POLITECNICO MILANO 1863

Computer Science and Engineering

PowerEnjoy Service - Integration Test Plan Document

January 13, 2017

Prof. Luca Mottola

Authors:

- ZHOU YINAN(Mat. 872686)
- ZHAO KAIXIN(Mat. 875464)
- ZHAN YUAN(Mat. 806508)

Contents

1 INTRODUCTION

1.1 Revision History

At this moment, this is the first version of the document.

1.2 Purpose and Scope

This document describes how the integration should proceed. Integration testing means that we need to verify all the components needed for the overall system should work correctly not only individually but also in combination. In this document, we provide the steps needed to follow in order to get a fully functional system. More specifically, the elements need to be tested, the testing strategy, sequence of integration, test description, tools and stubs will be presented in the following parts.

1.3 List of Definitions and Abbreviations

- RASD : Requirement Analysis and Specification Document
- DD : Design Document
- Guest : All the users of the system who have not performed a Log in operation yet
- User : After a Guest logs in, he/she becomes a User
- Subcomponent : each of the low level component realizing specific functionalities of the subsystem
- subsystem : a functional unit of the system

1.4 List of Reference Documents

- Assignment AA 2016-2017
- RASD
- DD

2 INTEGRATION STRATEGY

2.1 Entry Criteria

There are several entry criteria to be completed before the integration testing phase can begin.

- RASD and DD documents are completed
- Components have to be unit tested before the integration testing
- The required driver and stub have already been developed
- database is fully functioned

The application subsystem may not be fully developed at this moment, however the interface between Application tier and Server tier is a must for testing to proceed.

2.2 Elements to be Integrated

The system is divided into 3 subsystems according to the 3 tier architecture we chose in the DD : Application, Server, Database. This document mainly focuses on the integration testing for the Server side. The following components needed to be integrated:

- Guest Application Manager
- User Application Manager
- Car Application Manager
- Database Manager

The above components are the basic low-level components required for higher level functionalities of the system. Besides the components we need to develop by ourselves, some external systems and API are used:

- Google Map API
- Bank Service system

2.3 Integration Testing Strategy

For testing the integration of components, we choose the bottom-up approach. By bottom-up approach, we start by the components which have no dependency of other components and the very fundamental components providing services to all others. In our system, we start from the Database Manager component. The reason behind it is that basically all of our functions need Database Manager. Thus it is natural and easy to begin with it (the bottom level) and add other components step by step.

2.4 Sequence of Component/Function Integration

Basically we have three subsystems in our system : Application side, Server side and Database side. We will focus on the first two subsystem and neglect the last one because we'll use a DBMS from outside. What we will focus is the interface between the server and data base. We always assume DBMS is reliable.

In order to implement the testing, we divide the whole system into three subsystems :

- Guest application manager system
- User application manager system
- Car application manager system

In each subsystem, we'll proceed with the bottom-up approach.

2.4.1 Software Integration Sequence

We will test the three subsystems separately and in each subsystem, a bottom-up approach is used. The following paragraph describes how this approach works.

(1) Application side The application subsystem is composed of two components : User application and Car application. These two components are parallel and do not have any dependency on each other. The dependency of these two components lie on the Server side. Thus, these two components can be developed separately but can only be tested after the Server side is functional.

(2) Server side The Server has four components : Database Manager,

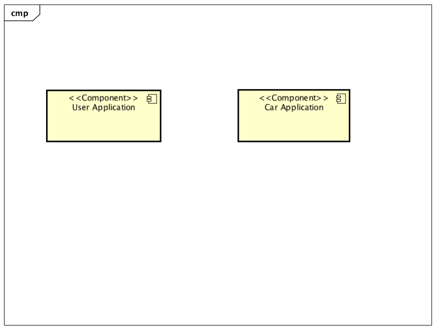

Figure 1: figure Application subsystem

Guest Application Manager, User Application Manager and Car Application Manager. We'll start from Database Manager and add other components step by step.

- 1 Database Manager

Database Manager is the most basic and fundamental component in our system, thus it will be tested firstly.

