Politecnico di Milano

Software Engineering 2

INTEGRATION TEST PLAN

PowerEnjoy

Authors:

Simone Bruzzechesse

Luca Franceschetti

Gian Giacomo Gatti

INDEX

1 INTRODUCTION

1.1 REVISION HISTORY

Version	Date	Author(s)	Description
1.0	29-12-2016	Simone Bruzzechesse, Luca Franceschetti, Gian Giacomo Gatti	Document created
1.1	15-01-2017	Simone Bruzzechesse, Luca Franceschetti, Gian Giacomo Gatti	Document completed

1.2 PURPOSE

This document represents the Integration Test Plan Document (ITPD) for Power Enjoy project, which describes the plans for testing the integration of Power Enjoy project's components. The purpose of this document is to highlight the main aspects regarding the organization of the integration testing activity for all components of our system.

1.3 SCOPE

The Integration Test Plan Document describes the plan for the integration testing, which takes as input software components (described in DD) that have been unit tested, groups them in larger aggregates, tests their interfaces, and delivers as its output the integrated system ready for system testing.

1.4 DEFINITIONS, ACRONYMS, ABBREVIATIONS

- RASD: Requirements Analysis and Specification Document
- DD: Design document
- DBMS: Database Management System
- API: Application Programming Interface
- UI: User interface
- GPS: Global Positioning System
- OS: Operating System

1.5 Reference Documents

- Our RASD document
- Our DD document
- Specification Document: Assignments AA 2016-2017.pdf
- Sample integration test plan document

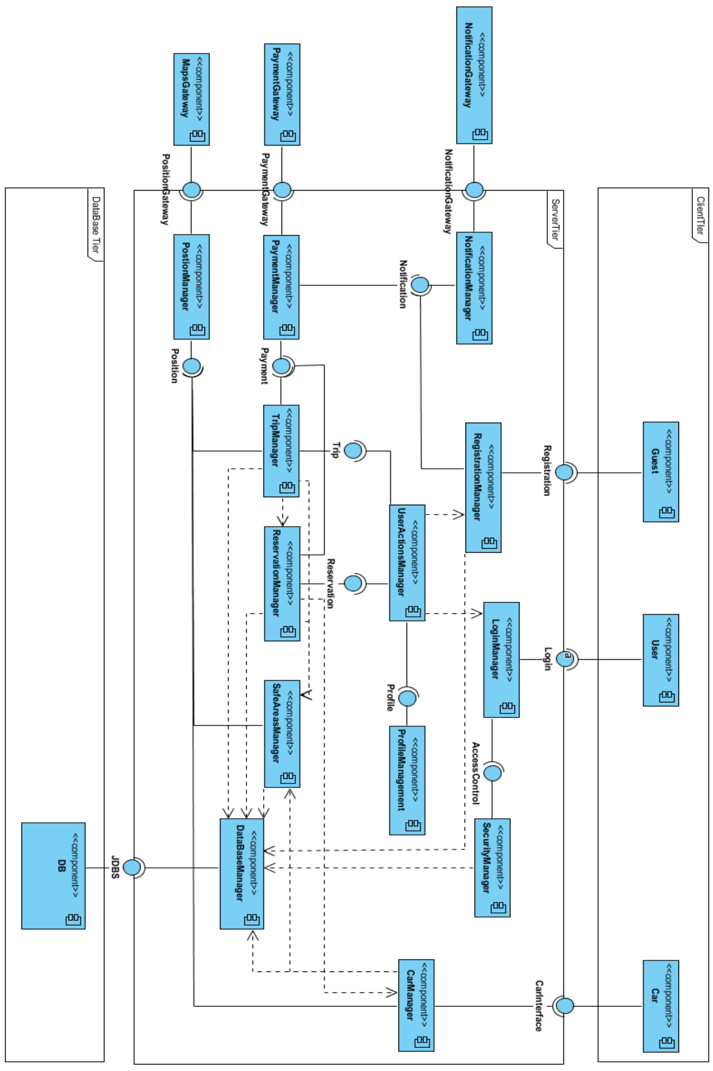
2 Integration Strategy

2.1 Entry Criteria

Before the integration test can begin, the RASD document and the DD document must be completed and successfully delivered. Then, all software components must have been unit tested: this is important because in case of failure we know the problem is in the implementation of interfaces and not in how modules have been developed.

