Pietro Crovari – Enrico Gnecco

PowerEnjoy

Integration Test Plan Document

V 1.0

Table Of Contents

Sommario

1. Introduction

1.1 Revision History

1.2 Purpose and Scope

The idea of the Integration Test Plan Document (ITPD) is to underline how the integration process should be done in the environment of PowerEnjoy.

A rigorous test plan will lead to a better comprehension, scalability and performance of the application.

Once defined the borders and everything related to the Design of the application, here the other side of the coin is faced: various kinds of test will be suggested and described to underline how the system should work and the stress and the load that it's capable to support; the result will be the creation of the idea about how the system fulfils the requirements and so the goals, ending up with the guarantee that such goals will be satisfied.

At the beginning of the integration process will be presented all the sub-components involved, how they will interact, focusing on the criteria and the rationale that leads this document. Different approaches are presented at different levels in order to underline the different layers of abstraction of the components and how they cooperate to fulfil the goals.

Secondly, the core of the document will be dedicated to all the integration tests, appropriately detailed.

Finally, a list of all the tools and a description of the environment used will be given and described

1.3 List of Definitions and Abbreviations

1.4 List of Reference Documents

- RAS Document (assignment 1)
- Design Document (assignment 2)
- VerificationTool.pdf

2. Integration Strategy

2.1 Entry Criteria

As the aim of the document has been described before, here the idea is to define what is needed to further analyse the integration tests and put them in practice.

The Requirements Analysis and Specification Document are Necessary steps of this document; they set the fundamental boundaries of PowerEnjoy and so the basic and intuitive level from where to start the testing part.

After that, it's mandatory to consider how much every component is deployed with respect to the functionalities it exploits. However, this leads to a consideration about the importance of the single component inside the environment of the application, so the discussion is merely to consider to define an order for a Critical Module Strategy. In detail, there must be a great focus on the Dispatcher, the UserManager, the CarManager and the FormManager, how this will be deployed is latterly described.

Concentrating on those will inevitably improve the performances of the systems avoiding possible errors or faults.

Moreover, all that concern the human interface of the system is managed by a separated server, also the client is considered without logic, and so, since everything is done by the business and persistence layers of the server, the discussion will cover these themes and how the integration among the internal components is performed.

2.2 Elements to be integrated

Essentially the elements to be integrated are those described in the Design Document, in particular the ones in the Component View Diagram. That schema underlines perfectly the internal structure of the system.
The logic of the server is provided and everything revolves around it; from this point of view, it is quite clear which are the components to test and then to be integrated one with the others.
The modularity chosen offers a great advantage. Even if this choice doesn't solve the problem of the requirements, because there is no focus on the developed features, here the granularity of the components helps to follow that strategy; from a low-level point of view, some components will see also a white box testing analysis.
To simplify the following procedure, the first two components will be faced in detail, the others will follow the same idea and so the sub-components will be simply listed.

UserManager: it can be seen as composed of sub-components:

- Info User Manager
- Rent Manager (with a Bill Manager inside)
- Reservation Manager

These sub-components allow a White Box Strategy for testing due to their simplicity, however, after that, an integration will be necessary and a Black Box approach is easier to be acted.

FormManager: composed of

- Damage Manager
- User Form Manager
- Maintenance Form Manager

The idea, as before, is to test singularly as a white box all the sub-components and then integrate them, managing them as black boxes.

OperatorManager: it considers also a Maintenance Manager that can be dealt as before: first as a white and then as a black box.

CarManager: Safe Area Controller, Position Manager, Sensors Info Manager

ExternalDataManager: Police Manager, Payment Manager

MapManager and **LoginManager** don't have any sub-component and so the integration can start with a black box approach immediately.

2.3 Integration Testing Strategy

As stated, the way in which the integration testing process will be carried on is different from a Critical Module Strategy: indeed, even if a lot of effort will be spent over the main components, a Bottom-Up approach is preferable due to the modular structure decided in the Design Document.
First of all, a white box approach will define the basic level of integration in our system; secondly, a black

box strategy is preferable due to the boundaries introduced in the previous document (DD).
That is not all: such a similar way of working can be good and efficient but it will not assure us on performances and real fulfilments of the goals; in other words, there is the need of a different approach from a high level point of view: a Thread Strategy, that will test the main features of the application, covering as many missed errors as possible.