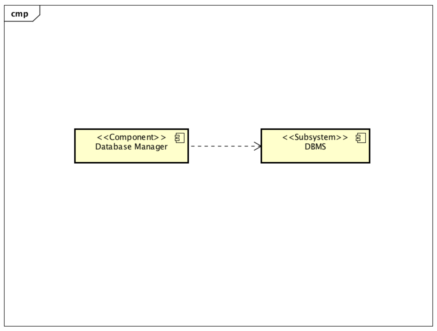

Figure 2: Database Manager

- 2.1 Gust Application Manager

After testing the Database Manager, we'll proceed with our testing procedure with Guest Application Manager which is responsible for user registration and log in. Note that this testing can be proceeded in parallel with the Car Application Manager.

- 2.2 Car Application Manager

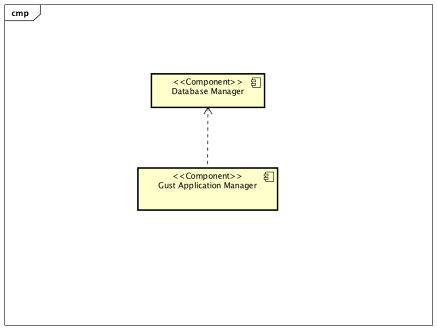

Figure 3: Guest Application Manager

Car Application Manager is responsible for managing the car information with database and all the controls in the car, including unlocking the car.

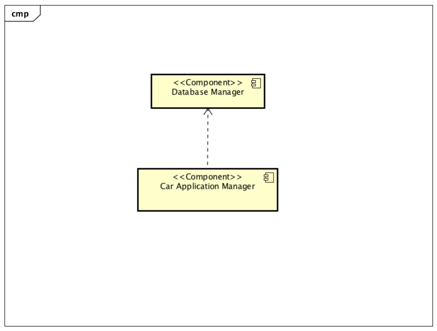

Figure 4: Car Application Manager

- 3 User Application Manager

All the services for the users can only be accessed after performing the log in operation. However User Application Manager and Guest Application Manager do not have dependency on each other. User Application Manager is responsible for rental services. More preciously, getting available cars, making reservations.

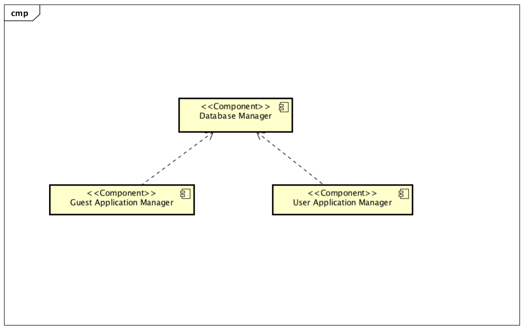

Figure 5: User Application Manager

2.4.2 Subsystem Integration Sequence

After testing the components and subsystems in the Server and Application sides, we can begin to test the integration of subsystems. Since a full functional requirement needs to operate on all the three subsystems, we integrate them all together.

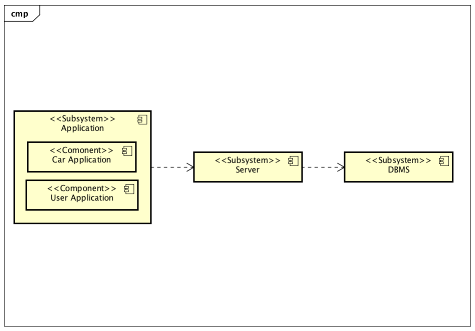

Figure 6: subsystems

Here presents the overall dependency graph for the whole system.

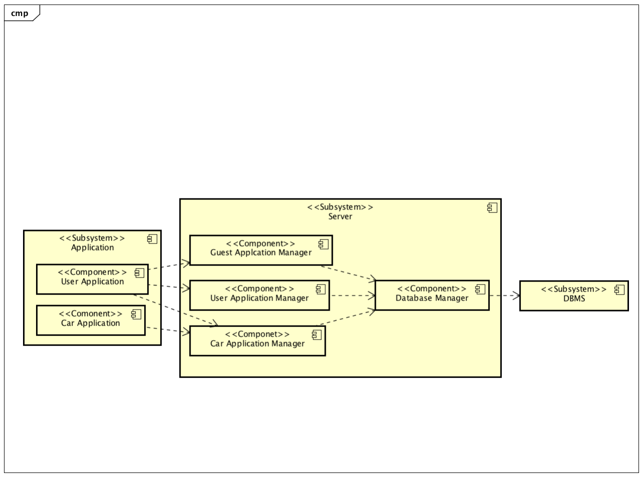

Figure 7: subsystems

3 Individual Steps and Test Description

In this chapter we will provide the description of tests on functions that are used in integration of components, including the brief description of the parts get involved and possible input values and corresponding output values. We will also include the effect that we expect to get for each function.
The notion (A,B) of the subtitle represents that component A will call component B with the function below it.

