Integration Test Plan Document

Filippo Agalbato
850481

Andrea Cannizzo
790469

January 21, 2016

Contents

Chapter 1

Introduction

1.1 Purpose and scope

This is the Integration Test Plan Document for the project software My Taxi Service. The goal of this document is to describe the system's required integration testing framework and infrastructure that ought to prove its compliance with specifications, starting from these and proceeding up to the design phase of the different testing procedures.

This document is targeted to the project testers, to correctly build their infrastructure so that the integration testing phase can proceed according to plan.

General information on the My Taxi Service project itself may be found in the reference documents, as specified below.

1.2 Definitions, acronyms and abbreviations

- Component: High level element of the architectural design phase. Components make up the structure of the system;
- Module: Low level element of the architectural design phase. Modules join together to build a single component.

1.3 References

- Specification Document: My Taxi Service project specification for the Academic Year 2015-16 – *Assignments 1 and 2 (RASD and DD).pdf*;
- Specification Document: Testing assignment specification for the Academic Year 2015-16 – *Assignment 4 – Integration test plan.pdf*;
- Agalbato, Cannizzo, *Requirement Analysis and Specification Document*;
- Agalbato, Cannizzo, *Design Document.*

Chapter 2

Integration strategy

2.1 Entry criteria

Before the integration testing phase proper may begin, it is important that the code has been inspected, manually or otherwise, and that unit tests have been run, so that any bug or issue found in this phase may be safely flagged as being a problem of integration and not something that works as it shouldn't at a lower level.

2.2 Elements to be integrated

As detailed in the *Design Document*, the architecture is organized in several higher level components themselves divided in lower level modules. In the integration phase it is important to test both the interaction between modules and that among components. Refer to the *Design Document* itself for a detailed description of the elements. Follows here a list with a symbolic name, referred to in the schemes later on.

2.2.1 Client Tier

Central Messenger

A1 A module that interfaces with GUI;

A2 A module that interfaces with GPS Reader;

A3 A module that interfaces with Server Central Messenger;

A4 A module that handles and forwards messages to modules.

GPS Reader

B1 A module that interfaces with Central Messenger in order to receive requests and send information;

B2 A module that reads GPS information from Mobile APIs.

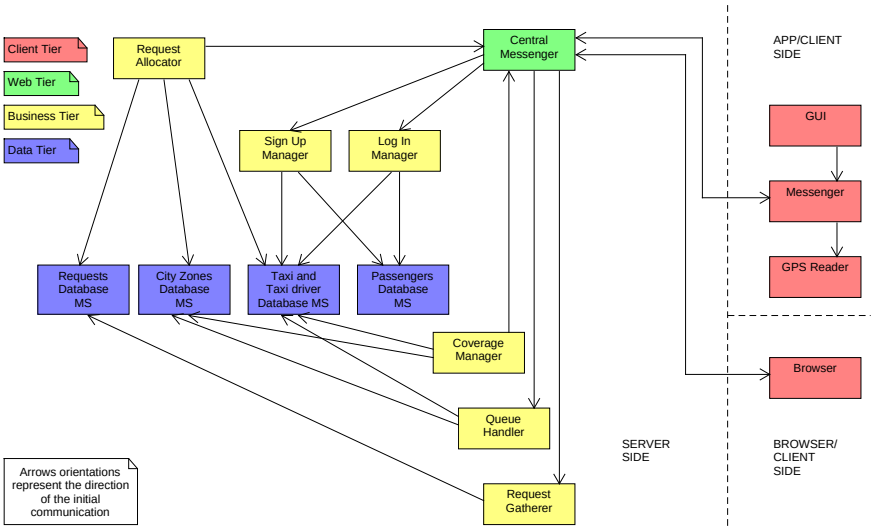

Figure 2.1: The different architectural components and their interaction – see the *Design Document*

Graphical User Interface

C1 A module that interfaces with Central Messenger in order to receive and send messages;

C2 A module that shows navigation pages.

