Politecnico di Milano

Software engineering 2

PowerEnjoy
ITPD v1.0

Alessandro Caprarelli	874206
Roberta Iero	873513
Giorgio De Luca	875598

January 14, 2017

Contents

Chapter 1

Introduction

1.1 Revision history

Table 1.1: Revision history

Version	Date
v1.0	15/01/2017

1.2 Purpose

This document represents the Integration Test Plan (ITP) and its main purpose is to describe the tests that we need to perform in order to verify that all the components of the software, previously identified in the Design Document, are correctly linked and work together. This document not only introduces the tests to be performed, but it also reasons the decision to take this set of tests, the order in which they will be executed and the expected result of each of them. In this step of the software design we consider components as black-boxes and the goal is to achieve a correct interoperability of them.

1.3 Scope

The aim of the project PowerEnjoy is to provide an automated service of car sharing. After a registration, a client can hire a car near him/her through the web or mobile

application and he/she can enjoy of all the extra services offered. The exact position of the client is determined by the GPS signal of the client's device or it's allowed to manually insert a specific address. So the system displays all the available cars in the client's close area. Then the client could make a reservation of a car, after which the system notifies the client with a message of confirmation with the car identifier. If the reservation procedure successfully ends the chosen car won't be available anymore for other clients. Moreover a client cannot hire more than one car at the same time. After the reservation the client has at most one hour to reach the car, when this time expires the system gives a penalty to the client and the car, previously hired, is available again for other clients. The system allows the client to cancel his/her reservation. When the client reaches the car, he/she can tell the system that is nearby through a specific button in the application and he/she starts to pay as soon as the engine ignites. During the travel the system supervises the current charge of the car and notifies it to the client through a screen located in the car. The system stops charging the amount of money that the client has to pay when he/she communicates through the application his/her decision to stop the rent. When the car is parked in a safe area and the client exits, the system locks the car automatically and starts the procedure of payment. The client is notified with the result of this procedure through an SMS, including the final fare.

1.4 Definition, acronyms, abbrevations

1.4.1 Definition

- **Guest client:** a person that is not already registered in the system or that has to log in.
- **Registered client:** a person who has valid access credentials to log in the system.
- **System administrator:** privileged user, in charge of managing administration processes and of updating business logic.
- **Reservation:** it is the action performed by a registered client that allow him/her to reserve an available car for maximum one hour.
- **Journey time = travel time:** time elapsed since the user starts the engine to the user parks the car and terminates the journey.
- **Available car:** a car that is not reserved by any user and has enough charge to be rented.

- **Unavailable car:** a car that is already reserved or damaged, so impossible to reserve.
- **Gps navigation:** it is the navigation system that is included in the car on board system. It could be used by the user to find direction to the final destination.
- **Final destination:** address where the user wants to go.
- **Safe area:** the region where is permitted to park and leave a car once the rent is terminated.
- **Power grid station:** the area where it's allowed users to park the cars, leaving them attached to the power grid.

1.4.2 Acronyms

- **ITP:** Integration Test Plan
- **DD:** Design Document
- **RASD:** Requirements Analysis and Specification Document
- **API:** Application Programming Interface
- **UI:** User Interface
- **DBMS:** Data Base management system

1.4.3 Abbrevetations

- **[In.m]:** n: level of integration. m: test number.

1.5 Reference documents

- RASD v1.1
- DD v1.0
- PowerEnjoy specification document (assignment).
- IEEE Std 1016tm-2009 Standard for Information Technology - System Design - Software Design Descriptions.

1.6 Document overview

The ITP is composed of five sections:

- Introduction: it provides a general description of the content of the Integration Plan Document and some additional information in order to be coherent with the RASD document and the Design Document, previously produced.
- Integration Strategy: this section introduces the strategy used for the integration, motivates the choices made and explains also the approach used to integrate. In fact the sequence to follow to perform the integration tests is specified.
- Individual Steps and Test Description: in this part all the test sets are defined and described according to the choices made in the previous section. In particular the expected result of each test case is specified.
- Tools and Test Equipment Required: this section contains a list of all the software tools and test equipment needed to perform the integration tests.
- Program Stubs and Test Data Required: in the last part of the document all the program stubs and all the special test data required for the integration test phase are identified and characterized.

