MY TAXI SERVICE

Integration Test Plan Document

Assignment 4

Amos Paribocci (854818); Lorenzo Pinosa (852231)

21/1/2016 - Version 1

Table of contents

1. Introduction

1.1 Revision history

No revision has been produced yet.

1.2 Purpose and scope

This document aims to describe MyTaxiService integration test plans. The software components interfaces should be tested according to what is described in the following paragraphs.

MyTaxiService platform scope has already been detailed in RASD and DD documents.

1.3 List of definitions and abbreviations

- User: any person who interacts with the system;
- Visitor: any user who has not registered an account or has not logged in;
- Customer/passenger: any user that will exploit the service in order to request a taxi to pick him up;
- Taxi driver: user to which customer requests are forwarded selectively by the system, and is expected to fulfill them in practice;
- Taxi zone: area of the logical city subdivision, to be used for taxi selecting and queueing management while serving a customer request;
- Busy (w.r.t. taxi): taxi that has been assigned a task and is still working on it (picking the customer up or moving towards the ride destination);
- Available (w.r.t. taxi): taxi that is ready to accept a new task;
- App: abbreviation for “mobile application”;
- System/backend: the part of the platform that will hold all the data and do all the processing on it, as well as handle instant communication and notification tasks. Mobile apps and the web service will rely upon this to work;
- MyTaxiService components: separate entities which cooperate to get MyTaxiService working and useful. They are the System Backend, the Customer Mobile App, the Driver Mobile App and the Web Service.
- Platform: what all MyTaxiService components build up;
- RASD: Requirements Analysis and Specification Document;
- DD: Design Document.

1.4 List of reference documents

- “Assignment 4 – Test Plan document”, provided in the Software Engineering 2 course context;
- MyTaxiService RASD (Requirements Analysis and Specifications Document);
- MyTaxiService DD (Design Document);

2. Integration strategy

2.1 Entry criteria

This document assumes that all MyTaxiService components and modules - i.e. System Backend modules, Web Service, Customer Mobile App and Taxi Driver Mobile App - are in a final development stage and ready for integration testing. This means that also Unit Testing has been already performed on each of them.

2.2 Elements to be integrated

Purpose of this document is to set up test that will cover the entire MyTaxiService platform. All the components and modules of MyTaxiService will be tested (see [DD] for further explanation and information about components and modules).

2.3 Integration testing strategy

From the highest point of view, tests will first fully cover the System Backend. Then the Backend will be considered as a single monolithic block, and tests will carry on covering the other components: the Mobile Apps and the Web Service.

The used testing strategy will be a functional one. This decision was strongly influenced by the System Backend design: since the modules composing it are not organized in a hierarchical structure but are rather cooperating modules carrying out one single functionality each, bottom-up or top-down testing strategies are inapplicable. Functional testing will refer to use cases and requirements detailed in [RASD], [DD].

Single function testing will follow an incremental approach. This means that if more than two System Backend modules are required to carry out a specific functionality, the first test case will cover the two exploited core modules and the others will be mocked using drivers/stubs written for unit tests. After that, in each following test case, one more real module will replace a stub and everything will be tested again.

Following the incremental approach described above, it could happen that a test case fails because two subsequently added modules have problems working with each other and not because they don't work with firstly added ones. To avoid this unwanted condition, we will take care of specifying additional smaller functional test cases and run them before bigger test cases that rely upon that feature. This will guarantee test cases consistency. An example is the feature test 2.4.1.1.

2.4 Sequence of tested features

2.4.1 System Backend

2.4.1.1 Notification feature

This functional test will prove that the Communication Module and the Notification module are able to correctly interact with each other. This will ensure that, when these modules are tested together with other ones, an eventual test failure is not due to a wrong interaction between them.

2.4.1.2 Taxi request Feature

Since almost all System Backend are exploited in order to carry out a Taxi Request, the incremental strategy is mandatory. To define the modules integration order we looked at the Taxi Request Sequence

Diagram (see [RASD], p. 16) and started by analyzing the flow of calls originated by Request Handler Module. We decided to include modules in test cases in the same order they are called/used by Request Handler, and to get the processing started by a direct Request Handler API call. Instead, the last test case will even integrate the Communication Manager such that the entire process is triggered by a message received by the Communication Manager itself, simulating a real-world scenario.

