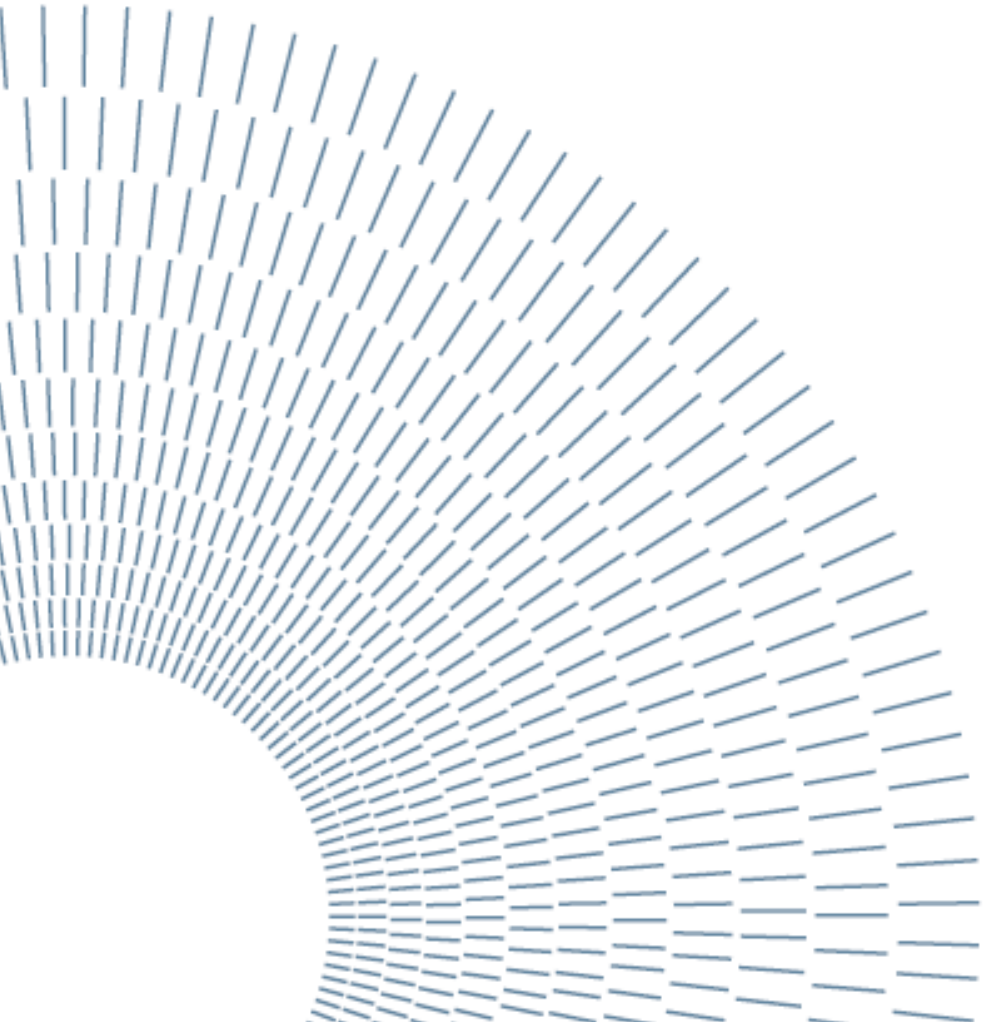
POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Software Engineering 2 2016/2017
Project Power EnJoy

# Integration Test Plan Document

V 1.0




Redacted by:

Leonardo Chiappalupi
CP 10453536

Ivan Bugli
CP 10453746

# Table of contents

# 1. Introduction

## 1.1 Revision history

- **V1.0** / 2017.01.15 / Leonardo Chiappalupi, Ivan Bugli / Initial release

## 1.2 Purpose

This document represents the Integration Test Plan Document for Power EnJoy, and it is aimed to show every step of the designed integration testing from selected strategies to individual tests and analysis.

The purpose of this phase – in the overall platform design – is to guarantee that the software behaves in a coherent and consistent way, component after component, until the final system in its integrity can be trusted not to expose unexpected situations or errors.

All the testing activity will also ensure that the platform will be reliable and consistent with the requirement analysis the various components are supposed to cover.

## 1.3 Scope

The scope of this document consists in all the components, sub-components, systems and sub-systems to be tested, that will be examined with the priority and in the order that will be fully detailed when the integration strategy will be presented.

The document will cover details about the approach that will be taken, the criteria for the individual tests, the conditions to be satisfied for the integration testing to be relevant, the sequence for the integration of the various parts, as well as all the data about the tools to be used, the input and output data of every test, and an extensive coverage on additional "environmental" and performance tests to be executed.

Every step of the integration testing design will be planned keeping in mind all the details already outlined in the previous documents.

## 1.4 Definitions, acronyms, abbreviations

### 1.4.1 Definitions

- **Sub-system:** a functional, high-level part of the whole system, grouping different components together
- **Component:** a lower-level unit of the system, dedicated to the management of a single (or a restricted selection of) feature(s).
- **Sub-component:** each of the individual single units possibly composed to create a component in its integrity.

Other definitions and abbreviations carry on from the previous documents. See sections related to glossary/definitions and their subsections for further details.

### 1.4.2 Acronyms and abbreviations

- **RASD:** Requirement Analysis and Specification Document
- **DD:** Design Document
- **DB:** Database
- **DBMS:** Database Management System
- **REST**: REpresentational State Transfer
- **API**: Application Programming Interface
- **OS**: Operative System
- **JEE**: Java Enterprise Edition
- **AWS**: Amazon Web Services
- **DPI**: Dots Per Inch
- **GSM:** Global System for Mobile communications

## 1.5 Reference documents

- **Assignments document:** Assignments AA 2016-2017.pdf
- **Requirement Analysis and Specification Document:** Requirement Analysis and Specification Document V1.1.pdf
- **Design Document:** Design Document V1.2.pdf
- **(*This*) Integration Test Plan Document:** Integration Test Plan Document V1.0.pdf

# 2. Integration strategy

## 2.1 Entry criteria

Before the integration testing of specific elements may begin, there are several conditions that are strictly necessary for this very testing in order to be valid and meaningful.

First of all, the two previous documents – the RASD and the DD – must be obviously be completely written, reviewed and approved. This is crucial in order to avoid specifications to change during the current phase, and to ensure that the integration testing covers all the possible aspects of the system to be.

Second, in order to make the integration testing possible, it is unavoidable to specify some requirements on the completeness of the development, both of single components as well as of some specific features of the general system. Since the integration testing will mainly focus on interactions between components at a feature level, we will not simply require a generic percentage of completion for those components; instead, we will try to list the core features that we need to be completely **written, reviewed, unit tested and approved** (in percentage respect to their entirety) before the integration testing can possibly start.

- **Mobile applications / Network communication – 90%:** independently from the mobile platform considered, integration testing will not focus on apps network access and management, thus it must be already working before this phase can start.
- **Mobile applications / Shared core logic – 90%:** mobile applications may still do not have their UI or final design completed, but the core logic must be almost completely functional in order to test its integration with the centralized system.
- **Web applications / RESTful API manager – 100%:** the web applications may be still in the work for the presentation layer, but they must be already able to manage requests from and to the API server.
- **Car applet / Network communication – 100%:** the on-board car software must already provide a fully functional network communication management, including errors handling etc.

These percentages and features are to be intended independently from the actual percentage of completeness of a component to be considered for integration, percentage that should be over **90%**. Also, for all components, it must be possible to at least define complete stubs: this means that method prototypes and skeletons must be available from the start of the integration testing.

Regarding the database setup, it is important that the lower level implementation – i.e. tables, schemas, triggers, etc. – have been implemented and tested, even if

only on a specific testing setup. For the integration testing to be meaningful, we should be able to check if the database populates and is edited as expected, thus it must be completely functional in this sense.

Finally, it is important that the external services that we are going to use in our systems have been correctly setup and licensed: this includes a valid PayPal account, a valid working connection with an external chosen bank (including account setup), API authorization tokens for the services (e.g. for Google Maps).

## 2.2 Elements to be integrated

As previously stated in the *Design Document*, the Power EnJoy platform is composed of a variety of different components that together allow the entire service to run. We already introduced, from both a high level and a lower level point of view, the distinction between system internal components – i.e. components that run server-side, dedicated to answering requests and executing commands – and external components, that may both act as clients (e.g. mobile applications) as well as only be commercial third party systems that we use as they are (e.g. the DBMS, or the PaymentHandler).