2.2.2 Web Tier

Central Messenger

D1 A module that interfaces with Queue handler;

D2 A module that interfaces with Request Allocator;

D3 A module that interfaces with Request Gatherer;

D4 A module that interfaces with Coverage Manager;

D5 A module that interfaces with Sign Up/Log In manager;

D6 A module that interfaces with Client Central Messenger;

D7 A module that interfaces with Client Browser;

D8 A module that handles and forward messages throw modules.

2.2.3 Business Tier

Sign Up Manager

E1 A *Communication module* that interfaces with Central Messenger;

E2 An *Analysis module* that, reading form *Data module*, analyses the correctness of data and it eventually confirms or rejects;

E3 A *Data module* that stores data in the correct database and read needed information.

Log In Manager

These modules perform exactly as the Sign Up Manager, so their functional integration testing is handled in the exact same way.

E1 A *Communication module* that interfaces with Central Messenger:

E2 An *Analysis module* that, reading form *Data module*, analyses the correctness of data and it eventually confirms or rejects;

E3 A *Data module* that stores data in the correct database and read needed information.

Coverage Manager

F1 A *Data module* that interfaces with databases making DBMS queries;

F2 An *Analysis module* that analyses information received from *Data module* and calculates the best coverage;

F3 A *Communication module* that interfaces with Central Messenger in order to send messages to Taxi Drivers selected by *Analysis module*.

Request Gatherer

G1 A *Communication module* that receives requests messages from Central Messenger;

G2 An *Analysis Module* that analyses data integrity and requests information received from *Communication module* and eventually it sends to *Data Module*;

G3 A *Data Module* that stores information interfacing with databases.

Request Allocator

H1 A *Data Module* that, periodically reading from databases, controls if there are requests to fulfil and updates taxi drivers information;

H2 An *Analysis module* that choses taxi diver to allocate to the requests;

H3 A *Communication module* that interfaces with Central Messenger in order to send notifications to taxi drivers.

Queue Handler

I1 A *Communication module* that receives availability messages from Central Messenger;

I2 A *Data module* that stores and updates information in database.

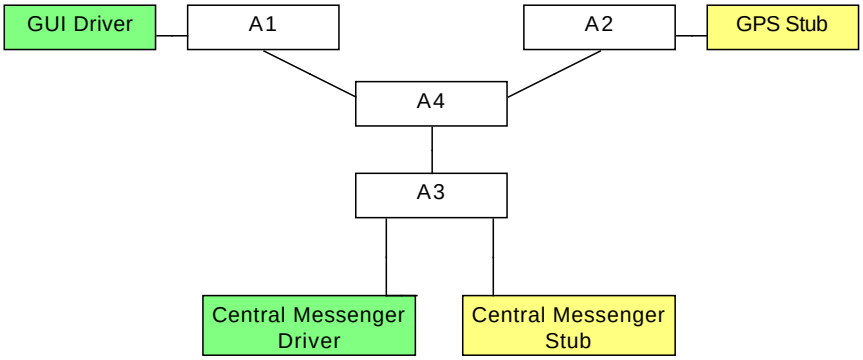

Figure 2.2: Integration testing in modules of the App Central Messenger component

2.3 Integration testing strategy

Since, as stated, the high level components are divided in low level modules, it is at first necessary to test the integration of the modules that make a single component before that same component can be tested against the others at the higher level.

A top down approach will be followed for the integration of the components. The process is very much simplified by the almost complete independence of most of the different components.

For the integration test of the modules inside components we have chosen to use a *Big Bang* approach that allows us to test the interaction between the modules of a specific component when they are all developed. We thought this was acceptable because of the small number of modules in each component. We also made this choice because in most cases modules are interfaces towards other components so their interaction is tested during integration test of components.

2.4 Sequence of integration

2.4.1 Modules

As stated, the chosen methodology is the *Big Bang* approach, due to the small number of modules each component is made of. Therefore, no explicit sequence is given for their integration.

