Software Engineering 2: "PowerEnJoy"
Integration Test Plan (V. 1.1)

Authors:
Sergio CAPRARA
Soheil GHANBARI
Erica TINTI

Milan, Italy
15/01/2017

Table of contents

1. Introduction

1.1 Revision History

Version	Date	Authors	Description
1.0	15/01/2017	S. Caprara, S. Ghanbari, E. Tinti	First release
1.1	07/02/2017	S. Caprara, E. Tinti	Reference documents updated

1.2 Purpose and Scope

In this document, we are providing details on how the components described in the Design Document will be integrated. To ensure that the interaction between them will give the expected results, we are choosing the method to follow and we are keeping in mind that the Integration Test of a component will be done after having Unit Tested it.

In the following chapters, you will find detailed descriptions of the tests and the name of the tools to be used.

1.3 Definitions and Abbreviations

- **User:** the person registered to the system and allowed to access to its functions.
- **Operator:** a person with technical skills, that fixes car issues.
- **App:** short term used to define a mobile application.
- **Power Plug:** a column with one or more electricity socket where it is possible to charge the car.
- **Safe Area** (or Parking Area): a parking area with parking shared with all the other divers and not especially reserved to PowerEnjoy.
- **Special Parking Area** (or Power Station): a parking area reserved exclusively to PowerEnjoy cars where, for each parking space there is a Power Plug where it is possible to charge a car.
- **Car:** PowerEnjoy car.

- **Reservation:** the relation between a user and a car, that allows the user to start using the car. The reservation guarantees that no one else can reserve and use the reserved car till the end of the rental.
- **DB:** database, the collection of system data.
- **DAO:** Data Access Object.
- **Pojo:** Plain Old Java Object. Object having only getter and setter methods.

1.4 Reference Documents

The documents used as a reference to provide the design document are:

- Assignments AA 2016-2017.pdf
- Integration Test Plan Example.pdf
- Integration testing example document.pdf
- RASD_PowerEnjoy_Caprara_Ghanbari_Tinti_v1.1.pdf
- DesignDocument_PowerEnjoy_Caprara_Ghanbari_Tinti_v1.1.pdf

