POLITECNICO
MILANO 1863

Integration Testing Plan Document

Joan Ficapal Vila (876805), Nicolò Vendramin (879113)

January 18, 2017
v1.0

Revision History

Revision	Date	Author(s)	Description
0.1	01.01.17	N and J	Initiated the document
0.2	04.01.17	N and J	Wrote the introduction
0.3	08.01.17	N and J	Defined the Integration Strategy
0.4	10.01.17	N	Graphs of the Second Section
0.5	11.01.17	N and J	Started working on the tables
0.6	12.01.17	J	Added tests to the Tables
0.7	13.01.17	N and J	Added tests to the Tables
0.8	14.01.17	N	Wrote the final Section
1.0	15.01.17	N and J	First Release

Hours of work

- Joan Ficapal Vila : 10 hours
- Nicolò Vendramin : 10 hours

Contents

1 Introduction

1.1 Purpose and Scope

This document is the Integration Testing Plan Document for the PowerEnjoy application.
It is produced in order to specify in detail how the testing process must be performed in order to guarantee a correct and deep analysis of the application's behaviour.
Integration testing must be considered a key activity in order to ensure the correct functioning of the cohoperation between all the subsystems composing the software to be developed. As a matter of fact the unit testing itself is not sufficient to make sure that the overall system in its complex works as established.

To outline in a clear way the main aspects concerning this part of the testing activity we are going to include in this document, in a precise but concise way, the following informations:

- Description of the components that must go through integration testing
- The integration strategy to be followed during the testing activity
- The precedences to be followed during the process
- A description of the testing activities including the input and the expected effect for the more relevant parts
- An outline of the tools and equipments needed.

1.2 Definitions, Acronyms and Abbreviations

1.2.1 Acronyms

- **API:** Access point of Interface. This term is used to indicate the interfaces exposed to access a software service from another software service.
- **DD:** Design Document. This document.
- **RASD:** Requirements Analysis and Specifications Document. The reference document for the specifications of the system to be developed.
- **DAG:** Direct Acyclic Graph. A DAG is a graph made up of nodes and directed archs in which no cycles are present.

1.2.2 Definitions

- **Unit Testing:** With unit testing is meant that activity that make sure that every basic unit that composes the software is correctly working as expected. With unit we indicate the smaller atom of which the software is composed. In case of a Java program we can assume Classes as units.
- **Integration Testing:** With integration testing is meant that phase of software testing in which individual modules are tested together as a group.

1.3 Reference Documents

Here we attached the list of all the documents that are kept as references in the writing of this document.

- PowerEnjoy RASD Document.
- PowerEnjoy Design Document.
- Course Slides.
- Templates provided at lesson.

2 Integration Strategy

2.1 Entry Criteria

In order to ensure that the testing process is completed with good results some conditions must to be met. At first is required that both the RASD and the DD are completed and defined. This condition must be guaranteed to have a complete and detailed description of the system requirements and design on which to base the testing activities.
In addition before to start the very integration process it is required to pass the single components of the system with unit testing. It's important that before integrating two components each of them have passed all the unit tests that were designed for them. We assume that for the components included in the request manager and controller the percentage of coverage reached before starting their integration is between 95% and 100%. For what the Client Application is concerned, since it's a graphical interface, the unit testing needed before the integration should be at least 70%; making sure that the only non tested components are the parts related to the very presentation and not the ones related to the data exchange process.
An additional condition that must be met before starting the integration testing activity is that the functioning of all the API has been checked to be as expected from the documentation. In order to meet this hypothesis a detailed checking activity must be done in order to verify the behaviour of the APIs provided by the external services used. in the software.

2.2 Elements to be integrated

Since a precise description of all the components is given in the DD we invite the reader to refer to that document in case of need.
The integration testing is made in order to verify the joint operation of the following group of components.
As stated before we assume that all the components belonging to the following list, before being integrated satisfy the unit level coverage specified in the previous paragraph. The unit to integrate are:

- Mobile Client
- Request Manager
- Controller
    - Registration and Login Manager
    - Pricing and Discount Manager
    - Service Manager
    - Resource Manager