2.4.1.3 Taxi booking feature

Taxi booking feature is split in two different capabilities: one is related to Bookings management, such as booking creation, retrieval by user, modification and deleting, and the other is related to schedule code execution to effectively trigger a Taxi Request when needed.

We reflected this subdivision in test cases, testing first Bookings Management and then Taxi Request scheduling.

Incremental testing strategy was applied to Booking Management by only testing first Request Handler and Booking Management Module together and then by also integrating Communication Module. This way, as for Taxi Request feature, processing will first be triggered by a Test Driver call to Request Handler API and then by a message received by Communication Manager, as it will at run-time. For each phase there will be a test case for booking creation, user bookings retrieval, booking modification and deletion, for a total of 8 test cases.

The 9th test case will ensure that Taxi Booking module is able to correctly interact with the Request Handler to place a Taxi Request, as scheduled.

2.4.1.4 Update taxi position feature

Updating a Taxi Location requires the cooperation of three modules: Communication Module, Request Handler and Taxi Location Handler. Following the same rationale explained in 2.4.1.2 and 2.4.1.3 feature, we tested first Request Handler and Taxi Location Handler and then included also the Communication Module.

2.4.1.5 Plug-in management feature

Using a stub of a plug-in, here we will test that the Plug-in Manager is able to correctly hook methods of all modules and call plug-in methods. There is a test case for each System Backend module.

2.4.1.6 Service registry feature

Purpose of these test is to check that all modules can correctly publish their services to the Service Registry. There is a test case for each System Backend module.

2.4.2 Customer Mobile App

In this section there are test cases useful to check if the Mobile App can correctly communicate with the System Backend to carry out certain features.

2.4.2.1 Taxi booking feature

Retrieve, post, edit and delete Taxi Bookings.

2.4.2.2 Taxi request feature

Post a new Taxi Request.

2.4.2.3 Receive notification feature

Receive a notification, such as incoming taxi or no taxi available.

2.4.2.4 Display ride history feature
Retrieve the list of previous rides.

2.4.3 Taxi Driver Mobile App
As for Customer Mobile App, in this section there are test cases useful to check if the App can correctly communicate with the System Backend to carry out the following features.

2.4.3.1 Receive notification feature
Receive a notification, such as an incoming taxi request.

2.4.3.2 Send notification feature
Send notifications, such as taxi request accepted notification.

2.4.3.3 Update Taxi Location feature
Update the taxi location.

2.4.4 Web Service
Even the Web Service communication with System Backend needs to be tested, by checking features 2.4.4.1-2.4.4.3. We decided to use the Web Service to stress test the System Backend, with test 2.4.4.4

2.4.4.1 Taxi booking feature
Retrieve, post, edit and delete Taxi Bookings.

2.4.4.2 Taxi request feature
Post a new Taxi Request.

2.4.4.3 Display ride history feature
Retrieve the list of previous rides.

2.4.4.4 Performance test
By simulating multiple concurrent requests to the Web Service, easy to do with already-existent tools, we can even stress test the System Backend and analyze if the results comply with what stated in [RASD].

3. Individual test description

3.1 System Backend features

3.1.1 Send notification feature

3.1.1.1 Send notification feature, Test 1

3.1.1.1.1 Test case identifier

BF1T1

3.1.1.1.2 Test item(s)

`NotificationModule, CommunicationModule`

3.1.1.1.3 Input specification

Call `sendNotificationToCustomer` on `NotificationModule`.

3.1.1.1.4 Output specification

Check that the `NotificationModule` correctly calls the `sendNotification` method in `CommunicationModule`.

3.1.1.1.5 Environmental needs

Demo Customer (see chapter 6).

3.1.1.2 Send notification feature, Test 2

3.1.1.2.1 Test case identifier

BF1T2

3.1.1.2.2 Test item(s)

`NotificationModule, CommunicationModule`

3.1.1.2.3 Input specification

The `CommunicationModule` is asked to elaborate a Customer notification.

