Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

Integration Test Plan myTaxiService

Student: Federico Gatti [Matricola: 852377]
Student: Luca Fochetta [Matricola: 792935]

AA 2015–2016

Contents

Chapter 1

Introduction

1.1 Purpose

This document describes the plans for testing the integration of the *myTaxiService* components. The purpose of this document is to test the interfaces between the components as described in Design Document, also verify functional, performance, and reliability requirements of the system.

Software Integration Testing is performed according to the Software Development Life Cycle (SDLC) after module and functional tests. The cross-dependencies for software integration testing are: schedule for integration testing, strategy and selection of the tools used for integration, define the cyclomatical complexity of the software and software architecture, reusability of modules and life-cycle / versioning management.

1.2 Scope

The aim of the project is to create a software, called *myTaxiService*, which can manage the queue of the taxi requests in a city. The system is composed by 4 main parts:

- 2 front-end applications used by the passengers called PMA and PWA
- 1 front-end application used by the Taxi Drivers called TMA
- 1 back-end system called QTM

The system will be able to suggest the best distribution of the taxi in the city zone to maximize passengers' satisfaction and the quality of the service. A passenger who sends a request, using a web application or mobile application, can see from the application the waiting time and the code of the taxi that accepted the request. The passenger position can be determinate from GPS or if GPS information are incorrect, or aren't available, the passenger can insert manually the information of the location. An authenticated passenger can reserve a taxi by specifying the origin and the destination of the ride. In this case the passenger must place the reservation at least two hours before the ride. The reservation

request will be sent like a normal request ten minutes before the specified time. An authenticated user can always delete his reservation or modify it. A Taxi driver can accept or reject a request.

1.3 Definitions, Acronyms, Abbreviations

- RASD: Requirement Analysis and Specication Document
- DD: Design Document
- PMA: Passenger Mobile Application
- PWA: Passenger Web Application
- PA: Passenger Application refers indiscriminately at PMA and PWA
- TMA: Taxi Mobile Application
- QTM: Queue and Taxi Manager
- DB: Data Base
- DBMS: Data Base Management System
- UI: User Interface

1.4 List of Reference Documents

1. *myTaxiService* RASD
2. *myTaxiService* DD
3. Integration Test Plan Example
4. Wikipedia

1.5 Document Structure

The document is composed by five sections.

1. The first section, this one, defines the goal of the document and a general idea of *myTaxiService* functionalities
2. The second section describes the precondition for the integration, the test integration strategy used and justify the choices, also provides the steps to perform all the integration tests.

3. The third section defines the test sets and test cases starting from the steps described in section two

4. The fourth section lists the software tools used to perform the test integration, also described the case when automated test is impossible

5. The fifth section described the program drivers, stubs and special test data required for each integration step.

Chapter 2

Integration Strategy

2.1 Entry Criteria

In order to perform Integration Testing is required that all modules of the system are tested before using unit testing. All the single modules work perfectly.

Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

2.2 Elements to be Integrated

There are two kinds of elements to be integrated that are the modules that composed a single sub-system and the sub-system. First of all we must integrate all the modules that composed a sub-system, after that, if it's all correct, we must check the integration between two different sub-system.

All the sub-system and they interaction are described and in the DD, but we want illustrate a more detailed schema useful for the integration test.

Differently from DD is added the UI and is improve the representation of the links .

UI is the module that allow the client to interact with the system.