Hence, we will proceed as follows: we will group the various internal components into *sub-systems*, and we will progressively integrate each component of a sub-system until every single sub-system is completed; then, we will integrate the various sub-systems together and at the end, we will integrate the entire Power EnJoy system with all the external components, to check the consistency and completeness of tested features and integrations.

First, it is necessary to introduce the sub-systems division, that is:

- **User services system:** ClientReqDisp, UsersManagement, ReservationManager, RideController, CarDataService, PaymentProcessor, AssistanceProvider;
- **Management system:** SystemManReqDisp, FleetParksOrganizer, AssistanceProvider;
- **Operators system:** OperatorReqDisp, AssistanceProvider, CarMaintenanceHandler.

Please note that several components of those rely on one of the two third party commercial components that we are using as they are, i.e. on the **DBMS** component: this means that the integration with the **DBMS** component, even if external, cannot be tested at the end but needs to be examined at the very beginning. The same is valid for the couple **PaymentProcessor** / **PaymentHandler**: so, commercial third party components integration won't be examined at the end like we will actually do for other external components, but during each sub-system integration sequence.

Once all the sub-systems will be tested and integrated, we will be able to complete the platform by introducing external higher-level components, that are:

- **User mobile/web application**
- **System manager web app**
- **Operators mobile app**
- **Car applet**

Sub-systems shared components and external ones will drive the full system integration sequence and testing.

## 2.3 Integration Testing Strategy

To guide the integration sequence, we are going to mainly follow the bottom-up approach.
This choice is particularly useful in our case for a number of reasons: first, Power EnJoy has a lot of components that are strictly connected one to another and that form a network of dependencies and usages that makes integration testing risky and tricky. By sorting our components from the bottom to the top ones, we will be reasonably sure that we are integrating and testing them in a 'safe' way, because all 'lower' components have been already integrated when introducing a new one in. This strategy is also the one chosen for the development, thus this approach is the most suitable for making integration testing and development proceed almost parallel. Additionally, in the design of the system we used a strong layering of features, separating clients from core components using request dispatchers and so on. The bottom-up approach allows us to focus on 'core' components first, and then progressively integrate the other ones upon them.

Since not every component nor every sub-system is strictly related to other ones though, when explicit dependencies won't appear to be present, we will use the critical first approach, always giving priority to components that are more important for the main features of the platform. Combining these two approaches will allow to test and integrate riskiest components first, and avoid to have 'domino' effects when severe errors or issues are found.

⚠ *The DBMS and PaymentHandler integration with our components, even if analyzed during the integration sequence, won't be part of a dedicated series of testing: given that database access operations are in-depth integrated into the various components' methods, and so is the PaymentHandler for payment operations, we find it would be highly unpractical and pointless to test the integration at this level of detail. Additionally, testing upper components integration in our bottom-up approach, we will be already able to also test this subset of connections, in a more convenient and integrated way.*

## 2.4 Sequence of Component/Function integration

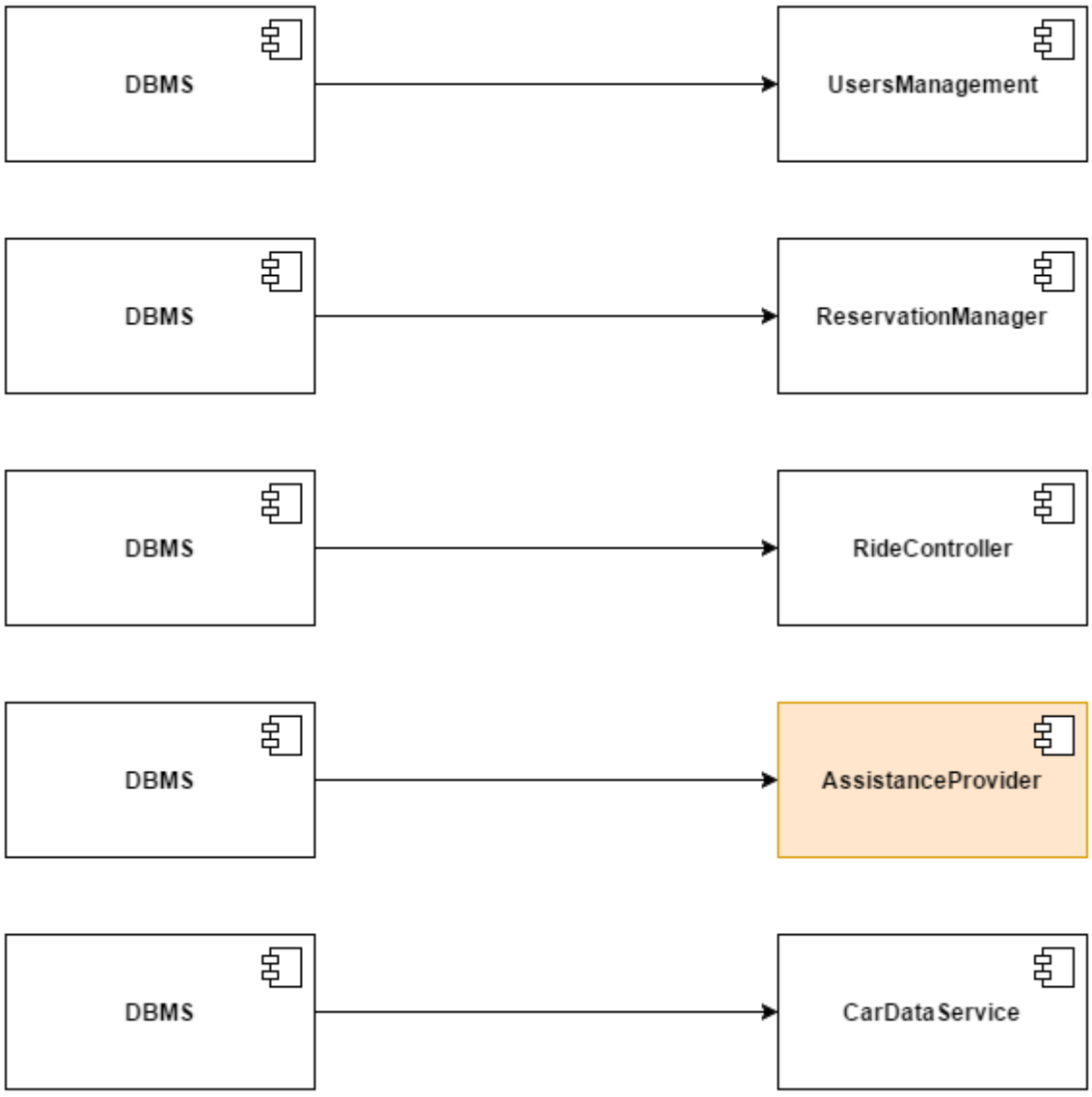
The following sections show the proposed components integration sequences for every sub-system, and then the sub-systems integration towards the final structure of the platform.

In diagrams, arrows represent dependencies: an arrow going from A to B means that B needs A to work so the integration sequence must be A, then B.

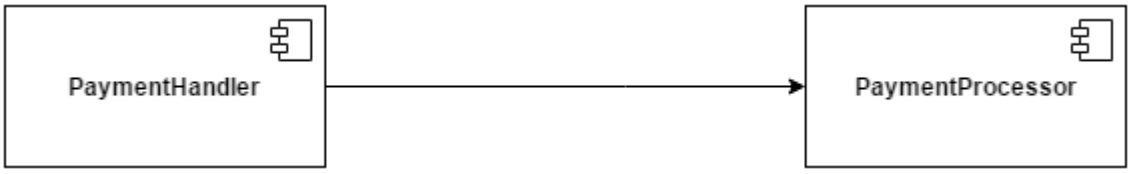
### 2.4.1 Software integration sequence

#### USER SERVICES SYSTEM

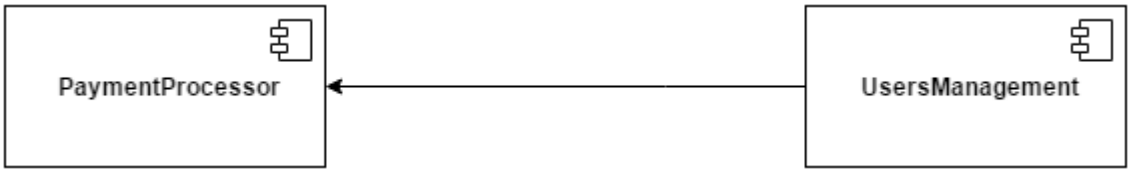
The first integration related to the *User services system* takes into account the connection between the different components of this subsystem and the DBMS. The DBMS is a commercial third-party system, so it does not need to be tested alone.