2.2 Elements to be integrated

In this paragraph, we are going to list all components that must be integrated. We report our component diagram (taken from Design Document) for a clearer comprehension of interfaces and main components.



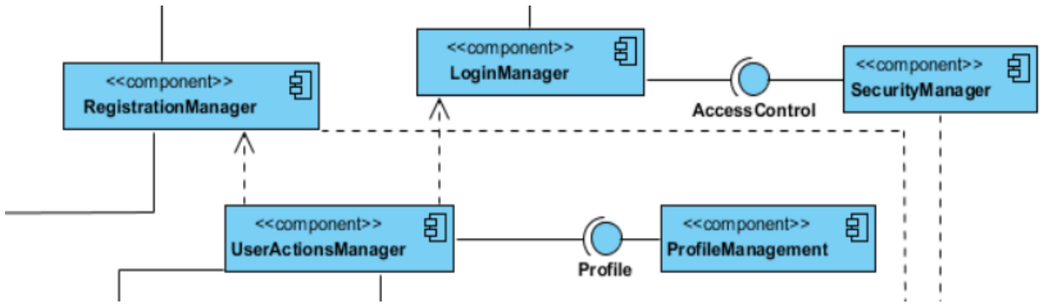
DataBase Tier
ServerTier
ClientTier
<<component>> MapsGateway
<<component>> PaymentGateway
<<component>> NotificationGateway
PositionGateway
PaymentGateway
NotificationGateway
<<component>> PositionManager
<<component>> PaymentManager
<<component>> NotificationManager
Notification
Position
Payment
<<component>> TripManager
Trip
<<component>> RegistrationManager
Registration
<<component>> Guest
<<component>> UserActionsManager
<<component>> ReservationManager
Reservation
<<component>> LoginManager
Login
<<component>> User
Profile
<<component>> SafeAreasManager
<<component>> ProfileManagement
AccessControl
<<component>> DB
JDBS
<<component>> DataBaseManager
<<component>> SecurityManager
<<component>> CarManager
CarInterface
<<component>> Car

2.3 INTEGRATION TESTING STRATEGY

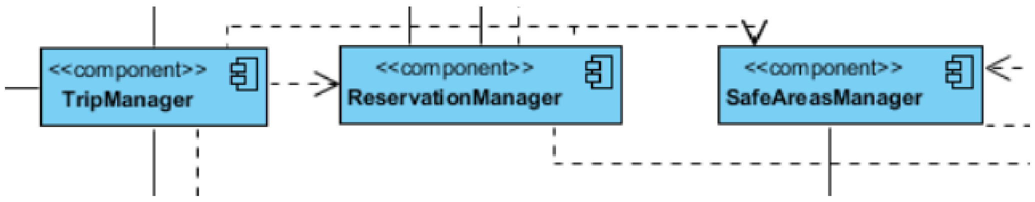
As integration testing strategy, we decided to use a mixture of bottom-up and functional grouping integration strategies. We chose bottom-up approach since we already know the architecture of the software and all components have been implemented and unit-tested, so we group components which do not rely on other components. Then, we decided to adopt the functional grouping where we group components with similar functionalities, so we try to avoid malfunctioning while managing to integrate larger number of components. After grouping these components, we integrate them with other components which are interfaces with external system. Observe that there is no need to test DMBS modules since they are commercial components and they have already been tested from their software house, as well as other external system such as payment system, notification system and maps system.