2.2.1 Application Logic

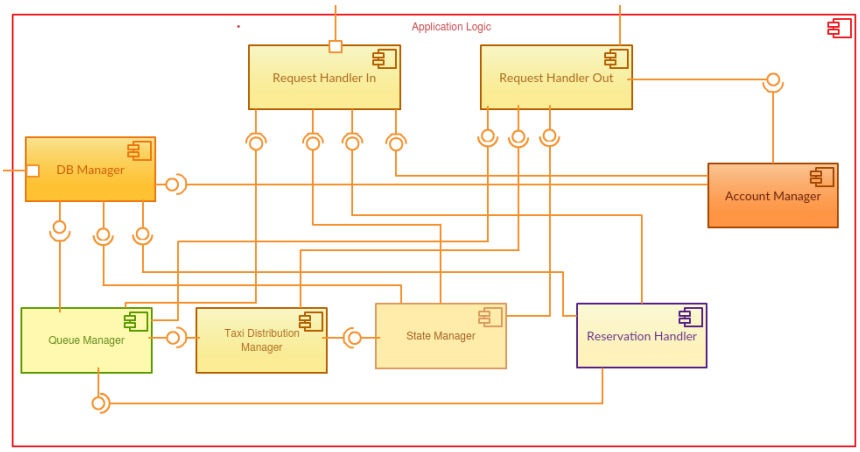


Figure 2.1: Application Logic sub-system

2.2.2 TMA

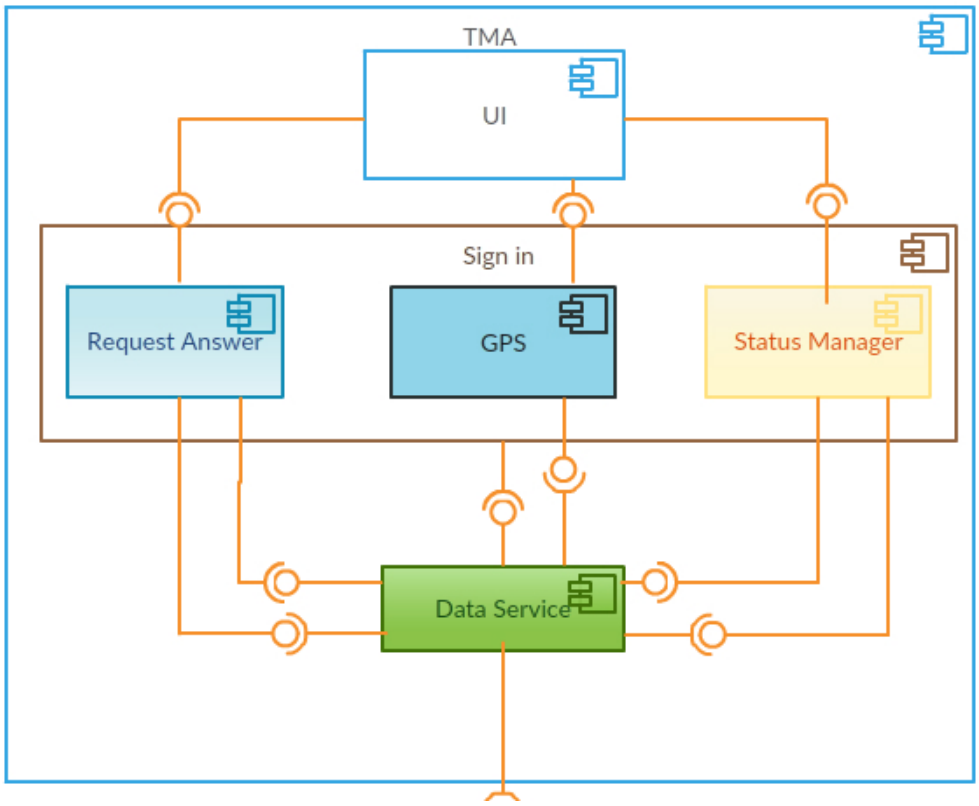


Figure 2.2: TMA sub-system

2.2.3 PMA

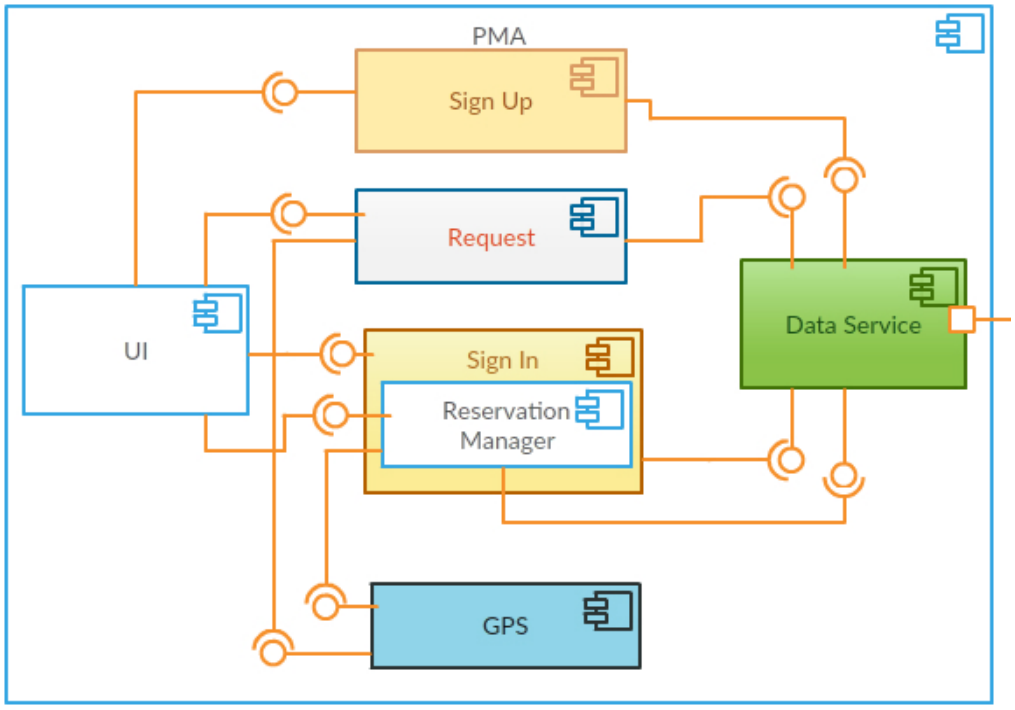


Figure 2.3: PMA sub-system

2.3 Integration Testing Strategy

We choose to use a Bottom Up approach for testing the integration for each modules in a single subsystem and a Sandwich approach for testing the integration between subsystem.

2.3.1 Bottom Up

Bottom Up Testing is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher level components. The process is repeated until the component at the top of the hierarchy is tested.

All the bottom or low-level modules, procedures or functions are integrated and then tested. After the integration testing of lower level integrated modules, the next level of modules will be formed and can be used for integration testing. This approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage.

The main motivations for using the Bottom-up integration are:

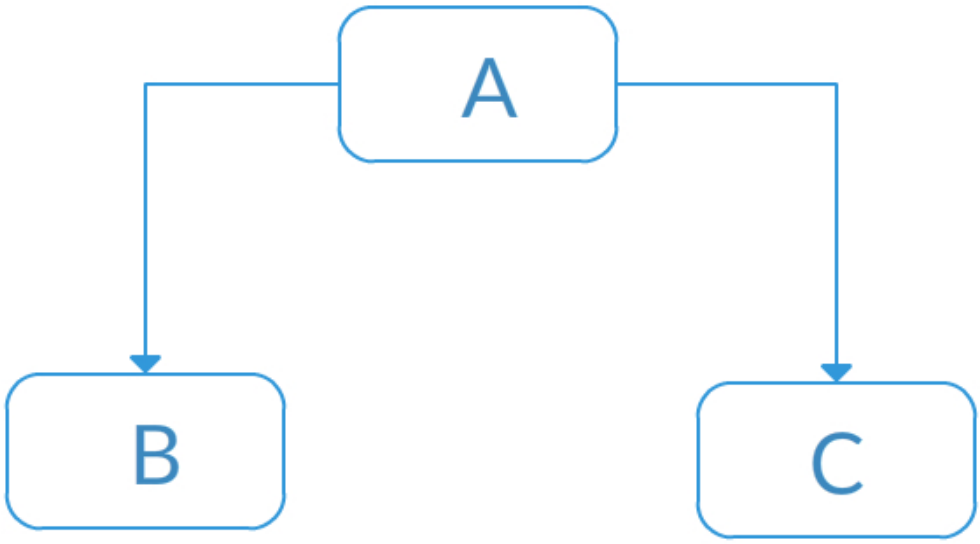
- Drivers are often more easier to implement than Stubs
- Advantageous if major flaws occur toward the bottom of the program

- Test conditions are easier to create
- Observation of test results is easier
- Bottom-up is more adequate for objected oriented design than Top-down

Def. A **driver** is a main program that accepts test data and passes this test to the component to be tested and prints relevant results. Drivers must be kept for future integration tests.

Bottom-up Steps:

1. Test B, C individually (using drivers)
2. Test A such that it calls B. If an error occurs we know that the problem is in A or in the interface between A and B
3. Test A such that it calls C. If an error occurs we know that the problem is in A or in the interface between A and C



2.3.2 Sandwich

Sandwich Testing is an approach to combine top down testing with bottom up testing.