Chapter 2

Integration strategy

This section describes decisions and criteria chosen for the ITP of the system of "PowerEnJoy".

2.1 Entry criteria

Before starting the Integration Testing phase, it is necessary to have successfully completed the Unit Testing of all the modules of the software. So from now on we assume that all the single modules of the system of PowerEnJoy work correctly. In addition to that we need some prerequisites to be satisfied:

- The software of PowerEnJoy must be code-completed.
- The system must satisfy all the requirements of the RASD and all the decisions taken in the DD.
- The Database should be ready and its tables are populated with initial data.

2.2 Elements to be integrated

The following image represents the Component View of the system of PowerEnJoy, as it has been designed in the DD. We bring it here again because it clearly shows all the components of Power EnJoy that are going to be tested for integration and all the dependences existing between them. These dependencies are particularly important because they give us a guideline to follow identifying the Integration Testing Strategy.

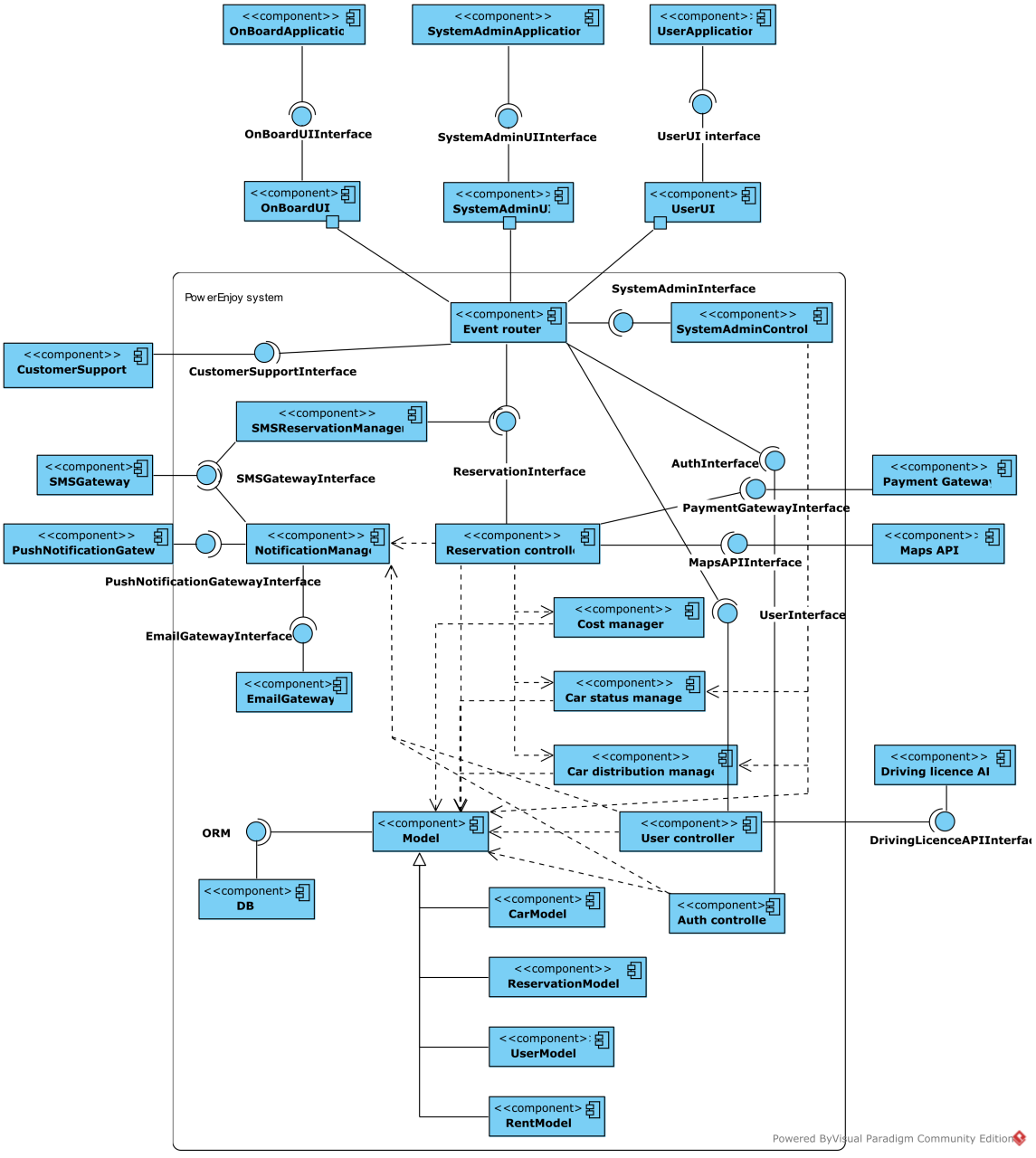

Figure 2.1: Component diagram

2.3 Integration Testing Strategy