3.1.1.2.4 Output specification

Check that `CommunicationModule` correctly calls `NotificationModule` to deliver the received notification.

3.1.1.2.5 Environmental needs

Demo Customer, BF1T1 succeeded.

3.1.2 Taxi request feature

3.1.2.1 Taxi request feature, Test 1

3.1.2.1.1 Test case identifier

BF2T1

3.1.2.1.2 Test item(s)

`RequestHandler, TaxiLocationHandler`

3.1.2.1.3 Input specification

Call `createNewRequest` on `RequestHandler`.

3.1.2.1.4 Output specification

Check that `RequestHandler` correctly calls `findTaxis` method in `TaxiLocationHandler`.

3.1.2.1.5 Environmental needs

Mock taxi data and geospatial data. `TaxiSelectionHandler, NotificationModule, CommunicationModule` stubs.

3.1.2.2 Taxi request feature, Test 2

3.1.2.2.1 Test case identifier

BF2T2

3.1.2.2.2 Test item(s)

`RequestHandler, TaxiLocationHandler, TaxiSelectionHandler`

3.1.2.2.3 Input specification

Call `createNewRequest` on `RequestHandler`.

3.1.2.2.4 Output specification

Check that `RequestHandler` correctly calls `selectTaxi` method in `TaxiSelectionHandler`.

3.1.2.2.5 Environmental needs

Mock taxi data and geospatial data, test BF2T1 succeeded. `NotificationModule`, `CommunicationModule` stub (see 6.1).

3.1.2.3 Taxi request feature, Test 3

3.1.2.3.1 Test case identifier

BF2T3

3.1.2.3.2 Test item(s)

`RequestHandler, TaxiLocationHandler, TaxiSelectionHandler, NotificationModule`

3.1.2.3.3 Input specification

Call `createNewRequest` on `RequestHandler`.

3.1.2.3.4 Output specification

Check that `NotificationModule` correctly calls `sendNotificationToCustomer` method in `RequestHandler`.

3.1.2.3.5 Environmental needs

Mock taxi data, mock taxi driver data, test BF2T2 succeeded. `CommunicationModule` stub.

3.1.2.4 Taxi request feature, Test 4

3.1.2.4.1 Test case identifier

BF2T4

3.1.2.4.2 Test item(s)

`RequestHandler, TaxiLocationHandler, TaxiSelectionHandler, NotificationModule, CommunicationModule`

3.1.2.4.3 Input specification

The `CommunicationModule` is asked to elaborate a Taxi Request.

3.1.2.4.4 Output specification

Check that the `CommunicationModule` correctly calls the `createNewRequest` method in `RequestHandler`, and that method is executed as predicted while calling all other modules.

3.1.2.4.5 Environmental needs

Demo Customer (see chapter 6).

3.1.2.4.6 Environmental needs

Mock taxi data, mock taxi driver data, test BF2T3 succeeded.

3.1.3 Taxi booking feature

3.1.3.1 Taxi booking feature, Test 1

3.1.3.1.1 Test case identifier

BF3T1

3.1.3.1.2 Test item(s)

`RequestHandler, BookingManagementModule`

3.1.3.1.3 Input specification

Call `createNewBooking` on `RequestHandler`.

3.1.3.1.4 Output specification

Check that `RequestHandler` correctly calls `addBookingRequest` method in `BookingManagementModule`.

3.1.3.1.5 Environmental needs

Demo Customer, mock geospatial data.

3.1.3.2 Taxi booking feature, Test 2

3.1.3.2.1 Test case identifier

BF3T2

3.1.3.2.2 Test item(s)

`RequestHandler, BookingManagementModule`

3.1.3.2.3 Input specification

Call `modifyBooking` on `RequestHandler`.

3.1.3.2.4 Output specification

Check that `RequestHandler` correctly calls `modifyBooking` method in `BookingManagementModule`.

3.1.3.2.5 Environmental needs

Demo Customer, mock geospatial and booking data.

