Software Engineering 2: PowerEnJoy

Integration Test Plan Document (ITPD)

Version 1.0

Politecnico di Milano, A.A. 2016/2017

Agosti Isabella, 874835
Cattivelli Carolina, 879359

January 15, 2017

Contents

1 INTRODUCTION **4**
- 1.1 Revision History . . . 4
- 1.2 Purpose and Scope . . . 4
- 1.3 List of Definitions and Abbreviations . . . 5
  - 1.3.1 Definitions . . . 5
  - 1.3.2 Acronyms and Abbreviations . . . 6
- 1.4 List of Reference Documents . . . 6

2 INTEGRATION STRATEGY **7**
- 2.1 Entry Criteria . . . 7
- 2.2 Elements to be Integrated . . . 7
- 2.3 Integration Testing Strategy . . . 8
- 2.4 Sequence of Component/Function Integration . . . 8
  - 2.4.1 Software Integration Sequence . . . 8
  - 2.4.2 Subsystem Integration Sequence . . . 14

3 INDIVIDUAL STEPS AND TEST DESCRIPTION **15**
- 3.1 User Manager System . . . 15
  - 3.1.1 User: Registration by filling out a form . . . 15
  - 3.1.2 User: First login . . . 16
- 3.2 Registered User Manager System . . . 17
  - 3.2.1 Registered User: Login . . . 17
  - 3.2.2 Registered User: View profile . . . 18
  - 3.2.3 Registered User: Manage personal information . . . 19
  - 3.2.4 Registered User: Cancel current reservation . . . 20
  - 3.2.5 Registered User: Report an issue after unlocking the car 21
- 3.3 Employee Manager System . . . 23
  - 3.3.1 Employee: Manage car's information . . . 23
- 3.4 Ride Manager System . . . 24
  - 3.4.1 Registered User: Make reservation . . . 24

4 TOOLS AND TEST EQUIPMENT REQUIRED **26**
- 4.1 Tools . . . 26
- 4.2 Test Equipment . . . 27

5 PROGRAM STUBS AND TEST DATA REQUIRED **28**
5.1 Program Stubs and Drivers 28
5.2 Test Data 29

6 EFFORT SPENT **31**
6.1 Agosti Isabella 31
6.2 Cattivelli Carolina 31

1 INTRODUCTION

1.1 Revision History

Version	Date	Author(s)	Summary
1.0	15/01/17	Isabella Agosti, Carolina Cattivelli	Initial release

1.2 Purpose and Scope

The purpose of this document is to describe how we plan to accomplish the integration test, listing all the tests that will be performed on the PowerEnJoy application.

In particular we will describe how the tests will be executed, which components will be tested and in which order, making sure that the requirements listed in the *RASD* are fulfilled without exhibiting unexpected behaviors.

1.3 List of Definitions and Abbreviations

1.3.1 Definitions

Keyword	Definitions
User	A person that interacts with the PowerEnJoy mobile or web application to register to the system.
Registered user	A person who already registered to the system, that interacts with the PowerEnJoy mobile or web application in various ways.
Employee	A member of the PowerEnJoy staff.
Car-sharing service	Model of car rental where people rent cars for short periods of time, often by the hour.
Electric car	Automobile that is propelled by one or more electric motors, using electrical energy stored in rechargeable batteries.
Registration	The act or process of filling out an online form providing credentials and payment information.
Log-in	Process by which a user gains access to the system by identifying and authenticating himself/herself.
Reservation	Arrangement through which a registered user holds a car for his use at a later time.
Safe area	Area whose position is predefined by the management system. Safe areas are the only ones in which a user is allowed to park a car.
Special safe area	Special type of safe area where a car can be recharged.
Discount percentage	Discount applied on the users last ride only in certain circumstances.
Low battery	The cars battery level is considered low when less than 20%.

1.3.2 Acronyms and Abbreviations

Acronym/Abbreviation	Definition
RASD	Requirements Analysis and Specification Document
DD	Design Document
ITPD	Integration Test Plan Document
AA	Anno Accademico (Academic Year)
DB	Database

1.4 List of Reference Documents

- The project description document: *Specifications document: Assignments AA 2016-2017.pdf.*
- The PowerEnJoy Requirements Analysis and Specification Document: *RASD.pdf.*
- The PowerEnJoy Design Document: *DD.pdf.*
- The Integration Test Plan Document example: *Integration testing example document.pdf.*

2 INTEGRATION STRATEGY

2.1 Entry Criteria

In order for the integration testing to be possible and to produce meaningful results, there are some criteria that must be met.

