myTaxiService
Integration Test Plan

Monica Magoni 854091
Alberto Cibari 852689

21st January 2016

Contents

Page

1 Introduction **3**

1.1 Revision History . . . 3

1.2 Purpose and Scope . . . 3

1.2.1 Purpose . . . 3

1.2.2 Scope . . . 3

1.3 List of Definitions and Abbreviations . . . 4

1.4 List of Reference Documents . . . 5

2 Integration Strategy **6**

2.1 Entry Criteria . . . 6

2.2 Elements to be integrated . . . 6

2.2.1 Client . . . 8

2.2.2 Server . . . 8

2.2.3 Database . . . 9

2.3 Integration Testing Strategy . . . 9

2.4 Sequence of Component/Function Integration . . . 9

2.4.1 Software Integration Sequence . . . 9

2.4.2 Subsystem Integration Sequence . . . 12

3 Individual Steps and Test Description **13**

3.1 Integration Steps Server . . . 13

3.2 Integration Steps Client . . . 17

3.3 Integration Steps Subsystems . . . 19

4 Tools and Test Equipment Required **21**

5 Program Stubs and Test Data Required **22**

1 Introduction

1.1 Revision History

Revision history of the document:

- Version: 1.0.
  Latest update: 21 january 2016.

1.2 Purpose and Scope

1.2.1 Purpose

The purpose of this document is to describe the test plan we have chosen in order to manage the integration tests of our system. Integration tests are done to exercise the interactions of components and subsystems that were presented in the DD.

1.2.2 Scope

The aim of our project is to optimize the taxi service of a large city. Passengers, once registered to the application, will have the possibility to request a taxi through either the web application or the mobile application. When a request from a passenger arrives, the system-to-be will be able to look for an available taxi (through a fair management of the taxi queues) and send to the passenger the code of the incoming taxi and the waiting time.
Moreover, our system will also offer a mobile application for taxi drivers so that they will be able to share their availability and confirm a certain call.
Another functionality that our system will offer to users is the reservation of a taxi. Users will have the possibility to reserve a taxi in advance, by specifying the origin and the destination of their journey. The reservation must occur at least two hours before the ride and the system will inform the taxi driver ten minutes before the scheduled time.

Our system will provide some programmatic interfaces for the development of additional functions, such as taxi sharing.

1.3 List of Definitions and Abbreviations

In order to avoid confusion, we want to specify the definition and abbreviation of some words that will be often used in this document.
List of definitions:

- PASSENGER: for passenger we mean a person, already registered in the system, who has requested or reserved a taxi either through the web or the mobile applications.
- USER: a person who is already registered in the system that can use the application to request or reserve a taxi.
- GUEST: a person who is not registered in the system.
- REQUEST: message sent by the user's application to the system in order to require a taxi or make a reservation.
- CALL: a task received by the taxi drivers after a user made a request.
- TAXI ZONE: for taxi zone we mean a zone that is approximately of 2km2. Each taxi zone has a queue of taxi associated.
- QUEUE: for queue we mean the list of taxi available at a specific moment in a certain zone.

List of abbreviations:

- RASD: Requirements Analysis and Specification Document.
- DD: Design Document.

1.4 List of Reference Documents

List of the documents we referred to:

- AA 2015-2016 Software Engineering 2, Project goal, schedule and rules.
- Requirement Analysis and Specification Document (our first deliverable).
- Design Document (our second deliverable).
- Software Engineering 2 Project, AA 2015-2016 Assignments 4 – Test plan.
- Integration text plan example (spinGRID).

2 Integration Strategy

2.1 Entry Criteria

In this section we define the criteria that must be met before starting the integration tests.
First of all, the RASD, the DD and the Integration Test Plan document must be delivered, complete and updated. Moreover, referring to the code, these criteria must be satisfied:

- Unit tests must be done to classes and functions and cover at least an average of the 90% of the lines of code. This criteria must be met in order to avoid mistakes in the phase of integration: indeed, if no unit tests have been done before the start of the integration, it will probably happen that errors will be located in the single classes or functions. The problem is that, during the integration, it will be very hard to find those mistakes. For instance, simple array-out-of-bound or division by 0 exceptions in a single function might cause the crash of the entire integration test for trivial reasons.
- The JavaDoc must be complete and updated in order to simplify the integration tests. In particular, public classes and public methods should have a detailed JavaDoc documentation.
- The code inspection has to be performed; in this way, some errors or possible errors can be avoided and testers will save time while doing the integration tests.