We identify two main groups:

- User basic functionalities, which includes:
  - RegistrationManager
  - LoginManager
  - SecurityManager
  - UserActionManager
  - ProfileManagement



- Trip management functionalities, which includes:
    - TripManager
    - ReservationManager
    - SafeAreasManager



2.4 Sequence of Component/Function Integration

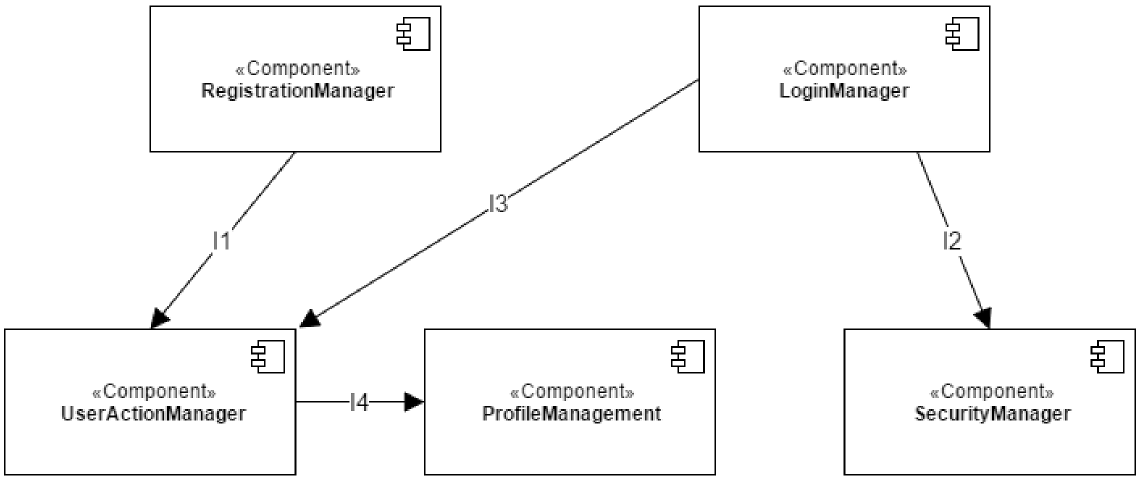
2.4.1 Software Integration Sequence

First of all, we want to integrate components with DatabaseManager, to check if they can correctly store data and they can get properly information from the database. Components that would be integrated with the DatabaseManager are:

- SecurityManager
- RegistrationManager
- CarManager
- TripManager
- ReservationManager
- SafeAreasManager

Then, we want to integrate components between them. Since we adopt functional grouping as integration strategy, we start integrating components which are responsible for offering user's basic functionalities; we are going to call this group of components *User basic functionalities* to identify it and recall it again later in this document. The IDs represents the order in which the integration testing should proceed.

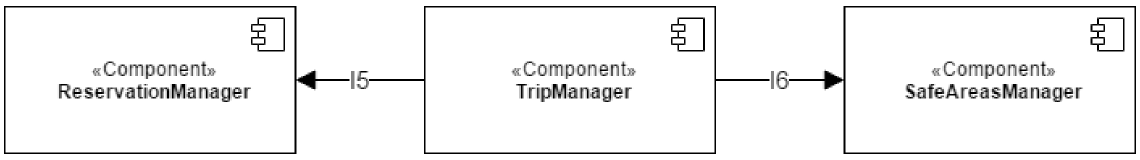
2.4.1.1 User basic functionalities


«Component» RegistrationManager
«Component» LoginManager
I1
I3
I2
«Component» UserActionManager
I4
«Component» ProfileManagement
«Component» SecurityManager

ID	Integration Test	Paragraph
I1	RegistrationManager → UserActionManager	3.1.1
I2	LoginManager → SecurityManager	3.1.2
I3	LoginManager → UserActionManager	3.1.3
I4	UserAction → ProfileManagement	3.1.4

Our second functional grouping regards components that accomplish functionalities to properly manage a trip. We are going to call this group of components *Trip management functionalities* to identify it.