2.3 Integration Testing Strategy

To design the strategy to be used for the integration testing we took into account different factors among which the more importants are the architectural structure of the software described in the DD and the willing to optimize the work during the testing process.
The best solution identified to fit the above mentioned needs is to adopt a Bottom-Up and Core-module-first approach. In practical terms this means that at first we are going to integrate the low level components between them and only after that we will test the communication between

the high level components. The core-module-first approach means that we will give a major priority to the integration of those components that form the controller system in order to be sure that all the core functionalities of the application are provided in the correct way and that the controller works as an integrated system.
At the end of the integration testing we allocate some human resources to the further verification through usage of the system. For this final verification activity we don't give precise indication and we invite the dedicated staff to follow the possible use cases outlined in the DD, and to repeatedly test them in different environmental conditions, and following the different possible branches described in the use cases' tables making sure to cover, as much as possible, both exceptional and non-exceptional behaviours of the system.

2.4 Sequence of component/ function integration

2.4.1 Subsystem 1

- I1. Resource Manager → DBMS
- I2. Registration / Login Manager → Existing System
- I3. Registration / Login Manager → Resource Manager
- I4. Service Manager → Existing System
- I5. Service Manager → Resource Manager
- I6. Service Manager → Registration / Login Manager
- I7. Price and Discount Manager → Billing System
- I8. Price and Discount Manager → Resource Manager
- I9. Service Manager → Price and Discount Manager

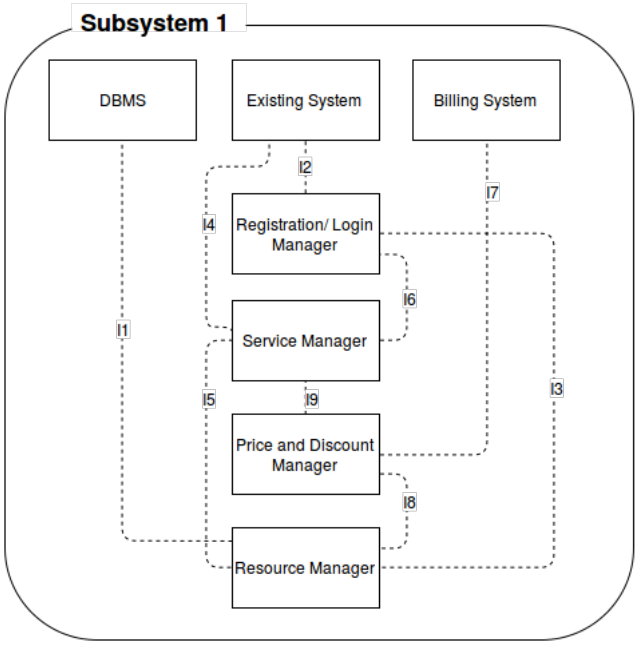

Figure 1: Subsystem 1, Integration Strategy

2.4.2 Subsystem 2

- I10. BookVehicle, CancelBooking, OpenVehicle, Conclude → Service Manager
- I11. Login, Logout, Register → Registration / Log in Manager
- I12. ShowVehicles, ShowCarDetails, ShowAccountInfo → Resource Manager

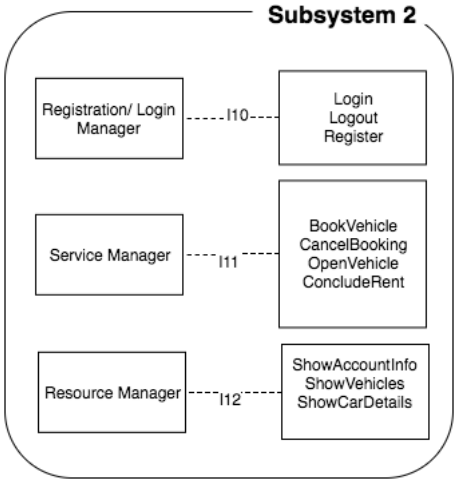

Figure 2: Subsystem 2, Integration Strategy

2.4.3 Subsystem 3

- I12. Mobile Client → Request Manager

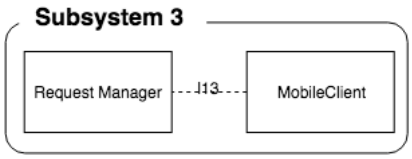

Figure 3: Subsystem 3, Integration Strategy

2.5 Further Conisderation On The Sequence of Integration

Before concluding the section about the integration strategy we attach the direct acyclic graph of the dependencies between the different integrations steps. This is provided for two basics reasons:

- to provide a further check that all the components are integrated in a correct sequence
- if in case of need the integration strategy cannot be followed as above, or it is more efficient to change it, the DAG must be kept as a reference and the new strategy should ensure that all the dependencies are managed as specified by the graph.