First of all, *RASD* and *DD* must have been fully written. This is required in order to have a complete overview of the interactions between the different components of the system and of the functionalities they offer.

Secondly, all the application's functionalities and components must be able to correctly interact with one another.

2.2 Elements to be Integrated

The components that need to be integrated, referring to our *DD*, are:

- **Client**, that provides a **User Interface** and a connection between client and server through the **Connection Manager**.
- **Employee Manager**, that provides an **Account Manager** for the login process and a **Car Manager** to manage the information of the company's cars.
- **User Manager**, that provides a **Registration Manager**, **Notification Manager** and **Email Gateway** for the registration process and an **Account Generator** for the first login.
- **Registered User Manager**, that provides an **Account Manager** for the login, view profile, manage personal information, view promotions, view reservations list functionalities, and a **Reservation Manager** to cancel a reservation.
- **Ride Manager**, that provides a **Reservation Manager** to make a reservation, a **Car Manager** to generate a map with the available cars, a **Reservation Generator** to create a reservation, a **Notification Manager** and **Email Gateway** to send a notification in case a car is abandoned for more than ten minutes.

2.3 Integration Testing Strategy

We will adopt a bottom-up testing strategy, integrating first the subcomponents and then the higher level components.

Using this approach, we will start integrating together those components that do not depend on others, or that only depend on already developed components.

Doing so, we can start performing the integration testing as soon as the required components have been developed, in order to maximize parallelism and efficiency.

2.4 Sequence of Component/Function Integration

Since we chose to adopt a bottom-up strategy, we will start from the lower level component and then move upward.

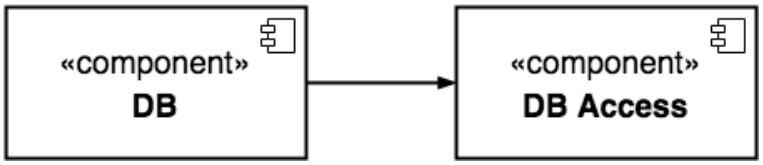
As a notation, an arrow going from component C1 to component C2 means that C1 is necessary for C2 and so it must have already been implemented.

2.4.1 Software Integration Sequence

In this section we identify the sequence in which the software components will be integrated within each subsystem.

Database Access System

The first two components to be integrated are the **Database** and the **Database Access**. We start from here because every other component relies on the **Database Access System** to perform queries.

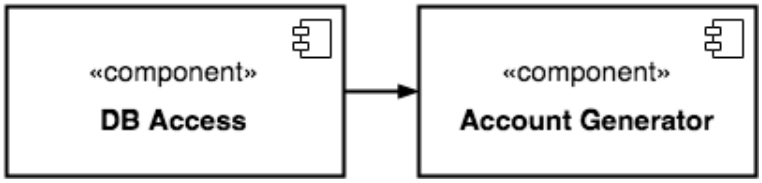


User Manager System

Since the user is the main actor of our application, the second step in the integration process is to appropriately connect the subcomponents implementing the **User Manager System**.

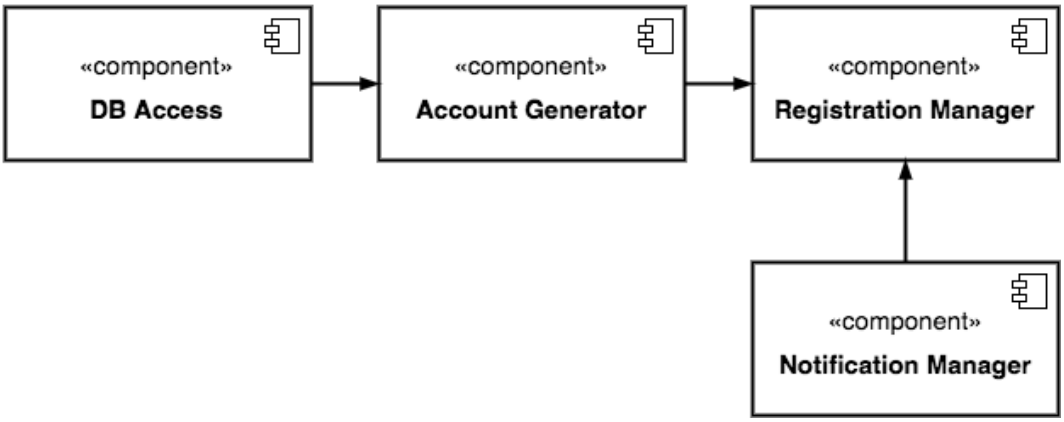
In the following diagrams, we show which components are going to be integrated together and in which order, following the bottom-up approach.

