PowerEnjoy - Integration Test Plan Document

January 14, 2017

A.Y. 2016/2017
Bolshakova Liubov, matr. 876911
Gao Xiao, matr. 876265
Kang Shuwen, matr. 876245

Contents

1 INTRODUCTION

1.1 Purpose

The integration test plan document is intended to describe how to accomplish the integration test. At first a high-level description of integration is provided to guide the specific testing. This document also includes the schedule and the processes of components which should be tested ,as well as the descriptions of the tools which are used.

1.2 Scope

The total project is for design a software named *PowerEnJoy* which is an auto-servicing of cars for citizens. The aim of our software is not only supports the management of cars and provides a good quality of services to simplify the transportation, but also supports the payment and reservation services.

1.3 Glossary

- Car: the cars that supplied for the car-sharing service in the *PowerEnJoy* system.
- Car information: the basic information that helps guests and users to make decisions, include the dump energy, location information, distance to the setting location, the passenger capacity.
- Starting position: the current position of user or the positions user input to start a ride.
- Available car: the car has dump energy more than 50
- Available queue: a queue that maintains available cars
- Sensors: the GPS and power plug sensor, weight sensor, display screen, battery sensor, door state sensor, locks of door in the car, and the sensor on the power grid.
- System: the whole system, which include the electric devices and the *PowerEnJoy* system background.
- Ride: in this system, the ride process is started with ignite the engine and ended with all passengers leave the car.

2 INTEGRATION STRATEGY

2.1 Entry Criteria

In this section, we define some entry criteria, which are the conditions hold before the integration phase takes place. And these entry criteria are aimed at ensure the feasibility and correctness of the integration tests.

- Reference Documents: The *Requirement Analysis and Specification Document* and *Design Document* are completed.
- Code: Every component to be tested is code completed, which means that all functionalities to be tested have been completely implemented.
- Documentation: The documentation for each class and methods is fully provided and is understandable to testers.
- Unit Test: Every component is unit-tested by using JUnit and is bug free.
- Data: Test dataset is provided and functioned.
- Tester: Manual testers are available.

2.2 Elements to be integrated

In this section we identify the elements among which the integration test will be performed, by referring to the low-level components specified in the *Design Document*(as shown in Figure 1). For the sake of explicitness, we present the components according to the 3-tier architecture we designed in *Design Document*.

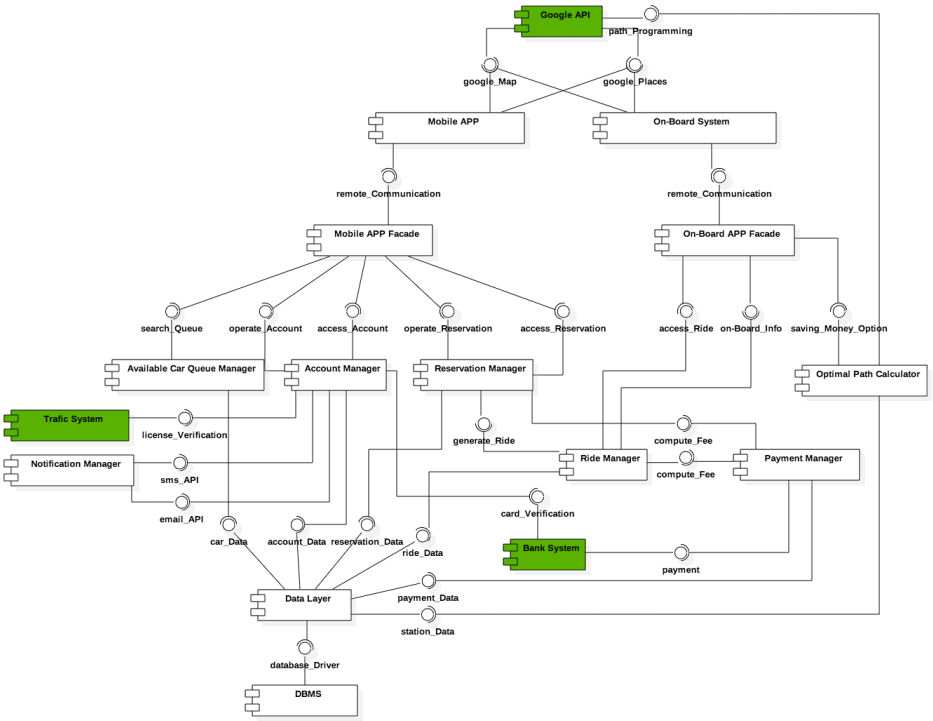

Figure 1: Low Level Component Diagram

2.2.1 GUI Tier

Since the two components for presentation included in the *GUI* tier work independently, there are basically two elements to be integrated:

- Mobile APP component
- On-Board System component

2.2.2 Application Tier

The *Application* tier provides the entire business logic of the system. We list all the elements to be integrated as follows:

- Mobile APP Facade
- On-Board APP Facade
- Available Queue Manager
- Account Manager
- Reservation Manager

- Optimal Path Calculator
- Notification Manager
- Ride Manager
- Payment Manager
- Data Layer

2.2.3 Data Tier

The *Data* tier contains the database used for data storage and the commercial database management system, namely the DBMS. These two components are coupled together and work properly according to the entry criteria we defined. For this reason, the only element to be integrated would be the DBMS component.

2.2.4 External Components

Except of those components contained in the 3-Tier architecture, there is also a group of external components necessarily to be integrated into the *PowerEnJoy*, system, which are

- Google API,
- Bank System and
- Traffic System.

2.3 Integration Test Strategy

In this section, we discuss about the ***bottom-up*** strategy, which we used for integration test, by explaining how we make the choice.

2.3.1 System Partition

In order to better organize the test, we divide the entire system into three subsystems. Each subsystem contains several elements identified previously.

- Application Server: This subsystem works as the core of the entire system, and it contains all the elements we identified in the *Application Tier* and in the *External Components*.
- Mobile Client: This subsystem contains one of the components in *GUI Tier*,namely the *Mobile APP*, as well as the *Google API* in the *External Components*.

- On-Board Client: Similar to the previous one, this subsystem includes the other components in *GUI Tier*,which is the *Mobile APP*, together with the *Google API*.

2.3.2 Integration Process

Since we have divided the entire system into three subsystems, our integration process will normally be performed in two phases:

- Local Integration: In this phase, we perform the integration of components within the subsystem.
- Overall Integration: In this phase, three individual subsystems will be integrated.

2.3.3 Strategy Comparison

- *Bottom-up strategy*: With this approach, we should start from the components in the bottom level,which have least dependency of other components and provide fundamental services to others, and add other components step by step. This approach is quite suitable to our system since we can reduce the total amount of needed stubs for integration accomplishment.
- *Top-down strategy*: This kind of strategy is suggested in case of timing problem, because it is more focused on testing the functionalities provided to the end-user rather than considering about the real performance of low-level components.
- *Sandwich strategy*: This kind of strategy is more difficult to plan and implement in case of high-interdependent systems.

2.4 Sequence of Components

In this section we discuss about the integration sequence for the *PowerEnJoy* system. Firstly, we preform the local integration individually in three subsystems that we have defined. And after that, we implement the overall integration for the entire system, namely carry out the integration between subsystems. Since the Application Server contains the most number of components to be integrated as well as a complex structure, our focus would be on the discussing about this particular subsystem.

2.4.1 Application Server

To deal with the order of integration for such many components in this subsystem, we first illustrate the dependency relation between each component, which is shown in Figure 2. And thanks to the dependency relation, we

can simply distinguish six levels (as shown in Figure 3) for the Application Server, each of them contains several individual components which are independent with each other, so that the components within the same level can be tested in parallel. Between each two levels, there exists dependency, which means that the higher level cannot be integrated until the lower level works properly.

