POLITECNICO DI MILANO

COMPUTER SCIENCE AND ENGINEERING

MASTER OF SCIENCE

# ITPD

# PowerEnjoy

Software engineering II project

Giovanni Agugini, Matteo Foglio, Tommaso Massari

# Contents

# 1 Introduction

## 1.1 Revision History

Table 1: revisione

| Version | Date | Author(s) | Summary |
|---|---|---|---|
| 1.1 | 14/01/2017 | Giovanni Agugini, Matteo Foglio e Tommaso Massari | Initial release |

## 1.2 Purpose and Scope

This document is the official Integration Test Plan Document for PowerEnjoy, an electric car-sharing service for sustainable mobility urban areas. The integration testing phase aimes at exercising interfaces and modules interaction in an incremental way such that the software is always under revision. Therefore, the purpose of this document is to plan an integration test of the whole software. In the following sections we're going to provide:

- The criteria that must be met before integration test may begin
- The strategy to adopt for the integration test
- An order or a sequence of components/functions that will be integrated, depending on the strategy decided in the point above
- The types of tests that will be used to verify the integration for each step of the sequence. Then we're going to provide a general description of the result expected from the test set
- The tools and the test equipments needed to accomplish the integration. We'll also provide an explanation of how and why we're going to use them
- Program stubs or special test data required for each integration step
- The effort spent in terms of number of hours each group member has spent

That has to be done in order to avoid inconsistencies, side effects, dynamic mismatches and so on during the development process.

## 1.3 List of Definitions, Acronyms and Abbreviations

### 1.3.1 Definitions

- Subsystem: a high-level functional unit of the system
- Strategy: the way used to test the integration

### 1.3.2 Acronyms

- DD: Design Document
- RASD: Requirement Analysis and Specification Document
- API: Application Programming Interface
- GPS: Global Positioning System
- UI: User Interface
- EJB: Enterprise Java Beans

### 1.3.3 Abbreviations

- WebApp as for Web Application

## 1.4 List of Reference Documents

- The previous documents of the project: Design Document and Requirements Analysis and Specification Document (DD.pdf and RASD.pdf)
- The project description document: Assignment AA 2016/2017.pdf
- Example of ITPD Document: Sample Integration Test Plan Document.pdf
- Slides from the course Software Engineering 2 - Professor Elisabetta Di Nitto and Professor Luca Mottola - Politecnico Di Milano
- GitHub documentation on the web - www.github.com
- StackOverflow forum: www.stackoverflow.com

# 2 Integration Strategy

## 2.1 Entry Criteria

The Integration Test requires the following conditions to be satisfied:

- Requirements Analysis and Specification Document must be entirely completed
- Design Document must be entirely completed
- Unit Tests must entirely completed for the Model part of the MVC. The Unit Tests of the other components can be completed during the Integration Test but they must reach a percentage of 90% before being integrated in the Integration Test.

It should be noted our system include third-app subsystem that have already been set by their software house and thus they are immediately available for the Integration Tests.

## 2.2 Elements to be Integrated

### 2.2.1 Higher Level Identication

The Integration Tests will be structured on two different level:

- Integration Tests among components that constitute the Application Layer
- Integration Test among higher level subsystem: Web App, On-board PC, Application Logic, Model, DB
- Integration Test among our system and the gateways of external outsourcers

Following the bottom-up approaces we will start from the first one and go on with the latter.

### 2.2.2 Components Dependencies Identification

Since the Application Layer include components that are strongly related one to each other, we are going to list the functions used by every components in order to identify dependencies between components. These dependencies are necessary to identify the invidual Integration Tests: every call from a function of a component A to a function of a component B must be tested with an Integration Test since has not been tested previously with a Unit Test.

**CLIENT CONTROLLER** ClientController.loginClient() calls:

- UserController.verifyCredentials()

ClientController.logoutClient() calls:

- ClientController.checkAutorization()

ClientController.userRequest() calls:

- ClientController.checkAutorization()
- UserController.registerClient()
- UserController.editClientData()

ClientController.rentalRequest() calls:

- ClientController.checkAutorization()
- RentalController.createNewRental()
- RentalController.endRental()
- RentalController.enableMoneySaving()

ClientController.areaRequest() calls:

- AreaController.isPositionInSafeArea()
- AreaController.getNearbyCars()
- AreaController.getGridStations()

**OPERATOR CONTROLLER** OperatorController.loginOperator() calls:

- UserController.verifyCredentials()

OperatorController.logoutOperator() calls:

- OperatorController.checkAutorization()

OperatorController.userRequest() calls:

- OperatorController.checkAutorization()
- UserController.editClientData()
- UserController.validateClient()
- UserController.blockClient()

OperatorController.rentalRequest() calls:

- OperatorController.checkAutorization()

- RentalController.createNewRental()
- RentalController.endRental()

OperatorController.carRequest() calls:

- CarController.changeState()
- CarController.lookForCarsToBeMoved()
- CarController.lookForOrdinaryMaintenance()

OperatorController.areaRequest() calls:

- AreaController.isPositionInSafeArea()
- AreaController.getNearbyCars()
- AreaController.getGridStations()

**RENTAL CONTROLLER** RentalController.createNewRental() calls:

- RentalController.createReservation()
- RentalController.createRide()
- CarController.changeState()

RentalController.endRental() calls:

- RentalController.compureFare()
- PaymentController.createPayment()
- CarController.changeState()
- CarController.enableDiscountAndFees()

RentalController.computeFare() calls:

- CarController.getCarInfo()