First, we proceed by integrating together the **Account Generator** subcomponent with the **Database Access System**.



Then, we proceed by integrating together the **Registration Manager** subcomponent with the **Database Access System** and the **Notification Manager**.

At this point, the two subcomponents of the **User Manager System** are ready to be integrated together.

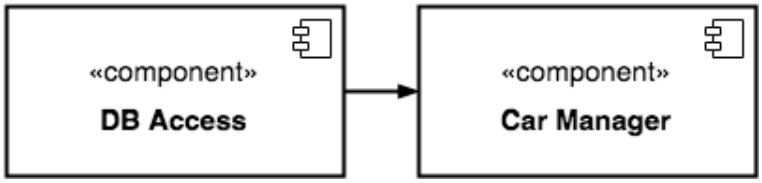


Registered User Manager System

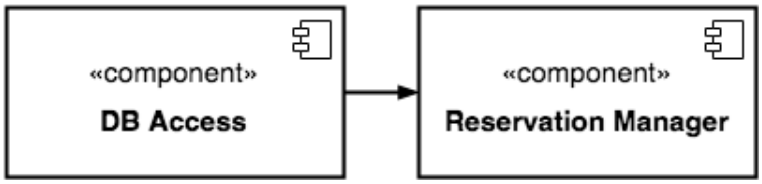
The third step in the integration process is to appropriately connect the subcomponents implementing the **Registered User Manager System**.

In the following diagrams, we show which components are going to be integrated together and in which order, following the bottom-up approach.

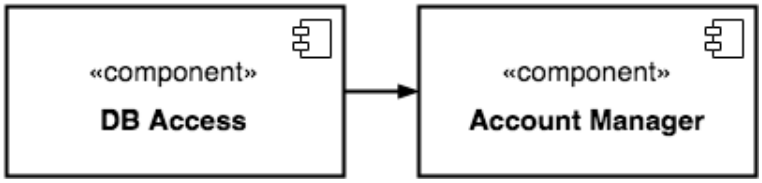
First, we proceed by integrating together the **Car Manager** subcomponent with the **Database Access System**.



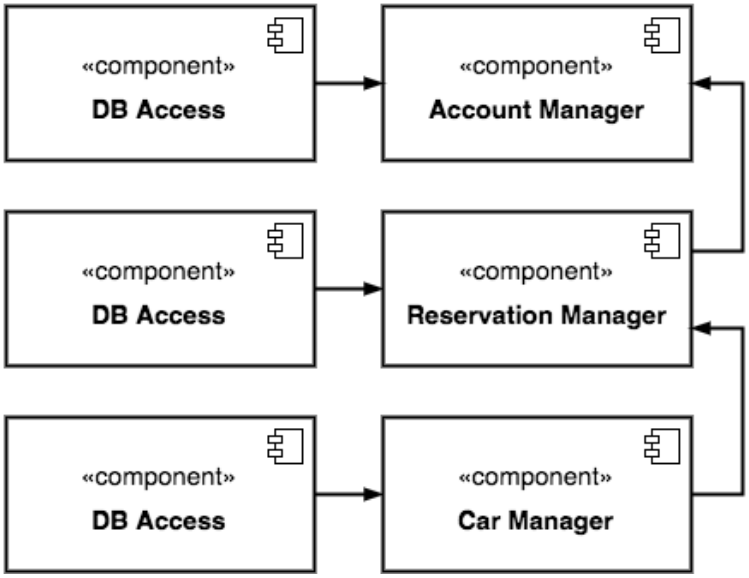
Then, we proceed by integrating together the **Reservation Manager** subcomponent with the **Database Access System**.



Then, we proceed by integrating together the **Account Manager** subcomponent with the **Database Access System**.



At this point, the three subcomponents of the **Registered User Manager System** are ready to be integrated together.

