PowerEnJoy
Integration Test Plan Document
Version 1

Patricia Abbud
Maddalena Andreoli Andreoni
Paolo Cudrano

Contents

1 Introduction

1.1 Revision History

Version	Date	Summary
1.0	15/01/2017	First Release

1.2 Purpose and Scope

The document you are approaching to read is the **Integration Testing Plan Document** for the **PowerEnJoy** system. It aims to describe in the most complete and unambiguous way the process we will adopt to perform integration testing on our system. Integration testing endeavours to excercise and assess the system's interfaces and its modules (at a macro-component level and at a subcomponent level) in order to identify and fix bugs and unexpected and undesired behaviour in their interactions between each other. It will be done concurrently with the system development, in an incremental way.
The document will be divided into four content sections. **Section 2** describes the **integration strategy** that we choose preliminiarily to the compilation of the document. It starts with listing the prerequisites that have to be met before the testing may begin (*Entry Criteria*); then it lists the elements to be integrated and describes the approach we will take to integration testing (*Integration Testing Strategy*). It then proceeds to describe the order of integration testing more in detail (*Sequence of component/function integration*).
Section 3 will delve into the specific test sets needed for integration. For each of them, the document provides a list of expected outcomes to a given input or situation. **Section 4** provides an in-depth list of all tools and equipment that we will need to perform integration testing on our system, for integration of both the server side of the application and the client side. **Section 5** will describe in details the scaffolding components (stubs and drivers) needed by the testing procedures and the test data necessary to support them.

1.3 List of Definitions and Abbreviations

Provided here is a short list of definitions and acronyms often used inside this document. Please refer to the previous documents' chapter on definitions and abbreviations for any term not described here.

1.3.1 Definitions

- **Macro-component**. We define as macro-component a whole logical section of the system. These macro-component have their own independent task or purpose in the system.
- **Subcomponent**. We define as subcomponent the elements of which a macro-component is made of; these carry out specific computations needed to perform the task of their macro-component.

1.3.2 Acronyms

- **RASD**: Requirement Analysis and Specification Document
- **DD**: Design Document
- **ITPD**: Integration Testing Plan Document
- **JEE**: Java Enterprise Edition
- **MSO**: Money Saving Option
- **OS**: Operating System

1.4 List of Reference Documents

Here is provided a list of documents we used as reference for the ITPD:

PowerEnJoy RASD : the Requirement Analysis and Specification Document for *PowerEnJoy*'s system.

PowerEnJoy DD : the Design Document for *PowerEnJoy*'s system.

Project Assignment : the project assignment document, *Assignments AA 2016-2017.pdf*.

2 Integration strategy

2.1 Entry criteria

In order for the integration process to be performed, a set of preconditions must be met. In particular:

Database The complete schema provided by the DD must be implemented and tested. The database access system must be at 100% of its development, and the database must be fully accessible.

Application logic Referencing to the components designed in section *2.2 - Component view* of the DD, they must all be unit tested before their integration starts. In addition, their development completion ratio must be so that all the functionalities needed for the integration are already fully developed.

Once begun, the integration testing follows and guides the development, as explained in subsection 2.3.

2.2 Elements to be integrated

As explained in the DD, the three layers in which the system is divided are characterized as follows.

Data A database holds and gives access to all the data needed by the application. As anticipated in subsection 2.1, this layer does not participate actively in the integration process. Indeed, since its development isn't particularly demanding in terms of resources, it is assumed to be already completed once the integration begins. This way, every logic component will be able to access the data layer during the unit testing phase, and integrate with it as a prerequisite for the proper integration phase.

Application logic The components on which most of the integration effort will be spent are those intensively described in section *2.2 - Component view* of the DD. These represent the main software components of the system and are responsible to provide the functionalities required by it. In particular, the integration will proceed on two different abstraction layers (one considering only macro-components, the other considering the internal interactions among subcomponents), and a particular attention will be paid for the subcomponents residing on different machines (i.e. server and client). For this last reason, when dealing with subcomponents we will graphically differentiate them according to their deployment location, coloring in yellow the ones on client machines and in light-blue the ones on server machines.

View The view layer won't be affected by the integration process. It will be unit tested during its development, and since its instances are each connected only with a single application logic component, their integration will be straightforward and won't need a delicate testing. The view functioning will be however tested during the more general *system testing* phase.

In this analysis, the website, for its simple and self-contained nature, will not be taken into account as it will not need an integration phase, jumping directly to the system testing phase.

As a result, the set of macro-components to be considered for the integration testing becomes definite. Following are enlisted the macro-components on which the integration will be performed (at an internal-level first, and at a system-level after):

- **User Location Handler**
- **Operator Location Handler**
- **Car Location Handler**
- **User Management server-side** with subcomponents: Profile Manager, License Manager
- **User Management client-side** with subcomponent: User Handler
- **Car Manager**
- **Access Manager**
- **Parking Manager**
- **Car Employment Manager**
- **Maintenance Manager client-side** with subcomponent: Operator Handler
- **Maintenance Manager server-side** with subcomponents: Emergency Report Handler, Report Status Handler, Dispatch Manager
- **Money Saving Option Manager**
- **Backend Manager client-side** with subcomponent: Admin Handler
- **Backend Manager server-side** with subcomponent: Backend Manager
- **Payment Manager**

2.3 Integration testing strategy

The integration of the system will be guided by a bottom-up approach. This strictly comes from an analysis of the dependency structure of the components diagram provided in the DD, restructured in Figure 1 and Figure 2 to better highlight the key modules and their connections.

This strategy allows for a close interaction between the development phase and the proper testing phase, with a benefit for the quality of the software-to-be as well as for the project management. The goal set with this choice is to make development and testing actually going in parallel during standard operating periods.
The approach used to integrate the different subcomponents inside a single macro-component