RentalController.enableMoneySaving() calls:

- CarControlller.getCarInfo()
- AreaController.getGridStations()

RentalController.enableDiscountAndFees()

- CarController.getCarInfo()

**PAYMENT CONTROLLER** PaymentController.createPayment() calls:

- NotificationController.sendRequest()
- NotificationController.receiveRequest()
- NotificationController.sendPaymentRequest()
- UserController.blockClient()

PaymentController.sendPaymentRequest() calls:

- NotificationController.sendRequest()

**CAR CONTROLLER** CarController.lookForCarToBeMoved() calls:

- CarController.getCarInfo()
- AreaController.computeBetterPosition()

CarController.lookForOrdinaryMaintenance() calls:

- CarController.getCarInfo()
- NotificationController.sendRequest()

## 2.3 Integration Testing Strategy

We will base out Integration Tests on the bottom-up approace. This choice will give us several advantages:

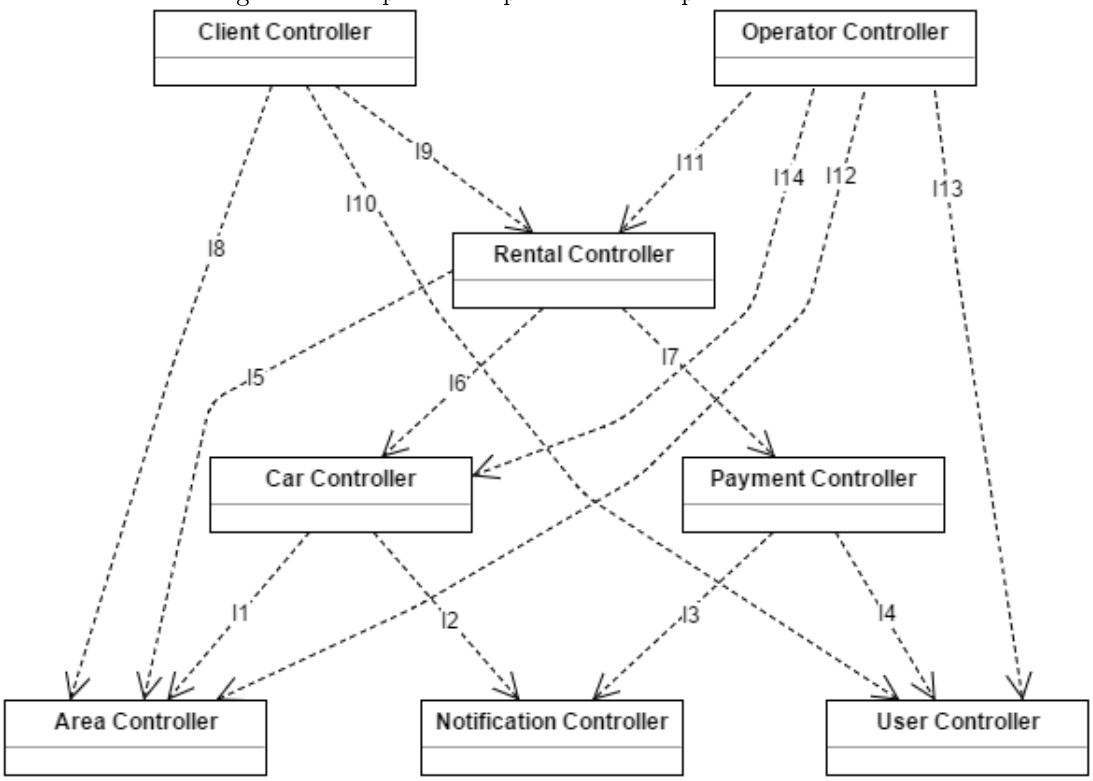
- Allows us to parallelize the development and Unit Tests of single components with the Integration Tests. In fact, for the first Integration Tests it is required that only the Unit Tests of the lowest components have been completed
- Allows us to find bugs rapidly since we can start Integration Test on compoenents right after the completion of their Unit Test and developments
- Allows us to use only drivers of components for the Integration Test, avoiding the need of stubs. Other approaches, like the "Riskiest Component First" or the "Thread" one, would have required the use of several drivers and stubs with a delay in the test deliveries due to the huge number of links between components in our project compared to the number of components.

## 2.4 Sequence of Component/Function Integration

### 2.4.1 Software Integration Sequence

Following the bottom-up approach, the integration test start from the low level components to high level ones. Thanks to this approach many integration tests can be parallelized as shown below. We assume that the external components (i.e. Gateway API Google Maps, Gateway Mail, Gateway Payment Agency and Gateway Maintenance Agency) of our system are already functioning in the right way. Here are the dependencies between components that we found.

Figure 1: Component Dependencies Graph




Thanks to the bottom-up strategy, the integration test can be easily parallelized in order to improve efficiency in the development and testing process. Here below a summary of the parallelization that can be done in tha test phase.