3.1.3.3 Taxi booking feature, Test 3

3.1.3.3.1 Test case identifier

BF3T3

3.1.3.3.2 Test item(s)

`RequestHandler, BookingManagementModule`

3.1.3.3.3 Input specification

Call `removeBookingRequest` on `RequestHandler`.

3.1.3.3.4 Output specification

Check that `RequestHandler` correctly calls `removeBookingRequest` method in `BookingManagementModule`.

3.1.3.3.5 Environmental needs

Mock booking data.

3.1.3.4 Taxi booking feature, Test 4

3.1.3.4.1 Test case identifier

BF3T4

3.1.3.4.2 Test item(s)

`RequestHandler, BookingManagementModule`

3.1.3.4.3 Input specification

Call `getBookingRequestList` on `RequestHandler`.

3.1.3.4.4 Output specification

Check that `RequestHandler` correctly calls `getBookingRequestList` method in `BookingManagementModule`.

3.1.3.4.5 Environmental needs

Mock booking data.

3.1.3.5 Taxi booking feature, Test 5

3.1.3.5.1 Test case identifier

BF3T5

3.1.3.5.2 Test item(s)

`RequestHandler, BookingManagementModule, CommunicationModule`

3.1.3.5.3 Input specification

The `CommunicationModule` is asked to elaborate a user request of a new booking creation.

3.1.3.5.4 Output specification

Check that the `CommunicationModule` correctly calls `createNewBooking` on the `RequestHandler` and reply to the request.

3.1.3.5.5 Environmental needs

Demo Customer, mock geospatial data, test BF3T1 succeeded.

3.1.3.6 Taxi booking feature, Test 6

3.1.3.6.1 Test case identifier

BF3T6

3.1.3.6.2 Test item(s)

`RequestHandler, BookingManagementModule, CommunicationModule`

3.1.3.6.3 Input specification

The `CommunicationModule` is asked to elaborate a user request to modify a booking.

3.1.3.6.4 Output specification

Check that the `CommunicationModule` correctly calls `modifyBooking` on the `RequestHandler` and reply to the request.

3.1.3.6.5 Environmental needs

Demo Customer, mock geospatial and booking data, test BF3T2 succeeded.

3.1.3.7 Taxi booking feature, Test 7

3.1.3.7.1 Test case identifier

BF3T7

3.1.3.7.2 Test item(s)

`RequestHandler, BookingManagementModule, CommunicationModule`

3.1.3.7.3 Input specification

The `CommunicationModule` is asked to elaborate a user request to cancel a booking.

3.1.3.7.4 Output specification

Check that the `CommunicationModule` correctly calls `removeBookingRequest` on the `RequestHandler` and reply to the request.

3.1.3.7.5 Environmental needs

Mock booking data, test BF3T3 succeeded.

3.1.3.8 Taxi booking feature, Test 8

3.1.3.8.1 Test case identifier

BF3T8

3.1.3.8.2 Test item(s)

`RequestHandler, BookingManagementModule, CommunicationModule`

3.1.3.8.3 Input specification

The `CommunicationModule` is asked to elaborate a user request of bookings retrieval.

3.1.3.8.4 Output specification

Check that the `CommunicationModule` correctly calls `getBookingRequestList` on the `RequestHandler` and reply to the request with the retrieved data.

3.1.3.8.5 Environmental needs

Mock booking data, test BF3T4 succeeded. Backend modules stub (see 6.1).

3.1.3.9 Taxi booking feature, Test 9

3.1.3.9.1 Test case identifier

BF3T9

3.1.3.9.2 Test item(s)

`RequestHandler, BookingManagementModule`

3.1.3.9.3 Input specification

None. The `BookingManagerModule` should wake up automatically

3.1.3.9.4 Output specification

Check that `BookingManagementModule` correctly awake at the right timestamp (as requested by the user) and calls `createNewRequest` on `requestHandler`.

3.1.3.9.5 Environmental needs

Demo Customer, mock bookings data.

3.1.4 Update taxi position feature

3.1.4.1 Update taxi position feature, Test 1

3.1.4.1.1 Test case identifier

BF4T1

3.1.4.1.2 Test item(s)

`RequestHandler, TaxiLocationHandler`

3.1.4.1.3 Input specification

Call `updateTaxiPosition` on `RequestHandler`.