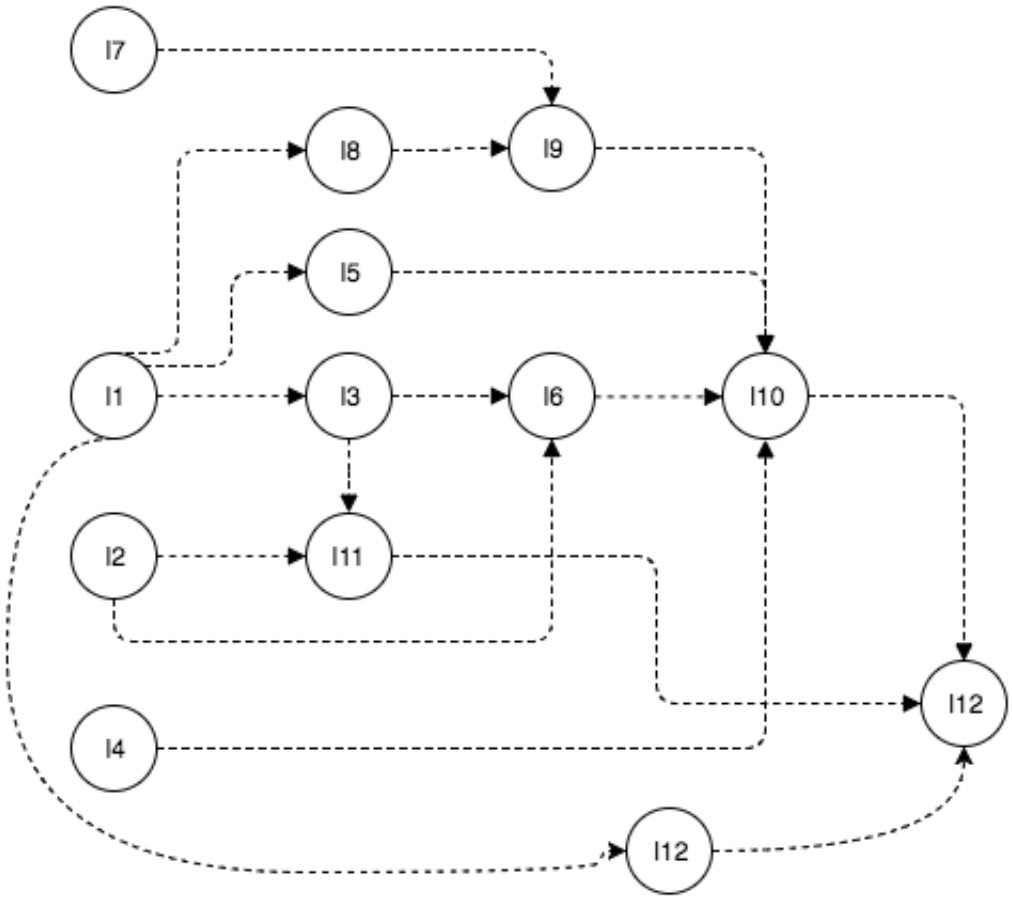

Figure 4: Direct Acyclic Graph of Integration Dependencies

3 Individual Steps and Test Description

Subsystem 1

Resource Manager, DBMS

	update_information(client, car, area, ride, booking, bill)
Input	Effect
An empty array	A NullArgumentException is raised
A non null client not correctly formatted	An InvalidClientFormatException is raised.
A non null car not correctly formatted	An InvalidCarFormatException is raised.
A non null area not correctly formatted	An InvalidAreaFormatException is raised.
A non null ride not correctly formatted	An InvalidRideFormatException is raised.
A non null booking not correctly formatted	An InvalidBookingFormatException is raised.
A non null bill not correctly formatted	An InvalidBillFormatException is raised.
A non empty array of formally valid arguments	The database is updated only for the intersection of the non null parameters. Returns True.