Lastly, it should be noticed that the discussion will cover only those components to be developed, in other words already present technologies, or those developed by others, are not considered here. So the old and the new databases, the external services, the car technology and the operators' technology as those are assumed to be already present or deployed by a third part (see RASD).

2.4 Sequence of Component/Function Integration

The aim of this chapter is to describe the order of the integration, and so the related testing, of the sub-components and macro-components of the application.

2.4.1 Software Integration Sequence

As already mentioned, here the strategy will follow a Bottom-Up approach, but with an order that follows a Critical Module Paradigm.

RentManager

ID	Integration Test	Aim
I1	RentManager → BillManager	Instantiation

UserManager

ID	Integration Test	Aim
I2	UserManager → InfoUserManager	Instantiation
I3	UserManager → RentManager	Instantiation
I4	UserManager → ReservationManager	Instantiation
I5	ReservationManager → RentManager	Passage of information at Rent beginning
I6	RentManager → InfoUserManager	Info user i.e. payment method
I7	ReservationManager → InfoUserManager	User's suitability

FormManager

ID	Integration Test	Aim
I8	FormManager → DamageManager	Instantiation
I9	FormManager → UserFormManager	Instantiation
I10	FormManager →MaintenanceFormManager	Instantiation
I11	UserFormManager → DamageManager	Management damaged car

CarManager

ID	Integration Test	Aim
I12	CarManager → PositionManager	Instantiation
I13	CarManager → SafeAreaController	Instantiation
I14	CarManager → SensorsInfoManager	Instantiation
I15	PositionManager → SafeAreaController	Position management
I16	SensorsInfoManager → SafeAreaController	Parked car notification

ExternalDataManager

ID	Integration Test	Aim
I17	ExternalDataManager → PoliceManager	Instantiation
I18	ExternalDataManager → PaymentManager	Instantiation
I19	*Suppressed*	

OperatorsManager

ID	Integration Test	Aim
I20	OperatorsManager → MaintenanceManager	Instantiation

Server Components

ID	Integration Test	Aim
I21	Dispatcher → UserManager	Instantiation
I22	Dispatcher → OperatorManager	Instantiation
I23	Dispatcher → FormManager	Instantiation
I24	Dispatcher → CarManager	Instantiation
I25	Dispatcher → TimerManager	Timer management
I26	UserManager → ExternalDataManager	Data Check (i.e. at registration)
I27	UserManager → ExternalDataManager	Payment
I28	UserManager → LoginManager	Login control
I29	UserManager → CarManager	Available cars research
I30	UserManager → MapManager	Map generation
I31	UserManager → CarManager	Car reservation
I32	UserManager → FormManager	Form generation
I33	UserManager → CarManager	Car use
I34	OperatorsManager → LoginManager	Login control
I35	OperatorsManager → CarManager	Cars research
I36	OperatorsManager → MapManager	Map generation
I37	OperatorsManager → CarManager	Beginning of maintenance
I38	OperatorsManager → FormManager	Maintenance form
I39	OperatorsManager → CarManager	End of maintenance
I40	CarManager → UserManager	End of Rent

2.4.2 Sub-Systems Integration Sequence

Here a high-level description will be given in order to give an idea of how components should be integrated together. To be noticed here is the absence of the Thread Analysis, which will be faced later on.

ID	Integration Test	Aim
I41	Operator → Server	Login
I42	Operator → Server	Car Research
I43	Operator → Server	Maintenance Begin
I44	Operator → Server	Maintenance End
I45	Client → Server	Login
I46	Client → Server	Car Research
I47	Client → Server	Reservation
I48	Client → Server	Registration
I49	Client → Server	Rent
I50	Server → Car	Unlock

I51	Car → Server	Position
I52	Car → Server	End of rent
I53	Server → Client	Notification of payment

3. Individual Steps and Test Description

As mentioned before, the integration test phase will assume the databases working correctly and loaded with a dataset sufficient to execute any test.