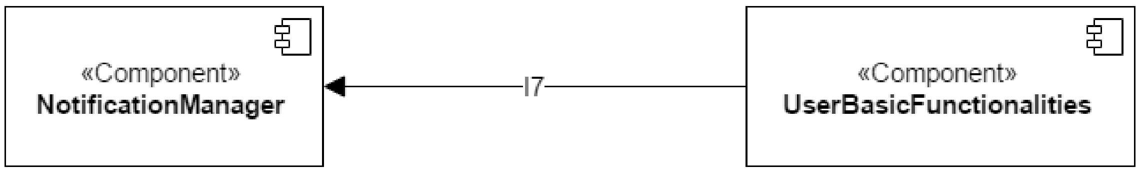
2.4.1.2 Trip management functionalities



ID	Integration Test	Paragraph
I5	TripManager → ReservationManager	3.1.5
I6	TripManager → SafeAreasManager	3.1.6

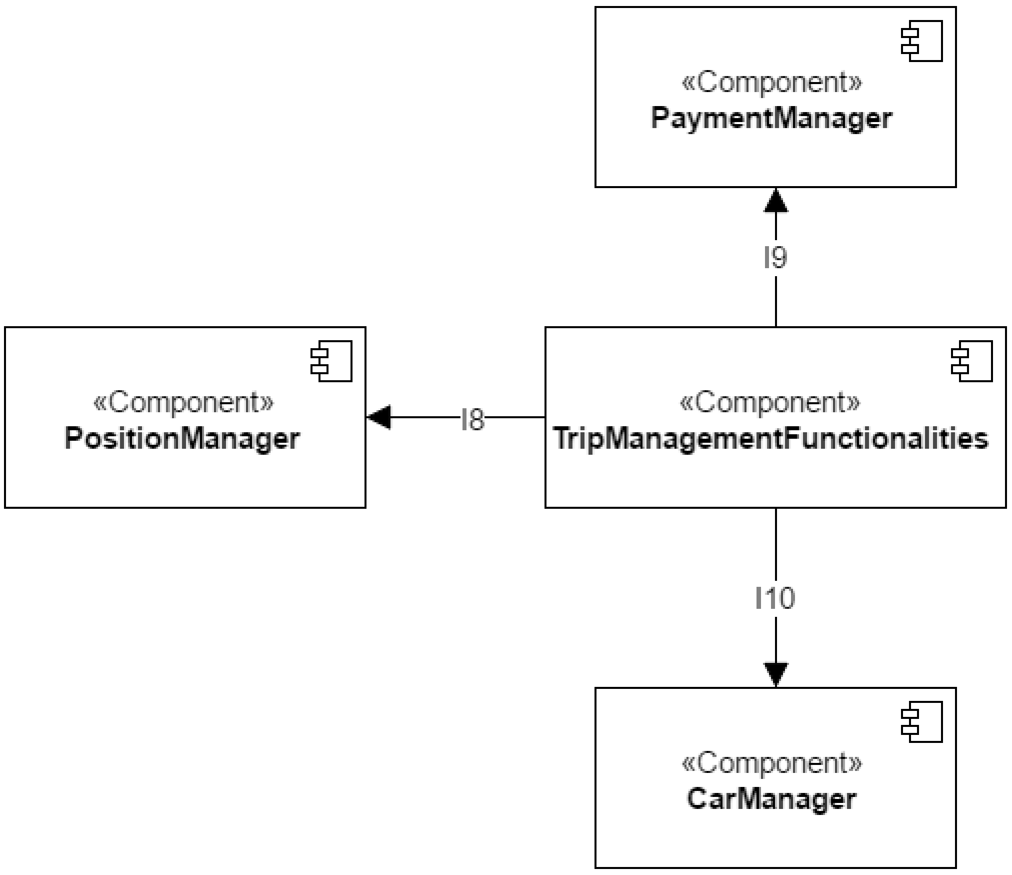
2.4.1.3 External system

Now we integrate these two main groups with other components which work as interfaces for external system.
So, first User basic functionalities: we call the result User subsystem.