2.4.2 Components

Since a top down (from the user interface) approach will be used, the integration order will be as follow:

1. Client App GUI;
2. Client App Messenger;

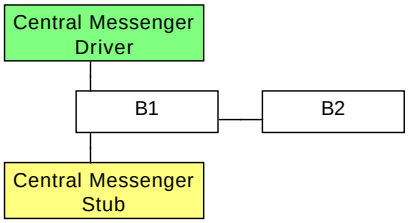

Figure 2.3: Integration testing in modules of the App GPS Reader component

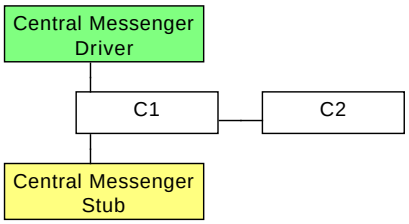

Figure 2.4: Integration testing in modules of the App Graphical User Interface component

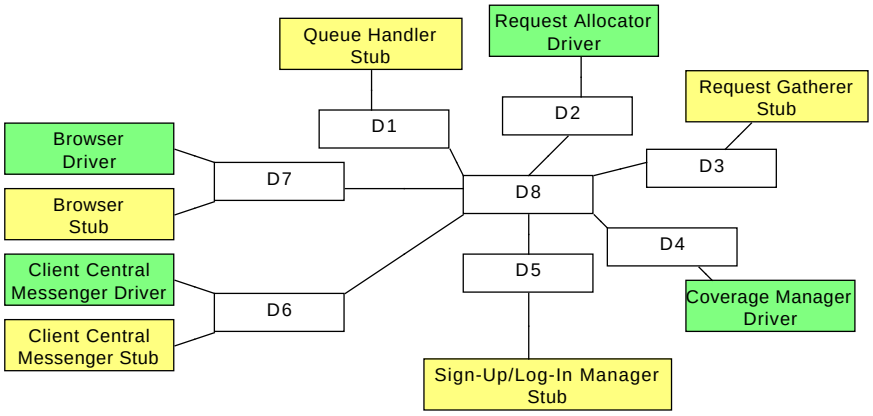

Figure 2.5: Integration testing in modules of the Server Central Messenger component

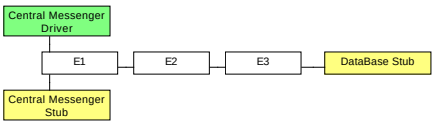

Figure 2.6: Integration testing in modules of the Server Sign-Up Manager and Log-In Manager components

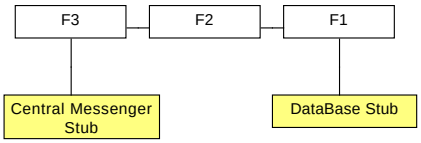

Figure 2.7: Integration testing in modules of the Server Coverage Manager component

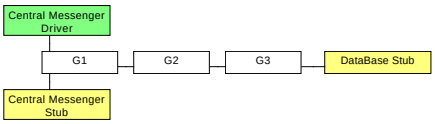

Figure 2.8: Integration testing in modules of the Server Coverage Manager component

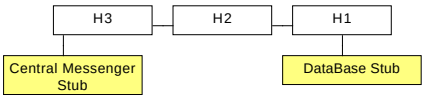

Figure 2.9: Integration testing in modules of the Server Request Allocator component

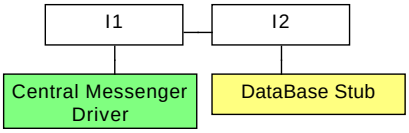

Figure 2.10: Integration testing in modules of the Server Queue Handler component