RentManager

1. Integration Test Case I1

Test Case Identifier	I1T1
Test Item(s)	RentManager → BillManager
Input Specification	Create typical billing request after a ride
Output Specification	Check if the BillManager is correctly instantiated, with the correct discounts selected
Environmental Needs	Rent Manager

2. Integration Test Case I2

Test Case Identifier	I2T1
Test Item(s)	UserManager → InfoUserManager
Input Specification	Create typical Users' information
Output Specification	Check if the UserManager correctly instantiates an InfoUserManager, passing the right data
Environmental Needs	User Manager

3. Integration Test Case I3

Test Case Identifier	I3T1
Test Item(s)	UserManager → RentManager
Input Specification	Create typical Rent Information
Output Specification	Check whether the UserManager correctly instantiates a RentManager
Environmental Needs	UserManager

4. Integration Test Case I4

Test Case Identifier	I4T1
Test Item(s)	UserManager → ReservationManager
Input Specification	Typical creation request
Output Specification	Check if the UserManager is able to correctly instantiate a ReservationManager, passing the correct information
Environmental Needs	UserManager

5. Integration Test Case I5

Test Case Identifier	I5T1
Test Item(s)	ReservationManager → RentManager
Input Specification	Typical Reservation Information
Output Specification	Check if the ReservationManager is able to ask for the creation of a RentManager, using its data.
Environmental Needs	I3, I4 succedeed

Test Case Identifier	I5T2
Test Item(s)	ReservationManager → RentManager
Input Specification	Typical Reservation Information
Output Specification	Check if the ReservationManager correctly destroy itself once the RentManager is correctly created
Environmental Needs	I3, I4 succeeded

6. Integration Test Case I6

Test Case Identifier	I6T1
Test Item(s)	RentManager → InfoUserManager
Input Specification	Suitable user Id
Output Specification	Check if the InfoUserManager communicates the correct payement for that hypothetical User
Environmental Needs	I2, I3 succeeded

7. Integration Test Case I7

Test Case Identifier	I7T1
Test Item(s)	ReservationManager → InfoUserManager
Input Specification	User's peronal information with a valid driving licence
Output Specification	Receive the confirmation of the validity of the driving licence
Environmental Needs	I2, I4 succeeded, internet connection

Test Case Identifier	I7T2
Test Item(s)	ReservationManager → InfoUserManager
Input Specification	User's peronal information without a valid driving licence
Output Specification	Receive the segnalation of the missing driving licence
Environmental Needs	I2, I4 succeeded, internet connection

Test Case Identifier	I7T3
Test Item(s)	ReservationManager → InfoUserManager

Input Specification	User's peronal information with a valid credit card number
Output Specification	Receive the confirmation of the validity of the credit card
Environmental Needs	I2, I4 succeeded, internet connection

Test Case Identifier	I7T4
Test Item(s)	ReservationManager → InfoUserManager
Input Specification	User's peronal information with a false credit card
Output Specification	Receive the segnalation of the missing credit card
Environmental Needs	I2, I4 succeeded, internet connection

8. Integration Test Case I8

Test Case Identifier	I8T1
Test Item(s)	FormManager → DamageManager
Input Specification	Create damage segnalation
Output Specification	Check whether a DamageManager is correctly instantiated
Environmental Needs	Form Manager

9. Integration Test Case I9

Test Case Identifier	I9T1
Test Item(s)	FormManager → UserFormManager
Input Specification	Form Request from a hypothetical User
Output Specification	Check whether a form with the correct questions is instantiated and associated to the correct User
Environmental Needs	Form Manager

10. Integration Test Case I10

Test Case Identifier	I10T1
Test Item(s)	FormManager →MaintenanceFormManager
Input Specification	Form Request from a hypothetical Operator
Output Specification	Check whether a form with the correct questions is instantiated and associated to the correct Operator
Environmental Needs	Form Manager