We decide to adopt a Bottom Up approach in order to test the integration of modules of the system of PowerEnJoy. Bottom-up testing is a type of integration testing that integrates code units and tests them together, before testing the entire system. In this approach testing is conducted from sub module to main module, if the main module is not developed a temporary program called 'driver' is used to simulate the main module. For example if we look at the following figure, first of all we test module 4, 5, 6 and 7 individually using drivers. Then we test module 2 such that it calls 4 and 5 separately. If an error occurs we know that there is a problem in one of the modules, or in the interface between them. Then we test module 1 such that it calls module 3 and again if an error occurs we know that the problem is in module 3 or in the interface between module 1 and module 3.

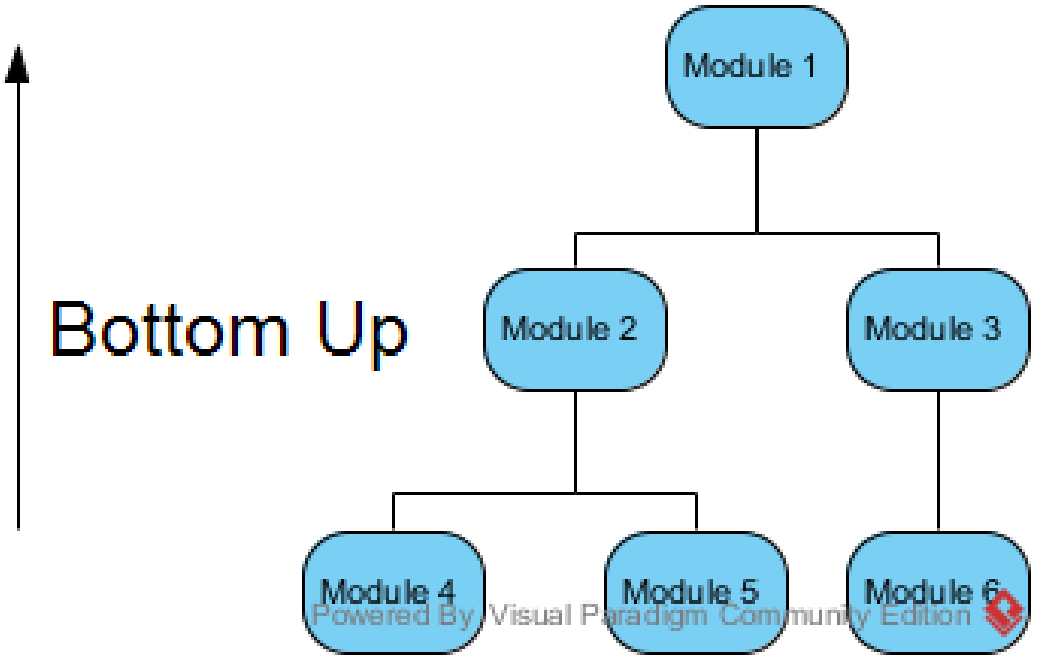

Figure 2.2: Bottom up strategy diagram

2.4 Sequence of Component/Function Integration

The bottom up strategy provides a simple workflow to test modules and makes easier the formulation and the observation of tests.

2.4.1 Software Integration Sequence

The sequence of integration tests depends on the strategy used to verify the correctness of tests. In this case, the bottom up strategy has the advantage of providing simple creation of conditions of the test.