ID	Integration Test	Paragraph
I7	UserBasicFunctionalities → NotificationManager	3.1.7

We are going to call this subsystem *User Subsystem.*

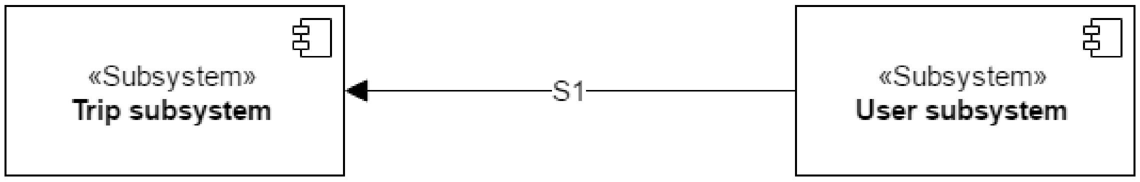


ID	Integration Test	Paragraph
I8	TripManagementFunctionalities→ PositionManager	3.1.8
I9	TripManagementFunctionalities→ PaymentManager	3.1.9
I10	TripManagementFunctionalities→ CarManager	3.1.10

We are going to call this subsystem *Trip Subsystem*.

2.4.2 Subsystem Integration Sequence

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



ID	Integration Test	Paragraph
S1	User subsystem → Trip subsystem	3.1.11

3 Individual Steps and Test Description

We report in this section the steps of the testing.

We report below how to interpret the tables' headers:

- **Test Case Identifier**: identifies the test case, where the code corresponds to the one reported in the diagrams in section 2.4.
- **Tested Items**: identifies the component to be tested in the following format: *Component1, Component 2* where *Component1* calls methods of *Component2*.
- **State of system before test**: Condition and state of system before executing the integration testing.
- **Input Specification**: Input or context that are required to perform integration testing
- **Output Specification**: Output or final context that the integration testing must produce.
- **Environmental needs**: Precondition needed before proceeding with the integration testing of these components.

3.1 Integration Test Cases

3.1.1 Integration Test Case I1

Test Case Identifier	I1T1
Tested Items	RegistrationManager, UserActionManager
State of system before test	Guest is in the home page.
Input specification	Create correct RegistrationManager input
Output specification	Check if new data are correctly stored and guest is redirected to home page, changing his/her status in user. (See User State Diagram at section 3.7 in RASD document)
Environmental needs	-

Test Case Identifier	I1T2
Tested Items	RegistrationManager, UserActionManager
State of system before test	Guest is in the home page.

Input specification	Create wrong RegistrationManager input
Output specification	Check if an error is correctly detected and guest is still at home page and an error message should appear explaining the problem.
Environmental needs	-

3.1.2 Integration Test Case I2

Test Case Identifier	I2T1
Tested Items	LoginManager, SecurityManager
State of system before test	User is in the home page.
Input specification	Create correct LoginManager input
Output specification	Check if inserted data are correct, check if correct security methods are called. Check if user section is correctly created.
Environmental needs	I1T1 – I1T2 succeeded

Test Case Identifier	I2T1
Tested Items	LoginManager, SecurityManager
State of system before test	User is in the home page.
Input specification	Create wrong LoginManager input
Output specification	Check if correct security methods are called. Check if login is refused. Check if user is still at home page and an error message should appear to explain the problem.
Environmental needs	I1T1 – I1T2 succeeded

3.1.3 Integration Test Case I3

Test Case Identifier	I3T1
Tested Items	LoginManager, UserActionManager
State of system before test	User is in the home page.
Input specification	Create correct LoginManager input.
Output specification	Check if correct functions are called in UserActionManager. Check if user is correctly redirected to map view.
Environmental needs	I2T1 – I2T2 succeeded

Test Case Identifier	I3T2
Tested Items	LoginManager, UserActionManager
State of system before test	User is in the home page.
Input specification	Create wrong LoginManager input.
Output specification	Check if correct functions are called in UserActionManager. Check if user is still at home page and an error message should appear explaining the problem.
Environmental needs	I2T1 – I2T2 succeeded

3.1.4 Integration Test Case I4

Test Case Identifier	I4
Tested Items	UserActionManager, ProfileManagement
State of system before test	Logged User is in the map view page.
Input specification	Call method to view personal information
Output specification	Check if correct functions are called, check if information are correctly shown and if they are up to date. Check if user is correctly redirected to personal information page and he/she is able to return to map view.
Environmental needs	I2T1 – I2T2 succeeded

