Politecnico di Milano

Software Engineering 2

MyTaxiService

Integration Test Plan Document

Author:
Andrea Tirinzoni

Author:
Matteo Papini

January 19, 2016

Contents

1 Introduction

1.1 Revision History

- version 1.0, dd/01/2016;

1.2 Purpose

This is the Integration Test Plan Document for the myTaxiService system. It is meant to describe the integration testing process of the system's components. This document is used by testers during the integration phase.

1.3 Scope

The aim of this project is to develop a system to improve the taxi service of Milan, accessible both via web and mobile applications. For further details see RASD, section 1.2.

1.4 Definitions, Acronyms and Abbreviations

1.4.1 Definitions

- Entity bean: an object representing data of the system, persisted in the database; for further details, see DD;
- Persistence layer: the set of all entity beans;
- Logic component: a component of the business tier (see DD, section 2.3);
- Web layer: the set of all web tier components (i.e., managed beans and web pages);
- Client layer: the native driver application (the only one having application logic);

1.4.2 Acronyms

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

1.4.3 Abbreviations

1.5 Reference Documents

- RASD.pdf;
- DD.pdf;

2 Integration Strategy

2.1 Entry Criteria

Before the integration process starts, all the components must have been developed and unit tested. In particular, this situation is expected to be reached in the following way:

- First, the entity beans are developed and unit tested;
- Then, the application components (enterpise beans), the web tier components (managed beans and web pages) and the client application (the native app for drivers) are developed. All these features can be developed simultaneously, provided that stubs have been previously created;
- As soon as one of the before mentioned elements is completed, it is unit tested (in order to do that, mocks of the invoked functions may be created);
- All tests must have been succesfully completed.

2.2 Elements to be integrated

The system is composed of two subsystems: the Central System and the Client Application.

The Central System includes the following components to be integrated:

- The persistence layer;
- All the logic components (see DD, section 2.3);
- The web layer;
- The web services;

The Client Application can be identified with a single component, called Client Layer.

2.3 Integration Testing Strategy

The selected strategy for integration testing is bottom-up. The main advantages of this choice are the following:

- The system is built incrementally, starting from components that do not depend on other ones. In this way a working subsystem is available at each step, giving an idea on how the overall project is sorting out;
- The sequence of integration partially mirrors the way the system was designed, and presumably also the implementation process. This make the testing process more intuitive;
- No stubs need to be created;

