



POLITECNICO
MILANO 1863

Integration Test Plan Document

Version: 1.0

Reda Aissaoui, Jinling Xing, Lidong Zhang

January 15, 2017

Content

1. Introduction
 1. Purpose and Scope
 2. Definitions and Abbreviations
 3. Reference Documents
2. Integration Strategy
 1. Entry Criteria
 2. Elements to be Integrated
 3. Integration Testing Strategy
 4. Sequence of Component/Function Integration
3. Individual Steps and Test Description
4. Tools and Test Equipment Required
5. Program Stubs and Test Data Required
6. Effort Spent

1. Introduction

1. Purpose and Scope

In our project, the Integration Testing Plan Document is an important part, which can guarantee that all components can work together. It is also a way to find bugs earlier in the development process.

The main purpose of this document is present the approach for the integration testing . The Integration Testing Plan Document shows the way and chronology to organize all the testing of subcomponents. We will provide these sections as follows: - To introduce the integration testing subsystems and their subcomponents involved in the integration activity. - The Elements to be Integrated and their entry criteria. - A description of the Integration Testing Strategy. - The Sequence of Components needs to be integrated, including Software Integration Sequence and Subsystem Integration Sequence. - Individual steps and test description and their input data and the expected output. - All the tools used in testing, and a description of the operating environment in which all the tests will be performed.

2. Definitions and Abbreviations

1. Definitions

- Subsystem: a single, pre-defined work environment and a high-level functional unit of the system.
- Subcomponent: Implementation of subsystem functions

2. Abbreviations

- RASD: Requirements Analysis and Specification Document
- DD: Design Document
- ITPD: Integration Test Plan Document
- API: Application Programming Interface
- GUI: Graphical User Interface
- DBMS: Database Management System
- GPS: Global Positioning System
- SDK: Software Development Kit
- CRUD: Create, Read, Update and Delete
- Req: Requirement
- App: Application

3. Reference Documents

- Requirements Analysis and Specification Document

- Design Document
- Assignments AA 2016-2017
- Integration testing example document
- IEEE standard on requirement engineering

2. Integration Strategy

1. Entry Criteria

In order to initiate the integration testing process, our system should satisfy some preconditions. First, the components should be developed. Secondly, the unit tests of each component should be successful. Furthermore, if any bug was detected during the unit test, it should be fixed before the integration tests.

Besides, the following documents should have been completed, reviewed, approved to do unit testing phase, for instance, the Requirements Analysis and Specification Document and the Design Document. W.R.T integration testing phase, we assume that the project has already being code-complete. And there are no missing features or media elements. The product satisfies the performance and memory requirements specified by the Functional Spec. All priority bugs have been fixed and closed. Internal documentation has been updated to reflect the current state of the product.

Also, we should as far as possible to keep the percentage of completion of every component with respect to its functionalities as:

- 100% for the Data Persistence component
- At least 90% for the **Car Management** subsystem
- At least 90% for the **Reservation and Billing Management** subsystem
- At least 80% for the **System Administration and Account Management** subsystems
- At least 70% for the **Client Application**

2. Elements to be Integrated

As we presented before in the design document our system is composed by many components and units. So in this section, we will list all the main components which will be integrated in this phase. And we concern the integration phase as two levels of abstraction:

- High-level components integration testing
- Lower-level components integration testing

For high-level integration testing, as we introduced 3-tier structure to build our system(PowerEnjoy) in Design Document so we will follow the structure to

process the testing, 1. Presentation tier (mobile client, web client, web server), 2. Logic tier (application server), 3. Persistence tier (DB server).

At lower-level integration testing, we decided to integrate those components which are highly depending on one another to offer the higher level functionalities of PowerEnjoy. In this case, these components will be involved : Ride controller, Bill controller, Reservation controller, Economic controller. And we assume that the car controller and charge station and the interaction between car and charge station will be well integration tested by third part.

3. Integration Testing Strategy

The goal of integration testing is to uncover errors that may arise when two or more components are integrated together. As stated before, unit tests are performed and validated before conducting the integration tests. The latter provides assurance that bugs arise from the integrations and not from the components (units) themselves.