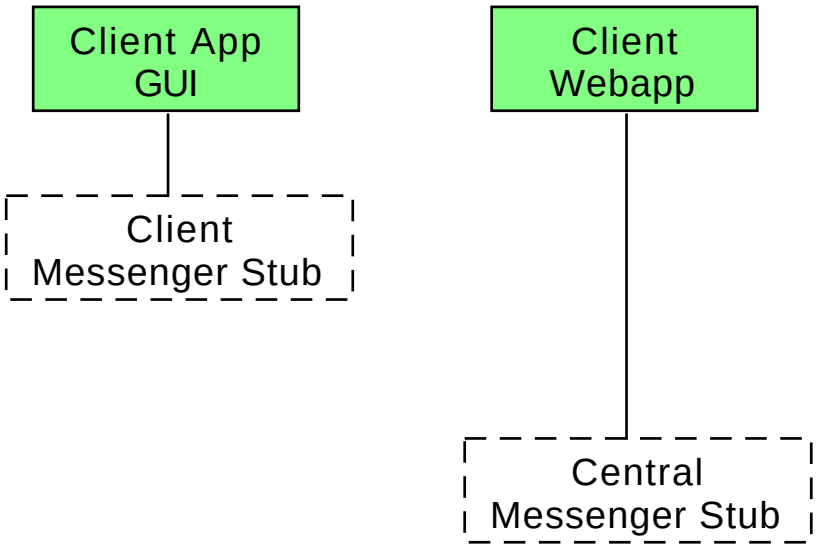

Figure 2.11: First step in integration testing for components

3. Client App GPS Reader;
4. Client Webapp (on par with the three previous components);
5. Central messenger;
6. Log in manager;
7. Sign up magager;
8. Request allocator;
9. Coverage manager;
10. Queue handler;
11. Request gatherer;
12. All Database Management Systems.