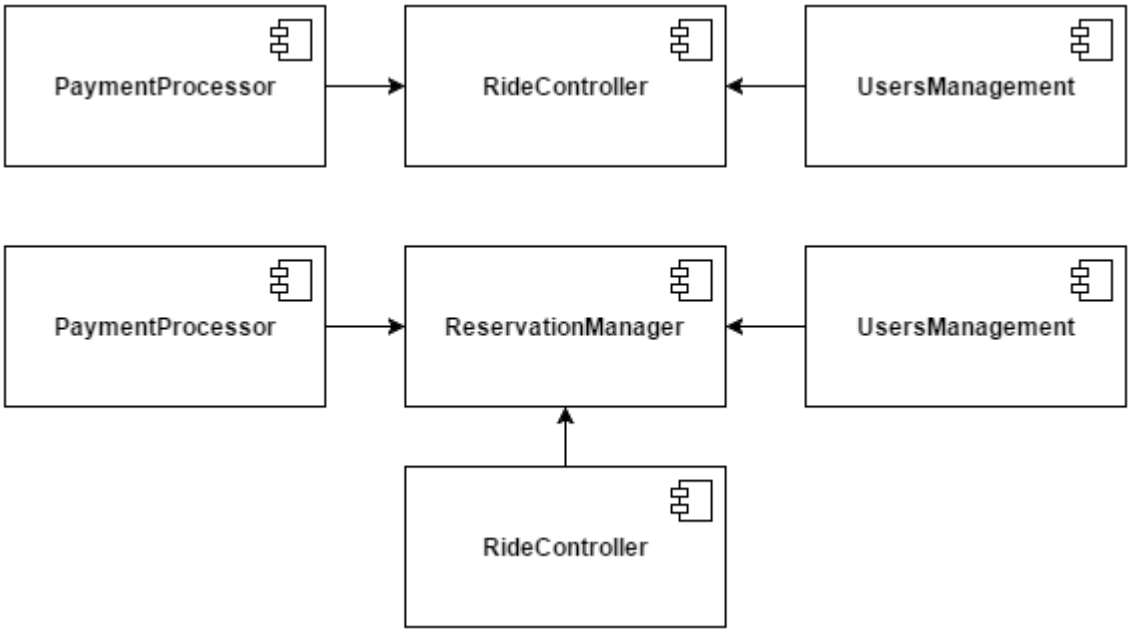
Continuing the integration with external third party components (following the line tracked in the previous section), the next one is the components couple for payments:



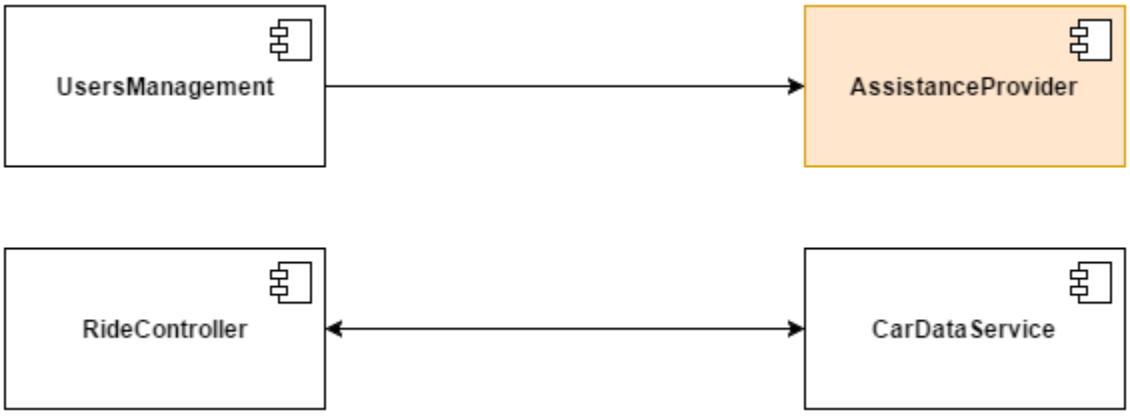

Then, we can integrate PaymentProcessor with UserManagement, following the dependency:



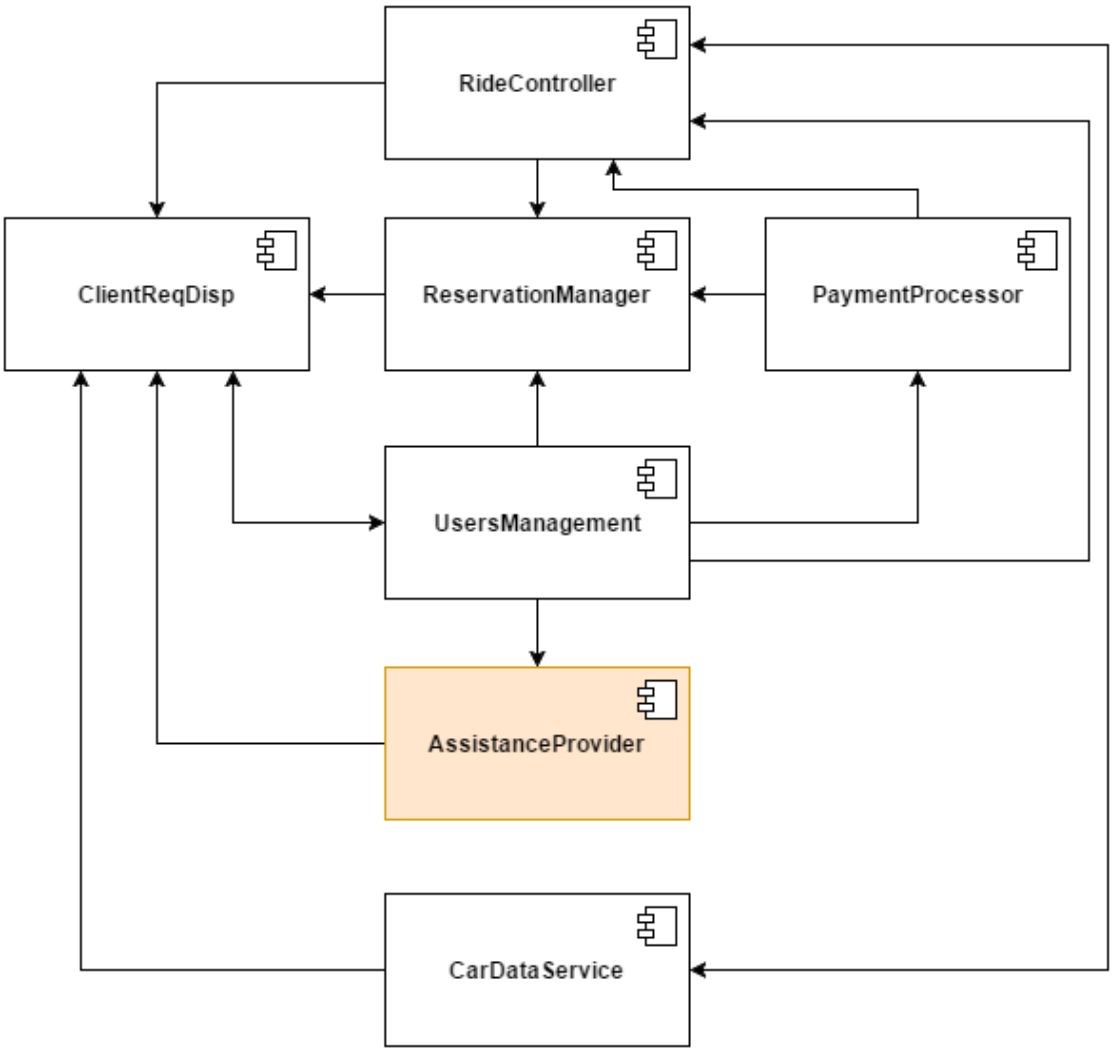

Since both PaymentProcessor and UsersManagement have now been integrated, and do not need further dependencies, we shall move to introducing RideController and ReservationManager, which only rely on those two components. Since RideController is a dependency for ReservationManager, we introduce it as second:



Finally, we complete the dependencies for AssistanceProvider and CarDataService:




We have introduced all components and integrated them, bottom-up, for the User services system. Now we can build up the entire subsystem by integrating the ClientReqDisp component:



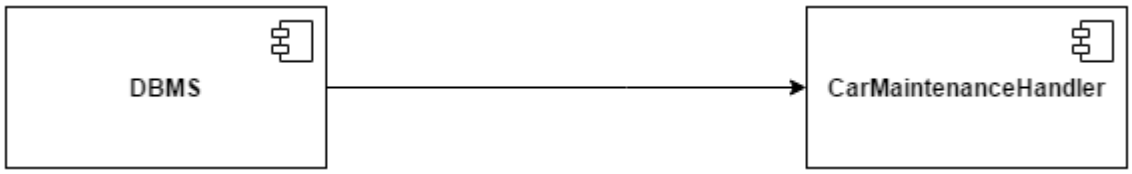
**MANAGEMENT SYSTEM**

For the *Management system*, we follow the same outline of the previous sub-system integration. We start from the connection with the DBMS:


Then, since the dependencies of the shared component *AssistanceProvider* have already been solved when integrating the *User services system* sub-system, we can directly integrate the *SystemManReqDisp* and build up the final sub-system integration:


**OPERATORS SYSTEM**

Once more, we start integrating the components that connect to the DBMS:




Now we can integrate the *CarMaintenanceHandler* component. This one is a dependency of CarDataService, which is part of another subsystem, *User services system*. We show here its integration, but it will be relevant in the sub-systems integration phase:

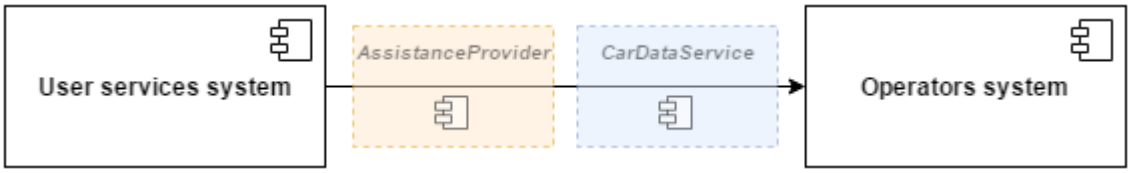

We can now integrate the last component and complete this sub-system:

### 2.4.2 Subsystem integration sequence

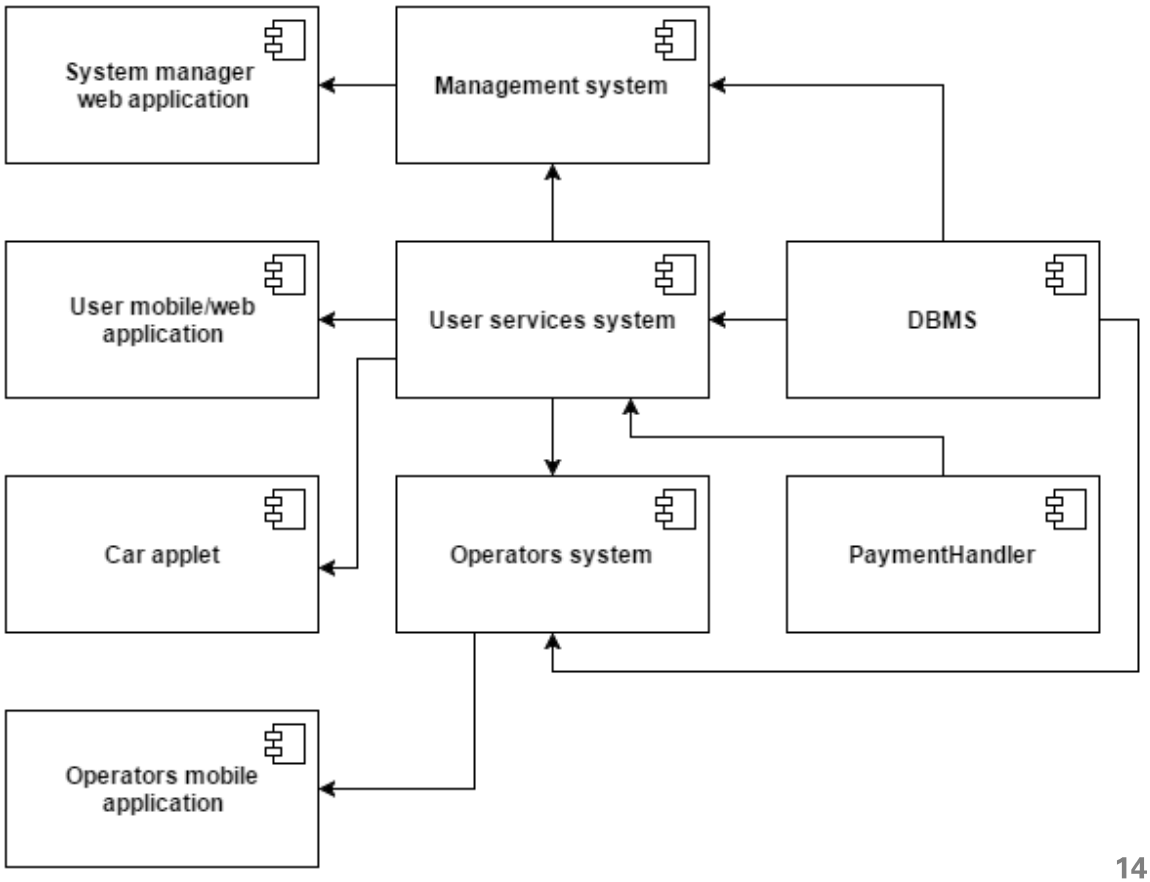
The sub-system integration sequence is delicate, since all the three subsystems share one component (*AssistanceProvider*) and *User services system* also includes another component that has a dependency into *Operators system*.

We start by integrating together *Management system* and *User services system*, via the *AssistanceProvider* shared component (the non-formal graphical representation is only for the sake of clarity):


Then, we integrate *Operators system*, via both the *AssistanceProvider* component and the *CarDataService/CarMaintenanceHandler* dependency:




We are now ready to integrate the whole system together, by integrating the three sub-systems as well as the other external components mentioned in the previous sections and in this very one:



# 3. Individual steps and test description

In this section, we will provide a detailed description of the test to be performed on the components of our system. We will test every method involved and used by the components highlighting the expected results, given various inputs, including the ones resulting in an error.

Tests are presented following the integration sequence previously explained, for every new couple of components introduced: if a new component A has been integrated, and needs B to work, the test for the couple A-B will target B's exposed methods that A will actually use. Components couples are indicated at the beginning of every test, as subsections titles (e.g. JustIntegratedComponent, TestedMethodsComponent).

The ← icon means that one or multiple preivous test(s) must be repeated (for the sake of brevity, the full test case has not been repeated more than once)

The methods we'll be referring to are consistent with Section 2.5 of the DD document previously redacted, where we described every internal interface of our system listing every method that can be possibly called.

⚠ *As already stated in the previous section of this document we are not going to present test descriptions for the DBMS and PaymentHandler integrations: see note in Section 2.3*: Integration Testing Strategy.

Note that not every exception raised as an effect is a basic Java exception; the custom ones are exceptions supposed to be defined in the code during the development.

Moreover, it's important to notice that when we refer to an 'invalid' input we mean an input that doesn't match the format of the expected parameter; for example, if a string is used when an integer value is required. For this reason, an exception is raised to prevent the mismatched parameter to be fed to the method, causing the system to crash. When the input is simply wrong or incompatible with the others no exception is raised, but it's the method itself that manages the error, returning a specific value to let that the caller know an error occurred.