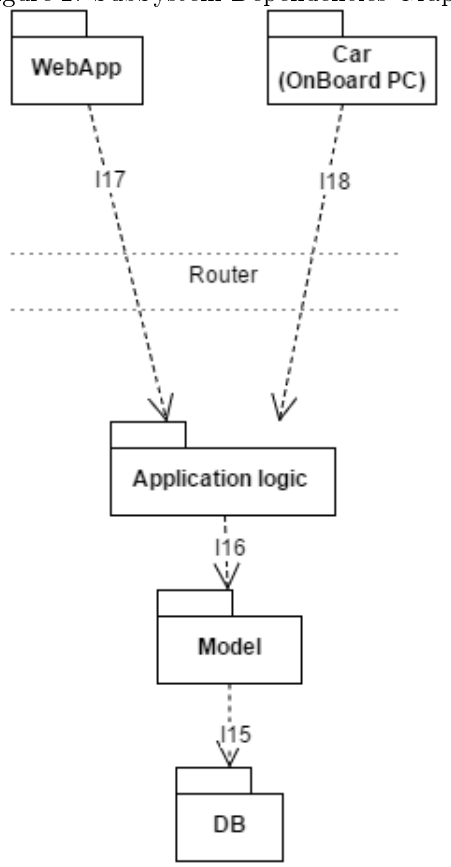
Table 2: Parallelization Table

| Step | Integration Tests |
|---|---|
| 1 | I1 I2 I3 I4 |
| 2 | I5 I6 I7 |
| 3 | I8 I9 I10 I11 I12 I13 I14 |

### 2.4.2 Subsystem Integration Sequence

After having tested the integration between components, we now arrive at a higher-level integration stage: the subsystem integration. Here below is the dependencies graph of the subsystems of our software.

Figure 2: SubSystem Dependencies Graph



# 3 Individual Steps and Test Description

## 3.1 Test Overview

Listed below are the integration tests planned for the software along with the components involved and accordingly to the chosen strategy.

Table 3: Test Overview

| Integration Testing | Components Involved |
|---|---|
| I1 | Car Controller –> Area Controller |
| I2 | Car Controller –> Notification Controller |
| I3 | Payment Controller –> Notification Controller |
| I4 | Payment Controller –> User Controller |
| I5 | Rental Controller –> Area Controller |
| I6 | Rental Controller –> Car Controller |
| I7 | Rental Controller –> Payment Controller |
| I8 | Client Controller –> Area Controller |
| I9 | Client Controller –> Rental Controller |
| I10 | Client Controller –> User Controller |
| I11 | Operator Controller –> Rental Controller |
| I12 | Operator Controller –> Area Controller |
| I13 | Operator Controller –> User Controller |
| I14 | Operator Controller –> Car Controller |

## 3.2 Component Test Description

### 3.2.1 I1 - Integration Test 1

Table 4: Integration Test 1

| Test Items | Car Controller –> Area Controller |
|---|---|
| Type of Tests | • Car Controller is able to ask a better position of a determined car (accordingly to the uniform distribution algorithm, see DD) to Area Controller |

Table 5: AreaController.computeBetterPosition(car: Car): Position

| Input | Effect |
|---|---|
| A valid Car car (i.e. consistent object) | Returns a Position position which is the best position of the car to be placed. If the car is already in a good position, accordingly to the uniform distribution, returns the same position. |
| An invalid Car car | Returns an InvalidCarException |

#### 3.2.2 I2 - Integration Test 2

Table 6: Integration Test 2

| Test Items | Car Controller –> Notification Controller |
|---|---|
| Type of Tests | • Car Controller is able to send a request of Ordinary Maintenance to Notification Controller with valid input data |

Table 7: NotificationController.sendRequest(request: Request): void

| Input | Effect |
|---|---|
| A valid Request request with the car info (position, maintenance) | Notification Controller calls the Maintenance Agency Gateway sending the appropriate request related to the specific car |
| An invalid Request request (i.e. invalid car info) | Returns an InvalidRequestException |

#### 3.2.3 I3 - Integration Test 3

Table 8: Integration Test 3

| Test Items | Payment Controller –> Notification Controller |
|---|---|
| Type of Tests | • Notification Controller is able to receive payment request from Payment Controller and to check if the request is consistent and coherent with the situation. Later on it has to forward the request to the external Payment Agency and to get back a transaction result. Finally Notification Controller has to communicate the result to the Payment Controller. |

Table 9: NotificationController.sendRequest(req: Request): void

| Input | Effect |
|---|---|
| A valid Request request of payment with the payment order computed by the Rental Controller | Notification Controller captures the payment order from Payment Controller and then calls the method to interact with the external Payment Agency |
| An invalid Request request (i.e. invalid payment order, inconsistent coordinates) | Returns an InvalidRequestException |

Table 10: NotificationController.receiveRequest(req: Request): boolean

| Input | Effect |
|---|---|
| A valid Request request that has been already sent to the Notification Controller to be forwarded to the external Payment Agency | Notification Controller takes the request and find the correspondence with the payment result from the Payment Agency. if the result is a payment accepted returns true; otherwise if the payment has been rejected returns false |
| An invalid request or a request not yet forwarded to the Payment Agency | Returns an InvalidRequestException |

Table 11: NotificationController.sendPaymentRequest(req: Request): void

| Input | Effect |
|---|---|
| A valid Request request corresponding to a payment order rejected by the payment agency | Notification Controller receives again a payment request to be forwarded to the external Payment Agency |
| An invalid Request request | Returns an InvalidRequestException |

#### 3.2.4 I4 - Integration Test 4

Table 12: Integration Test 4

| Test Items | Payment Controller –> User Controller |
|---|---|
| Type of Tests | • Payment Controller is able to block the client if the payment of the last ride has not been finalized |

Table 13: UserController.blockClient(payOrder: Order, client: Client): block

| Input | Effect |
|---|---|
| A valid payment order that has not been accepted by the external Payment Agency and a valid client who cannot fulfill the payment | Returns true and updates the client to be blocked |
| A client who does not correspond to a missing payment or a payment order which is not linked to the client | Returns CannotBlockException |