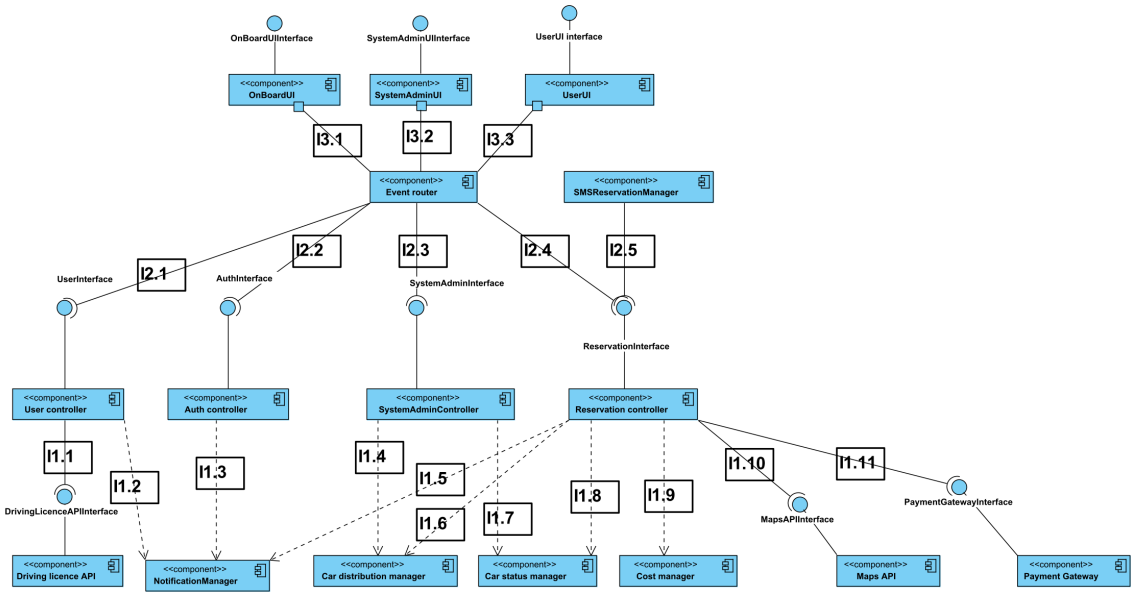

Figure 2.3: Integraton tests sequence

The order to follow is based on the type of component we are going to test:

- **Manager components** are the most important to be tested, because they represent the logic of the application; managers must be tested with specific inputs and various test environments to guarantee as much as possible the coverage of all the cases of the domain. It's suggested to start tests from easier functions/processes to more complex ones: in this way the next-level test can be run with no prerequisites on the atomic actions required.
- **Gateway components** are tested when the model and the controllers work as expected. Calls to external services could require an answer to the previous

request: the services we call have a particular service for tests and return us always a positive answer. Processes at this point involve an important number of components: as before, start from easier to more complex ones.

- **UI components** are the last type of component to be tested because if each test has been positively passed before, we expect to have no surprise in the presentation of processed data. The order to follow is the same of the workflow present in the BCE.

Chapter 3

Individual Steps and Test Description

3.1 Integration test case I1.1

- **Test case identifier:** I1.1
- **Test items:** UserController, Driving licence API
- **Purpose:** To check the response of the Driving licence API called after the registration process.
- **Input specification:** The UserController receives requests from the Event router.
- **Output specification:** Driving licence valid or not.
- **Environmental needs:** Driving licence API stub, EventRouter Driver.

3.2 Integration Test Case I1.2

- **Test case identifier:** I1.2
- **Test items:** UserController, NotificationManager
- **Purpose:** To check if the NotificationManager is properly called in different scenarios: it is called when an user submits a registration form in order to send

an email containing a password and as well when an user wants to change the password in order to send an email with the new password.

- **Input specification:** The UserController receives requests from the Event router.
- **Output specification:** Positive/negative answer about the conclusion of the operation
- **Environmental needs:** NotificationManager stub, EventRouter Driver.

