Politecnico di Milano

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

Computer Science and Engineering

Software Engineering 2 project

myTaxiService
Integration Test Plan Document

Version 1

Prof. Elisabetta Di Nitto

Andrea Autelitano	Matr. 849869
Marco De Cobelli	Matr. 858360
Matthew Rossi	Matr. 858880

Academic year 2015-2016

TABLE OF CONTENTS

1 Introduction

1.1 Revision History

Version	Date	Author(s)	Description
1.0	20-01-2016	Marco De Cobelli, Andrea Autelitano, Matthew Rossi	Integration Test Plan Document v1.0

1.2 Purpose and Scope

1.2.1 Purpose

This document describes the plans for testing the integration of the created components. The purpose of integration testing is to test the interfaces between the components described in [SDD, Chapter 2]: each software module shall be combined and tested as a group. Functional requirements described in [RASD, Chapter 3] shall be tested as well. Every team member who cooperates in the integration tests should read this document.

1.2.2 Scope

The aim of this project is to optimize the taxi service of a large city thanks to the simplification of the access to the service for the users and the fair management of taxi queues.
The registered users will be able to request or book a taxi from either a mobile application or a web site and Taxi drivers will use the mobile application to accept the requests of the users.
The city is divided into zones of approximately 2km^2, each of these with a different queue of taxis.
The system will choose the taxi from the zone that is nearest to the user that has made the request/reservation. Usually, it is their own zone.
Furthermore, when a taxi becomes available, either because its taxi driver logs into the system or the related taxi driver reports the termination of a ride, its identifier will be added at the end of the queue of the current zone in which he is located.
When the first taxi driver in a queue receives a request, they can freely choose whether to accept it or to refuse it. In the first case, a notification is sent to the client saying that one taxi driver has accepted their request, in the second case the request is sent to the second taxi driver in the queue and so on.
Moreover, there is the possibility to reserve a taxi from a certain origin to a specific destination within two hours before the ride; in this case, 10 minutes before the ride the system will allocate it to the first taxi driver in the queue.
Finally, besides the traditional interfaces, programmatic interfaces are provided in order to enable the development of additional services like the taxi sharing option. This option allows users to share the cost of a ride with other people after specifying the starting point, the destination of the ride, the number of people to carry on and the availability to share the ride.
In any case, the system automatically calculates the route of the taxi toward the destinations using Google Maps and it provides the fee for each person in the taxi and notifies the users and the Taxi driver who has accepted the ride before the departure.

1.3 List of Definitions and Abbreviations

1.3.1 Definitions

Client	A person that uses the application in order to request rides
Taxi driver	A person that uses their taxi in order to provide the ride
User	Both a Client and a Taxi driver
Route	The ride that has been booked by a Client
Ride	The ride that has been allocated to a Taxi driver and that can be composed of a single route or in the case of a shared ride, more than one route
Typical data	Data that are of the type required by a specific mask (e.g. a String for the name of the customer, an Integer for the CVV of a credit card…)
Abnormal data	Data that are NOT of the type required by a specific mask (e.g. an Integer for the name of the customer, a String for the CVV of a credit card…)

1.3.2 Abbreviations

RASD	Requirements Analysis and Specification Document
SDD	Software Design Description
UI	User Interface
OS	Operating System
DBMS	Database Management System
IDE	Integrated Development Environment

1.4 List of Reference Documents

1.4.1 External Documentation

ID	Name of the document	Date
[A0]	AA 2015-2016 Software Engineering 2 - Project goal, schedule and rules	October 2015
[A1]	Software Engineering 2 Project, AA 2015-2016 Assignments 1 and 2	October 2015
[A4]	Software Engineering 2 Project, AA 2015-2016 Assignment 4, Test Plan	January 2016
[EX1]	Spingrid - Integration Test Plan	May 2006
[JUD]	JUnit documentation	December 2015
[MFPD]	Maven Failsafe Plugin documentation	December 2015
[AD]	Arquillian documentation	August 2015

1.4.2 Internal Documentation

ID	Name of the document	Version	Date
[RASD]	myTaxiService - Requirements Analysis and Specification Document	2.0	December 2015
[SDD]	myTaxiService – Software Design Description	1.0	December 2015
[UTPD]	myTaxiService – Unit Test Plan Document		

2 Integration Strategy

2.1 Entry Criteria

Before the integration test can begin, the following document must have been delivered:

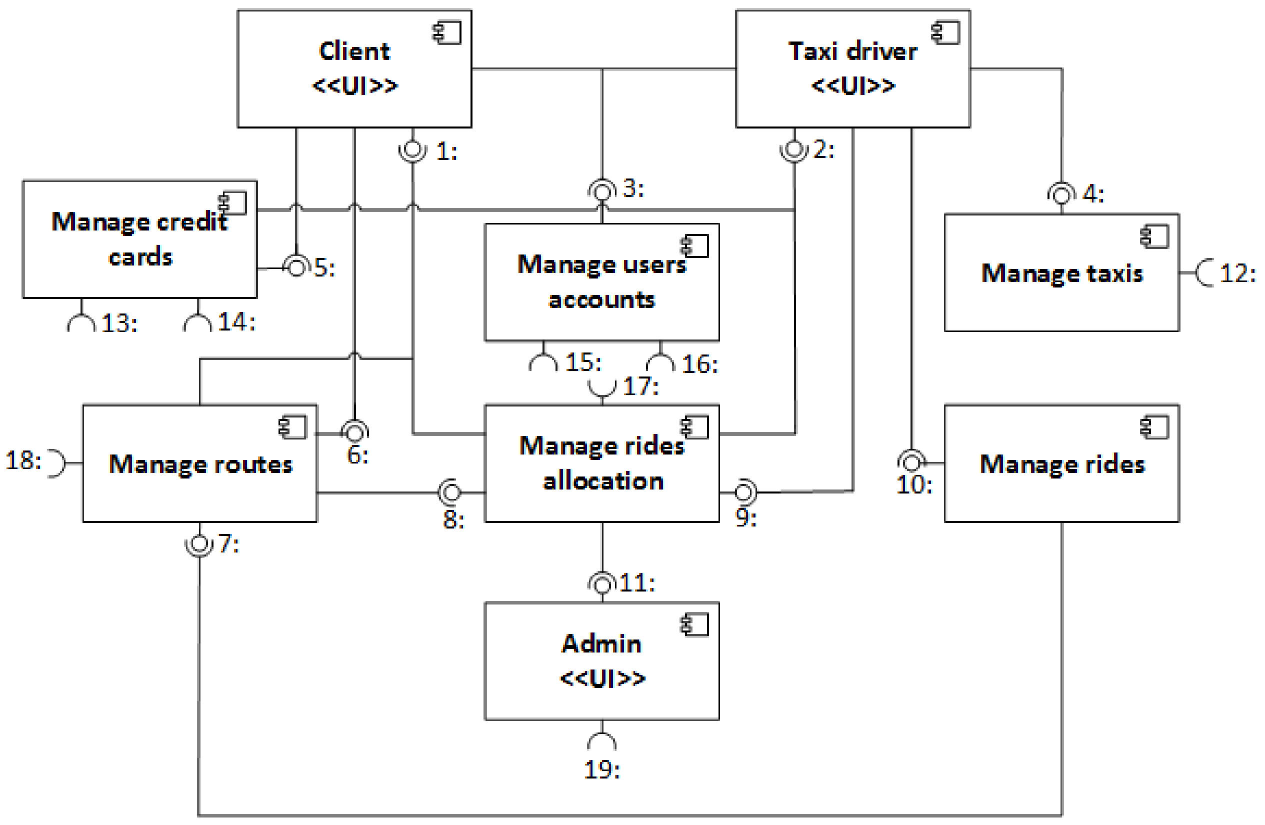
- myTaxiService - Requirements Analysis and Specification Document [RASD]
- myTaxiService – Software Design Description [SDD]
- myTaxiService – Unit Test Plan Document [UTPD]