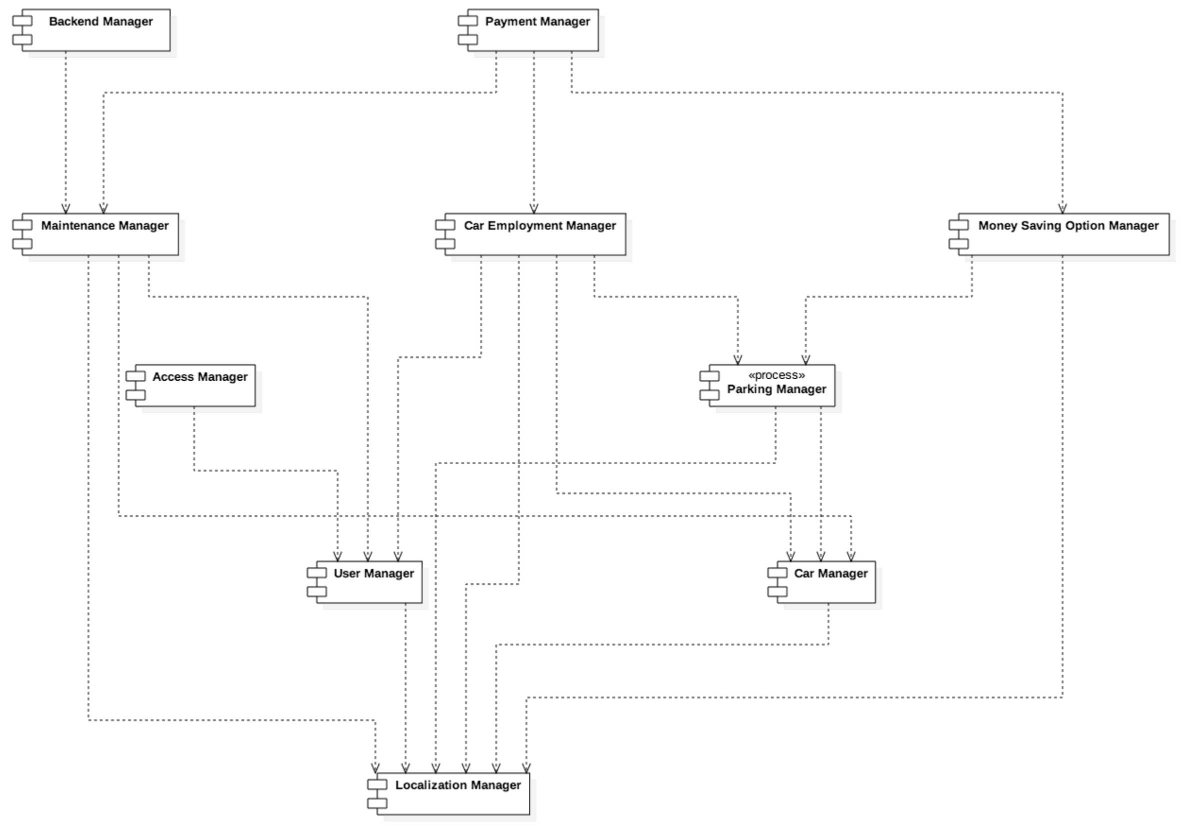


Figure 1: Dependency tree of components: high level view.

is instead selected according to its level of criticality; in particular, the choice revolves around a "nested" bottom-up approach on one hand, and a more complex critical-first approach on the other.
A criticality analysis underlines the following critical macro-components.

- **Car Manager.** The integration of the car system software with the underlying hardware and the sensors it is managing involves a moderate number of uncontrollable variables and introduces with them a certain amount of risk. In fact, it is the only point of contact with the actual car, thus becoming the contact point between two rather complex systems.

- **Money Saving Option Manager.** Even if not crucial for the basic functioning of the system, this macro-component is responsible for a large amount of computations, carried on by means of complex algorithms. These must be tested as soon as possible in order to obtain the desired results. In addition, the main algorithm provided by the *Parking Distribution Handler* is designed to be implemented with the not-so-common paradigm of fuzzy logic (see DD *section 3*), and as any not-well-spread development tool, this may suffer for less stability and minor flaws, which must be discovered and managed as soon as possible.

- **Payment Manager.** The payment handling is known to be one of the most critical activities since it involves money transactions. In particular, since the system relies on an external company for the actual payment realizations, its most critical subcomponent is the Invoice Manager, responsible for the interactions with the external *Payment Services Provider*.

As a result, when the said macro-components are to be developed, their internal integration will follow a critical-first approach. This increases the chances that any significant problem is found as soon as possible, and therefore should provide benefits in the project management (rescheduling of tasks, reallocation of resources, etc.) aside from the integration process itself.
The rest of the macro-components, given their less risky nature, will be internally developed following a bottom-up strategy, where the subcomponents are considered all equally-critical and only the "need" dependencies will drive the process.

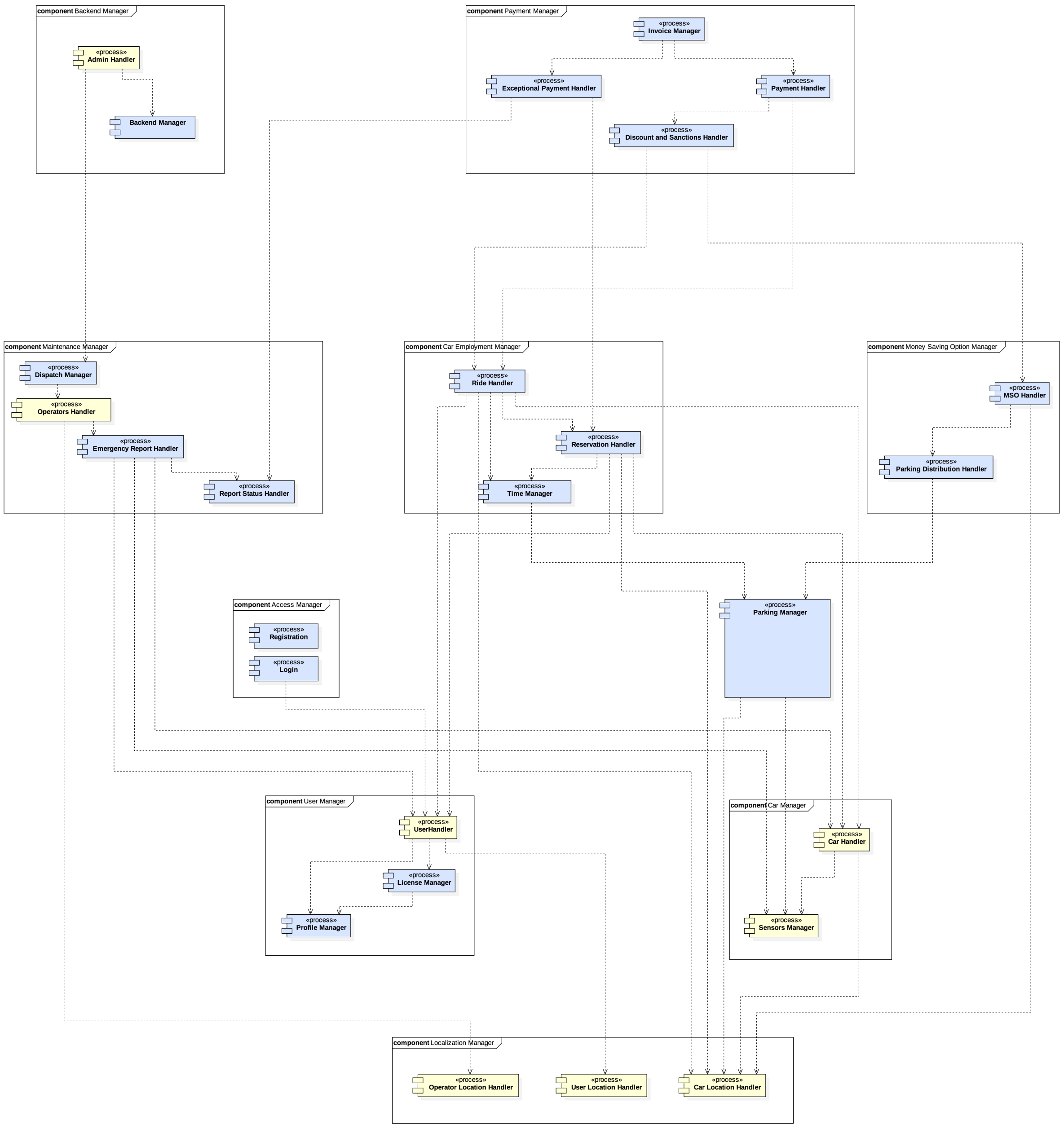

component Backend Manager
«process» Admin Handler
Backend Manager
component Payment Manager
«process» Invoice Manager
«process» Exceptional Payment Handler
«process» Payment Handler
«process» Discount and Sanctions Handler
component Maintenance Manager
«process» Dispatch Manager
«process» Operators Handler
«process» Emergency Report Handler
«process» Report Status Handler
component Car Employment Manager
«process» Ride Handler
«process» Reservation Handler
«process» Time Manager
component Money Saving Option Manager
«process» MSO Handler
«process» Parking Distribution Handler
component Access Manager
«process» Registration
«process» Login
«process» Parking Manager
component User Manager
«process» UserHandler
«process» License Manager
«process» Profile Manager
component Car Manager
«process» Car Handler
«process» Sensors Manager
component Localization Manager
«process» Operator Location Handler
«process» User Location Handler
«process» Car Location Handler

Figure 2: Dependency tree of components: complete system.