3.3 Integration Test Case I1.3

- **Test case identifier:** I1.3
- **Test items:** AuthController, NotificationManager
- **Purpose:** To check if the NotificationManager is properly called when an user sends a "recover password" request in order to send an email containing the steps to recover the password.
- **Input specification:** The AuthController receives requests from the Event router.
- **Output specification:** Positive/negative answer about the conclusion of the operation
- **Environmental needs:** NotificationManager stub, EventRouter Driver.

3.4 Integration Test Case I1.4

- **Test case identifier:** I1.4
- **Test items:** SystemAdminController, CarDistributionManager
- **Purpose:** To test if the SystemAdminController calls correctly the methods of the CarDistributionManager when the System Administrator sends a request to check the distribution of the cars along the city or in a specific Safe Area.
- **Input specification:** The EventRouter calls the SystemAdminController component.

- **Output specification:** The expected output is a positive response if the number of cars in the city is higher than the minimum acceptable (previously set) and if the car are correctly distributed along the city. In the other case the output, according to the request performed by the System Administrator, can be: a list of the plates of the cars with too low battery to be available and their positions; a list of area in which there are not many cars available and the plates of the cars; a list of area with a great number of cars and the plates of the cars.
- **Environmental needs:** EventRouter Driver, Test Database filled with data concerning the specific Test Case.

3.5 Integration Test Case I1.5

- **Test case identifier:** I1.5
- **Test items:** ReservationController, NotificationManager
- **Purpose:** To check if the ReservationController calls correctly the methods of NotificationManager when it receives a request from the SMSReservationManager or from the Event router. The purpose of the ReservationController is to activate the NotificationManager to send communications to the client. In particular during the process of reservation when the client reserves a car through SMSs and at the end of the ride, with the recap, to the client, independently from the way in which he/she has reserved the car.
- **Input specification:** The ReservationController receives requests from the SMSReservationManager or from the Event router.
- **Output specification:** Positive/negative answer about the conclusion of the operation.
- **Environmental needs:** NotificationManager Stub, EventRouter Driver.

3.6 Integration Test Case I1.6

- **Test case identifier:** I1.6
- **Test items:** ReservationController, CarDistributionManager

- **Purpose:** To check the communication between the ReservationController and the CarDistributionManager. The ReservationController uses the methods of the CarDistributionManager for example when the client would make a reservation and is looking for a car near his/her position or when a client enables the 'save money' option during a rent, in order to find out where to live the car to obtain a discount.
- **Input specification:** The ReservationController receives a request from the EventRouter.
- **Output specification:** List of available cars; list of Safe Area.
- **Environmental needs:** EventRouter Driver, Test Database filled with data concerning the specific Test Case

3.7 Integration Test Case I1.7

- **Test case identifier:** I1.7
- **Test items:** SystemAdminController, CarStatusManager
- **Purpose:** To test if the request of the SystemAdminController to change the state of a car on the Database is performed correctly. In particular this is done when the System Administrator choose to check or change the status of a car.
- **Input specification:** The SystemAdminController receives a request from the EventRouter.
- **Output specification:** Modification of data of the Database, positive/negative answer about the conclusion of the operation.
- **Environmental needs:** EventRouter Driver, Test Database filled with data concerning the specific Test Case.

3.8 Integration Test Case I1.8

- **Test case identifier:** I1.8
- **Test items:** ReservationController, CarStatusManager

- **Purpose:** To check if the status of a car in the Database is correctly changed after a request made by the ReservationController. In particular this happens every time a car is reserved or the rent finishes.
- **Input specification:** We can distinguish different kind of situations:
  - The client does a reservation through the application, so the ReservationController receives a request from the EventRouter.
  - The client does a reservation through SMS so the SMSReservationManager calls the ReservationController.
  - The client terminates his/her rent through the OnBoard application, so the ReservationController receives a request from the EventRouter.
  - The time to reach the car for the client has expired, so the ReservationController calls the methods of the CarStatusManager to change the status of the car.
- **Output specification:** Modification of data of the Database, positive/negative answer about the conclusion of the operation.
- **Environmental needs:** EventRouter Driver, SMSReservationManager Driver, Test Database filled with data concerning the specific Test Case.