3.1.4.1.4 Output specification

Check that `RequestHandler` correctly calls `updateTaxiPosition` on `TaxiLocationHandler`.

3.1.4.1.5 Environmental needs

Demo taxi driver.

3.1.4.2 Update taxi position feature, Test 2

3.1.4.2.1 Test case identifier

BF4T2

3.1.4.2.2 Test item(s)

`RequestHandler, TaxiLocationHandler, CommunicationModule`

3.1.4.2.3 Input specification

A smartphone running the Taxi Driver Mobile App changes his position.

3.1.4.2.4 Output specification

Check that the `CommunicationModule` calls `updateTaxiPosition` in `RequestHandler`, which correctly calls `updateTaxiPosition` method in `TaxiLocationHandler`.

3.1.4.2.5 Environmental needs

Mock taxi and taxi driver data, test BF4T1 succeeded.

3.1.5 Plugin management feature

3.1.5.1 Plugin management, Test 1

3.1.5.1.1 Test case identifier

BF5T1

3.1.5.1.2 Test item(s)

`PluginManager, RequestHandler`

3.1.5.1.3 Input specification

Call all methods that trigger a plug-in callback on `RequestHandler`.

3.1.5.1.4 Output specification

Check that the `RequestHandler` correctly calls `onModulePluginCallback` on `PluginManager` and, after the `PluginManager` loads and executes the plugin, the computation control returns back to `RequestHandler`.

3.1.5.1.5 Environmental needs

Plug-in stub.

3.1.5.2 Plugin management, Test 2

3.1.5.2.1 Test case identifier

BF5T2

3.1.5.2.2 Test item(s)

`PluginManager, CommuniationModule`

3.1.5.2.3 Input specification

Call all methods that trigger a plug-in callback on `CommunicationModule`.

3.1.5.2.4 Output specification

Check that the `CommunicationModule` correctly calls `onModulePluginCallback` on `PluginManager` and, after the `PluginManager` loads and executes the plugin, the computation control returns back to `CommunicationModule`.

3.1.5.2.5 Environmental needs

Plug-in stub.

3.1.5.3 Plugin management, Test 3

3.1.5.3.1 Test case identifier

BF5T3

3.1.5.3.2 Test item(s)

`PluginManager, TaxiLocationHandler`

3.1.5.3.3 Input specification

Call all methods that trigger a plug-in callback on `TaxiLocationHandler`.

3.1.5.3.4 Output specification

Check that the `TaxiLocationHandler` correctly calls `onModulePluginCallback` on `PluginManager` and, after the `PluginManager` loads and executes the plugin, the computation control returns back to `TaxiLocationHandler`.

3.1.5.3.5 Environmental needs

Plug-in stub.

3.1.5.4 Plugin management, Test 4

3.1.5.4.1 Test case identifier

BF5T4

3.1.5.4.2 Test item(s)

`PluginManager, TaxiSelectionHandler`

3.1.5.4.3 Input specification

Call all methods that trigger a plug-in callback on `TaxiSelectionHandler`.

3.1.5.4.4 Output specification

Check that the `TaxiSelectionHandler` correctly calls `onModulePluginCallback` on `PluginManager` and, after the `PluginManager` loads and executes the plugin, the computation control returns back to `TaxiSelectionHandler`.

3.1.5.4.5 Environmental needs

Plug-in stub.

3.1.5.5 Plugin management, Test 5

3.1.5.5.1 Test case identifier

BF5T5

3.1.5.5.2 Test item(s)

`PluginManager, NotificationModule`

3.1.5.5.3 Input specification

Call all methods that trigger a plug-in callback on `NotificationModule`.

3.1.5.5.4 Output specification

Check that the `NotificationModule` correctly calls `onModulePluginCallback` on `PluginManager` and, after the `PluginManager` loads and executes the plugin, the computation control returns back to `NotificationModule`.

3.1.5.5.5 Environmental needs

Plug-in stub.