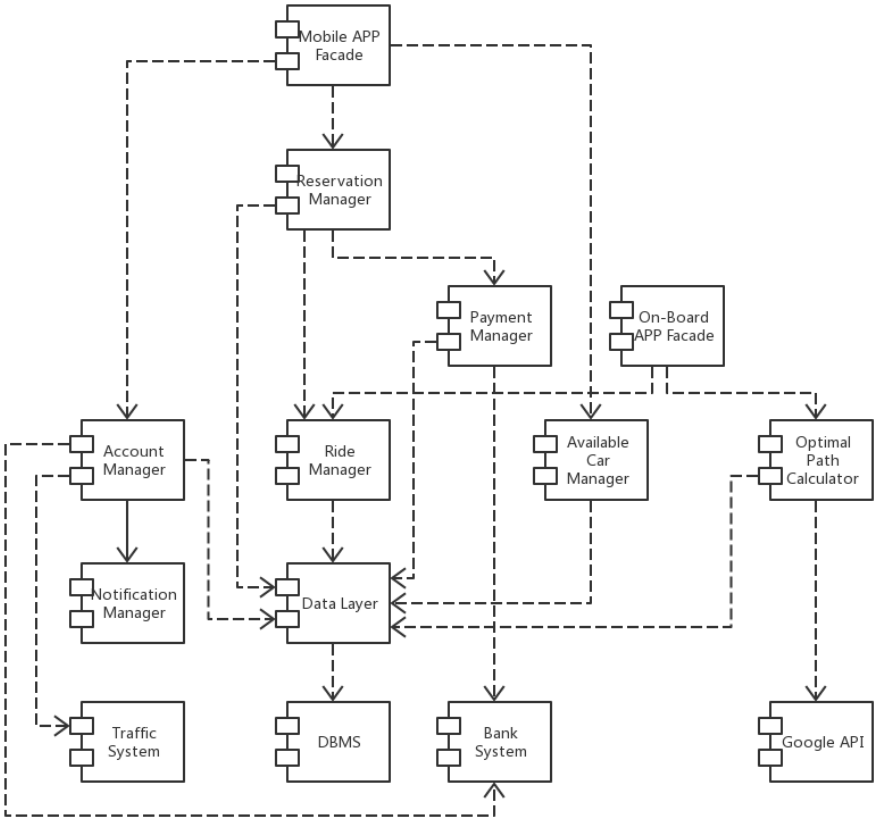

Figure 2: Dependency Relation

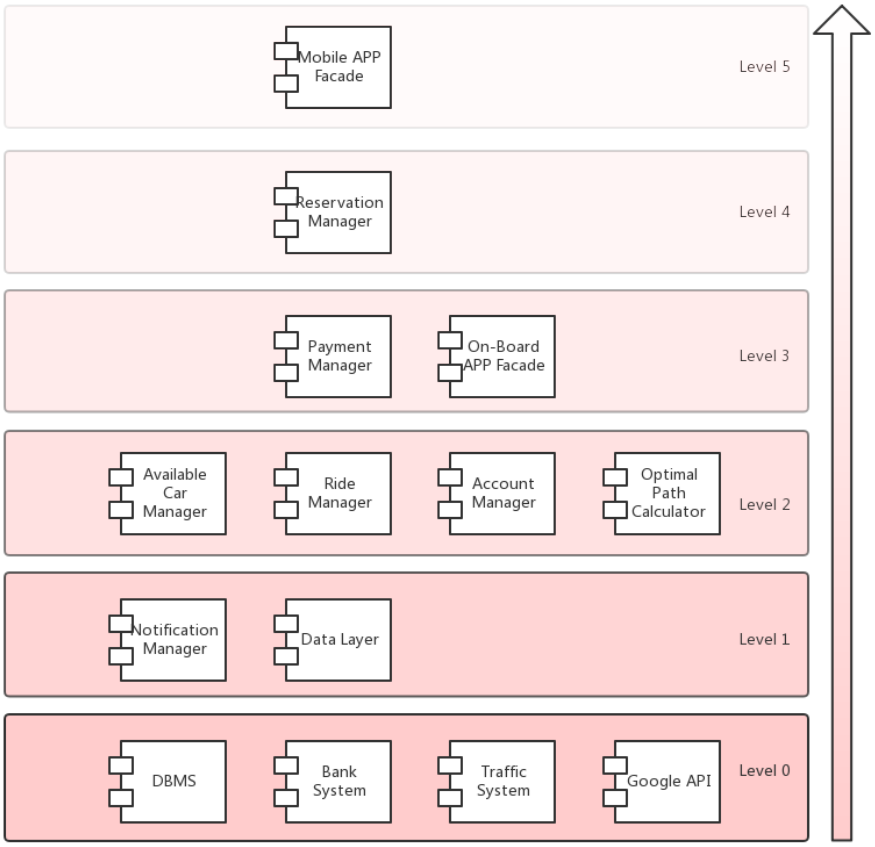

Figure 3: Bottom-up Level

2.4.2 Mobile Client

There are simply two components to be integrated within this subsystem. The dependency relation among them is illustrated in Figure 4, according to which the integrated order might be decided.

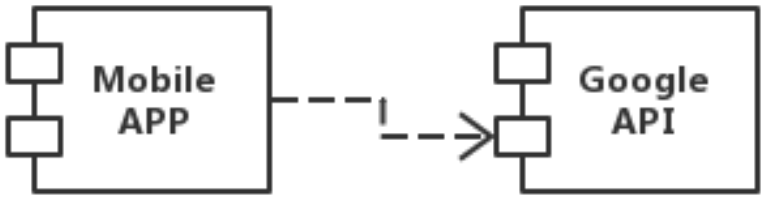

Figure 4: Dependency Relation

2.4.3 On-Board Client

Similar to the previous one, this subsystem also contains two components to be tested, and the sequence can be determined by means of Figure 5.