3.9 Integration Test Case I1.9

- **Test case identifier:** I1.9
- **Test items:** ReservationController, CostManager
- **Purpose:** To test if the ReservationController calls correctly the CostManager every time the OnBoard application sends the information about the last ride, in order to obtain the related cost.
- **Input specification:** The ReservationController receives a request from the EventRouter.
- **Output specification:** The ReservationController receives the final cost for the last ride.
- **Environmental needs:** EventRouter Driver, Test Database filled with data concerning the specific Test Case.

3.10 Integration Test Case I1.10

- **Test case identifier:** I1.10
- **Test items:** ReservationController, Maps API
- **Purpose:** ReservationController uses Maps API to let the client visualize the list of available cars near his/her position in a map and to calculate the distance between the client and each car. The purpose is to test the external calls to the service.
- **Input specification:** ReservationController sends to MapsAPI the collection of coordinates of the available cars in order to find the shortest path, and the client's coordinates.
- **Output specification:** A collection of points representing a subset of the input one ordered from the nearest point to the furthest one.
- **Environmental needs:** MapsAPI Stub, Test Database filled with data concerning the specific Test Case.

3.11 Integration Test Case I1.11

- **Test case identifier:** I1.11
- **Test items:** ReservationController, Payment Gateway
- **Purpose:** When a rent ends, the ReservationController starts the procedure to apply the fee; the purpose is to test the correctness of the call.
- **Input specification:** The ReservationController sends to the PaymentGateway all the information needed to complete the transaction (client's payment information, amount to pay).
- **Output specification:** Change of data in the Database of PowerEnJoy; warning that the payment failed.
- **Environmental needs:** PaymentGateway Stub, Test Database filled with data concerning the specific Test Case.

3.12 Integration Test Case I2.1

- **Test case identifier:** I2.1
- **Test items:** EventRouter, UserController
- **Purpose:** The client can see/modify his/her data in the profile. The purpose is to test the correct interaction with the UserController and the effectiveness of changes requested by the user.
- **Input specification:** The EventRouter receives a request from the UserUI.
- **Output specification:** Change of data in the Database of PowerEnJoy; positive/negative answer about the conclusion of the operation.
- **Environmental needs:** UserUI Driver, Test Database filled with data concerning the specific Test Case.

3.13 Integration Test Case I2.2

- **Test case identifier:** I2.2
- **Test items:** EventRouter, AuthController
- **Purpose:** The purpose is to test the login process executed by a registered client or by a System Administrator, who receives the access to PowerEnJoy service if and only if he/she inserts the correct credentials.
- **Input specification:** The EventRouter receives a requests from the UserUI or the SystemAdminUI.
- **Output specification:** The system grants the access if the credentials corresponds to the saved ones in the Database of PowerEnJoy.
- **Environmental needs:** UserUI Driver, SystemAdminUI Driver, Test Database filled with data concerning the specific Test Case.

3.14 Integration Test Case I2.3

- **Test case identifier:** I2.3
- **Test items:** EventRouter, SystemAdminController

- **Purpose:** To test if the communication between the EventRouter and the SystemAdminController works correctly. The EventRouter calls methods of the SystemAdminController every time the System Administrator performs a request on his application. For example when the SystemAdministrator wants to change the status of a car in the Database or when he/she has to check the distribution of the cars.
- **Input specification:** the EventRouter receives a request from the SystemAdminUI.
- **Output specification:** The expected output is a positive response if the number of cars in the city is higher than the minimum acceptable (previously set) and if the car are correctly distributed along the city. In the other case the output, according to the request performed by the System Administrator, can be: a list of the plates of the cars with too low battery to be available and their positions; a list of area in which there are not many cars available and the plates of the cars; a list of area with a great number of cars and the plates of the cars.
- **Environmental needs:** SystemAdminUI Driver, CarDistributionManager, CarStatusManager, Test Database filled with data concerning the specific Test Case.