2.2 Elements to be integrated

Referring to our DD, we have identified the main subsystems that are part of our system and that will need to be integrated:

- Client: it is formed of the components that are located on the client tier.

- Server: this subsystem contains all the components that manage the business logic and the creation of the web interfaces.
- Database: this high-level component contains all the data that have to be kept in the system.

The following image shows the Component Diagram we have presented in DD; we have reported this diagram to remind the detailed interaction between the components.

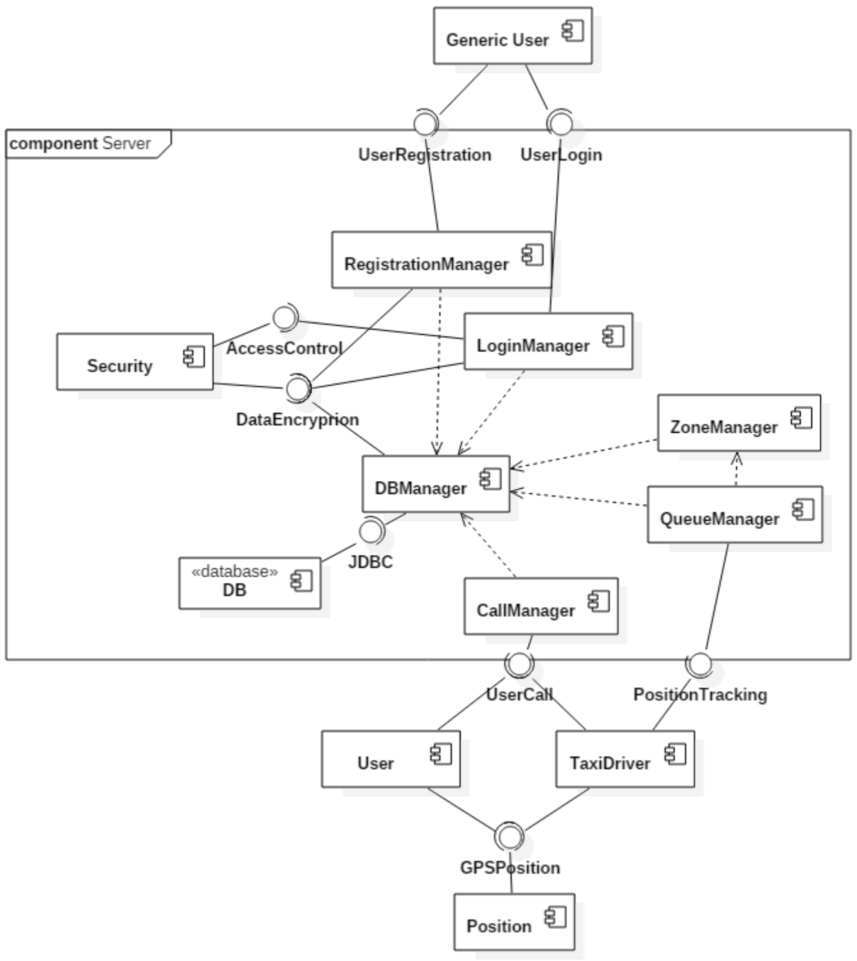

Figure 1: Component Diagram of our system

2.2.1 Client

We now list the components of the client according to their level by starting from the highest level:

- WebManager: it manages the web interfaces on the client side.
- AppManager: it manages the application on the client side.
- Passenger: it contains classes and functions concerning the passengers.
- Taxi: it contains classes and functions concerning the taxi drivers.
- Position: responsible for tracking the position of passengers and taxi.

These components have been deeply described in the DD.