Table 16: Integration Test 6

| Test Item | Rental Controller –> Car Controller |
|---|---|
| Type of Tests | • Rental Controller can correctly change the car state<br>• Rental Controller can correclty retrieve car information linked to the ride (status, battery, discount) |

### 3.2.5 I5 - Integration Test 5

Table 14: Integration Test 5

| Test Items | Rental Controller –> Area Controller |
|---|---|
| Type of Tests | • Rental Controller is able to get the grid stations nearby a position within a specified radius<br>• Area Controller can provide a list of grid stations pertinent with the request |

Table 15: AreaController.getGridStations(pos: Position, radius: Integer): List<GridStation>

| Input | Effect |
|---|---|
| A valid Position position and a valid Integer radius (i.e. radius > = 0) | Returns the list of the grid stations within a tot km radius from the position given by the input |
| A negative radius | Returns NegativeRadiusException |
| An invalid position (e.g. GPS coordinates not coherent with the operative area) | Returns InvalidPositionException |
| Invalid position, invalid radius | Returns InvalidInputException |

#### 3.2.6 I6 - Integration Test 6

Table 17: CarController.changeState(state: State, car: Car): boolean

| Input | Effect |
|---|---|
| A valid state State which is pertinent with the previous actions, a valid car Car | Returns true if the car state has been correctly updated |
| An invalid state | Returns InvalidStateException |
| An invalid car | Returns InvalidCarException |
| Invalid state, invalid car | Returns InvalidInputException |

Table 18: CarController.getCarInfo(descriptionList: List<String>, car: Car): HashMap <String, Object>

| Input | Effect |
|---|---|
| A valid description list of the information required of the car Car and a valid car | Returns an HashMap of the info required with the description as key and the information in the form of an Object as value |
| Invalid description list and invalid car | Returns InvalidInputException |

#### 3.2.7 I7 - Integration Test 7

Table 19: Integration Test 7

| Test Items | Rental Controller –> Payment Controller |
|---|---|
| Type of Tests | • Rental Controler is able to send a payment order after having collected all the info about the car of the ride |

Table 20: PaymentController.createPayment(payorder Order, client Client): void

| Input | Effect |
|---|---|
| A valid payorder Order and a valid client | Triggers the creation of the payment request of the ride by merging the payment order and the payment data of the client |
| Invalid payorder (e.g. total fare $< 0$) | Returns InvalidOrderException |
| Invalid client | Returns InvalidClientException |
| Invalid payorder, invalid client | ReturnsInvalidInputException |

#### 3.2.8 I8 - Integration Test 8

Table 21: Integration Test 8

| Test Items | Client Controller –> Area Controller |
|---|---|
| Type of Tests | • CC uses Area Controller to get cars and grid stations near a given position and radius<br>• CC uses AC to know is a given position is inside the safe area |

Table 22: AreaController.isPositionInSafeArea(position: Position): boolean

| Input | Effect |
|---|---|
| Valid position | Returns true if the position is in a safe area, false otherwise |
| Invalid position | Returns InvalidPositionException |

Table 23: AreaController.getNearbyCars(position: Position, radius: Integer): List<Car>

| Input | Effect |
|---|---|
| Valid position, valid radius (i.e. radius >= 0) | Returns a list containing the cars |
| Invalid position (e.g. GPS coordinates are not coherent with operative area), valid radius | Returns InvalidPositionException |
| Valid position, invalid radius | Returns NegativeRadiusException |
| Invalid position, invalid radius | Return InvalidInputException |

Table 24: AreaController.getGridStations(position: Position, radius: Integer): List<GridStation>

| Input | Effect |
|---|---|
| Valid position, valid radius (i.e. radius >= 0) | Returns a list containing the cars of which distance from the Position position is not more than the one indicated in the parameter radius |
| hInvalid position (e.g. GPS coordinates are not coherent with operative area) , valid radius | Returns InvalidPositionException |
| Valid position, invalid radius | Returns NegativeRadiusException |
| Invalid position, invalid radius | Return InvalidInputException |

:

#### 3.2.9 I9 - Integration Test 9

Table 25: Integration Test 9

| Test Items | Client Controller –> Rental Controller |
|---|---|
| Type of Tests | • CC is able to create terminate a rental by calling function on RC<br>• CC is able to enable Money Saving Option on currently active rental |

Table 26: RentalController.createNewRental(user: User) : void

| Input | Effect |
|---|---|
| A valid and active client | A new rental is created in the system meaning that the client is now in the "Reservation Phase": the car is reserved for the client who will be able to use it as soon as he reaches the vehicles |
| A blocked client | Returns BlockedClientException |
| An invalid client object | Returns InvalidClientException |

Table 27: RentalController.endRental(user: User): void

| Input | Effect |
|---|---|
| A client who is renting a car | The rental of the user User is termined, all information are written in the database and payment has been managed |
| A client who hasn't started a rental yet | Returns NonExistentRentalException |
| An invalid client object | Returns InvalidClientException |

Table 28: RentalController.enableMoneySaving(user: User): void

| Input | Effect |
|---|---|
| A client who is renting a car | The Money Saving Option has been enabled for the current rental of the User user |
| A client who hasn't started a rental yet | Returns NonExistentRentalException |
| An invalid client object | Returns InvalidClientException |

#### 3.2.10 I10 - Integration Test 10

Table 29: Integration Test 10

| Test Items | Client Controller –> User Controller |
|---|---|
| Type of Tests | • CC is able to retrieve information from the UC in order to check if credentials used to login are valid<br>• CC is able to register a new client by the use of UC<br>• CC is able to modify client data by the use of UC |

