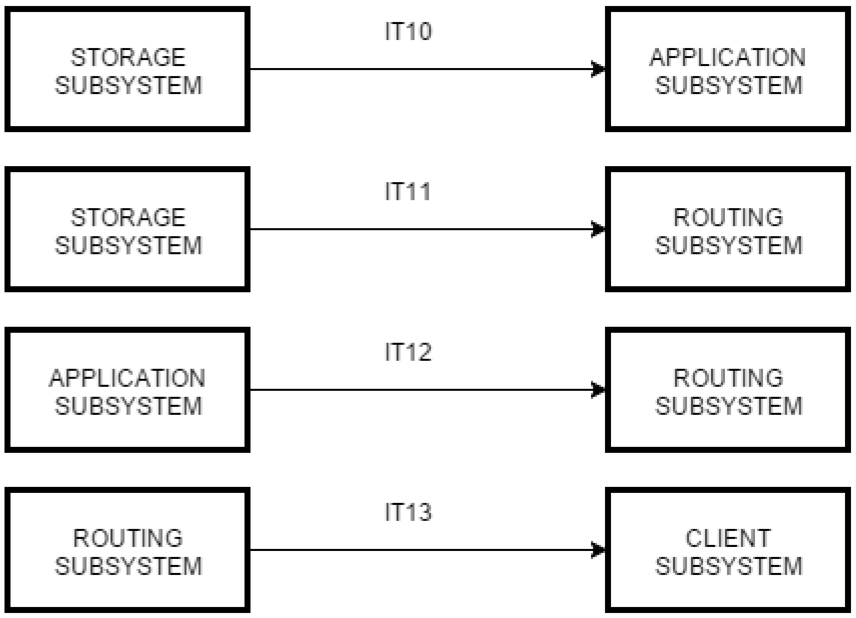


STORAGE SUBSYSTEM AND CLIENT SUBSYSTEM

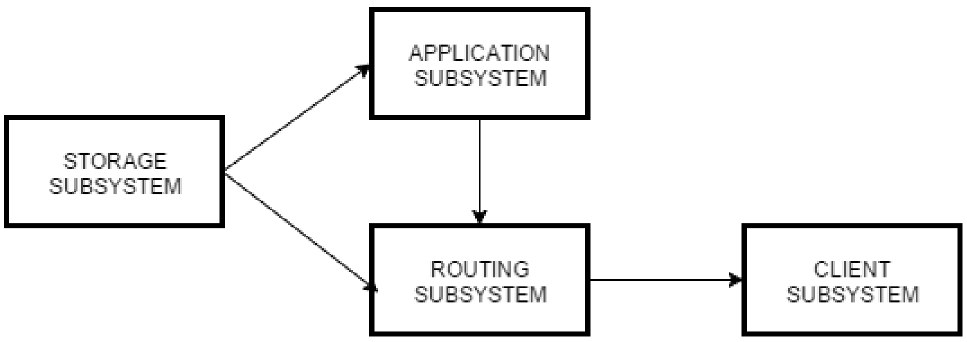
These subsystems do not need any integration at software level.

2.4.2 Subsystem Integrations Sequence

After all the software integrations, we integrate all the subsystems that they form in a unique one:



The entire integrated system is shown below:



3. INDIVIDUAL STEPS AND TEST DESCRIPTION

Test Case Identifier	IT1
Test Item(s)	ZoneManager – QueueManager
Input Specification	Create the typical input for ZoneManager
Output Specification	Check if the correct methods are called in QueueManager
Environmental Needs	/
Purpose	This test case checks whether the calls made by the driver (QueueManager) work as expected. For instance: • getQueue() • getNearestQueue() – (not strictly necessary since this method is not used in QueueManager)

Test Case Identifier	IT2
Test Item(s)	ZoneManager – RideManager
Input Specification	Create the typical input for ZoneManager
Output Specification	Check if the correct methods are called in RideManager
Environmental Needs	/
Purpose	This test case checks whether the calls made by the driver (RideManager) work as expected. For instance: • getQueue() • getNearestQueue()

Test Case Identifier	IT3
Test Item(s)	ZoneManager – PollingManager
Input Specification	Create the typical input for ZoneManager
Output Specification	Check if the correct methods are called in PollingManager
Environmental Needs	/
Purpose	This test case checks whether the calls made by the driver (PollingManager) work as expected. For instance: • getQueue() • getNearestQueue() – (not strictly necessary since this method is not used in PollingManager)

Test Case Identifier	IT4
Test Item(s)	QueueManager – PollingManager
Input Specification	Create the typical input for QueueManager
Output Specification	Check if the correct methods are called in PollingManager
Environmental Needs	IT3 succeded
Purpose	This test case checks whether the calls made by the driver (PollingManager) work as expected. For instance: • addTaxi() • removeTaxi() • getFirst() – (not strictly necessary since this method is not used in PollingManager) • moveLastPosition() – (not strictly necessary since this method is not used in PollingManager)

Test Case Identifier	IT5
Test Item(s)	QueueManager – TaxiManager
Input Specification	Create the typical input for QueueManager
Output Specification	Check if the correct methods are called in TaxiManager
Environmental Needs	IT1 succeded
Purpose	This test case checks whether the calls made by the driver (TaxiManager) work as expected. For instance: • addTaxi() • removeTaxi() • getFirst() – (not strictly necessary since this method is not used in TaxiManager) • moveLastPosition() – (not strictly necessary since this method is not used in TaxiManager)