2.2.2 Server

We now list the components of the server according to their level by starting from the highest level:

- Registration Manager: responsible for the registration phase of a guest.
- Login Manager: responsible for the login phase, whether the user is a taxi driver or a passenger.
- Zone Manager: it manages the zones of the city.
- Queue Manager: it manages all the queues in which the city is divided into.
- Call Manager: responsible for managing the calls of passengers.
- DB Manager: it manages the interaction with the database.
- Security: responsible for granting the security to all the system.

These components have been deeply described in the DD.

2.2.3 Database

In this subsystem we have not identified any components because we do not have to develop this part; we assume that the database has been already tested and it need only to be integrated with the system.

2.3 Integration Testing Strategy

The integration test strategy we have chosen is a mixed approach; for the most part, it is a bottom-up strategy but in few cases we have used the top-down approach.
The reason why we have widely used the bottom-up model is that it allows to start integrating from the simplest components of the system. In this way, the complexity of the integration will increase gradually step by step. In few parts of the integration we have preferred a top-down approach because it was more convenient to use stubs instead of drivers.

2.4 Sequence of Component/Function Integration

2.4.1 Software Integration Sequence

In order to show the sequence of the integration, we distinguish into the two subsystem server and client.
In the images shown in this section, an arrow that goes from a general integration test T1 to another integration test T2 means that, before doing the integration test T2, the integration test T1 must be performed.

2.4.1.1 Server

The next figure shows the sequence of the integration of the components of the server.