Table 30: UserController.verifyCredentials(username: String, password: String): boolean

| Input | Effect |
|---|---|
| Valid credentials | Returns true |
| Invalid credentials | Returns false |

Table 31: UserController.registerClient(data: HashMap<Name,Value>): void

| Input | Effect |
|---|---|
| Valid client data | A new client is created in the database but the account needs to be approved in order to allow the client to rent a car |
| Invalid client data | Returns InvalidInputException |

Table 32: UserController.editClientData(data: HashMap<Name,Value>): void

| Input | Effect |
|---|---|
| Valid client data | Client data are updated in the database with the new data contained in the Client client object |
| Invalid client data | Returns InvalidInputException |

#### 3.2.11 I11 - Integration Test 11

Table 33: Integration Test 11

| Test Items | Operator Controller –> Rental Controller |
|---|---|
| Type of Tests | • OC is able to create terminate a rental by calling function on RC |

Table 34: RentalController.createNewRental(user: User): void

| Input | Effect |
|---|---|
| A valid operator | A new rental is created in the system meaning that the operator is now in the "Reservation Phase": the car is reserved for the operator who will be able to use it as soon as he reaches the vehicles |
| An invalid operator object | Returns InvalidOperatorException |

Table 35: RentalController.endRental(user: User): void

| Input | Effect |
|---|---|
| An operator who is renting a car | The rental of the user User is termined, all information are written in the database |
| An operator who hasn't started a rental yet | Returns NonExistentRentalException |
| An invalid operator object | Returns InvalidOperatorException |

#### 3.2.12 I12 - Integration Test 12

Table 36: Integration Test 12

| Test Items | Operator Controller –> Area Controller |
|---|---|
| Type of Tests | • OC uses Area Controller to get cars and grid stations near a given position and radius<br>• OC uses AC to know is a given position is inside the safe area |

Table 37: AreaController.isPositionInSafeArea(position: Position): boolean

| Input | Effect |
|---|---|
| Valid position | Returns true if the position is in a safe area, false otherwise |
| Invalid position | Returns InvalidPositionException |

Table 38: AreaController.getNearbyCars(position: Position, radius: Integer): List<Car>

| Input | Effect |
|---|---|
| Valid position, valid radius (i.e. radius >= 0) | Returns a list containing the cars |
| Invalid position (e.g. GPS coordinates are not coherent with operative area), valid radius | Returns InvalidPositionException |
| Valid position, invalid radius | Returns NegativeRadiusException |
| Invalid position, invalid radius | Return InvalidInputException |

Table 39: AreaController.getGridStations(position: Position, radius: Integer): List<GridStation>

| Input | Effect |
|---|---|
| Valid position, valid radius (i.e. radius >= 0) | Returns a list containing the cars of which distance from the Position position is not more than the one indicated in the parameter radius |
| Invalid position (e.g. GPS coordinates are not coherent with operative area) , valid radius | Returns InvalidPositionException |
| Valid position, invalid radius | Returns NegativeRadiusException |
| Invalid position, invalid radius | Return InvalidInputException |

#### 3.2.13 I13 - Integration Test 13

Table 40: Integration Test 13

| Test Items | Operator Controller –> User Controller |
|---|---|
| Type of Tests | • OC is able to retrieve information from the UC in order to check if credentials used to login are valid<br>• OC is able to validate a client by the use of UC<br>• OC is able to modify client data by the use of UC<br>• OC is able to block a client by the use of UC |

Table 41: UserController.verifyCredentials(username: String, password: String): boolean

| Input | Effect |
|---|---|
| Valid credentials | Returns true |
| Invalid credentials | Returns false |

Table 42: UserController.editClientData(data: HashMap<Name,Value>): void

| Input | Effect |
|---|---|
| Valid client | Client data are updated in the database with the new data contained in the Client client object |
| Invalid client | Returns InvalidInputException |

Table 43: UserController.validateClient(client: Client): void

| Input | Effect |
|---|---|
| Valid client | The client account is validated meaning that the client can now rent cars and access to the services of the Power Enjoy company |
| Non existent client | Returns InvalidClientException |

Table 44: UserController.blockClient(client: Client): void

| Input | Effect |
|---|---|
| Valid client | The client account is now blocked meaning that the client cannot rent cars anymore until he will be validates again |
| Non existent client | Returns InvalidClientException |

#### 3.2.14 I14 - Integration Test 14

Table 45: Integration Test 14

| Test Items | Operator Controller –> Car Controller |
|---|---|
| Type of Tests | • OC uses CC to get the list of the cars to be moved<br>• OC uses CC to get the list of cars that needs an ordinary maintenance<br>• OC uses CC to modify the status of a car |

Table 46: CarController.changeState(state: State, car: Car): boolean

| Input | Effect |
|---|---|
| Valid state, valid car | The car state is updated with the one given as parameter |
| Valid state, invalid car | Returns InvalidCarException |
| Invalid state, invalid car | Returns InvalidStateException |
| Invalid state, invalid car | Returns InvaliInputException |

Table 47: CarController.lookForCarsToBeMoved(position: Position, radius: Integer): HashMap<Car, Position>

| Input | Effect |
|---|---|
| Valid position, valid radius (i.e. radius >= 0) | The method analize the cars of which distance from the Position position is not more than the one indicated in the parameter radius. The list of cars returned contains only the ones that should be moved with their corrispondent suggested position |
| Invalid position (e.g. GPS coordinates are not coherent with operative area) , valid radius | Returns InvalidPositionException |
| Valid position, invalid radius | Returns NegativeRadiusException |
| Invalid position, invalid radius | Return InvalidInputException |