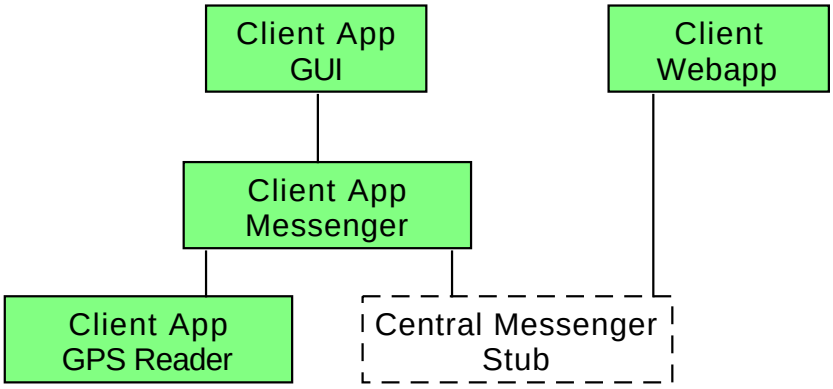

Figure 2.12: Second step in integration testing for components

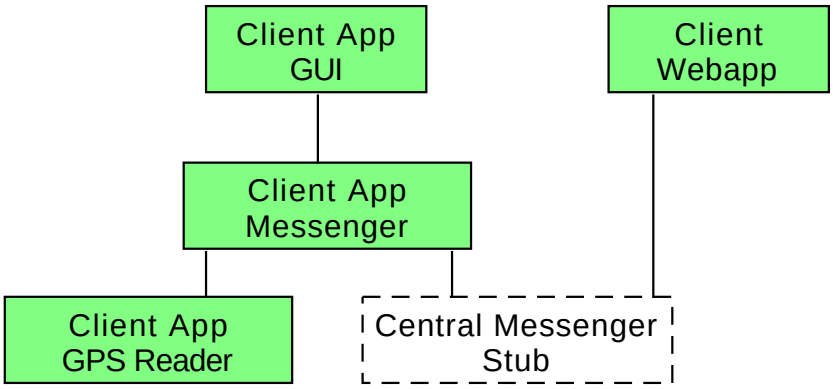

Figure 2.13: Third step in integration testing for components

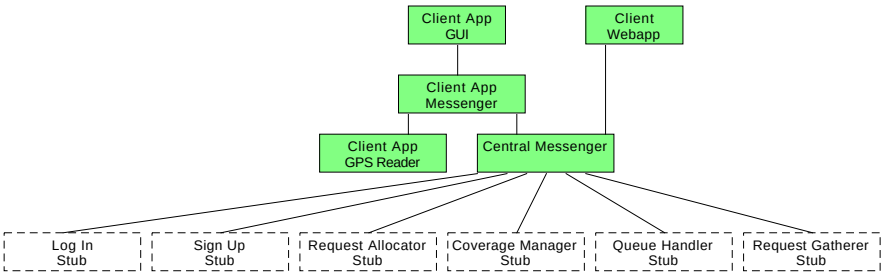

Figure 2.14: Fourth step in integration testing for components

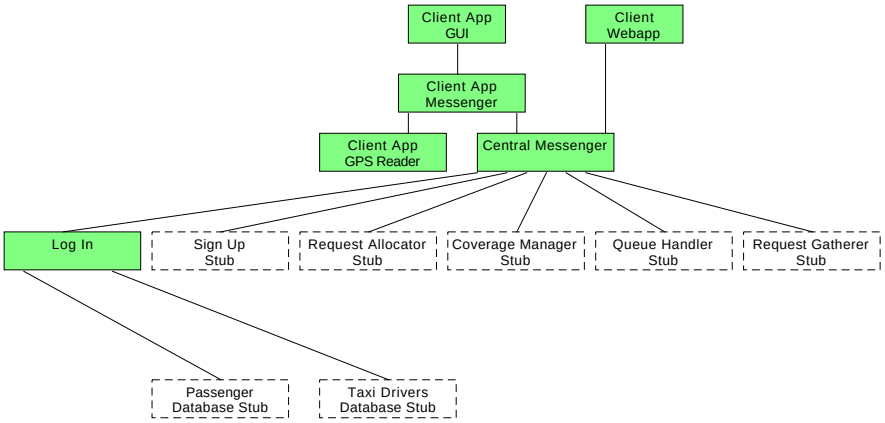

Figure 2.15: Fifth step in integration testing for components

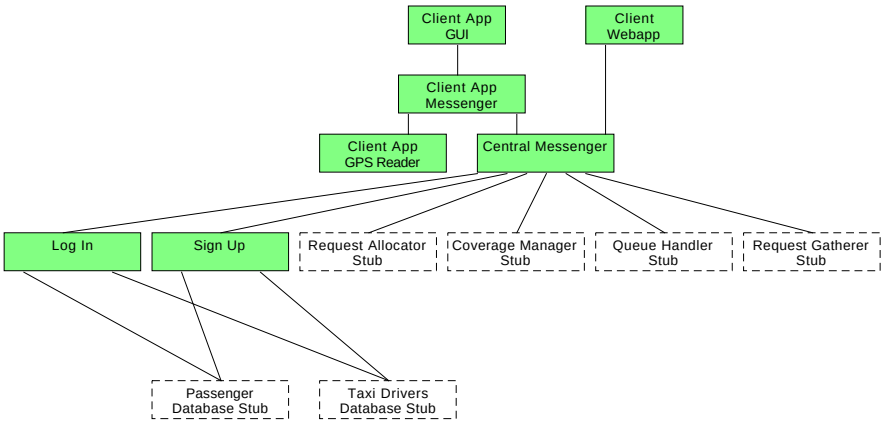

Figure 2.16: Sixth step in integration testing for components

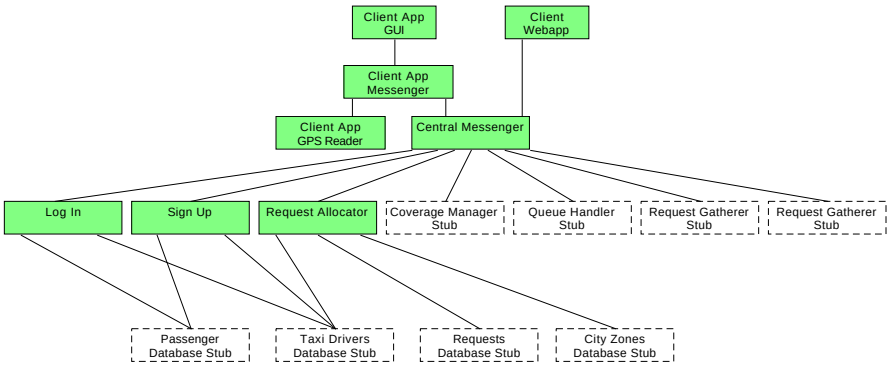

Figure 2.17: Seventh step in integration testing for components

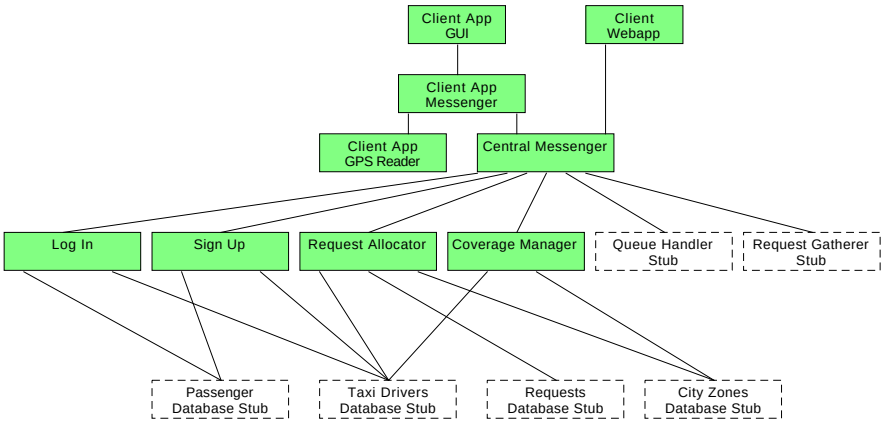

Figure 2.18: Eighth step in integration testing for components

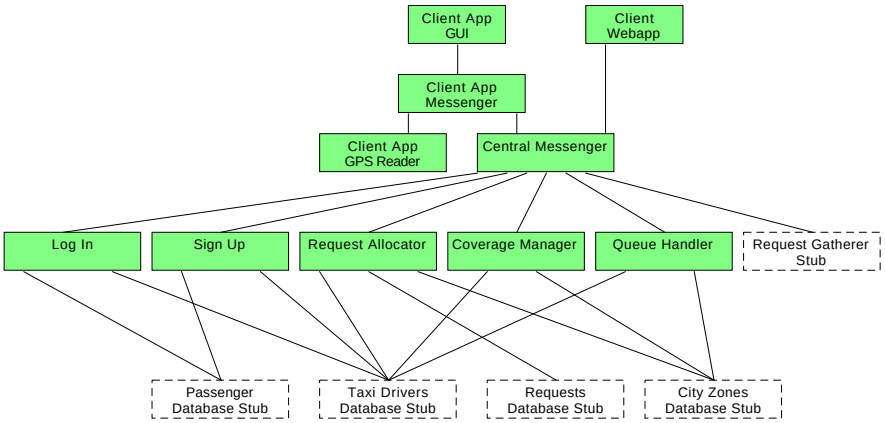

Figure 2.19: Ninth step in integration testing for components

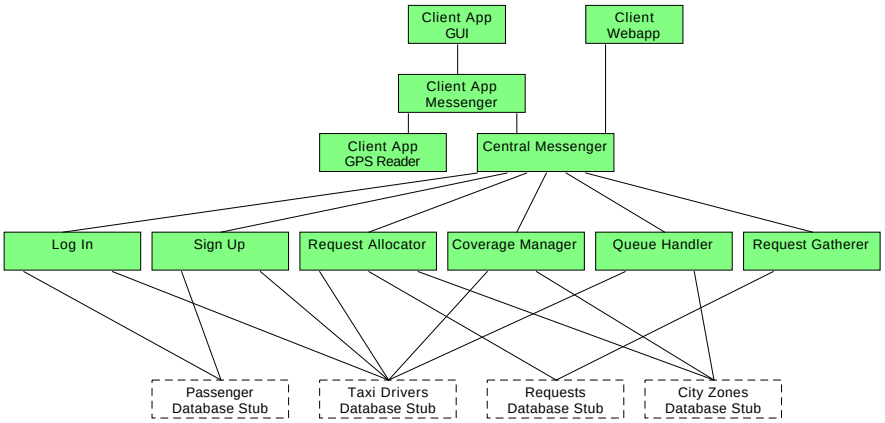

Figure 2.20: Tenth step in integration testing for components

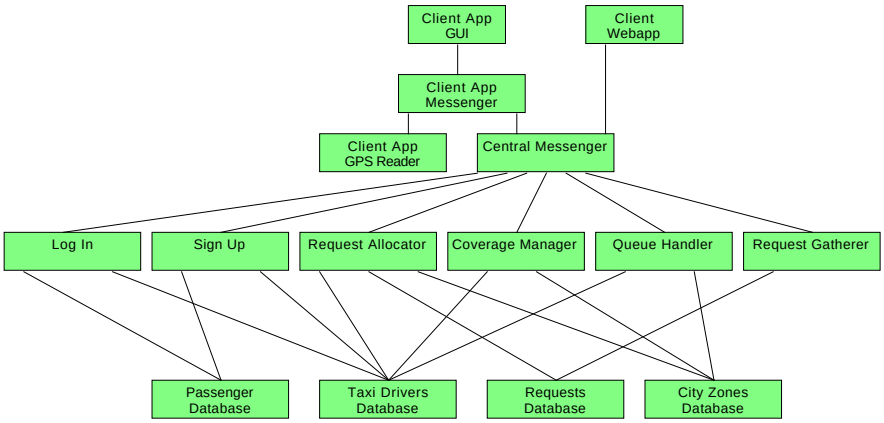

Figure 2.21: Final step in integration testing for components: all components integrated

Chapter 3

Individual steps and test description

Identifier	Component Integration 1
Purpose	• Test whether the App GUI properly handles user input; • Test whether the App GUI properly relays user input to the App Central Messenger.
Test item(s)	• App GUI;
Input Specification	An assortment of typical user inputs
Output specification	Check if the proper messages are relayed
Environmental needs	None

Identifier	Component Integration 2
Purpose	• Test whether the App Central Messenger properly receives user input from the GUI; • Test whether the App Central Messenger properly generates reactive output data to be displayed by the GUI.