The main advantage of the Bottom-Up approach is that bugs are more easily found. With Top-Down, it is easier to find a missing branch link.

This methodology of testing can be useful because the client application software is independent from the server application so in this way we can test the two different subsystem integration concurrently also Sandwich approach is suitable for large projects having several subprojects..

At the end the test converges to a middle layer called target layer.

To perform Sandwich integration we must know how Top-down integration works.

Top-down Steps:

1. Test A individually (use stubs for B and C)
2. Test A such that it calls B (stub for C). If an error occurs we know that the problem is in B or in the interface between A and B
3. Test A such that it calls C (stub for B). If an error occurs we know that the problem is in C or in the interface between A and C

2.4 Sequence of Component/Function Integration

Considering that our system in composed by several subsystem and each subsystem is composed by several modules we must to start with testing the integration of each modules of a single subsystem, to certify the correct implementation, after that we can test the Integration between the different sub-system that compose the entire software.

2.4.1 Software Integration Sequence

2.4.1.1 Application Logic Integration Steps

1. Creation of the Driver that replace the Request Handler In
2. Test Queue Manager, State Manager, Reservation Handler and Account Manager individually using the Request Handler In Driver
3. Remove Request Handler In Driver and test Request Handler In such that it calls Queue Manager, State Manager, Reservation Handler and Account Manager
   (a) **ID**: AP-I1, **Test**: Request Handler In → Queue Manager
   (b) **ID**: AP-I2, **Test**: Request Handler In → State Manager
   (c) **ID**: AP-I3, **Test**: Request Handler In → Reservation Handler
   (d) **ID**: AP-I4, **Test**: Request Handler In → Account Manager
4. Creation of the Driver that replace the Queue Manager, State Manager, Reservation Handler and Account Manager
5. Test DataBase Manager using Queue Manager, State Manager, Reservation Handler and Account Manager Driver
6. Remove Queue Manager, State Manager, Reservation Handler and Account Manager Driver and test Queue Manager, State Manager, Reservation Handler and Account Manager such that them call DataBase Manager
   (a) **ID**: AP-I5, **Test**: Queue Manager → Database Manager
   (b) **ID**: AP-I6, **Test**: State Manager → Database Manager
   (c) **ID**: AP-I7, **Test**: Reservation Handler → Database Manager

(d) **ID**: AP-I8, **Test**: Account Manager → Database Manager

7. Creation of Taxi Distribution Manager
8. Test Queue Manager and State Manager individually using Taxi Distribution Manager Driver
9. Remove Taxi Distribution Manager Driver and test Taxi Distribution Manager such that it calls Queue Manager and State Manager
   (a) **ID**: AP-I9, **Test**: Taxi Distribution Manager → Queue Manager
   (b) **ID**: AP-I10, **Test**: Taxi Distribution Manager → State Manager
10. Creation of the Drivers that replace Queue Manager, State Manager, Taxi Distribution Manager and Account Manager Driver
11. Test Request Handler Out using Queue Manager, State Manager, Taxi Distribution Manager and Account Manager Driver
12. Remove Queue Manager, State Manager, Taxi Distribution Manager and Account Manager Driver and test Queue Manager, State Manager, Taxi Distribution Manager and Account Manager such that them call Request Handler Out
    (a) **ID**: AP-I11, **Test**: Queue Manager → Request Handler Out
    (b) **ID**: AP-I12, **Test**: State Manager → Request Handler Out
    (c) **ID**: AP-I13, **Test**: Taxi Distribution Manager → Request Handler Out
    (d) **ID**: AP-I14, **Test**: Account Manager → Request Handler Out

2.4.1.2 TMA Integration Steps

1. Creation of the Driver that replace the Data Service
2. Test Status Manager, Request Answer and GPS individually using Data Service Driver
3. Remove the Data Service Driver and test Data Service such that it calls Request Answer, Status Manager and GPS
   (a) **ID**: TMA-I1, **Test**: Data Service → Request Answer
   (b) **ID**: TMA-I2, **Test**: Data Service → Status Manager
   (c) **ID**: TMA-I3, **Test**: Data Service → GPS
4. Creation of the Drivers that replace Request Answer, Status Manager and Sign In
5. Test Data Service using Request Answer, Status Manager and Sign in individually using Driver

6. Remove Request Answer, Status Manager ans Sign In Driver and test Request Answer, Status Manager and Sign In such that them call Data Service

   (a) **ID**: TMA-I4, **Test**: Request Answer → Data Service

   (b) **ID**: TMA-I5, **Test**: Status Manager → Data Service

   (c) **ID**: TMA-I6, **Test**: Sign In →Data Service

7. Creation of the Driver that replace UI

8. Test Status Request Answer, GPS and Status Manger individually using UI Driver

9. Remove UI Driver and test UI such that it calls Request Answer, Sign In and Status Manger

   (a) **ID**: TMA-I7, **Test**: UI → Request Answer

   (b) **ID**: TMA-I8, **Test**: UI → Sign In

   (c) **ID**: TMA-I9, **Test**: UI → Status Manager

2.4.1.3 PMA

1. Creation of the Driver that replace the Sign Up, Sign In, Request Reservation Manger

2. Test Data Service using Sign Up, Sign In, Request and Reservation Manager Driver

3. Remove Sign Up Driver Sign In, Request and Reservation Manager Driver and test Sign Up, Sign In, Request and Reservation Manager such that them call Data Service

