Politecnico di Milano
A.A. 2015-2016

Software Engineering 2: myTaxiService

Integration **T**est **P**lan

version 1.0

Matteo Bulloni (852676), Marco Cannici (852527)

21 January 2016

Contents

1 Introduction

1.1 Revision History

Document Title	Integration Test Plan Document
Version	1.0
Date of Change	N/A

1.2 Purpose and scope

1.2.1 Purpose

This document aims to describe the Integration Testing Plan (ITP) for the components of MyTaxiService application, which have already been described in the Design Document (DD). The ITP is written in order to state in which sequence the tests have to take place, and how exactly each single interface between a pair of components has to be tested before actually proceeding with the integration.

1.2.2 Scope

The ITP presented in this document refers to the integration of the whole system described in the DD, starting from the integration of the internal components of the various subsystems of the application and ending with the integration of the subsystem themselves, to complete the composition of the system's full structure.

1.3 List of Definitions and Abbreviations

- **DD (Design Document)**: the already delivered Design Document of MyTaxiService application, providing instructions about the implementation structure of the system.
- **Driver Client:** the client application for Driver Users (the taxi drivers), installed on each taxi terminal.
- **Driver Object:** an instance of a Driver Object, the ones used in the Drivers Model to store frequently accessed informations about currently operating taxi drivers (i.e. logged Driver Users).
- **Driver User:** user account associated to a taxi driver, can be also used to directly refer to a currently operating taxi driver (e.g. the driver user switches his state to "unavailable").
- **Passenger Client:** the client application for Passenger Users, available either as a mobile app or from web browser.
- **Passenger User:** user account associated to a person who uses MyTaxiService client application for passengers. Can also be used to directly refer to the physical person using the passenger application (e.g. the passenger user sends a ride request).

- **Request:** request object representing either a ride or a reservation request, either pending, waiting for a driver user ready to take care of it, or associated to the driver user (and his correspondent Driver object) who is currently dealing with it.

1.4 List of Reference Documents

- Project Description Document
- Assignment Description Document
- Requirements Analysis and Specification Document of MyTaxiService application, delivered on 06/11/2015
- Design Document of MyTaxiService application, delivered on 04/12/2015
- Integration Test Plan Example

2 Integration Strategy

2.1 Entry Criteria

Here is a list of the criteria that each specific component has to meet before actually being integration tested:

- The description of the components involved in the integration test and the descriptions of their interfaces, both contained in the DD, must have been released
- The components involved must have been already implemented
- The classes and functions of the component being integrated must have been successfully individually tested
- The drivers needed to perform the integration test must have been created
- The input data necessary for the test must have been designed and produced

2.2 Elements to be integrated

The integration testing must be performed for all the components that are part of the logic, presentation and data tiers of MyTaxiService system, excluding the database itself and the web server (which is part of the web tier). This means that all the components of both the server and the two different kind of client applications (driver and passenger) will have to undergo the tests in order to be integrated.

2.3 Integration Testing Strategy

Considering that the server is the only complex components of the system (see bottom part of section 2.4.1 for further clarifications), the main task of the integration, apart from the obvious need to integrate the clients and the server itself, concerns the integration of the server's internal components. To accomplish this task, we have chosen to follow a bottom-up integration testing strategy, which comes natural and convenient for at least two main reasons:

- having developed the application following the MVC pattern, it comes easy to integrate controller components starting from the ones that merely interact with the local model and the database (like the Queue Manager, or the Driver Manager), to later take care of integrating the more complex controllers, that are interacting with other controllers, when these last ones are already part of the tested and integrated part of the system. Starting integration from the simplest components allows to spot and solve potential bugs at a low level, instead of risking to have to track down malfunctions found in more complex components that are due to lower-level issues, which just propagated their consequences along the way from bottom to up.

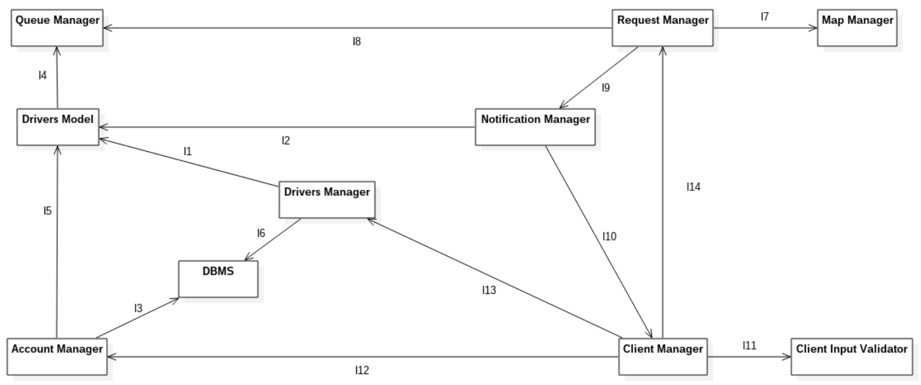
- exploiting the design choice that, due to their high degree of specialization, lower level controllers aren't directly interacting with each other, we can carry forward multiple integration tests in parallel, involving different pairs of components, especially in the early phase of the process. This leads to building up various "component islands" at the same time, to be later connected together.