3.15 Integration Test Case I2.4

- **Test case identifier:** I2.4
- **Test items:** EventRouter, ReservationController
- **Purpose:** The EventRouter dispatches several events to ReservationController, that is the most important component of the system of PowerEnJoy. The purpose is to do a test for each scenario associated to a client's action, for example the requests may be : provide a set of car near a fixed point; reserve a selected car; cancelation of a previous reservation; start a rent; terminate a rent.
- **Input specification:** The EventRouter receives a request from the UserUI or from the OnBoardUI.
- **Output specification:** depending on the kind of request performed the output may be:
  - positive/negative answer about the conclusion of the operation;

  - set of positions of cars near the client's position;

- **Environmental needs:** UserUI Driver, OnBoardUI Driver, NotificationManager Stub, CarDistributionManager, CostManager, CarStatusManager, MapsAPI Stub, PaymentGateway Stub, Test Database filled with data concerning the specific Test Case.

3.16 Integration Test Case I2.5

- **Test case identifier:** I2.5
- **Test items:** SMSReservationManager, ReservationController
- **Purpose:** To check if the ReservationController is properly called from the SMSReservationManager when a client makes a reservation through an SMS. The SMSReservationManager calls the ReservationController: when the client expresses his/her intention to rent a car, when a client decides to cancel his/her previous request of reservation, when the client asks to unlock the rented car.
- **Input specification:** The SMSReservationManager receives a request from the SMSGateway.
- **Output specification:** positive/negative answer about the conclusion of the operation.
- **Environmental needs:** SMSGateway Driver, NotificationManager Stub, CarDistributionManager, CostManager, CarStatusManager, PaymentGateway Stub, Test Database filled with data concerning the specific Test Case.

3.17 Integration Test Case I3.1

- **Test case identifier:** I3.1
- **Test items:** OnBoardUI, EventRouter
- **Purpose:** To test the communication between the OnBoardUI and the EventRouter. The OnBoardUI calls the EventRouter every time the client does an action on the OnBoard application. Some actions may be for example: the insertion of the pin code, the choice to terminate the rent, to enable the 'save money' option or to choose the 'Pit Stop Mode'.

- **Input specification:** Actions performed by the client on the interface of the on board application.
- **Output specification:** Push notification on the interface, containing the positive/negative result of the operation.
- **Environmental needs:** OnBoardUI Driver, AuthController, ReservationController, CarDistributionManager, CarStatusManager, CostManager, MapsAPI Stub, PaymentGateway Stub.

3.18 Integration Test Case I3.2

- **Test case identifier:** I3.2
- **Test items:** SystemAdminUI, EventRouter
- **Purpose:** To test if the SystemAdminUI calls correctly the EventRouter every time the System Administrator makes a request through his/her application. Some of these requests are for example login/logout, changing the status of a car in the Database or checking the distribution of the cars.
- **Input specification:** Actions performed by the System Administrator on the interface of his/her web application.
- **Output specification:** The expected output is a positive response if the number of cars in the city is higher than the minimum acceptable (previously set) and if the car are correctly distributed along the city. In the other case the output, according to the request performed by the System Administrator, can be: a list of the plates of the cars with too low battery to be available and their positions; a list of area in which there are not many cars available and the plates of the cars; a list of area with a great number of cars and the plates of the cars.
- **Environmental needs:** SystemAdminUI Driver, SystemAdminController, AuthController, CarDistributionManager, CarStatusManager, Test Database filled with data concerning the specific Test Case.

3.19 Integration Test Case I3.3

- **Test case identifier:** I3.3

- **Test items:** UserUI, EventRouter
- **Purpose:** To test the communication between the UserUI and the EventRouter every time the UserUI needs to call the EventRouter. This happens every time a client does an action on his/her interface of the application. Some of these actions are: registration, login/logout, change the information of the account, make/cancel a reservation.
- **Input specification:** Actions performed by the client on the interface of his/her web or mobile application.
- **Output specification:** notification on the interface of the web/mobile application as result of the request performed.
- **Environmental needs:** ClientUI Driver, UserController, AuthController, ReservationController, DrivingLincenceAPI Stub, NotificationManager Stub, CarDistributionManager, CarStatusManager, CostManager, MapsAPI Stub, PaymentGateway Stub.

Chapter 4

Tools and Test Equipment Required

This section describes tools and techniques required in order to do all tests listed in the previous sections.

