Integration Test Plan
"MyTaxiService" Application
A.Y. 2015 - 2016

Ennio Visconti (mat. 790382)
Simone Zocchi (mat. 852910)
Khanh Huy Paolo Tran (mat. 852496)

January 21, 2016

POLITECNICO
MILANO 1863

Contents

1 Introduction

1.1 Revision History

Version	Date	Authors	Summary
1.0	21/01/2016	Ennio Visconti, Simone Zocchi, Khanh Huy Paolo Tran	Document creation.

1.2 Purpose and Scope

This document describes the key points and the plan for the correct integration testing of myTaxiService application.
As an integration testing plan, this document is intended to provide developers the minimum guidelines about when the testing process should start, and what at every step should be done in order to ensure interfaces are meeting requirements.

1.3 Definitions and Abbreviations

- API: Application Program Interface
- CI: Continuous Integration
- DB: Database

1.4 Reference Document

- Software Engineering 2 Project, AA 2015-2016 Assignments 4 – Test plan
- Requirements Analysis & Specification Document - "myTaxiService" Application, AY 2015-2016; Ennio Visconti, Simone Zocchi, Khanh Huy Paolo Tran
- Design Document - "myTaxiService" Application, AY 2015-2016; Ennio Visconti, Simone Zocchi, Khanh Huy Paolo Tran

2 Integration Strategy

2.1 Entry Criteria

1. Unit tests executed for Path Calculator, Payment Manager and Zone Manager components that are important from an algorithmic point of view.
2. Sample data must be in the Database, containing examples of zones, taxis and shared rides, in order to make the algorithm be executed correctly.

2.2 Elements to be integrated

Our architectural description highlighted five main subsystem, divided between client and server. The most important one is the "BackEnd" subsystem, which contains all the business component of the application. This subsystem must be integrated with the "Data Abstraction Layer" and with the Client side, in particular with the "Common Front End Layer", and the last one must be, in turn, integrated with the "Mobile" and "Web" subsystems.
The subsystems of the client side will not be deeply analyzed because they deal with topics that are not relevant for this course. For this reason we will consider, mainly, the components of the "BackEnd" subsystem.
The components we will take into account are: Authentication manager, Communication manager, Payments manager, Taxi queue manager, Rides manager, Request manager, Path calculator and Zone manager.

2.3 Integration Testing Strategy

The component structure identified in the DD is not a tree structure that allows a top down or bottom up strategy for the integration. Instead we selected the most critical components of the application both from the algorithmic perspective and from the role they have inside the application execution process.

While algorithmic-critical components don't concern the integration issue much because they can be easily replaced by stubs, key-role ones are much more relevant.

The two riskiest modules identified also belong to two different execution threads. So while starting with these components, we will integrate others less crucial which also belong to the same thread.

Furthermore, to make sure that effective development flow doesn't create too much diverging items, different periodic integration cycles should be followed, as described in Figure 1, in order to deal with those problems as soon as they appear, but not in a time consuming way as it is if integrating the whole project every time something new is added.

We decided to use an approach based on the critical modules.