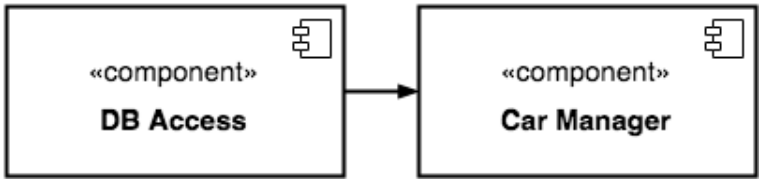


Employee Manager System

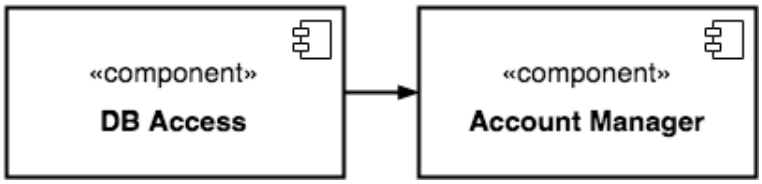
The fourth step in the integration process is to appropriately connect the subcomponents implementing the **Employee Manager System**.

In the following diagrams, we show which components are going to be integrated together and in which order, following the bottom-up approach.

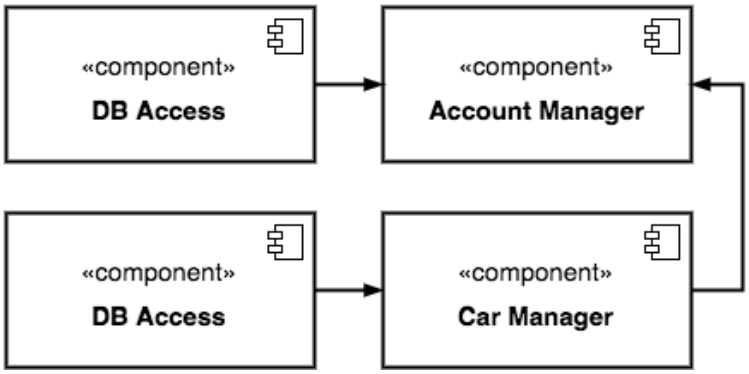
First, we proceed by integrating together the **Car Manager** subcomponent with the **Database Access System**.



Then, we proceed by integrating together the **Account Manager** subcomponent with the **Database Access System**.



At this point, the two subcomponents of the **Employee Manager System** are ready to be integrated together.

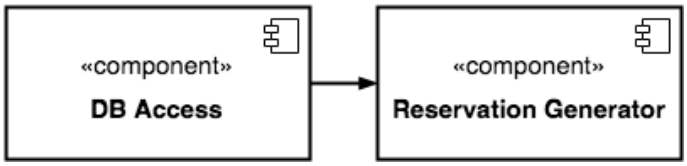


Ride Manager System

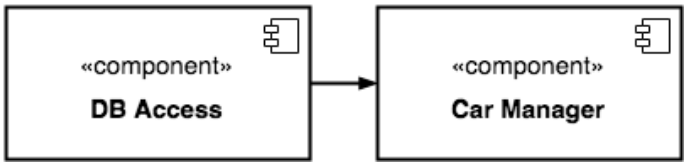
The last step in the integration process is to appropriately connect the subcomponents implementing the **Ride Manager System**.

In the following diagrams, we show which components are going to be integrated together and in which order, following the bottom-up approach.

First, we proceed by integrating together the **Reservation Generator** subcomponent with the **Database Access System**.

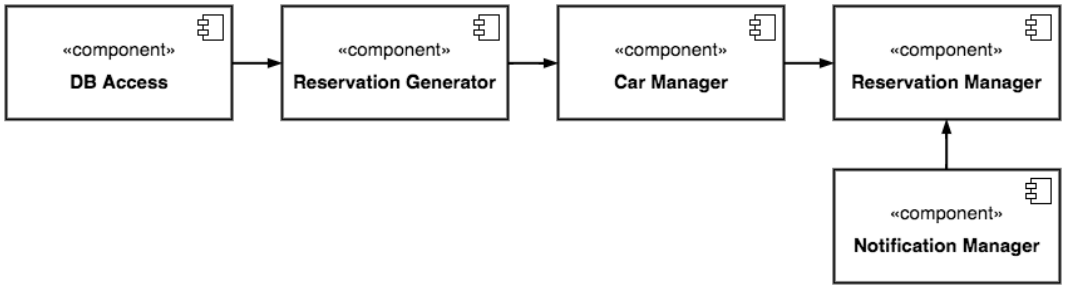


Then, we proceed by integrating together the **Car Manager** subcomponent with the **Database Access System**.



