**Politecnico di Milano**

**Software Engineering 2**

**Integration test plan**

**PowerEnjoy**

**Authors:**

**Simone Bruzzechesse**

**Luca Franceschetti**

**Gian Giacomo Gatti**

# Index

[Index 2](#_Toc471123361)

[1 Introduction 3](#_Toc471123362)

[1.1 Revision History 3](#_Toc471123363)

[1.2 Purpose 3](#_Toc471123364)

[1.3 Scope 3](#_Toc471123365)

[1.4 Definitions, Acronyms, Abbreviations 3](#_Toc471123366)

[1.5 Reference Documents 4](#_Toc471123367)

[2 Integration Strategy 5](#_Toc471123368)

[2.1 Entry Criteria 5](#_Toc471123369)

[2.2 Elements to be integrated 5](#_Toc471123370)

[2.3 Integration Testing Strategy 6](#_Toc471123371)

[2.4 Sequence of Component/Function Integration 7](#_Toc471123372)

[2.4.1 Software Integration Sequence 7](#_Toc471123373)

[2.4.2 Subsystem Integration Sequence 9](#_Toc471123374)

[3 Individual Steps and Test Description 10](#_Toc471123375)

[4 Performance analysis 11](#_Toc471123376)

[4.1 Mobile Performance Analysis 11](#_Toc471123377)

[4.2 Desktop Performance Analysis 11](#_Toc471123378)

[5 Tools and Test Equipment Required 12](#_Toc471123379)

[6 Program Stubs and Test Data Required 13](#_Toc471123380)

[6.1 Program Stubs and Drivers 13](#_Toc471123381)

[7 Appendix 15](#_Toc471123382)

[7.1 Used tools 15](#_Toc471123383)

[7.2 Effort spent 15](#_Toc471123384)