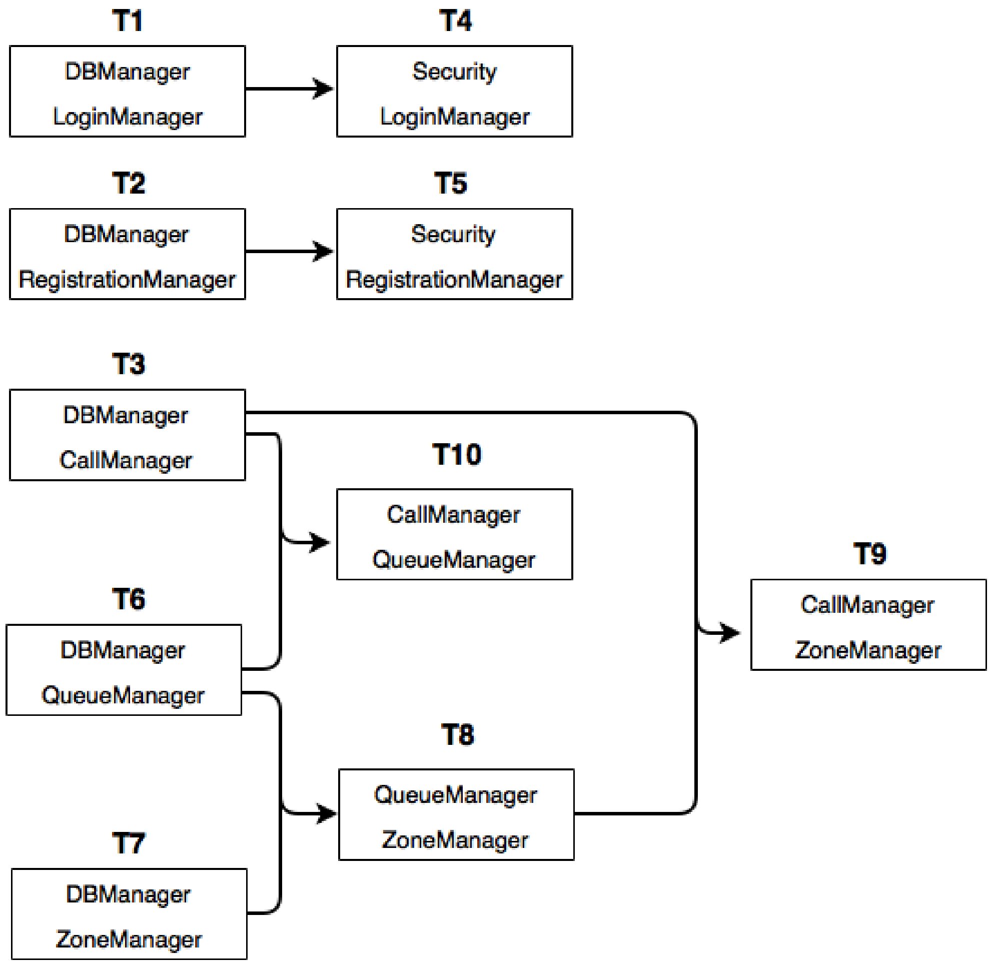

Figure 2: Sequence of integration in the server

2.4.1.2 Client

The next figure shows the sequence of the integration of the components of the client.
In particular, we want to underline that we decided to give priority to tests that concern taxi drivers; indeed, those tests will be performed before the ones regarding passengers. This choice is due to the fact that in our project the central role is played by taxi rather than passengers.

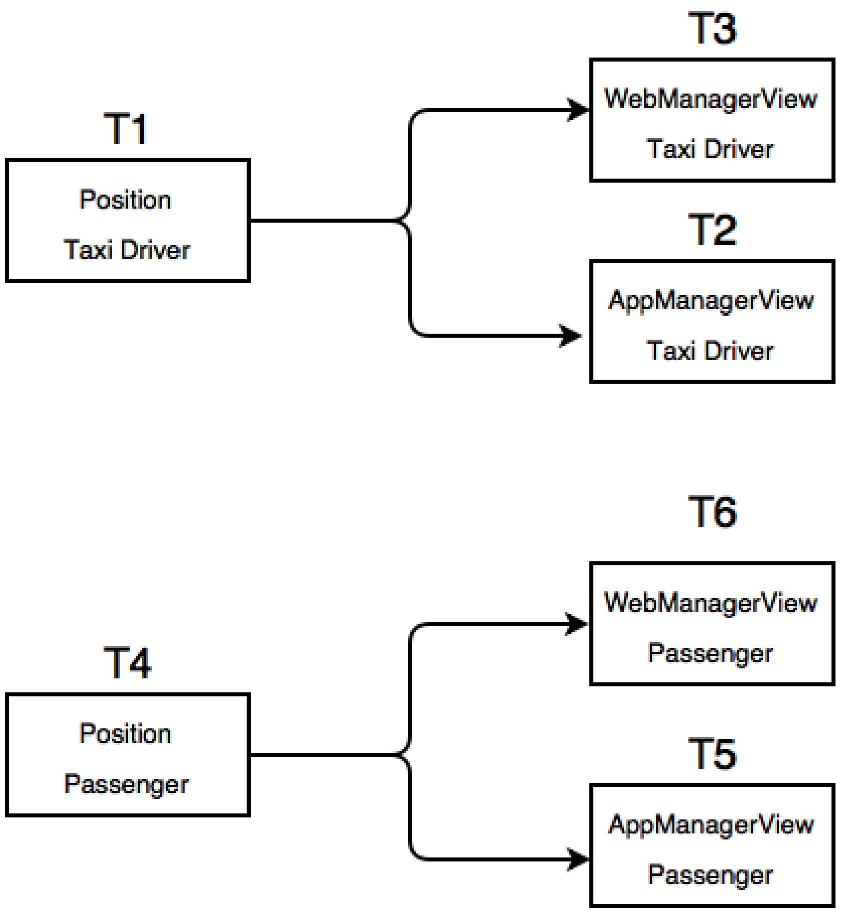

Figure 3: Sequence of integration in the client

2.4.2 Subsystem Integration Sequence

The next figure shows the sequence of the integration of the main subsystems.
The integration of subsystems will start from the database and the server; then, we will integrate the server and the client. This choice seems reasonable since the database and the server are the most important parts of the whole system: the server manages the business logic, while the database manages the data. So, we decided to give highest priority to this two subsystems rather than the client.