3.1.5 Integration Test Case I5

Test Case Identifier	I5T1
Tested Items	TripManager, ReservationManager
State of system before test	Logged user is in the Car Reserved page. The user made a reservation.
Input specification	Decline reservation
Output specification	Check if the reservation is correctly declined. Check if any fees must be applied and check if car is set "available" again. Check if user is correctly redirected to map view.
Environmental needs	-

Test Case Identifier	I5T2
Tested Items	TripManager, ReservationManager
State of system before test	Logged user is in the Car Reserved page. The user made a reservation.
Input specification	Confirm reservation
Output specification	Check if data are correctly passed to the tablet in the car. Check if functions to open the car are ok. Check if correct functions are called to properly start the trip. Check if user is correctly redirected to Trip Manager page.
Environmental needs	-

Test Case Identifier	I5T3
Tested Items	TripManager, ReservationManager
State of system before test	Logged user is in the Car Reserved page. The user made a reservation.
Input specification	Reservation expired.
Output specification	Check if time is correctly expired. Check if the car is set "available" again. Check if user has no more reservation pending. Check if correct methods to charge penalty fee are called. Check if user is redirected to map view.
Environmental needs	-

3.1.6 Integration Test Case I6

Test Case Identifier	I6T1
Tested Items	TripManager, SafeAreasManager
State of system before test	User is doing the trip and he/she wants to finish the trip.
Input specification	End trip in a safe area.
Output specification	Check if the car is left in a safe area. Check if the car is left in some special area that guarantees extra discount. Check if the car is plugged. Check if the car can correctly close itself. Check if car is et "available" again. Check if there are no more reservation pending on the user. Check if user is correctly redirected to map view.
Environmental needs	I5T2 succeeded

Test Case Identifier	I6T2
Tested Items	TripManager, SafeAreasManager
State of system before test	User is doing the trip and he/she wants to finish the trip.
Input specification	End trip NOT in a safe area.
Output specification	Check if the car is left in a safe area. Check if an error message is shown to the user. Check that car cannot be closed until it's parked in a safe area.
Environmental needs	I5T2 succeeded

3.1.7 Integration Test Case I7

Test Case Identifier	I7T1
Tested Items	UserBasicFunctionalities, NotificationManager
State of system before test	Guest is in the registration page.
Input specification	Correct data are insert to properly complete the registration.
Output specification	Check if a confirmation email is correctly sent to the new user. Check if user is redirected to home page ready to login.
Environmental needs	I1T1 – I1T2 succeeded

Test Case Identifier	I7T2
Tested Items	UserBasicFunctionalities, NotificationManager
State of system before test	Guest is in the registration page.
Input specification	Wrong data are insert so it's not possible to properly complete the registration.
Output specification	Check if NO confirmation email is sent. Check if guest is still in the registration page and an error message should appear explaining the problem.
Environmental needs	-

3.1.8 Integration Test Case I8

Test Case Identifier	I8
Tested Items	TripManagementFunctionalities, PositionManager
State of system before test	User is in Reservation or Trip phase.
Input specification	GPS position is requested.
Output specification	Check if correct position is returned. Check the precision of the position.
Environmental needs	-

3.1.9 Integration Test Case I9

Test Case Identifier	I9T1
Tested Items	TripManagementFunctionalities, PaymentManager
State of system before test	User correctly ended the trip.
Input specification	End trip, exit and pay.
Output specification	Check if the correct amount is calculated, check if correct amount is charged to correct user. Check if user is redirected to map view.
Environmental needs	I6T1 succeeded

Test Case Identifier	I9T2
Tested Items	TripManagementFunctionalities, PaymentManager
State of system before test	User made a reservation but it expired.
Input specification	Time to confirm or decline registration is finished.
Output specification	Check if the correct fee is charged. Check if the conditions for this charging are respected. Check if user is redirected to map view again.
Environmental needs	I5T3 succeeded