Test item(s)	• App GUI; • App Central Messenger.
Input Specification	Typical user input data
Output specification	Check if the data is correct and is relayed and displayed properly
Environmental needs	Integration tests up to CI1 have been executed

Identifier	Component Integration 3
Purpose	• Test whether the App GPS Reader properly relays user input to the App Central Messenger and to the GUI; • Test whether the App GUI provides proper access to the GPS reader functionalities.
Test item(s)	• App GUI; • App Central Messenger; • App GPS reader.
Input Specification	Typical input data to access GPS readings in-app
Output specification	Check if the data is relayed and displayed properly
Environmental needs	Integration tests up to CI2 have been executed

Identifier	Component Integration 4
Purpose	• Test whether the App Central Messenger properly relays well-structured messages to the Server Central Messenger; • Test whether the Client Webapp properly relays well-structured messages to the Server Central Messenger; • Test whether the Server Central Messenger responds properly to client requests by sending the correct answers (webpages to the Webapp, messages to the App); • Test whether the Server Central Messenger answers the right queries with the right data.
Test item(s)	• App Central Messenger; • Client Webapp; • Server Central Messenger.
Input Specification	Typical well-formed message data
Output specification	Check if the data is relayed properly over the internet
Environmental needs	Integration tests up to CI3 have been executed

Chapter 4

Tools and test equipment required