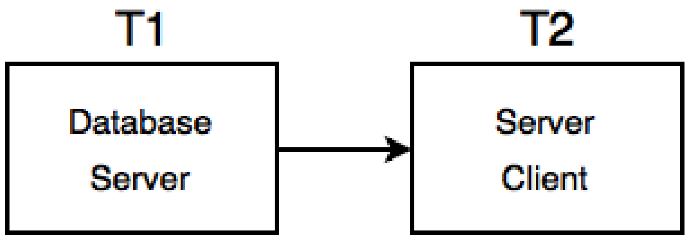

Figure 4: Sequence of integration of the subsystems

3 Individual Steps and Test Description

In this section, we describe what are the individual steps of each integration test. In particular, we will use a table for each of them.
The tables presented from now on are listed according to the sequence presented in the previous section 2.4.1.[1]

3.1 Integration Steps Server

Sequence	Integration test: T1
Components	DBManager - LoginManager
Stubs/Drivers	DBManagerDriver
Input	Actions and data that concern the login process: login, logout.
Output	Checks that given an input action and the data, the information returned are right; in this case: if the operation was successful, it must return the information of the user that tries to login, otherwise it has to return an error.
Test requirements	

[1]Section "Integration Strategy" - Sequence of Component/Function Integration

Sequence	Integration test: T2
Components	DBManager - RegistrationManager
Stubs/Drivers	DBManagerDriver
Input	Actions and data that concern the registration process: name, surname, e-mail, password, phone number and taxi licence ID.
Output	Checks that given an input action and the data, the information returned are right; in this case it has to check if the registration data are saved correctly on the DB.
Test requirements	

Sequence	Integration test: T3
Components	DBManager - Call Manager
Stubs/Drivers	DBManagerDriver
Input	Actions and data that concern the management of a call: request creation, request acceptance, request termination.
Output	Checks that, given an input action and the data, the information returned are right otherwise throws an error.
Test requirements	

Sequence	Integration test: T4
Components	Security - Login Manager
Stubs/Drivers	SecurityStub
Input	Actions and data that concern the login of a user.
Output	Checks that the login data are valid and if already exists a session for the user.
Test requirements	Integration test T1

Sequence	Integration test: T5
Components	Security - Registration Manager
Stubs/Drivers	SecurityStub
Input	Actions and data that concern the registration process.
Output	Checks that the registration data are unique (e-mail and taxi licence ID) and the security of the information.
Test requirements	Integration test T2

Sequence	Integration test: T6
Components	DBManager - QueueManager
Stubs/Drivers	DBManagerDriver
Input	Actions and data that concern the zone management (positions, taxi lists).
Output	DBManager should provide the right information required without errors and in the right format.
Test requirements	

Sequence	Integration test: T7
Components	DBManager - ZoneManager
Stubs/Drivers	DBManagerDriver
Input	Actions and data that concern the queue management (taxi drivers id, taxi positions).
Output	DBManager should provide the right information required without errors and in the right format.
Test requirements	

Sequence	Integration test: T8
Components	QueueManager - ZoneManager
Stubs/Drivers	QueueManagerDriver
Input	Actions and data that concern the taxi driver (GPS position, queue id)
Output	Checks that the status of a taxi driver is changed correctly.
Test requirements	Integration test T6, T7

Sequence	Integration test: T9
Components	CallManager - ZoneManager
Stubs/Drivers	ZoneManagerDriver
Input	The CallManager sends the GPS information to the ZoneManager to look for a specific queue.
Output	The queue identifier that correspond to the GPS position. This queue identifier will be used by the QueueManager to look for the first free TaxiDriver.
Test requirements	Integration test T3, T8

Sequence	Integration test: T10
Components	CallManager - QueueManager
Stubs/Drivers	QueueManagerDriver
Input	The CallManager sends the taxi driver identifier and an action to do to the QueueManager.
Output	The result of the operation otherwise an error.
Test requirements	Integration test T3, T6

3.2 Integration Steps Client

Sequence	Integration test: T1
Components	Position - TaxiDriver
Stubs/Drivers	PositionStub
Input	The action of getting the GPS position of the taxi driver.
Output	The GPS position parsed and ready to be sent to the server.
Test requirements	

Sequence	Integration test: T2
Components	TaxiDriver - AppManagerView
Stubs/Drivers	ManagerViewDriver
Input	Commands to send to the TaxiDriver controller.
Output	The correct controller method is called and the right page fragment is loaded.
Test requirements	T1

Sequence	Integration test: T3
Components	TaxiDriver - WebManagerView
Stubs/Drivers	ManagerViewDriver
Input	Commands to send to the TaxiDriver controller.
Output	The correct controller method is called and the right web page is loaded.
Test requirements	T1