# Introduction

## Revision History

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

## 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.

## 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.

## 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
* ETA: Estimated Time of Arrival
* OS: Operating System

## Reference Documents

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

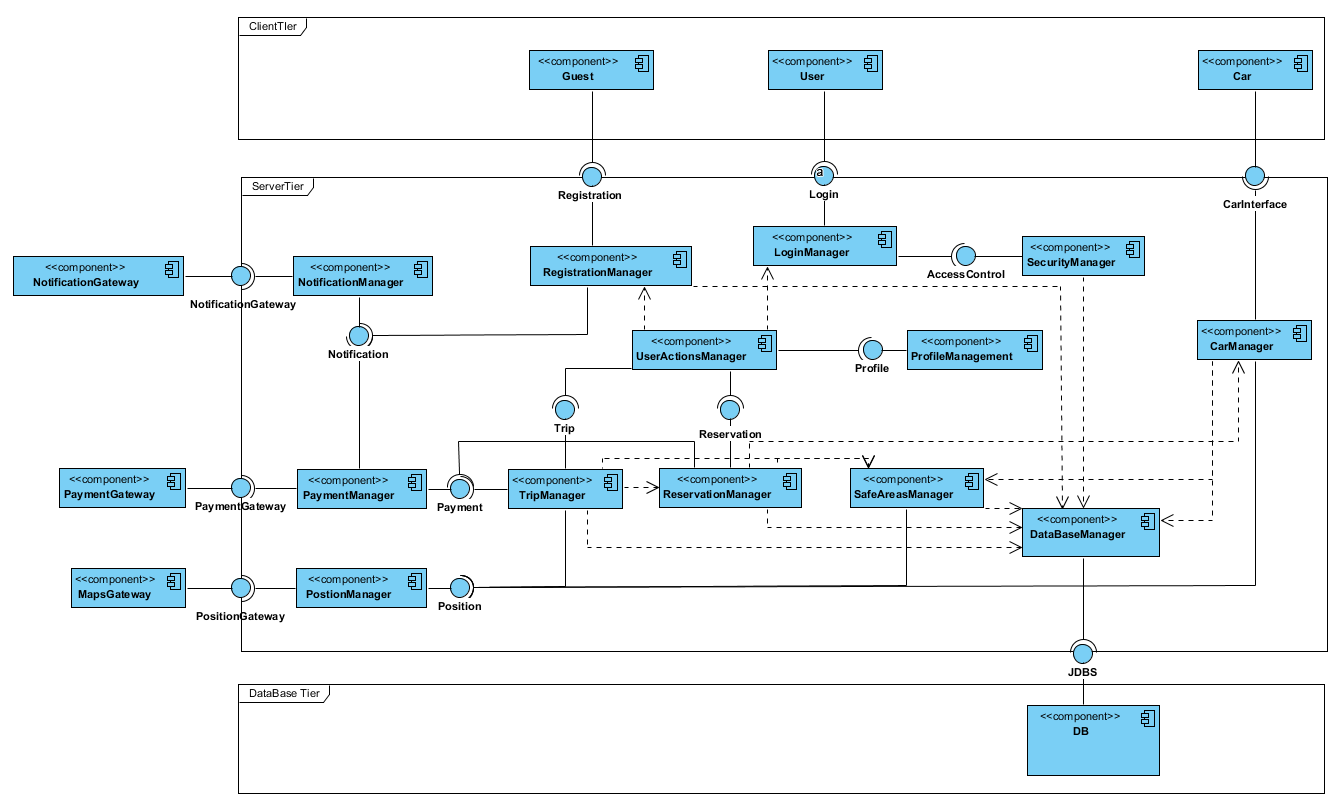
# Integration Strategy

## 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.

## Elements to be integrated

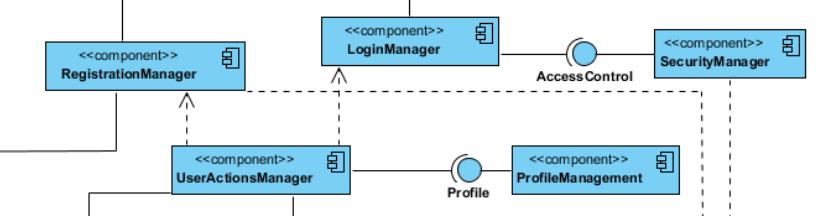
In this paragraph, we are going to list all components that must be integrated. We report our component diagram (taken from Design Document) for a clearer comprehension of interfaces and main components.

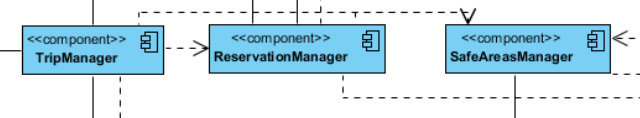


## Integration Testing Strategy

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



## Sequence of Component/Function Integration

### Software Integration Sequence

The IDs represents the order in which the integration testing should proceed.

* User basic functionalities



|  |  |  |
| --- | --- | --- |
| ID | Integration Test | Paragraph |
| I1 | RegistrationManager 🡪 UserActionManager |  |
| I2 | LoginManager 🡪 SecurityManager |  |
| I3 | LoginManager 🡪 UserActionManager |  |
| I4 | UserAction 🡪 ProfileManagement |  |

* Trip management functionalities



|  |  |  |
| --- | --- | --- |
| ID | Integration Test | Paragraph |
| I5 | TripManager 🡪 ReservationManager |  |
| I6 | TripManager 🡪 SafeAreasManager |  |

Then we integrate these two main groups with other components which work as interfaces.

So, first User basic functionalities: we call the result User subsystem.



|  |  |  |
| --- | --- | --- |
| ID | Integration Test | Paragraph |
| I7 | UserBasicFunctionalities 🡪 ReservationManager |  |
| I8 | UserBasicFunctionalities 🡪 SafeAreasManager |  |

Then trip management functionalities: we call the result Trip subsystem.



|  |  |  |
| --- | --- | --- |
| ID | Integration Test | Paragraph |
| I9 | TripManagementFunctionalities🡪 PositionManager |  |
| I10 | TripManagementFunctionalities🡪 PaymentManager |  |
| I11 | TripManagementFunctionalities🡪 CarManager |  |

### Subsystem Integration Sequence

Now we first integrate both User subsystem and Trip subsystem with DBMS, then we integrate them together.

# Individual Steps and Test Description

# 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.

## 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.

## 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.

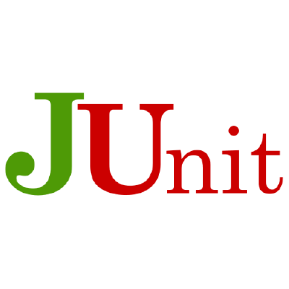
Indeed, almost every personal computer nowadays has 2GB of RAM and a dual-core CPU. Our application requires much less than that specification because it’s a little portion of the system in which the user can register his profile and only modify some information about himself.

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

# 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.

# Program Stubs and Test Data Required

## 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. |

## 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

# Appendix

## Used tools

## Effort spent