We opted for a bottom-up strategy because of many reasons. First, we are using a lot of already-developed or commercially available components that interact directly with our low level components. Since those components are already tested, they represent reliable building blocks to integrate from. Second, using a bottom-up approach removes the need to develop stubs that are needed in a top-down approach. In addition to that, our main datasources (database and car parameters) are already tested which removes the need for stubs. Furthermore, the low-level modules are the most critical in our application (model, localization and car communication) and almost all other components depend on them. So the bottom-up approach ensures that these components are integrated first making sure that they are bug-free earlier. In case errors are uncovered, they can be fixed early in the process.

4. Sequence of Component/Function Integration

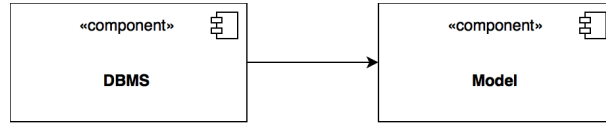
In the section below, we present the sequence of component integrations following the bottom-up approach described above. The arrows describe the dependencies; an arrow going from component A to B means that component A depends on component B.

The model is present in all the integrations because it is an important component as all the other components use it to persist data.

Model

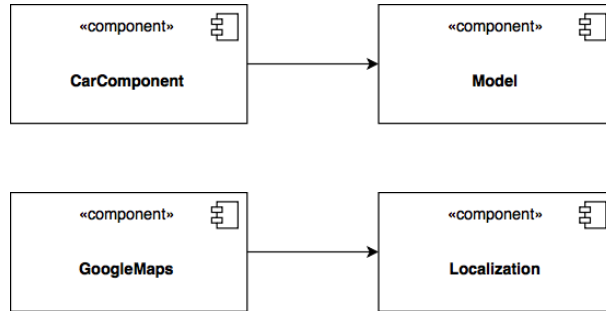
Since the model is the most depended-on component of the system, we will start the integration test with it. In this test, we have to make sure that all

operations (Create, Read, Update and Delete) are successfully transferred to the DBMS and applied in the database. This test checks also the data integrity.



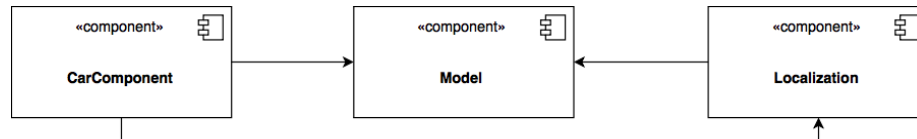
Google Maps & Car Component

In the process of integrating external components, the next step is integrating the Car Component and the Google Maps Component with the model and Localization Component respectively. The goal is to have these integrations before starting the integration of components we developed. The Car Component is the component that is made available by the car maker to communicate with the car and get all the variables. The Google Maps component is the component used to the translation of GPS Coordinates into places and also other localization concerns.



Car management subsystem

The Car Management subsystem takes care of gathering data about cars and persist it in the database. It is the system that makes sure of the constant communication with cars. It is composed of three elements: Car Component, Localization and Model. The Car Component and Localization should be integrated with the model to ensure data persistence. The Car Component use the Localization one to calculate distances and localizations.

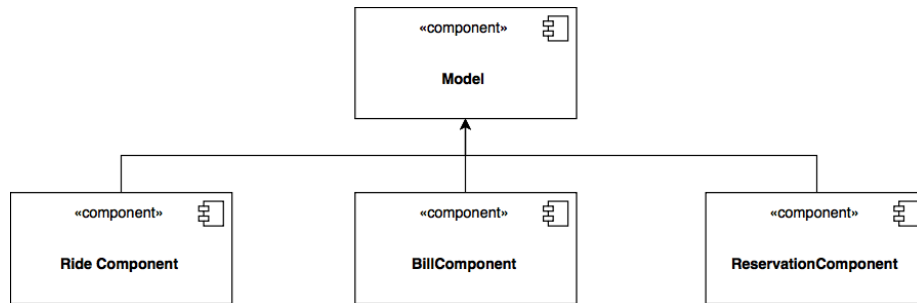


Operations management subsystem

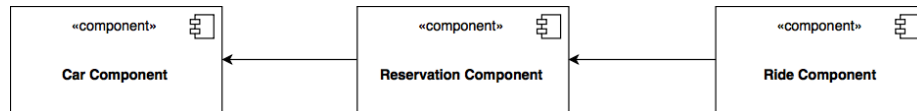
This subsystem represents the core of daily operations performed by PowerEnjoy. It takes care of reservations, billing and rides. Concerning the components of this subsystem. We start by integrating the reservation and bill components. Then we integrate bill component and ride component.