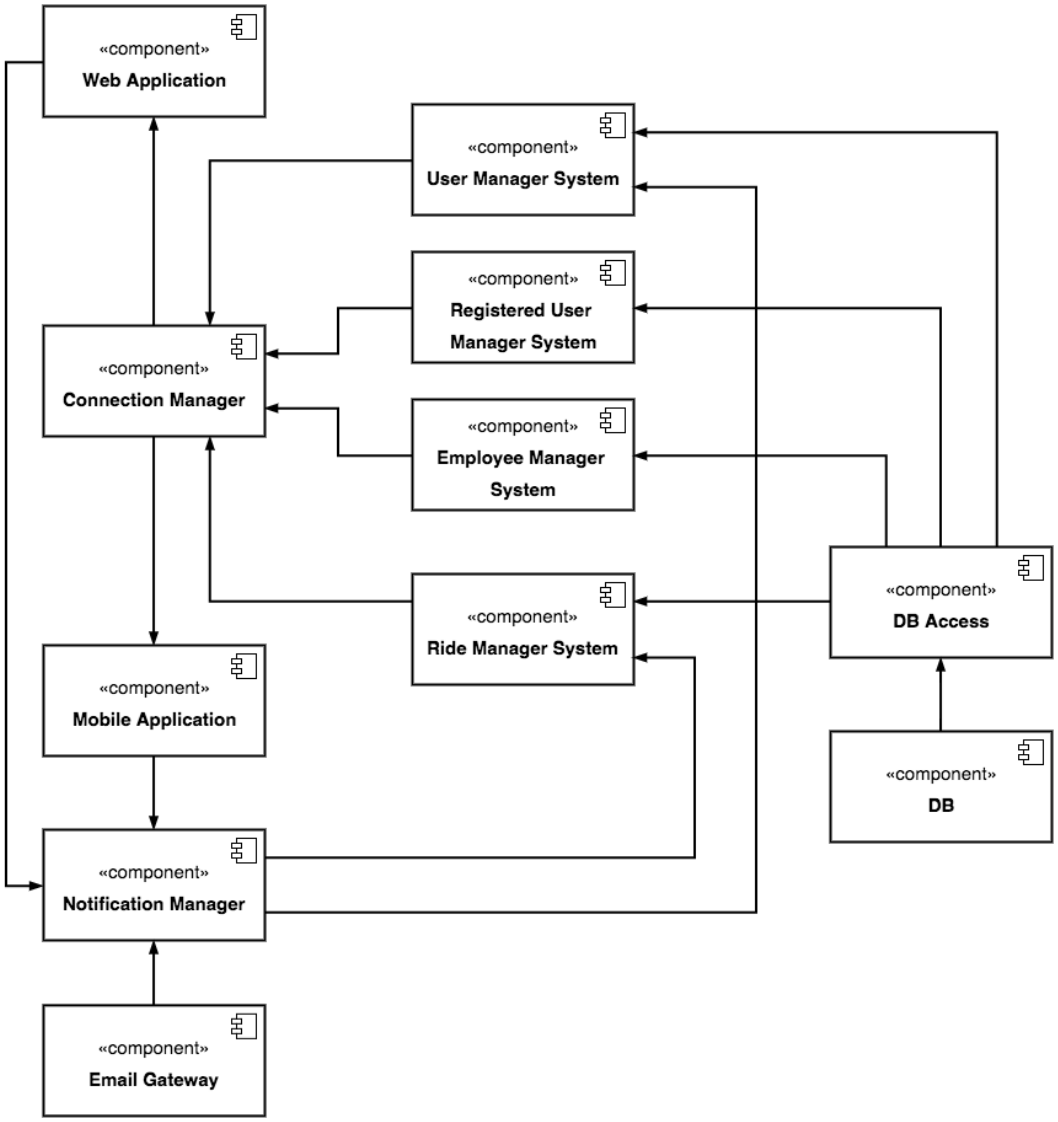
Then, we proceed by integrating together the **Reservation Manager** subcomponent with the **Database Access System** and the **Notification Manager**.

At this point, the three subcomponents of the **Ride Manager System** are ready to be integrated together.



2.4.2 Subsystem Integration Sequence

In the following diagram we present the order in which subsystems will be integrated to create the full PowerEnJoy infrastructure.



3 INDIVIDUAL STEPS AND TEST DESCRIPTION

In this chapter we describe, for each step of the integration process presented before, the type of tests that will be used to verify that the elements integrated in the step perform as expected.