   (a) **ID**: PMA-I1, **Test**: Sign Up → Data Service

   (b) **ID**: PMA-I2, **Test**: Sign In → Data Service

   (c) **ID**: PMA-I3, **Test**: Request → Data Service

   (d) **ID**: PMA-I4, **Test**: Reservation Manager → Data Service

4. Creation of the Driver that replace Request and Reservation Manager

5. Test GPS using Request Driver and Reservation Manager Driver

6. Remove Request and Reservation Manager Driver and test Request and Reservation Manager such that them call GPS

   (a) **ID**: PMA-I5, **Test**: Request → GPS

   (b) **ID**: PMA-I6, **Test**: Reservation Manager→ GPS

7. Creation of the Driver that replace UI

8. Test Sign Up, Request, Sign In and Reservation Manager using UI Driver

9. Remove UI Driver and test UI such that it calls Sign Up, Request, Sign In and Reservation Manager

   (a) **ID**: PMA-I7, **Test**: UI → Sign Up

   (b) **ID**: PMA-I8, **Test**: UI → Request

   (c) **ID**: PMA-I9, **Test**: UI → Sign In

   (d) **ID**: PMA-I10, **Test**: UI → Reservation Manager

2.4.1.4 PWA

The integration test isn't necessary for this subsystem

2.4.2 Subsystem Integration Sequence

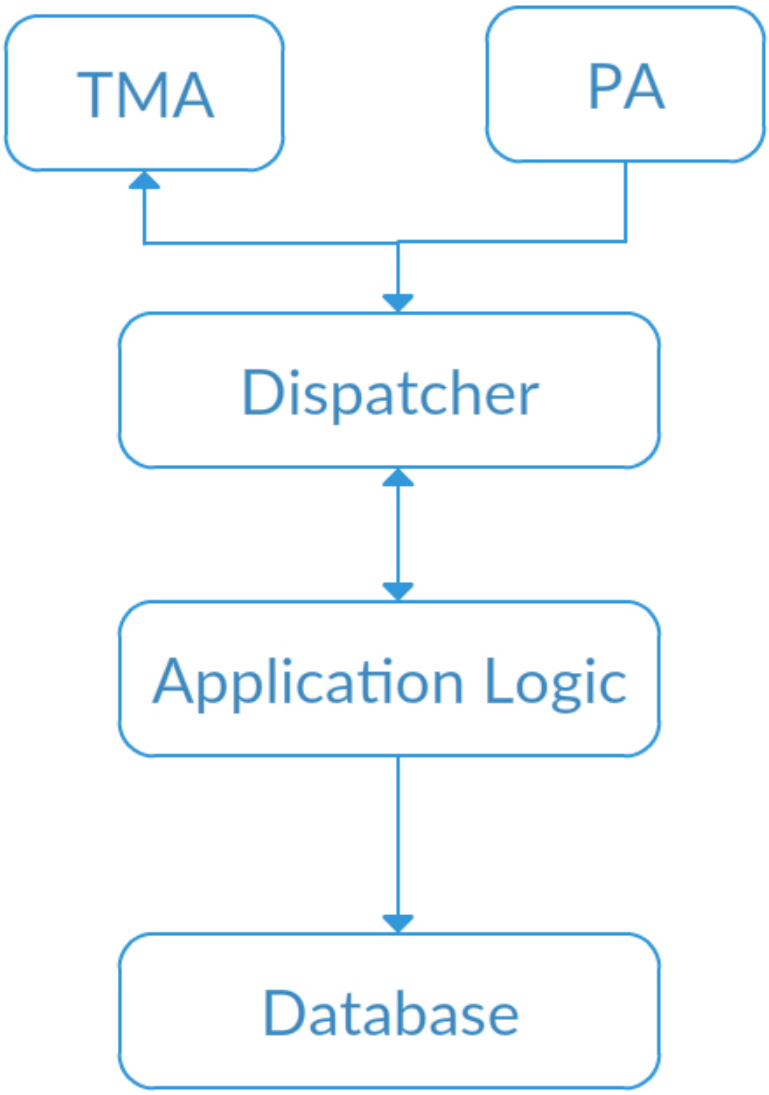


Figure 2.4: Sub-system

2.4.2.1 Top layer

According to Sandwich Integration approach for the top layer we must use a Top-down integration

Steps:

1. Creation of the Stub for the Dispatcher
2. Test PMA individually using Dispatcher Stub
3. Remove the Dispatcher Stub and test PMA such that it calls the Dispatcher
   (a) **ID**: TD-I1, **Test**: PMA → Dispatcher
4. Creation of the Stub for the Dispatcher
5. Test TMA individually using Dispatcher Stub
6. Remove the Dispatcher Stub and test TMA such that it calls the Dispatcher
   (a) **ID**: TD-I2, **Test**: TMA → Dispatcher
7. Creation of the Stub for the TMA
8. Test Dispatcher individually using TMA Stub
9. Remove the TMA Stub and test Dispatcher such that it calls the TMA
   (a) **ID**: TD-I3, **Test**: Dispatcher → TMA

2.4.2.2 Lower layer

Steps:

1. Creation of the Driver for the Application Logic
2. Test Database using Application Logic Driver
3. Remove Application Logic Driver and test Application Logic such that it calls Database
   (a) **ID**: BU-I1, **Test**: Application Logic → Database
4. Creation of the Driver for the Application Logic
5. Test Dispatcher individually using Application Logic Driver
6. Remove Application Logic Driver and test Application Logic such that it calls the Dispatcher
   (a) **ID**: BU-I2, **Test**: Application Logic → Dispatcher

7. Creation of the Driver for the Dispatcher
8. Test Application logic using Dispatcher Driver
9. Remove Dispatcher Driver and test Dispatcher such that it calls Application Logic
    (a) **ID**: BU-I3, **Test**: Dispatcher → Application Logic

Chapter 3

Individual Steps and Test Description

3.1 Application Logic

3.1.1 Integration test case AP-I1

Test Case Identifier	AP-I1
Test Item	Request Handler In → Queue Manager
Input Specification	Create typical Request input
Output Specification	Check if the correct functions are called in the Queue Manager
Tested Functionalities	1. Check that when searchForATaxi is called on requestHandlerIn extractTaxi is called on Queue Manager 2. Check that when answerRequest is called on requestHandlerIn removeTaxi is called on Queue Manager
Tested non Functionalities	-
Environmental Needs	Queue Manager Driver

3.1.2 Integration test case AP-I2

Test Case Identifier	AP-I2
Test Item	Request Handler In → State Manager
Input Specification	Create typical Request input
Output Specification	Check if the correct functions are called in the State Manager
Tested Functionalities	1. Check that when answerRequest or modifyRequest is called on requestHandlerIn changeStatus is called on Queue Manager
Tested non Functionalities	-
Environmental Needs	State Manager Driver

3.1.3 Integration test case AP-I3

Test Case Identifier	AP-I3
Test Item	Request Handler In → Reservation Handler
Input Specification	Create typical Request input
Output Specification	Check if the correct functions are called in the Reservation Handler
Tested Functionalities	1. Check that when sendReservationData is called on requestHandlerIn createNewReservation is called on Reservation Handler 2. Check that when sendModifyReservation is called on requestHandlerIn modifyReservation is called on Reservation Handler
Tested non Functionalities	-
Environmental Needs	reservation Handler Driver