In addition to that, all software components described in [SDD] must have been implemented in order to reduce the number of drivers (software modules that simulate higher level component when using a bottom-up approach) or stubs (software modules that simulate lower level component when using a top-down approach) to be coded. Moreover, all software components must have been unit tested: this is important because in the case of a failed integration test case we can be sure that the problem was not linked to how the module had been implemented but only to an error in the implementation of the interface. Interfaces with external systems must have been already tested as well.

Finally, before an actual test case can begin, this document must have been completed and approved.

2.2 Elements to be Integrated

The image below shows the high level component diagram of the whole system since all the components of the system will be integrated together in order to accomplish the integration test plan.

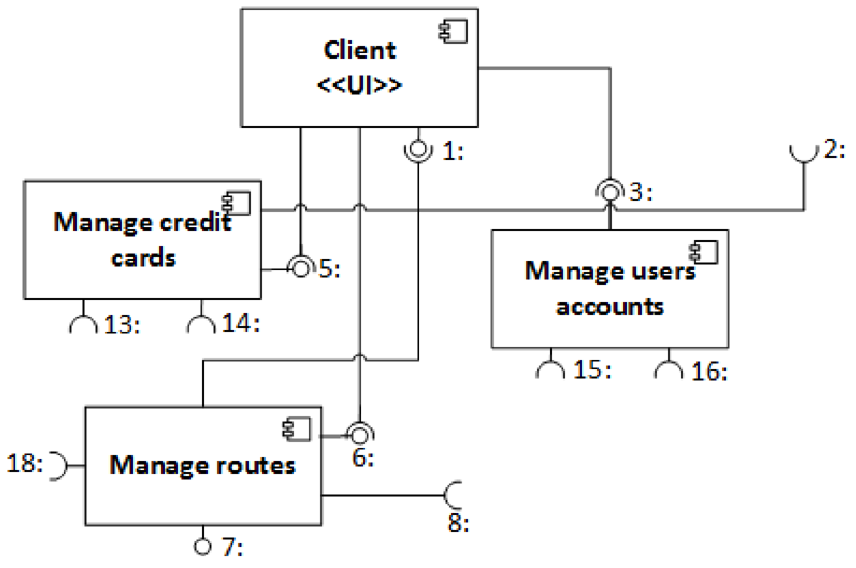


2.3 Integration Testing Strategy

As integration testing strategy we have decided to apply a mixture of the functional grouping, bottom-up and top-down approaches.
The choice of both bottom-up and top-down approaches are quite obvious: the former choice has been done since we already know the architecture of the software to be developed (see [SDD, Chapter 2]) and all components have already been implemented and unit-tested, while the latter has been done in order to test the interfaces used by a calling component (i.e. user interfaces). However, using only this approach might lead to unexpected problems during the integration of a larger number of components, therefore we have decided to apply also a functional grouping approach.
We have identified three major subsystems in which we can group components with similar functionalities:

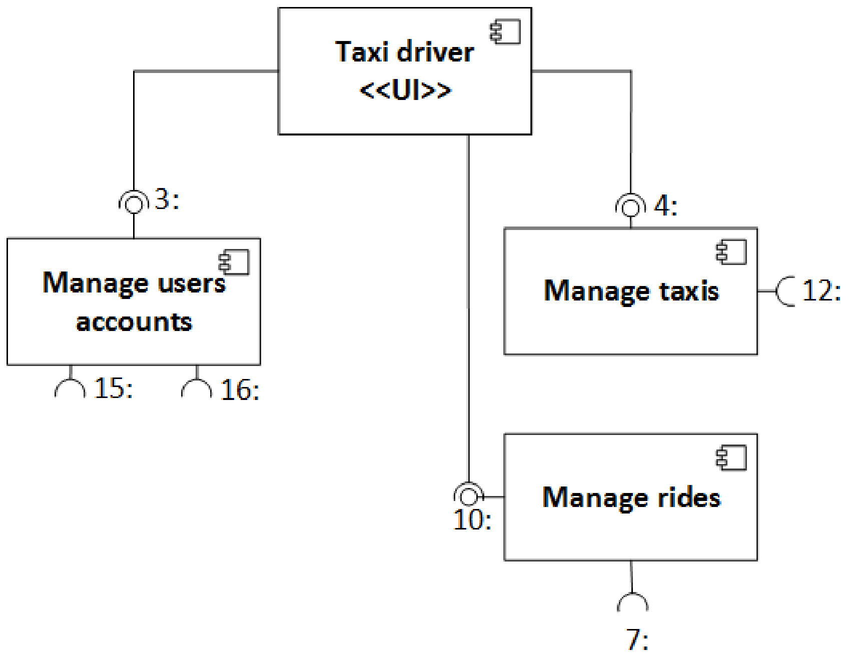
- **Client functionalities**, with components
  - Client <<UI>>
  - Manage credit cards
  - Manage users accounts
  - Manage routes

  and interfaces 3, 5 and 6.



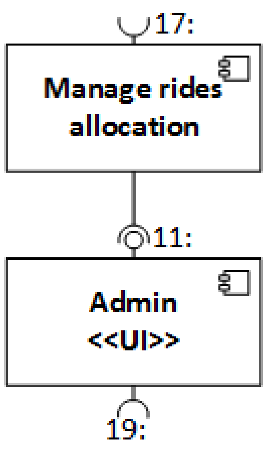
- **Taxi Driver functionalities**, with components
  - Taxi driver <<UI>>
  - Manage users accounts
  - Manage rides
  - Manage taxis

  and interfaces 3, 4, 10.



- **Administration functionalities**, with components
  - Admin <<UI>>
  - Manage rides allocation

  and interface 11.