3.1.5.6 Plugin management, Test 6

3.1.5.6.1 Test case identifier

BF5T6

3.1.5.6.2 Test item(s)

`PluginManager, BookingManagementModule`

3.1.5.6.3 Input specification

Call all methods that trigger a plug-in callback on `BookingManagementModule`.

3.1.5.6.4 Output specification

Check that the `NotificationModule` correctly calls `onModulePluginCallback` on `PluginManager` and, after the `PluginManager` loads and executes the plugin, the computation control returns back to `BookingManagementModule`.

3.1.5.6.5 Environmental needs

Plug-in stub.

3.1.6 Service registry feature

3.1.6.1 Service registry feature, Test 1

3.1.6.1.1 Test case identifier

BF6T1

3.1.6.1.2 Test item(s)

`ServiceRegistry, RequestHandler`

3.1.6.1.3 Input specification

Initialize the `RequestHandler`.

3.1.6.1.4 Output specification

Check that the `RequestHandler` services are correctly forwarded to the `ServiceRegistry`.

3.1.6.1.5 Environmental needs

None.

3.1.6.2 Service registry feature, Test 2

3.1.6.2.1 Test case identifier

BF6T2

3.1.6.2.2 Test item(s)

`ServiceRegistry, CommuniationModule`

3.1.6.2.3 Input specification

Initialize the `CommunicationModule`.

3.1.6.2.4 Output specification

Check that the `CommunicationModule` services are correctly forwarded to the `ServiceRegistry`.

3.1.6.2.5 Environmental needs

None.

3.1.6.3 Service registry feature, Test 3

3.1.6.3.1 Test case identifier

BF6T3

3.1.6.3.2 Test item(s)

`ServiceRegistry, TaxiLocationHandler`

3.1.6.3.3 Input specification

Initialize the `TaxiLocationHandler`.

3.1.6.3.4 Output specification

Check that the `TaxiLocationHandler` services are correctly forwarded to the `ServiceRegistry`.

3.1.6.3.5 Environmental needs

None.

3.1.6.4 Service registry feature, Test 4

3.1.6.4.1 Test case identifier

BF6T4

3.1.6.4.2 Test item(s)

`ServiceRegistry, TaxiSelectionHandler`

3.1.6.4.3 Input specification

Initialize the `TaxiSelectionHandler`.

3.1.6.4.4 Output specification

Check that the `TaxiSelectionHandler` services are correctly forwarded to the `ServiceRegistry`.

3.1.6.4.5 Environmental needs

None.

3.1.6.5 Service registry feature, Test 5

3.1.6.5.1 Test case identifier

BF6T5

3.1.6.5.2 Test item(s)

`ServiceRegistry, NotificationModule`

3.1.6.5.3 Input specification

Initialize the `NotificationModule`.

3.1.6.5.4 Output specification

Check that the `NotificationModule` services are correctly forwarded to the `ServiceRegistry`.

3.1.6.5.5 Environmental needs

None.

3.1.6.6 Service registry feature, Test 6

3.1.6.6.1 Test case identifier

BF6T6

3.1.6.6.2 Test item(s)

`ServiceRegistry, BookingManagementModule`

3.1.6.6.3 Input specification

Initialize the `BookingManagementModule`.

3.1.6.6.4 Output specification

Check that the `BookingManagementModule` services are correctly forwarded to the `ServiceRegistry`.

3.1.6.6.5 Environmental needs

None.

3.2 Customer Mobile App features

3.2.1 Taxi booking feature

3.2.1.1 Taxi booking feature, Test 1

3.2.1.1.1 Test case identifier

CF1T1

3.2.1.1.2 Test item(s)

Customer Mobile App (using GUI for input), System Backend

3.2.1.1.3 Input specification

The mobile app user submits a new taxi booking.

3.2.1.1.4 Output specification

The mobile app user is notified about the request success and the System Backend data is changed and consistent with the user action (data consistency between server and smartphone).

3.2.1.1.5 Environmental needs

Demo Customer stored in the Backend.

3.2.1.2 Taxi booking feature, Test 2

3.2.1.2.1 Test case identifier

CF1T2

3.2.1.2.2 Test item(s)

Customer Mobile App (using GUI for input), System Backend

3.2.1.2.3 Input specification

The mobile app user edits an existing taxi booking.