Since a top-down approach has been used to plan the integration testing suites for the components, it is necessary to use a specific framework to not only create and run automated tests but also to build and manage the necessary components stubs. This said, *Arquillian* looks like the perfect tool for the job, with support from *JUnit*'s test cases and infrastructure. We reserve the possibility of using manual testing for the Graphical User Interface parts.

At this point in the process, we are exploring just the integration level of the several different possible test sequences. How the single units have been tested, and how the system as a whole will be tested, is not part of this document. Thus, no tools are indicated for those activities.

Chapter 5

Program stubs and testing data required

Beyond the basic functionalities required to any stub (that is, providing a mock of the functions it is called upon to mimic so that the related component may be tested in safety), running integration testing suites on this project requires some implementation of the four databases that have been mentioned (Requests, City zones, Taxi and Taxi driver, Passengers). These should contain well-strctured and valid data, used by many components to test their own functionalities.

Follows a list of all the required stubs:

- stub uno
- stub due
- stub qualcosa

Appendix A

Document and work information

A.1 Revisions

This is the first version of this document. There are currently no revisions.

A.2 Tools used

TeXworks editor With PDFLaTeX, for composing and editing this document;

UMLet For drawing all kinds of diagrams.

A.3 Overall time spent

The authors spent about 20 hours of their time, equally divided among them, working on this document.