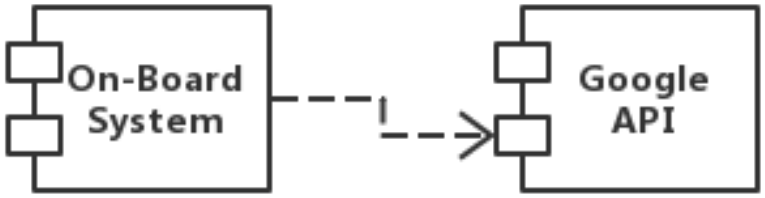

Figure 5: Dependency Relation

2.4.4 Overall System

In this phase, we have finished the individual subsystem testing and move on to the overall system integration. Since each subsystem is tested, we can assume that every functionality that we designed before has been implemented and works properly. Therefore the integration of subsystem would focus on the communication between each other. Again, we present the dependency relation of three subsystems in Figure 6, and accordingly figure out the bottom-up level for them (as shown in Figure 7. Not surprisingly, the subsystems within the same level can be tested in parallel.

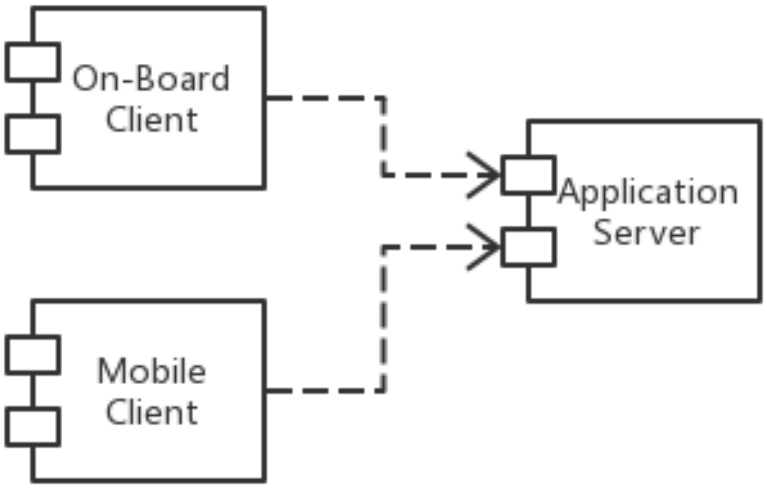

Figure 6: Dependency Relation

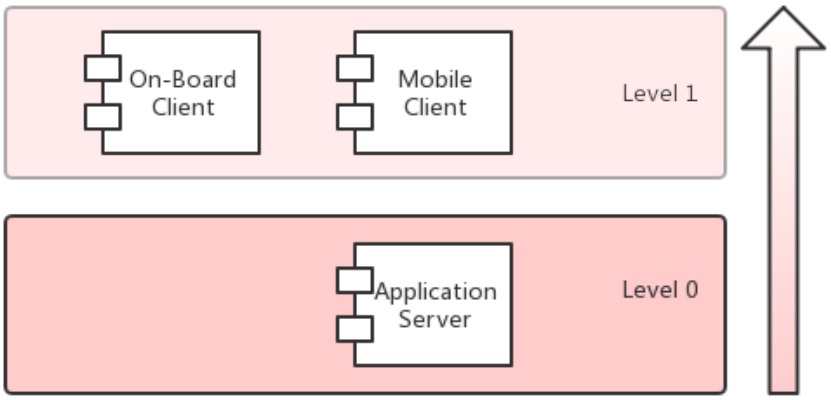

Figure 7: Overall Bottom-up Level

2.4.5 Integration Order

In this section we provide an alternative solution for the integration order of the entire system, which is illustrated in the table bellow. Notice that the

integration order within the same bottom-up level might be decided arbitrarily since they are in parallel, therefore there exists numbers of solutions for the actual integration order which can be accepted.