3.1 User Manager System

3.1.1 User: Registration by filling out a form

Test Case ID	#1
Test Case Name	User: Registration by filling out a form
Components Involved	Client Manager, Notification Manager, Registration Manager, Account Generator, DB Access, DB
Precondition	In case the user wants to use the mobile application, he/she needs to download it first.
Main Path	1. The **Client Manager** sends a registration request to the **Registration Manager**. 2. The **Registration Manager** sends back the registration form. 3. Once the form is filled, the **Client Manager** sends it to the **Registration Manager**. 4. The **Registration Manager** verifies the data and, if correct, requests the creation of a user to the **Account Generator**. 5. The **Account Generator** creates the new account and requests access to the **DB Access** to add the new user. 6. The **DB Access** adds the user to the **DB**.
Expected Result	The user receives a password via email and is able to see the website homepage.

3.1.2 User: First login

Test Case ID	#2
Test Case Name	User: First login
Components Involved	Client Manager, Registration Manager, DB Access, DB
Precondition	The user has previously registered to the system.
Main Path	1. Once the login form is filled, the **Client Manager** sends it to the **Registration Manager**. 2. The **Registration Manager** requests access to the **DB Access**. 3. The **DB Access** formulates a query and sends it to the **DB**. 4. After receiving the query response from the **DB**, the **Registration Manager** verifies if the credentials are correct and, if so, logs the user in.
Expected Result	The user is granted the access and is able to see his/her personal page.

3.2 Registered User Manager System

3.2.1 Registered User: Login

Test Case ID	#3
Test Case Name	Registered User: Login
Components Involved	Client Manager, Account Manager, DB Access, DB
Precondition	The user has previously completed the registration process and made the first login.
Main Path	1. Once the login form is filled, the **Client Manager** sends it to the **Account Manager**. 2. The **Account Manager** requests access to the **DB Access**. 3. The **DB Access** formulates a query and sends it to the **DB**. 4. After receiving the query response from the **DB**, the **Account Manager** verifies if the credentials are correct and, if so, logs the user in.
Expected Result	The registered user is granted the access and is able to see his/her personal page.

3.2.2 Registered User: View profile

Test Case ID	#4
Test Case Name	Registered User: View profile
Components Involved	Client Manager, Account Manager, DB Access, DB
Precondition	The registered user is logged into the system.
Main Path	1. The **Client Manager** sends a view profile request to the **Account Manager**. 2. The **Account Manager** requests access to the **DB Access**. 3. The **DB Access** formulates a query and sends it to the **DB**. 4. After receiving the query response from the **DB**, the **Account Manager** generates the profile page.
Expected Result	The registered user is able to see his/her profile page.

3.2.3 Registered User: Manage personal information

Test Case ID	#5
Test Case Name	Registered User: Manage personal information
Components Involved	Client Manager, Account Manager, DB Access, DB
Precondition	The registered user is logged into the system.
Main Path	1. The **Client Manager** shows the registered user his/her personal page. 2. The registered user applies some changes to his/her information and, once he is finished, the **Client Manager** sends a save changes request to the **Account Manager**. 3. The **Account Manager** verifies the correctness of the registered user input. If it is correct, the **Account Manager** requests access to the **DB Access**. 4. The **DB Access** updates the information in the **DB**.
Expected Result	The registered user's personal information is updated and he/she is able to see his/her profile page.

3.2.4 Registered User: Cancel current reservation

Test Case ID	#6
Test Case Name	Registered User: Cancel current reservation
Components Involved	Client Manager, Account Manager, Reservation Manager, Car Manager, DB Access, DB
Precondition	The registered user is logged into the system.
Main Path	1. The **Client Manager** sends a view reservations list request to the **Account Manager**. 2. The **Account Manager** requests access to the **DB Access**. 3. The **DB Access** formulates a query and sends it to the **DB**. 4. After receiving the query response from the **DB**, the **Account Manager** verifies the registered user reservations existence. If there exist at least one reservation, the **Account Manager** generates the reservations page. 5. The **Client Manager** sends a cancel current reservation request to the **Reservation Manager**. 6. The **Reservation Manager** requests access to the **DB Access**. 7. The **DB Access** formulates a query and sends it to the **DB**. 8. After receiving the query response from the **DB**, the **Account Manager** verifies the existence of the registered user's current reservations. If it exists, the **Reservation Manager** sends a cancel current reservation request to the **Car Manager**.