We have also integrate with components that are not in this subsystem. The first one that is important is the model because it is used in data persistence of all the components. All the components need to be integrated with it. The other one is the economic component and localization one.



The process of going from a reservation to an actual ride require the gathering of car informations (locking, distance travelled, charge...). Therefore, the reservation, ride and car components need to be integrated as well in this step.

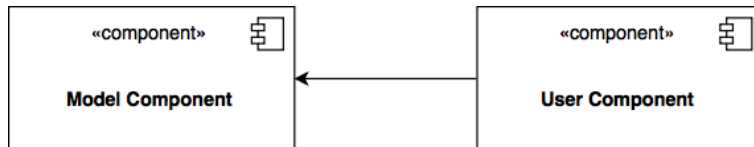


We need to integrate the economic component, the one that takes care of suggesting dropping point for discounts, with the localization component. The localization component is the one that manages the safe zones and charging stations.

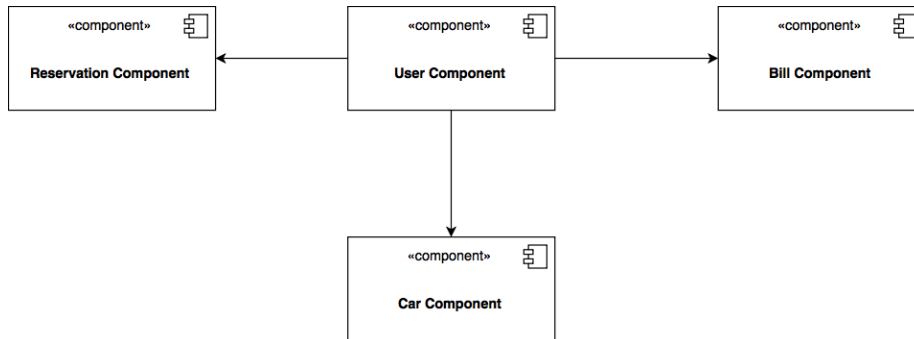


Account management subsystem

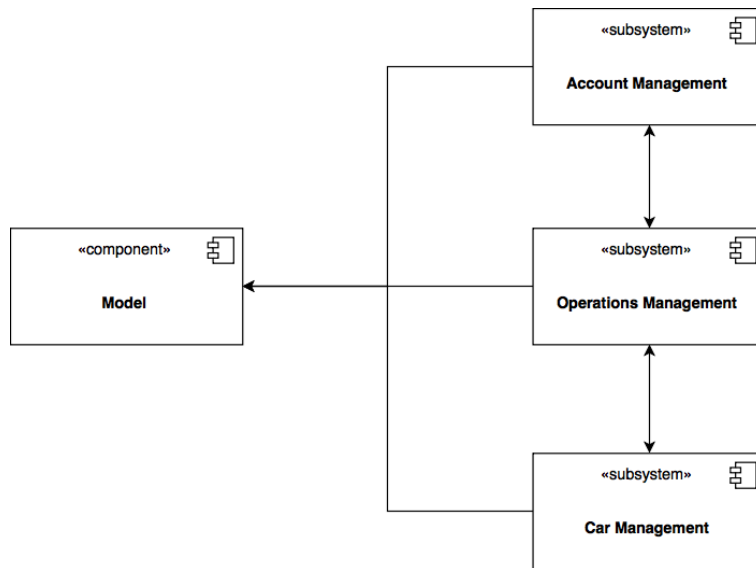
This subsystem takes care of all the account operations including login, creation and modification. For persistence matters, we start by integrating with the model.