2.4 Sequence of Component/Function Integration

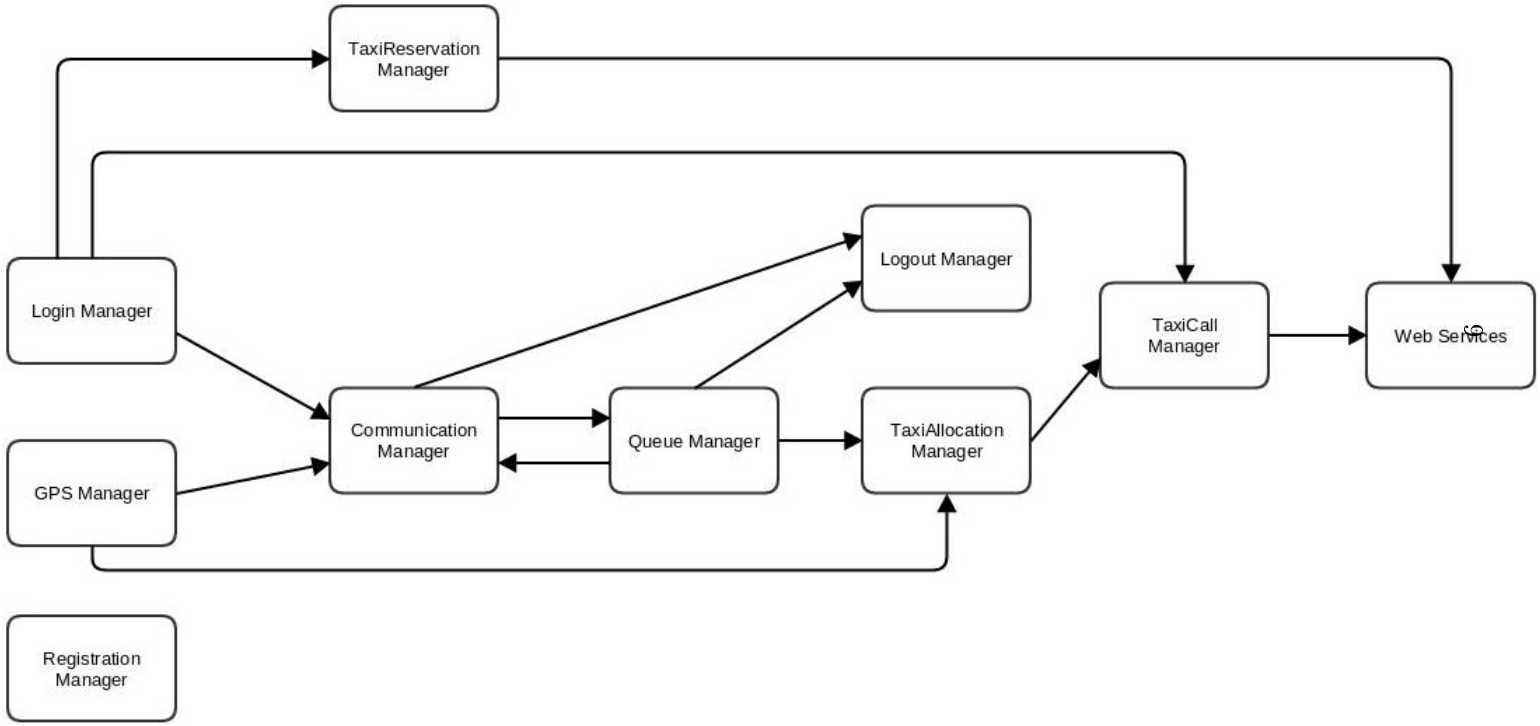
2.4.1 Software Integration Sequence

Central System

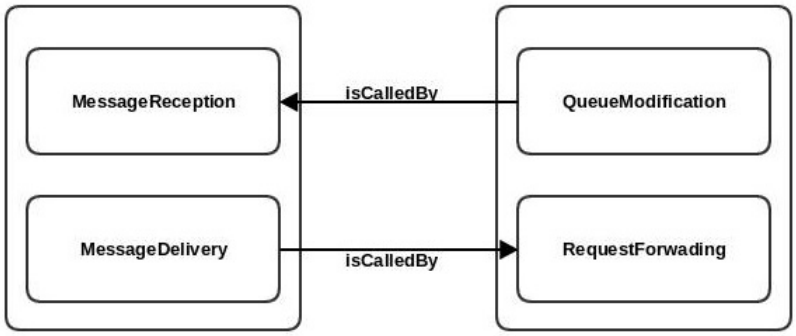
The components are to be integrated according to the following sequence:

- No integration test is needed for the persistence layer with any other component (the unit tests are enough in this case);
- The Login manager, Registration manager, GPS manager are added to the system (note that they do not invoke any other component);
- The Taxi Reservation manager is integrated with the Login manager;
- The Queue manager's QueueModification interface is added to the system (note that this sub-component does not invoke any other component);
- The Communication manager is integrated with the Queue manager's QueueModification interface;
- The Communication manager is integrated with the GPS manager and the Login Manager;
- The Queue manager's RequestForwarding interface is integrated with the Communication manager;
- The Taxi Allocation manager is integrated with the GPS manager and the Queue manager;
- The Logout manager is integrated with the Communication manager and the Queue manager;
- The Taxi Call manager is integrated with the Login manager and the Taxi Allocation manager;
- The Web Services are integrated with the Taxi Reservation manager and the Taxi Call manager;
- The Web Layer is integrated with: Login manager, Registration manager, Taxi Call manager, Taxi Reservation manager;

The dependencies between components are represented in the following picture:


TaxiReservation Manager
Login Manager
Logout Manager
TaxiCall Manager
Web Services
Communication Manager
Queue Manager
TaxiAllocation Manager
GPS Manager
Registration Manager

Note that there is a cyclic dependence. This is easily solved in order to mantain a pure bottom-up approach, as follows: first the Communication Manager is integrated with a subcomponent of the Queue Manager, corresponding to the QueueModification interface (see DD 2.6), then the remaining subcomponent of the QueueManager is integrated with the Communication Manager.

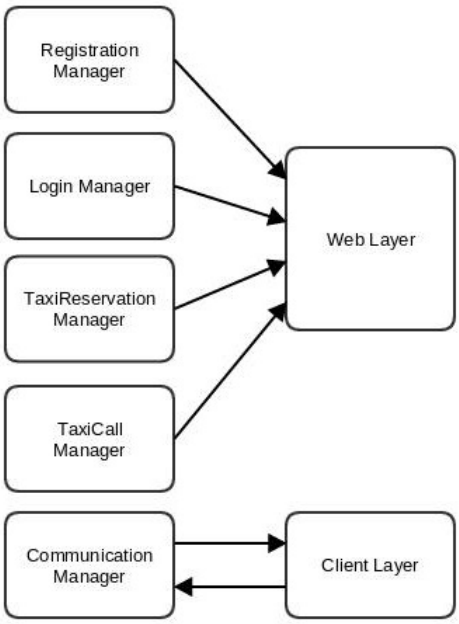


Client Application

Since this subsystem is constituted by a single component, no integration testing is needed locally: it suffices that unit testing has been completed.