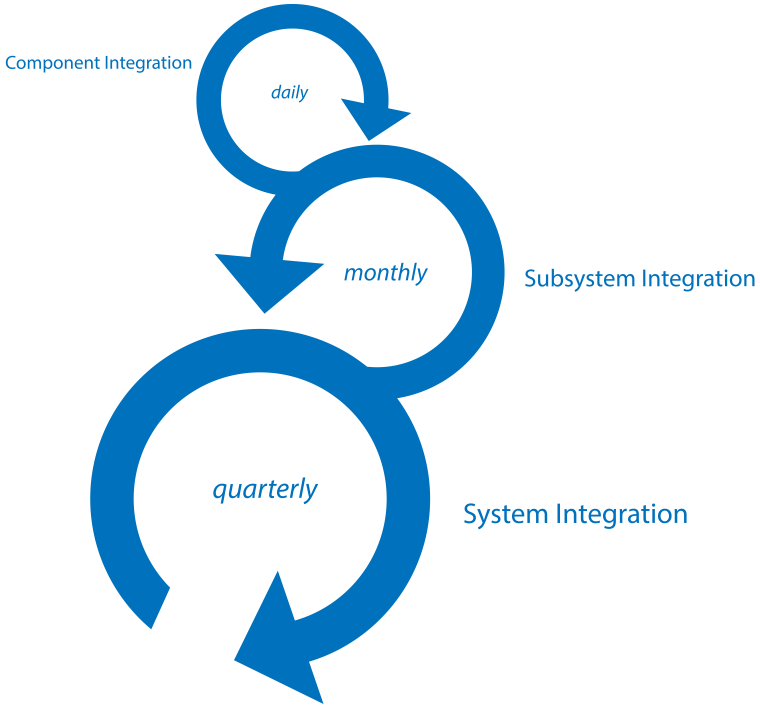

Figure 1: Integration Phases Cycle

2.4 Sequence of Component/Function Integration

2.4.1 Software Integration Sequence

The interactions between backend components we are going to test are the following in this order:

High Priority [HP]

1. *Request manager* $\longleftrightarrow$ *Taxi queue manager* (findTaxi() and changeTaxiState() methods)
2. *Ride manager* $\longleftrightarrow$ *Taxi queue manager* (changeTaxiState() method)
3. *Ride manager* $\longleftrightarrow$ *Payment manager* (calculateFare() and payPalPayment() methods)

Medium Priority [MP]

1. *Taxi queue manager* $\longleftrightarrow$ *Zone manager* (findQueue() method)
2. *Taxi queue manager* $\longleftrightarrow$ *Path calculator* (pathCalculation() method)

Low Priority [LP]

1. *Communication manager* $\longleftrightarrow$ *Request manager* (enqueueRequest() method; dispatchResponses() and dispatchToPassenger() on the other way)
2. *Communication manager* $\longleftrightarrow$ *Ride manager* (start(ride) and end(ride) methods)
3. *Communication manager* $\longleftrightarrow$ *Zone manager* (updateTaxiPosition() method)

Really Low Priority [RLP]

1. *Path calculator* $\longleftrightarrow$ *Google Maps* (usage of GoogleMaps API)
2. *Payment manager* $\longleftrightarrow$ *PayPal* (usage of PayPal API)

2.4.2 Subsystem Integration Sequence

Since the architectural design pattern chosen at design time is a Client-Server pattern, subsystem integration can start in parallel both from the client-side and from the server-side. The integration order for the subsystems is the one described in the picture below.

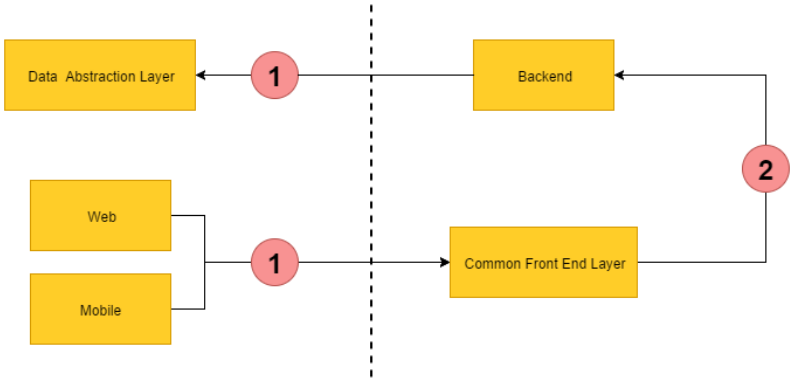

Figure 2: Subsystem Integration Sequence

3 Individual Steps and Test Description

3.1 Component Integration

Test Case Identifier	HP.1
Test Item(s)	Request manager → Taxi queue manager
Input Specification	A request from the queue is taken in charge.
Output Specification	The correct methods in *Taxi queue manager* are invoked. A taxi is correctly returned and its state is set as expected.
Environmental Needs	Taxi instances must be available to be exploited.

Test Case Identifier	HP.2
Test Item(s)	Ride manager → Taxi queue manager
Input Specification	A ride which needs to start.
Output Specification	The state of the taxi is correctly modified according to the previous state and the one indicated.
Environmental Needs	The passenger and the taxi driver must be willing to terminate the ride.

Test Case Identifier	HP.3
Test Item(s)	Ride manager → Payment manager
Input Specification	A fare request is sent.
Output Specification	The correct fare is calculated by *Payment manager* and is returned.
Environmental Needs	A ride must be finished and available.

Test Case Identifier	MP.1
Test Item(s)	Taxi queue manager → Zone manager
Input Specification	A location of a request is sent to the *Taxi queue manager*.
Output Specification	A list of the taxis of the proper zone around the given location is returned.
Environmental Needs	Zone and Taxi data should be available to be retrieved.