## 3.1 User Services System

### 3.1.1 PaymentProcessor, UsersManagement

disableInsolventUser(String username)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| A non existent username | An UnregisteredUserException is raised |
| A valid username | The user is found in the database and is marked as insolvent |

enableUser(String username)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| A non existent username | An UnregisteredUserException is raised |
| A valid username | The user is found in the database and is enabled |

### 3.1.2 RideController, PaymentProcessor

executeTransaction(String user, Ride ride, float amount)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid parameter | An InvalidArgumentException is raised |
| A not registered user | An UnregisteredUserException is raised |
| A not existent ride | An UnknownRideException is raised |
| Valid information | The transaction is executed via the payment handler |

getPaymentResult(String username)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid parameter | An InvalidArgumentException is raised |
| A not registered user | An UnregisteredUserException is raised |
| A valid user | The payment result for the specified user is retrieved and returned |

### 3.1.3 RideController, UsersManagement

getCompleteUserData(String username)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| A non existent username | An UnregisteredUserException is raised |
| A valid username | The user is found in the database; his data are retrieved and returned |

positionShare(String username)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| A non existent username | An UnregisteredUserException is raised |
| A valid username | The current position of the specified user is returned |

### 3.1.4 ReservationManager, PaymentProcessor

payFee(Reservation reservation)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid parameter | An InvalidArgumentException is raised |
| A not existent reservation | An UnknownReservationException is raised |
| A valid reservation whose time did not expire | The fee is not paid; 'False' is returned |
| A valid reservation whose timer expired | The fee for the specified reservation is processed and automatically paid via the payment handler; 'True' is returned |

### 3.1.5 ReservationManager, UsersManagement

← getCompleteUserData(String username)

### 3.1.6 ReservationManager, RideController

createRideForRes(String user, Reservation res, List shUs)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid parameter | An InvalidArgumentException is raised |
| A non existent user | An UnregisteredUserException is raised |
| Valid information | A reservation for the specified user, with every eventual shared user, is created and saved into the database |

### 3.1.7 AssistanceProvider, UsersManagement

← getCompleteUserData(String username)

### 3.1.8 CarDataService, RideController

getRideTime(Ride ride)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid ride | An InvalidArgumentException is raised |
| A non existent ride | An UnknownRideException is raised |
| Valid information | The time of the specified ride is retrieved from the database and is returned |

getRidePrice(Ride ride)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid ride | An InvalidArgumentException is raised |
| A non existent ride | An UnknownRideException is raised |
| Valid information | The price of the specified ride is retrieved from the database and is returned |

startCounting(Ride ride)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existent ride | An UnknownRideException is raised |
| An existent ride which is already being counted | A UnavailableRideException is raised |
| A valid ride | The counter for the specified ride is started |

stopCounting(Ride ride)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existent ride | An UnknownRideException is raised |
| An existent ride whose counter has already been stopped | A UnavailableRideException is raised |
| A valid ride | The counter for the specified ride is stopped |

### 3.1.9 RideController, CarDataService

unlock(Car car)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid parameter | An InvalidArgumentException is raised |
| A not registered car | An UnknownCarException is raised |
| A valid car | The given car is queried from the database and then it's unlocked |

lock(Car car)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid parameter | An InvalidArgumentException is raised |
| A not registered car | An UnknownCarException is raised |
| A valid car | The given car is queried from the database and then it's locked |

getPosition(Car car)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid parameter | An InvalidArgumentException is raised |
| A not registered car | An UnknownCarException is raised |
| A valid car | The current position of the car, asked to the car itself, is returned |

updateTime(Car car, Time time)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid parameter | An InvalidArgumentException is raised |
| A not registered car | An UnknownCarException is raised |
| Valid information | The car is queried from the database and then the updated time is sent to the car |

### 3.1.10 ClientReqDisp, RideController

getAllRidesForUser(User user)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid user | An InvalidArgumentException is raised |
| A non existent user | An UnregisteredUserException is raised |
| Valid information | A list of every ride of the given user is retrieved from the database and is returned |

termRide(User user, Ride rd, ParkingArea pa, CarLeftConditions clc)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid parameter | An InvalidArgumentException is raised |
| A non existent ride | An UnknownRideException is raised |
| A non existent user | An UnregisteredUserException is raised |
| A non existent parking area | An UnknownParkingAreaException is raised |
| Valid information | The specified ride for the given user is retrieved and labelled as terminated. Then information about the parking area it has been left and the car conditions are saved into the database and are linked to the just terminated ride |

### 3.1.11 ClientReqDisp, ReservationManager

getReservableCars(Location center, int range)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid location center or range | An InvalidArgumentException is raised |
| Valid and consistent information | A list of all the reservable cars within the specified range from the given center point is retrieved from the database and returned |

getParkingAreas(Location center, int range)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid location center or range | An InvalidArgumentException is raised |
| Valid and consistent information | A list of all the parking areas within the specified range from the given center point is retrieved from the database and returned |

createReservation(String user, Car car)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid user or car | An InvalidArgumentException is raised |
| A non existent username | An UnregisteredUserException is raised |
| Valid information | The specified user is retrieved from the database and a new reservation for the given car is created and associated to him |

checkReservationStatus(Reservation reservation)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid reservation | An InvalidArgumentException is raised |
| A valid reservation | The requested reservation is retrieved from the database; then its status is returned |

addSharedUser(String user, Reservation reservation, String userSharing)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid user, sharing user or reservation | An InvalidArgumentException is raised |
| A non existent user or sharing user | An UnregisteredUserException is raised |
| Valid information | The requested reservation of the specified user is retrieved from the database and the sharing user is added. The updated reservation is then put into the database |

markSucceded(String user, Reservation reservation)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid user or reservation | An InvalidArgumentException is raised |
| A non existent user | An UnregisteredUserException is raised |
| Valid information | The requested reservation for the specified user is retrieved from the database and it's then marked as succeded |

cancelReservation(String user, Reservation reservation)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid user or reservation | An InvalidArgumentException is raised |
| A non existent user | An UnregisteredUserException is raised |
| Valid information | The requested reservation for the specified user is retrieved from the database and it's then marked as canceled |

getActiveReservations()

| Input | Effect |
|---|---|
| - | The database is queried to return a list containing all the currently active reservations |

### 3.1.12 ClientReqDisp, UsersManagement

attemptLogin(String mail, String password)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid e-mail address or password | An InvalidArgumentException is raised |
| A wrong password | Information are checked into the database and 'False' is returned |
| An e-mail address not present in the database | An UnregisteredUserException is raised |
| Valid and consistent information | Information are validated checking into the database and 'True' is returned |

askRegistration(RegisteredUser candidate)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| A guest with invalid information | An InvalidArgumentException is raised |
| A guest with valid and consistent information | The guest information is put in the database becoming a registered user and 'True' is returned |

askUpdate(String username, RegisteredUser newData)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A non existent username | An UnregisteredUserException is raised |
| Valid and consistent information | The user is found in the database and his information are updated with the provided data; then 'True' is returned |

### 3.1.13 UsersManagement, ClientReqDisp

confirmPosition(String user, Location pos)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existent user | An UnregisteredUserException is raised |
| Valid information | The specified position of the given user is forwarded to the caller |

### 3.1.14 ClientReqDisp, AssistanceProvider

createAssistanceRequest(RegisteredUser who, String msg, int time, Ride rd)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid parameter | An InvalidArgumentException is raised |
| A non existent ride | An UnknownRideException is raised |
| A non existent user | An UnregisteredUserException is raised |
| Valid information | An assistance request involving all the given information is created and added to the database. Then the newly created request is linked to the specified user |

### 3.1.15 ClientReqDisp, CarDataService

getIgnitionStatus(Car car)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid parameter | An InvalidArgumentException is raised |
| A not registered car | An UnknownCarException is raised |
| A valid car | The current ignition status of the car, asked to the car itself, is returned |

## 3.2 System management system

### 3.2.1 SystemManReqDisp, AssistanceProvider

flagRequestAsTaken(AssistanceRequest ar, User byWho)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid parameter | An InvalidArgumentException is raised |
| A non existent assistance request | An UnknownAssistanceRequest is raised |
| A non existent operator/system manager | An UnknownOperatorException / UnknownSystemManagerException is raised |
| Valid information | The specified operator/system manager is linked to the given assistance request and it is removed from the list of the available ones |

flagRequestForOperators(AssistanceRequest ar)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid parameter | An InvalidArgumentException is raised |
| A non existent assistance request | An UnknownAssistanceRequest is raised |
| Valid information | The specified assistance request is flagged so that it become visible in the list of requests maganed by the operators |

getAssistanceRequestForManagers()

| Input | Effect |
|---|---|
| - | The database is queried to return a list containing all the assistance requests related to the managers |

flagRequestSolved(AssistanceRequest ar)

| Input | Effect |
|---|---|
| - | The specified assistance request is flagged as solved in the database |