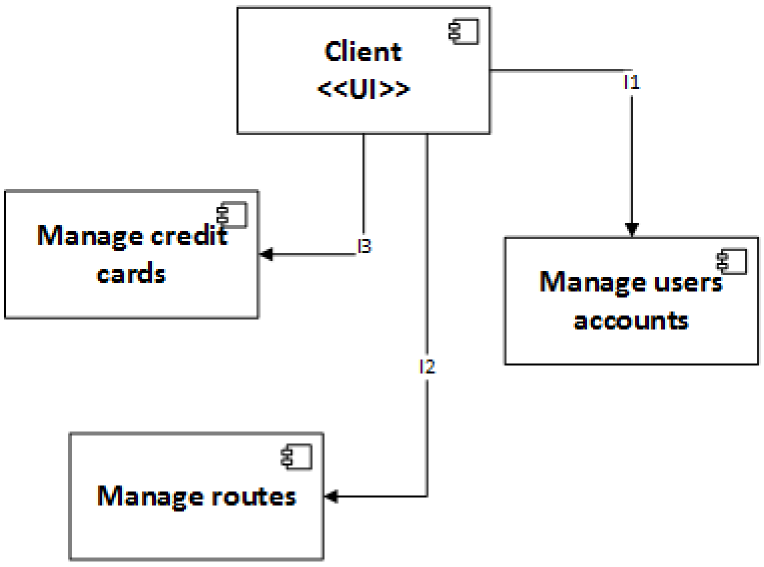
First, the integration among the components of each subsystem shall be tested; then, we will proceed with the integration of the three subsystems using the remaining interfaces (1, 2, 7, 8, 9). We have decided to apply this approach to verify the integration of components that will take part in the same course of events during the normal usage of the software by the System Administrator, the Client or the Taxi Driver.

2.4 Sequence of Component/Function Integration

2.4.1 Software Integration Sequence

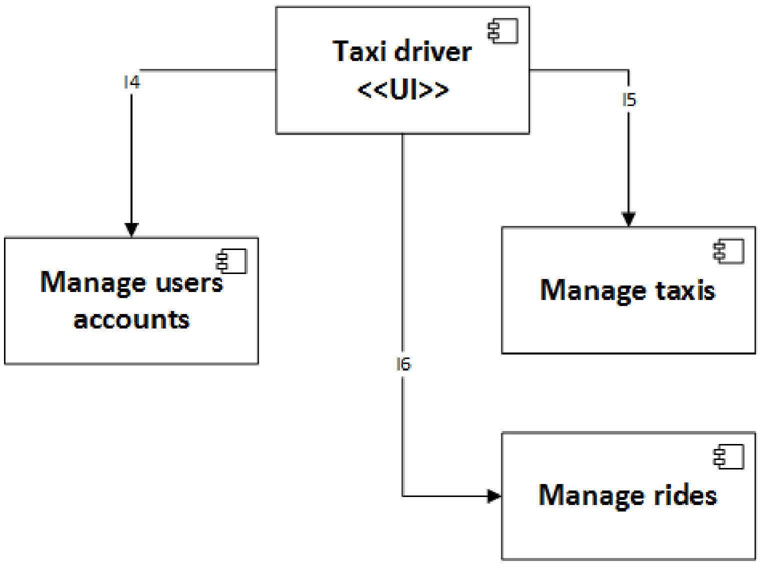
The IDs represent the order in which the integration testing should proceed; however, the integration of components of different subsystems can be run in parallel.

2.4.1.1 Client functionalities



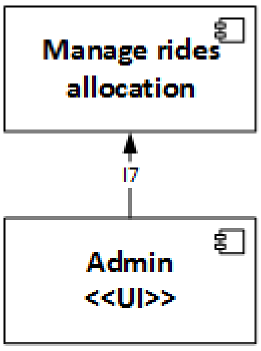
ID	Integration test	Paragraphs
I1	Client <<UI>> → Manage users accounts	3.1.1 3.2.1
I2	Client <<UI>> → Manage routes	3.1.2 3.2.1
I3	Client <<UI>> → Manage credit cards	3.1.3 3.2.1

2.4.1.2 Taxi Driver functionalities



ID	Integration test	Paragraphs
I4	Taxi Driver <<UI>> → Manage users accounts	3.1.4 3.2.2
I5	Taxi Driver <<UI>> → Manage taxis	3.1.5 3.2.2
I6	Taxi Driver <<UI>> → Manage rides	3.1.6 3.2.2

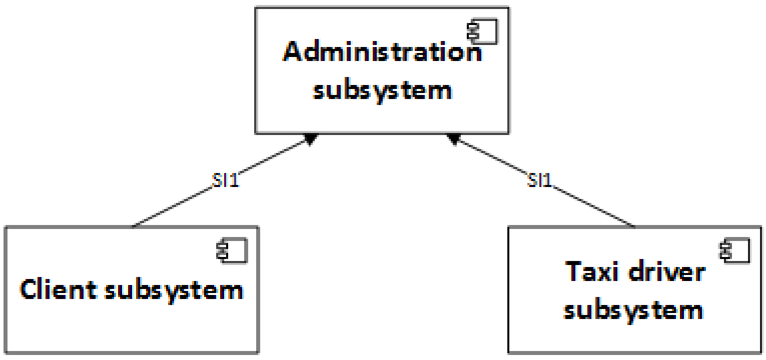
2.4.1.3 Administration functionalities



ID	Integration test	Paragraphs
I7	Admin <<UI>> → Manage rides allocation	3.1.7 3.2.3

2.4.2 Subsystem Integration Sequence

The IDs represent the order in which the integration testing should proceed; however, each subsystem must have been already tested.



ID	Integration test	Paragraphs
SI1	Client subsystem, Taxi driver subsystem → Administration subsystem	3.1.8 3.2.4

3 Individual Steps and Test Description

3.1 Integration test cases

3.1.1 Integration test case I1

Test Case Identifier	I1T1
Test Item(s)	Client <<UI>> → Manage users accounts
Input Specification	Create typical Client <<UI>> input in order to test the registration
Output Specification	Check if the client information is successfully stored and the activation email is sent
Environmental Needs	-

Test Case Identifier	I1T2
Test Item(s)	Client <<UI>> → Manage users accounts
Input Specification	Create abnormal Client <<UI>> input in order to test the registration
Output Specification	Check if the client is still at the registration interface with error explanation
Environmental Needs	-

Test Case Identifier	I1T3
Test Item(s)	Client <<UI>> → Manage users accounts
Input Specification	Create typical Client <<UI>> input in order to test the login
Output Specification	Check if the client session is correctly created
Environmental Needs	I1T1 – I1T2 succeeded

Test Case Identifier	I1T4
Test Item(s)	Client <<UI>> → Manage users accounts
Input Specification	Create abnormal Client <<UI>> input in order to test the login
Output Specification	Check if is still at the login interface with error explanation
Environmental Needs	I1T1 – I1T2 succeeded

Test Case Identifier	I1T5
Test Item(s)	Client <<UI>> → Manage users accounts
Input Specification	Create typical Client <<UI>> input in order to test the information modification
Output Specification	Check if the client profile is modified accordingly to the inserted input
Environmental Needs	I1T1 – I1T4 succeeded

