Politecnico di Milano
A.A. 2015-2016
Software Engineering 2 Project

My Taxi Service

Integration Test Document
(ITD)

Giovanni Brena (858328), Andrea Canale (858638)

Contents

1. Introduction

1.1. Revision History

This document is currently at revision: 1.
No previous revisions.

1.2. Purpose and Scope

This document provides a strategy to achieve a complete and fully tested integration between components and software modules of MyTaxiService project. The main topic is to ensure the correct behavior of any interface connecting modules or subsystems through an integration process.

1.3. Reference Documents

The following documents has been used as references for MyTaxiService Project:

- MyTaxiService Requirement Analysis and Specification Document (RASD)
- MyTaxiService Design Document (DD)

The following documents has been used as external guidelines while writing this ITD:

- Assignement 4 – integration test plan
- Integration Test Plan Example

2. Integration Strategy

2.1. Entry Criteria

All components have to be unit tested before the integration test in order to provide atomic robustness to the system.
The following items must be delivered before integration testing begin:

- MyTaxiService Requirement Analysis and Specification Document (RASD)
- MyTaxiService Design Document (DD)
- Integration Testing Plan Document

The following documents has been used as external guidelines while writing this ITD:

- Assignement 4-integration test plan
- Integration Test Plan Example

2.2. Elements to be integrated

All subsystems and components will be submitted to integration test through a step-by-step integration process.

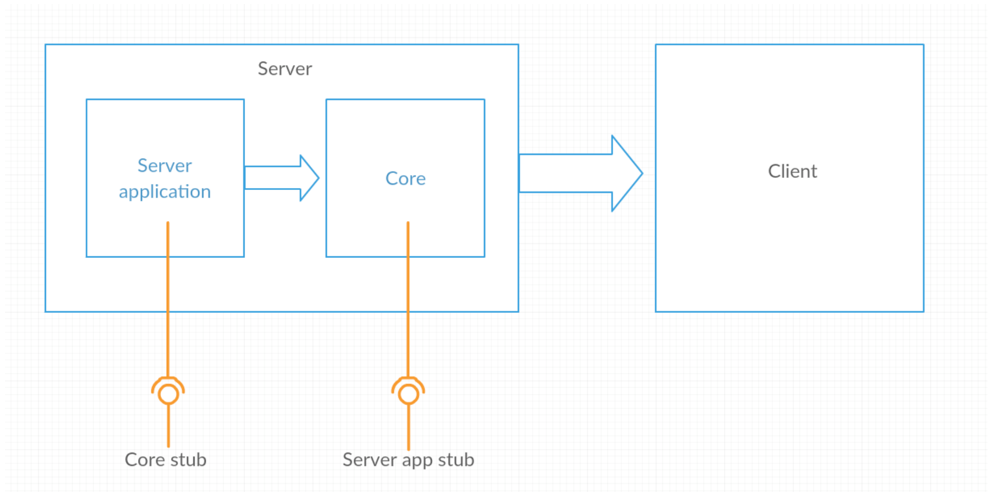
- Client Subsystem
- Server Application Subsystem
- Core Subsystem

2.3. Integration Testing Strategy

A bottom-up testing strategy will be followed, mixed with top-down approach for high-level integration. Subsystems modules will be integrated first, then process will join subsystems together.

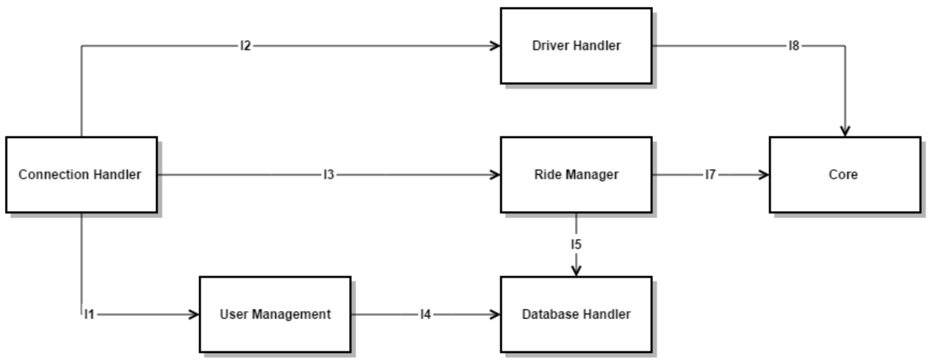
2.4. Components Integration

Subsystems integration will based on the following chart, providing integrity to the Server subsystems as base to Client integration.