	get_account_details(userid)
Input	Effect
A null parameter	A NullArgumentException is raised
A userid not correctly formatted	An InvalidArgumentFormatException is raised.
Inexistent userid	A InvalidArgumentValueException is raised
A non null formally valid clientid	Returns the object of the entered client
	find_available_cars(location)
A null parameter	A NullArgumentException is raised
An invalid location	An InvalidLocationException is raised.
Formally valid arguments	Returns an array containing the available cars for that position

	get_car_information(car_id)
Input	Effect
A null parameter	A NullArgumentException is raised
An invalid car_id	An InvalidCarException is raised.
A non existent car	An InvalidCarException is raised.
Formally valid arguments	Returns the car object with that car_id

	set_booking_timer(booking_id)
A null parameter	A NullArgumentException is raised
An invalid booking_id	An InvalidCarException is raised.
A non existent booking	An InvalidCarException is raised.
Formally valid arguments	The entry containing the booking_id initial time is set as the current time in the database.

	stop_booking_timer(booking_id)
Input	Effect
A null parameter	A NullArgumentException is raised
An invalid booking_id	An InvalidCarException is raised.

A non existent booking	An InvalidCarException is raised.
Formally valid arguments	The entry containing the booking_id final time is set as the current time in the database.

Registration / Login Manager, Existing System	

	request_registration(name, surname, email, birthday, password, driving_license_n, driving_license_expdate, card_n, card_sec_code)
Input	Effect
A null parameter	A NullArgumentException is raised
User already registered	A InvalidArgumentValueException is raised
Exception on the format of the data	A InvalidArgumentValueException is raised
Formally valid arguments	The user is registered on the existing system which automatically inserts an entry into the database. Returns True.

	request_login(user,password)
Input	Effect
A null parameter	A NullArgumentException is raised
Inexistent user	A InvalidArgumentValueException is raised
A null password	A InvalidArgumentValueException is raised
A valid user and password combination, which however is not the correct one	Returns an InvalidCredentialError.
Formally valid arguments	The existing system checks the session cookie in the database and returns it.

Registration / Login Manager, Resource Manager	

	register_cookie_session(user_id)
Input	Effect
A null parameter	A NullArgumentException is raised
A non null user id but not correctly formatted	An InvalidFormatException is raised
A well formatted user id but not present in the database	An UserNotKnownException is raised
A known user id	After being checked, from the user id is created a cookie session id and the new couple is saved in the database.

	identify_user(cookie_session)
Input	Effect
A null parameter	A NullArgumentException is raised
A non null cookie id but not correctly formatted	An InvalidFormatException is raised
A well formatted cookie id but not present in the database	An UserNotKnownException is raised
A well formatted and known cookie id	The user id of the user identified by that session id is returned.

	delete_cookie_session(cookie_session)
Input	Effect
A null parameter	A NullArgumentException is raised
A non null cookie id but not correctly formatted	An InvalidFormatException is raised
A well formatted cookie id but not present in the database	An UserNotKnownException is raised

A well formatted and known cookie id	The given cookie session is deleted from the database. Meaning that the session is concluded.

	get_account_info(userid)
Input	Effect
A null parameter	A NullArgumentException is raised.
Formally valid argument	The userid is used to call the get_account_details function in the DBMS through the Resource Manager. Returns the result of this function.

	find_available_cars(location)
Input	Effect
A null parameter	A NullArgumentException is raised.
Formally valid argument	The location is used to call the find_available_cars function in the DBMS. Returns the result of this function.

Service Manager, Existing System

	request_booking(userid,user_location,car)
Input	Effect
A null parameter	A NullArgumentException is raised.
A non existent car	An InvalidCarException is raised.
An already taken car	An InvalidCarException is raised.
A userid not correctly formatted	An InvalidArgumentFormatException is raised.
An invalid user_location	An InvalidLocationException is raised.
A user that has another active booking already	An InvalidUserException is raised.
A user_location which is outside the city	An InvalidLocationException is raised.
A valid set of parameters	The existing system books the car for that user and updates the database. Check RASD for information of specific outcomes of this operation. Returns True.

	cancel_booking(userid,car)
Input	Effect
A null parameter	A NullArgumentException is raised.
A non existent car	An InvalidCarException is raised.
An already taken car	An InvalidCarException is raised.
A userid not correctly formatted	An InvalidArgumentFormatException is raised.
A user without active booking	An InvalidUserException is raised.
A valid set of parameters	The existing system cancels the booking for that user and updates the database. Check RASD for information of specific outcomes of this operation. Returns True.

	request_open(userid, user_location, car, booking)
Input	Effect
A null parameter	A NullArgumentException is raised.
A userid not correctly formatted	An InvalidArgumentFormatException is raised.
An invalid user_location	An InvalidLocationException is raised.
A user_location far from the car	An InvalidLocationException is raised.
A non existent car	An InvalidCarException is raised.