Then, after the server internal components have been integrated to become a whole, as last phase of the integration process we will perform the integration between with the clients.

2.4 Sequence of Component/Function Integration

2.4.1 Software Integration Sequence

Here is a diagram presenting the general order in which the integration process (and thus the actual integrations too) between the server components is supposed to be carried on. Keep present, however, that it is not a strictly sequential process: as described next, some integration tests can be performed in parallel one with each other.



Here we provide a table containing, for each integration test to be performed among the server components, which test(s) and corresponding integrations must be completed in order to make it possible to start this new integration. This constraints are to be read as transitive relations of course: if Ix is preceded by Iy and Iy is preceded by Iz, Ix is preceded by Iz too.

I1	-
I2	-
I3	-
I4	I1
I5	I3
I6	-
I7	-
I8	I4, I7
I9	I8
I10	I9
I11	I8
I12	I11
I13	I11
I14	I11

Integration Precedences Table

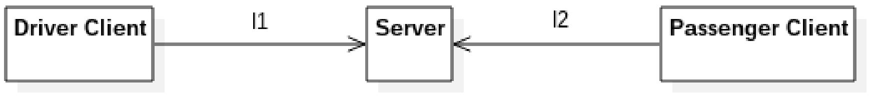
Looking at the table it is immediately clear that, to exploit the potential of the bottom-up integration policy, various integration tests are allowed and supposed to be performed in parallel, especially during the early phase. Then, through the integrations of the controller components that are interacting with each other, the different "component islands" will be integrated together.

For what concerns the clients, as described in the DD they have a very simple and thin structure, almost exclusively composed by the presentation layer of the application, and are thus considered as whole components; this means that there is no need of internal integration tests for them (however the mandatory unit testing, prior to the integration phase, will of course perform all the necessary evaluations and debugging of the client applications).

2.4.2 Subsystem Integration Sequence

About the integration between the subsystems, which are basically the two different types of clients and the server, our intent is to perform it after all the components of the server have been integrated with each other, so that its internal structure has become a whole working component. As already stated, since the structure of the clients is very simple, they have been considered as composed by a unique component, and so they are immediately ready for the integration with the server.

We will first operate the integration testing for the Driver Client, and then the one for the Passenger Client, as the last step of this entire process.



3 Individual Steps and Test Description

In this section we present, for each of the integration tests to be performed, which functions of the exposed interface of the component being integrated are going to be included in the test. For each of these function tests, we state the main goal of the test, the environmental needs (i.e. which drivers/stubs have to be developed to perform the test) and give a brief description of the requested inputs and expected outputs of the test. Remember that for each specific integration test, the preceding integrations that are considered mandatory to be able to perform that new integration are to be intended as implied environmental needs, according to the table presented in section 2.4.1, and won't be rewritten here for each single test.

Server Components Integration

3.1 Driver Manager -> Divers Model Integration [I1]

3.1.1 [I1T1] updateDriverPosition() Integration Test Case

Test Goal	Driver Manager -> Drivers Model Test that the positions of the currently active driver users, i.e. the ones that are present in the Drivers Model as Driver objects, is correctly updated by calling the method
Input Conditions	A list of Driver objects whose position must be updated
Output Conditions	Check that the position of each Driver is correctly updated by updateDriverPosition(), according to the Coordinate returned by the "getDriverPosition()" method
Environmental Needs	Client Manager stub that implements at least the getDriverPosition() method

3.1.2 [I1T2] completeRequest() Integration Test Case

Test Goal	Driver Manager -> Drivers Model Test that DriverManager's completeRequest() method, called by the Client Manager, correctly modifies the status of the Driver object associated to the driver who completed the request
Input Conditions	A Driver object with a Request associated to it, not yet completed

Output Conditions	● Check that the status of the Request is set to “completed” ● Checks that the Driver object is disassociated from the completed Request ● Check that the Availability Status of the Driver is set to “available” ● Check that the value of the Request’s field “missingClient” has been correctly set according to the value provided by the caller
Environmental Needs	● A test driver of the Client Manager that calls Driver Manager’s completeRequest() method ● A test stub that implements the DBMS method storeRequestOnPassenger()

3.1.3 [I1T3] updateDriverStatus() Integration Test Case

Test Goal	Driver Manager -> Drivers Model Test that the Driver Manager’s updateDriverStatus() method correctly updates the target Driver object’s Availability status
Input Conditions	A Driver object whose status has to be updated
Output Conditions	Checks that the Driver object’s Availability status is updated accordingly to the value provided by the caller (either “true” or “false”, meaning “available” or “not available”)
Environmental Needs	A test driver of the Client Manager that calls Driver Manager’s updateDriverSatus() method