Main Path	9. The **Car Manager** requests access to the **DB Access**. 10. The **DB Access** cancels the registered user's current reservation from the **DB**. 11. After receiving the query response from the **DB**, the **Car Manager** updates the car's information.
Expected Result	The registered user's current reservation no longer exists and he/she is able to see his/her past reservations' list with the current reservation labeled as "Deleted".

3.2.5 Registered User: Report an issue after unlocking the car

Test Case ID	#7
Test Case Name	Registered User: Report an issue after unlocking the car
Components Involved	Client Manager, Account Manager, Reservation Manager, Car Manager, DB Access, DB
Precondition	The registered user has previously logged into the system, made a reservation and selected a car.
Main Path	1. The **Client Manager** sends a view reservations list request to the **Account Manager**. 2. The **Account Manager** requests access to the **DB Access**. 3. The **DB Access** formulates a query and sends it to the **DB**.

Main Path	4. After receiving the query response from the **DB**, the **Account Manager** verifies the registered user reservations existence. If there exist at least one reservation, the **Account Manager** generates the reservations page. 5. The **Client Manager** sends a unlock car in current reservation request to the **Car Manager**. 6. The **Car Manager** unlocks the car and the **Account Manager** generate the car's information page. 7. The **Client Manager** sends a view reservations list request to the **Account Manager**. 8. The **Account Manager** sends a cancel current reservation request to the **Reservation Manager**. 9. The **Reservation Manager** requests access to the **DB Access**. 10. The **DB Access** cancels the registered user's current reservation from the **DB**. 11. After receiving the query response from the **DB**, the **Car Manager** updates the car's information.
Expected Result	The registered user's current reservation no longer exists and he/she is able to see his/her past reservations' list with the current reservation labeled as "Deleted".

3.3 Employee Manager System

3.3.1 Employee: Manage car's information

Test Case ID	#8
Test Case Name	Employee: Manage car's information
Components Involved	Client Manager, Car Manager, DB Access, DB
Precondition	The employee is logged into the system.
Main Path	1. The **Client Manager** sends a get car's information request to the **Car Manager**. 2. The **Car Manager** requests access to the **DB Access**. 3. The **DB Access** formulates a query and sends it to the **DB**. 4. After receiving the query response from the **DB**, the **Car Manager** (after verifying the plates existence in case the employee entered a plate) sends the selected car's information to the **Client Manager**. 5. Once the employee updates the car's information, the **Client Manager** sends save changes request to the **Car Manager**. 6. The **Car Manager** verifies the employee's input and, if correct, requests access to the **DB Access**. 7. The **DB Access** updates the car's information in the **DB**.
Expected Result	The cars information is changed and the employee visualizes the list of all PowerEnJoy cars.

3.4 Ride Manager System

3.4.1 Registered User: Make reservation

Test Case ID	#9
Test Case Name	Registered User: Make reservation
Components Involved	Client Manager, Account Manager, Reservation Manager, Car Manager, Reservation Generator, DB Access, DB
Precondition	The registered user is logged into the system.
Main Path	1. The **Client Manager** sends a make reservation request to the **Car Manager**. 2. The **Car Manager** generates a basic map and sends it to the **Client Manager**. 3. The **Client Manager** sends either a track my position request or an insert specific address request to the **Account Manager**. 4. The **Account Manager** verifies either the registered user's GPS or the specific address inserted and, if correct, adds the available cars to the basic map and sends it to the **Client Manager**. 5. The **Client Manager** sends either an exit request to the **Account Manager** which generates the registered user's personal page, or a select a car request to the **Car Manager**. 6. The **Car Manager** sends a generate reservation request to the **Reservation Generator**. 7. The **Reservation Generator** requests access to the **DB Access**.

Main Path	8. The **DB Access** adds the reservation to the **DB**. 9. After receiving the query response from the **DB**, the **Car Manager** updates the car's information.
Expected Result	The registered user is able to see the page with all his previous and current reservations.

4 TOOLS AND TEST EQUIPMENT REQUIRED

In this paragraph we identify all tools and test equipment needed to accomplish the integration.