Test Case Identifier	I1T6
Test Item(s)	Client <<UI>> → Manage users accounts
Input Specification	Create abnormal Client <<UI>> input in order to test the information modification
Output Specification	Check if the client profile is not modified
Environmental Needs	I1T1 – I1T4 succeeded

Test Case Identifier	I1T7
Test Item(s)	Client <<UI>> → Manage users accounts
Input Specification	Create typical Client <<UI>> session in order to test the logout
Output Specification	Check if the client session is closed
Environmental Needs	I1T1 – I1T4 succeeded

Test Case Identifier	I1T8
Test Item(s)	Client <<UI>> → Manage users accounts
Input Specification	Create abnormal Client <<UI>> session in order to test the logout
Output Specification	Check that no modification is done on the client session
Environmental Needs	I1T1 – I1T4 succeeded

Test Case Identifier	I1T9
Test Item(s)	Client <<UI>> → Manage users accounts
Input Specification	Create typical Client <<UI>> input in order to test the password recovery
Output Specification	Check if an email is sent to the client with the new password
Environmental Needs	I1T1 – I1T2 succeeded

Test Case Identifier	I1T10
Test Item(s)	Client <<UI>> → Manage users accounts
Input Specification	Create abnormal Client <<UI>> input in order to test the password recovery
Output Specification	Check that no email is sent and the client is still at password recovery interface
Environmental Needs	I1T1 – I1T2 succeeded

3.1.2 Integration test case I2

Test Case Identifier	I2T1
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create typical Client <<UI>> input in order to test the creation of a not shared request
Output Specification	Check if the request is stored and sent to the ride allocator
Environmental Needs	I1 succeeded, Manage ride allocation stub

Test Case Identifier	I2T2
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create abnormal Client <<UI>> input in order to test the creation of a not shared request
Output Specification	Check that no request is stored or sent
Environmental Needs	I1 succeeded, Manage ride allocation stub

Test Case Identifier	I2T3
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create typical Client <<UI>> input in order to test the creation of a shared request
Output Specification	Check if the request is stored and sent to the ride allocator
Environmental Needs	I1 succeeded, Manage ride allocation stub

Test Case Identifier	I2T4
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create abnormal Client <<UI>> input in order to test the creation of a shared request
Output Specification	Check that no request is stored or sent
Environmental Needs	I1 succeeded, Manage ride allocation stub

Test Case Identifier	I2T5
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create typical Client <<UI>> input in order to test the creation of a not shared reservation
Output Specification	Check if the reservation is stored and sent to the ride allocator
Environmental Needs	I1 succeeded

Test Case Identifier	I2T6
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create abnormal Client <<UI>> input in order to test the creation of a not shared reservation
Output Specification	Check that no reservation is stored or sent
Environmental Needs	I1 succeeded

Test Case Identifier	I2T7
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create typical Client <<UI>> input in order to test the creation of a shared reservation
Output Specification	Check if the reservation is stored and sent to the ride allocator
Environmental Needs	I1 succeeded

Test Case Identifier	I2T8
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create abnormal Client <<UI>> input in order to test the creation of a shared reservation
Output Specification	Check that no reservation is stored or sent
Environmental Needs	I1 succeeded

Test Case Identifier	I2T9
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create typical Client <<UI>> session in order to test the routes visualization
Output Specification	Check if all the pending and concluded routes are shown to the client
Environmental Needs	I1 and I2T1- I2T9 succeeded

Test Case Identifier	I2T10
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create abnormal Client <<UI>> session in order to test the routes visualization
Output Specification	Check that no routes are shown
Environmental Needs	I1 and I2T1- I2T9 succeeded

Test Case Identifier	I2T11
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create typical Client <<UI>> input in order to test the modification of a not shared route
Output Specification	Check if the route is modified accordingly to the inserted input
Environmental Needs	I1, I2T1- I2T2 and I2T5-I2T6 succeeded

Test Case Identifier	I2T12
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create abnormal Client <<UI>> input in order to test the modification of a not shared routes
Output Specification	Check that no route is modified
Environmental Needs	I1, I2T1- I2T2 and I2T5-I2T6 succeeded

Test Case Identifier	I2T13
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create typical Client <<UI>> input in order to test the modification of a shared routes
Output Specification	Check if the route is modified accordingly to the inserted input
Environmental Needs	I1, I2T3- I2T4 and I2T7-I2T8 succeeded

Test Case Identifier	I2T14
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create abnormal Client <<UI>> input in order to test the modification of a shared routes
Output Specification	Check that no route is modified
Environmental Needs	I1, I2T3- I2T4 and I2T7-I2T8 succeeded

Test Case Identifier	I2T15
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create typical Client <<UI>> input in order to test the cancelation of a not shared routes
Output Specification	Check if the route is cancelled
Environmental Needs	I1, I2T1- I2T2 and I2T5-I2T6 succeeded

Test Case Identifier	I2T16
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create abnormal Client <<UI>> input in order to test the cancelation of a not shared routes
Output Specification	Check that the route is still there
Environmental Needs	I1, I2T1- I2T2 and I2T5-I2T6 succeeded

Test Case Identifier	I2T17
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create typical Client <<UI>> input in order to test the cancelation of a shared routes
Output Specification	Check if the route is cancelled
Environmental Needs	I1, I2T3- I2T4 and I2T7-I2T8 succeeded

Test Case Identifier	I2T18
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create abnormal Client <<UI>> input in order to test the cancelation of a shared routes
Output Specification	Check that the route is still there
Environmental Needs	I1, I2T3- I2T4 and I2T7-I2T8 succeeded

Test Case Identifier	I2T19
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create typical Client <<UI>> input in order to test the price recalculation
Output Specification	Check if the price is recalculated accordingly to the changes in the destination
Environmental Needs	I1 and I2T1-I2T8 succeeded, Manage ride driver

Test Case Identifier	I2T20
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create abnormal Client <<UI>> input in order to test the price recalculation
Output Specification	Check that no price changes are done
Environmental Needs	I1 and I2T1-I2T8 succeeded, Manage ride driver

3.1.3 Integration test case I3

Test Case Identifier	I3T1
Test Item(s)	Client <<UI>> → Manage credit cards
Input Specification	Create typical Client <<UI>> input in order to test the addition of a new credit card
Output Specification	Check if the credit card is stored
Environmental Needs	I1 succeeded

Test Case Identifier	I3T2
Test Item(s)	Client <<UI>> → Manage routes
Input Specification	Create abnormal Client <<UI>> input in order to test the addition of a new credit card
Output Specification	Check that no credit card is inserted into the system
Environmental Needs	I1 succeeded

Test Case Identifier	I3T3
Test Item(s)	Client <<UI>> → Manage credit cards
Input Specification	Create typical Client <<UI>> input in order to test the cancelation of a credit card
Output Specification	Check if the selected credit card is cancelled
Environmental Needs	I1 and I3T1-I3T2 succeeded