11. Integration Test Case I11

Test Case Identifier	I11T1
Test Item(s)	UserFormManager → DamageManager
Input Specification	Create a form with the segnalation of a damage
Output Specification	Check whether a DamageManager is correctly instantiated and if it contains the right information
Environmental Needs	I8, I9 succeeded

12. Integration Test Case I12

Test Case Identifier	I12T1
Test Item(s)	CarManager → PositionManager
Input Specification	Create a position to manage by a hypothetical car
Output Specification	Check whether a PositionManager is correctly instantiated
Environmental Needs	CarManager

13. Integration Test Case I13

Test Case Identifier	I13T1
Test Item(s)	CarManager → SafeAreaController
Input Specification	Create a position within a Safe Area
Output Specification	Check whether a SafeAreaController is correctly instantiated
Environmental Needs	CarManager

14. Integration Test Case I14

Test Case Identifier	I14T1
Test Item(s)	CarManager → SensorsInfoManager
Input Specification	Create a SensorInfo collection of data
Output Specification	Check whether a SensorInfoManager is correctly instantiated
Environmental Needs	CarManager

15. Integration Test Case I15

Test Case Identifier	I15T1
Test Item(s)	PositionManager → SafeAreaController
Input Specification	Create a position within a SafeArea
Output Specification	Check whether the SafeAreaController authorizes the parking
Environmental Needs	I12, I13 succeeded

Test Case Identifier	I15T2
Test Item(s)	PositionManager → SafeAreaController
Input Specification	Create a position outside SafeAreas
Output Specification	Check whether the SafeAreaController forbids the parking
Environmental Needs	I12, I13 succeeded

16. Integration Test Case I16

Test Case Identifier	I16T1
Test Item(s)	SensorsInfoManager → SafeAreaController
Input Specification	Create the swiching the motor off notification

Output Specification	Check if the SensorsInfoManager report correctly the notification, asking for the SafeArea check and obtaining a correct answer
Environmental Needs	I15, I14 succedeed

17. Integration Test Case I17

Test Case Identifier	I17T1
Test Item(s)	ExternalDataManager → PoliceManager
Input Specification	Create a driving license check request
Output Specification	Check whether a PoliceManager is correctly instantiated
Environmental Needs	ExternalDataManager

18. Integration Test Case I18

Test Case Identifier	I18T1
Test Item(s)	ExternalDataManager → PaymentManager
Input Specification	Create a patment method check request
Output Specification	Check whether a PaymentManager is correctly instantiated
Environmental Needs	ExternalDataManager

Test Case Identifier	I18T2
Test Item(s)	ExternalDataManager → PaymentManager
Input Specification	Create a payment request
Output Specification	Check whether a PaymentManager is correctly instantiated
Environmental Needs	ExternalDataManager

19. Integration Test Case I19

Suppressed

20. Integration Test Case I20

Test Case Identifier	I20T1
Test Item(s)	OperatorsManager → MaintenanceManager
Input Specification	Create maintenance request by a hypothetical operator
Output Specification	Check whether a MaintenanceManager is correctly instantiated
Environmental Needs	OperatorsManager

21. Integration Test Case I21

Test Case Identifier	I21T1
Test Item(s)	Dispatcher → UserManager
Input Specification	Create login request from a suitable user

Output Specification	Check whether a UserManager is correctly instantiated, and the information are correctly submitted to.
Environmental Needs	Dispatcher

Test Case Identifier	I21T2
Test Item(s)	Dispatcher → UserManager
Input Specification	Create registration request from a suitable user
Output Specification	Check whether a UserManager is correctly instantiated, and the information are correctly submitted to.
Environmental Needs	Dispatcher

22. Integration Test Case I22

Test Case Identifier	I22T1
Test Item(s)	Dispatcher → OperatorManager
Input Specification	Create login request from a suitable operator
Output Specification	Check whether an OperatorManager is correctly instantiated, and the information are correctly submitted to it
Environmental Needs	Dispatcher

23. Integration Test Case I23

Test Case Identifier	I23T1
Test Item(s)	Dispatcher → FormManager
Input Specification	Create compiled form to elaborate
Output Specification	Check if a FormManager is correctly instantiated and the form is correctly submitted to.
Environmental Needs	Dispatcher