ID	Component	Dependency	Subsystem
I01	DBMS	NULL	Application Server
I02	Notification Manager	NULL	Application Server
I03	Data Layer	DBMS	Application Server
I04	Available Car Manager	Data Layer	Application Server
I05	Ride Manager	Data Layer	Application Server
I06	Account Manager	Data Layer	Application Server
I07	Optimal Path Calculator	Data Layer	Application Server
I08	Optimal Path Calculator	Google API	Application Server
I09	Payment Manager	Data Layer	Application Server
I10	Payment Manager	Bank System	Application Server
I11	On-Board APP Facade	Ride Manager	Application Server
I12	On-Board APP Facade	Optimal Path Calculator	Application Server
I13	Reservation Manager	Data Layer	Application Server
I14	Reservation Manager	Ride Manager	Application Server
I15	Reservation Manager	Payment Manager	Application Server
I16	Mobile APP Facade	Account Manager	Application Server
I17	Mobile APP Facade	Reservation Manager	Application Server
I18	Mobile APP Facade	Available Car Manager	Application Server
I19	On-Board System	Google API	On-Board Client
I20	On-Board System	On-Board APP Facade	On-Board Client
I21	Mobile APP	Google API	Mobile Client
I22	Mobile APP	Mobile APP Facade	Mobile Client

Table 1: Integration Order

3 INDIVIDUAL STEP AND TEST DESCRIPTION

For a better understanding we are going to show a high-level description of the integration tests that will be performed on PowerEnJoy system. The bottom-up strategy choice implies the use of drivers: every driver is associated to a specific software component, and is used to simulate the input coming from a already tested component,this ensure us to get rid of factors which are surely be right, then we can focus on the factors only associated to the testing component. The following tables are structured as:

- Test Case ID: an identifier for the analyzed integration test.
- Test Items: the couple of involved components; the arrow represent the depend on relationship.
- Input Specification: the high-level description of the input coming from the right element of the couple specified at Test Item row.
- Output Specification: the high-level description of the expected output of the input above.
- Environmental Needs: the conditions under which the test can be executed, in terms of:
  - already performed tests
  - already implemented drivers

3.1 Application Server

3.1.1 DBMS Integration Test

Test Case ID	I01
Test Item	DBMS
Input Specification	The test of DBMS is execute the data search function from every GUI in users' terminal.
Output Specification	Check if the correct data was responded according to the SQL search in the database.
Environmental Needs	The DBMS is connected with the system, the database Driver already been implemented.

3.1.2 Notification Manager Integration Test

Test Case ID	I02
Test Item	Notification Manager
Input Specification	Simulate the Notification Manager component, the test should also cover the exceptional and edge cases related to the sms API and email API interface.
Output Specification	Check if the email addresses or telephone numbers can get the present information.
Environmental Needs	The Notification Manager should connected to the system.

3.1.3 Data Layer Integration Test

Test Case ID	I03
Test Item	Data Layer $\rightarrow$ DBMS
Input Specification	Manipulate the DBMS with the data from the Data Layer or search the data in DBMS by the requests from Data Layer.
Output Specification	Check if the correct methods are invoked, along with the correct value and parameter types.
Environmental Needs	The DBMS must have been already implemented.

3.1.4 Available Car Manager Integration Test

Test Case ID	I04
Test Item	Available Car Manager → Data Layer
Input Specification	Simulate the Available Car Manager component with the input from the Data layer,the test should also cover the exceptional and edge cases related to the search Queue interface and car Data interface.
Output Specification	Check if the correct methods are used; Check if the parameters are in the correct type according to the methods, as well as the correctness of values.
Environmental Needs	The Data Layer driver must have been already implemented.

3.1.5 Ride Manager Integration Test

Test Case ID	I05
Test Item	Ride Manager→ Data Layer
Input Specification	Simulate the Ride Manager component with the input from the Data layer, the test should also cover the exceptional and edge cases related to the access Ride and access Fee Interface and ride Data interface.
Output Specification	Check if the correct methods are used; Check if the parameters are in the correct type according to the methods, as well as the correctness of values.
Environmental Needs	The Data Layer driver must have been already implemented.

3.1.6 Account Manager Integration Test

Test Case ID	I06
Test Item	Account Manager →Data Layer
Input Specification	Simulate the Account Manager component with the input from the Data layer, the test should also cover the exceptional and edge cases related to the operate Account and access Account Interface and account Data interface.
Output Specification	Check if the correct methods are used; Check if the parameters are in the correct type according to the methods, as well as the correctness of values.
Environmental Needs	The Data Layer driver must have been already implemented.