The user component needs to be integrated with the following components: car, reservation and bill components.



As soon as the previous integrations are done, we can process the subsystems integration. The following diagram shows a high-level integration between the subsystems.



3. Individual Steps and Test Description

1. External Systems

1. DBMS, Model

For each entity that we have, we need to check if the entity is saved entirely in the database. We need also to perform data validation on the entities. We also need to be able to perform all CRUD operations. The following test should be performed to all entites.

saveEntity(Entity)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullArgumentException
An invalid entity	Throw InvalidEntityException
A valid entity	Return ID of saved entity

getEntity(EntityID)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullArgumentException
An inexistent ID	Throw InexistentEntityException
An existent ID	Return entity

updateEntity(ID, Entity)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullArgumentException
An inexistent ID	Throw InexistentEntityException
An existent ID and invalid entity	Throw InvalidEntityException
An existent ID and valid entity	Return success message

deleteEntity(EntityID)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullArgumentException
An inexistent ID	Throw InexistentEntityException
An existent ID	Return success message

2. Google Maps Component, Localization

getPlaceName(GPSCoordinates)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullPointerException
Valid GPS Coordinates	Name of the place/street

getPlaceName(GPSStaart, GPSEnd)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullPointerException
Valid GPS Coordinates	Return the distance

3. CarComponents, Model

updateCarLocalization()	
<i>Input</i>	<i>Effect</i>
No input	All the cars information is updated in the database

2. Car Management System

1. Localization, Model

getLocalization(map)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullPointerException
A non-null parameter	Return localization

getCarLocalization(carID)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullPointerException
A inexistent carID	Throw InvalidArgumentValueException
A valid parameter	Return the current localization information of the car

2. Car Component, Model

getCarStatus(carID)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullPointerException
A non-null parameter	Return the car's status and details

getReservedCarStatus(carID, userID, reservationID)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullPointerException
carID and userID and reservationID are invalid	Throw InvalidArgumentValueException
carID != reservation(info).carID	Throw InvalidArgumentValueException
Valid parameters	Return details and current status of the car to the user

isAvailableCar(carID)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullPointerException
A non-null parameter	Return true or false

3. Car Component, Localization Component

updateCarLocalization(carID, localization)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullPointerException
A inexistent carID	Throw InvalidArgumentValueException
A valid parameter	Return the current localization infromation of the car

isCarInSafeArea(carID, localiaztion)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullPointerException
A non-null parameter	Return true or false

3. Operations Management System

1. Reservation Component, Model

insertReservation(reservation)	
<i>Input</i>	<i>Effect</i>
A null parameter	A NullPointerException is raised.
By means of Model, A reservation with an id already existent in the database	An InvalidArgumentValueException is raised.
Formally valid arguments	By means of Model, an entry containing the reservation details inserted into the database.

deleteReservation(reservation)	
<i>Input</i>	<i>Effect</i>
A null parameter	A NullPointerException is raised.
By means of Model, A reservation with an id already existent in the database.	An InvalidArgumentValueException is raised.
Formally valid arguments	By means of Model, an entry containing the reservation data is deleted from the database.

updateReservationList(reservationList)	
<i>Input</i>	<i>Effect</i>
A null parameter	A NullPointerException is raised.
An empty array	A NullPointerException is raised.
An array containing somenull values	An InvalidArgumentValueException is raised.
An array of non-null, but inexistent reservations	An InvalidArgumentValueException is raised.
An array of valid and existing reservations	By means of Model, the corresponding entries in the database are updated to set the reservation as completed.

2. Bill Component, Model

getBillInfo(UserId, ride)	
<i>Input</i>	<i>Effect</i>
A null user id	A NullArgumentException is raised.
An invalid user id	An InvalidArgumentValueException is raised.
A non-existing ride	A NullArgumentException is raised.
A valid user id and ride	By means of Model, Th bill information can be get from database.