### 3.2.2 SystemManReqDisp, FleetParksOrganizer

createParkingArea(ParkingArea newArea)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| An already existent parking area | An AlreadyExistentAreaException is raised |
| A valid area | The new area is created |

editParkingArea(ParkingArea oldArea, ParkingArea newArea)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existent old parking area | An UnknownParkingAreaException is raised |
| An already existent new parking area | A UnavailableAreaException is raised |
| Valid parking areas | The old parking area is updated with the new one |

getParkingAreaList()

| Input | Effect |
|---|---|
| - | A list of every parking area is returned |

getParkingAreaByName(String name)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existing name | An UnknownAreaException is raised |
| A valid name | The parking area with the specified name is returned |

getCarList()

| Input | Effect |
|---|---|
| - | A list of every car in the fleet is returned |

getCarByPlate(String plate)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existing plate | An UnknownCarException is raised |
| A valid plate | The car with the specified plate is returned |

addCar(Car newCar)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| An already existing car | An UnavailableCarException is raised |
| A valid car | The new car is added to the fleet |

removeCar(Car car)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existing car | An UnknownCarException is raised |
| A valid car | The specified car is removed from the fleet |

editCarPlate(Car car, String newPlate)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existing car | An UnknownCarException is raised |
| An already assigned plate | An AlreadyUsedPlateException is raised |
| Valid information | The plate of the specified car is updated with the new value |

## 3.3 Operators system

### 3.3.1 CarDataService, CarMaintenanceHandler

makeCarAvailable(Car car, int battery, Boolean plugged)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existing car | An UnknownCarException is raised |
| An already available car | An AlreadyAvailableCarException is raised |
| A too low battery level | A BatteryTooLowException is raised |
| Valid information | The specified car is marked as available and its battery level is updated |

markCarOutOfOrder(Car car)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existing car | An UnknownCarException is raised |
| An already out of order car | An AlreadyOutOfOrderCarException is raised |
| A valid car | The specified car is marked as out of order |

markCarLowBattery(Car car)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existing car | An UnknownCarException is raised |
| An already low battery car | An AlreadyLowBatteryCarException is raised |
| A valid car | The specified car is marked as a low battery one |

### 3.3.2 OperatorsReqDisp, CarMaintenanceHandler

getLowBatteryCars()

| Input | Effect |
|---|---|
| - | A list of every car with low battery is returned |

getOutOfOrderCars()

| Input | Effect |
|---|---|
| - | A list of every out of order car with is returned |

markCarAssigned(Car car, Operator who)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existing car | An UnknownCarException is raised |
| An already assigned car | An AlreadyAssignedCarException is raised |
| A not existing operator | An UnknownOperatorException is raised |
| Valid information | The specified car is assigned to the given operator, being added to his task list; the car is marked as assigned |

### 3.3.3 OperatorsReqDisp, AssistanceProvider

getAssistanceRequestForOperators()

| Input | Effect |
|---|---|
| - | The database is queried to return a list containing all the assistance requests related to the operators |

← flagRequestAsTaken(AssistanceRequest ar, User byWho)

← flagRequestAsSolved(AssistanceRequest ar)

## 3.4 Subsystem integration

### 3.4.1 Management system, User services system

← repeat tests of section 3.2.1

### 3.4.2 Operators system, User services system

← repeat tests of section 3.3.1

← repeat tests of section 3.3.3

### 3.4.3 User mobile/web app., User services system (ClientReqDisp)

login(String mail, String password)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid parameter | An InvalidArgumentException is raised |
| A not registered mail | An UnregisteredUserException is raised |
| Valid information | The specified user information are checked and the user is logged into the system |

register(RegisteredUser candidate)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| A guest with invalid information | An InvalidArgumentException is raised |
| A guest with valid and consistent information | The guest information is put in the database becoming a registered user |

update(String username, RegisteredUser newData)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existent username | An UnregisteredUserException is raised |
| Valid and consistent information | The user is found in the database and his information are updated with the provided data |

getMapMarks(Location center, int range, boolean parks)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A valid location and range and 'True' as third parameter | A list containing every relevant element on the map within the given area, including parking areas, is returned |
| A valid location and range and 'False' as third parameter | A list containing every relevant element on the map within the given area, excluding parking areas, is returned |

reserveCar(Car which, String user)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existent user | An UnregisteredUserException is raised |
| An unregistered car | An UnknownCarException is raised |
| A user who has an already active reservation | A ConcurrentReservationException is raised |
| A user with no active reservations and an already reserved car | A ConcurrentReservationException is raised |
| A user with no active reservations and a not reserved car | The specified car is marked as reserved by the indicated user |

getTimer(Reservation reservation)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| An already expired reservation | A ReservationExpiredException is raised |
| A valid and currently active reservation | The time remaining for the specified reservation to be canceled automatically by the system is returned |

askUnlock(Reservation reservation, Location position)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| An already expired reservation | A ReservationExpiredException is raised |
| Valid information | The reservation timer is stopped and the car unlock method is called only if the user requesting the unlock is near the car. Then the ride timer is started |

askAssistance(String user, String message, Ride context)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existent user | An UnregisteredUserException is raised |
| A not existent ride | An UnknownRideException is raised |
| A valid ride not belonging to the user | A MismatchingRideException is raised |
| Valid information | An assistance request involving the specified user and ride, and including the indicated message, is created |

shareRide(Reservation which, String user)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existent user | An UnregisteredUserException is raised |
| A not existent/expired reservation | An UnknownReservationException is raised |
| Valid information | The specified user is added as shared user to the indicated reservation |

askLock(String user, Ride ride)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existent user | An UnregisteredUserException is raised |
| A user positioned far from the car he's trying to lock | The position of the user and the position of the car related to the given ride are retrieved and then compared. 'False' is returned and the lock request is not processed |
| A user positioned near the car he's trying to lock, within a maximum tolerance range | The position of the user and the position of the car related to the given ride are retrieved and then compared. 'True' is returned and the lock request is processed |

retrieveRidesList(String user)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existent user | An UnregisteredUserException is raised |
| A valid user | A list containing every ride the specified user has ever had is returned |

### 3.4.4 System man. web app., Management system (SystemManReqDisp)

getCarsInFleet(List<Status> filter, String plateFilter)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| Valid information | A list of all the cars registered in the fleet and matching the given filters, about status and plate, are returned |

getParkingAreas(Location position, int range)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| Valid information | A list of every registered parking area within the specified range from the given position is returned |

updatePark(ParkingArea oldArea, List<Location> newBounds, String newName, Bool plugArea)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existent parking area | An UnknownParkingAreaException is raised |
| An empty list of bounds | 'False' is returned and the method terminates without updating the area |
| Valid information and 'True' as last parameter | The parking area is updated using the given parameters and it's then also marked as a plug area |
| Valid information and 'False' as last parameter | The parking area is updated using the given parameters and it's then also marked as a non-plug area |

addPark(String name, List<Location> bounds, Bool plug)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| An already used name | An AlreadyUsedNameException is raised |
| An empty list of bounds | 'False' is returned and the method terminates without creating the area |
| Valid information and 'True' as last parameter | The parking area is created and added using the given parameters; it's then also marked as a plug area |
| Valid information and 'False' as last parameter | The parking area is created and added using the given parameters; it's then also marked as a non-plug area |

removePark(String name)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existent parking area | An UnknownParkingAreaException is raised |
| A valid and existent parking area | The parking area is removed |

addCar(String newCarPlate)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A valid plate | A car with the specified plate is added to the fleet |

removeCar(String oldCarPlate)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A plate not associated with any car in the fleet | An UnknownCarException is raised |
| A plate associated with a car in the fleet | The car with the specified plate is removed from the fleet |

getAssistanceList()

| Input | Effect |
|---|---|
| - | A list containing every assistance request is returned |

forwardToOperators(AssistanceRequest request)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existent request | An UnknownRequestException is raised |
| A valid assistance request | The specified assistance request is removed from the list of request managed by the system managers and added to the list of requests managed by the operators |

markSolved(AssistanceRequest request)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existent request | An UnknownRequestException is raised |
| A valid assistance request | The specified assistance request is marked as solved and removed from the list |

### 3.4.5 Operators mobile application, Operators system (OperatorsReqDisp)

getUnavailableCars()

| Input | Effect |
|---|---|
| - | A list of every unavailable car is returned |

getAssistanceRequests()

| Input | Effect |
|---|---|
| - | A list of every assistance request managed by the operators is returned |

takeChargeOfCar(Car car)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existent car | An UnknownCarException is raised |
| An existent car which is not out of order | A UnavailableCarException is raised |
| A valid car which is in need of assistance | The specified car is removed from the list visible to every operator and added to the tasks of the operator who took it in charge |

takeChargeOfAssistanceRequest(AssistanceRequest request)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existent assistance request | An UnknownAssistanceReqException is raised |
| An existent assistance request which is not in the list of the ones managed by the operator | An UnavailableAssistanceRequest is raised |
| A valid assistance request in the list of the ones managed by operators | The specified assistance request is removed from the list visible to every operator and added to the tasks of the operator who took it in charge |

markIssueSolved(Car car)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existent car | An UnknownCarException is raised |
| An existent car which hadn't been taken in charge | A UnavailableCarException is raised |
| A valid car which had been taken in charge | The specified car is removed from the list of task of the operator who took it in charge and it's marked as available again |

markRequestSolved(AssistanceRequest request)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| Invalid information | An InvalidArgumentException is raised |
| A not existent assistance request | An UnknownAssistanceReqException is raised |
| An existent assistance request which hadn't been taken in charge | An UnavailableAssistanceRequest is raised |
| A valid assistance request which had been taken in charge | The specified assistance request is removed from the list of task of the operator who took it in charge and is marked as solved |