3.1.7 Optimal Path Calculator Integration Test

Test Case ID	I07
Test Item	Optimal Path Calculator →Data Layer
Input Specification	Simulate the Optimal Path Calculator component with the input from the Data layer, the test should also cover the exceptional and edge cases related to the saving Money Option Interface and station Data interface.
Output Specification	Check if the correct methods are used; Check if the parameters are in the correct type according to the methods, as well as the correctness of values.
Environmental Needs	The Data Layer driver must have been already implemented.

Test Case ID	I08
Test Item	Optimal Path Calculator →Google API
Input Specification	Simulate the Optimal Path component with the input from the Google API, the test should also cover the exceptional and edge cases related to the saving Money Option Interface and google path Programming interface.
Output Specification	Check if the correct methods are used; Check if the parameters are in the correct type according to the methods, as well as the correctness of values.
Environmental Needs	The Google API must have been already connected to the system.

3.1.8 Payment Manager Integration Test

Test Case ID	I09
Test Item	Payment Manager→ Data Layer
Input Specification	Simulate the Payment Manager component with the input from the Data layer, the test should also cover the exceptional and edge cases related to the payment Interface and payment Data interface.
Output Specification	Check if the correct methods are used; Check if the parameters are in the correct type according to the methods, as well as the correctness of values.
Environmental Needs	The Data Layer driver must have been already implemented.

Test Case ID	I10
Test Item	Payment Manager →Bank System
Input Specification	Simulate the Payment Manager component with the input from the Bank System, the test should also cover the exceptional and edge cases related to the card Verification Interface,payment interface.
Output Specification	Check if the correct methods are used; Check if the parameters are in the correct type according to the methods, as well as the correctness of values.
Environmental Needs	The Bank System must have been already connected with the system.

3.1.9 On-Board APP Facade Integration Test

Test Case ID	I11
Test Item	On-Board APP Facade →Ride Manager
Input Specification	Simulate the On-Board APP Facade component with the input from the Ride Manager, the test should also cover the exceptional and edge cases related to the On-Board APP Facade Interface.
Output Specification	Check if the correct methods are used; Check if the parameters are in the correct type according to the methods, as well as the correctness of values.
Environmental Needs	The Ride Manager driver must have been already implemented.

Test Case ID	I12
Test Item	On-Board System Facade →Optimal Path Calculator
Input Specification	Simulate the On-Board System Facade component with the input from the Optimal Path Calculator. the test should also cover the exceptional and edge cases related to the on-Board Info Interface, saving Money Option interface and remote Communication interface.
Output Specification	Check if the correct methods are used; Check if the parameters are in the correct type according to the methods, as well as the correctness of values.
Environmental Needs	The Optimal Path Calculator driver must have been already implemented.

3.1.10 Reservation Manager Integration Test

Test Case ID	I13
Test Item	Reservation Manager→ Data Layer
Input Specification	Simulate the Reservation Manager component with the input from the Data Layer. the test should also cover the exceptional and edge cases related to the operate Reservation Interface,access Reservation interface.
Output Specification	Check if the correct methods are used; Check if the parameters are in the correct type according to the methods, as well as the correctness of values.
Environmental Needs	The Data Layer driver must have been already implemented.

Test Case ID	I14
Test Item	Reservation Manager →Ride Manager
Input Specification	Simulate the Reservation Manager component with the input from the Ride Manager. the test should also cover the exceptional and edge cases related to the Reservation Manager Interface and Ride Manager Interface.
Output Specification	Check if the correct methods are used; Check if the parameters are in the correct type according to the methods, as well as the correctness of values.
Environmental Needs	The Ride Manager driver must have been already implemented.

Test Case ID	I15
Test Item	Reservation Manager →Payment Manager
Input Specification	Simulate the Reservation Manager component with the input from the Payment Manager. the test should also cover the exceptional and edge cases related to the Reservation Manager Interface and Payment Manager interface.
Output Specification	Check if the correct methods are used; Check if the parameters are in the correct type according to the methods, as well as the correctness of values.
Environmental Needs	The Payment Manager driver must have been already implemented.

3.1.11 Mobile APP Facade Integration Test

Test Case ID	I16
Test Item	Mobile APP Facade →Account Manager
Input Specification	Simulate the Mobile APP Facade component with the input from the Account Manager. the test should also cover the exceptional and edge cases related to the Account Manager Interface and .
Output Specification	Check if the correct methods are used; Check if the parameters are in the correct type according to the methods, as well as the correctness of values. Environmental Needs
Environmental Needs	The Account Manager driver must have been already implemented.