3.2 Notification Manager -> Drivers Model Integration [I2]

3.2.1 [I2T1] sendRideRequestToDriver() Integration Test Case

Test Goal	Notification Manager -> Drivers Model Check that the Notification Manager correctly modifies the Busyness flag of a Driver object when the corresponding driver is forwarded a ride request
Input Conditions	A Driver object with the Busyness flag set to false and without a Request of any kind currently associated to him
Output Conditions	● Check that, in any case, the Busyness flag becomes set to true just before forwarding the request and to false after the driver's response is received ● Check that the Driver is set to "not available" and the accepted Request is associated to him only when sendAcceptRefuseNotification() method of the Client Manager returns a positive response
Environmental Needs	● A test driver of the Request Manager that calls Notification Manager's sendRideRequestToDriver() method ● A test stub of the Client Manager component that has to be called with sendAcceptRefuseNotification() method, and return either "true" or "false"

3.3 Account Manager -> DBMS Integration [I3]

3.3.1 [I3T1] getAccountInformations() Integration Test Case

Test Goal	Account Manager -> DBMS Test that the DBMS, by properly querying the database, correctly answers the Account Manager during login verification
Input Conditions	● A valid pair of username and password to be compared to the existing copy stored in the database ● A pair of username and password

	with a username that doesn't exist in the database
Output Conditions	<ul><li>Check that the method correctly returns the password associated with the provided username if the username exists</li><li>Check that the method returns null if the provided username doesn't exist in the database</li></ul>
Environmental Needs	A test driver that calls the Account Manager's login() method

3.3.2 [I3T2] getReservationHistory() Integration Test Case

Test Goal	Account Manager -> DBMS Test that the correct informations are retrieved by the DBMS
Input Conditions	An existing Passenger Account, whose information are stored in the database, with a non-empty list of reservations
Output Conditions	Check that the method correctly queries the database to retrieve the list of all the Reservations associated with the provided PassengerAccount
Environmental Needs	A test driver of the Client Manager that calls the Account Manager's getReservationHistory() method

3.3.3 [I3T3] deleteReservation() Integration Test Case

Test Goal	Account Manager -> DBMS Test that the DBMS correctly modifies the information stored in the database
Input Conditions	An existing Passenger Account, whose informations are stored in the database, and a Reservation associated to it
Output Conditions	<ul><li>Check that the provided Reservation has been removed from those associated with the PassengerAccount</li><li>Check that the other reservations' data has not been changed</li></ul>
Environmental Needs	A test driver of the Client Manager that calls the Account Manager's deleteReservation()

	method

3.3.4 [I3T4] changePassword() and changeEmail() Interaction Test Case

Test Goal	Account Manager -> DBMS Tests that the DBMS correctly modifies the information stored in the database
Input Conditions	An existing Account of any kind, whose informations are stored in the database, his current password, and the new password (or new email address)
Output Conditions	Check that the new password (or email address) is correctly stored in the database, overwriting the old one, but only if the method was called providing the correct current password, to authorize the operation
Environmental Needs	A test driver of the Client Manager that calls the Account Manager's changePassword() and changeEmail() methods

3.3.5 [I3T5] createAccount() Integration Test Case

Test Goal	AccountManager -> DBMS Test that the DBMS correctly creates a new account of the proper type with the provided informations
Input Conditions	● A valid username (strictly matching with a Driver ID in the database if a Driver Account is being created) ● A valid password ● An existing email address
Output Conditions	Check that a new account has been added to the database with the provided informations (email, password, etc...) only if there wasn't any account that already used the provided username or email address
Environmental Needs	A test driver of the Client Manager that calls AccountManager's createPassengerAccount(), createDriverAccount() and createdAdministratorAccount() methods

3.3.6 [I3T6] deleteAccount() Integration Test Case

Test Goal	AccountManager -> DBMS Test that the DBMS correctly deletes the provided Account
Input Conditions	An existing Account object, to be deleted from the database
Output Conditions	Check that the provided account is deleted from the list of registered users in the database
Environmental Needs	A test driver of the Client Manager that calls AccountManager’s deleteAccount() method

3.4 Model -> Queue Manager Integration [I4]

3.4.1 [I4T1] notifyChangedStatus() Integration Test Case

Test Goal	Drivers Model -> Queue Manager Test that the Queue Manager correctly observes (w.r.t. the observer pattern relationship) the Availability status field of the Driver objects of the Drivers Model
Input Conditions	● A Driver object whose position is inside a certain CityZone ● The observer relationship between the QueueManager component and the Driver component has been established
Output Conditions	● Check that the Queue Manager component is notified by calling the notifyChangedStatus() method, sending as parameter the modified Driver object ● Check that the Queue Manager correctly updates the queue of available drivers related to the CityZone where the Driver is currently in (by adding or removing it depending on the status change)
Environmental Needs	-

3.4.2 [I4T2] notifyNotBusy() Integration Test Case

Test Goal	Drivers Model -> Queue Manager Test that the Queue Manager correctly observes (w.r.t. the observer pattern relationship) the Busyness flag of the Driver objects of the Drivers Model
Input Conditions	● A Driver object whose Busyness field has to be updated ● The observer relationship between the QueueManager component and the Driver component has been established
Output Conditions	● Check that the Queue Manager component is notified by calling the notifyNotBusy() method whenever the field changes from “false” to “true” sending as parameter the modified Driver object ● Check that the Queue Manager component correctly adds the Driver to the queue of available drivers corresponding to the zone where the driver is currently in
Environmental Needs	-