24. Integration Test Case I24

Test Case Identifier	I24T1
Test Item(s)	Dispatcher → CarManager
Input Specification	Create communication from a Car
Output Specification	Check whether a CarManager is correctly instantiated, and the information are correctly submitted to it
Environmental Needs	Dispatcher

25. Integration Test Case I25

Test Case Identifier	I25T1
Test Item(s)	Dispatcher → TimerManager
Input Specification	Create reservation confirmation

Output Specification	Check if the Dispatcher instantiate a Timer linked to the reservation
Environmental Needs	Dispatcher

Test Case Identifier	I25T2
Test Item(s)	Dispatcher → TimerManager
Input Specification	Create an expired rent timer
Output Specification	Check if the Dispatcher reveals the extinguished timer and alerts the UserManager
Environmental Needs	Dispatcher

26. Integration Test Case I26

Test Case Identifier	I26T1
Test Item(s)	UserManager → ExternalDataManager
Input Specification	Create registration request with valid payment method and driving license
Output Specification	Check if the External data manager validates user's information
Environmental Needs	I17, I18, I21 succeeded

Test Case Identifier	I26T2
Test Item(s)	UserManager → ExternalDataManager
Input Specification	Create registration request with invalid payment method and/or driving license
Output Specification	Check if the new user is registered as not suitable
Environmental Needs	I17, I18, I21 succeeded

27. Integration Test Case I27

Test Case Identifier	I27T1
Test Item(s)	UserManager → ExternalDataManager
Input Specification	Create payment request
Output Specification	Check if the payment request is correctly accomplished
Environmental Needs	I17, I18, I21 succeeded

28. Integration Test Case I28

Test Case Identifier	I28T1
Test Item(s)	UserManager → LoginManager
Input Specification	Create login request from a suitable user
Output Specification	Check if the user is allowed to login
Environmental Needs	I21 succeeded

29. Integration Test Case I29

Test Case Identifier	I29T1

Test Item(s)	UserManager → CarManager
Input Specification	Create a user's request for a research
Output Specification	Check if the user manager correctly produces a list with all the available cars
Environmental Needs	I21 succeeded

Test Case Identifier	I29T2
Test Item(s)	UserManager → CarManager
Input Specification	Create a user's request for a research which has no result (e.g. outside the e.g. with no car available)
Output Specification	Check if the user manager correctly produces a list with no cars in it
Environmental Needs	I21 succeeded

30. Integration Test Case I30

Test Case Identifier	I30T1
Test Item(s)	UserManager → MapManager
Input Specification	Create a list of car and their position
Output Specification	Check whether a UserManager is returned a map with the correct position of all the cars in the list displayed on it
Environmental Needs	I21 succeeded

31. Integration Test Case I31

Test Case Identifier	I31T1
Test Item(s)	UserManager → CarManager
Input Specification	Create a reservation request
Output Specification	Check if the reservation is correctly accomplished, being stored on the database and the car is marked as booked
Environmental Needs	I21, I24 succeeded

32. Integration Test Case I32

Test Case Identifier	I32T1
Test Item(s)	UserManager → FormManager
Input Specification	Create a request of unlocking a car of a previous reservation
Output Specification	Check whether the UserManager correctly obtains a form to send to the User
Environmental Needs	I21, I23 succeeded

33. Integration Test Case I33

Test Case Identifier	I33T1

Test Item(s)	UserManager → CarManager
Input Specification	Create a validation of a hypothetical compiled form the form manager
Output Specification	Check if the carManager sends the notification to unlock the car
Environmental Needs	I21, I24 succeeded

34. Integration Test Case I34

Test Case Identifier	I34T1
Test Item(s)	OperatorsManager → LoginManager
Input Specification	Create login request from a suitable operator
Output Specification	Check if the user is allowed to login
Environmental Needs	I22 succeeded

Test Case Identifier	I34T2
Test Item(s)	OperatorsManager → LoginManager
Input Specification	Create login request from a not suitable user
Output Specification	Check if the user is forbidden from accessing the system
Environmental Needs	I22 succeeded