Test Case ID	I17
Test Item	Mobile APP Facade →Reservation Manage
Input Specification	Simulate the Mobile APP Facade component with the input from the Reservation Manager. the test should also cover the exceptional and edge cases related to the Reservation Manager Interface and Mobile APP Facade interface.
Output Specification	Check if the correct methods are used; Check if the parameters are in the correct type according to the methods, as well as the correctness of values.
Environmental Needs	The Reservation Manager driver must have been already implemented.

Test Case ID	I18
Test Item	Mobile APP Facade →Available Car Manager
Input Specification	Simulate the Mobile APP Facade component with the input from the Available Car Manager. the test should also cover the exceptional and edge cases related to the Available Car Manager Interface and Mobile APP Facade interface.
Output Specification	Check if the correct methods are used; Check if the parameters are in the correct type according to the methods, as well as the correctness of values.
Environmental Needs	The Available Car Manager driver must have been already implemented.

3.2 On-Board Client

3.2.1 On-Board System Integration Test

Test Case ID	I19
Test Item	On-Board System →Google API
Input Specification	Simulate the On-Board System component with the input from the Google API. the test should also cover the exceptional and edge cases related to the On- Board System interface,Google API Interface.
Output Specification	Check if the correct methods are used; Check if the parameters are in the correct type according to the methods, as well as the correctness of values.
Environmental Needs	The Goolge API must have been already implemented.

Test Case ID	I20
Test Item	On-Board System→ On-Board APP Facade
Input Specification	Simulate the On-Board System component with the input from the On-Board APP Facade. the test should also cover the exceptional and edge cases related to the Reservation Manager Interface.
Output Specification	Check if the correct methods are used; Check if the parameters are in the correct type according to the methods, as well as the correctness of values.
Environmental Needs	The On-Board APP Facade driver must have been already implemented.

3.3 Mobile Client

3.3.1 Mobile APP Integration Test

Test Case ID	I21
Test Item	Mobile APP → Google API
Input Specification	Simulate the Mobile APP component with the input from the Google API. the test should also cover the exceptional and edge cases related to the Reservation Manager Interface.
Output Specification	Check if the correct methods are used; Check if the parameters are in the correct type according to the methods, as well as the correctness of values.
Environmental Needs	The Google API must have been already connected to the system.

Test Case ID	I22
Test Item	Mobile APP → Mobile APP Facade
Input Specification	Simulate the Mobile APP component with the input from the Mobile APP Facade. the test should also cover the exceptional and edge cases related to the Reservation Manager Interface.
Output Specification	Check if the correct methods are used; Check if the parameters are in the correct type according to the methods, as well as the correctness of values.
Environmental Needs	The Mobile APP Facade driver must have been already implemented.

4 TOOLS AND TEST EQUIPMENT REQUIRED

4.1 Tools

In order to make sure that each components within the system works appropriately, we use some effective testing tools, which will be discussed in detail in the next few paragraphs.

4.1.1 JUnit

JUnit is a unit testing framework for the Java programming language. It has a very important development in test-driven field. We make the decision to use it mainly because that it simplifies the unit testing of components.

4.1.2 Arquillian

Arquillian is a JUnit-based test framework. Indeed Arquillian represents an optimal solution in a such context where more components have to be tested and integrated among them. Thanks to this testing tool, the integration tests and the functional tests could be as simple as unit tests. In accordance with the design document, Java EE will be the main framework for the development of our system. That's why Arquillian should be the best solution in these case in which the integration among multiple enterprise components has to be tested.

4.1.3 Database

- MySQL Database
- MySQL Connector/J 5.1.40

4.2 Test Equipment

These testing devices would be used to test both the Mobile Application and the On-Board 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.

4.2.1 Mobile Phone

For Mobile Client testing:

- At least one Android smart phone, which is running Android operation system(version 4.0 (Ice Cream Sandwich) or higher).
- At least one IOS smart phone, which is running IOS operating system(version 7.1.2 or higher).
- At least one Windows smart phone, which is running Windows operating system.

4.2.2 On-Board Embedded System

On-Board physical equipment which is normally provided on the *PowerEnJoy* car.

5 PROGRAM DRIVERS AND TEST DATA REQUIRED

5.1 Program Drivers and Stubs

In our testing phase, we use a bottom-up approach to compose the components of the system. Therefore, we also use the bottom-up framework to compose the integration and testing.

In order to complete the testing of a single component in the Application Server subsystem, we are going to use a number of drivers to drive each component for simulating the real system. What's more, we need the drivers to perform the necessary function call for testing.