Test Case Identifier	I3T4
Test Item(s)	Client <<UI>> → Manage credit cards
Input Specification	Create abnormal Client <<UI>> input in order to test the cancelation of a credit card
Output Specification	Check that no credit cards is cancelled
Environmental Needs	I1 and I3T1-I3T2 succeeded

Test Case Identifier	I3T5
Test Item(s)	Client <<UI>> → Manage credit cards
Input Specification	Create typical Client <<UI>> session in order to test the visualization of their own credit cards
Output Specification	Check if all and only the Client credit cards are shown
Environmental Needs	I1 and I3T1-I3T2 succeeded

Test Case Identifier	I3T6
Test Item(s)	Client <<UI>> → Manage credit cards
Input Specification	Create abnormal Client <<UI>> session in order to test the visualization of their own credit cards
Output Specification	Check that no credit cards are shown
Environmental Needs	I1 and I3T1-I3T2 succeeded

Test Case Identifier	I3T7
Test Item(s)	Client <<UI>> → Manage credit cards
Input Specification	Create typical Client <<UI>> input in order to test the payment of a route using a new credit card
Output Specification	Check if the route payment transaction is completed successfully
Environmental Needs	I2 succeeded, Taxi driver stub

Test Case Identifier	I3T8
Test Item(s)	Client <<UI>> → Manage credit cards
Input Specification	Create abnormal Client <<UI>> input in order to test the payment of a route using a new credit card
Output Specification	Check that no payment is done
Environmental Needs	I2 succeeded, Taxi driver stub

Test Case Identifier	I3T9
Test Item(s)	Client <<UI>> → Manage credit cards
Input Specification	Create typical Client <<UI>> input in order to test the payment of a route using a selected credit card
Output Specification	Check if the route payment transaction is completed successfully
Environmental Needs	I2 succeeded, Taxi driver stub

Test Case Identifier	I3T10
Test Item(s)	Client <<UI>> → Manage credit cards
Input Specification	Create abnormal Client <<UI>> input in order to test the payment of a route using a selected credit card
Output Specification	Check that no payment is done
Environmental Needs	I2 succeeded, Taxi driver stub

3.1.4 Integration test case I4

Test Case Identifier	I4T1
Test Item(s)	Taxi Driver <<UI>> → Manage users accounts
Input Specification	Create typical Taxi Driver <<UI>> input in order to test: - Registration
Output Specification	Check that the Taxi Driver information is successfully stored and the activation email is sent
Environmental Needs	-

Test Case Identifier	I4T2
Test Item(s)	Taxi Driver <<UI>> → Manage users accounts
Input Specification	Create abnormal Taxi Driver <<UI>> input in order to test: - Registration
Output Specification	Check that the Taxi Driver is still at the registration interface with error explanation
Environmental Needs	-

Test Case Identifier	I4T3
Test Item(s)	Taxi Driver <<UI>> → Manage users accounts
Input Specification	Create typical Taxi Driver <<UI>> input in order to test: - Login
Output Specification	Check that the Taxi Driver session is correctly created and that he/she’s logged in the system as a Taxi Driver
Environmental Needs	I4T1, I4T2 succeeded

Test Case Identifier	I4T4
Test Item(s)	Taxi Driver <<UI>> → Manage users accounts
Input Specification	Create abnormal Taxi Driver <<UI>> input in order to test: - Login
Output Specification	Check that the Taxi Driver session isn’t created and that he/she’s logged in the system as a Taxi Driver
Environmental Needs	I4T1, I4T2 succeeded

Test Case Identifier	I4T5
Test Item(s)	Taxi Driver <<UI>> → Manage users accounts
Input Specification	Create typical Taxi Driver <<UI>> input in order to test: - Modification of personal information
Output Specification	Check that the information of the Taxi Driver is modified
Environmental Needs	I4T1-I4T4 succeeded

Test Case Identifier	I4T6
Test Item(s)	Taxi Driver <<UI>> → Manage users accounts
Input Specification	Create abnormal Taxi Driver <<UI>> input in order to test: - Modification of personal information
Output Specification	Check that the information of the Taxi Driver is NOT modified

Environmental Needs	I4T1-I4T4 succeeded

Test Case Identifier	I4T7
Test Item(s)	Taxi Driver <<UI>> → Manage users accounts
Input Specification	Create typical Taxi Driver <<UI>> input in order to test: - Password recovery
Output Specification	Check if an email is sent to the Taxi Driver with the new password
Environmental Needs	I4T1-I4T4 succeeded

Test Case Identifier	I4T8
Test Item(s)	Taxi Driver <<UI>> → Manage users accounts
Input Specification	Create abnormal Taxi Driver <<UI>> input in order to test: - Password recovery
Output Specification	Check that no email is sent and the Taxi Driver is still at password recovery interface
Environmental Needs	I4T1-I4T4 succeeded

Test Case Identifier	I4T9
Test Item(s)	Taxi Driver <<UI>> → Manage users accounts
Input Specification	Create typical Taxi Driver <<UI>> session in order to test: - Logout
Output Specification	Check that the Taxi Driver session is closed
Environmental Needs	I4T1, I4T2 succeeded

Test Case Identifier	I4T10
Test Item(s)	Taxi Driver <<UI>> → Manage users accounts
Input Specification	Create abnormal Taxi Driver <<UI>> session in order to test: - Logout
Output Specification	Check that no modification is done on the Taxi Driver session
Environmental Needs	I4T1, I4T2 succeeded

3.1.5 Integration test case I5

Test Case Identifier	I5T1
Test Item(s)	Taxi Driver <<UI>> → Manage taxis
Input Specification	Create typical Taxi Driver <<UI>> input in order to test: - Addition of a new taxi
Output Specification	Check that the taxi information is inserted into the system
Environmental Needs	I4 succeeded

Test Case Identifier	I5T2
Test Item(s)	Taxi Driver <<UI>> → Manage taxis
Input Specification	Create abnormal Taxi Driver <<UI>> input in order to test: - Addition of a new taxi
Output Specification	Check that the taxi information is NOT inserted into the system
Environmental Needs	I4 succeeded

Test Case Identifier	I5T3
Test Item(s)	Taxi Driver <<UI>> → Manage taxis
Input Specification	Create typical Taxi Driver <<UI>> input in order to test: - Modification of a taxi
Output Specification	Check that the taxi information is modified
Environmental Needs	I4 succeeded I5T1, I5T2 succeeded

Test Case Identifier	I5T4
Test Item(s)	Taxi Driver <<UI>> → Manage taxis
Input Specification	Create abnormal Taxi Driver <<UI>> input in order to test: - Modification of a taxi
Output Specification	Check that the taxi information is NOT modified
Environmental Needs	I4 succeeded I5T1, I5T2 succeeded