2. Integration Strategy

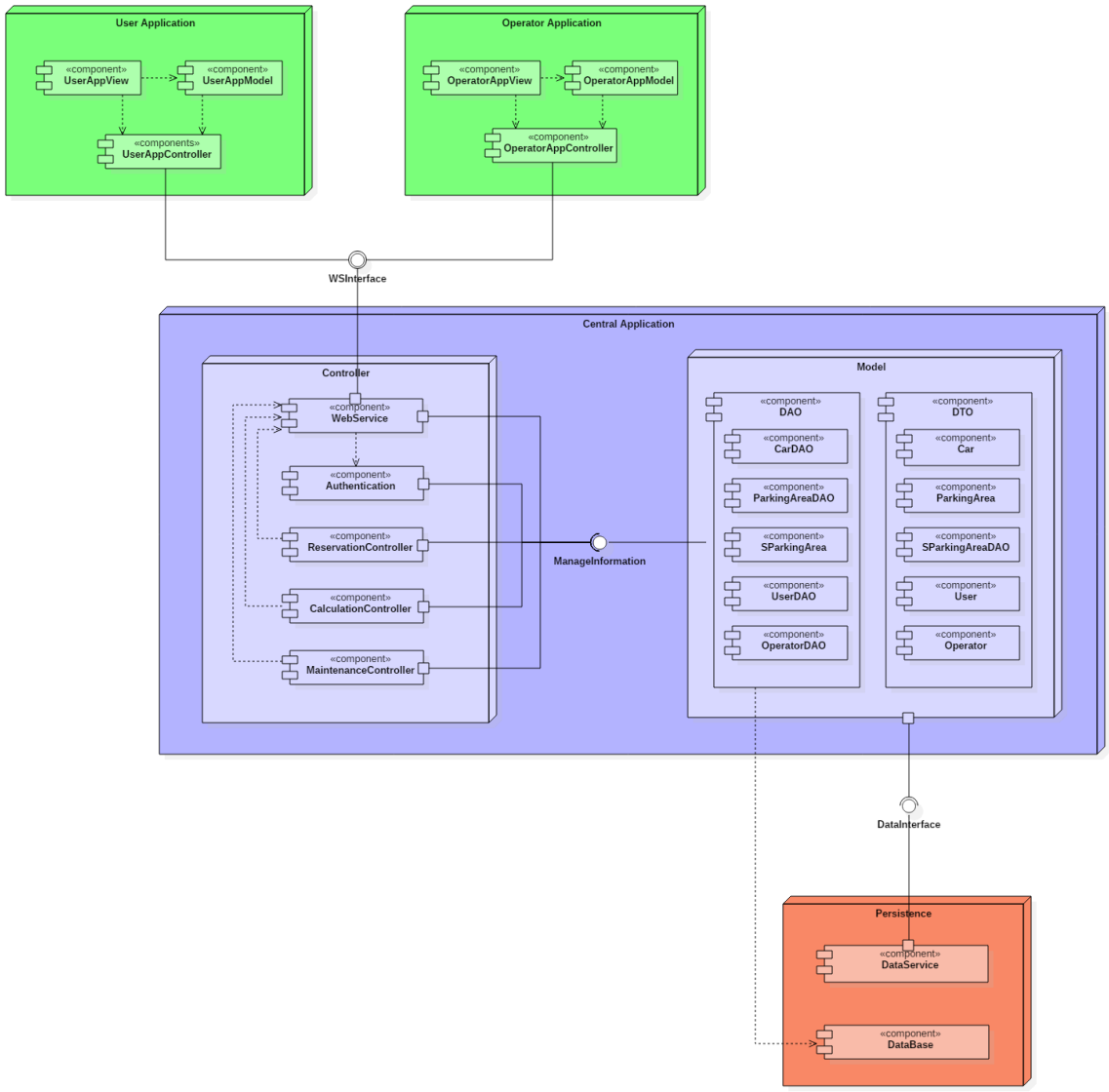
2.1 Entry Criteria

Only when the Integration Test Plan document will be complete, it will be possible to start with the integration testing phase.

Before starting the Integration test of a component, its classes and methods in the test must be completely developed and unit tested successfully. This way, we can ensure that the produced results are meaningful.

As we will discuss in the following chapters, it won't be necessary to wait all the components to be complete and fully tested before starting the integration testing phase.

2.2 Elements to be Integrated



Our system has many components, organised in two levels of granularity.

The higher level of components of our system that need to be integrated are:

- User Application
- Operator Application
- Controller

- Model
- Database

The listed components are divided into lower level components, that will need integration testing.

More in detail, the Controller is composed by WebService, Authentication, Maintenance Controller, Reservation Controller, Calculation Controller.

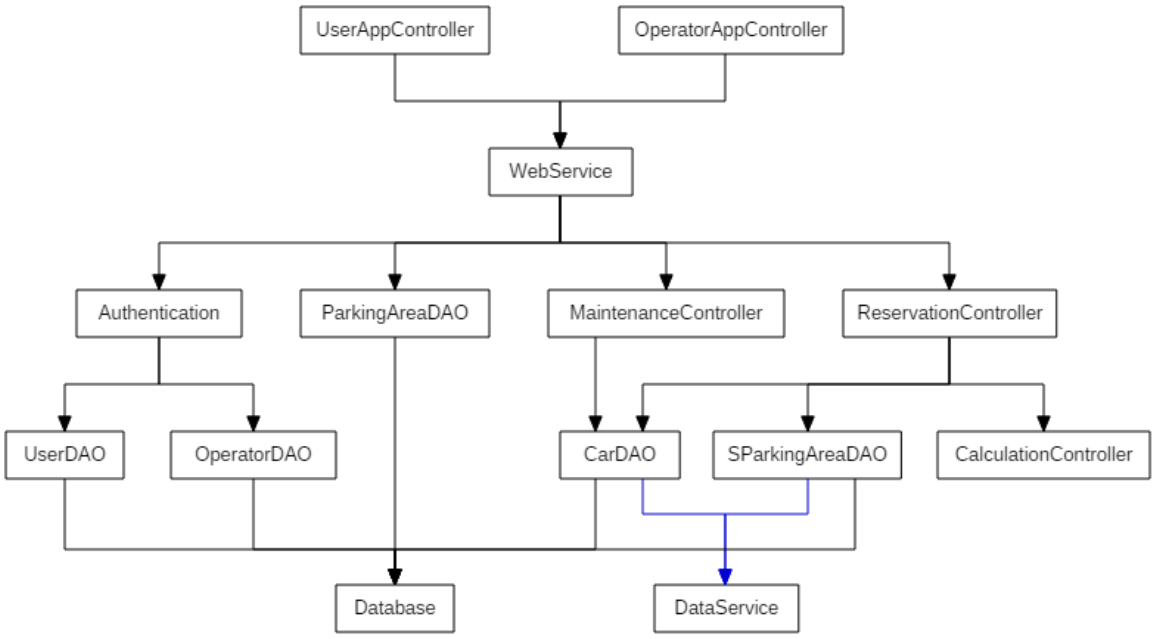
The Model has DAO components and some Pojo components. The last one don't need to be considered for integration. Note that DAO components won't be unit tested.

