



Politecnico di Milano

PowerEnjoy Service - Integration Test Plan Document

January 9, 2017

Version 1.1

Authors:

- Domenico FAVARO (Mat. 837995)
- Matheus FIM (Mat. 876069)
- Caio ZULIANI (Mat. 877266)

Prof. Elisabetta DI NITTO

Contents

1	Introduction	2
1.1	Revision History	2
1.2	Purpose and Scope	2
1.3	Definitions and Abbreviations	2
1.4	Reference Documents	3
2	Integration Strategy	4
2.1	Entry Criteria	4
2.2	Elements to be Integrated	4
2.3	Integration Testing Strategy	5
2.4	Sequence of Component/Function Integration	5
2.4.1	Software Integration Sequence	5
2.4.2	Subsystem Integration Sequence	9
3	Individual Steps and Test Description	10
4	Performance Analysis	11
5	Required Tools and Test Equipment	12
5.1	Tools	12
5.2	Test Equipment	13
6	Required Program Stubs and Test Data	13
6.1	Program Stubs	13
6.2	Test Data	13
7	Effort Spent	14
8	Changelog	15

Introduction

Revision History

This section records all revisions to the Document.

Version	Date	Authors	Summary
1.1	15/01/16	Domenico Favaro, Caio Zuliani, Matheus Fim	Initial Release

Purpose and Scope

The Integration Test Plan Document (ITPD) serves to present the integration sequence and testing for all Subsystems and Components that conform PowerEnjoy Car Sharing Service. This is a key part to guarantee the functioning and quality of the software. The Document will present the division of the System in Subsystems and Components that will endure individual testing as independent modules and then be subject to integration on the whole System.

Definitions and Abbreviations

- **RASD:** Requirements And Specifications Document.
- **DD:** Design Document.
- **ITPD:** Integration Test Plan Document.
- **SDK:** Software Development Kit
- **App:** Application, referring to Web or Mobile App.
- **Subsystem:** Part of the system the generally encapsulates one or more features.
- **Component:** Self sustained part of the System that provides with functionalities and is part of one or more subsystems.
- **Bottom-up:** Referring to Bottom-up testing. Each component at lower hierarchy is tested individually and then the components that rely upon these components are tested.
- **Top-down:** Top-down integration testing is an integration testing technique used in order to mock or simulate the behaviour of the lower-level modules that are not yet integrated.
- **Mock:**Simulation that mimic the behavior of certain objects and fuctions in controlled ways, done to test the behavior of some other object.

For other concepts concerning the Service definition look in the **Glossary** section of the RASD and DD.

Reference Documents

- Specification Document: Assignments AA 2016-2017.pdf
- PowerEnjoy Requirements And Specifications Document (RASD)
- PowerEnjoy Design Document (DD)
- Example Document - Integration testing example document.pdf
- Testing Tools Documents:
 - Mockito
 - JMeter

Integration Strategy

Entry Criteria

We define the criteria that must be met before integration testing of the system components. We consider Integration a part of the production development. In order for production to start all documentation must first be written and up to date, including RASD and DD, to have a clear and full scope of the system components functionalities and importance. Once in production, the integration of a single component can be done when the following criteria is met:

- The Component feature must be 100% complete, that is all classes and functions must have been implemented.
- No tickets must be opened for the Component, no bugs or considered missing features must be present.
- Individual component testing must have been performed, using JUnit to test its classes and functions.
- All the interfaces the Component has to communicate to have to be present or at least mocked to be able to test its coupling.

Elements to be Integrated

As stated in the Design Document, our system is composed by several High level Components presented in 3 tiers. Specifically these components are:

- Client Tier:
 - User Client Component
 - CRM Client Component
 - Car Component
- Server Tier:
 - User Controller
 - CRM Controller
 - Car Controller
 - Reservation Controller
 - Ride Controller
 - Payment Controller
 - User Report Controller

- Email Helper
- Location Helper
- Chat Service
- EIS Tier:
 - Database

Integration Testing Strategy

Our approach following the 3-tiered structure will follow a **Bottom-up** strategy, working on components that do not depend on others to function first.