3.1 Guest application management system

3.1.1 guest application manager, database manager

Boolean register(credential)	
input	output
A non ~~empty~~ credentials of user Valid	the guest has already registered into database
Valid credential	The user will actually be registered into database

String signIn(email,password)	
input	output
Empty email of user found	An InvalidArgumentValueException is raised
Invalid password of user	An InvalidArgumentValueException is raised
Valid information	information of user will be sent back

3.1.2 user application,guest application management

String register(credential)	
input	output value or effect
Null parameter in some field	A NullArgumentException of correspondence field is raised
Invalid parameter's form in some field	An InvalidArgumentValueException of correspondence field is raised
Valid parameters	a string will showed to notify the success

3.2 User application manager system

3.2.1 User application manager,database manager

car[] getCarAvailable(location,DISTANCE)	
input	output
Null parameter	A NullArgumentException is raised
Invalid/no-found/out-coverage position	An InvalidArgumentValueExption is raised
Valid parameter	a list of car that respect the condition will be extracted from the database

String setReservation(user,car)	
input	output
Null parameter	A NullArgumentException is raised
the car is still locked by other reservation	An InvalidArgumentValueException is raised
Valid parameter	the reservation will insert into the database and a string to notify the situation

Reservation[] getListActiveReservation(user,car)	
input	output
Null parameter	A NullArgumentException is raised
Inexistence reservation with parameters	An InvalidArgumentValueException is raised
Valid parameters	a list of reservation still in status of active will be extracted from the database

String cancelReservation(user,car,reservation)	
input	output
Null parameter	A NullArgumentException is raised
Inexistence reservation with parameters	An InvalidArgumentValueException is raised
Valid parameter	the status of the reservation will be changed into cancelled and a string to nority the situation

3.2.2 User application, User application manager

car[] getCarAvailable(position)	
input	output
Null parameter	A NullArgumentException is raised
Valid position	return the list of the car available in certain distance.

void reserve(car,user)	
input	output
Null parameter	A NullArgumentException is raised
Valid argument of parameters	the reservation will be send to server

3.3 Car application manager system

3.3.1 Car application manager, Database manager

startRide(user,car)	
input	output
Null parameter	A NullArgumentException is raised
The car is not available for user	An InvalidArgumentValueException is raised
The user is not rending the car passed as parameter	An InvalidArgumentValueException is raised
Valid Parameter	the ride will be inserted into the database

updateRide(user,car,state)	
input	output
Null parameter	A NullArgumentException is raised
Invalid state insert	An InvalidArgumentValueException is raised
The ride inexistence	An InvalidArgumentValueException is raised
Valid Parameter	the field of the ride is be updated using state

insertCar(car)	
input	output
Null parameter	A NullArgumentException is raised
Invalid car specification	An InvalidArgumentValueException is raised
The carID already exist	An InvalidArgumentValueException is raised
Valid parameter	the car is inserted into database

deleteCar(car)	
input	output
Null parameter	A NullArgumentException is raised
The carID inexist	An InvalidArgumentValueException is raised
Valid parameter	the car is removed from database

insertSafeArea(area)	
input	output
Null parameter	A NullArgumentException is raised
Invalid area specification	An InvalidArgumentValueException is raised
The area already exist	An InvalidArgumentValueException is raised
Valid parameter	the area is inserted into database

deleteSafeArea(area)	
input	output
Null parameter	A NullArgumentException is raised
The area inexist	An InvalidArgumentValueException is raised
Valid parameter	the area is removed from database

updateSafeArea(area,status)	
input	output
Null parameter	A NullArgumentException is raised
Invalid status	An InvalidArgumentValueException is raised
The area inexist	An InvalidArgumentValueException is raised
Valid parameter	the area is updated with status

3.3.2 User application, car application manager

void openTheDoor(car,user,location)	
input	output
Null parameter	A NullArgumentException is raised
The location is so far from the car	An InvalidArgumentValueException is raised
The car is not available for user	An InvalidArgumentValueException is raised
Valid parameter	the door of the car will be unlocked