Data low level components are the DataService and the Database.

Concerning the two mobile applications, we will only integrate the User App Controller and Operator App Controller with the rest of the system.

2.3 Integration Testing Strategy

Following the "called by" concept, we defined a component hierarchy having the data components at the bottom, DAO components on a higher level, the controllers upon them and finally the mobile application. The following figure shows the diagram obtained:



The bottom-up approach has been chosen for testing the integration among components: tests will proceed starting from the bottom of the hierarchy and moving to the top on each step.

Due to the many critical points of our system, this strategy represents for us a safer way for testing and an easier method for finding and correcting software faults. We refer in particular to Data components and ReservationController, because most of the interactions of our system use them.

This approach will include the use of drivers, as discussed in ***$5.1***, for the case in which a component is ready to be tested and its direct parent is not completely written yet.

2.4 Sequence of Component/Function Integration

In the following table, we provide the list of integration tests required.

ID	Integration Test	Paragraph
I1	UserDAO → Database	3.1

I2	OperatorDAO → Database	3.2
I3	ParkingAreaDAO → Database	3.3
I4	CarDAO → Database CarDAO → DataService	3.4
I5	SParkingAreaDAO → Database SParkingAreaDAO → DataService	3.5
I6	Authentication → UserDAO Authentication → OperatorDAO	3.6
I7	MaintenanceController → CarDAO	3.7
I8	ReservationController → CarDAO ReservationController → SParkingAreaDAO ReservationController → CalculationController	3.8
I9	WebService → Authentication WebService → ParkingAreaDAO WebService → MaintenanceController WebService → ReservationController	3.9
I10	UserAppController → WebService	3.10
I11	OperatorAppController → WebService	3.11

3. Individual Steps and Test Description

The following paragraphs contain the detail of the Test Cases defined in the previous chapter.

3.1 Integration test case I1

Test Case Identifier	I1T1
Test Item(s)	UserDAO → Database
Input Specification	User
Output Specification	The database correctly performs the required action.
Purpose	Test select, update, insert and delete operations required from the DAO on the database.
Dependencies	N/A

3.2 Integration test case I2

Test Case Identifier	I2T1
Test Item(s)	OperatorDAO → Database
Input Specification	Operator
Output Specification	The database correctly performs the required action.
Purpose	Test select, update, insert and delete operations required from the DAO on the database.
Dependencies	N/A

3.3 Integration test case I3

Test Case Identifier	I3T1
Test Item(s)	ParkingAreaDAO → Database
Input Specification	ParkingArea
Output Specification	The database correctly performs the required action.
Purpose	Test select, update, insert and delete operations required from the DAO on the database.
Dependencies	N/A

3.4 Integration test case I4

Test Case Identifier	I4T1
Test Item(s)	CarDAO → Database
Input Specification	Car
Output Specification	The database correctly performs the required action.
Purpose	Test select, update, insert and delete operations required from the DAO on the database.
Dependencies	N/A

Test Case Identifier	I4T2
Test Item(s)	CarDAO → DataService
Input Specification	Car
Output Specification	The information of the required car is provided.
Purpose	Verify that the DataService can correctly get information on the required car.
Dependencies	N/A

3.5 Integration test case I5

Test Case Identifier	I5T1
Test Item(s)	SParkingAreaDAO → Database
Input Specification	SpecialParkingArea
Output Specification	The database correctly performs the required action.
Purpose	Test select, update, insert and delete operations required from the DAO on the database.
Dependencies	N/A

Test Case Identifier	I5T2
Test Item(s)	SParkingAreaDAO → DataService
Input Specification	Request for the reservation of a power plug of a certain special parking area.
Output Specification	The DataService provides the correct response and reserves the designated power plug for the specified power station.
Purpose	Test the correct reservation of the power plug in the special parking area sent as an input.
Dependencies	N/A

3.6 Integration test case I6

Test Case Identifier	I6T1
Test Item(s)	Authentication → UserDAO
Input Specification	User

Output Specification	The DAO correctly verifies information entered for the authentication.
Purpose	Ensure that the request from the Authentication controller to the DAO receives the right feedback.
Dependencies	Test Case I1T1 succeeded.