3.4.3 [I4T3] notifyChangedZone() Integration Test Case

Test Goal	Drivers Model -> Queue Manager Test that the Queue Manager correctly observes (w.r.t. the observer pattern relationship) the CityZone field of the Driver objects of the Drivers Model
Input Conditions	● A Driver object whose “cityZone” field has to be updated ● The observer relationship between the Queue Manager component and the Driver component has been established
Output Conditions	● Checks that the Queue Manager component is notified by calling its notifyChangedZone() method whenever the “cityZone” field changes, and the modified Driver is passed as parameter ● Checks that, if the Driver’s Availability status is set on “true”, the QueueManager component

	correctly moves the Driver from the old queue to the new one, corresponding to the new city zone in which the driver is currently in
Environmental Needs	-

3.5 Account Manager -> Drivers Model Integration [I5]

3.5.1 [I5T1] createDriverObject() Integration Test Case

Test Goal	Account Manager -> Drivers Model Test that the Account Manager's login() method correctly creates a Driver object (corresponding to the logged in driver user) in the Drivers Model whenever a driver user successfully logs in, through the createDriverObject() method of Drivers Model
Input Conditions	A valid pair of username and password associated to an existing Driver user
Output Conditions	● Check that a Driver Object with Driver ID and Taxicab ID corresponding to the ones of the logged in driver user has been created in the Drivers Model ● Check that the created object has its Availability Status set to "true", its Availability Timestamp set to at most 5 seconds ago, its Busyness Flag set to "false", and the Current Ride set to null
Environmental Needs	A test driver of the Client Manager component that calls Account Manager's login() method

3.5.2 [I5T2] deleteDriverObject() Integration Test Case

Test Goal	Account Manager -> Drivers Model Test that the Account Manager’s logout() method correctly removes the Driver object corresponding to the driver user who logged out from the Drivers Model whenever a driver user logs out, through the deleteDriverObject() method of Drivers Model
Input Conditions	The Driver user for which we are simulating the logout has to actually be present in the central database
Output Conditions	Ensure that the Driver Object with Driver ID and Taxicab ID corresponding to the ones of the logged out driver user is removed from the Drivers Model, if there was one
Environmental Needs	A test driver of the Client Manager component that calls Account Manager’s logout() method

3.6 Driver Manager -> DBMS Integration [I6]

3.6.1 [I6T1] storeRequestOnPassenger() Integration Test Case

Test Goal	Driver Manager -> DBMS Tests that after a request is successfully completed, it is also stored in the database
Input Conditions	A request that is going to be completed
Output Conditions	Checks that the correct information about the completed request has been stored in the database
Environmental Needs	A test driver of the Client Manager that calls Driver Manager’s completeRequest() method

3.7 Request Manager -> Map Manager Integration [I7]

3.7.1 [I7T1] getCityZoneFromCoordinate() Integration Test Case

Test Goal	Request Manager -> Map Manager Tests that the Request Manager component correctly interacts with the Map Manager for retrieving the right zone
Input Conditions	A Coordinate inside a certain City Zone
Output Conditions	Checks that addNewRequest() method correctly calls the Map Manager's getCityZoneFromCoordinate() method
Environmental Needs	A test driver of the Client Manager that calls the Request Manager's addNewRequest() method

3.7.2 [I7T2] estimateWaitingTime() Integration Test Case

Test Goal	Request Manager -> Map Manager Tests that the Request Manager component correctly interacts with the MapManger component to estimate the waiting time
Input Conditions	A Passenger object and a Coordinate inside a certain City Zone
Output Conditions	● Checks that if the sendRideRequestToDriver() method returns a positive answer, the method calls estimateWaitingTime() method to retrieve the waiting time associated with the correct driver
Environmental Needs	● A test driver of the Client Manager that calls the Request Manager's addNewRequest() method ● A test stub of the QueueManager component that has to be called with gerFirstAvailableDriver() method

3.8 Request Manager -> Queue Manager Integration [I8]

3.8.1 [I8T1] getFirstAvailableDriver() Reservation Integration Test Case

Test Goal	Request Manager -> Queue Manager Test that the Request Manager, handling a Reservation request, correctly calls the Queue Manager to retrieve the correct driver
Input Conditions	A Reservation, scheduled by the Request Manager component, has to be performed at a given date
Output Conditions	● Checks that 10 minutes before the Reservation has to be performed, the Request Manager calls getFirstAvailableDriver() method of the Queue Manager component, using as input parameter the cityZone associated to the impending Reservation
Environmental Needs	-

3.8.2 [I8T2] getFirstAvailableDriver() Ride Integration Test Case

Test Goal	Request Manager -> Queue Manager Test that the Request Manager component, when a new request has been received, correctly calls the getFirstAvailableDriver() method
Input Conditions	A Ride Request to be handled
Output Conditions	Check that the method calls the getFirstAvailableDriver() method with the Ride's cityZone as input parameter, and the first available driver in the given zone is correctly retrieved
Environmental Needs	A test driver of the Client Manager that calls Request Manager's addNewRide() method invoking the driver allocation algorithm