An already taken car	An InvalidCarException is raised.
The booking doesn't correspond to that user or it has expired	An InvalidBookingException is raised.
A valid set of parameters	The existing system allows the user to open the car, changes the required parameters to indicate that the car is not longer booked but in a ride, and the same with the user. It also updates the database. Check RASD for information of specific outcomes of this operation. Returns True.

	request_rideend(userid, user_location, car, car_location, ride)
Input	Effect
A null parameter	A NullArgumentException is raised.
A userid not correctly formatted	An InvalidArgumentFormatException is raised.
An invalid user_location	An InvalidLocationException is raised.
The user_location is not the same as the car_location	An InvalidLocationException is raised.
The car_location doesn't correspond to a safe zone	An InvalidLocationException is raised.
A non existent car	An InvalidCarException is raised.
The ride is not associated with the car or the user	An InvalidRideException is raised.
A valid set of parameters	The existing system performs the ride termination and changes the required parameters to indicate that the car is not longer ocupated but iddle, and the same with the user. It also updates the database. Check RASD for information of specific outcomes of this operation. Returns True.

Service Manager, Resource Manager

	check_expired_bookings()
Input	Effect
—	The function must be called to check the list of bookings that have expired. Returns a list of booking ids.

Service Manager, Registration / Login Manager

	verify_identity(cookie_session, user_id)
Input	Effect
A null parameter	A NullArgumentException is raised.
A non null user id but not correctly formatted	An InvalidFormatException is raised
A well formatted user id but not present in the database	An UserNotKnownException is raised
A non null cookie id but not correctly formatted	An InvalidFormatException is raised
A well formatted cookie id but not present in the database	An UserNotKnownException is raised
A well formatted and valid couple of user id and cookie id	The login and registration manager is asked to check whether the identity is correct. In case the cookie session corresponds to the given user it returns True, otherwise False

Price and Discount Manager, Billing System

	charge_user(userid, ammount, name, surname, card_n, card_sec_code)
Input	Effect
A null parameter	A NullArgumentException is raised
A userid not correctly formatted	An InvalidArgumentFormatException is raised.
Negative ammount	A InvalidArgumentValueException is raised

Negative ammount	A InvalidArgumentValueException is raised
Either the card_n or the sec_code is not correctly formated	A InvalidArgumentValueException is raised
The account to charge can't be found	A InvalidCardException is raised
A non empty array of formally valid arguments	The user is charged. Returns True.

Price and Discount Manager, Resource Managers	

	check_car_is_plugged(car_id)
Input	Effect
A null parameter	A NullArgumentException is raised
A non null car id but not correctly formatted	An InvalidFormatException is raised
A well formatted car id but not present in the database	An CarNotKnownException is raised
A known car id	The Resource managers queries the existing system to check whether the given car is plugged or not. The result of that check is returned.

	get_ride_info(user_id)
Input	Effect
A null parameter	A NullArgumentException is raised
A non null user id but not correctly formatted	An InvalidFormatException is raised
A well formatted user id but not present in the database	An UserNotKnownException is raised
A known user id	The data base is queried and the ride active for that user in that moment is returned. In case no ride is active for that user an InvalidArgumentException is raised.

	check_position_is_safe_area(coordinate)
Input	Effect
A null parameter	A NullArgumentException is raised
A non null coordinate but not correctly formatted	An InvalidFormatException is raised
A well formatted coordinate	The resource manager checks in the Database the list of all the safe areas, basing on the bounds of the safe area establishes whether the point belongs or not to the safe area. The method should return true in case the car is in a safe area false otherwise.