Test Case Identifier	I6T2
Test Item(s)	Authentication → OperatorDAO
Input Specification	Operator
Output Specification	The DAO correctly verifies information entered for the authentication.
Purpose	Ensure that the request from the Authentication controller to the DAO receives the right feedback.
Dependencies	I2T1 succeeded.

3.7 Integration test case I7

Test Case Identifier	I7T1
Test Item(s)	MaintenanceController → CarDAO
Input Specification	Car
Output Specification	The DAO handles all the incoming requests from the MaintenanceController and can provide the correct answer.
Purpose	Verify the functions that: ▪ get the list of cars that need maintenance, ▪ set the status of a car under maintenance, ▪ perform the end of the maintenance request.
Dependencies	Test Case I4T1 succeeded.

3.8 Integration test case I8

Test Case Identifier	I8T1
Test Item(s)	ReservationController → CarDAO
Input Specification	Car
Output Specification	The DAO handles all the incoming requests from the ReservationController and can provide the correct answer.
Purpose	Verify the functions that: ▪ get the list of the available cars, ▪ set the status of a car in use, ▪ set the status of a car as available after release.
Dependencies	I4T1 succeeded.

Test Case Identifier	I8T2
Test Item(s)	ReservationController → SParkingAreaDAO
Input Specification	SpecialParking, PowerPlug
Output Specification	The DAO handles all the incoming requests from the ReservationController and can provide the correct answer.
Purpose	Verify the functions that: ▪ get the list of the available power plugs in the special parking areas, ▪ set the status of a power plug and confirm the reservation.
Dependencies	I5T1 succeeded.

Test Case Identifier	I8T3
Test Item(s)	ReservationController → CalculationController
Input Specification	ReservationInfo

Output Specification	The CalculationController can provide the expected response to the incoming request.
Purpose	Verify that the result provided by the CalculationController is correct for the entered parameters.
Dependencies	N/A

3.9 Integration test case I9

Test Case Identifier	I9T1
Test Item(s)	WebService → Authentication
Input Specification	User or Operator, Request
Output Specification	The WebService sends the correct request to the Authentication controller and gets the expected response.
Purpose	Verify that the Authentication controller can correctly verify the entered authentication information for both the user and the operator. Ensure that the controller can also correctly perform the registration of a new user (new operator not allowed).
Dependencies	I6T1 and I6T2 succeeded.

Test Case Identifier	I9T2
Test Item(s)	WebService → ParkingAreaDAO
Input Specification	None
Output Specification	The DAO provides the list of the parking areas.
Purpose	Test the initialization of the parking areas. The DAO should get the correct information on the database and should return the list of all the parking areas.

Dependencies	I3T1 succeeded.

Test Case Identifier	I9T3
Test Item(s)	WebService → MaintenanceController
Input Specification	OperatorRequest
Output Specification	The MaintenanceController correctly interprets the request made by the WebService.
Purpose	Test the functions through which the operator can: ▪ take in charge a car for maintenance, ▪ set a car as repaired and ready to use.
Dependencies	Test Case I7T1 succeeded.

Test Case Identifier	I9T4
Test Item(s)	WebService → ReservationController
Input Specification	UserRequest
Output Specification	The ReservationController correctly interprets the request made by the WebService.
Purpose	Test the functions through which the user can: ▪ request for a reservation, ▪ request a power plug, ▪ release a car.
Dependencies	I8T1, I8T2, I8T3 succeeded.

3.10 Integration test case I10

Test Case Identifier	I10T1
Test Item(s)	UserAppController → WebService
Input Specification	User, Request

Output Specification	The WebService dispatches correctly the Request and provides the expected result.
Purpose	Verify that the Controller can send requests to the WebService and can get the correct response, for the functions to: ▪ register a new user, ▪ authenticate an already registered user, ▪ reserve a car, ▪ end the use of a car.
Dependencies	I9T1 and I9T4 succeeded.