3.9 Request Manager -> Notification Manager Integration [I9]

3.9.1 [I9T1] confirmedValidReservationToPassenger() Integration Test Case

Test Goal	Request Manager -> Notification Manager Test that the method is called with the proper parameters, and correctly builds and forwards the notification to confirm a passenger who successfully completed a reservation request that his ride has been reserved
Input Conditions	A well formed reservation request is received by the Request Manager
Output Conditions	Check that the notification containing the reservation date, time and location is correctly built and forwarded to the Client Manager, destinated to the passenger who made the reservation
Environmental Needs	● A test driver of the Client Manager component calling the addNewReservation() method of the Request Manager ● A test stub of the Client Manager component to be called by the Notification Manager with the sendInformativeNotification() method after the notification is built

3.9.2 [I9T2] sendRideRequestToDriver() Integration Test Case

Test Goal	Request Manager -> Notification Manager Test that the method is called with the proper parameters, and correctly builds and forwards the notification to ask the driver if he wants to take care or not of the pending ride request
Input Conditions	A getFirstAvailableDriver() call to the Queue Manager during the handleRideRequest() routine of the Request Manager returns an available driver
Output Conditions	Check that the notification asking the driver if he is ready to take care of the request, containing the ride's starting location and

	the username of the passenger, is correctly built and forwarded to the Client Manager, and that the response ("true" or "false", meaning request accepted or refused) is correctly received by the Request Manager
Environmental Needs	A test stub of the Client Manager component to be called by the Notification Manager with the sendAcceptRefuseNotification() method after the notification is built, that returns either true or false

3.9.3 [I9T3] sendReservationRequestToDriver() Integration Test Case

Test Goal	Request Manager -> Notification Manager Test that the method is called with the proper parameters, and correctly builds and forwards the notification to the driver who has to take care of a reserved ride
Input Conditions	A getFirstAvailableDriver() call to the Queue Manager during the handleScheduledReservation() routine of the Request Manager returns an available driver
Output Conditions	Check that the notification containing the reservation place and time and the passenger's username is correctly built, and forwarded to the Client Manager, destinated to the driver who has to take care of the ride
Environmental Needs	A test stub of the Client Manager component to be called by the Notification Manager with the sendInformativeNotification() method after the notification is built

3.9.4 [I9T4] acceptedRequestToPassenger() Integration Test Case

Test Goal	Request Manager -> Notification Manager Test that the method is called with the proper parameters, and correctly builds and forwards the notification to confirm a passenger that his ride request has been accepted, as soon as a driver accepts it

Input Conditions	A sendRideRequestToDriver() call during the handleRideRequest() routine of the Request Manager returns a positive answer
Output Conditions	Check that the notification containing the the taxicab ID associated to the driver who took care of the request and the estimated waiting time computed through the Map manager is correctly built, and forwarded to the Client Manager, destinated to the passenger who made the request
Environmental Needs	A test stub of the Client Manager component to be called by the Notification Manager with the sendInformativeNotification() method after the notification is built

3.9.5 [I9T5] remindReservationToPassenger() Integration Test Case

Test Goal	Request Manager -> Notification Manager Test that the method is called with the proper parameters, and correctly builds and forwards the notification to tell the passenger the incoming taxi ID, as soon as the reserved ride is assigned to a driver
Input Conditions	A getFirstAvailableDriver() call to the Queue Manager during the handleScheduledReservation() routine of the Request Manager returned an available driver
Output Conditions	Check that the notification containing the reservation place and time and the taxicab ID associated to the driver who took care of the request, destinated to the passenger who made the reservation, is correctly built and forwarded to the Client Manager
Environmental Needs	A test stub of the Client Manager component to be called by the Notification Manager with the sendInformativeNotification() method after the notification is built

3.9.6 [I9T6] queuedRequestToPassenger() Integration Test Case

Test Goal	Request Manager -> Notification Manager Test that the method is called with the proper parameters, and correctly builds and forwards the notification to ask the passenger who just made a ride request if he still wants to confirm it or not after it has been queued
Input Conditions	A getFirstAvailableDriver() call to the Queue Manager during the handleRideRequest() routine of the Request Manager returns no driver
Output Conditions	Check that the notification asking the passenger what he wants to do is correctly built and forwarded to the Client Manager, and that the response ("true" or "false", meaning confirm the request or cancel it) is correctly received by the Request Manager
Environmental Needs	A test stub of the Client Manager component to be called by the Notification Manager with the sendAcceptRefuseNotification() method after the notification is built, that returns either true or false

3.10 Notification Manager -> Client Manager Integration [I10]

3.10.1 [I10T1] sendAcceptRefuseNotification() Integration Test Case

Test Goal	Notification Manager -> Client Manager Test that the Notification Manager correctly interact with the Client Manager to forward messages to the client and to wait for a response
Input Conditions	A Ride and a Reservation request to build the notifications
Output Conditions	● Check that the Notification Manager's sendRideRequestToDriver(), sendReservationRequestToDriver() and queuedRequestToPassenger() methods call the Client Manager's sendAcceptRefuseNotification()

	method with a proper notification object that contains the correct information about the request ● Check that the sendAcceptRefuseNotification() method waits an answer from the Client Manager and returns it as soon as it is received from the client
Environmental Needs	-