35. Integration Test Case I35

Test Case Identifier	I35T1
Test Item(s)	OperatorsManager → CarManager
Input Specification	Create an operator request for a research
Output Specification	Check if the carManager correctly produces a list with all the out-of-order cars
Environmental Needs	I22 succeeded

36. Integration Test Case I36

Test Case Identifier	I36T1
Test Item(s)	OperatorsManager → MapManager
Input Specification	Create a list of cars and their position
Output Specification	Check whether a OperatorsManager is returned a map with the correct position of all the cars in the list displayed on it
Environmental Needs	I22 succedeed

37. Integration Test Case I37

Test Case Identifier	I37T1
Test Item(s)	OperatorsManager → CarManager
Input Specification	Create a request for taking a car under maintenance

Output Specification	Check whether the car is marked as underMaintenance, and it receive the notification
Environmental Needs	I22 succeeded

38. Integration Test Case I38

Test Case Identifier	I38T1
Test Item(s)	OperatorsManager → FormManager
Input Specification	Create a request for an end-of-maintenance form
Output Specification	Check if the OperatorsManager is returned a correct form to submit
Environmental Needs	I22, I23 succeeded

39. Integration Test Case I39

Test Case Identifier	I39T1
Test Item(s)	OperatorsManager → CarManager
Input Specification	Create a form compiled by an hypothetical operator
Output Specification	Check if the car is made available again or not according to the analysis of the form, performed by the formManager
Environmental Needs	I22, I23, I24 succeeded

40. Integration Test Case I40

Test Case Identifier	I40T1
Test Item(s)	CarManager → UserManager
Input Specification	Create a notification of end of rent by a car previously rented
Output Specification	Check if the payment is processed and the rent stored as well.
Environmental Needs	I21, I24 succeeded

41. Integration Test Case I41

Test Case Identifier	I41T1
Test Item(s)	Operator → Server
Input Specification	Create a login request by a hypothetical operator's device
Output Specification	Check if operator is allowed to access to his personal area
Environmental Needs	I34 succeeded

42. Integration Test Case I42

Test Case Identifier	I42T1
Test Item(s)	Operator → Server

Input Specification	Create a research request by an operator's device
Output Specification	Check if the map with the results is returned correctly.
Environmental Needs	I35, I36 succeeded

43. Integration Test Case I43

Test Case Identifier	I43T1
Test Item(s)	Operator → Server
Input Specification	Create a qrCode communication by an Operator's device
Output Specification	Check if the car is unlocked and updated as under maintenance
Environmental Needs	I37 succeeded

44. Integration Test Case I44

Test Case Identifier	I44T1
Test Item(s)	Operator → Server
Input Specification	Create a qrCode communication of an Under-Maintenance car of a complete job by an Operator's device
Output Specification	Check if the car is locked and updated as available
Environmental Needs	I39 succeeded

Test Case Identifier	I44T2
Test Item(s)	Operator → Server
Input Specification	Create a qrCode communication of an Under-Maintenance car of an incomplete job by an Operator's device
Output Specification	Check if the car is locked and updated as out of order again
Environmental Needs	I39 succeeded

45. Integration Test Case I45

Test Case Identifier	I45T1
Test Item(s)	Client → Server
Input Specification	Create a login request by a User's Device
Output Specification	Check either the user is allowed to access
Environmental Needs	I28 succeeded

46. Integration Test Case I46

Test Case Identifier	I46T1
Test Item(s)	Client → Server
Input Specification	Create a request for searching a car

Output Specification	Check if the map is correctly displayed
Environmental Needs	I29, I30 succeeded

47. Integration Test Case I47

Test Case Identifier	I47T1
Test Item(s)	Client → Server
Input Specification	Create a reservation request from a User's device
Output Specification	Check if the reservation succeeds
Environmental Needs	I31 succeeded

48. Integration Test Case I48

Test Case Identifier	I48T1
Test Item(s)	Client → Server
Input Specification	Create a registration request from a User's device
Output Specification	Check if the user is correctly registered
Environmental Needs	I2 succedeed