Test Case Identifier	I5T5
Test Item(s)	Taxi Driver <<UI>> → Manage taxis
Input Specification	Create typical Taxi Driver <<UI>> session in order to test: - Removal of a taxi
Output Specification	Check that the selected taxi is deleted
Environmental Needs	I4 succeeded I5T1, I5T2 succeeded

Test Case Identifier	I5T6
Test Item(s)	Taxi Driver <<UI>> → Manage taxis
Input Specification	Create abnormal Taxi Driver <<UI>> session in order to test: - Removal of a taxi
Output Specification	Check that the selected taxi is NOT deleted
Environmental Needs	I4 succeeded I5T1, I5T2 succeeded

3.1.6 Integration test case I6

Test Case Identifier	I6T1
Test Item(s)	Taxi Driver <<UI>> → Manage rides
Input Specification	Create typical Taxi Driver <<UI>> input in order to test: - Standard termination of the current ride
Output Specification	Check that the ride is considered concluded from the system and the relative information is correctly modified
Environmental Needs	I4, I5 succeeded

Test Case Identifier	I6T2
Test Item(s)	Taxi Driver <<UI>> → Manage rides
Input Specification	Create abnormal Taxi Driver <<UI>> input in order to test: - Standard termination of the current ride
Output Specification	Check that the ride is still pending and the relative information is NOT modified
Environmental Needs	I4, I5 succeeded

Test Case Identifier	I6T3
Test Item(s)	Taxi Driver <<UI>> → Manage rides
Input Specification	Create typical Taxi Driver <<UI>> input in order to test: - Termination of a ride before have reached the destination
Output Specification	Check that the ride is stopped and the relative information is correctly modified
Environmental Needs	I4, I5 succeeded

Test Case Identifier	I6T4
Test Item(s)	Taxi Driver <<UI>> → Manage rides
Input Specification	Create abnormal Taxi Driver <<UI>> input in order to test: - Termination of a ride before have reached the destination
Output Specification	Check that the ride is NOT stopped and the relative information is NOT correctly modified
Environmental Needs	I4, I5 succeeded

Test Case Identifier	I6T5
Test Item(s)	Taxi Driver <<UI>> → Manage rides
Input Specification	Create typical Taxi Driver <<UI>> input in order to test: - Change of the price for the clients of the current ride when it is stopped by the Taxi Driver before have reached the destination
Output Specification	Check that the prices for the clients change correctly
Environmental Needs	I4, I5 succeeded, Manage routes (stub)

Test Case Identifier	I6T6
Test Item(s)	Taxi Driver <<UI>> → Manage rides
Input Specification	Create abnormal Taxi Driver <<UI>> input in order to test: - Change of the price for the clients of the current ride when it is stopped by the Taxi Driver before have reached the destination
Output Specification	Check that the prices for the clients don’t change
Environmental Needs	I4, I5 succeeded, Manage routes (stub)

Test Case Identifier	I6T7
Test Item(s)	Taxi Driver <<UI>> → Manage rides
Input Specification	Create typical Taxi Driver <<UI>> session in order to test: - Visualization of the rides already completed
Output Specification	Check that the completed rides are correctly visualized
Environmental Needs	I4, I5 succeeded

Test Case Identifier	I6T8
Test Item(s)	Taxi Driver <<UI>> → Manage rides
Input Specification	Create abnormal Taxi Driver <<UI>> session in order to test: - Visualization of the rides already completed
Output Specification	Check that the completed rides are NOT visualized
Environmental Needs	I4, I5 succeeded

Test Case Identifier	I6T9
Test Item(s)	Taxi Driver <<UI>> → Manage rides
Input Specification	Create typical Taxi Driver <<UI>> session in order to test: - Visualization of the pending rides
Output Specification	Check that the pending rides are correctly visualized
Environmental Needs	I4, I5 succeeded

Test Case Identifier	I6T10
Test Item(s)	Taxi Driver <<UI>> → Manage rides
Input Specification	Create abnormal Taxi Driver <<UI>> session in order to test: - Visualization of the pending rides
Output Specification	Check that the pending rides are NOT visualized
Environmental Needs	I4, I5 succeeded

3.1.7 Integration test case I7

Test Case Identifier	I7T1
Test Item(s)	Admin <<UI>> → Manage rides allocation
Input Specification	Testing of the correct retrieval of information on zone queues status
Output Specification	Check that the information on the queues status shown is correct
Environmental Needs	-

Test Case Identifier	I7T2
Test Item(s)	Admin <<UI>> → Manage rides allocation
Input Specification	Testing of the correct retrieval of information on statistics of concluded rides
Output Specification	Check that the statistics shown are correct
Environmental Needs	-

3.1.8 Integration test case SI1

Test Case Identifier	SI1T1
Test Item(s)	Client subsystem, Taxi driver subsystem → Administration subsystem
Input Specification	Create typical Client <<UI>> input in order to test the payment of a route using a new credit card
Output Specification	Check if the Taxi Driver <<UI>> receives and shows the notification of the occurred payment transaction
Environmental Needs	I1-I7 and TP1-TP3 succeeded

Test Case Identifier	SI1T2
Test Item(s)	Client subsystem, Taxi driver subsystem → Administration subsystem
Input Specification	Create abnormal Client <<UI>> input in order to test the payment of a route using a new credit card
Output Specification	Check that no notification is sent to the Taxi Driver <<UI>>
Environmental Needs	I1-I7 and TP1-TP3 succeeded

Test Case Identifier	SI1T3
Test Item(s)	Client subsystem, Taxi driver subsystem → Administration subsystem
Input Specification	Create typical Client <<UI>> input in order to test the payment of a route using a selected credit card
Output Specification	Check if the Taxi Driver <<UI>> receives and shows the notification of the occurred payment transaction
Environmental Needs	I1-I7 and TP1-TP3 succeeded

Test Case Identifier	SI1T4
Test Item(s)	Client subsystem, Taxi driver subsystem → Administration subsystem
Input Specification	Create abnormal Client <<UI>> input in order to test the payment of a route using a selected credit card
Output Specification	Check that no notification is sent to the Taxi Driver <<UI>>
Environmental Needs	I1-I7 and TP1-TP3 succeeded

Test Case Identifier	SI1T5
Test Item(s)	Client subsystem, Taxi driver subsystem → Administration subsystem
Input Specification	Create typical Client <<UI>> input in order to test the creation of a not shared request
Output Specification	Check if Taxi Driver <<UI>> receives and shows the notification of the ride in order to allocate it successfully
Environmental Needs	I1-I7 and TP1-TP3 succeeded

Test Case Identifier	SI1T6
Test Item(s)	Client subsystem, Taxi driver subsystem → Administration subsystem
Input Specification	Create abnormal Client <<UI>> input in order to test the creation of a not shared request
Output Specification	Check that no notification is sent to the Taxi Driver <<UI>>
Environmental Needs	I1-I7 and TP1-TP3 succeeded