This implies following the next tier order: EIS -> Server -> Client for development and testing.

Inside each Tier Bottom-up strategy will be used again to integrate independent modules first and then those that depend on others. This strategy will help in contrast to Top-down to minimize the mock-up testing to be done, testing on top of already deployed modules. The order in which the 'Bottom' modules will be picked will follow a Critical-Module-First Integration Strategy, giving priority to those that will have dependencies of other modules, this will help not only to spot any error on critical modules first but also to unblock the integration of dependant modules earlier on.

Sequence of Component/Function Integration

Following the Bottom-up strategy we'll integrate the components that have no dependencies on other modules. First we'll present the component integration within each subsystem and then the subsystems integration order.

Software Integration Sequence

For each subsystem, we'll identify the sequence in which the software components will be integrated within the subsystem.

EIS Tier - DBMS: As shown in the DD our DBMS is not dependent on any module and even if it's a System already present for some structures (Cars, CRM) we have to add the entities that we'll serve the purpose of our System, that is Users, Reservations, Rides, Payments, User Reports. These will be entities that will be added by our System, specifically be the Controllers so this module has to be integrated first to answer the queries for the rest of the subsystems.

Server Tier: In the DD we present the High level Component structure. Based on the dependencies shown in this structure we'll select the order to test and deploy these components. The Helper Components de-

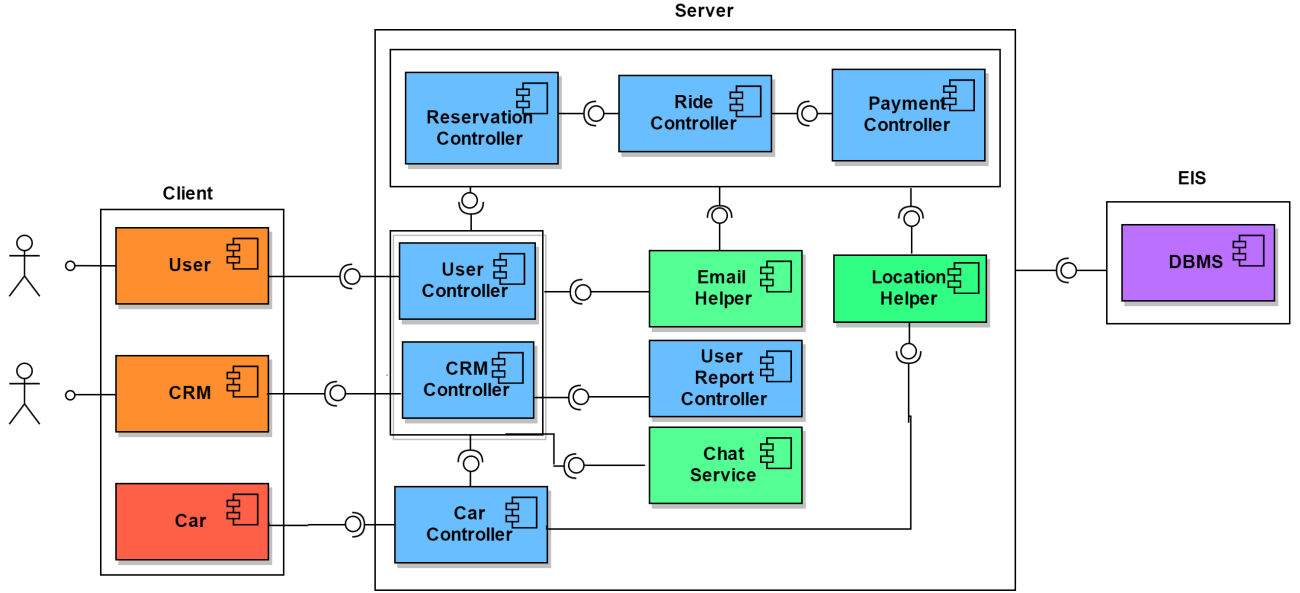


Figure 1: Component Structure

pend on no other components so according to the Bottom-up strategy they will be the first ones to integration. However not all Helpers are critical to other components so we'll prioritize the Critical Helper Components first, that is Location Helper as User, Car and Ride Controllers depend on it.

