POLITECNICO
MILANO 1863

Integration Test Plan Document - myTaxiService

Ciceri, Filippo
filippo.ciceri@mail.polimi.it

Cesaro, Federico
federico.cesaro@mail.polimi.it

Capecchi, Luca
luca.capecchi@mail.polimi.it

January 21, 2016

Contents

1 Introduction

1.1 Revision History

This is the first revision of this document.

1.2 Purpose and Scope

1.2.1 Purpose

The purpose of this document is to guide how to accomplish the integration test of the myTaxiService software. This document will show how our software should respond to given inputs, and all the software and strategy needed to test the software.

1.2.2 Scope

myTaxiService is a tool used in the city of X in order to simplify taxi usage for customers and taxi drivers. Our tests will aim to verify the correct behaviour of each step in the booking process of a ride.

1.3 List of Definitions and Abbreviations

Guest A guest to the service, namely someone who may be a driver or a registered user but has not been authenticated.

Registered User/User A user of the service who has gone through the registration process, whose information is stored within the service's database.

Driver A user of the service that provides their taxi for the service. They are registered in a manner different from that of other users.

Customer Any person trying to use a taxi through the application, both a registered users and guests.

Ride Generic term to talk about a taxi trip

Instant Ride A ride ordered directly through the application which instantly assigns a taxi to a customer.

Reservation/Booked Ride A ride booked in advance through the application.

Ride Locking/Locked Ride Term used to indicate reservations that have been "locked" from cancellation. A reservation is considered locked when it can no longer be cancelled or modified (2 hours before the start of the trip).

2 Integration Strategy

2.1 Entry Criteria

We assume that all the functions have been unit tested and that every component is working well. Our aim is to test interactions between each component so we need that every part of our system to not have internal problems.

2.2 Elements to be Integrated

The components we want to test are the ones described in our Design Document, more precisely in the Component View.
This is the list of the Components with their SubComponents whose interactions need to be tested:

Client Components

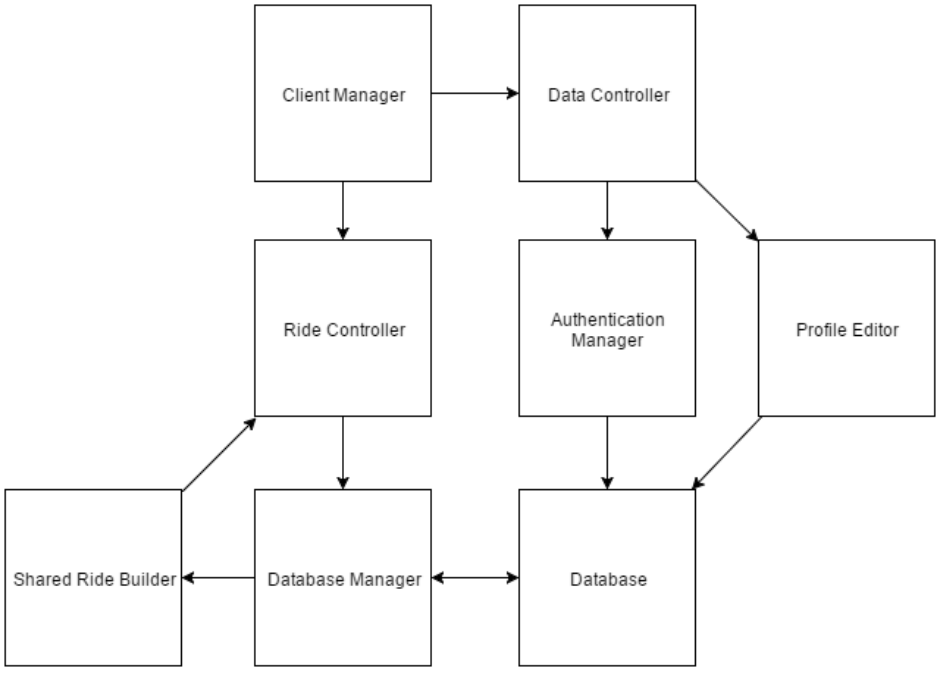
- Client Manager

Ride Manager Components

- Ride Controller
- Queue Manager
- Shared Ride Builder
- Database Manager

Data Manager Components

- Data Controller
- Authentication Manager
- Profile Editor



2.3 Integration Testing Strategy

We decided to use the bottom-up approach. We thought this was the best strategy in order to better test the connection between each level of our project. Before testing the top level we need to be sure that lower levels work in order to understand where hypothetical problems or issues may arise.

2.4 Sequence of Component/Function Integration

Test will have to be performed sequentially to verify the complete functionality of the system. As a result, we will follow the sequence illustrated in Section 2.2. First of all, we will test the User Management functions, like login and signup. Once those have been attained, we will look into the client's request forwarding and proper client-server authentication. We will test whether ride requests are properly stored and elaborated. Once we have verified that, we will see whether customer requests will be elaborated completely, and then verify if the interaction with taxi drivers is fully functional.