Test Case Identifier	SI1T7
Test Item(s)	Client subsystem, Taxi driver subsystem → Administration subsystem
Input Specification	Create typical Client <<UI>> input in order to test the creation of a shared request
Output Specification	Check if Taxi Driver <<UI>> receives and shows the notification of the ride in order to allocate it successfully
Environmental Needs	I1-I7 and TP1-TP3 succeeded

Test Case Identifier	SI1T8
Test Item(s)	Client subsystem, Taxi driver subsystem → Administration subsystem
Input Specification	Create abnormal Client <<UI>> input in order to test the creation of a shared request
Output Specification	Check that no notification is sent to the Taxi Driver <<UI>>
Environmental Needs	I1-I7 and TP1-TP3 succeeded

Test Case Identifier	SI1T9
Test Item(s)	Client subsystem, Taxi driver subsystem → Administration subsystem
Input Specification	Create typical Client <<UI>> input in order to test the creation of a not shared reservation
Output Specification	Check if Taxi Driver <<UI>> receives and shows the notification of the ride in order to allocate it successfully
Environmental Needs	I1-I7 and TP1-TP3 succeeded

Test Case Identifier	SI1T10
Test Item(s)	Client subsystem, Taxi driver subsystem → Administration subsystem
Input Specification	Create abnormal Client <<UI>> input in order to test the creation of a not shared reservation
Output Specification	Check that no notification is sent to the Taxi Driver <<UI>>
Environmental Needs	I1-I7 and TP1-TP3 succeeded

Test Case Identifier	SI1T11
Test Item(s)	Client subsystem, Taxi driver subsystem → Administration subsystem
Input Specification	Create typical Client <<UI>> input in order to test the creation of a shared reservation
Output Specification	Check if Taxi Driver <<UI>> receives and shows the notification of the ride in order to allocate it successfully
Environmental Needs	I1-I7 and TP1-TP3 succeeded

Test Case Identifier	SI1T12
Test Item(s)	Client subsystem, Taxi driver subsystem → Administration subsystem
Input Specification	Create abnormal Client <<UI>> input in order to test the creation of a shared reservation
Output Specification	Check that no notification is sent to the Taxi Driver <<UI>>
Environmental Needs	I1-I7 and TP1-TP3 succeeded

Test Case Identifier	SI1T13
Test Item(s)	Client subsystem, Taxi driver subsystem → Administration subsystem
Input Specification	Create typical Taxi Driver <<UI>> session in order to test: - Change of the availability state of a Taxi Driver
Output Specification	Check that the availability state of the Taxi Driver is changed
Environmental Needs	I1-I7 and TP1-TP3 succeeded

Test Case Identifier	SI1T14
Test Item(s)	Client subsystem, Taxi driver subsystem → Administration subsystem
Input Specification	Create abnormal Taxi Driver <<UI>> session in order to test: Change of the availability state of a Taxi Driver
Output Specification	Check that the availability state of the Taxi Driver is NOT changed
Environmental Needs	I1-I7 and TP1-TP3 succeeded

Test Case Identifier	SI1T15
Test Item(s)	Client subsystem, Taxi driver subsystem → Administration subsystem
Input Specification	Create typical Taxi Driver <<UI>> session in order to test: - Change of the correct sending of the current zone position
Output Specification	Check that the position sent is correct
Environmental Needs	I1-I7 and TP1-TP3 succeeded

Test Case Identifier	SI1T16
Test Item(s)	Client subsystem, Taxi driver subsystem → Administration subsystem
Input Specification	Create abnormal Taxi Driver <<UI>> session in order to test: - Change of the correct sending of the current zone position
Output Specification	Check that the position is NOT sent
Environmental Needs	I1-I7 and TP1-TP3 succeeded

Test Case Identifier	SI1T17
Test Item(s)	Client subsystem, Taxi driver subsystem → Administration subsystem
Input Specification	Create typical Taxi Driver <<UI>> input in order to test: - Change of the price for the clients of the current ride when it is stopped by the Taxi Driver before have reached the destination
Output Specification	Check that the prices for the clients change correctly
Environmental Needs	I1-I7 and TP1-TP3 succeeded

Test Case Identifier	SI1T18
Test Item(s)	Client subsystem, Taxi driver subsystem → Administration subsystem
Input Specification	Create abnormal Taxi Driver <<UI>> input in order to test: - Change of the price for the clients of the current ride when it is stopped by the Taxi Driver before have reached the destination
Output Specification	Check that the prices for the clients don’t change
Environmental Needs	I1-I7 and TP1-TP3 succeeded

3.2 Integration test procedures

3.2.1 Integration test procedure TP1

Test Procedure Identifier	TP1
Purpose	This test procedure verifies whether the Client subsystem: • can handle Client input • can output requested information to the Client
Procedure Steps	Execute I3 after I2 and I2 after I1

3.2.2 Integration test procedure TP2

Test Procedure Identifier	TP2
Purpose	This test procedure verifies whether the Taxi Driver subsystem: • can handle Taxi driver input • can output requested information to the Taxi driver
Procedure Steps	Execute I6 after I5 and I5 after I4

3.2.3 Integration test procedure TP3

Test Procedure Identifier	TP3
Purpose	This test procedure verifies whether Administrator subsystem: • can handle Administrator input • can output requested information to the Administrator
Procedure Steps	Execute I8

3.2.4 Integration test procedure TP4

Test Procedure Identifier	TP4
Purpose	This test procedure verifies whether the system: • can handle Client input • can handle Taxi driver input • can handle Administrator input • can output requested information to the Client • can output requested information to the Taxi driver • can output requested information to the Administrator
Procedure Steps	Execute SI1

4 Tools and Test Equipment Required

As tools for the integration testing we have decided to exploit both automatic and manual approach. In the first phase (integration of components of the same subsystem), applying manual testing every time an integration test needs to be executed is surely a waste of time and resources: therefore, some automatic testing tools shall be used. In particular, the ones we have chosen are:

- **JUnit** (version 4.12) and **Arquillian** (version 1.1.10) to create test cases in the development environment.
    - JUnit has been chosen because it is an open source framework that is easy to use and it is one of the most popular for the Java programming language, therefore it has a large community that can help solving potential problems.
    - Arquillian has been chosen because it can be used to manage the life cycle of containers, to wrap together test cases, to test dependency classes and resources and it offers two different modes, which will be both exploited: the In-Container mode to test the interaction among components in the server side, and Client mode to test the interaction between client and server components;

- **Maven Failsafe Plugin** (version 2.19.1) to automate setup, execution and verification of the tests. The Maven lifecycle has four phases for running integration tests:
    - *pre-integration-test*: for setting up the integration test environment.
    - *integration-test*: for running the integration tests.
    - *post-integration-test*: for tearing down the integration test environment.
    - *verify*: for checking the results of the integration tests.

  Using Maven, the testing team will not need to manually repeat the preparation and verification phases each time something has been modified in the test cases and/or in the code.