For each controller we have to implement Entity and Session Beans that will contain the methods and functions for managing the corresponding entites and logic so everyone of them will have Data Access Utilities to communicate with the DBMS. Their Integration order will then depend on the other controllers they depend on, meaning the other controllers they have to interface with.

An arrow suggest the Component on the right depends on the one on the left and will be integrated after all the previous one have been integrated.

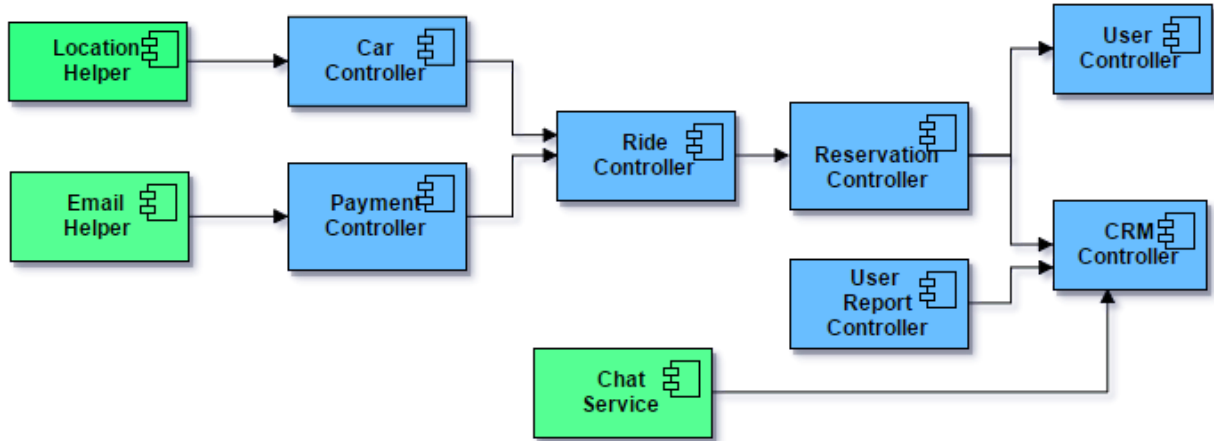


Figure 2: Server Components Integration

Client Tier: The Client Tier is based in two main components, User and CRM. Even if they are different they share many common functionalities including Login, Car Localization, Car Detail and Chat Service. They can be deployed simultaneously as they do not depend on each other and to test the logic subsystem from the Client side we can deploy one functionality at a time when all the dependencies have been met.

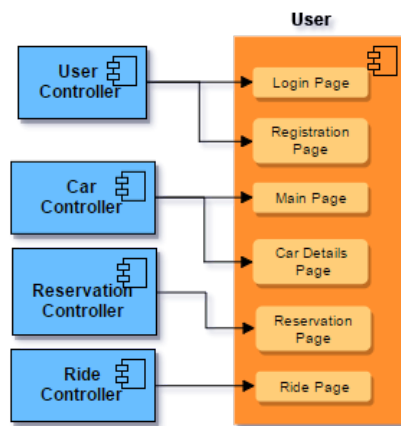


Figure 3: User Component Dependencies

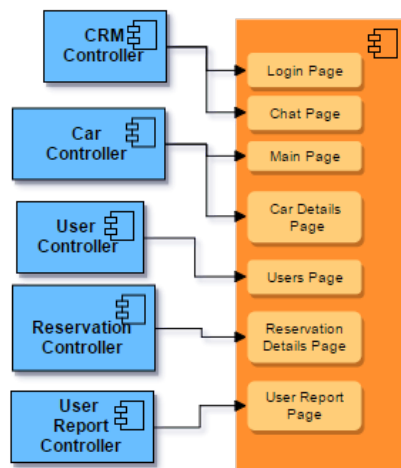


Figure 4: CRM Component Dependencies

Subsystem Integration Sequence

As mentioned before our subsystems corresponding to the Data, Logic and Presentation Tier will be integrated in that order. Before integrating a subsystem all the components inside it must be integrated as well.