3.11 Integration test case I11

Test Case Identifier	I11T1
Test Item(s)	OperatorAppController → WebService
Input Specification	Operator, Request
Output Specification	The WebService dispatches correctly the Request and provides the expected result.
Purpose	Verify that the Controller can send requests to the WebService and can get the correct response, for the functions to: ▪ authenticate an existing operator, ▪ take in charge a car, ▪ end the maintenance of a car.
Dependencies	I9T1 and I9T3 succeeded.

4. Tools and Test Equipment Required

4.1 Tools

For supporting and automating Integration Tests we will use two testing tools: JUnit and Arquillian.

Junit is a famous Java framework for performing unit tests but it also can be used for performing integration tests. It is useful to easily check the result returned by methods and we will use it for both unit testing and writing drivers.

Arquillian is an integration testing framework, available for JVM, that seamlessly integrates with Junit. This framework gives the possibility to run tests directly in the container, and this guarantees us to perform a more realistic test. Handling all aspects of test execution, Arquillian makes the integration testing phase simpler, for example, making easier the management of the container lifecycle, the deployment of the archive to test and the capture of results and failures.

Even if we are using some automated testing tool, we cannot avoid to perform manual tests, too. In particular, these are useful to evaluate usability and reactivity of the system through the mobile application, and the interaction with the provided user interface.

4.2 Test Equipment

We have already discussed the fact that our system is based on a client-server architecture.

For testing the central application (consisting in the server side of the system) we will deploy it on a machine explicitly made for working on integration tests and that we define as Testing Server.

The client side consists of the two mobile applications developed for users and operators. The tests will be made using a set of different smartphones, based on the Android and iOS operating systems.

On those devices, we will deploy the mobile applications and the set of tests will be completely run on all of them, by interacting with the Testing Server.

We may consider devices with different versions of the operating system installed, to make sure that the application correctly runs on all of them.

5. Program Stubs and Test Data Required

5.1 Program Stubs and Drivers

For the integration testing of our system we decided to adopt a bottom-up approach. Because of this, we won't need any program stub, but we will only use the drivers listed here:

- **CarDAO Driver**, **SParkingAreaDAO Driver**: these two modules will invoke the methods exposed by the **CarDAO** and **SParkingAreaDAO** components, for the interaction with the **DataService**. As a recall, the DataService provides a way to make possible the interaction between our system and the system installed on cars and power plugs.
- **Authentication Driver**: the driver module should call the methods of the **Authentication** component to test its interaction with the **UserDAO** and the **OperatorDAO**.
- **MaintenanceController Driver**: this module will invoke the methods exposed by the **MaintenanceController** component to test its interaction with the **CarDAO** component.
- **ReservationController Driver**: this driver will call the methods used by the **ReservationController** component for the interaction with the **CarDAO**, **SParkingAreaDAO** and **CalculationController** components.
- **WebService Driver**: the module should invoke all the methods defined in the **WebService** for the interaction with the **Authentication**, **ParkingAreaDAO**, **MaintenanceController** and **ReservationController** components.
- **UserAppController Driver**, **OperatorAppController Driver**: the drivers will invoke the methods of the **UserAppController** and **OperatorAppController** for their interaction with the **WebService** component.

Drivers are not to be used for testing the integration between **DAO components** and the **Database**.

5.2 Test Data

The Database needs to be filled with meaningful data in order to run tests and tools should be used to avoid wasting time writing one record at a time for all the tables.

IBM DB2 Test Database Generator is a tool provided by IBM that can be used for generating data. It allows the definition of:

- the structure of the table,
- constraints on how the data should be generated,
- the output format (SQL, CSV, XML).

Using this tool will be helpful as it will accelerate the testing phase.

Integration tests should also verify the responses of the system in specific cases, such as:

- Null parameters in method call,
- invalid login credentials (User or Operator),
- invalid register information (applies to Users),
- User or Operator with an expired driving licence,
- Car parked outside of the valid Parking Area,
- Reservation of a Car that has already been reserved,
- Payment refused.

More detailed information can be found in the description of the Test Cases, provided in chapter 3.

6. Effort Spent

To make this document we have spent:

- Sergio Caprara, 14 hours
- Soheil Ghanbari, 8 hours
- Erica Tinti, 14 hours