### 3.4.6 Car applet, User services system (CarDataService)

reEnableAfterCharge(int newBatteryLevel)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid parameter | An InvalidArgumentException is raised |
| A battery level below the minimum treshold | A LowBatteryLevelException is raised and the car that made the request is not re-enabled |
| A battery level above the minimum treshold | The car is re-enabled, flagging it in the database |

reportStatus(Car car, Status status, CarLeftConditions conditions)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid parameter | An InvalidArgumentException is raised |
| A not registered car | An UnknownCarException is raised |
| Valid information | The specified status is updated in the database with the conditions provided, for the indicated car |

queryRideStatus(Car car)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid parameter | An InvalidArgumentException is raised |
| A not registered car | An UnknownCarException is raised |
| Valid information | The specified car is queried from the database and its status is returned |

reportIgnited(Car car)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid parameter | An InvalidArgumentException is raised |
| A not registered car | An UnknownCarException is raised |
| An already ignited car | An AlreadyIgnitedCarException is raised |
| A valid car | The car reports that it has been ignited |

reportParked(Car car, ParkingArea where)

| Input | Effect |
|---|---|
| A null parameter | A NullArgumentException is raised |
| An invalid parameter | An InvalidArgumentException is raised |
| A not registered car | An UnknownCarException is raised |
| A not registered parking area | An UnknownParkingAreaException is raised |
| Valid information | The car is queried from the database and then is marked as parked in the given parking area |

# 4. Tools and test equipment required

## 4.1 Testing tools

To partially automate the integration tests described so far, we decided to use some famous software tools, that can speed up the testing phase as well as keeping it consistent and manageable.

The following tools provide useful functions for both test components integration and their individual features and functions, in particular:

**Arquillian integration testing framework** will support the actual integration testing, as it provides container testing and components *injection* testing, all dedicated to JEE environments. Since our application runs using this framework, we think *Arquillian* is the best choice. Additionally, this testing tool provides instruments to test database connections, and in general the interaction between our app and the database itself.

Even if we inserted these checks among the *Entry criteria,* using Arquillian we can have a second DB connection check-up once the various components have been integrated together.

For additional specific testing activities instead, we will use **JUnit**, the popular Java testing framework. Even it primarily used for unit testing, *JUnit* is an excellent tool to also exam components interactions, in the form of specific method invocations with particular parameters.

For this integration testing, we will use *JUnit* to check the correctness of returned parameters, the kind of exceptions in case of manually triggered errors, and so on.

Finally, to test the interaction between our system and mobile applications, we will use the bundled tools of the various SDKs for smartphone operative systems; these will support the classic integration testing by providing additional tools to troubleshoot problems with the smartphone apps. In particular, we are going to use the **Android SDK** for the Android app, the **XCode suite** for the iOS app and **Visual Studio + Windows Mobile SDK** for the WP app.

## 4.2 Testing equipment

### 4.2.1 Mobile applications

To correctly test the interaction between the platform and mobile applications, several devices varying in form-factor, screen size and resolution, brands and OS versions will be needed.

The following schema outlines a reasonable selection of these devices:

- **ANDROID**
  - One device for every major Android release from the first supported one to the latest release available:
    - 4.1 (JellyBean – API 16)
    - 4.4 (KitKat – API 19)
    - 5.0 (Lollipop – API 21)
    - 6.0 (Marshmallow – API 23)
    - 7.0 (Nougat – API 24)
  - One device for every DPI supported:
    - hdpi (240dpi)
    - xhdpi (320dpi)
    - xxhdpi (480dpi)
    - xxxhdpi (640dpi)
  - One device for every notable screen resolution:
    - 800x480
    - 1280x720
    - 1920x1080
    - 2560x1440
  - One device from every widely-spread manufacturer, to test customized Android versions:
    - Samsung
    - LG
    - Huawei
    - Sony
    - Motorola/Lenovo

- **IOS**
  - One device for every major iOS release from the first supported one to the latest release available:
    - iOS 7
    - iOS 8
    - iOS 9
    - iOS 10
  - Every device currently supported by Apple:
    - iPhone, from 5S to 7Plus
    - iPad, from 4 to Mini 4 (incl. Air and Air 2)
- **WINDOWS PHONE**
  - One device for every major WP release from the first supported one to the latest release available:
    - WP 7
    - WP 7.5
    - WP 8
    - WP 8.1
    - Windows 10 Mobile
  - One device for every notable screen resolution:
    - 800x480
    - 1280x720
    - 1920x1080

This granularity over mobile devices will guarantee Power EnJoy a smooth launch on all the three stores without major bugs impacting just one or two devices but widely used by the people.

For the same operative system, different kinds of testing devices should be combined in the greatest number of forms possible, thus ensuring that all the trickiest combinations have not been left untested (e.g. every screen size for every OS version, etc.).

### 4.2.2 Web browsers

Since Power EnJoy also provide a website for both users and *System Managers*, we must ensure that all its features are available for the largest user base possible.

We will need sessions of all the most popular web browsers (updated to latest versions), running on all the most popular desktop operative systems:

- **WEB BROWSERS**
  - Mozilla Firefox
  - Google Chrome
  - Microsoft Internet Explorer
  - Microsoft Edge
  - Opera
- **DESKTOP OS**
  - Microsoft Windows
  - Apple OSX / macOS
  - Linux Ubuntu

There are no particular requirements on the PCs specifications required for the testing.

### 4.2.3 Amazon Web Services testing

As explained in detail in Section 2.7.2: *Amazon AWS integration*, Power EnJoy will run on a scalable cloud architecture managed by Amazon Web Services. Although there are no specific requirements for the various services setup at testing time, it should be noted that a crucial part of the integration testing is indeed the deployment of Power EnJoy over the various AWS components.

Thus, to correctly perform these kinds of tests, **a working setup** of Amazon RDS, Amazon Elastic Beanstalk and relative Amazon EC2 instances **will be needed**, as well as a working AWS account with a sufficient budget for testing activities *pay-as-you-go* costs.

### 4.2.4 Car applet

To test the communication with cars and their integration in the overall system, they can be emulated using a custom software, since they transfer data with the server using an internet connection over cellular network. This 'mock' software, with internet access, can simulate the actual behavior of cars and their responses/requests in the various situations. Although this is enough for Power EnJoy own integration testing, **we think that a prototype car** equipped with the applet and all the accessory systems **should also be used**, to make sure that the modules to be deployed on the car do not behave unexpectedly once they actually are.

# 5. Program stubs and test data required

## 5.1 Program Stubs and Drivers

To perform all the tests listed in this document, and to be able to call all exposed methods of the different components, a series of drivers will be needed for every phase of the proposed integration sequence.

These drivers will be used in conjunction with JUnit (and, when opportune, with Arquillian too), following the bottom-up approach that we introduced previously in this document.

The drivers are:

- **PaymentProcessor Driver**, to test PaymentProcessor integration with PaymentHandler component by calling all exposed methods;
- **UsersManagement Driver**, to test the integration of the UsersManagement component with PaymentProcessor, ReservationManager and RideController components by calling all its exposed methods;
- **Reservations and Ride Driver**, to test ReservationsManager and RideController components together as they are integrated with UsersManagement and PaymentProcessor, by calling both components exposed methods;
- **AssistanceProvider Driver**, to verify AssistanceProvider component exposed methods when it is first integrated with the UsersManagement component;
- **CarDataService Driver**, to test CarDataService component methods - both when integrated with RideController in the *User services system* as well as when integrated with CarMaintenanceHandler in the *Operators system*;
- **ClientReqDisp Driver**, to test the ClientReqDisp component integration with RideController, ReservationsManager, AssistanceProvider, CarDataService and UsersManagement components using its exposed methods;
- **SystemManReqDisp Driver**, to verify SystemManReqDisp component integration with FleetParkOrganizer and AssistanceProvider, by calling all its exposed methods;
- **OperatorsReqDisp Driver**, that will call all OperatorsReqDisp component exposed methods to test its integration with both AssistanceProvider and CarMaintenanceHandler;

- **User services system Driver**, that will test all the methods of the relative sub-system (both the ones used by mobile/web apps and by the car applet) to check its integration with the other two sub-systems, *Management system* and *Operators system*;
- **Management system Driver**, that will call all exposed methods of *Management system* (called by system managers' web application) to test its integration with *User services system* and *Operators system*;
- **Operators system Driver**, that will test all the methods of *Operators system* (called by operators' mobile app) to check the integration of this sub-system with the other two, *User services system* and *Management system*.