49. Integration Test Case I49

Test Case Identifier	I49T1
Test Item(s)	Client → Server
Input Specification	Create a rent request from a User's device
Output Specification	Check if the rent is processed properly
Environmental Needs	I31 succeeded

50. Integration Test Case I50

Test Case Identifier	I50T1
Test Item(s)	Server → Car
Input Specification	Create an Unlock request
Output Specification	Check if the car is accessible
Environmental Needs	I32 succeeded

51. Integration Test Case I51

Test Case Identifier	I51T1
Test Item(s)	Car → Server
Input Specification	Create a position update from a car
Output Specification	Check if position on the database is correctly updated
Environmental Needs	I12 succeeded

52. Integration Test Case I52

Test Case Identifier	I52T1
Test Item(s)	Car → Server
Input Specification	Create a notification of end of rent from an used car
Output Specification	Check whether the end of rent is treated properly
Environmental Needs	I24 succeeded

53. Integration Test Case I53

Test Case Identifier	I53T1
Test Item(s)	Server → Client
Input Specification	Create a notification of successful payment
Output Specification	Check if the payment is successfully notified to the user
Environmental Needs	I26 succeeded

Integration Test Procedure

Test Procedure Identifier	TP1
Purpose	This test procedure verifies if the dispatcher component: ✓ Can handle clients' input ✓ Can handle operators' input ✓ Can handle cars input ✓ Can handle timeout input ✓ Can send requested information to a Client ✓ Can send requested information to an Operator ✓ Can send requested information to a Car
Procedure Steps	Execute I21 to I24, I41 to I53

Test Procedure Identifier	TP2_1
Purpose	This test procedure verifies if the client application: ✓ Can handle login request ✓ Can handle registration request ✓ Can handle Car Research Request ✓ Can handle Car Reservation request ✓ Can handle Form display ✓ Is sufficiently easy to be used by a large segment of the population
Procedure Steps	Execute I45 to I49

Test Procedure Identifier	TP2_2
Purpose	This test procedure verifies if the client application:

	✓ Is accessible (i.e. w.r.t colour-blind or aged people) ✓ Is sufficiently easy to be used by a large segment of the population ✓ Crates satisfaction and not frustration in users
Procedure Steps	Ask several people to accomplish some easy tasks on the application and ask them for feedback

Test Procedure Identifier	TP3
Purpose	This test procedure verifies if the operator application: ✓ Can handle login request ✓ Can handle Car Research Request ✓ Can handle begin and end of Maintenance Request ✓ Can handle Maintenance Form display
Procedure Steps	Execute I41 to I44

Test Procedure Identifier	TP4
Purpose	This test procedure verifies if the Car Logic: ✓ Can receive Server Requests ✓ Can unlock itself when asked ✓ Can communicate position updates ✓ Can notify end of rent ✓ Can correctly notify rent information(the ones for calculate bills)
Procedure Steps	Execute I50 to I52

4. Tools and Test Equipment Required

As already mentioned, various approaches for testing and integration can be applied for the PowerEnjoy application.
From a low-level point of view, even a white box analysis can be carried on to identify errors or troubles that can verify at the end of the deployment. Fortunately, Laravel framework offers various tools, and packages, to make a proper analysis and testing on the objects we are going to integrate.

However, that can't be the entire solution, indeed there is a lack between a low level and a functional level: there is the need of tools for a black box analysis. To do so various toolbox are considered: first, a mock-up tool is necessary; secondly, a tool verifying the correct behaviour of injections and instantiations by containers or internal managers. Here the usage of Laravel shows the other side of the coin, indeed, we can't use the frameworks or the applications presented during the lectures but we are forced to adopt different solutions. Those are: an intense use of PHPUnit, also for the white box analysis; PHP DI for the managing of dependencies injection; and finally, the adoption of Prophecy framework for the mock-ups. Lastly it could be possible to test performances adopting JMeter also on PHP.

The idea is to adopt a well-known framework to perform proper analysis and integration tests, already detailed, building the application. PHPUnit has been chosen because it is one of the most used frameworks

used in this kind of tasks and it offers an ad-hoc solution for PHP-based web applications. This adoption will impact also on the costs of teaching how to use this tool and so on the effort spent for the entire integration procedure.