3.1.4 Integration test case AP-I4

Test Case Identifier	AP-I4
Test Item	Request Handler In → Account Manager
Input Specification	Create typical Request input
Output Specification	Check if the correct functions are called in the Account Manager
Tested Functionalities	1. Check that when signUpRequest is called on requestHandlerIn signUpDataCheck is called on Account Manager
Tested non Functionalities	-
Environmental Needs	Account Manager Driver

3.1.5 Integration test case AP-I5

Test Case Identifier	AP-I5
Test Item	Queue Manager → Database Manager
Input Specification	Create insertion, removal and return of a Taxi in the queue
Output Specification	Check if the correct functions are called in the Database Manager
Tested Functionalities	1. Check that when selectTaxi, insertTaxi and removeTaxi is called on QueueManager query method is called on Database Manager
Tested non Functionalities	-
Environmental Needs	AP-I1 succeeded

3.1.6 Integration test case AP-I6

Test Case Identifier	AP-I6
Test Item	State Manager → Database Manager
Input Specification	Create request of change and return of status
Output Specification	Check if the correct functions are called in the Database Manager
Tested Functionalities	1. Check that when selectStatusTaxi or changeStautusQuery is called on State Manager query method is called on Database Manager
Tested non Functionalities	-
Environmental Needs	AP-I2 succeeded

3.1.7 Integration test case AP-I7

Test Case Identifier	AP-I7
Test Item	Reservation Handler → Database Manager
Input Specification	Create Request of insertion, removal and return of a Reservation
Output Specification	Check if the correct functions are called in the Database Manager
Tested Functionalities	1. Check that when selectTaxiReservation, modifyTaxiReservation or deleteTaxiReservation is called on Reservation Handler query method is called on Database Manager
Tested non Functionalities	-
Environmental Needs	AP-I3 succeeded

3.1.8 Integration test case AP-I8

Test Case Identifier	AP-I8
Test Item	Account Manager → Database Manager
Input Specification	Create Request of insertion, removal and return of an Account
Output Specification	Check if the correct functions are called in the Database Manager
Tested Functionalities	1. Check that when selectAccount, modifyAccount or insertAccount is called on Account Manager query method is called on Database Manager
Tested non Functionalities	-
Environmental Needs	AP-I4 succeeded

3.1.9 Integration test case AP-I9

Test Case Identifier	AP-I9
Test Item	Taxi Distribution Manager → Queue Manager
Input Specification	Create typical request taxi redistribution
Output Specification	Check if the correct functions are called in the Queue Manger
Tested Functionalities	1. Check that when extractTaxiDistribution is called on Taxi Distribution Manager selectTaxi is called on Queue Manager
Tested non Functionalities	-
Environmental Needs	Taxi Distribution Driver

3.1.10 Integration test case AP-I10

Test Case Identifier	AP-I10
Test Item	Taxi Distribution Manager → State Manager
Input Specification	Create typical request taxi redistribution
Output Specification	Check if the correct functions are called in the State Manager
Tested Functionalities	1. Check that when extractTaxiDistribution is called on Taxi Distribution Manager selectTaxiState is called on State Manager
Tested non Functionalities	-
Environmental Needs	Taxi Distribution Driver

3.1.11 Integration test case AP-I11

Test Case Identifier	AP-I11
Test Item	Queue Manager → Request Handler Out
Input Specification	Create typically send of a selected Taxi
Output Specification	Check if the correct functions are called in the Request Handler Out
Tested Functionalities	1. Check that when extractTaxiDistribution is called on Taxi Distribution Manager selectTaxi is called on Queue Manager
Tested non Functionalities	-
Environmental Needs	AP-I5 succeeded

3.1.12 Integration test case AP-I12

Test Case Identifier	AP-I12
Test Item	State Manager → Request Handler Out
Input Specification	Create typically send a modification of a Taxi Status
Output Specification	Check if the correct functions are called in the Request Handler Out
Tested Functionalities	1. Check that when newStatus is called on State Maner sendNewStatus is called on Request Handler Out
Tested non Functionalities	-
Environmental Needs	AP-I6 succeeded

3.1.13 Integration test case AP-I13

Test Case Identifier	AP-I13
Test Item	Taxi Distribution Manager → Request Handler Out
Input Specification	Create typically send of distribution Taxis
Output Specification	Check if the correct functions are called in the Request Handler Out
Tested Functionalities	1. Check that when newDistribution is called on State Maner sendNewAreaOfCompetence is called on Request Handler Out
Tested non Functionalities	-
Environmental Needs	AP-I9 & AP-I10

3.1.14 Integration test case AP-I14

Test Case Identifier	AP-I14
Test Item	Account Manager → Request Handler Out
Input Specification	Create typical Account check
Output Specification	Check if the correct functions are called in the Request Handler Out
Tested Functionalities	1. Check that when sendAccountCheck is-called on State Maner sendCheckedAccount is called on Request Handler Out
Tested non Functionalities	-
Environmental Needs	AP-I8 succeeded

3.2 TMA

3.2.1 Integration test case TMA-I1

Test Case Identifier	TMA-I1
Test Item	Data Service → Request Answer
Input Specification	Create typical new ride request
Output Specification	Check if the correct functions are called in the Request Answer
Tested Functionalities	1. Check that when sendAnswerRequest is called on Data Service showRequest is called on Request Answer
Tested non Functionalities	-
Environmental Needs	Data Service Driver

3.2.2 Integration test case TMA-I2

Test Case Identifier	TMA-I2
Test Item	Data Service → Status Manager
Input Specification	Create typical current Status modification
Output Specification	Check if the correct functions are called in the Status Manager
Tested Functionalities	1. Check that when sendChangeStatusData is called on Data Service showStatus is called on Status Manager
Tested non Functionalities	-
Environmental Needs	Data Service Driver

3.2.3 Integration test case TMA-I3

Test Case Identifier	TMA-I3
Test Item	Data Service → GPS
Input Specification	Create typical position request
Output Specification	Check if the correct functions are called in the GPS
Tested Functionalities	1. Check that when is called getTaxiPosition on DataService getPosition is called on GPS
Tested non Functionalities	1. getPosition must return a value in 5s considering Internet connection
Environmental Needs	Data Service Driver

3.2.4 Integration test case TMA-I4

Test Case Identifier	TMA-I4
Test Item	Request Answer → Data Service
Input Specification	Create typical answer request
Output Specification	Check if the correct functions are called in the Data Service
Tested Functionalities	1. Check that when is called rejectRequest on Request Answer sendRejectRequestData is called on DataService 2. Check that when is called acceptRequest on Request Answer sendAcceptedRequest-Data is called on DataService
Tested non Functionalities	1. Data Service must check all input data
Environmental Needs	TMA-I1 succeeded