Sequence	Integration test: T4
Components	Position - Passenger
Stubs/Drivers	PositionStub
Input	The action of getting the GPS position of the passenger.
Output	The GPS position parsed and ready to be sent to the server.
Test requirements	

Sequence	Integration test: T5
Components	Passenger - AppManagerView
Stubs/Drivers	ManagerViewDriver
Input	Commands to send to the Passenger controller.
Output	The correct controller method is called and the right page fragment is loaded.
Test requirements	T4

Sequence	Integration test: T6
Components	Passenger - WebManagerView
Stubs/Drivers	ManagerViewDriver
Input	Commands to send to the Passenger controller.
Output	The correct controller method is called and the right web page is loaded.
Test requirements	T4

3.3 Integration Steps Subsystems

Sequence	Integration test: T1
Subsystems	Database - Server
Input	A list SQL queries.
Output	The right data for every query as specified in the query.
Test requirements	

Sequence	Integration test: T2
Subsystems	Server - Client
Input	A set of actions and data regarding: • Registration of a new user. • Login of a user. • Creation of a request. • Locating a user by GPS position.
Output	The expected outputs for each action listed in the 'Input' fields are • The registration result and a mail in case of success. • The login result and a session id in case of success. • The result for the creation of the request otherwise an error. • Acknowledgment message
Test requirements	T1

4 Tools and Test Equipment Required

List of tools that will be used to perform integration testing:

- Manual testing: there will be an accurate selection of the most crucial functionalities (i.e. functions with exceptional parameters) to be manually tested.
- JUnit: although this framework is mainly known for unit tests (indeed we will use it also for unit testing but this is not concerned in this document), it will be also used during the integration phase.
- Mockito: this framework will be used to mock stubs and drivers that are needed in the integration tests.

Moreover, JMeter will be used to set up tests in order to analyze the performances of the system. Indeed, we think that it is very useful to use this framework to verify if the non-functional requirements of our system (described in the RASD) are satisfied.
In particular, with JMeter we will see how the server and the database behave under a heavy load and with a great number of virtual users (simulated with thread group) simultaneously connected.

5 Program Stubs and Test Data Required

Name	DBManagerDriver
Components of reference	DBManager
Purpose	This driver generates various kind of requests (login requests, registration requests, call requests) and send them to the DBManager. Once received the response, it checks if it is correct.

Name	SecurityStub
Components of reference	Security
Purpose	This stub checks that the information that come from the LoginManager and the RegistrationManager is correct before registering or logging in. Also it checks that given the username and password of a user, it has only one session open in the system.

Name	QueueManagerDriver
Components of reference	QueueManager
Purpose	This component simulates various actions to send to the QueueManager and checks the response is correct. • Checks that a taxi driver is correctly added to a queue • Checks that the position of a taxi driver is changed when it changes his status • Checks that the right taxi driver is chosen when a call request is created

Name	ZoneManagerDriver
Components of reference	ZoneManager
Purpose	This component simulates various action to send to the ZoneManager and checks the response from the ZoneManager is correct. • It checks that the the creation of a zone is correct. • Given a GPS position checks that the queue searched by the ZoneManager is correct.

Name	PositionStub
Components of reference	Position
Purpose	This stub simulates a GPS device creating GPS coordinates whenever the TaxiDriver and Passenger components request them, and send them to these component in the right formatted way.

Name	ManagerViewDriver
Components of reference	WebManagerView, AppManagerView
Purpose	This component substitute the view part in a MVC pattern. Its role is to check if the data to print on screen, coming from the TaxiDriver and Passenger components, are formatted in the right way, otherwise returns an error to these components. Also it creates some actions to send to the controller components and checks if the correct methods are called.

Appendix

Tools used

- ShareLaTeX: to write this document. [2]
- GitHub: to share our work. [3]
- Draw.io: to create the images. [4]

Hours of work

- Alberto Cibari: 10 hours
- Monica Magoni: 10 hours

[2]`https://it.sharelatex.com`
[3]`https://github.com`
[4]`https://www.draw.io`