Table 48: CarController.lookForOrdinaryMaintenance(position: Position, radius: Integer): List<Car>

| Input | Effect |
|---|---|
| Valid position, valid radius (i.e. radius >= 0) | The method analize the cars of which distance from the Position position is not more than the one indicated in the parameter radius. The list of cars returned contains only the ones of which the state is set to "Ordinary Maintenance" |
| Invalid position (e.g. GPS coordinates are not coherent with operative area) , valid radius | Returns InvalidPositionException |
| Valid position, invalid radius | Returns NegativeRadiusException |
| Invalid position, invalid radius | Return InvalidInputException |

## 3.3 Subsytem Integration Test

### 3.3.1 I15 - Integration Test 15

Table 49: Integration test I15

| Test items | Model –> DB |
|---|---|
| Type of Tests | • DBMS accept Model requests<br>• DBMS correctly and consistently answers to the Model's requests<br>• Model is consistent and dynamic, an update to the DBMS implies the same update to the model and viceversa<br>• Correct management of data errors<br>• Congruence of the data formats |

**Test Cases:**

- Data creation: assure that the data that is about to be created is not already present in the DBMS, then create it and check its existence. The first check should return empty and the second should return the data created before
- Data search: look for a data that is known to be in the DBMS
- Data modification: modify a data in DBMS and then check that the modification has occurred.
- Data elimination: delete a data that is known to be in the DBMS and then search for it. It should not be found.

The tests should be done to all components of the model, this includes:

- User, generalization for Client and Operator
- Rental
- Reservation
- Ride
- Car
- Payment

- Area
- Position
- Grid Station

#### 3.3.2 I16 - Integration Test 16

Table 50: Integration test I16

| Test items | Application logic –> Model |
|---|---|
| Type of Tests | • Controllers can access information provided by the Model<br>• Controllers can modify data (creation, modification, deletion) on the Model (and consistently with the DBMS)<br>• Correct management of data incompatibilities |

To check the correctness of the interaction between the Applcation logic and the Model all methods that access directly to the Model's data should be tested:

- User controller
  - userController.verifyCredentials()
  - userController.registerClient()
  - userController.editClient()
  - userController.validateClient()
- Rental controller
  - rentalController.createNewRental()
  - rentalController.createReservation()
  - rentalController.createRide()
  - rentalController.endRental()
- Car controller
  - carController.changeState()
  - carController.lookForCarToBeMoved()
  - carController.lookForOrdinaryMaintenance()
  - carController.getCarInfo()

- Payment controller
  - paymentController.createPayment()
- Area controller
  - areaController.isPositionInSafeArea()
  - areaController.getGridStations()
  - areaController.getNearbyCars()

#### 3.3.3 I17 - Integration Test 17

Table 51: Integration test I17

| Test items | WebApp –> Application Logic |
|---|---|
| Type of Tests | • WebApp and Application logic communicates with the same formats and protocols<br>• WebApp can correcly display the answers received<br>• Requests are correctly understood and managed by the Application logic and they are delivered to the corresponding controller (if more than one, in the right order)<br>• Management of data error and communication errors |

In order to make sure that the WebApp displays the correct page and the correct (and updated) data, both for the operator side and the client side, all the methods of the User Interface should be tested by the manual execution of some operations on the WebApp. The following tables describes the operations to be done in order to complete this section of the Integration Test. It should be mentioned that the effects explained below are fully described in the RASD by the use of mockups and requirements.

Table 52: Client Registration

| Test Operations | Effect |
|---|---|
| • Register a new client | • A client is created in the database waiting to be approved<br>• The app shows that the client is now waiting for the approval |

Table 53: Edit Client Data

| Test Operations | Effect |
|---|---|
| • Edit client data | • The app shows the new data |

Table 54: Rent a car with enough money

| Test Operations | Effect |
|---|---|
| Complete the rental of a car:<br>• Reserve a car<br>• Access the car<br>• Drive the car<br>• End rental<br>• Try to reserve a new car | • The rental has been correctly managed by the Application Layer<br>• The app behaves showing the correct page as explained in the RASD<br>• The app allows the client to make new operations |

Table 55: Rent a car without enough money

| Test Operations | Effect |
|---|---|
| Complete the rental of a car:<br>• Reserve a car<br>• Access the car<br>• Drive the car<br>• End rental<br>• Try to reserve a car | • The rental has been correctly managed by the Application Layer<br>• The client receives a request of the payment<br>• The client is blocked<br>• The app doesn't allow the client to make new operations |

Table 56: Cancel a reservation

| Test Operations | Effect |
|---|---|
| Cancel a reservation;:<br>• Reserve a car<br>• Cancel reservation within 1 hour | • The reservation cancellation has been correctly managed by the Application Layer<br>• The car is now available<br>• The app allows the client to make new operations |

Table 57: Reservation expiration

| Test Operations | Effect |
|---|---|
| Let the reservation expires;:<br>• Reserve a car<br>• Wait 1 hour | • The reservation expiration has been correctly managed by the Application Layer<br>• The car is now available<br>• The client pays a fee<br>• The app allows the client to make new operations |

**Client Side**

Table 58: Approve a client

| Test Operations | Effect |
|---|---|
| Approve a client:<br>• A client register to the service<br>• An operator approve the client | • The app doesn't allow the client to rent a car anymore |