2.4 Sequence of component/function integration

In the present section, the general criteria outlined before are translated to the actual sequence of integration steps to follow in the development and testing of the system. Their presentation will be made exploiting two levels of detail, in a bottom-up fashion: subsubsection 2.4.1 refers to the integration of the various subcomponents in the context of a particular macro-component, while subsubsection 2.4.2 increases the abstraction level, focusing on how the different macro-components will be assembled to form the whole system.

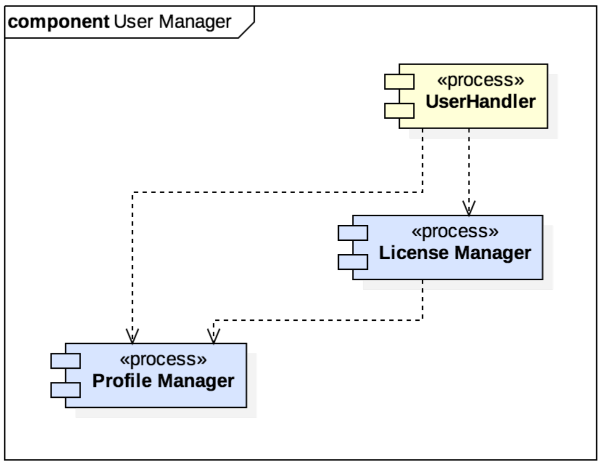
2.4.1 Software integration sequence

Localization Manager
This component does not require internal integration and will only function as an helper for other components.

User Manager

Internal integration strategy: bottom-up.
Integration order:

- Profile Manager
- License Manager
- User Handler

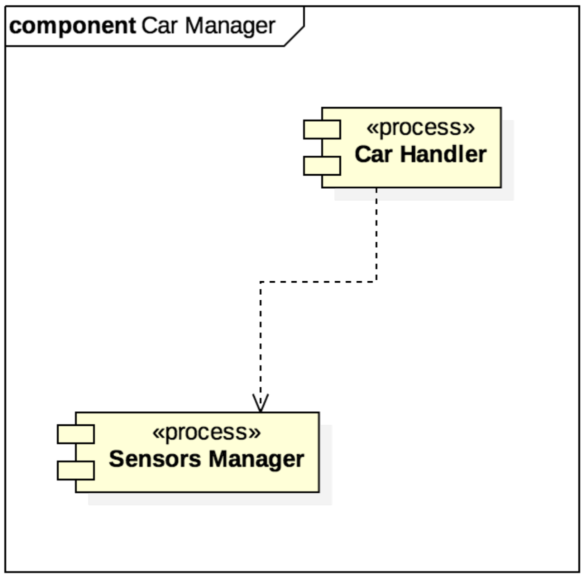


Car Manager

Internal integration strategy: critical-first.
Integration order:

- Sensors Manager
- Car Handler

Notes: The integration starts from the Sensors Manager because most of the criticality is related to the communication and management of the system's hardware, functionalities provided by it.



Parking Manager

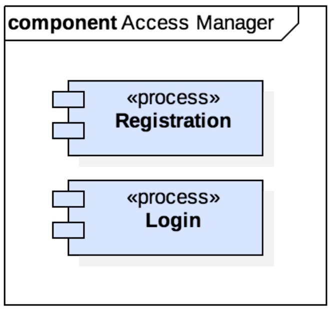
This component is atomic and thus does not require internal integration.

Access Manager

Internal integration strategy: bottom-up.
Integration order:

- Login
- Registration

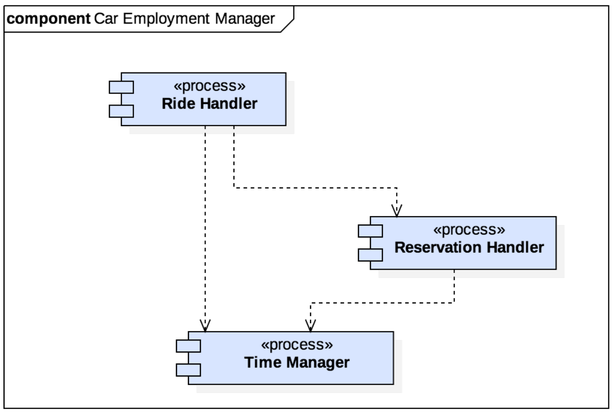
Notes: The integration starts from the Login because it must be subsequently integrated to the rest of the system, while Registration is independent from it.



Car Employment Manager

Internal integration strategy: bottom-up.
Integration order:

- Time Manager
- Reservation Handler
- Ride Handler

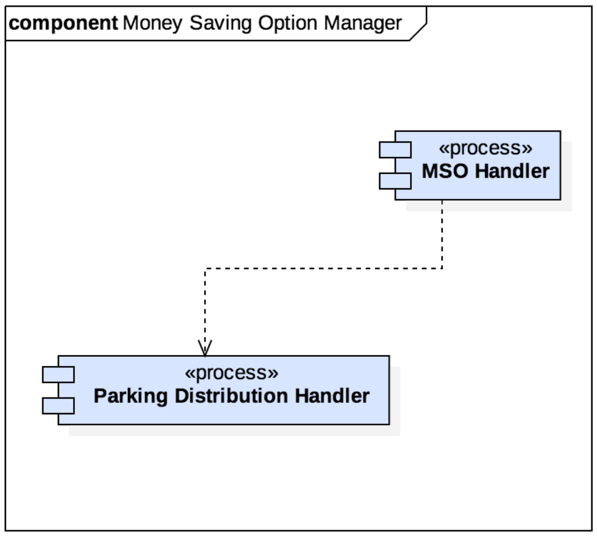

component Car Employment Manager
«process» Ride Handler
«process» Reservation Handler
«process» Time Manager

Money Saving Option

Internal integration strategy: critical-first.
Integration order:

- Parking Distribution Handler
- MSO Handler

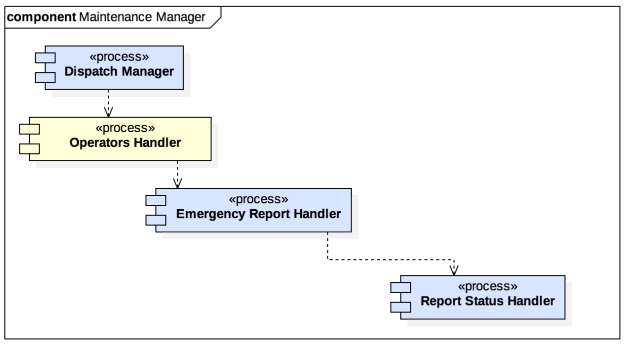
Notes: The Parking Distribution Handler is considered to be the most critical subcomponent between the two, since it has to deal with the core MSO algorithm, moreover exploiting not completely supported tools (as FCL for the fuzzy system).


component Money Saving Option Manager
«process» MSO Handler
«process» Parking Distribution Handler

Maintenance Manager

Internal integration strategy: bottom-up.
Integration order:

- Report Status Handler
- Emergency Report Handler
- Operators Handler
- Dispatch Manager


component Maintenance Manager
«process» Dispatch Manager
«process» Operators Handler
«process» Emergency Report Handler
«process» Report Status Handler

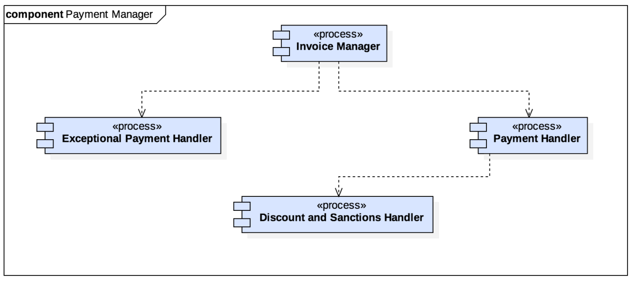
Payment Manager

Internal integration strategy: critical-first.
Integration order:

- Invoice Manager
- Payment Handler
- Exceptional Payment Handler

– Discount and Sanction Handler

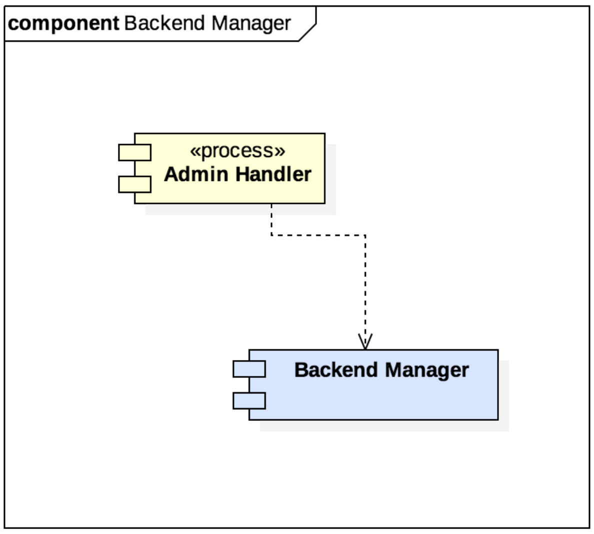
Notes: invoice Manager is considered to be the most critical for its responsibility to interact with the external *Payment Services Provider*. The two payment handlers are considered then, and the standard one is privileged because of its importance for the core functioning of the system. Finally the Discount and Sanction Handler is considered to require less effort and is associated to an inferior level of criticality.


component Payment Manager
«process» Invoice Manager
«process» Exceptional Payment Handler
«process» Payment Handler
«process» Discount and Sanctions Handler

Backend Manager

Internal integration strategy: bottom-up.
Integration order:

– Backend Manager
– Admin Handler



2.4.2 Subsystem integration sequence

The integration of the macro-components is performed, as said, in a bottom-up fashion. Below are the 5 steps in which this process has been divided for scheduling reasons. Inside a single step, multiple macro-components participate in the integration, each one relying on the macro-components in the previous levels.

Step 0: Localization Manager
The localization manager been developed and unit tested is the starting point for the integration.

Localization Manager

Figure 3: Subsystem integration sequence: step 0.

Step 1: User Manager, Car Manager

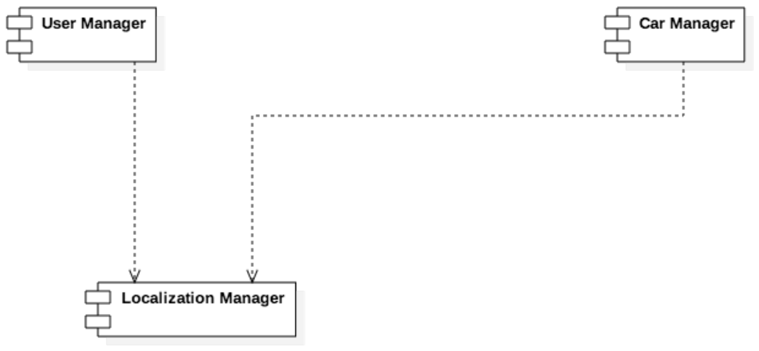


Figure 4: Subsystem integration sequence: step 1.

Step 2: Access Manager, Parking Manager

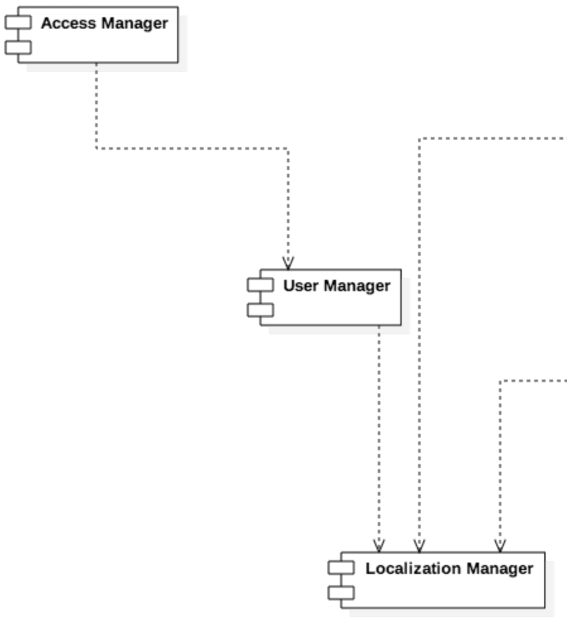


Figure 5: Subsystem integration sequence: step 2.

Step 3: Maintenance Manager, Car Employment Manager, Money Saving Option Manager

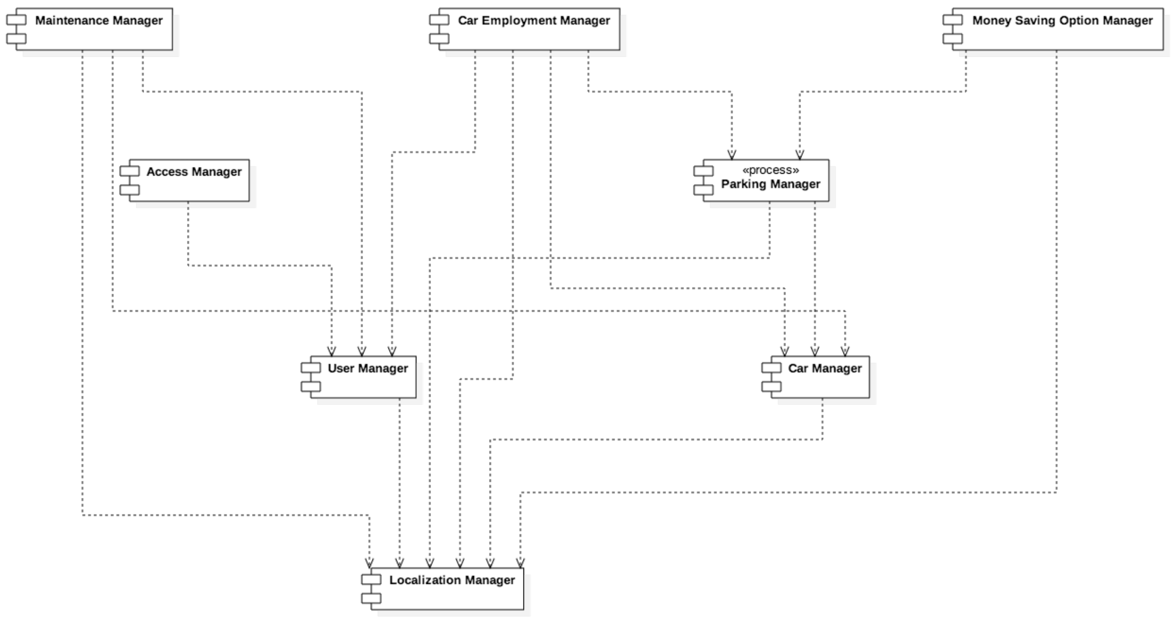


Figure 6: Subsystem integration sequence: step 3.