3.2.1.2.4 Output specification

The mobile app user is notified about the operation success and the System Backend data is changed and consistent with the user action (data consistency between server and smartphone).

3.2.1.2.5 Environmental needs

Demo Customer, existing taxi booking data from the user (test CF1T1 succeeded).

3.2.1.3 Taxi booking feature, Test 3

3.2.1.3.1 Test case identifier

CF1T3

3.2.1.3.2 Test item(s)

Customer Mobile App (using GUI for input), System Backend

3.2.1.3.3 Input specification

The mobile app user deletes an existing taxi booking.

3.2.1.3.4 Output specification

The mobile app user is notified about the operation success and the System Backend data is changed and consistent with the user action (data consistency between server and smartphone).

3.2.1.3.5 Environmental needs

Demo Customer, existing taxi booking data from the user (test CF1T1 succeeded).

3.2.1.4 Taxi booking feature, Test 4

3.2.1.4.1 Test case identifier

CF1T4

3.2.1.4.2 Test item(s)

Customer Mobile App (using GUI for input), System Backend

3.2.1.4.3 Input specification

The mobile app user request his bookings history.

3.2.1.4.4 Output specification

The mobile app user is shown the requested data, which is consistent with the data stored in the Backend.

3.2.1.4.5 Environmental needs

Demo Customer, existing taxi booking data from the user (test CF1T1 succeeded).

3.2.2 Taxi request feature

3.2.2.1 Taxi request feature, Test 1

3.2.2.1.1 Test case identifier

CF2T1

3.2.2.1.2 Test item(s)

Customer Mobile App (using GUI for input), System Backend

3.2.2.1.3 Input specification

The mobile app user asks for a new taxi request.

3.2.2.1.4 Output specification

The mobile app user is notified about the operation success and the System Backend starts processing the taxi request (i.e. test 3.1.2.4).

3.2.2.1.5 Environmental needs

Demo Customer stored in the Backend.

3.2.3 Receive notification feature

3.2.3.1 Receive notification feature, Test 1

3.2.3.1.1 Test case identifier

CF3T1

3.2.3.1.2 Test item(s)

Customer Mobile App (using GUI for input), System Backend

3.2.3.1.3 Input specification

Refer to Test Procedure 4.2.

3.2.3.1.4 Output specification

Check that the Customer correctly receives a notification from the Backend.

3.2.3.1.5 Environmental needs

Demo Customer stored in the Backend.

3.2.4 Display ride history feature

3.2.4.1 Display ride history feature, Test 1

3.2.4.1.1 Test case identifier

CF5T1

3.2.4.1.2 Test item(s)

Customer Mobile App (using GUI for input), System Backend

3.2.4.1.3 Input specification

The mobile app user asks for the ride history.

3.2.4.1.4 Output specification

The mobile app user is shown his ride history, consistent to which is stored in the System Backend.

3.2.4.1.5 Environmental needs

Demo Customer, existing ride history for the user.

3.3 Taxi Driver Mobile App features

3.3.1 Receive notifications feature

3.3.1.1 Receive notifications feature, Test 1

3.3.1.1.1 Test case identifier

DF1T1

3.3.1.1.2 Test item(s)

Taxi Driver Mobile App (using GUI for input), System Backend

3.3.1.1.3 Input specification

Refer to Test Procedure 4.2.

3.3.1.1.4 Output specification

Check that the Taxi Driver is notified about the taxi ride request.

3.3.1.1.5 Environmental needs

Demo Customer and Taxi Driver.

3.3.2 Send notifications feature

3.3.2.1 Send notifications feature, Test 1

3.3.2.1.1 Test case identifier

DF2T1

3.3.2.1.2 Test item(s)

Taxi Driver Mobile App (using GUI for input), System Backend

3.3.2.1.3 Input specification

Refer to Test Procedure 4.2.

3.3.2.1.4 Output specification

Check that the System Backend data is consistent with what the Taxi Driver's decision.

3.3.2.1.5 Environmental needs

Demo Customer and Taxi Driver, test DF1T1 succeeded.

