Software Engineering 2 Project - Integration Test Plan Document

Gabriele Giossi (id: 854216)

v1.0 20/01/2016

Contents

1 Introduction

1.1 Revision History

ver 1.0: initial release and revision 20/01/2016

1.2 Purpose and Scope

The purpose of this Integration Test Plan Document is to provide a guideline for the Integration Test Phase of the MyTaxiService web application; the integration test is a critical testing part of modules, that will test and certify module interaction and capabilities prior to release, in order to make sure there are no critical faults.

As a disclaimer, please note that the present document is subject, as with most of the documentation relative to the MyTaxiService application, to changes, reviews and revisions during all the development phases of the application, and as such, is in continuous evolution, as is the application itself - any change from the original release version of the document will be noted in the Revision History section.

MyTaxiService will be a web-based application, supported both on smartphones and PCs; its aim is, as said, optimizing the taxi service of a city: the system divides the city in which it operates into "zones" of about 2 square kilometers, and to each of them computes a taxi queue

When a passenger, by means of the application, requests a taxi ride, the system assigns an available taxi, the first in queue near the starting point of the ride; the taxi driver will confirm a ride before actually carrying it out.

Some of the benefits of implementing the MyTaxiService system are that it is possible to achieve a real-time balance of numbers of rides requested in a given zone with respect to the available taxis in a zone, providing taxi drivers with an almost-real-time response to spikes of ride requests, improving service quality while simplyfing the interaction of requesting a taxi ride and finding a taxi for any passenger, wherever in the city he might be.

1.3 List of Definitions and Abbreviations

- Users: any human actor that interfaces with the app in order to request or carry out a ride, in short passengers and taxi drivers;
- System: the server-side portion of the application that manages queues, assigns taxis to requests, tracks the cab GPS systems, and manages/stores user data;
- RASD: Requirements Analysis and Specification Document, the document that formalizes the requirements of the system to develop; it will be referred to in several points of this document;

- HTTP: HyperText Transfer Protocol, the cornerstone of the web as we know it, it is the procotol used to send and receive requests that MyTaxiService will need to interface with in order to be a proper web-based service;
- DBMS: DataBase Management System, a system that is used to organize, interrogate to retrieve and present information retrieved from an arbitrarily large organized data amount;
- GPS: Global Positioning System, a triangulation-based system to provide real-time localization in the world of an entity;
- API:Application Program Interface, a set of routines, protocols and tools for building software applications;
- Module/Subsystem: for all intents and purposes, it has been decided that, given the scale of the project, every sub-component of the woftware will be addressed as subsystem or module, with the same meaning: in short, it is assumed there is be no difference between them

1.4 List of Reference Documents

The documents referenced in this Integration Test Plan Document are the following:

- The informal specification for MyTaxiService;
- The RASD (Requirements Analysis and Specification Document), latest version;
- The Design Document (latest version);

All of them are available on the repository on which this document is released.

2 Integration Strategy

2.1 Entry criteria

The general entry criteria for each module of MyTaxiService to enter Integration Test Phase are the following:

- Each module must have been fully implemented in its core functionalities, any accessory or non-critical function can be temporarily replaced by a stub or mock-up; these will be clearly stated andwill be implemented in future releases;
- Each module must have received approval following a code inspection activity, in order to have an a priori knowledge of possible critical cases at code level;
- Each module must have fully passed a suite of unit tests;

If all these conditions hold true, then modules can be submitted for Integration Testing.

2.2 Elements to be integrated

Here follows a list of all components and subsystems that will have to be integrated for MyTaxiService:

1. Web and application servers;
2. Administrator Interface module;
3. Client platforms;
4. Zone manager module;
5. GPS System module;
6. Queue manager module;
7. DBMS and Data Base Structure;
8. Account Manager module;
9. Request manager module;

2.3 Integration testing strategy

The integration testing strategy proposed is a critical modules first strategy: based on the design of the application, the first modules to undergo integration testing will be the most critical ones, whereas the less critical modules will be integrated and tested only after all the more important modules are successfully integrated; the rationale for this choice is that, being MyTaxiService a rather

complex application, but still dependent on several key modules that will handle most, if not all, the data traffic, these must be integrated and thoroughly tested after each integration with each new module in order to be able to discover any possible faults or erratic behaviour caused by a particular module as soon as possible and to attempt making fault detection easier.

The approach is to iteratively deploy (integrate) each module, following a priority list of modules, and then running every time a series of tests that will certify the correct interaction between each module and the newly deployed one - tests will obviously increase in complexity the more modules are fully integrated, but a small series of them will be run after every integration, in order to certify consistency of the application.