Step 4: Backend Manager, Payment Manager

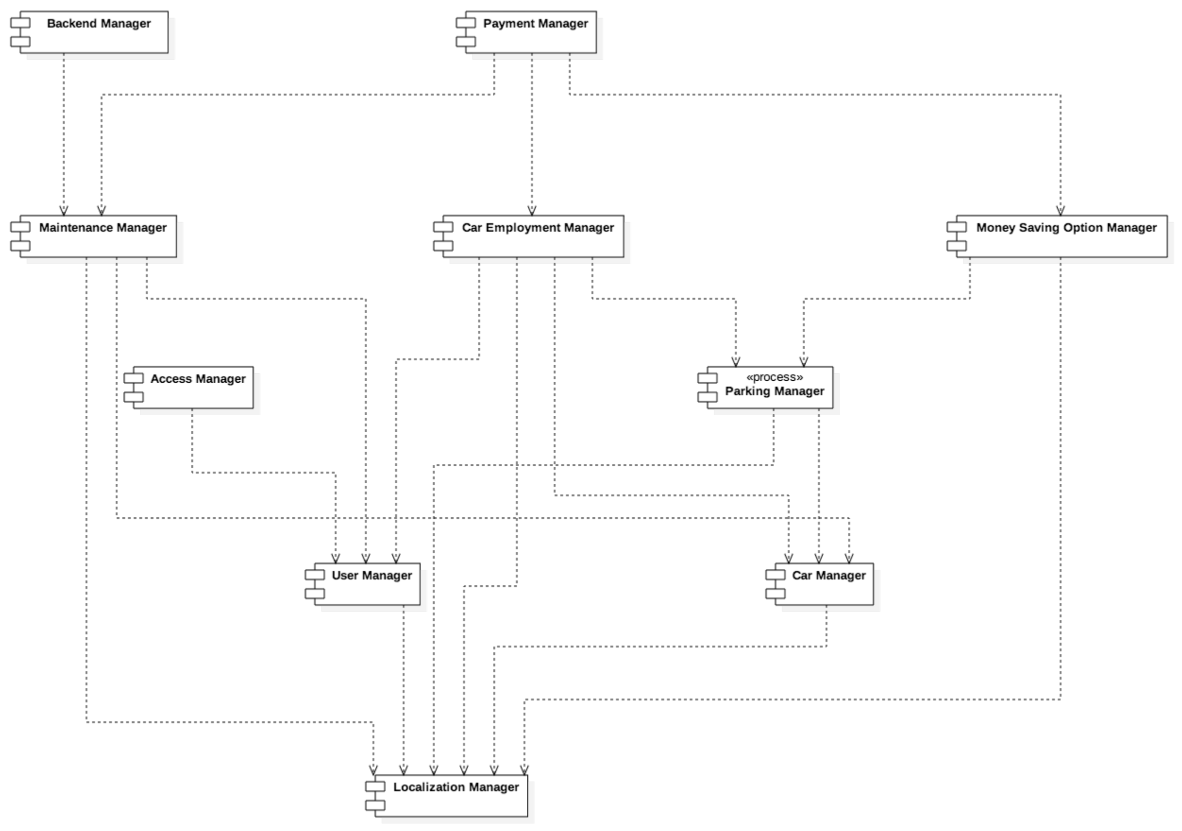
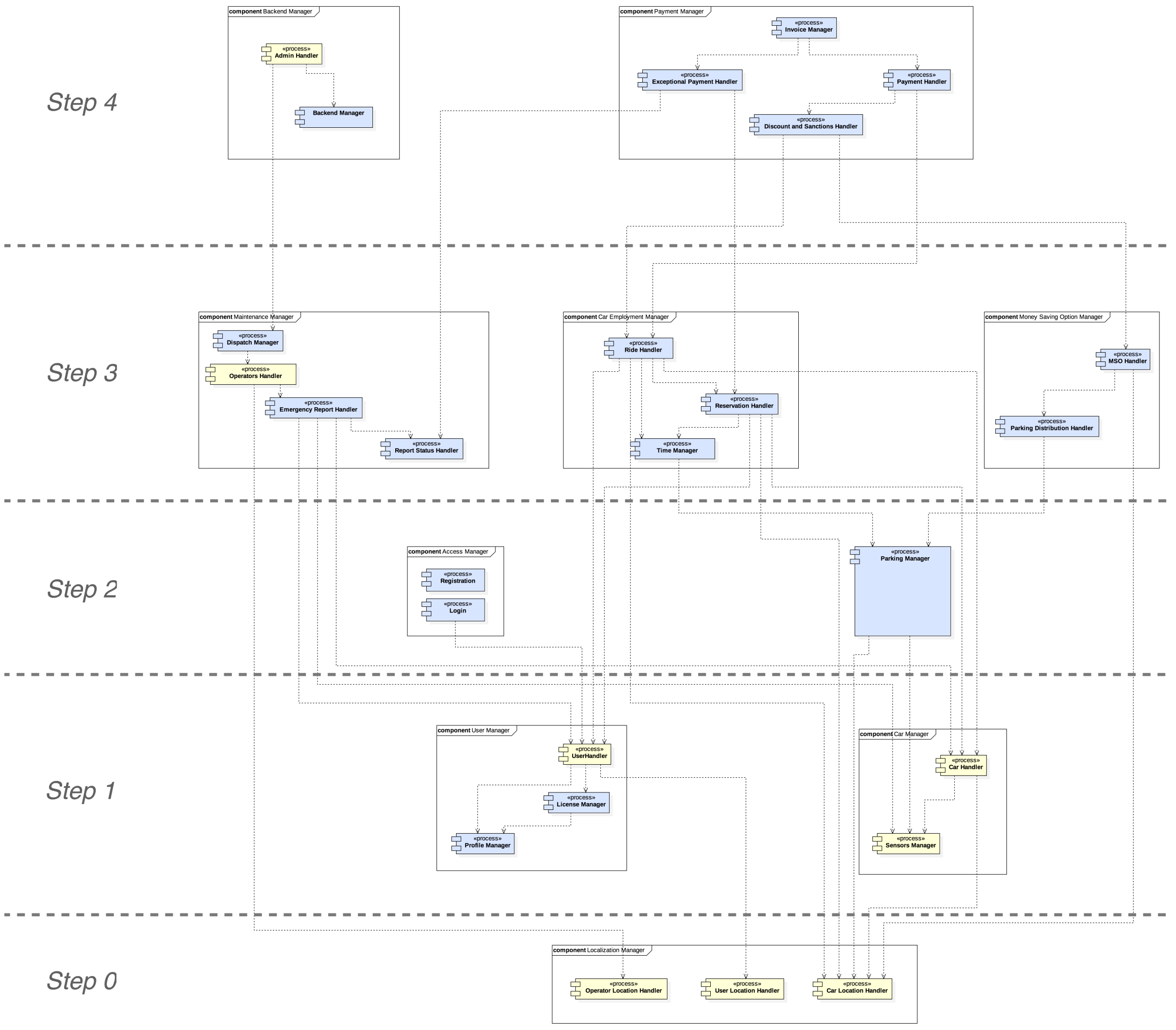


Figure 7: Subsystem integration sequence: step 4.