3.2.5 Integration test case TMA-I5

Test Case Identifier	TMA-I5
Test Item	Status Manager → Data Service
Input Specification	Create typical Status request change
Output Specification	Check if the correct functions are called in the Data Service
Tested Functionalities	1. Check that when is called ackChangeSta-tus on Status Manager sendAckChangeS-tatus is called on DataService 2. Check that when is called newStatus on Status Manager sendRequestNewStatus is called on DataService
Tested non Functionalities	Data Service check all input data
Environmental Needs	TMA-I2 succeeded

3.2.6 Integration test case TMA-I6

Test Case Identifier	TMA-I6
Test Item	Sign In →Data Service
Input Specification	Create typical Sign In request
Output Specification	Check if the correct functions are called in the Data Service
Tested Functionalities	1. Check that when is called sendRequestLogIn on Sign In sendLogInRequestData is called on DataService
Tested non Functionalities	1. Data Service check all input data
Environmental Needs	Sign In Driver

3.2.7 Integration test case TMA-I7

Test Case Identifier	TMA-I7
Test Item	UI → Request Answer
Input Specification	Create typical IO Request Answer
Output Specification	Check if the correct functions are called in the Request Answer
Tested Functionalities	1. Check that when clickedAcceptRequest is called in UI acceptRequest is called in Request Answer 2. Check that when clickedRejectRequest is called in UI acceptRequest is called in Request Answer
Tested non Functionalities	1. Check that the response time not using Internet is smaller than 50ms in 99.9% of cases
Environmental Needs	TMA-I4 succeeded

3.2.8 Integration test case TMA-I8

Test Case Identifier	TMA-I8
Test Item	UI → Sign In
Input Specification	Create typical IO Sign In
Output Specification	Check if the correct functions are called in the Sign In
Tested Functionalities	1. Check that when clickedSignIn is called in UI sendRequestLogIn is called in Sign In
Tested non Functionalities	1. Check that the response time not using Internet is smaller than 50ms in 99.9% of cases
Environmental Needs	TMA-I6 succeeded

3.2.9 Integration test case TMA-I9

Test Case Identifier	TMA-I9
Test Item	UI → Status Manager
Input Specification	Create typical IO change of Status
Output Specification	Check if the correct functions are called in the Status Manager
Tested Functionalities	1. Check that when clickedChenageStatus is called in UI sendRequestLogIn is called in Status Manager
Tested non Functionalities	Check that the response time not using Internet is smaller than 50ms in 99.9% of cases
Environmental Needs	TMA-I5 succeeded

3.3 PMA

3.3.1 Integration test case PMA-I1

Test Case Identifier	PMA-I1
Test Item	Sign Up → Data Service
Input Specification	Create typical Sign Up request
Output Specification	Check if the correct functions are called in the Data Service
Tested Functionalities	1. Check that when is called sendRequestSignUp on Request Answer sendSignUpRequestData is called on DataService
Tested non Functionalities	Data Service check all input data
Environmental Needs	Sign Up Driver

3.3.2 Integration test case PMA-I2

Test Case Identifier	PMA-I2
Test Item	Sign In → Data Service
Input Specification	Create typical Sign In request
Output Specification	Check if the correct functions are called in the Data Service
Tested Functionalities	1. Check that when is called sendRequestLogIn on Sign In sendLogInRequestData is called on DataService
Tested non Functionalities	Data Service check all input data
Environmental Needs	Sign In Driver

3.3.3 Integration test case PMA-I3

Test Case Identifier	PMA-I3
Test Item	Request → Data Service
Input Specification	Create typical taxi request
Output Specification	Check if the correct functions are called in the Data Service
Tested Functionalities	1. Check that when is called submitRequest on Request sendRequestData is called on DataService
Tested non Functionalities	Data Service check all input data
Environmental Needs	Request Driver

3.3.4 Integration test case PMA-I4

Test Case Identifier	PMA-I4
Test Item	Reservation Manager→ Data Service
Input Specification	Create typical Reservation request or Reservation manage
Output Specification	Check if the correct functions are called in the Data Service
Tested Functionalities	1. Check that when is called submitReservation on Reservation Manager sendReservation is called on DataService 2. Check that when is called submitModifyReservation on Reservation Manager sendModifyReservationData is called on DataService
Tested non Functionalities	Data Service check all input data
Environmental Needs	Reservation Manager Driver

3.3.5 Integration test case PMA-I5

Test Case Identifier	PMA-I5
Test Item	Request → GPS
Input Specification	Create typical Position request
Output Specification	Check if the correct functions are called in the GPS
Tested Functionalities	1. Check that when is called placeRequest on Request getPosition is called on GPS
Tested non Functionalities	1. GPS responds in 3s in 90% of cases
Environmental Needs	Request Driver

3.3.6 Integration test case PMA-I6

Test Case Identifier	PMA-I6
Test Item	Reservation Manager→ GPS
Input Specification	Create typical Position request
Output Specification	Check if the correct functions are called in the GPS
Tested Functionalities	1. Check that when is called placeReservation on Reservation Manager getPosition is called on GPS
Tested non Functionalities	1. GPS responds in 3s in 90% of cases
Environmental Needs	Reservation Manager Driver

3.3.7 Integration test case PMA-I7

Test Case Identifier	PMA-I7
Test Item	UI → Sign Up
Input Specification	Create typical IO Sign Up
Output Specification	Check if the correct functions are called in the Sign Up
Tested Functionalities	1. Check that when is called submittSignUp on UI sendSignUpData is called on Sign Up
Tested non Functionalities	1. Check that the response time not using Internet is smaller than 50ms in 99.9% of cases
Environmental Needs	PMA-I1 succeeded

3.3.8 Integration test case PMA-I8

Test Case Identifier	PMA-I8
Test Item	UI → Request
Input Specification	Create typical IO Taxi Request
Output Specification	Check if the correct functions are called in the Request
Tested Functionalities	1. Check that when is called sendRequest on UI submitRequest is called on Request
Tested non Functionalities	1. Check that the response time not using Internet is smaller than 50ms in 99.9% of cases
Environmental Needs	PMA-I3 succeeded