2.4.2 Subsystem Integration Sequence

The integration between the two subsystems reduces to the integration between the Client Layer and the Communication Manager:



The cyclic dependence is not a problem in this case since the only two components have already been tested in the previous phase.

3 Individual Steps and Test Description

3.1 Integration Test Description

We provide a description of the tests for each step of the integration. In each one of them, in addition to the call checks specified below, it is also checked that the components exchange compatible data.

3.1.1 TaxiReservation Manager -> Login Manager

- Preconditions: Login Manager integrated with the system;
- Drivers: Web Layer Driver;
- Inputs: the input for a taxi reservation (see DD, section 2.6). Valid data for the requesting client must be present in the database;

- Tests: checks whether isLoggedIn() is correctly called in the Login Manager;

3.1.2 Communication Manager -> Queue Manager(QueueModification)

- Preconditions: the Queue Manager's QueueModification interface has been added to the system;
- Drivers: TaxiDriver Application Driver;
- Inputs: cell phone number(s) (see DD,section 2.6);
- Tests: checks whether addDriver() and removeDriver() are correctly called in the Queue Manager;

3.1.3 Communication Manager -> GPS Manager

- Preconditions: GPS Manager integrated with the system, 3.1.2 has already been performed;
- Drivers: TaxiDriver Application Driver;
- Inputs: cell phone number(s) (see DD,section 2.6);
- Tests: checks whether localize() is correctly called in the GPS Manager;

3.1.4 Communication Manager -> Login Manager

- Preconditions: Login Manager integrated with the system;
- Drivers: TaxiDriver Application Driver;
- Inputs: the input for a login request (see DD,section 2.6);
- Tests: checks whether loginDriver() is correctly called in the Login Manager;

3.1.5 Queue Manager -> Communication Manager

- Preconditions: Communication Manager fully integrated with the system (3.1.2, 3.1.3, 3.1.4 have been successfully executed);
- Drivers: TaxiAllocation Manager Driver;
- Inputs: a request (see DD, section 2.6);
- Tests: checks whether forward() is correctly called in the Queue Manager;

3.1.6 TaxiAllocation Manager -> GPS Manager

- Preconditions: GPS Manager integrated with the system, Queue Manager integrated with the system (3.1.5);
- Drivers: TaxiCall Manager Driver;
- Inputs: a request (see DD, section 2.6);
- Tests: checks whether localize() is correctly called in the GPS Manager;

3.1.7 TaxiAllocation Manager -> Queue Manager

- Preconditions: GPS Manager integrated with the system, Queue Manager integrated with the system (3.1.5);
- Drivers: TaxiCall Manager Driver;
- Inputs: a request (see DD, section 2.6);
- Tests: checks whether forward() is correctly called in the Queue Manager;

3.1.8 Logout Manager -> Communication Manager

- Preconditions: Communication Manager fully integrated with the system (3.1.2, 3.1.3, 3.1.4 have been successfully executed);
- Drivers: none (note that no component calls the Logout Manager);
- Inputs: a TaxiDriver is set to free. Valid data for the driver must be present in the database;
- Tests: checks whether checkStatus() is correctly called in the Communication Manager;

3.1.9 Logout Manager -> Queue Manager

- Preconditions: Queue Manager integrated with the system (3.1.5);
- Drivers: none (note that no component calls the Logout Manager);
- Inputs: a TaxiDriver is set to free. Valid data for the driver must be present in the database;
- Tests: checks whether removeDriver() is correctly called in the Queue Manager;

3.1.10 TaxiCall Manager -> Login Manager

- Preconditions: Login Manager integrated with the system
- Drivers: Web Layer Driver;
- Inputs: the input for a taxi call (see DD, section 2.6). Valid data for the requesting client must be present in the database;
- Tests: checks whether isLoggedIn() is correctly called in the Login Manager;

3.1.11 TaxiCall Manager -> TaxiAllocation Manager

- Preconditions: TaxiAllocation Manager integrated with the system (3.1.6, 3.1.7)
- Drivers: Web Layer Driver, Web Services Driver;
- Inputs: the input for a taxi call (see DD, section 2.6). Valid data for the requesting client must be present in the database;
- Tests: checks whether allocate() is correctly called in the TaxiAllocation Manager;

3.1.12 Web Services -> TaxiCall Manager

- Preconditions: TaxiCall Manager integrated with the system (3.1.10, 3.1.11)
- Drivers: Client Driver (who calls the web services);
- Inputs: the input for a taxi call through the web services (see DD, section 2.6). Valid data for the requesting client must be present in the database;
- Tests: checks whether callExternal() is correctly called in the TaxiCall Manager;