Complete system

The complete system integration is summarized in the following image, displaying also the internal structure of each macro-component.



Step 4
Step 3
Step 2
Step 1
Step 0
component Backend Manager
«process» Admin Handler
Backend Manager
component Payment Manager
«process» Invoice Manager
«process» Exceptional Payment Handler
«process» Payment Handler
«process» Discount and Sanctions Handler
component Maintenance Manager
«process» Dispatch Manager
«process» Operators Handler
«process» Emergency Report Handler
«process» Report Status Handler
component Car Employment Manager
«process» Ride Handler
«process» Reservation Handler
«process» Time Manager
component Money Saving Option Manager
«process» MSO Handler
«process» Parking Distribution Handler
component Access Manager
«process» Registration
«process» Login
«process» Parking Manager
component User Manager
«process» UserHandler
«process» License Manager
«process» Profile Manager
component Car Manager
«process» Car Handler
«process» Sensors Manager
component Localization Manager
«process» Operator Location Handler
«process» User Location Handler
«process» Car Location Handler

Figure 8: Complete system and integration steps.

3 Individual steps and test description

This section contains a detailed description of the tests to be performed on each pair of components that need to be integrated. Those pairs will be identified by a $< caller, called >$ coupled notation that will contain every method called by the $< caller >$ on the $< called >$.

3.0.1 User Manager

User Handler, Profile Manager

CheckData(user: Credentials):bool	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Right credentials	True
Wrong credentials	False

ChangeData(user: Credentials, changedData: Credentials)	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Old credentials and New credentials are the same	An IllegalArgumentException is raised
New credentials are not valid	An IllegalArgumentException is raised
New credentials are valid	User's credentials are changed

User Handler, License Manager

CheckLicenseValidity(user: Credentials): bool	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Valid driving license	True
Not valid driving license	False

User Handler, User Location Handler

SelectZone(location: Location, radius: float)	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Zone selected	Boundaries are defined

LocateUser(genericUser: Credentials):Location	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Valid parameter	Information about User's location

LocateAddress(address: String):Location	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Valid parameter	Location of the entered address

Emergency Report Handler, User Handler

Notify(command: Command)	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Valid parameter	Notification is recieved

Reservation Handler, User Handler

ReserveCar(car: Car)	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Valid parameter	Car is picked for reservation process

Ride Handler, User Handler

PickupCar(car: Car)	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Valid parameter	The *pickup process* is started

License Manager, Profile Manager

CheckData(user: Credentials):bool	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Right credentials	True
Wrong credentials	False

ChangeData(user: Credentials, changedData: Credentials)	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Old credentials and New credentials are the same	An IllegalArgumentException is raised
New credentials are not valid	An IllegalArgumentException is raised
New credentials are valid	User's credentials are changed

3.0.2 Car Manager

Reservation Handler, Car Handler

CheckCar():Status	
Input	*Effect*
Nothing	Information about car status

Ride Handler, Car Handler

ChangeState(state: State)	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Valid parameter	The car's state is changed

NotifyEmergency(signal: Stream)	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Valid parameter	Notification about the emergency

Car Handler, Sensors Manager

LockCar()	
Input	*Effect*
Nothing	Car is locked

UnlockCar()	
Input	*Effect*
Nothing	Car is unlocked

NotifyIgnition()	
Input	*Effect*
Nothing	Notification is recieved

PassengersDetection()	
Input	*Effect*
Nothing	Number of detected passengers

Car Handler, Emergency Report Handler

NotifyEmergency(signal: Stream)	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Valid parameter	Notification about the emergency

Ride, Car Location Handler

IsCarOutOfBoundaries(location: Location):bool	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Car's location is inside the selected zone	True
Car's location is outside the selected zone	False

Reservation Handler, Car Location Handler

LocateSelectedCar(car: Car): Location	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Valid parameter	Location of the selected car

Parking Manager, Car Location Handler

FindAreas(location: Location, radius: float): ParkingArea	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Valid input	The parking area near the location

MSO Handler, Car Location Handler

FindAreas(location: Location, radius: float): ParkingArea	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Valid input	The parking area neat the location where the user should park

3.0.3 Employment

Time Manager, Parking Manager

IsPluggingAllowed(timewindow: Time, curtime: Time, car: Car): bool	
Input	*Effect*
One or more null parameters	A NullArgumentException is raised
$curtime > timewindow$	False
$curtime <= timewindow$	True

IsParked(car: Car, area: ParkingArea): bool	
Input	*Effect*
One or more null parameters	A NullArgumentException is raised
Car is parked in that safe area	True
Car is not parked in that safe area	False

Ride Handler, Time Manager

Duration(ride: Ride): Time	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Valid parameter	The time duration of the ride at the time of the request

Reservation Handler, Time Manager

SetExpirationTime(time: Time, reservation: Reservation)	
Input	*Effect*
One or more null parameters	A NullArgumentException is raised
The reservation already has an existing expiration time 0	A IllegalArgumentException is raised
The reservation doesn't have an expiration time	The expiration time of the reservation is set as *time*

CheckExpirationTime(reservation: Reservation): bool	
Input	*Effect*
Null parameter	A NullArgumentException is raised
The expiration time for the reservation is still running	True
The expiration time for the reservation has finished	False

Ride Handler, Reservation Handler

CheckReservation(): bool	
Input	*Effect*
The reservation is still standing	True
The reservation is expired	False

Discount and Sanctions Handler, Ride Handler

GetRideInfo(): Ride	
Input	*Effect*
Nothing	All the information about the ride

Payment Handler, Ride Handler

GetRideInfo(): Ride	
Input	*Effect*
Nothing	All the information about the ride

Parking Distribution Handler, Parking Manager

GetCars(parkingArea: ParkingArea): List$< Car >$	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Valid input	The list of cars parked in parkingArea

ExceptionalPaymentHandler, ReservationHandler

CheckReservation(reservation: Reservation): bool	
Input	*Effect*
The reservation is still standing	True
The reservation is expired	False

3.0.4 Money Saving Option

MSO Handler, Parking Distribution Handler

GetAreas(destination: Location, radius: float): List$< ParkingArea >$	
Input	*Effect*
One or more null parameters	A NullArgumentException is raised
$radius \leq 0$	A IllegalArgumentException is raised
Destination outside city boundaries	A IllegalArgumentException is raised
Valid input	The list of parking areas in a radius around the destination

Discount and Sanctions Handler, MSO Handler

CalculateSolution(destination: Location): ParkingArea	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Destination outside city boundaries	A IllegalArgumentException is raised
Valid input	The parking area where the user should park

3.0.5 Backend

Operators Handler, Operator Location Handler

LocateUser(genericUser: Credentials): Location	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Given credentials belong to an Admin or a User	A IllegalArgumentException is raised
Valid credentials	The location of the Operator

Dispatch Manager, Operators Handler

FindOperator(location: Location, radius: float): List< *Operator* >	
Input	*Effect*
Null location parameter	A NullArgumentException is raised
$radius \leq 0$	A IllegalArgumentException is raised
Valid parameters	A list of operators in a radius around the location

CheckAvailability(operator: Credentials): bool	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Given credentials belong to an Admin or a User	A IllegalArgumentException is raised
Available operator	True
Unavailable operator	False