3.3.3 Update location feature

3.3.3.1 Update location feature, Test 1

3.3.3.1.1 Test case identifier

DF3T1

3.3.3.1.2 Test item(s)

Taxi Driver Mobile App (using GUI for input), System Backend

3.3.3.1.3 Input specification

The taxi driver changes his location.

3.3.3.1.4 Output specification

Check that the System Backend data is consistent with the location change.

3.3.3.1.5 Environmental needs

Demo taxi driver.

3.4 Web Service features

3.4.1 Taxi booking feature

3.4.1.1 Taxi booking feature, Test 1

3.4.1.1.1 Test case identifier

WF1T1

3.4.1.1.2 Test item(s)

Web Service (using the page GUI for input), System Backend

3.4.1.1.3 Input specification

The user issues a booking request.

3.4.1.1.4 Output specification

The user is notified about the operation success and the System Backend starts processing the taxi request (i.e. test 3.1.3.5).

3.4.1.1.5 Environmental needs

Device supporting the Web Service (i.e. desktop PC).

3.4.2 Taxi request feature

3.4.2.1 Taxi request feature, Test 1

3.4.2.1.1 Test case identifier

WF2T1

3.4.2.1.2 Test item(s)

Web Service (using the page GUI for input), System Backend

3.4.2.1.3 Input specification

The user issues a taxi request.

3.4.2.1.4 Output specification

The mobile app user is notified about the operation success and the System Backend starts processing the taxi request (i.e. test 3.1.2.4).

3.4.2.1.5 Environmental needs

Device supporting the Web Service (i.e. desktop PC).

3.4.3 Display ride history feature

3.4.3.1 Display ride history feature, Test 1

3.4.3.1.1 Test case identifier

WF3T1

3.4.3.1.2 Test item(s)

Web Service (using the page GUI for input), System Backend

3.4.3.1.3 Input specification

The user asks to see the taxi ride history.

3.4.3.1.4 Output specification

The user is shown the taxi ride history, consistent to which is stored in the System Backend.

3.4.3.1.5 Environmental needs

Device supporting the Web Service (i.e. desktop PC).

3.4.4 Performance assessment

3.4.4.1 System Load-Testing and Stress-Testing

3.4.4.1.1 Test case identifier

P1T1

3.4.4.1.2 Test item(s)

System Backend.

3.4.4.1.3 Input specification

Apache JMeter is used to login as many different customers and to issue taxi requests.

3.4.4.1.4 Output specification

The System Backend should behave well with respect to the load and don't crash.

3.4.4.1.5 Environmental needs

Requirement 5.1. Stub for Taxi Driver Mobile App to simulate accept/refusal.

4. Test Procedures

4.1 System Backend

Execute all test cases in the same order they are presented in this document.

4.2 Mobile Apps

Execute test cases 3.2.1, 3.2.4 and 3.3.3.

Execute the remaining test cases in the following order: 3.2.2 (Taxi request from Customer Mobile App), 3.3.1 (Receive notification in Taxi Driver App), 3.3.2 (Send notification in Taxi Driver App), 3.2.3 (Receive notification in Customer Mobile App). To proceed with the following test case the previous one must has succeeded. This test procedure will simulate a real-world taxi request making this integration testing even more valuable.

4.3 Web Service

Execute all test cases in the same order they are presented.

5. Tools and test equipment required

5.1 Apache JMeter

We suggest the adoption of *Apache JMeter™* in order to perform load and stress testing on the system backend. Further information about the software are available here: http://jmeter.apache.org/.

6. Program stubs and test data required

6.1 Unit test driver and stubs

Stubs and driver used for the unit testing of all modules and components should be kept available for this integration testing as many test cases require them. See Environmental Needs for each test case.

6.2 Demo Customer

In many of the test cases, a Demo Customer is required in order to perform function calls: fake Customer data should be present in the database and associated to a smartphone available to the testers.

7. Appendix

7.1 Hours of work

This is the time spent in order to redact this document:

- Amos Paribocci: 8 hours;
- Lorenzo Pinosa: 7 hours.