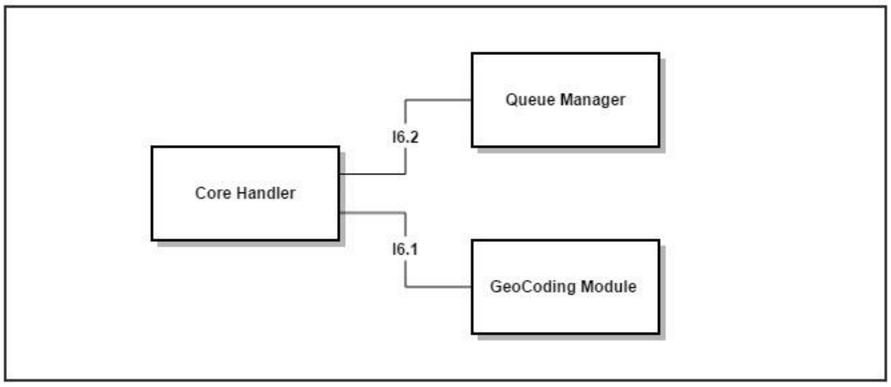
2.5. Software Integration

- Server Application Components



ID	Integration Test	Paragraphs
I1	ConnectionHanlder->UserManagement	3.1
I2	ConnectionHandler->Driver Handler	3.2
I3	ConnectionHandler->Ride Manager	3.3
I4	UserManagement->DatabaseHandler	3.4
I5	RideManager->DatabaseHandler	3.5
I6	(SubSystem)Core Integration Test	3.8
I7	Driver Handler->Core	3.6
I8	RideManager->Core	3.7

- Core's Components



ID	Integration Test	Paragraphs
I6.1	CoreHandler->GeoCodingModule	3.8.1
I6.2	UserManagement->Queue Manager	3.8.2

3. Steps and Test Description

3.1 Integration Test I1

Test Case Identifier	I1
Test Item(s)	ConnectionHanlder->UserManagement
Input Specifications	Methods calls from ConnectionHandler for Personal data management.
Output Specifications	The correct information of the user must be returned or modified.
Environmental Needs	Client Driver

3.2 Integration Test I2

Test Case Identifier	I2
Test Item(s)	ConnectionHandler->Driver Handler
Input Specifications	Methods call from ConnectionHandler to manage position and status of a Taxi Driver.
Output Specifications	The status or the position of the driver has to be correctly handled.
Environmental Needs	I1 succeeded

3.1 Integration Test I3

Test Case Identifier	I3
Test Item(s)	ConnectionHandler->Ride Manager
Input Specifications	Methods call from ConnectionHandler to request,start,interrupt or terminate a Ride and get or make a Reservation .
Output Specifications	Requests have to be correctly handled and the outcome returned.
Environmental Needs	I1 succeeded

3.4 Integration Test I4

Test Case Identifier	I4
Test Item(s)	UserManagement->DatabaseHandler
Input Specifications	Store,update personal data queries.
Output Specifications	The queries return correct results.
Environmental Needs	I1 succeeded

3.5 Integration Test I5

Test Case Identifier	I5
Test Item(s)	RideManager->DatabaseHandler
Input Specifications	Typical queries to Database from Ride Manager to manage Call and Reservation data.
Output Specifications	The queries return correct results.
Environmental Needs	I1 succeeded

3.6 Integration Test I7

Test Case Identifier	I7
Test Item(s)	RideManager->Core
Input Specifications	Methods call from RideManager for request a Driver given the coordinates of the passenger.
Output Specifications	The reference of an available Taxi Driver or "noTaxiAvailable" message.
Environmental Needs	I3 succeeded,I6(SubSystem) succeeded

3.7 Integration Test I8

Test Case Identifier	I8
Test Item(s)	Driver Handler->Core
Input Specifications	Methods call from Driver Handler for update position or status of the Driver.
Output Specifications	The status or the position of the driver has to be correctly handled.
Environmental Needs	I2 succeeded,I6(SubSystem) succeedded

3.8 Integration Test of Core SubSystem

3.8.1 Integration Test I6.1

Test Case Identifier	I6.1
Test Item(s)	CoreHandler->GeoCodingModule
Input Specifications	Method Code from CoreHandler to request a Zone of a given GPS position or request Path data from two GPS position.
Output Specifications	The correct Zone or Path data of input GPS positions.
Environmental Needs	N/A

3.8.2 Integration Test I6.2

Test Case Identifier	I6.2
Test Item(s)	Core Handler->Queue Manager
Input Specifications	Method from Core Handler, to manage the queue of Driver or request a Driver by GPS position.
Output Specifications	The queque has to be correctly managed and the first Driver available has to be returned.
Environmental Needs	N/A

4. Tools and Test Equipment

The software used to automate the integration testing are the following:

JUnit: is the most used framework for unit testing in Java. We plan to use it for unit tests of the single components, but it is also used to do integration testing together with Mockito and Arquillian.

Arquillian: is a test framework which can also manage the test of the containers and their integration with JavaBeans (dependency injection). We mainly use it for that purpose.

Mockito: is an open-source test framework useful to generate mock objects, stubs and drivers.We use it in several test cases to mock stubs and drivers for the components to test.

5. Program Stubs and Test Data

In order to perform the integration test without having developed the entire system we need stubs and drivers for that component that still doesn't exist.

Test Database: the testing environment must include a DBMS configured in the same way that it will be deployed.

Lightweight Client: to test the Business Tier(Server) without the client application,we need just a simple client that interact with the Server by HTTP requests.

Core Stub: Used to provide Core Component's outputs to perform Server App

Server App Stub : Used to provide Server's outputs for Core testing.