3. Ride Component, Model

- startRide

startRide(carId, currentLocation)	
<i>Input</i>	<i>Effect</i>
A null location	A NullArgumentException is raised.
A non-existing location	An InvalidArgumentValueException is raised.
A location far from the current location	An InvalidArgumentValueException is raised.
A current location	By means of Model, the corresponding location in the database are selected to start rides.

updateRideInfo(CarId,RideInfoAvailable)	
<i>Input</i>	<i>Effect</i>
A null location	A NullArgumentException is raised.
A non-existing CarId	An InvalidArgumentValueException is raised.
A set of valid parameters	By means of Model, the corresponding ride information in the database are updated

getRideInfo(CarId,RideInfoAvailable)	
<i>Input</i>	<i>Effect</i>
A non-existing CarId	An InvalidArgumentValueException is raised.
A set of valid parameters	By means of Model, Returns the stored ride information in the database

endRide(CarId, currentLocation)	
<i>Input</i>	<i>Effect</i>
A null location	A NullArgumentException is raised.
A non-existing CarId	An InvalidArgumentValueException is raised.
A valid CarId and current-Location, the Car is on a ride and current Location is inside city	By means of Model, the ride is considered closed and is,nal-ized in the database.

4. Reservation Component, Car Component

- ReservationCar

reserveCar(clientId, clientLocation)	
<i>Input</i>	<i>Effect</i>
A null parameter	A NullPointerException is raised.
A clientId not correctly formatted	An InvalidArgumentValueException is raised.
A client location whose co-ordinates are invalid	An InvalidArgumentValueException is raised.
A destination whose co-ordinates are invalid	An InvalidArgumentValueException is raised.
A passengerLocation outside the city	An InvalidArgumentValueException is raised.
A valid set of parameters	A new reservation is created and handled

isCarAvailable(reservation, location)	
<i>Input</i>	<i>Effect</i>
A null parameter	A NullPointerException is raised.
An inexistent location	An InvalidArgumentValueException is raised.
A zone with invalid locations	An InvalidArgumentValueException is raised.
A valid set of parameters	Returns true if a Car driver is available to serve the reservation, false otherwise.

5. Reservation Component, Bill Component

checkBillInfo(reservation, ride)	
<i>Input</i>	<i>Effect</i>
A null parameters	A NullArgumentException is raised.
An invalid parameters	An InvalidArgumentValueException is raised.
A non-existing ride or reservation	A NullArgumentException is raised.
A valid reservation and ride	From the reservation and ride, the corresponding bill can be check by user

ConfirmBillInfo(reservation, ride)	
<i>Input</i>	<i>Effect</i>
A null parameters	A NullArgumentException is raised.
An invalid parameters	An InvalidArgumentValueException is raised.
A non-existing ride or reservation	A NullArgumentException is raised.
A valid reservation and ride	From the reservation and ride, the corresponding bill needs to be confirmed when the user check-out.

6. Reservation Component, Ride Component

RideOfReservation(Client, reservation)	
<i>Input</i>	<i>Effect</i>
A null parameter	A <code>NullArgumentException</code> is raised.
A client location whose coordinates are invalid	An <code>InvalidLocationException</code> is raised.
An inexistent car driver	An <code>InvalidArgumentValueException</code> is raised.
A valid car driver and a reservation	Returns the reservation information and ride information of passengers location, destination and passengers number and whether share car or not.

7. Bill Component, Ride Component

calculateBill(ride)	
<i>Input</i>	<i>Effect</i>
A null parameters	A NullArgumentException is raised.
An invalid parameters	An InvalidArgumentValueException is raised.
A non-existing ride	A NullArgumentException is raised.
A valid ride	According to the formulas we write on DD the bill can be calculated

getBillInfo(UserId)	
<i>Input</i>	<i>Effect</i>
A null location	A NullArgumentException is raised.
A location whose coordinates are invalid	A InvalidArgumentException is raised.
A non-existing car id	A NullArgumentException is raised.
UserId is valid	User can get the bill information which stored in the database

8. Ride Component, Localization Component

MonitorCurrentLocation(CarId, location)	
<i>Input</i>	<i>Effect</i>
A null location	A NullArgumentException is raised.
A location whose coordinates are invalid	An InvalidArgumentValueException is raised.
A non-existing car id	A NullArgumentException is raised.
CarId is valid, location is inside city	By GPS ,its status is set to available and its location is written in the database and monitor on the user application of his current ride.