3.3.3 Car application, car application manager

void startRide(user,car,money save option)	
input	output
Null parameter	A NullArgumentException is raised
Valid parameter	the information will be send to the server

void endRide(user,car,state)	
input	output
Null parameter	A NullArgumentException is raised
Valid parameter	the information will be send to the server

string getCurrentPrice(user,car)	
input	output
Null parameter	A NullArgumentException is raised
The car is not being renting by user	A InvalidArgumentValueException is raised
Valid parameter	the current charge will be returned and will be showed on the screen

void variationCost(state) discount	
input	output
Null parameter	A NullArgumentException is raised
The state inexistence	A NullArgumentValueException is raised
Valid parameter	the variation applied will be managed by server

3.3.4 Car application manager, Car application

void sendMsg(car,string)	
input	output
Null parameter	A NullArgumentException is raised
The car is not in riding or renting	An InvalidArgumentValueException
Valid informations	the string will be showed on the screen of the car

3.4 Intergration with external system

3.4.1 Guest manager&User manager&Car manager , Gmail API

sendMail(email,string)	
input	output
Null parameter	A NullArgumentException is raised
Invalid email	An InvalidArgumentValueException is raised
The email inexist	An InvalidArgumentValueException is raised
Valid parameter	the email contain the string is sent to user

3.4.2 User manager&Car manager , Google Map

double[] findCoordinates(location)	
input	output
Null parameter	A NullArgumentException is raised
Invalid location	An InvalidArgumentValueException is raised
Valid parameter	return the longitude and latitude of the location

Map getMap(location)	
input	output
Null parameter	A NullArgumentException is raised
Invalid/inexistence location	An InvalidArgumentValueException is raised
Valid parameter	return the map around the location

3.4.3 Car application manager, Bank

payment(user,motive,cost)	
input	output
Null parameter	A NullArgumentException is raised
Invalid motive	An InvalidArgumentValueException is raised
No record that user done action before	An InvalidArgumentValueException is raised
Valid parameter	the cost will be charge to system account

4 Tools and Test Equipment Required

4.1 Tools

With the purpose of using the PowerEnJoy Service more effectively, and ensure each components of this system can work appropriately, we should make a use of some effective and automated testing tools. These testing tools could help us to test the components of the system without rewriting the code and could make the testing easier.
As far as we are concerned, the main business logic component are running in the JEE runtime environment, in this case, we choose two mainly testing tools for the testing.

4.1.1 JUnit

With no doubt, the first one is the JUnit. JUnit is a unit testing framework for the Java programming language. JUnit has a very important development in test-driven field. Nowadays, this tool is primarily devoted to unit test activities, and the people all around the world are more willing to use it in the unit testing. Because it's easy to use, and the user can verify the interactions between components can produce the expected results. There are some characters of the JUnit.

- JUnit is an open source framework for writing and running tests.
- Comments are provided to identify the test method.
- Providing the assertions to test expected results.
- JUnit tests allow you to write code faster and improve quality.
- JUnit elegant and simple. Not so complicated, less time-consuming.
- JUnit tests can run automatically and check their own results and provide immediate feedback. So there is no need to manually sort out the test results of the report.
- JUnit tests can be organized into test suites, including test cases, and even other test kits.
- JUnit displays the progress in a bar. If it works well, it would be green; if it fails, the display would turn red.

4.1.2 Arquillian

There is also another widely used testing tool, called Arquillian integration testing framework. Arquillian is a new test framework which is JUnit-based, and developed by JBoss. The main purpose of this testing tool is to simplify the coding in Java development project, when the developer is working with the integration test and the functional test. Thanks to this testing tool, the integration tests and the functional tests could be as simple as unit tests. Arquillian can be used in the Web container, and it interacts with the container in three main ways.

- 1. Embedded. Arquillian and Web containers run in the same JVM.
- 2. Managed. Arquillian decides when to start, close the Web container to deploy to the container, and run the tests.
- 3. Remote (remote). The developer starts the Web container in advance, Arquillian connects the container and runs the test into the container.

4.2 Test Equipment

As we all know, the accomplishment of the code does not mean the testing is finished. further more, the integration testing activities have to be performed within the specific testing environment.
Since this PowerEnjoy Service should be used in the client side and the backend side, we should define the characteristics of the devices that have to be used in these two sides, and survey whether the performances of these devices are appropriated.
For the client side, the client uses the smartphone to reserve the car and process the requirement. Therefore, for the testing environment, the follow devices are required.