Operators Handler, Emergency Report Handler

CheckStatus(report: Report): Status	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Valid parameter	The complete status information of the report

UpdateRecord(report: Report, newStatus: Status)	
Input	*Effect*
One or more null parameters	A NullArgumentException is raised
Valid parameters	The status of the report gets updated

Emergency Report Handler, Report Status Handler

CloseReport(report: Report, operator: Credentials)	
Input	*Effect*
One or more null parameters	A NullArgumentException is raised
Given credentials belong to an Admin or a User	A IllegalArgumentException is raised
The operator is not assigned to the record	A IllegalArgumentException is raised
Valid parameters	The reports becomes *closed*

Exceptional Payment Handler, Report Status Handler

CheckStatus(report: Report): Status	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Valid parameter	The complete status information of the report

Admin Handler, Dispatch Manager

AssignOperator(report: Report, operator: Credentials	
Input	*Effect*
One or more null parameters	A NullArgumentException is raised
Valid parameters	The operator is assigned to the report

AdminHandler, Backend Manager

ChangeParameter(paramToChange: Parameter, newValue: Parameter)	
Input	*Effect*
One or more null parameters	A NullArgumentException is raised
Valid parameters	The new value is assigned to the parameter

CheckParameter(paramToChange: Parameter, newValue: Parameter): bool	
Input	*Effect*
One or more null parameters	A NullArgumentException is raised
The new value is within the acceptable range for the parameter	True
The new value is not within the acceptable range for the parameter	False

3.0.6 Authentication

Login, User Handler

Notify()	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Valid parameter	A notification is sent

3.0.7 Payment

Payment Handler, Discounts and Sanctions Handler

ApplyDiscounts(discount: float, payment: Payment)	
Input	*Effect*
One or more null parameters	A NullArgumentException is raised
Valid parameters	Discounts are applied to the payment

Invoice Manager, Payment Handler

Invoice(paymentService: External Interface, payment: Payment, creditCardInfo: Credentials):bool	
Input	*Effect*
One or more null parameters	A NullArgumentException is raised
The user has credit	True
The user doesn't have credit	False

Invoice Manager, Exceptional Payment Handler

AssignFineForReport(fine: float, report: Report)	
Input	*Effect*
One or more null parameters	A NullArgumentException is raised
Valid parameters	Fine is assigned

AssignFineForReservation(fine: float, reservation: Reservation)	
Input	*Effect*
One or more null parameters	A NullArgumentException is raised
Valid parameters	Fine is assigned

Emergency Report Handler, Report Status Handler

UpdateRecord(report: Report, newStatus: Status)	
Input	*Effect*
One or more null parameters	A NullArgumentException is raised
Valid parameters	Status of the report is upadated

CheckStatus(report: Report): Status	
Input	*Effect*
Null parameter	A NullArgumentException is raised
Valid parameter	The status information of the report

4 Tools and test equipment required

4.1 Tools

The testing activities will be performed with the help of various tools, each one designed for a particular task or application domain. In this section, a summary of their capabilities and the ways they are planned to be used is presented. Since they usually are designed to work on a specific platform and/or technology, a distinction will be made on this parameter.

Application Server

Technology used JEE
Testing tools

Mockito A framework for creating mocks object, useful in integration testing (besides basic unit testing). The mocks can be used to efficiently define stubs to support the scaffolding and test if a component interacts properly with the interfaces on which it depends. In particular, we will use it in the testing of the sub-components integrated with critical-first approach, as exposed in subsubsection 5.1.1.
JUnit Besides for unit testing, JUnit can be exploited to create drivers not only to test a single component, but also it's integration with the underlying portion of the system. It will be used in fact for every driver specified in subsubsection 5.1.2.
Arquillian An integration testing framework to execute test cases against the containers of a JEE application. It will be used to verify the correctness of the dependency injections of the components, as well as the injection of the database resources.
JMeter A tool to perform performance analysis on the interconnection of a system's components. In particular, it will be exploited to evaluate the non-functional requirements specified in *section 3.2* of the RASD.

Mobile apps - Android, Car app

Technology used Java for Android
Testing tools

Android Testing Support Library A framework that provides a set of APIs for testing Android application, including JUnit 4 and UI tests. It will be used in particular to test the integration of the logic components residing on the client. It should be noted that for all the applications, the components physically deployed on the clients are a small set, since the main application logic is located on the server. For this reason, here the integration testing will not be particularly demanding.
Android Device Monitor, Android Studio tools, SDK tools The performance test will be executed using the set of tools provided by Android Studio and the Android SDK, as the Android Device Monitor.
Manual testing The view of the Android mobile apps will be verified with manual testing. In fact, as explained in subsection 2.2, the testing will require only

simple operations and would not justify the overhead introduced by complicated UI analysis tools.

Mobile apps - iOS

Technology used Objective-C.
Testing tools

Xcode IDE, XCTest, OCmock The development IDE provides also analysis tools which will be exploited to test the application and analyze its performances. In addition, external tools such as XCTest and OCmock will be used.
Manual testing As for the *Android mobile app*, the view will be manual tested.

Mobile apps - Windows Phone

Technology used C#.
Testing tools

Visual Studio tools, WindowsPhoneTestFramework, other external tools The integration testing and performance evaluation will be conducted with the set of tools provided by Microsoft, as well as some external tools such as WindowsPhoneTestFramework.
Manual testing As for the *Android mobile app*, the view will be manual tested.

Admin application

Technology used JEE.
Testing tools

JUnit As for the *application server*, Junit will be used to setup the drivers mentioned in subsubsection 5.1.2.
Arquillian As for the *application server*, Arquillian will be used to verify the the dependency injections system.
Manual testing As for the *Android mobile app*, the view of the Admin application will be verified with manual testing, since the functionalities it provides are very limited.

4.2 Test equipment

In order to perform the actual tests on an environment similar to the production one, a proper equipment is required. Here is a list of what we plan to use in the integration phase and, more importantly, in the system testing phase.

Mobile apps

The most frequent cause of issues in mobile application is the screen size and density. Therefore, the app must be verified on multiple devices. In particular:

- **User mobile app** For its criticity, it must be tested on at least one device per screen size: 4.5", 5", 6". If possible, different screen densities must be verified as well.
- **Operator mobile app** Less effort can be allocated here, testing at least on a 5" smartphone and on a 10" tablet, since this app is for internal use only.
- **Car client** Since the cars embed only one type of device, the test should be performed mainly on it. Anyway, from time to time the application should be verified to work with other similar tablets as well, to be prepared for an eventual change of the embedded devices (i.e. for economical reasons, deterioration, etc.).

To overcome eventual issues related to specific vendors' devices or hardware components, the applications that make use of Bluetooth or GPS must be tested on devices of different brands (in particular for Android).

Admin application

The desktop application can be tested on a desktop similar to the ones used by the administrators. Even if JEE is mostly platform-independent, it is mandatory that the testing is performed on machines running the same OS, to ensure complete compatibility.

App server

Depending on the actual environment on which the app server will run, different testing options are available. In particular, cloud services companies usually provide tools and consoles to manage this aspect. In any case, we require that a proper testing environment is set up and made accessible to the developers team to constantly test the application-to-be. The test environment must present the same main characteristics of the production environment (same OS, same JEE implementation, etc.), but may have a more limited amount of resources allocated.