2.4 Sequence of Component/Function Integration

2.4.1 Software Integration Sequence

For each module identified in section 2.2, here follows the order in which every software component inside these subsystem will be integrated

1. Servers
   (a) First software integration will be the web and application servers;
   (b) The application core software will then be integrated;
2. Administrator Interface module;
   (a) Statistics evaluation software will be the first integration in the subsystem;
   (b) Diagnostics will follow;
3. Client platforms;
   (a) Client executables will be integrated and their communication to the application core software will be tested;
   (b) First integration will be connection features;
   (c) When all modules are integrated, request and authentication features will be integrated;
4. Zone manager module;
5. GPS System module;
6. Queue manager module;
   (a) Queue manager executable will be integrated, but not tested; it will be tested whether it creates faults/erratic behaviour in the other system software;

7. Database subsystem;

    (a) the database software chosen for the application will be deployed, setup and tested;

8. Account Manager module;

    (a) Account manager software and its interaction with the database will be integrated and tested;

9. Request manager module;

    (a) Its exectuable software will be integrated and tested;

Modules 4 onwards will be very simple to integrate, as they are all mostly devoted to one single function; thus, even listing several functions to be integrated at different steps seemed misleading; they will all be integrated and tested as a monolythic entity.

2.4.2 Subsystem Integration Sequence

As an introductorynote, here it will be explained the order in which the subsystems will be integrated for integration testing, and the tests that will be performed after the integration; note that the tests indicate the communication from a source subsystem/software to a target subsystem/module; details will be refined as development progresses.

1. Server: it is the most critical module and thus will be implemented first - the following tests (internal to the subsystem) will be performed:

    (a) Web server-> application server;

    (b) Application server -> web server;

2. Administrator Interface subsystem: second module to be integrated - the following test will be performed:

    (a) Admin Interface -> Application core

3. Client subsystemss: several types of client will be integrated into the system - with the following tests:

    (a) Client -> Application core

4. Database subsystem: the subsystem will initialize persistency of data and the following test will be carried out:

    (a) Administrator Interface -> Database

5. Account manager module: the module will enable to test authentication and session management - the following test will be carried out:
   (a) Client -> Application core -> Account Manager
   (b) Account Manager -> Application Core -> Client
6. Zone manager:
7. Gps System:
8. Queue Manager:
9. Request Manager: the final subsystem to be integrated will provide a first version of MyTaxiService to be tested - the tests that will be carried out will be:
   (a) Request manager -> Queue manager;
   (b) Request manager -> Application Core;
   (c) Client -> Request Manager;

3 Individual Steps and Test Description

For each specific integration test, these tasks will be needed:

1. Design the integration test;
2. Design drivers and input test data if needed;
3. Setup driver and test data;
4. Integrate subsystem;
5. Perform after-integration tests of already tested parts in order to check for new problems;
6. Perform the actual test.
7. Write and submit a test report that will contain the outcome of the test, any errors encountered and possible solutions;

The tests will be based on 3 different general procedures, and every test carried out will have to refer to one of these procedures, or a combination of them in case the module needs to test many types of interactions; note that the tests have to be FULLY completed successfully for a module to pass the integration test, i.e. if only a single condition for the test is not satisfied the test is not passed and the module is to be reviewed fully starting from unit level - this is subject to decision following the release of test reporting documentation.

3.1 Test Procedure 1

This test procedure will focus on the application core's capacities, it will verify:

- The application core can receive command-line and function call inputs;
- The application core can output requested data correctly;
- The application core can output calls to the correct subsystems;

3.2 Test Procedure 2

This test procedure will focus on client/server communication, it will verify:

- The clients can output requests with the correct protocol to the server;
- The server can receive these requests and correctly output them to the application core;
- The server can output responses to the clients;

3.3 Test Procedure 3

This test procedure will verify data persistency, it will certify:

- The account manager can receive command-line input
- The account manager can receive data to store;
- The account manager can query the database for requested data and retrieve it;
- The account manager can output desired data;

4 Tools and Test Equipment Required

A server machine with Internet access on which the server side components will be integrated will be needed; moreover, it will be necessary to have several other platforms (mobile and desktop), with different operating systems (iOS, Android, Windows, Mac OS) in order to test connectivity and possible erratic interactions from different client versions.

The various machines will communicate through the Internet, thus an active Internet connection is required on all devices that are involved in testing.

To perform the integration tests, a good tool to evaluate for use in this case is Arquillian `http://arquillian.org` - refer to the website for all the specifications of the tool.

Please note that as of the first redaction of this Integration Test Plan document the tests have just been formalized but not designed - thus, it can be possible that during the test design phase new tools, stubs or requirements for the test data or tool may arise - further document releases will include modifications if needed.

5 Program Stubs and Test Data Required

This section will deal with listing any stubs or test data that will be required in order to successfully perform the integration test - these stubs and/or data will simulate runtime configurations and data inputs and will be needed in order to check whether each module receives, interprets and outputs data in the correct way.

Test data Required will be:

1. Several mock-up data forms for accounts in order to test data persistency;
2. Data about the urban context in which MyTaxiService will operate - in short, GPS data of the city and its division into zones.
3. Mock-up data about drivers and taxis;

Program stubs that are required will be:

1. Possible new versions of the client application to be considered for development, in order to check whether different output formats are accepted correctly by the application;

Any other program stub thatis discovered to be necessary further in the development process will be listed here, with a small description.

As a final note, please consider that test data and program stubs will need to be developed and used in the simplest way possible; re-using of stubs and data is strongly suggested, in order to speed up the testing process;

6 Appendix

6.1 Working hours

The team that redacted this document worked towards producing this deliverable the following number of hours;

Gabriele Giossi: 12 hours;