3.10.2 [I10T2] sendInformativeNotification() Integration Test Case

Test Goal	Notification Manager -> Client Manager Test that the Notification Manager correctly interact with the Client Manager to forward messages to the client
Input Conditions	A Reservation request to build the notifications
Output Conditions	Check that the Notification Manager's confirmedValidReservationToPassenger() and remindReservationToPassenger() methods calls the Client Manager's sendInformativeNotification with a proper notification object
Environmental Needs	-

3.11 Client Manager -> Cient Input Validator Integration [I11]

3.11.1 [I11T1] validateReservationDate() Integration Test Case

Test Goal	Client Manager -> Client Input Validator
Input Conditions	A Reservation request is sent by a Passenger client
Output Conditions	● Check that the Client Manager's passengerRequestReservation() method correctly calls the Client Input Validator method to check that the reservation date is feasible ● Check that the reservation in forwarded to the Request Manager only if the Client Input Validator has

	returned a positive answer
Environmental Needs	A test driver of the Passenger Client that calls ClientManager's passengerRequestReservation() method

3.11.2 [I11T2] validateReservationDeletion() Integration Test Case

Test Goal	Client Manager -> Client Input Validator
Input Conditions	A Reservation deletion request is sent by a Passenger client
Output Conditions	● Check that the Client Manager's passengerDeleteReservation() method correctly calls the Client Input Validator method to check that the user can actually delete the provided reservation ● Check that the computation is forwarded to the Account Manager only if the Client Input Validator has returned a positive answer
Environmental Needs	A test driver of the Passenger Client that calls ClientManager's passengerDeleteReservation() method

3.11.3 [I11T3] validateCompleteRequest() Integration Test Case

Test Goal	Client Manager -> Client Input Validator Tests that the Client Manager component, before forwarding the completion request to the Driver Manager component, verifies that the request is actually feasible
Input Conditions	A Driver client is communicating that he has terminated current ride
Output Conditions	● Checks that driverCompleteRide() method correctly calls the Client Input Validator's validateCompleteRequest() to check that the driver that has requested to complete the ride actually has a Request associated to him that can be completed ● Checks that only if the above criteria is met the action request is

	forwarded to the Driver manager component, in order to be actually completed
Environmental Needs	A test driver of the Driver Client that calls ClientManager’s driverCompleteRide() method

3.11.4 [I11T4] validateChangedStatus() Integration Test Case

Test Goal	Client Manager -> Client Input Validator Tests that the Client Manager component, before forwarding the computation to the Driver Manager, verifies that request is actually feasible
Input Conditions	A Driver client is changing his Availability status
Output Conditions	● Checks that driverChangeStatus() method correctly calls the Client Input Validator’s validateChangedStatus() method to verify that the driver can actually perform the requested status change ● Checks that only if Cient Input Validator has returned a positive answer the action request is forwarded to the Driver Manager
Environmental Needs	A test driver of the Driver Client that calls ClientManager’s driverChangeStatus() method

3.11.5 [I11T5] validateAddress() Integration Test Case

Test Goal	Client manager -> Client Input Validator Tests that when a request for a ride or a reservation is received, the Client Manager component verifies that provided addresses feasible
Input Conditions	A Ride or Reservation request is received from a Passenger client
Output Conditions	● Checks that the method correctly interacts with the Client Input Validator to verify that the provided address/addresses are inside the zone in which the service is available and, in the case of reservation, that the “from” and “to” addresses are

	different ● Checks that only those requests that meet the above criteria are accepted and forwarded to the Request Manager component
Environmental Needs	A test driver of the Passenger Client that calls ClientManager's passengerRequestReservation() and passengerRequestRide() methods

3.11.6 [I11T6] validatePassword() Integration Test Case

Test Goal	Client Manager -> Client Input Validator Checks that Client Manager correctly interacts with the Client Input Validator component to verify that the provided password meets all the security parameters
Input Conditions	A password that meets or not all the possible combinations of requirements listed below
Output Conditions	● Checks that the register() method correctly calls Client Input Validator's validatePassword() method ● Checks that only passwords with at least 8 characters, at least one digit and a non alphanumeric character are accepted and forwarded to the Account Manager component
Environmental Needs	A test driver of the Passenger Client that calls ClientManager's register() method

3.11.7 [I11T7] validateEmail() Integration Test Case

Test Goal	Client Manager -> Client Input Validator Checks that Client Manager correctly interacts with the Client Input Validator component to verify that the provided email is well formed
Input Conditions	An email that meets or not the required mail structure
Output Conditions	● Checks that the register() correctly calls Cient Input Validator's vaidateEmail() method ● Checks that only well formed emails are accepted and forwarded to the

	Account Manager component
Environmental Needs	• A test driver of the Passenger Client that calls ClientManager’s register() method