Figure 5: Subsystem Integration Sequence

Individual Steps and Test Description

Performance Analysis

Even though the performance analysis is evaluated within the system as a whole, it was agreed that while testing the components, the isolated performance will be taken into account, as to correct unacceptable behavior, such as too slow response time, as soon as possible.

The aim of the performance analysis is to check the reliability of the application under normal usage conditions, providing benchmarks and identifying the response time, utilization and throughput of the application. So, for this test, the expected workload should be considered as in terms of the biggest city where the application will be implemented, taking into account an average usage for a long period with peaks of heavy traffic.

Both the server side and the client side can affect the performance of the application and for this reason need to be checked. Also, it is necessary to consider the performance of all the different client interfaces. Particularly, the mobile application, the web application and the screen inside the car should all have satisfactory behavior, considering all kinds of users that can operate each interface.

Some important requirements to be evaluated:

- For the mobile application it is considered that the target public of the app can have any kind of Smartphone. So, the test will be made in low-end devices. This includes low ram availability, small internal space allocation and low processing power capacity.
- All the interfaces need to respond properly with slow network situations, and be reliable in situations of unstable network.

Required Tools and Test Equipment

Tools

In order to guarantee the most reliable system possible, when integrating components all the individual tests will be carried out once again, to make sure the integration did not cause new bugs to occur. For this reason, all the used programs provide tests automation tools, minimizing the rework.

The tools chosen for the Java EE and the EIS tiers were specifically three and each has its own scope of tests.

Firstly, the JUnit Framework will take care of the individual components testing. This is basically a way to certify that results produced by the components matches the theoretical value.

Secondly, the Arquillian framework helps keeping the integration testing simple. Testing the components of an application is challenging, so this framework focus on the interaction with the system, providing tools to check that the right components are being injected and the interactions with the database are occurring in a normal way.

Finally, Jmeter framework brings the application closer to the real world by performing load tests and performance tests. This tool provides emulations of loads on the server, network and objects providing solid data to analyze the overall performance of the systems under various conditions. However, it is still necessary to test components withheld in the Client tier, specifically the mobile applications. In this matter, it will be used tools provided with each platform (IOS and Android have their own performance analysis tool that comes along with the SDK of the plataform). And, as stated in the previous session, particularly harsh situations as low battery, bad network coverage and low available memory need to be taken into account.

These tools ensure that every aspect of the application is tested in the proper way and provide all the necessary features to accomplish the proposed points of this document.

Test Equipment

Required Program Stubs and Test Data

Program Stubs

Test Data

Effort Spent

Date	Domenico	Caio	Matheus
27/12/16	2h	2h	2h
28/12/16	-	-	-
29/12/16	1h	-	-
30/12/16	2h	-	-
31/12/16	-	-	-
01/01/17	-	-	-
02/01/17	2h	-	-
03/01/17	2h	-	-
04/01/17	2h	-	-
05/01/17	-	-	-
06/01/17	-	-	-
07/01/17	-	-	-
08/01/17	-	-	6h
09/01/17	-	-	-
10/01/17	-	-	-
11/01/17	-	-	-
12/01/17	-	-	-
13/01/17	-	-	-
14/01/17	-	-	-

Changelog

As the project and design decisions may change during the development this document is also prone to change. We'll document every version in this part.

- **Version 1.1:** 15/01/2017