3.1.13 Web Services -> TaxiReservation Manager

- Preconditions: TaxiReservation Manager integrated with the system
- Drivers: Client Driver (who calls the web services);
- Inputs: the input for a taxi reservation through the web services (see DD, section 2.6). Valid data for the requesting client must be present in the database;
- Tests: checks whether reserveExternal() is correctly called in the TaxiReservation Manager;

3.1.14 Web Layer -> Login Manager

- Preconditions: points from 3.1.1 to 3.1.13 have been successfully executed;
- Drivers: Client Driver (who calls the web layer);
- Inputs: the input for a login request (see DD, section 2.6). Valid data for the requesting client must be present in the database;
- Tests: checks whether loginPassenger() is correctly called in the Login Manager;

3.1.15 Web Layer -> Registration Manager

- Preconditions: points from 3.1.1 to 3.1.13 have been successfully executed;
- Drivers: Client Driver (who calls the web layer);
- Inputs: the input for a registration request (see DD, section 2.6). Valid data for the requesting client must be present in the database;
- Tests: checks whether register() is correctly called in the Registration Manager;

3.1.16 Web Layer -> TaxiCall Manager

- Preconditions: points from 3.1.1 to 3.1.13 have been successfully executed;
- Drivers: Client Driver (who calls the web layer);
- Inputs: the input for a taxi call (see DD, section 2.6). Valid data for the requesting client must be present in the database;
- Tests: checks whether call() is correctly called in the TaxiCall Manager;

3.1.17 Web Layer -> TaxiReservation Manager

- Preconditions: points from 3.1.1 to 3.1.13 have been successfully executed;
- Drivers: Client Driver (who calls the web layer);
- Inputs: the input for a reservation (see DD, section 2.6). Valid data for the requesting client must be present in the database;
- Tests: checks whether reserve() is correctly called in the TaxiReservation Manager;

3.1.18 Client Layer -> Communication Manager

- Preconditions: the Central System and the Client Layer have been individually tested (all their components have been integrated);
- Drivers: none (the two systems are indipendently run);
- Inputs: a taxi driver must be registered and present in the database;
- Tests: manually test that remote calls to login(), setFree() and setBusy() work (after setFree(), the queue position must be received and showed);

3.1.19 Communication Manager -> Client Layer

- Preconditions: the Central System and the Client Layer have been individually tested (all their components have been integrated);
- Drivers: none (the two systems are indipendently run);
- Inputs: a taxi driver must be registered and present in the database;
- Tests: manually generate a request and check whether it is correctly forwarded to the driver; manually disconnect the driver and check that it is automatically logged out;

3.2 Final System Check

When all the integration tests have been successfully completed, the final system can be tested as a whole. This is done manually by checking all the system features through the web interface and the mobile interfaces. This is just a simple check to verify whether the final system is running correctly, and it is not meant to substitute the final system validation. For a brief description of system testing, see Appendix (Section 6.1).

4 Tools and Test Equipment Required

The following tools are needed:

- Mokito: in order to create stubs of the subsystems (Central System and Client Application);
- Arquillian: to perform integration testing;

Manual testing is also used in the final subsystem integration (points 3.1.18 and 3.1.19). In this case, manual testing has been chosen because the integration involves already tested subsystems which are deployed on different machines, whose functions can be directly checked without any automatic tool.

5 Program Stubs and Test Data Required

5.1 Program Stubs

The following stubs are needed:

- TaxiDriver Application stub: this is used when integrating the Central System components. It mocks the external interface of the Client Layer (i.e., the remote functions called by the Communication Manager);
- Central System stub: this is used when unit-testing the Client Layer. It mocks the Communication Manager's MessageReception interface, called remotely by the Client Application;
- GPS stub: it mocks a real GPS device from which the GPS Manager obtains its data. It is used during the whole Central System's component integration;
- External Web Service stubs: they mock each external web service called in either of the system's components (e.g., the SMS Web Service, Traffic Web Service, etc.);

5.2 Test Data Required

The required data has already been mentioned for each test in section 3. To summarize, we need:

- Data of at least one registered user (present in the database);
- Data of at least one TaxiDriver (present in the database);
- Valid input data for taxi calls and reservations (i.e., valid meeting time, valid destination, etc.);
- At least one zone and one queue in the system;
- At least one taxi in the system;

6 Appendix

6.1 System Testing

We provide here a brief description of what we expect from the final system tests.
First of all the system is tested in its entirety, considering all the communication interfaces.
Then systems performances are checked against the requirements specified in the RASD document, section 3.3. This could be achieved by using JMeter. Note that further data, not specified in section 5.2, will be required.
Finally, alpha and beta tests are performed for direct validation.

6.2 Spent Hours

- Andrea Tirinzoni: ~15h
- Matteo Papini: ~15h