Secondly, in order to perform a proper integration test phase there is the need of an interface-driven analysis; in other words, there is the need of a tool which offers mock-ups capable of interacting with the objects analysed. This is the aim of the adoption of Prophecy. Unfortunately, this mocking framework is not as known as PHPUnit and it can come out to be an extra source of costs, but it is mandatory for a proper integration phase of PowerEnjoy application.

PHP DI has been chosen because it offers a good, practical solution for an intense usage of dependencies, it has a properly designed syntax to simplify all the operations related to class injections; it can be very useful during the white box analysis, even if a lighter code will help even during a later phase of the application without any doubt.

Anyway, to perform an efficient and effective white box analysis it's considerably useful to adopt a manual testing, supported by the choices of frameworks already presented.

These operations can't be done without a proper equipment, required in order to run and support all the devices and platforms. Some of them are assumed to be already present, such as operator devices, databases and car systems, and no further details are presented, but all the others are here discussed.

Starting from the clients' devices, there is a great variety of physical instruments and Operating Systems that can occur, so there is the need of testing PowerEnjoy application on the following systems:

- Smartphones and tablets of ordinary sizes (3''-6'' and 7''-12'' respectively) with Android OS;
- Windows phone, same sizes as before;
- iOS smartphones and tablets for each member of the respective families;
- Typical set of desktop and computers.

However, in order to reduce the costs, it's mandatory to observe that several testing hardware devices can be emulated and they are not strictly necessary to the fulfilment of the integration testing phase. Even if there must be a separate way to achieve the reading of the QR codes, because virtual devices can't perform such a task.

In addition, a statistical pre-analysis should be performed to obtain the most common choices made by the customers.

Indeed, the driving criteria is to choose devices as widely adopted as possible, and so there isn't any fixed solution but, maybe during a maintenance phase, there will be the need of changes acquiring new platforms, new hardware and so enlarging the company possibilities.

The importance of the choice of these components is underlined by the fact that we are going to develop and test both the web application and the mobile one on the same mobile devices. This is a direct consequence of the choice of relying on a server which performs all the operations and using the devices only to show the results and give a way to the customer to ask for a certain type of service.

On the other hand, from the business point of view, there is the need of a server to guarantee all the correct behaviours. As already presented in the RAS Document and D Document, the adopted choice is a Server with a running Apache Web Server distribution on which a Laravel Framework could work; and for the persistency part a MySql database is preferable due to the low costs, but it's not mandatory, however it's strongly suggested a unique choice between the old and the new database.

5. Program Stubs and Test Data Required

5.1 Program Stubs

As said, the integration will follow a bottom-up analysis. For this reason, there will be necessary a driver for every single component, able to instantiate it and call every function to be tested. This will lead to a complete control of single components during all the integration phase.

The development of these drivers will be done during the integration phase, according all the test that must be performed on every single component, as described in the previous sections

5.2 Test Data

As specified in tests in Section 3, the following list of set of data is necessary in order to guarantee the execution of all the described tests:

- A set users composed of valid dataset and invalid ones; In example:
  - ✓ Users with missing information
  - ✓ Dataset with NULL values
  - ✓ Users with expired Driving Licence
  - ✓ Users too young for having a Driving Licence
  - ✓ Users with a false credit card
  - ✓ Users with wrong credentials
  - ✓ Users researching car in position outside the admitted area (i.e. in other cities)
- A list of dataset for cars with:
  - ✓ Cars in every admitted status
  - ✓ Cars waiting for the form compilation w
  - ✓ Cars missing of any information (i.e. because of a failure of the car logic)
  - ✓ Cars asking for SafeAreas correctly parked
  - ✓ Cars asking for SafeAreas parked outside them
  - ✓ Cars submitting end of rent information covering every possible situation (in other words, with every possible combination of discounts applicable)
- A set of operators with both valid and invalid credentials

6. Hours of Work

- Pietro Crovari: 21 hrs;
- Enrico Gnecco: 22 hrs;

7. Change Log