3.1.10 Integration Test Case I10

Test Case Identifier	I10
Tested Items	TripManagementFunctionalities, CarManager
State of system before test	User is in the map view and he/she would like to select some cars to check its status and information.
Input specification	Select car.
Output specification	Check if correct data are transmitted and properly shown. Check if user is redirected to car information page.
Environmental needs	-

3.1.11 Integration Test Case S1

Test Case Identifier	S1T1
Tested Items	User subsystem, Trip subsystem
Input specification	Login successful
Output specification	Check if user is redirected to map view. Check if map is centred to user's position. Check if user can select car and correct car information are reported.
Environmental needs	I11 – I8 succeeded

Test Case Identifier	S1T2
Tested Items	User subsystem, Trip subsystem
Input specification	Make reservation.
Output specification	Check if reservation is correctly associated to the user. Check if timer for reservation expiring is correctly set and started.
Environmental needs	-

Test Case Identifier	S1T3
Tested Items	User subsystem, Trip subsystem
Input specification	Confirm reservation.
Output specification	Check if user can properly open the car, checking his/her position with car's one. Check if user is correctly associated to a new trip.
Environmental needs	I5T2 succeeded

Test Case Identifier	S1T4
Tested Items	User subsystem, Trip subsystem
Input specification	Decline reservation.
Output specification	Check if reservation is correctly declined and there are no more pending reservations on related user. Check, since user can make one reservation at the time, if he/she is now able to reserve a car again.
Environmental needs	I5T1 succeeded

Test Case Identifier	S1T5
Tested Items	User subsystem, Trip subsystem
Input specification	Reservation expired.
Output specification	Check if the correct fee is charged to the correct related user. Check if user has no more reservation pending and he/she is able to reserve a car again.
Environmental needs	I5T3

4 PERFORMANCE ANALYSIS

While a full fledged performance analysis of the entire **PowerEnjoy** infrastructure will be executed only in the system integration phase, it is still useful to perform some preliminary measures on components whose performances can be tested in isolation.

4.1 MOBILE PERFORMANCE ANALYSIS

It is appropriate to verify that the applications for all the target mobile platforms have reasonable CPU and main memory usages. Performance requirements of mobile devices are specified in **RASD** document.

Furthermore, even though no strict value is fixed at this point, the storage occupation should be reasonably small in order to guarantee the maximum utilization by the user that not have performant devices.

However, this number should be reconsidered during the development phase considering the improvements in the smartphone and tablet technology that may occur meanwhile. These tests will be performed using the appropriate performance analysis tool provided with the SDK of each mobile platform.

4.2 DESKTOP PERFORMANCE ANALYSIS

Performances of desktop application depend on the browser utilized by the user, so we must develop an application that can be executed on the most common hardware platform and through technologies supported by all the commercial browser.

Indeed, almost every personal computer nowadays has 2GB of RAM and a dual-core CPU. Our application requires much less than that specification because it's a little portion of the system in which the user can register his profile and only modify some information about himself.

These tests will be performed using the appropriate performance analysis tool provided with the SDK of each operating system supported and each browser.

5 Tools and Test Equipment Required

List of tools that will be used to perform integration testing:

- **Manual testing**: there will be an accurate selection of the most crucial functionalities (i.e. functions with exceptional parameters) to be manually tested.
- **JUnit**: although this framework is mainly known for unit tests (indeed we will use it also for unit testing but this is not concerned in this document), it will be also used during the integration phase.
- **Mockito**: this framework will be used to mock stubs and drivers that are needed in the integration tests.

Moreover, **JMeter** will be used to set up tests to analyse the performances of the system. Indeed, we think that it is very useful to use this framework to verify if the non-functional requirements of our system (described in the RASD) are satisfied. With JMeter we will see how the server and the database behave under a heavy load and with a great number of virtual users (simulated with thread group) simultaneously connected.