Table 59: Block a client

| Test Operations | Effect |
|---|---|
| Block a client:<br>• An operator block a client<br>• A client try to reserve a car | • The app doesn't allow the client to rent a car anymore |

Table 60: Change the state of a car

| Test Operations | Effect |
|---|---|
| Change the state of a car:<br>• An operator change the state of the car | • The app shows the new state of the car |

**Operator Side**

### 3.3.4 I18 - Integration Test 18

Table 61: Integration test I18

| Test items | Car (On board PC) –> Application logic |
|---|---|
| Type of Tests | • The on board PC correctly sends the car info to the Application Logic<br>• The App on the On-Board PC and Application logic communicates with the same formats and protocols<br>• The On-Board PC can read data provided by the Application Logic and eventually updates its status |

Table 62: On_Board computer

| Test Operations | Effect |
|---|---|
| During a ride make use of the following function:<br>• Take enough passengers in order to enable a discount<br>• Show grid stations<br>• Use the system to navigate to a given address<br>• Enable money saving option | • The On-Board PC must execute all the requested actions |

## 3.4 Integration test with External Gateways

**Gateway Google Maps API**

To test the communication with the Gateway Google Maps API the best method is the manual test. It's possible to visualize the correct functioning of the Map on the GUI, both from a device where the WebApp is installed and an On board PC.

Table 63: Integration test I19

| Test items | Car (On board PC) –> Gateway Google Maps API<br>WebApp –> Gateway Google Maps API<br>Notification controller –> Gateway Google Maps API |
|---|---|
| Type of Tests | • The devices (WebApp and On board PC) and the Area controller correclty send requests to the Gateway<br>• A consistent response is received and visualized on both the WebApp and the On board PC, and correct informations are received by the Area controller. |

Table 64: Integration test I20

| Test items | Notification Controller –> Gateway Mail |
|---|---|
| Type of Tests | • Emails are correctly sent and received |

### Gateway Mail

To test the communication with the Gateway Mail the best method is the manual test. It's possible to create a sample e-mail, for example a payment request email, and send it through the Gateway Mail to an Operator email address. The e-mail should be correctly delivered.

### Gateway Payment Agency

To test the communication with the Gateway Payment Agency the best method is the manual test. It's possible to create a sample payment, for example a small amount of money payment request, and send it through the Gateway Payment

Table 65: Integration test I21

| Test items | Notification Controller –> Gateway Payment Agency |
|---|---|
| Type of Tests | • Payment are correctly sent and processed by the Payment Agency<br>• Consistent responses are correctly received |

Table 66: Integration test I22

| Test items | Notification Controller –> Gateway Maintenance Agency |
|---|---|
| Type of Tests | • Maintenance requests are correctly sent and received by the Payment Agency<br>• Consistent responses are correctly received |

Agency to the Payment Agency. The payment should be correctly processed and the confirmation received.

**Gateway Maintenance Agency**

To test the communication with the Gateway Maintenance Agency the best method is the manual test. It's possible to create a sample maintenance request, for example associeted to a Car used for testing purposes, and send it through the Gateway Maintenance Agency to the Maintenance Agency. The request should be correctly processed and the confirmation received.

# 4 Tools and Test Equipment Required

In order to accomplish the test process we need several automated tools and test equipment that are suited to support our work. In the following sections are listed:

- Tools for **unit** testing, **integration** testing and **performance** testing
- Tools for **code quality evaluation**
- Equipment tools for both **server** and **client side**

## 4.1 Tools

Here are different tools depending on the specific functionality and execution environment.

### 4.1.1 Unit Testing

For this type of test we are going to use **JUnit**, a unit testing framework which is a central element of the testing practice. In fact every time a small portion of software is done we're going to test it by using **JUnit** with a white-box approach.

Figure 3: JUnit


Another unit testing tool that we're going to use is **Mockito**. Thanks to this useful mocking tool we can abstract dependencies and have predictable results and check that the interaction between the testee and the mock is correct.

Figure 4: Mockito

**QUnit** is a powerful, easy-to-use, JavaScript unit testing framework. It's used by the jQuery project to test its code and plugins but is capable of testing any generic JavaScript code (and even capable of testing JavaScript code on the server-side). QUnit is especially useful for regression testing: whenever a bug is reported, write a test that asserts the existence of that particular bug. Then fix it and commit both. Every time you work on the code again, run the tests. If the bug comes up again - a regression - you'll spot it immediately and know how to fix it, because you know what code you just changed.

Figure 5: QUnit

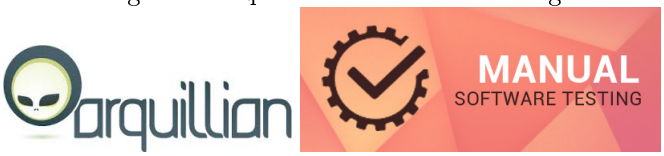

### 4.1.2 Integration testing

We're going to use **Mockito** for Java components on the main server. **Arquillian** is a platform that simplifies integration testing for Java middleware. It deals with all the plumbing of container management, deployment, and framework initialization so the developer can focus on the task of writing real tests. **Arquillian** minimizes the burden on the developer by covering aspects surrounding test execution; some of these aspects are as follows:

- Managing the life cycle of the container (start/stop)
- Bundling the test class with the dependent classes and resources into a deployable archive
- Enhancing the test class (for example, resolving @Inject, @EJB, and @Resource injections)
- Deploying the archive to test applications (deploy/undeploy)

Finally we use **manual testing** to test if the WebApp runs correctly on a client device and the on-board computer is correctly functioning.