4.1 Tools

- **Mocha.js:** it's a JavaScript test framework running on NodeJS, featuring browser support, asynchronous testing, test coverage reports, and use of any assertion library. The NodeJS concurrency model makes writing integration tests simple using this test framework. The key idea is that mocha tests can start and stop a NodeJS server. Once your mocha tests start a server, the same process can start a client to send requests to NodeJS server without any need of messy multithreading.
- **Sinon.js:** it's a standalone test spies, stubs and mocks for JavaScript. No dependencies are required and works perfectly with Mocha.js.
- **Chai.js:** it's an assertion library , a tool to verify that things are correct. This makes it a lot easier to test the code, so you don't have to do thousands of if statements.
- **Nightwatch.js:** It's an easy to use End-to-End testing tool for browser-based apps. This is useful to automate tests of web GUIs.
- **Calabash** it's a cross-platform, supporting native Android and Ios apps, testing framework to automate tests of mobile app GUIs.

- **Manual testing:** some components of the system, like GUIs, are difficult to test programmatically and need manual testing. Nightwatch and Calabash are good during the development process, in order to detect in early stages bugs in user interfaces. Before deploying to the production environment a human test is mandatory in order to have feedbacks and possible improvements of the user experience.

4.2 Testing environment

Tests are going to be executed in a particular environment physically different from the production one: the aim is to test each functionality and its correctness and, in future releases, to verify that no regressions have risen up. For this reason the testing environment has to be as much as possible similar to the production one from the point of view of data, in order to recreate the most realistic situations. Stubs must be implemented considering different scenarios, providing all possible results of the functions involved into processes of the analysis: in other words, stubs must provide all the inputs/outputs accepted by the domain of PowerEnJoy.

Chapter 5

Program Stubs, Drivers and Test Data Required

5.1 Program Stubs

5.1.1 Driving licence API

- **Test involved:** I1.1,I3.3
- **Description:** It provides the validity of a driving licence inserted by client in PowerEnJoy System

5.1.2 NotificationManager

- **Test involved:** I1.2, I1.3, I1.5, I2.4, I3.3
- **Description:** Returns the confirmation about the data transmission of the notification(s)

5.1.3 Maps API

- **Test involved:** I1.10, I2.4, I3.1, I3.3
- **Description:** It provides data about distances, locations, shortest path about a set given in input

5.1.4 Payment Gateway

- **Test involved:** I1.11, I2.4, I2.5, I3.1, I3.3
- **Description:** It returns the result of the payment procedure

5.2 Program Drivers

5.2.1 OnBoard UI

- **Test involved:** I2.4, I3.1
- **Description:** It provides data about all the possible requests that a client can do during a rent, since the beginning and It also provides data about the ride

5.2.2 SystemAdmin UI

- **Test involved:** I2.2, I2.3, I3.2
- **Description:** It simulates the system administrator's actions

5.2.3 ClientUI

- **Test involved:** I3.3
- **Description:** It simulates the client's actions

5.2.4 EventRouter

- **Test involved:** I1.1, I1.2, I1.3, I1.4, I1.5, I1.6, I1.7, I1.8, I1.9
- **Description:** Depending on the test involved, the driver provides the correct message to the target controller

5.2.5 SMSGateway Driver

- **Test involved:** I2.5
- **Description:** It simulates the receiving of SMSs from the client

5.2.6 SMSReservationManager

- **Test involved:** I1.8
- **Description:** It provides the essential data in order to finalize a reservation

5.3 Test Data Required

In order to test correctly most of the components it is needed to have the databases populated with reasonable real data. This database will reside in the testing environment described in the section 4.2. In this way, the operations done during the tests do not change data in the production database.

Chapter 6

Used tools and working hours

6.1 Used tools

The tools used to create this document are the following:

- MikTex 2.9: to format the document using LaTex.
- Visual paradigm: to create diagrams.

6.2 Working hours

	Alessandro	Roberta	Giorgio
29/12	3	3	3
30/12			
31/12			
1/1			
2/1	1	1	2
3/1		1	1
4/1	1	1	
5/1		1	2
6/1	2		
7/1	1	1	1
8/1		1	2
9/1	1		
10/1	1	2	
11/1			2
12/1	3	3	3
13/1	2	2	2
Total	15	16	18