4.1 Tools

In order to test the various components of PowerEnJoy more effectively, we are going to make usage of these automated testing tools:

- **Manual testing**, to test the website and the mobile application.
- **Arquillian**, that allows us to execute test cases against Java containers.
  http://arquillian.org
- **Mockito**, that allows us to abstract dependencies and have predictable results, and also check that the interaction between the tester and the mock is correct. We are going to use it to mock stubs for the components to test.
  http://site.mockito.org
- **JUnit**, that is a simple framework to write repeatable tests. We are going to use it for unit tests on the single components and also for integration testing together with Mockito and Arquillian.
  http://junit.org/junit4/

4.2 Test Equipment

For what concerns the mobile side of the testing environment, the following devices are required:

- At least one iOS smartphone for each member of the iOS smartphone family.
- At least one iOS tablet for each member of the iOS tablet family.
- At least one Android smartphone for each member of the Android smartphone family.
- At least one Android tablet for each member of the Android tablet family.
- At least one Windows smartphone for each member of the Windows smartphone family.

The web application will be tested using normal desktop and notebook computers. There are no specific requirements on display resolution nor operating system. The only features required are: Flash Player 11.2 or later, modern browser with AJAX support and Ethernet or Wi-Fi connection.

5 PROGRAM STUBS AND TEST DATA REQUIRED

In this paragraph we identify any program stubs and special test data required for each integration step based on the testing strategy and test design.

5.1 Program Stubs and Drivers

As we have mentioned in the Integration Testing Strategy section of this document [2], we are going to adopt a bottom-up approach for components integration and testing.

Because of this choice, here follows a list of all the drivers that will be developed to actually perform the necessary method invocations on the components to be tested, together with their specific role.

- **DB Access Driver**, that will invoke the methods exposed by the **Database Access System** component in order to test its interaction with the **DB**.
- **Account Generator Driver**, that will invoke the methods exposed by the **Account Generator** component, in order to test its interaction with the **DB Access** component.
- **Registration Manager Driver**, that will invoke the methods exposed by the **Registration Manager** component, in order to test its interaction with the **DB Access**, the **Notification Manager** and the **Account Generator** components.
- **Car Manager Driver**, that will invoke the methods exposed by the **Car Manager** component, in order to test its interaction with the **DB Access** and the **Reservation Generator** components.
- **Reservation Manager Driver**, that will invoke the methods exposed by the **Reservation Manager** component, in order to test its interaction with the **DB Access**, the **Notification Manager** and the **Car Manager** components.
- **Account Manager Driver**, that will invoke the methods exposed by the **Account Manager** component, in order to test its interaction with the **DB Access**, the **Car Manager** and the **Reservation Manager** components.

- **Reservation Generator Driver**, that will invoke the methods exposed by the **Reservation Generator** component, in order to test its interaction with the **DB Access** component.

Since the entire system is not yet fully developed, in order to perform integration testing we need to use stubs to simulate the presence of clients until they are fully developed.

5.2 Test Data

In order to be able to perform the set of tests that we have specified, we are going to need:

- A list of both valid and invalid candidate users to test the **Registration Manager** and the **Account Generator** components. The set should contain instances exhibiting the following problems:
  - Null object.
  - Null fields.
  - Username already taken.
  - Driver license not compliant with the legal format.
  - Invalid card code.
  - Invalid card number.
  - Invalid email.
- A list of both valid and invalid candidate registered users and employees to test the **Account Manager** component. The set should contain instances exhibiting the following problems:
  - Null object.
  - Null fields.
  - Invalid username/password.
  - Username already taken.
  - Driver license not compliant with the legal format.
  - Invalid card code.
  - Invalid card number.
  - Invalid email.
  - Invalid address.

  - GPS not active.

- A list of both valid and invalid candidate registered users and employees to test the **Reservation Manager** component. The set should contain instances exhibiting the following problems:

  - Null object.
  - No current or past reservations.

- A list of both valid and invalid candidate registered users and employees to test the **Car Manager** component. The set should contain instances exhibiting the following problems:

  - Null object.
  - Null fields.
  - Invalid plate.
  - Invalid values entered into the fields.
  - No car selected.

6 EFFORT SPENT

This section includes information about the number of hours each group member has worked towards the fulfillment of this deadline.

Since we decided to work together every day, the worked hours are going to be the same for each group member. We think this is the best way to achieve good results.

6.1 Agosti Isabella

- 22/12/2016: 1h
- 05/01/2017: 3h
- 09/01/2017: 2h
- 12/01/2017: 1h
- 13/01/2017: 3h

6.2 Cattivelli Carolina

- 22/12/2016: 1h
- 05/01/2017: 3h
- 09/01/2017: 2h
- 12/01/2017: 1h
- 13/01/2017: 3h