3.12 Client Manager -> Account Manager Integration [I12]

3.12.1 [I12T1] login() Integration Test Case

Test Goal	Client Manager -> Account Manager Test that the Client Manager component correctly forwards the computation to the Account Manager
Input Conditions	A pair of username and password
Output Conditions	• Check that the method correctly interacts with the Account Manager component by calling its login() method using the provided credentials as arguments • Check that the correct value is returned by the Client Manager, according to what is returned by the Account Manager
Environmental Needs	• A test driver of the Passenger client (or the client itself) that calls the Client Manager’s login() method

3.12.2 [I12T2] getReservationHistory() Integration Test Case

Test Goal	Client Manager -> Account Manager Test that the Client Manager component correctly forwards the computation to the Account Manager
Input Conditions	The Passenger client must be logged into the system as a Passenger user
Output Conditions	• Check that the method correctly interacts with the Account Manager component by calling its getReservationHistory() method using the provided Passenger Account as argument • Check that the list of reservation returned by the Account Manager

	component is actually returned by the Client Manager method
Environmental Needs	● A test driver of the Passenger client (or the client itself) that calls Client Manager's passengerReservationHistory() method

3.12.3 [I12T3] deleteReservation() Integration Test Case

Test Goal	Client Manager -> Account Manager Test that the Client Manager component correctly forwards the computation to the Account Manager
Input Conditions	● The Passenger client must be logged into the system as a Passenger user ● A passenger user and a reservation associated to him that has to be deleted
Output Conditions	● Checks that the method correctly interacts with the Account Manager component by calling its deleteReservation() method using the provided user and reservation as arguments
Environmental Needs	● A test driver of the Passenger client (or the client itself) that calls Client Manager's passengerDeleteReservation() method

3.12.4 [I12T4] changePassword() Integration Test Case

Test Goal	Client Manager -> Account Manager Test that the Client Manager component correctly forwards the computation to the Account Manager
Input Conditions	● The client must be logged into the system ● The username, the old password and the new password that has to be set to the user
Output Conditions	● Check that the method correctly interacts with the Account Manager component by calling its

	changePassword() method with the provided username and passwords
Environmental Needs	● A test driver of the Passenger client (or the client itself) that calls Client Manager's changePassword() method

3.12.5 [I12T5] changeEmail() Integration Test Case

Test Goal	Client Manager -> Account Manager Test that the Client Manager component correctly forwards the computation to the Account Manager
Input Conditions	● The Passenger client must be logged into the system as a Passenger user ● The username, the password and the new email address that has to be set to the user
Output Conditions	● Check that the method correctly interacts with the Account Manager component by calling its changeEmail() method with the provided username, password and email address
Environmental Needs	● A test driver of the Passenger client (or the client itself) that calls Client Manager's changeEmail() method

3.12.6 [I12T6] createAccount() Integration Test Case

Test Goal	Client Manager -> Account Manager Test that the Client Manager component correctly forwards the computation to the Account Manager
Input Conditions	● A username, password and email to create the account (notice that guests on the passenger clients can only create Passenger user accounts; Driver and Administrator accounts can be created by Administrator users, that must be logged in on the Passenger client on web browser, as explained in the RASD. Thus, only the proper "version" of this method is exposed to the specific kind of user that is logged in the

	application)
Output Conditions	● Check that the method correctly interacts with the Account Manager component by calling its register() method with the provided information
Environmental Needs	● A test driver of the Passenger client (or the client itself) that calls Client Manager's register() method

3.13 Client Manager -> Driver Manager Integration [I13]

3.13.1 [I13T1] completeRequest() Integration Test Case

Test Goal	Client Manager -> Driver Manager Test that the Client Manager component correctly interacts with the Driver Manager forwarding ride completion requests
Input Conditions	A logged in driver, a request associated to him that has not been completed yet and a value for the missingClient parameter
Output Conditions	Test that Driver Manager's completeRequest() method is called with the provided driver user and missingClient value as arguments
Environmental Needs	A test driver of the Driver client (or the client itself) that calls Driver Manager's driverCompleteRide() method

3.13.2 [I13T2] updateDriverStatus() Integration Test Case

Test Goal	Client Manager -> Driver Manager Test that the Client Manager component correctly interacts with the Driver Manager forwarding status change requests
Input Conditions	A logged in driver and a value for the Availability status to be set
Output Conditions	Test that Driver Manager's updateDriverStatus() is called with the provided driver and status value as argument

Environmental Needs	A test driver of the Driver client (or the client itself) that calls Driver Manager's updateDriverStatus() method

3.14 Client Manager -> Request Manager Integration [I14]

3.14.1 [I14T1] addNewRide() Integration Test Case

Test Goal	Client Manager -> Request Manager Test that the Client Manager component correctly interacts with the Request Manager forwarding the computation of new ride requests
Input Conditions	A Passenger account and the meeting point coordinate
Output Conditions	Test that the Request Manager's addNewRide() method is called with the provided passenger account and coordinate as arguments
Environmental Needs	A test driver of the Passenger client (or the client itself) that calls Driver Manager's passengerRequestRide() method