For the correct integration testing of the upper components in our bottom-up approach, we will also need a few **stubs** to replace the actual presence of the mobile and web applications; these stubs, combined with the already mandatory completed part of those components (as detailed in the *Entry criteria* section previously), will receive responses from our servers and they will output them explicitly so that we can also check the integration between clients and the rest of the system. The same goes for the *Car applet*: we will need an additional stub that may be replaced with a real prototype if following the suggestions of section 4.2.4.

## 5.2 Test data

To complete the series of integration tests that we proposed, we will need some specific test data to use as input for the tests themselves. For the sake of brevity, we will not list both *Null inputs* and *Non-valid inputs* (i.e., mal-formatted input), that are obviously necessary for every test. Also, within the same component test data, repeated necessary inputs are repeated only once.

### 5.2.1 UsersManagement

- An email/username not already registered
- A wrong password for a valid email
- A valid couple of username/password registered in the DB
- A set of valid personal data of a user not already registered
- A set of valid personal data with updated info of a user already registered

### 5.2.2 ReservationManager

- A location (coordinates) and a range with some reservable cars within
- A location (coordinates) and a range with parking areas within
- An out-of-bounds/invalid/un-plausible location or range
- A valid reservation saved in the DB
- An invalid reservation/A reservation not saved in the DB
- Valid usernames of users not currently in a ride (for sharing)
- Some non-existent usernames/emails

### 5.2.3 RideController

- A valid reservation saved in the DB with the relative username and a list of registered sharing users
- Some non-existent usernames/emails
- An active (timer counting) ride saved in the DB
- An invalid ride/A ride not saved in the DB
- A registered username/email
- An invalid/non-existent parking area
- A parking area saved in the DB, free
- A set of plausible conditions for a car that just took a ride

### 5.2.4 AssistanceProvider

- A non-existent username/password
- An invalid ride/A ride not saved in the DB
- A registered username/email
- A message for requesting assistance, a plausible time/date for that message, and a valid registered ride active and associated to the user above

- An invalid assistance request/An assistance request not present in the DB
- Some valid assistance requests saved in the DB, not yet solved, for system managers
- Some valid assistance requests saved in the DB, not yet solved, for operators
- A non-existent operator's data
- A non-existent system manager data
- A valid operator's data registered in the DB
- A valid system manager data registered in the DB

### 5.2.5 CarDataService

- A non-existent car's data/A car not present in the DB
- A valid car data, saved in the DB
- A plausible battery level for a car, which has been just re-enabled after a charge (i.e. over 80%)
- A plausible battery level for a car, below the threshold for re-enabling cars (below 80%)
- An invalid battery level for a car
- A set of not plausible/invalid car conditions data
- A set of plausible car conditions data
- A ride saved in the DB and a car that is associated to that ride
- A valid car data not associated to any active ride
- A locked, registered car
- An unlocked, registered car
- An already ignited car
- A shutdown car
- An invalid parking area/a parking area not present in the DB
- An invalid/un-plausible time for a ride
- A set of a valid time for a ride, an active ride in the DB, a valid car associated to that ride

### 5.2.6 PaymentProcessor

- A non-existent username/password
- An invalid ride/A ride not saved in the DB
- An un-plausible amount for a ride payment
- A set of a valid username/email (registered), an active ride associated to that user, a plausible amount of money as total charged for that ride
- An invalid reservation/A reservation not saved in the DB
- A valid reservation, registered in the DB, whose time has not expired
- A valid reservation, registered in the DB and with its timer expired
- A valid username/email for which a transaction has just been completed

### 5.2.7 CarMaintenanceHandler

- A non-existent car/a car not registered in the DB
- An existent car's data, disabled, which battery is over 80%
- An existent car's data, disabled, which battery is below 80%
- An existent car, already available
- An existent car's data, which battery is below 20%
- An existent car's data, which battery is over 20%
- A set of an out-of-order car, valid and registered, and a valid operator, registered in the DB
- A invalid/non-existent operator

### 5.2.8 FleetParksOrganizer

- A new, valid parking area data
- An already existing parking area data
- Invalid data for a parking area
- The name of a parking area
- A plate of a valid car, registered in the DB
- An invalid plate/a plate of a car not present in the DB
- Valid data for a car not present in the DB
- Valid data of a car already present in the DB

### 5.2.9 ClientReqDisp

- An email/username not already registered
- A wrong password for a valid email
- A valid couple of username/password registered in the DB
- A set of valid personal data of a user not already registered
- A set of valid personal data with updated info of a user already registered
- A location (coordinates) and a range with some reservable cars and parks within
- An out-of-bounds/invalid/un-plausible location or range
- An invalid car, not in the DB
- A valid, registered car, reservable
- A valid, registered car, but already reserved
- A set of a valid registered username/email and valid registered car, but the user has already another reservation pending
- An invalid reservation/A reservation not saved in the DB
- A set of a valid reservation, registered in the DB, and a valid location close to the reservation's car position.
- A set of a valid reservation, registered in the DB, and a invalid/un-plausible location
- A set of a valid reservation, registered in the DB, and a valid location but too far from the reservation's car position

- A set of a valid username/password, a plausible assistance request message, and a valid registered active ride associated to that user
- A set of a valid username/password, a plausible assistance request message, and a valid registered active ride but not associated to that user
- A couple of a valid registered username/email and a reservation with timer not expired yet and not made from that user
- A couple of a valid registered username/email and a reservation with timer not expired yet but made from that user
- A couple of a valid registered username/email and a reservation with timer expired
- A valid, registered username/email
- A valid, plausible location for a user
- A couple of a valid registered username/email and an active ride associated to that user, with the user far from the ride's associated car
- A couple of a valid registered username/email and an active ride associated to that user, with the user close to the ride's associated car

### 5.2.10 SystemManReqDisp

- An invalid plate/a plate of a car not registered in the DB
- A plate of a car registered in the DB
- A list of status filters for cars in the DB
- A location (coordinates) and a range with some parks within
- An out-of-bounds/invalid/un-plausible location or range
- A valid/registered parking area
- An invalid parking area/not present in the DB
- A list of bounds for a parking area
- A new name for a parking area
- A name of an already existing parking area
- Some assistance requests in the DB
- A valid assistance request still assigned to system manager
- A valid assistance request already assigned to operators
- An invalid/not registered assistance request

### 5.2.11 OperatorsReqDisp

- A non-existent car/a car not registered in the DB
- An existent car, out of order
- An existent car, not out of order
- A valid assistance request still assigned to system manager
- A valid assistance request already assigned to operators
- An invalid/not registered assistance request
- An existent car data, which was just fixed by an operator

# 6. Appendices

## 6.1 Tools used

- Microsoft Office Word 2016
  *https://products.office.com/it-it/home*

  For redacting, reviewing, layout and graphic design of this document

- Draw.io
  *http://draw.io*

  For creating the ER diagram

## 6.2 Effort spent

The present document required almost the same time for both the curators (Leonardo Chiappalupi, Ivan Bugli).

The total time spent **apiece** on the creation of the paper is: ~20 Hours.

## 6.3 References


- **AMAZON WEB SERVICES:** http://aws.amazon.com
- **POWER ENJOY: REQUIREMENT ANALYSIS AND SPECIFICATION DOCUMENT – CHIAPPALUPI L., BUGLI I.** Politecnico di Milano, 2016
- **POWER ENJOY: DESIGN DOCUMENT – CHIAPPALUPI L., BUGLI I.** Politecnico di Milano, 2016
- **ANDROID VERSIONS:** https://www.android.com/history
- **IOS VERSIONS:** https://developer.apple.com
- **WINDOWS PHONE VERSIONS:** https://msdn.microsoft.com/it-it/library/windows/apps/hh202996(v=vs.105).aspx