Test Case Identifier	MP.2
Test Item(s)	Taxi queue manager → Path calculator
Input Specification	A request for a shared ride.
Output Specification	Returns the path a taxi must follow to accomplish correctly a shared ride request.
Environmental Needs	Taxi states must not change during path calculator's execution. Entry Criterion 2 must be respected.

Test Case Identifier	LP.1.1
Test Item(s)	Communication Manager → Request Manager
Input Specification	There is a new request ready to be enqueued.
Output Specification	The new incoming request is properly enqueued according to its type: request or reservation.
Environmental Needs	The queue to handle the reservations must be implemented.

Test Case Identifier	LP.1.2
Test Item(s)	Communication Manager ← Request Manager
Input Specification	A positive response arrives from the Driver of a matching taxi.
Output Specification	The request for the taxi driver is created and dispatched to the proper client. A notification for the Passenger is created and dispatched to the proper client. In particular the proper message is created.
Environmental Needs	N/A

Test Case Identifier	LP.2
Test Item(s)	Communication manager → Ride manager
Input Specification	Messages for starting and endind a ride.
Output Specification	The beginning and ending time are set in the relative Ride entity.
Environmental Needs	N/A

Test Case Identifier	LP.3
Test Item(s)	Communication manager → Zone manager
Input Specification	Updated position of a taxi.
Output Specification	Related zone queue is updated properly.
Environmental Needs	Taxi entity must be present to be updated.

Test Case Identifier	RLP.1
Test Item(s)	Path calculator → Google Maps
Input Specification	A path (with start and end locations) is given.
Output Specification	Intermediate nodes of the given path are returned.
Environmental Needs	N/A

Test Case Identifier	RLP.2
Test Item(s)	Payment manager → PayPal
Input Specification	Payment data.
Output Specification	Completed transaction is returned.
Environmental Needs	A completed ride is needed.

3.2 Subsystem Integration

Test Case Identifier	S.1
Test Item(s)	Back End → Data Abstraction Layer
Input Specification	Commands of Insert, Delete or Update are sent to DB.
Output Specification	Tuples are correctly inserted, deleted or modified in the DB.

Test Case Identifier	S.2
Test Item(s)	Common Front End Layer ↔ Web & Mobile
Input Specification	Notification and messages to be elaborated and displayed.
Output Specification	Incoming messages are interpreted in the proper way.

Test Case Identifier	S.3
Test Item(s)	Back End ↔ Common Front End Layer
Input Specification	Messages and notifications to be dispatched.
Output Specification	The communications are correctly dispathed to the right client.

4 Tools and Test Equipment Required

4.1 Travis CI - https://travis-ci.org/

Travis CI, as Continuous Integration Platforms in general, is becoming really popular in the latest years. It is useful in agile development environments, where code changes are integrated to the baseline daily.
It therefore will be used to make sure that the development process is consistent as the time goes by, dramatically reducing integration efforts at the end of the developing phase.

4.2 Mockito - http://mockito.org/

Mockito is a Mocking framework which is useful both for unit testing and for integration testing. It not only makes it possible to produce atomic unit tests, but it is also powerful in more complex contexts.
In particular it will be used to concretely create stubs and drivers which will be defined for specific tests later.

5 Program Stubs and Test Data Required

The test data indicated in this section are the ones that trigger the beginning of the procedure under test. Other data required during the execution has been inserted in the "Environmental Needs" of the single steps in the section 3.

HP.1 **Driver**: N/A
Stub: Zone manager and Path Calculator

HP.2 **Driver**: Communication Manager
Stub: N/A

HP.3 **Driver**: Communication Manager
Stub: N/A

MP.1 **Driver**: Request Manager
Stub: N/A

MP.2 **Driver**: Request Manager
Stub: Zone Manager

LP.1.1 **Driver**: Passenger Client
Stub: ReservationsWorker

LP.1.2 **Driver**: N/A
Stub: N/A

LP.2 **Driver**: N/A
Stub: Payment Manager and Taxi Queue Manager
Test data: Messages of a Driver Client for starting and Ending a Ride.

LP.3 **Driver**: N/A
Stub: N/A
Test data: Message from a taxi driver with his the current position.

RLP.1 **Driver**: Taxi queue manager
Stub: N/A

RLP.2 **Driver**: Ride Manager
Stub: N/A