3.14.2 [I14T2] addNewReservation() Integration Test Case

Test Goal	Client Manager -> Request Manager Tests that the Client Manager component correctly interacts with the Request manager forwarding the computation of new reservations
Input Conditions	A Passenger account and the reservation meeting and arrival location, and the reservation time and date
Output Conditions	Test that the Request Manager's addNewReservation() method is called with the provided reservation information as arguments
Environmental Needs	A test driver of the Passenger client (or the client itself) that calls Driver Manager's passengerRequestRide() method

Subsystems Integration

3.15 Clients-> Server Integrations [I1,I2]

The integration testing between the two types of clients and the server, which consists in verifying the correct method calls mapping between the interface externally exposed by the Client Manager (which is the server component that handles all the interactions with the clients) and the corresponding internal server component calls, is in fact carried on through the last integration of the server components integration (precisely I12,I13 and I14), when the Client Manager itself is being integrated with the rest of the server components. As specified in the tests descriptions in fact, if the Driver and Passenger clients have already been individually tested and are thus ready as wholes to be integrated, they can be used in place of the test programs to perform the method calls to the Client Manager needed for those tests. Those integration tests between the Client Manager and the internal server components are just verifying that methods are properly mapped from outside to inside, with the parameters correctly passed from the clients to the Client Manager and then forwarded to the internal components, and the response returned up to the calling client. Therefore, by testing for the Client Manager integrations using the actual clients instead of test drivers, we are in fact performing the client-server integration testing too: we check that method calls are correctly mapped, transferring the original call parameters, from the client to the internal server component that is the target of the method call, and that the response too is forwarded as it is back to the client, again through the Client Manager. After this, what remains to do to test that the integrated system works is to:

- Test that the integrated server and clients work in their actual environment instead of the testing one, starting from an empty system, filling the server's database step by step: start to create Driver accounts through the administrator, to check that the server correctly handles communications with driver clients alone, and then create Passenger accounts and do the same verification. Then start to run ride and reservations simulations, either through computer simulations or involving actual taxis and passengers, starting from a small number of them, to check that the system as a whole does work in its real environment;

- Run performance tests, e.g. progressively increasing the bunch of clients of the two types connected to the server and the number of requests to be handled and answered, to check that the system responds to the performance requirements stated in the RASD and that the service is finally ready to be launched.

4 Tools and Test Equipment Required

Here is the list of tools that have to be used to perform the integration tests listed in the previous section:

- Mockito
  Version: 1.10.19
  Link: http://site.mockito.org/

  This tool provides a framework that allows developers to create stub objects, to verify that the proper methods have been called with the provided arguments and for the expected number of times. This tool has to be used in all the integration tests listed above in which it is necessary to create test stubs to emulate the behaviour of components of the system and where it is necessary to ensure that certain methods are called only under certain conditions and a precise number of times.

- Arquillian
  Version: 1.1.10.Final
  Link: http://arquillian.org/

  This tool is a framework that allows to perform integration tests in enterprise applications, and it is useful when it is necessary to test components that interact with Java EE objects. It can be used in the integration tests listed above where it is necessary to create test drivers that interact with the components under testing and when it is necessary to ensure that the components correctly interact with remote services and databases.

- Apache JMeter™
  Version: 2.13
  Link: http://jmeter.apache.org/

  This tool is a software that allows to test the performances of a system. It can be used to perform the performance test listed in the previous section in order to test the strength of the system by gradually increasing the number of connected users (both passengers and drivers) and incoming requests.

5 Program Stubs and Test Data Required

Here we present which test drivers and stubs that are needed to be developed for the integration testing. Since in section 3 for each single test it is provided the description of all the needed stubs and drivers, and which function they shall perform, here we will just briefly summarize the situation, listing all the stubs and drivers that are needed along the way; please refer to individual tests of section 3 to better understand what exactly is needed for each test. However, we tried to keep them at the minimum possible, integrating components in an order such that the already integrated part of the system, even if obviously not complete, could be considered as capable of running the needed portion of functions or routines needed in testing the next component's integration.

5.1 Drivers

- **Client Manager** driver, needed in a large part of the internal server components integration tests, since it simulates function calls coming from clients;
- **Request Manager** driver, needed in integration test **I2** to simulate method calls to the Notification Manager component;
- **Passenger Client** driver, needed in integration test **I11** to simulate method calls to the Notification Manager component (but also the actual client could be used, if ready);

5.2 Stubs

- **Client Manager** stub, needed in a large part of the internal server components integration tests, since it simulates the interaction of the components with the external world, i.e. the clients, when they have to receive method calls from the server;
- **DBMS** stub, needed in integration test **I1**, to be called by the Driver Manager;
- **Queue Manager** stub, needed in integration test **I7**, to be called by the Request Manager;

5.3 Test Data Required

The data needed to run the integration tests isn't actually a lot, and depends on the specific test, but it's anyway either informations already owned by the company (e.g. data relative to drivers used to test Driver users registrations) or very simple to produce (e.g. sample ride or reservation requests), and only a few samples fulfilling certain different characteristics and criteria for each test. A bigger amount of data, even if of the same kinds as for the integration testing, will instead be required to run, for example, the performance testing.

6 Appendix

6.1 Hours of work

Here is how long it took to redact this document:

- Matteo Bulloni: ~ 17 hours
- Marco Cannici: ~ 13 hours