3.3.9 Integration test case PMA-I9

Test Case Identifier	PMA-I9
Test Item	UI → Sign In
Input Specification	Create typical IO Sign In Request
Output Specification	Check if the correct functions are called in the Sign In
Tested Functionalities	1. Check that when is called sendRequestSignIn on UI submitRequestSignIn is called on Sign In
Tested non Functionalities	1. Check that the response time not using Internet is smaller than 50ms in 99.9% of cases
Environmental Needs	PMA-I2 succeeded

3.3.10 Integration test case PMA-I10

Test Case Identifier	PMA-I10
Test Item	UI → Reservation Manager
Input Specification	Create typical IO handle Reservation request
Output Specification	Check if the correct functions are called in the Reservation Manager
Tested Functionalities	1. Check that when is called sendNewReservation on UI submitReservation is called on Reservation Manager 2. Check that when is called sendModifyOnReservation on UI submitModifyReservation is called on Reservation Manager
Tested non Functionalities	1. Check that the response time not using Internet is smaller than 50ms in 99.9% of cases
Environmental Needs	PMA-I6 succeeded

3.4 Sandwich Top Layer Integration

3.4.1 Integration test case TD-I1

Test Case Identifier	TD-I1
Test Item	PMA → Dispatcher
Input Specification	Create typical PMA input
Output Specification	Check if the correct functions are called in the Dispatcher
Tested Functionalities	1. Check that when a user performs a request it is received by Dispatcher
Tested non Functionalities	1. Check that a connection will be closed if a interaction using Internet attend more than 5s
Environmental Needs	All PMA-I succeeded

3.4.2 Integration test case TD-I2

Test Case Identifier	TD-I2
Test Item	TMA → Dispatcher
Input Specification	Create typical TMA input
Output Specification	Check if the correct functions are called in the Dispatcher
Tested Functionalities	1. Check that when a user performs an action it is received by Dispatcher
Tested non Functionalities	1. Check that a connection will be closed if a interaction using Internet attend more than 5s
Environmental Needs	All TMA-I succeeded

3.4.3 Integration test case TD-I3

Test Case Identifier	TD-I3
Test Item	Dispatcher → TMA
Input Specification	Create typical Dispatcher input
Output Specification	Check if the correct functions are called in the TMA
Tested Functionalities	1. Check that when an event is generated in Dispatcher a request is received by TMA
Tested non Functionalities	1. Check that a connection will be closed if a interaction using Internet attend more than 5s
Environmental Needs	TD-I2

3.5 Sandwich Lower Layer Integration

3.5.1 Integration test case BU-I1

Test Case Identifier	BU-I1
Test Item	Application Logic → Database
Input Specification	Create typical Application Logic input
Output Specification	Check if the correct functions are called in the Database
Tested Functionalities	1. Check that when the Application Logic receives a request that requires data a DBMS query is performed
Tested non Functionalities	1. Check that DB is performed at most in 200ms in 99.9% of times
Environmental Needs	All API-I succeeded

3.5.2 Integration test case BU-I2

Test Case Identifier	BU-I2
Test Item	Application Logic → Dispatcher
Input Specification	Create typical Application Logic input
Output Specification	Check if the correct functions are called in the Dispatcher
Tested Functionalities	1. Check that when an event is generated it is sent to the Dispatcher
Tested non Functionalities	-
Environmental Needs	BU-I1

3.5.3 Integration test case BU-I3

Test Case Identifier	BU-I3
Test Item	Dispatcher → Application Logic
Input Specification	Create typical Dispatcher input
Output Specification	Check if the correct functions are called in the Application Logic
Tested Functionalities	1. Check that when a message is received by Dispatcher an action is performed on Application Logic
Tested non Functionalities	-
Environmental Needs	BU-I3

Chapter 4

Tools and Test Equipment Required

Tools and Test Equipment Required

Into this paragraph we will speak about the tools we are going to use to support us during the integration test.

The tools we are going to use are:

- **JUnit**: It is a unit testing framework for the java programming language, but it can be very helpful for the integration test, despite its limitation.
- **ShrinkWrap**: It is a tool that allow us to assemble archive like JAR,EAR.
- **Arquillian**: It is an open source testing framework that by using both JUnit and ShrinkWrap, allow us to have a full control over which classes are injected during the integration test.
- **Jenkins**: It is an open-source server-based system that allow us to have continuous delivery and continuous integration of the software. Jenkins is very helpful because it give us the possibility to make an integration test in an automatic manner and help us to have a continuous check during the production of the code.
- **Jmeter**: It is use during the different phase of the testing for HTTP/HTTPS page both for client and system side, it can be used also for testing the database. It will be used also for the performance of our system by doing stress testing after the integration of our system will be complete.

4.1 Typology of testing

4.1.1 Manual testing

Manual testing is the process of manually testing software for defects. It requires a tester to play the role of an end user and use most of all features of the application to ensure correct behavior. To ensure completeness of testing, the tester often follows a written test plan that leads them through a set of important test cases.

4.1.2 Automated testing

In software testing, test automation is the use of special software (separate from the software being tested) to control the execution of tests and the comparison of actual outcomes with predicted outcomes. Test automation can automate some repetitive but necessary tasks in a formalized testing process already in place, or add additional testing that would be difficult to perform manually. Test automation is critical for continuous delivery and continuous testing.Through test automation developers and testers can easily and efficiently verify software quality, identify issues and speed time to market.

4.1.3 Our approach

The tools described in the previous section will be used for the test of modules that have to satisfy functional requirements; for the modules that have to satisfy not functional requirements, likes UI, we are going to check them using manual testing, because not functional requirements are impossible to check using automated testing. So the suggestion is to use the manual testing only when is strictly necessary.

Chapter 5

Program Stubs and Test Data Required

Program Stubs and Test Data Required

In order to perform Integration Test we are going to use some stub,driver or data that will substitute some sub-system during the integration to check the correctness of what we will build and, in case of error, we will be able to understand in which subsystem the problem should be.

Into the section 2.4 for each step of the integration of the various modules/subsystem we discuss about the driver/stubs we will use to do the integration test.

Here we are going to discuss of the different choose we have done during that step.

5.1 Integration test inside the subsystem

During the test of the different modules inside the subsystem, we decide to use only drivers.

5.1.1 PMA

Inside the Application logic we created different drivers to check the correctness of the different module that start to integrate. We will speak of each driver we used and why we use it.

5.1.1.1 Request Handler In Driver

We will create this driver in order to test this modules:

- Queue Manager
- State Manager
- Reservation Handler

- Account Manager

We substitute the Request Handler In whit the driver to test that the modules tested by that are able to work well and do the right operation depending on the information coming from the driver.