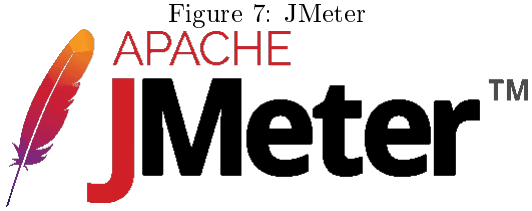
Figure 6: arquillian and manual testing




### 4.1.3 Performance testing

To evaluate the performance of the entire WebApp we decided to use JMeter, a GUI desktop application designed to load test functional behaviour for analyzing and measuring the performance of a variety of services, with a focus on web applications. JMeter supports variable parameterization, assertions (response validation), configuration variables and a variety of reports.

Figure 7: JMeter




## 4.2 Equipment required

Both client side and server side equipment is required

### 4.2.1 Server Side

On the server side we suggest to buy a subscription to the **EC2 service** (Amazon Elastic Compute Cloud) which let us create and execute a Linux virtual machine that is similar to the final server. The operating system is off course the same as the real one, and in our case it is **Ubuntu v 16.10** because we think that a Unix-based system would be better in terms of cost, maintenance and robustness.

Figure 8: Amazon EC2 and Ubuntu



Besides the need of a machine where to execute our software, we need at **least three cars** in order to test the server components related to the functionalities of the car.

### 4.2.2 Client Side

Our WebApp must to be run on smartphones, tablets and PCs. In order to maximize the coverage of devices available on the market we decide to test our software and all the principal technology brands. As it concerns PCs, we're going to test several web browsers.
Here are the selected devices to be tested with our software:

- Samsung Galaxy S6:
  - OS: Android OS, v 5.0.2 (Lollipop)
  - Display: 5.1"
  - Resolution: 1440 x 2560 pixels
  - Chipset: Exynos 7420 Octa
  - Memory: 32/64/128 GB, 3GB RAM
  - GPU: Mali-T760MP8
- Huawei P8 Lite:
  - OS: Android OS, v5.0.2 (Lollipop), upgradable to v6.0 (Marshmallow)
  - Display: 5"
  - Resolution: 720 x 1280 pixels
  - Chipset: HiSilicon Kirin 620
  - Memory: 16 GB, 2 GB RAM
  - GPU: Mali-450MP4

- Iphone 6:
  - OS: iOS 9.2.0
  - Display: 4.7”
  - Resolution: 750 x 1334 pixels
  - Chipset: Apple A8 (Dual-core 1.4 GHz Typhoon, ARM v8-based)
  - Memory: 16GB, 1GB RAM
- Samsung Galaxy Tab S2 9.7:
  - OS: Android OS, v5.0.2 (Lollipop), upgradable to v6.0.1 (Marshmallow) - T810, T815 Android OS, v6.0.1 (Marshmallow) - T813N, T819N
  - Display: 9.7”
  - Resolution: 1536 x 2048 pixels
  - Chipset: Exynos 5433 - T810, T815 Qualcomm MSM8976 Snapdragon 652 - T813N, T819N
  - Memory: 32/64 GB, 3 GB RAM
  - GPU: Mali-T760 MP6 - T810, T815 Adreno 510 - T813N, T819N
- Apple Ipad Air 2:
  - OS: iOS 8.1, upgradable to iOS 10.2
  - Display: 9.7”
  - Resolution: 1536 x 2048 pixels
  - Chipset: Apple A8X
  - Memory: 16/32/64/128 GB, 2 GB RAM
  - GPU: PowerVR GXA6850 (octa-core graphics)

As it concerns web browsers, we selected the following ones:

- Google Chrome, v55.0
- Mozilla Firefox, v45.4.0 ESR
- Internet Explorer for Windows, v11.0
- Microsoft Edge, v38.14393
- Opera, v41.0.

# 5 Program Stubs and Test Data Required

In this section it is shown for each step any program stub driver that are needed for the ingretion test. It is also shown if any extra test data are needed, and a short description is provided.

## 5.1 Component Integration

Table 67:

| Step | Driver needed | Extra test data |
|---|---|---|
| 1 | Car controller<br>Payment controller | Samples of car and samples of payment info |
| 2 | Rental controller | Samples of rentals |
| 3 | Client controller<br>Operator controller | Samples of user and operator |

## 5.2 Subsystem Integration

Table 68:

| Step | Driver needed | Extra test data |
|---|---|---|
| 1 | All the Model Components | Samples of data of all the Model in the DBMS |
| 2 | Car controller<br>Payment controller<br>User controller<br>Area controller<br>Rental controller<br>Notification controller | Samples of data of all the Model in the DBMS |
| 3 | WebApp and On board PC | Samples of data of all the Model in the DBMS<br>A Car for testing the Car Info to be sent from the On board PC<br>Samples of User Info to be sent from the WebApp |

# 6 Effort Spent

- Giovanni Agugini
  - 03/01/2017 : 3h
  - 05/01/2017 : 5h
  - 07/01/2017 : 5h
  - 15/01/2017 : 3h
- Matteo Foglio
  - 03/01/2017 : 3h
  - 05/01/2017 : 5h
  - 07/01/2017 : 5h
  - 15/01/2017 : 3h
- Tommaso Massari
  - 03/01/2017 : 3h
  - 05/01/2017 : 5h
  - 07/01/2017 : 5h
  - 15/01/2017 : 3h