getRideRoute(CarId,location)	
<i>Input</i>	<i>Effect</i>
A null location	A NullArgumentException is raised.
A location whose coordinates are invalid	A InvalidArgumentException is raised.
A non-existing car id	A NullArgumentException is raised.
CarId is valid, location is stored in the database	From the location user can get the ride route exists in the database

9. Economic Component, Localization Component

getEconomicInfo(UserId)	
<i>Input</i>	<i>Effect</i>
A non-existing User id	A NullArgumentException is raised.
UserId is valid	System will remind the economic information when the user make a reservation and on his location who wants to share cars

calculateEconomicRide(UserId, location)	
<i>Input</i>	<i>Effect</i>
A null location	A NullArgumentException is raised.
A location whose coordinates are invalid	A InvalidArgumentException is raised.
A non-existing User id	A NullArgumentException is raised.
UserId is valid	The ride can be paid economically because of sharing cars, the percentageof discount can be calculate by the formulas we given on the DD

MonitorEconomicRide(UserId, location)	
<i>Input</i>	<i>Effect</i>
A null location	A NullArgumentException is raised.
A location whose coordinates are invalid	A InvalidArgumentException is raised.
A non-existing User id	A NullArgumentException is raised.
UserId is valid	The economic ride information can be monitor on the userâ s application,

4. User Management System

1. User Component, Model

createUser(user)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullPointerException
A non-null parameter	Insert the user's information into the DB

updateUserInfo(user)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullPointerException
A non-null parameter	The user's information is updated in the DB

feedbackRequest(feedback)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullPointerException
A non-null parameter	The user's feedback information is inserted into the DB

updateUserPassword (userID, password, newPassword)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullPointerException
UserID doesn't match password	Throw InvalidArgumentValueException
A valid parameter	Update the user's password in DB

2. User Component, Car Component

unlockCar (userID, reservationInfo)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullArgumentException
UserID doesn't match the reservationInfo	Throw InvalidArgumentValueException
Valid parameter	Unlock the car

getAvailableCarList (userID, localization)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullArgumentException
An invalid localization	Throw InvalidArgumentValueException
A valid parameter	Return the available car list to the user

3. User Component, Bill Component

getBill(userID, ride)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullArgumentException
UserID dosen't match the ride information	Throw InvalidArgumentValueException
A valid parameter	Return the bill details to the user

checkHistoryBill(userID, date)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullArgumentException
A non-null parameter	Return the bill details to the user for those dates

4. User Component, Reservation Component

reservationRequest(userID, reservationInfo)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullPointerException
A non-null parameter	Return true

cancelReservation(userID, reservationID)	
<i>Input</i>	<i>Effect</i>
A null parameter	Throw NullPointerException
A reservationID that doesn't exist	Throw InvalidArgumentValueException which means the user didn't make a reservation
Valid userID and reservation ID	Remove the reservation from DB and add this operation into the log

4. Tools and Test Equipment Required

4.1 Tools

There are a ton of tools out there for automated testing, and just about any tool can be used for just about any testing methodology.

Testing the system is not how much on the technology we use, but on the difference between functional testing and system testing and on how we use tools in functional testing. Functional or feature testing is the process of testing newly written code to ensure it functions as designed. System or integration testing is the process of ensuring individual functional areas integrate (play nicely together). For what concerns the business logic components running in the Java Enterprise Edition runtime environment, we are going to take advantage of few tools to implement different functions in testing.

Mockito: Usually, Mockito helps us to cut all dependencies in unit tests, but it can be useful also in the integration tests where programmers often want to have all real applications layers.

In our case, we will use Mockito to do this testing on the main server and applications. To do so we want to have our integration tests, but without a real call to database. Because our dao layer is using some native queries to get data from database, and some embedded database doesn't support some query syntax.

Arquillian: A new testing framework developed at JBoss.org, empowers the developer to write integration tests for business objects that are executed inside a container or that interact with the container as a client. We will apply it to execute tests against a Java container in order to check that the interaction between a component and its surrounding execution environment is happening correctly.