As for the other two subsystems, On-Board Client and Mobile Client, we need proper stubs to provide function call response. Thanks to the stubs, each of our three subsystems can be tested individually.

In the following paragraph we list the drivers and stubs which are used during the testing, and provide further discussion about them.

- **DBMS driver**

  This driver is in charge of performing data exchange with DBMS, in order to test the integration between DBMS and the Database.

- **Data Layer driver**

  This driver is designed to invoke the methods provided by Data Layer, aiming at checking the integration between Data Layer and it's sub-level components.

- **Notification Manager driver**

  For the aim of testing the Notification Manager, we need this driver to simulate a notification call to the Notification Manager, so as to check the performance.

- **Available Car Manager driver**

  This driver is in charge of invoking the methods provided by Available Car Manager, testing the integration state to the sub-level components.

- **Ride Manager driver**

  The main purpose of this driver is to simulate a ride generate call, in order to invoke the methods within Ride Manager. By checking the response from Ride Manager we can observe the integration status between Ride Manager and it's sub-level components.

- **Account Manager driver**

This driver is designed to perform data exchange with Account Manager, aiming at testing the integration of Account Manager together with it's sub-level components.

- **Optimal Path Calculator driver**

  This driver basically provides position informations to Optimal Path Calculator, in order to invoke the algorithm so as to test it's integration with the Data Layer.

- **Payment Manager driver**

  This driver is in charge of checking the integration state of Payment Manager with lower level components by simulating a payment call to Payment Manager.

- **On-Board APP Facade driver**

  This driver simulates the function call from On-Board System, and hence it tests the integration of On-Board APP Facade with lower level components.

- **Reservation Manager driver**

  This driver is designed to perform data exchange with Reservation Manager, aiming at testing the integration of Reservation Manager together with it's sub-level components.

- **Mobile APP Facade driver**

  This driver simulates the function call from Mobile APP, and hence it tests the integration of Mobile APP Facade with lower level components.

- **On-Board client stub**

  This stub simulates the response of On-Board APP Facade, aiming at checking the information display and interaction performance on the On-Board System.

- **Mobile Client stub**

  This stub is designed to provide simulated response from Mobile APP Facade, for the aim of checking the information display and interaction performance on the Mobile APP.

5.2 Test Database

In order to perform the tests a database configured as specified in the design document must be used. This database might contain dummy data, crafted in a way and number that allow for exhaustive tests on the data layer. The

structure of the test database must comply with the specification of the E-R diagram provided in the design phase, which is shown in Figure 8.

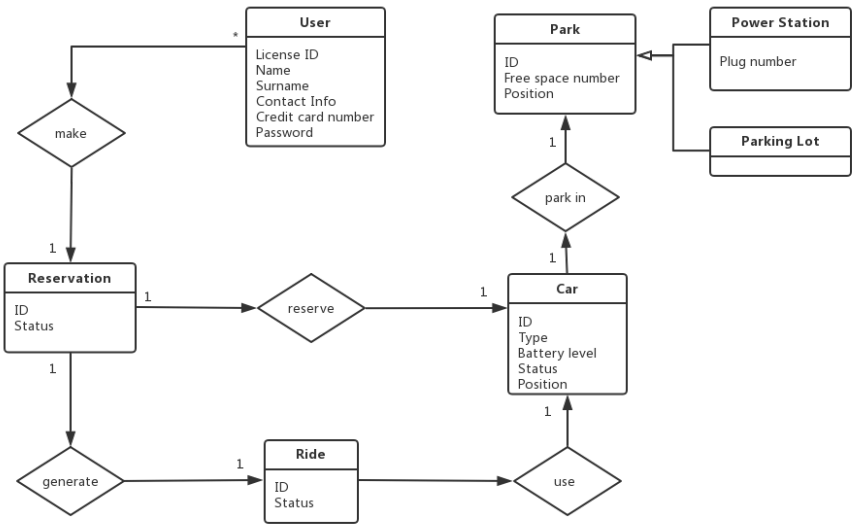

Figure 8: ER-diagram

6 EFFORT SPENT

Gao Xiao	20 Hours
Kang Shuwen	25 Hours
Liubov Bolshakova	10 Hours

7 REFERENCES

7.1 Reference Documents

- Specification Document Assignments AA 2016-2017
- RASD
- DD

7.2 Used Tools

The tools used to creat this document are:

- UMLStar: for UML models
- Github: for version control
- Latex: for typesetting
- ProcessOn: for network diagram