5 Program stubs and test data required

5.1 Program Stubs and Drivers

As we mentioned in subsection 2.3, we are going to use a bottom-up approach to integrate and test all macro-components of the system. This means that on the one hand we need a series of drivers to perform the method invocations while the higher-level components still haven't been developed. On the other hand, while it's true that in general we don't need stubs since the lower-level components will have been developed before testing the following level, in subsection 2.3 we specified that three macro-components needed a critical-first approach: these are the **Money Saving Option Manager**, the **Car Manager** and the **Payment Manager**. In the developing of these components we will start by integrating the most critical subcomponent and then the remaining ones. For these reason, the **Car Manager** integration and the **Money Saving Option Manager** integration will require drivers (since their critical component is the most low-level one); on the contrary, the **Payment Manager** will require the definition of stubs to simulate the potential invoked functions. For the specifics of each method invocation performed by the drivers, please refer to section 3.

5.1.1 Stubs

For the **Invoice Manager** integration we will need some program stubs:

- **Payment Handler Stub**: this testing module will emulate the invoked functions belonging to the **Payment Handler** subcomponent.
- **Exceptional Payment Handler stub**: this testing module will emulate the invoked functions belonging to the **Exceptional Payment Handler** subcomponent.

5.1.2 Drivers

Here follows a list of drivers needed by the system to perform testing according to our strategy. Following the bottom-up approach, drivers for subcomponents will be provided when needed as well as drivers emulating macro-components.

Localisation Manager

- **Localization Driver**: this testing module will invoke the methods exposed by the **Localization Manager** component.

User Manager

- **User Handler Driver**: this testing module will invoke the methods exposed by the **User Handler** subcomponent.
- **License Manager Driver**: this testing module will invoke the methods exposed by the **License Manager** subcomponent.

- **Profile Manager Driver**: this testing module will invoke the methods exposed by the **Profile Manager** subcomponent.
- **User Manager Driver**: this testing module will invoke the methods exposed by the **User Manager** component.

Car Manager

- **Car Handler Driver**: this testing module will invoke the methods exposed by the **Car Handler** subcomponent.
- **Car Manager Driver**: this testing module will invoke the methods exposed by the **Car Manager** component.

Parking Manager

- **Parking Manager Driver**: this testing module will invoke the methods exposed by the **Parking Manager** component.

Access Manager

- **Registration Driver**: this testing module will invoke the methods exposed by the **Registration** subcomponent.
- **Login Driver**: this testing module will invoke the methods exposed by the **Login** subcomponent.
- **Access Manager Driver**: this testing module will invoke the methods exposed by the **Access Manager** component.

MSO Manager

- **Parking Distribution Driver**: this testing module will invoke the methods exposed by the **Parking Distribution Handler** subcomponent.
- **MSO Driver**: this testing module will invoke the methods exposed by the **MSO Manager** component.

Car Employment Manager

- **Time Manager Driver**: this testing module will invoke the methods exposed by the **Time Manager** subcomponent.
- **Reservation Driver**: this testing module will invoke the methods exposed by the **Reservation Handler** subcomponent.
- **Employment Driver**: this testing module will invoke the methods exposed by the **Car Employment Manager** component.

Maintenance Manager

- **Report Status Driver**: this testing module will invoke the methods exposed by the **Report Status Handler** subcomponent.
- **Report Driver**: this testing module will invoke the methods exposed by the **Emergency Report Handler** subcomponent.
- **Operators Driver**: this testing module will invoke the methods exposed by the **Operators Handler** subcomponent.
- **Maintenance Driver**: this testing module will invoke the methods exposed by the **Maintenance Manager** component.

BackEnd Manager

- **Backend Driver**: this testing module will invoke the methods exposed by the **BackEnd Manager** subcomponent.

Payment Manager

- **Payment Handler Driver**: this testing module will invoke the methods exposed by the **Payment Handler** subcomponent.
- **Dicounts and Sanctions Handler Driver**: this testing module will invoke the methods exposed by the **Dicounts and Sanctions Handler** subcomponent.
- **Exceptional Handler Driver**: this testing module will invoke the methods exposed by the **Exceptional Payment Handler** subcomponent.
- **Invoice Manager Driver**: this testing module will invoke the methods exposed by the **Invoice Manager** subcomponent.
- **Payment Manager Driver**: this testing module will invoke the methods exposed by the **Payment Manager** component.

5.2 Test Data

In order to perform the tests specified in section 3 we will need a range of test data, both valid and invalid. This chapter will list those data in reference to the specific set of macro-components that will need them.

5.2.1 User Manager

- A list of valid and invalid **User's credentials**. The set should contain instances exhibiting the following cases:
  - Null objects

  - Null fields
  - Wrong credentials
  - Not valid driving license
  - Credentials are not registered

- A list of valid and invalid **User's locations**. The set should contain instances exhibiting the following cases:
  - Null objects
  - Null fields
  - Precise user's location is outside the selected zone
  - Entered address is outside the selected zone

5.2.2 Car Manager

- A list of valid and invalid **cars**. The set should contain instances exhibiting the following cases:
  - Null objects
  - Null fields
  - Selected car is employed by another user
  - Selected car is outside the city boundaries
- A list of valid and invalid **car's location**. The set should contain instances exhibiting the following cases:
  - Null objects
  - Null fields
  - Car's locatiosn is outside the city boundaries

5.2.3 Authentication

- A list of valid and invalid **candidate users** to test Registration component . The set should contain instances exhibiting the following cases:
  - Null objects
  - Null fields
  - Credentials are already registered
  - Invalid values filled in the form

5.2.4 Payment

- A list of valid and invalid **Payments**. The set should contain instances exhibiting the following cases:
    - Null objects
    - Null fields
    - Discount is applied to the payment that was not assigned

5.2.5 Employment

- A list of valid and invalid **Reservations**. The set should contain instances exhibiting the following cases:
    - Null object
    - Null fields
    - Reservation expired
    - Starting time for the reservation set in the future
    - Reservation on a car that is being used
    - Reservation on a nonexistent car
- A list of valid and invalid **Rides**. The set should contain instances exhibiting the following cases:
    - Null object
    - Null fields
    - Ride that has already ended
    - Ride on a car that is already being used
    - Ride on a nonexistent car
    - Ride associated with a user already using another car
- A list of valid and invalid **ParkingAreas**. The set should contain instances exhibiting the following cases:
    - Null object
    - Null fields
    - ParkingAreas containing a car that is located somewhere else
    - ParkingArea outside city boundaries
    - ParkingArea with more parked cars than slots
    - Full ParkingArea
    - Empty ParkingArea

5.2.6 Money Saving Option

- A list of valid and invalid **destinations**. The set should contain instances exhibiting the following cases:
  - Null object
  - Null fields
  - Destination is outside city boundaries
  - Destination has no parking areas within the radius requested by the MSO
  - Destination doesn't exist
  - Destination is the car's current location

5.2.7 Backend

- A list of valid and invalid **Parameters**. The set should contain instances exhibiting the following cases:
  - Null object
  - Null fields
  - Parameter value outside allowed range
- A list of valid and invalid **Emergency Reports**. The set should contain instances exhibiting the following cases:
  - Null object
  - Null fields
  - Report closed
  - Report without User assigned
  - Report without Car assigned
  - Report without Operator assigned
  - Report with an available Operator assigned

6 Effort spent

Abbud, Patricia around 26 hours of work;

Andreoli Andreoni, Maddalena around 30 hours of work;

Cudrano, Paolo around 34 hours of work.

7 Appendix

List of Figures