6 PROGRAM STUBS AND TEST DATA REQUIRED

6.1 PROGRAM STUBS AND DRIVERS

Name	**DBManagerDriver**
Components of reference	DataBaseManager
Purpose	This driver generates various kind of requests, like login request or registration request, and send them to the DataBaseManager. Once response is received, it checks if it is correct and coherent with the request.

Name	**LoginManagerStub**
Components of reference	LoginManager
Purpose	This stub provides a list of possible credential that are provided during login phase. Once the credential is inserted onto the specific fields, SecurityManager checks if they are correct through the "AccessControl" module. This stub is created with the aim of testing this two-specific component.

Name	**SecurityDriver**
Components of reference	SecurityManager
Purpose	This driver checks if the information that comes from LoginManager are correct, before logging in the user eventually. It checks also if the user that trying to logging into the system has only one session opened on his devices.

Name	**PositionStub**
Components of reference	PositionManager, MapsGateway
Purpose	This stub simulates a GPS device and create some position (coordinates) whenever system requires them, to test correctly every component of the system that should manipulate coordinates or positions to calculate distances or prices.

Name	TripManagerDriver
Components of reference	TripManager
Purpose	This component generates various kind of possible situation in which the system must compute some data, during or after the rent. For example, it checks if the display on the car shows always actual cost coherently. It checks also the possible discount/charges at the end of the rent.

Name	PaymentDriver
Components of reference	PaymentGateway, PaymentManager
Purpose	This component generates various payment request through PaymentManger and handle the response that came from PaymentGateway showing to the user the correct message after the successful or unsuccessful payment.

Name	NotificationManagerDriver
Components of reference	NotificationManager, NotificationGateway
Purpose	This driver generates notification that the system, through for example the screen of the car, must send to the user to understand if the notifications are coherent and displaced in real time. The system send a notification through the car's screen to tell if rent is finished, if the car is plug, how much is the total charge, etcetera; so, it's seriously important that this type of communication is dispatched in real time.

Name	CarStub
Components of reference	CarManager
Purpose	This stub replaces the "Car" module when the system must receive some information about car status. Indeed, when we test our system, it's important to understand if the information about car (like battery status or position) are utilized coherently during the rent and if information about passengers (caught by weight sensor on seats) are utilized correctly to calculate the discount after the rent.

Name	ReservationDriver
Components of reference	ReservationManager
Purpose	This driver must manage the reservation and must handle the possible competition during the reservation between different users.

6.2 TEST DATA

To perform the whole set of tests we have defined, we are going to need:

- A list of both valid and invalid drivers to test the **Security Manager** component. We must test also the following problem inside the instances of the set:
    - NULL Object
    - NULL Fields
    - Driver License not valid
    - Driving License expired
    - Driving License inserted incorrectly

- A list of both valid and invalid positions to test the **PositionsManager** component. We must test also the following problem inside the instances of the set:
    - NULL Object
    - NULL Fields
    - External position with respect to the city
    - Wrong coordinates
    - Empty coordinates

- A list of both valid and invalid reservation request to test the **ReservationManager** component. We must test also the following problem inside the instances of the set:
    - NULL Object
    - NULL Fields
    - Users that try multiple reservation
    - Users that try to reserve a car without a complete profile

- A list of both valid and invalid concluded trip to test the **TripManager** component. We must test also the following problem inside the instances of the set:
  - NULL Object
  - NULL Fields
  - Impossible situation like more than 5 passengers
  - Inexistent start/end position
  - Impossible battery level (like less than 0% or more than 100%)
  - Empty fields that are mandatory to calculate the trip costs

- A list of both valid and invalid user's profile to test the **RegistrationManager** component. We must test also the following problem inside the instances of the set:
  - NULL Object
  - NULL Fields
  - Invalid driving license
  - Invalid mobile phone number
  - Invalid payment method
  - Invalid email address
  - Username already used

7 Appendix

7.1 Hours of work

We report approximately how many hours each member has worked on this document.

- Simone Bruzzechesse: 10 hours
- Luca Franceschetti: 10 hours
- Gian Giacomo Gatti: 10 hours