3 Individual Steps and Test Description

3.1 Test Case Specification

3.1.1 User Functions

Test ID and Name	**IT1: User SignUp**
Tested Components	Data Controller → Profile Editor → Database
Input	Generic and valid sign-up data
Expected Output	Profile created on database
Environmental Needs	Server running, request forwarding working, database working

Test ID and Name	**IT2: Profile data edit**
Tested Components	Client Manager → Data Controller → Profile Editor → Database
Input	New user data
Expected Output	New profile data overwriting old information
Environmental Needs	IT1

3.1.2 User Data Interactions

Test ID and Name	**IT3: Client authentication**
Tested Components	Client Manager → Data Controller → Authentication Manager
Input	Valid login information
Expected Output	Client receives login token and correctly instantiates a driver or user class locally
Environmental Needs	IT1

3.1.3 Instant Call Creation Process

Test ID and Name	**IT4: Instant call insertion**
Tested Components	Client Manager → Ride Controller
Input	An instant taxi request
Expected Output	Request inserted immediately into queue
Environmental Needs	Server running, request forwarding

3.1.4 Request Handling

Test ID and Name	**IT5: User ride request insertion**
Tested Components	Client Manager → Ride Controller
Input	Generic ride request
Expected Output	Ride creation server-side
Environmental Needs	IT3

3.1.5 Ride Management

Test ID and Name	**IT6: Ride editing**
Tested Components	Ride Controller → Database Manager → Database
Input	Valid reservation editing request
Expected Output	Ride information changed in DB
Environmental Needs	IT5

Test ID and Name	**IT7: Ride edit blocking**
Tested Components	Ride Controller → Database Manager → Database
Input	Modification request on a locked ride
Expected Output	Request refused
Environmental Needs	IT5

3.1.6 Shared Ride Creation

Test ID and Name	**IT8: Shared Ride creation trigger**
Tested Components	Database → Database Manager → Shared Ride Builder → Ride Controller
Input	Shared ride locking trigger
Expected Output	Shared ride created and inserted in DB
Environmental Needs	IT5

3.1.7 Driver Tests

Test ID and Name	**IT9: Driver status change**
Tested Components	Client Manager → Ride Controller
Input	Driver status change Client-side
Expected Output	Status change Server-side
Environmental Needs	IT3

Test ID and Name	**IT10: Driver accept**
Tested Components	Client Manager → Ride Controller → Database Manager → Database
Input	Driver "accept" response
Expected Output	Ride handled with driver data, driver assigned and saved in DB
Environmental Needs	IT9

3.2 Test Procedures

Procedure ID	**TP1**
Procedural Description	This test aims to test the login and user profile editing functions
Procedural Steps	IT1, IT3, IT2

Procedure ID	**TP2**
Procedural Description	This test makes a user login, create a ride and edit it
Procedural Steps	IT3, IT5, IT6

Procedure ID	**TP3**
Procedural Description	This test aims to test the ride locking system with relation to editing, by attempting to edit a shared ride after locking and failing. In order to make the test, the editing request must be made after some time has passed in order to let the ride get locked, as opposed to an instant modification
Procedural Steps	IT5, IT8, IT7

Procedure ID	**TP4**
Procedural Description	Instant ride and response testing. In this test, an instant request is made, while the queue is empty. Subsequently, a taxi driver changes his working status to active and then accepting the ride request
Procedural Steps	IT4, IT9, IT10

4 Tools and Test Equipment Required

The following tools should be used to run the integration tests described previously:

Mockito (`http://site.mockito.org/`) is a test framework that can be used to create mock objects in java. It's purpose in testing shall be to create objects whenever a test requires them.

Arquillan (`http://arquillian.org/`) is a test framework that will handle container management and initialization. It will handle the test concering the correct functionality and operation of our glassfish server.

jUnit (`http://junit.org/`) as mentioned previously in the document, unit testing will have to be performed. Although we won't be using it for actual integration testing, it still is a vital component in performing tests on our service's components.

5 Program Stubs and Test Data Required

In order to perform integration testing prior to the actual deployment, a handful of components have to be simulated, and several objects must be instantiated to perform our test. This includes, for example a functioning database, along with created users and drivers. In the following page, we will list what we will need to be created prior to the tests.

Database: we will need a functioning database, with a functioning configuration, mimicking that one of the version that will be deployed. The test data in this database shall include mock users, taxi drivers, and an access interface that enables the creation of rides as well as trigger based notifications.

A running server: this server will handle incoming requests from the mock client, as well as sending ride requests to the database and elaborating shared ride locks to create and complete orders

A mock client: this client will run in order to simulated user requests which are required for all tests involving the client and its sub-components.

A driver instance: which will be used to test driver functions.

6 Appendix

6.1 Software and Tools Used

- MiKTeX (http://www.miktex.org/) to format and create this document.
- Draw.io (http://www.draw.io/) to create advanced sequence diagrams

6.2 Work Hours

Time spent on the creation of this ITPD:

- Filippo Ciceri: 11 hours
- Federico Cesaro: 10 hours
- Luca Capecchi: 10 hours