5.1.1.2 Queue Manager Driver

We will create this driver in order to test this modules:

- Database Manager
- Request Handler Out

We will test the Database Manager to be sure that it will be able to do the right operation and check of the data when it will be called by the Queue Manager.

The Request Handler Out will be tested to be sure that it will be able to receive the request coming from the Queue Manager and be able to do the right operation on them and in case to be able also to send it to the correct module.

5.1.1.3 State Manager Driver

We will create this driver in order to test this modules:

- Database Manager
- Request Handler Out

We will test the Database Manager to be sure that it will be able to do the right operation and check of the data when it will be called by the State Manager.

The Request Handler Out will be tested to be sure that it will be able to receive the request coming from the State Manager and be able to do the right operation in them and in case to be able also to send it to the correct module.

5.1.1.4 Reservation Handler Driver

We will create this driver in order to test this modules:

- Database Manager
- Request Handler Out

We will test the Database Manager to be sure that it will be able to do the right operation and check of the data when it will be called by the Reservation Handler.

The Request Handler Out will be tested to be sure that it is able to receive the request coming from the Reservation Handler and be able to do the right operation in them and in case to be able also to send it to the correct module.

5.1.1.5 Account Manager Driver

We will create this driver in order to test this modules:

- Database Manager
- Request Handler Out

We will test the Database Manager to be sure that it will be able to do the right operation and check of the data when it will be called by the Account Manager.

The Request Handler Out will be tested to be sure that it is able to receive the request coming from the Account Manager and be able to do the right operation in them and in case to be able also to send it to the correct module.

5.1.1.6 Taxi Distribution Manager Driver

We will create this driver in order to test this modules:

- Queue Manager
- State Manager

We will test the Queue Manager to be sure that it will be able to do the right operation when the Taxi Distribution manager ask for some operation(for example when it have to ask for a taxi, the Queue Manager have to return the correct taxi).

State Manager will be tested to be sure it will be able to do the right operation(for example change the state of a taxi) that the Taxi Manager requires.

5.1.2 TMA

Inside the TMA we will create some driver that we want to be used for the integration test. We will speak about the driver that will be created and what they are going to test.

5.1.2.1 Data Service Driver

We will create this driver in order to test this modules:

- Status Manager
- Request Answer
- GPS

We will test those modules to be sure that depending on the data that the Data Service give to them, they will be able to do the right operation (changing status, answer correctly to a request, give the position of the taxi).

5.1.2.2 Request Answer Driver

We will create this driver in order to to test the Data Service module.

This module will be tested to be sure that when the answer of a request will be send by the Request Answer, the Data Service will be able to check it and send it to the appropriate module.

5.1.2.3 Status Manager Driver

We will create this driver in order to test the Data Service module.

This module will be tested to be sure that when the Status Manager ask to change its status(for example when he have to change its status to "emergency") the Data Service will be able to check it and send it to the appropriate module.

5.1.2.4 Sign In Driver

We will create this driver in order to test the Data Service module.

This module will be tested to be sure that when the taxi driver(using the Sign In module) will ask to sign in, the Data Service will be able to check and send the data to the appropriate module.

5.1.2.5 UI Driver

We will create this driver in order to test the following modules:

- Request Answer
- GPS
- Status Manager

Those modules will be tested to be sure that they will be able to give the information the UI ask to show them to the taxi driver.

5.1.3 PMA

Inside the PMA we will create different driver that will be used for the integration test. We will speak about those driver and what they are going to test.

5.1.3.1 Sign Up Driver

We will create this driver in order to test the Data Service module.

This module will be tested in order to be sure that when the user use the Sign Up module, the Data Service will be able to check the data and in case to send to the correct module and give to the Sign Up the data it need.

5.1.3.2 Sign In Driver

We will create this driver in order to test the Data Service module.

This module will be tested in order to be sure that when the user use the Sign In module, the Data Service will be able to check the data and in case to send them to the correct module and give back the data the Sign In module need.

5.1.3.3 Reservation Manager Driver

We will create this driver in order to test the following modules:

- Data Service
- GPS

Data Service module will be tested in order to be sure that when the user use the Reservation Manager(checking his/her reservation, modify one of them, etc) module, the module will be able to check the data and in case to send them to the correct module and give back the data the Reservation Manager module need.

The GPS module will be tested to be sure that when a user have to insert a position, the GPS module will give the correct position of the user.

5.1.3.4 Request Driver

We will create this driver in order to test the GPS module.

This module will be tested in order to be sure that when a user want to do a request, he/she will be able to use the GPS to indicate his/her current position.

5.1.3.5 Driver

we will create this drive in order to test the following modules:

- Sign Up
- Request
- Sign In
- Reservation Manager

This module will be tested in order to be sure that they are able to give to the UI all the data it requires.

5.2 Integration test between subsystem

During the integration test between subsystem, we decide to use both drivers and stubs, depending on the case we will consider.

5.2.1 Dispatcher Stub

We will create this stub in order to test the following subsystem:

- PMA
- TMA

We will test this subsystem in order to be sure that they will be able to send the correct data to the Dispatcher and to be sure that they will be able also to do the right operation depending on the information the Dispatcher will send them.

5.2.2 TMA Stub

We will create this stub in order to test the Dispatcher subsystem.

We will test this subsystem in order to be sure that it will be able to make a preliminary check on the correctness on the data sent by the TMA and check also that it will be able to dispatch correctly the information given by the TMA.

5.2.3 Application Logic Driver

We will create this driver in order to test the following subsystem:

- Database
- Dispatcher

The Database will be tested in order to be sure that when the Application Logic have to receive some data from it, the data will be corrected and the one that it will have to receive. The Dispatcher will be tested in order to check if it will be able to send/receive the data of the Application Logic and, in case, to dispatch them correctly.

5.2.4 Dispatcher Driver

We will create this driver in order to test the Application Logic subsytem.

This will be tested in order to be sure that the data sent by the Dispatcher will be used correctly inside the Application Logic. Will be tested also the capacity of the Application Logic to provide correct data to the Dispatcher.

5.3 Special test data required

Considering the fact that the system interact with several external system we must check the integration between our system and the external systems that provide a service.

1. **ID**: EX-I1, **Test**: GPS of PMA→ Google Maps API
2. **ID**: EX-I2, **Test**: GPS of TMA→ Google Maps API