- At least one IOS smartphone, which is running IOS operating system.
- At least one Android smartphone, which is running Android operating system.
- At least One Windows smartphone, which is running Window operating system.
- At least one IOS tablet, which is running IOS operating system.

- At least one Windows tablet, which is running Windows operating system.

These testing devices would be used to test both the mobile applications and the Web version of the Web application. it is also should be noted that, the testing devices should be as general as possible. The range of the testing devices selection, should cover the wildest range of the possible configuration.
In fact, for satisfying the most general case, we should consider to survey the smartphone marker. If we want to get the most general testing result, we should use the most widely used devices, in order to better reflect the typical usage scenarios we would encounter in the real operating environment.
As for the backend testing, the business logic component and other components should be deployed in the real framework which would be used in the real business application. In this project, we are going to use some software component, such as:

- The Oracle Database Management System
- The JEE runtime
- Java Application Server

5 Program Stubs and Test Data Required

5.1 Program Stubs and Drivers

In this project, we are going to use a bottom-up approach to compose the components of this service system. Therefore, we also use the bottom-up framework to for integration and testing.
To finish the testing, we are going to use a number of drivers to drive each component for simulate the real system. What's more, we need the drivers to perform the necessary function for testing.
Here are the list of the drivers which are used in the testing

- Data Access Driver: this module would help the system to retrieve the information in the DBMS, such as the client account, the location of the car. This is the critical component of the testing, since the performance of service system need the interaction of information.

- User Application Manager Driver: this module in charge of helping the User Application Manager to accomplish its work. Such as processing the user requirements, interaction of the database manager and other subcomponents.

- Guest Application Manager Driver: this module would invoke the methods exposed by the Guest Application Manager subcomponent. This module is in charge of testing the interaction of the Guest and Guest Application Manager. What's more, this module would also be helpful to test the interaction of the Guest Application Manager and the Database Manager.

- Car Application Manager Driver: this module would invoke the methods exposed by the Car Application Manager subcomponent. With the help of this module, we can test the function of the Car Application Manager and the interaction between Car Application Manager and other subcomponents.

- Database Manager Driver: this module would invoke the methods exposed by the Database Manager subcomponent. The main purpose of this module is to be used for testing the interaction between Database Manager subcomponent and other subcomponents.

In general, if we build a service system with the bottom-up approach, there is no need to use any stubs in the development period. ~~But in the~~

~~testing, we could not go ahead without a few stubs~~.
In fact, the main purpose of these stubs is simulating the real service environment. For example, if we want to test the interaction of User Application Manager and Database Manager, we could use the stub to simulate the function of the Database Manager and accomplish the testing without finishing the whole coding of the Database Manager.

5.2 Test Data

With the purpose of testing the corresponding functions, we should have some specified testing data.

- A list of both valid and invalid candidate guest to test Guest Application Management component. The set should contain instances as the following:

  - Null object
  - Null fields
  - Guest not compliant with the legal format

- A list of both valid and invalid candidate guest to test User Application Management component. The set should contain instances as the following:

  - Null object
  - Null fields
  - Invalid e-mail address
  - Invalid password

- A list of both valid and invalid candidate guest to test Car Application Management component. The set should contain instances as the following:

  - Null object
  - Null fields
  - Location out of the range

- A list of both valid and invalid candidate guest to test Database Management component. The set should contain instances as the following:

- Null object
- Null fields

6 Effort Spent

29/12/2016 ZHOU YINAN 2h introduction and strategy
31/12/2016 ZHOU YINAN 2h introduction and strategy
07/01/2017 ZHAN YUAN .5h Individual step and test description
07/01/2017 ZHAO KAIXIN 3h Tools and Test Equipment Required
08/01/2017 ZHAO KAIXIN 2h Tools and Test Equipment Required
09/01/2017 ZHAO KAIXIN 2h Program Stubs and Test Data
09/01/2017 ZHAN YUAN 1.5h Individual step and test description
10/01/2017 ZHAN YUAN 2.5h Individual step and test description
11/01/2017 ZHAN YUAN 1h Individual step and test description
13/01/2017 ZHOU YINAN 2h modification and release the v1.0 document