Rest-Assured: As we used RESTful architecture pattern in our system and it(REST-API) as one of our system's core methods to make sure the interaction between system and server goes well.

We will use Rest-Assured to test the implementation of REST-API instead of using JUnit which is usually used to test it by JAVA programmers. From some point of views the latter is a white box test, programmers can clearly know which class is being tested, which method is being used but all of this are not from users' view to test which API is being used. REST Assured is a simple Java library for testing of REST services, it can write code directly and initiate the HTTP request to the server and verify the returned result.

4.2 Equipment required

4.2.1 Server side

In order to ensure the stability and fluency of the system, we are going to use **B/S** model to develop our server terminal. From economy, scalable, extensible, easy to develop and easy to maintain points we propose to buy a subscription to **Amazon EC2** rather than a physical server.

4.2.2 Client side

For our system (PowerEnjoy) it will be applied on both mobile-phone terminals and PC terminals (browsers). In order to increase the user experience, we will as far as possible to realize that the system can be used in different operation systems and vary devices. So our system will support Android (2.0 later), IOS(7.0 later), WindowsPhone(8.0 later) operation systems. And it will be ensure that the system can be run on at least 80% of different sizes of display screens. As for browsers, we request Mozilla Firefox, v36 above, Microsoft Edge, Internet Explorer for Windows, v9.0 above, Google Chrome, v55.0, Opera, v40 above, and Safari, v5 above.

5. Program Stubs and Test Data Required

1 Program Stubs and components

As we said before, we are going to adopt a bottom-up approach to component integration and testing. In this part, the purpose of these stubs is to write on a log that they have correctly received the messages.

User component

- This testing module will invoke the methods exposed by the user subcomponent, in order to test its interaction with the Model, the car component, the reservation component and bill component.

Car component

- This testing module will invoke the methods exposed by the car subcomponent, in order to test its interaction with the Model, the reservation component, the localization component and the ride component and the Mapping component.

Bill component

- TThis testing module will invoke the methods exposed by the bill subcomponent, in order to test its interaction with the reservation component, the user component and the economic component.

Ride component

- This testing module will invoke the methods exposed by the ride subcomponent, in order to test its interaction with the reservation component, the bill component and the economic component.

Economic component

- This testing module will invoke the methods exposed by the ride subcomponent, in order to test its interaction with the the bill component and the localization component.

Model

- This testing module will invoke the methods exposed by the Model, in order to test its interaction with all the other components and database.

Localization component

- This testing module will invoke the methods exposed by the localization subcomponent, in order to test its interaction with the car component and model.

Reservation component

- This testing module will invoke the methods exposed by the reservation subcomponent, in order to test its interaction with the the user component and car component.

2. Test Data

To test User component

In order to test the User component, we assume both valid and invalid users to exhibit the following problems:

- Null username
- Invalid username
- Null password
- Invalid password

To test Car component In order to test the Car component, we assume both valid and invalid users to exhibit the following problems: - A null parameter - A Invalid parameter - Driving license not compliant with the legal format

To test bill component - A null user id - A Invalid parameter - A non-existing ride or reservation

To test Ride component - An invalid user id - A non-existing car id - A null location - A non-existing location - UserID doesn't match the ride information

To test Economic component - A non-existing User id - A ride which is unfinished

To test Localization component - A null parameter - A non-existing location - A location far from the current location - A passenger location whose coordinates are invalid

To test Reservation component - A null parameter - An array containing some null values - An array of non-null, but inexistent reservations

6. Effort Spent

Reda Aissaoui

- 08/01/2017 2h
- 11/01/2017 3h
- 14/01/2017 4h
- 15/01/2017 6h

Jinling Xing

- 08/01/2017 5h
- 10/01/2017 2h
- 11/01/2017 0.5h
- 13/01/2017 7h
- 14/01/2017 3h
- 15/01/2017 0.5h

Lidong Zhang

- 08/01/2017 2h
- 09/01/2017 3h
- 10/01/2017 3h
- 13/01/2017 4h
- 14/01/2017 3h