In the second phase (integration of different subsystems), the software in its completeness will be available: therefore, it will be possible to apply manual testing. It will be used mainly for two different tasks:
- to test critical functionalities of the system (e.g. payment via credit card);
- to test the correct functioning of more complex use cases that involve a larger number of components and for which the usage of an user interface would simplify the task instead of writing data directly in the Java code.

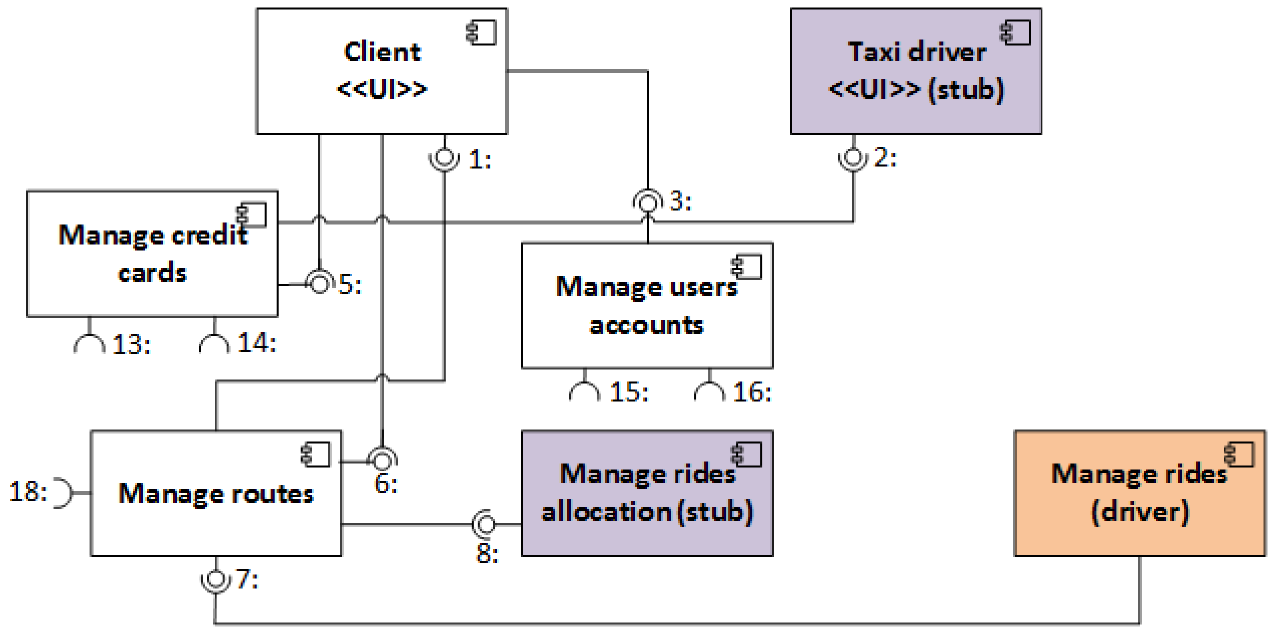
As testing environment, we have decided to use a local server machine with the following configuration:

OS	Ubuntu Server 14.04 LTS
Web Server	Apache HTTP Server 2.4
Application Server	Apache Tomcat 8
DBMS	PostgreSQL 9.4

The tests will be prepared on a desktop PC running Windows 10 Home with NetBeans 8.1 as IDE.
The same PC will be used in the subsystem integration phase: the web browsers Mozilla Firefox, Google Chrome and Microsoft Edge will be used to access the service.
These configuration were chosen in order to cover the greater number of possible clients that will use the service and to guarantee the portability of the system.

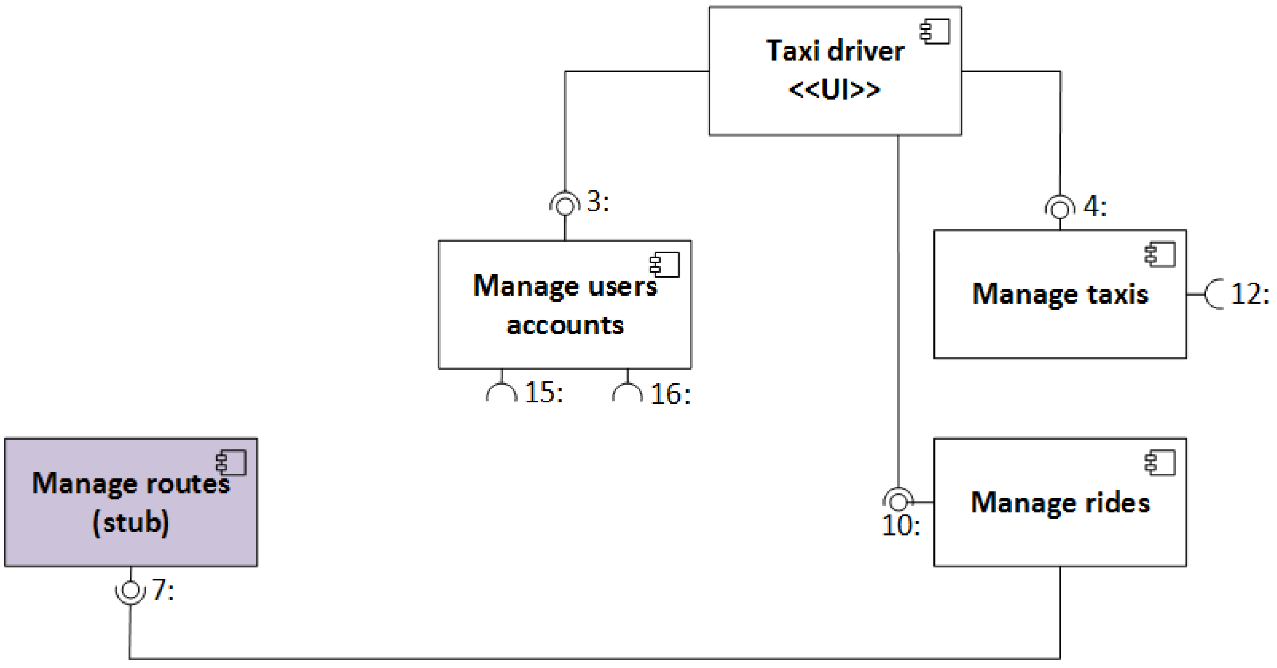
5 Required program stubs and drivers

5.1 Client functionalities



- Taxi driver<<UI>> (stub): Listens for push notification and shows them when received
- Manage rides allocation (stub): Listens for invocation of allocation functionalities sent by the Manage routes components and show them when received
- Manage rides (driver): Simulates a destination modification requested by a Taxi driver and then asks for a route price recalculation

5.2 Taxi driver functionalities



- Manage routes (stub): Listens for the invocation of the route price recalculation functionality and show the prices when recalculated.

6 Appendix

6.1 Used tools

- Microsoft Word 2013 to write this document
- Microsoft Visio 2013 to create UML diagrams

6.2 Hours of work

Each member of the group has worked on this document for 25 hours.