Test Case Identifier	IT6
Test Item(s)	QueueManager – RideManager
Input Specification	Create the typical input for QueueManager
Output Specification	Check if the correct methods are called in RideManager
Environmental Needs	IT2 succeded
Purpose	This test case checks whether the calls made by the driver (RideManager) work as expected. For instance: • addTaxi() – (not strictly necessary since this method is not used in RideManager) • removeTaxi() • getFirst() • moveLastPosition()

Test Case Identifier	IT7
Test Item(s)	RideManager – TaxiManager
Input Specification	Create the typical input for RideManager
Output Specification	Check if the correct methods are called in TaxiManager
Environmental Needs	IT6 succeded; GPS sample data available (stub)
Purpose	This test case checks whether the calls made by the driver (TaxiManager) work as expected. For instance: • acceptRide() • rejectRide() • endRide() • taxiInRide() • findTaxi() – (not strictly necessary since this method is not used in TaxiManager)

Test Case Identifier	IT8
Test Item(s)	RideManager – Cronjob
Input Specification	Create the typical input for RideManager
Output Specification	Check if the correct methods are called in CronJob
Environmental Needs	IT6 succeded
Purpose	This test case checks whether the calls made by the driver (Cronjob) work as expected. For instance: • acceptRide() – (not strictly necessary since this method is not used in Cronjob) • rejectRide() – (not strictly necessary since this method is not used in Cronjob) • endRide() – (not strictly necessary since this method is not used in Cronjob) • taxiInRide() – (not strictly necessary since this method is not used in Cronjob) • findTaxi()

Test Case Identifier	IT9
Test Item(s)	Security – Dispatcher
Input Specification	Create the typical input for Security
Output Specification	Check if the correct methods are called in Dispatcher
Environmental Needs	/
Purpose	This test case checks whether the calls made by the driver (Dispatcher) work as expected. For instance: • authentication() • authorization()

Test Case Identifier	IT10
Test Item(s)	Storage Subsystem – Application Subsystem
Input Specification	Create the typical input for Storage Subsystem
Output Specification	Check if the correct methods are called in the Application Subsystem and the correct data are retrieved
Environmental Needs	Sample data must be present in the Database
Purpose	Verifies if the Application Subsystem can retrieve data from the Storage Subsystem

Test Case Identifier	IT11
Test Item(s)	Storage Subsystem – Routing Subsystem
Input Specification	Create the typical input for Storage Subsystem
Output Specification	Check if the correct methods are called in the Routing Subsystem
Environmental Needs	Sample data must be present in the Database
Purpose	Verifies if the Routing Subsystem can retrieve data from the Storage Subsystem

Test Case Identifier	IT12
Test Item(s)	Application Subsystem – Routing Subsystem
Input Specification	Create the typical input for Application Subsystem
Output Specification	Check if the correct methods are called in the Routing Subsystem
Environmental Needs	IT1, IT2, IT3, IT4, IT5, IT6, IT7, IT8, IT10 succeded
Purpose	Verifies if Application Subsystem can handle correctly Routing Subsystem methods calls and can properly manage different types of requests

Test Case Identifier	IT13
Test Item(s)	Routing Subsystem – Client Subsystem
Input Specification	Create the typical input for Routing Subsystem
Output Specification	Check if the correct methods are called in the Client Subsystem
Environmental Needs	IT9, IT11, IT12 succeded
Purpose	Verifies if the Client Subsystem can correctly interact with the Routing Subsystem

4. TOOLS AND TEST EQUIPMENT REQUIRED

The software tools used to automate the integration testing are the following:

- Jmeter (http://jmeter.apache.org): it is a tool which may be used to test the performance of the following subsystems:
  - Routing: simulate a heavy load of requests in order to check if the non-functional requirements on the maximum number of simultaneously connected users and on the response times are respected
  - Application: simulate a heavy load of requests on the REST APIs. This subsystem can be also overloaded by a stress test on the Routing subsystem. Tests on both subsystems are useful to identify bottlenecks
  - Storage: check the performance of critical database queries on the test database, in order to know which indexes to add and compare the performance of different equivalent query formulations
- JUnit (http://junit.org): it is the most used framework for unit testing in Java. The use of it in testing the single components is not covered by this document. However, it is important to state that JUnit is also usually used to perform integration testing together with other tools, like Mockito and Arquillian
- Mockito (http://site.mockito.org): it is an open-source test framework useful to generate mock objects, stubs and drivers
- Arquillian (http://arquillian.org/): it is a framework useful to perform integration testing. It provides a series of environment configuration and utilities that can be used to test the integration of the different components of MyTaxiService.
- Manual testing: technique that is needed to simulate the input of the user (passenger or taxi driver) in the relative GUI in order to generate typical input data, useful to integrate the various components of MyTaxiService

5. PROGRAM STUBS AND TEST DATA REQUIRED

Specifications of particular input data or component’s stub/driver needed to perform the integration steps described in chapter 3 are included in the list below:

- Test database: In order to perform some test cases, sample user data should be inserted into the database (DataLayer component) and made available for testing. These test data includes a reduced set of instances of all the entities
- Lightweight API client: in order to test the REST APIs without the actual client applications, a simple API client which interacts with the Routing component by simple HTTP requests is needed. This driver needs to be scriptable in order for the tests to be automated
- External GPS stub: it is needed to replace the external GPS system. This stub should provide sample data needed to the RideManager component in order to correctly perform the test case IT7
- Drivers: generally, if some components may not be available yet for the integration test phase, they will be replaced with appropriate drivers (that take the part of those software component) in order to test the others

6. HOURS OF WORK

Since we are neighbors, we have worked **together almost all the time** at each other’s home and **equally shared all the tasks and efforts**. We have worked on this document **for a total of 9 hours**.

7. REFERENCES

Here is a short list of other references for this document:

- Slides of the Software Engineering 2 course (from the Beep Platform)