

Luca Scannapieco - 877145

Andrea Pasquali - 808733

Emanuele Torelli - 876210

15/01/2017

PowerEnJoy

Integration Test Plan Document

Table of contents

[1. Introduction 3](#_Toc471074781)

[1.1. Purpose and scope 3](#_Toc471074782)

[1.2. List of definitions and abbreviations 3](#_Toc471074783)

[2. Integration strategies 3](#_Toc471074784)

[2.1. Entry criteria 3](#_Toc471074785)

[2.2. Elements to be integreated 3](#_Toc471074786)

[2.3. Integration test strategies 3](#_Toc471074787)

[2.4. Sequence of component/function integration 3](#_Toc471074788)

[2.4.1. Software integration sequence 3](#_Toc471074789)

[2.4.2. Subsystem integration sequence 3](#_Toc471074790)

[3. Individual steps and test description 3](#_Toc471074791)

[4. Tools and test equipment required 3](#_Toc471074792)

[5. Program stubs and test data required 3](#_Toc471074793)

[6. Other info 3](#_Toc471074794)

[6.1. Reference documents 3](#_Toc471074795)

[6.2. Used tools 3](#_Toc471074796)

[6.3. Hours of work 3](#_Toc471074797)

[6.4. Changelog 3](#_Toc471074798)

# 1. Introduction

## 1.1. Purpose and scope

## 1.2. List of definitions and abbreviations

# 2. Integration strategies

## 2.1. Entry criteria

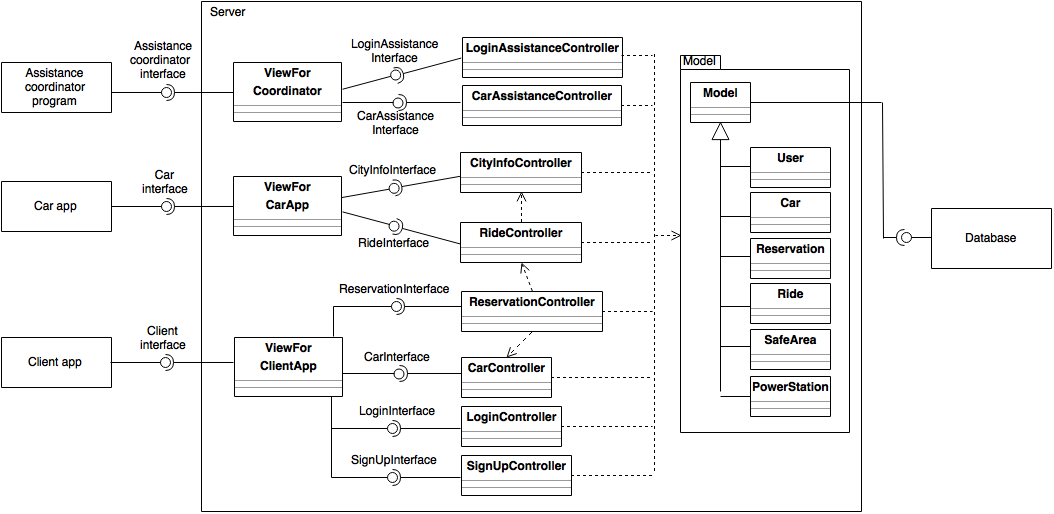
At each step of the software integration system described in the section 2.4.1 the following criteria must be met: all the functions of the components having outgoing arcs (only considering the directional arcs) must have been unit tested.

## 2.2. Elements to be integrated

As explained in the DD our system is built of five high level components: Car App, Client App and Assistance Coordinator Program for the client side; Database and Server for the server side.  
The server subsystem is obviously the most complex and it is, in turn, built of other components interacting among them and forming the MVC pattern. In particular the components to be integrated in the server subsystem are:

* The model reflecting the data in the database
* The controllers that are CityInfoController, ReservationController, CarController, RideController, LoginController, SignUpController, LoginAssistanceController, CarAssistanceController.
* The three views (one for each type of client)

For furtherly clarify the reasoning of the next two sections, we report our component diagram below.



## 2.3. Integration test strategies

We are going to use an incremental approach for integration testing. In particular, we will adopt essentially a bottom-up strategy with few slight modifications.

We will use the a purely bottom-up approach in order to build the component called “server” in the high level component diagram (DD chapter 2.2), that in essence represent the business layer of our application. Therefore, we will start integrating together the atomic subsystems of the server, i.e. the lower level components that do not depend on other components; then we will incrementally integrate the other subsystems that only depends on already integrated and tested components. This strategy, based on the hierarchical structure of the system, allows us to perform the integration test following the development process: as soon as components are released, we integrate them and test the integration. Furthermore, using bottom-up strategy for the server we reduce the overhead time needed to build stubs.  
In order to choose what to integrate among the atomic components we will follow the critical-module-first policy. In our case the most critical modules are the most used-one, such as the model that is the core of our MVC in the server side and therefore also the first component to be developed.   
For what concern the client side, we can say that we violate a bit the bottom-up strategy rules. In fact we are going to test the client side components such as Car App, Client App and Assistance Coordinator Program together with the server components even if the client side components use those of the server. This little modification of the strategy has the purpose of increase the parallelism of the work and consequently even the efficiency.

## 2.4. Sequence of component/function integration

In this section we are going to describe the order of integration of the components and subsystems of PowerEnjoy. An arc going from component A to component B means that component A needs to be implemented before component B; an unidirectional arc means that there is not such a dependency.

### 2.4.1. Software integration sequence

According with the critical-module-first policy described in the chapter 2.3 the first two elements to be integrated are the Database and the Model because they refer to the data of our system and thus they are the most used components.

// database 🡪 model

The next components to be integrated are the controllers that do not interact with other controllers such as: SignUpController, LoginController, LoginAssistanceController, CarAssistanceController.

// model -> CarController, CityInfoController, SignUpController, LoginController, LoginAssistanceController, CarAssistanceController

Now we can proceed adding the controllers interacting with the already implemented controllers: the next controller is RideController that only interact with CityInfoController.

//model -> RideController <- CityInfoController

The last controller to integrate is ReservationController that uses RideController and CarController.

//model -> ReservationController <- RideController, CarController

Once we have integrated all the controllers we can finally integrate the three views.

//views

As explained in the section 2.3 we can integrate the three components of the client side in parallel with those of the server side. We only need to implement the stubs for the three views used by the three client components. These stubs will be substituted by the real views once the server subsystem will be completely integrated.

//stubs->clients

### 2.4.2. Subsystem integration sequence

The following schema shows how the integration test proceeds looking at the high level components (or subsystems).

//schema

# 3. Individual steps and test description

# 4. Tools and test equipment required

# 5. Program stubs and test data required

# 6. Other info

## 6.1. Reference documents

## 6.2. Used tools

## 6.3. Hours of work

## 6.4. Changelog