Service Manager, Price and Discount Manager	

	charge_booking_expiration(user_id)
Input	Effect
A null parameter	A NullArgumentException is raised
A non null user id but not correctly formatted	An InvalidFormatException is raised
A well formatted user id but not present in the database	An UserNotKnownException is raised
A known user id	The Price and Discount Manager checks the price for the expired booking, retrieves all the data need for the payment and calls the function to charge the user in the Billing System.

	charge_for_ride(user_id, ride)
Input	Effect
A null parameter	A NullArgumentException is raised

A non null user id but not correctly formatted	An InvalidFormatException is raised
A well formatted user id but not present in the database	An UserNotKnownException is raised
A well formatted user id but not present in the database	An InvalidArgumentException is raised
A known user id and Ride but that are not assoicieted	An InvalidArgumentException is raised
A known user id and ride associated one to the other	The pricing and discount manager computes the algorithm to find the price of the ride and encharges the billing system to

	charge_user(userid, ammount, name, surname, card_n, card_sec_code)
Input	Effect
A null parameter	A NullArgumentException is raised
Formally valid arguments	The parameters are used to call the charge_user function in the Billing System API

	compute_price(discount, price)
Input	Effect
A null parameter	A NullArgumentException is raised
Formally valid arguments	The discounts are applied to the price and it is returned.

Subsystem 2

Login - Logout - Register - ShowAccountInfo, Registration / Login Manager

	register(name, surname, email, birthday, password, driving_license_n, driving_license_expdate, card_n, card_sec_code)
Input	Effect
A null parameter	A NullArgumentException is raised
Formally valid arguments	The parameters are used to call the request_registration function in the existig system API. Returns the result of this function.

	login(user,password)
Input	Effect
A null parameter	A NullArgumentException is raised
Formally valid arguments	The parameters are used to call the request_login function in the existig system API. Returns the result of this function.

	register(name, surname, email, birthday, password, driving_license_n, driving_license_expdate, card_n, card_sec_code)
Input	Effect
A null parameter	A NullArgumentException is raised
Formally valid arguments	The parameters are used to call the request_registration function in the existig system API. Returns the result of this function.

BookVehicle - CancelBooking - OpenVehicle - ShowVehicle - ShowCarDetails - ConcludeRent, ServiceManager

	book(userid,user_location,car)
Input	Effect

A null parameter	A NullArgumentException is raised.
Formally valid arguments	The parameters are used to call the request_booking function in the existing system. Returns the result of this function.

	cancel_booking(userid,car)
Input	Effect
A null parameter	A NullArgumentException is raised.
Formally valid arguments	The parameters are used to call the cancel_booking function in the existing system. Returns the result of this function.

	open_car(userid, user_location, car, booking)
Input	Effect
A null parameter	A NullArgumentException is raised.
Formally valid arguments	The parameters are used to call the request_open function in the existing system. Returns the result of this function.

	end_ride(userid, user_location, car, car_location, ride)
Input	Effect
A null parameter	A NullArgumentException is raised.
Formally valid arguments	The parameters are used to call the request_rideend function in the existing system. Returns the result of this function.

	start_booking_timer(booking_id)
Input	Effect
A null parameter	A NullArgumentException is raised
Formally valid arguments	The booking is used to call set_booking_timer function in the DBMS interface.

	stop_booking_timer(booking_id)
Input	Effect
A null parameter	A NullArgumentException is raised
Formally valid arguments	The booking is used to call stop_booking_timer function in the DBMS interface.

	get_account_info(userid)
Input	Effect
A null parameter	A NullArgumentException is raised.
Formally valid argument	The userid is used to call the get_accountdetails function in the DBMS. Returns the result of this function.

	find_available_cars(location)
Input	Effect
A null parameter	A NullArgumentException is raised.
Formally valid argument	The location is used to call the find_available_cars function in the DBMS. Returns the result of this function.

	get_car_information(car_id)
Input	Effect
A null parameter	A NullArgumentException is raised.
Formally valid argument	The car_id is used to call the get_car_information function in the DBMS. Returns the result of this function.

	get_car_information(car_id)
Input	Effect
A null parameter	A NullArgumentException is raised.
Formally valid argument	The car_id is used to call the get_car_information function in the DBMS. Returns the result of this function.

Subsystem 3

Mobile Client, Request Manager

	show_notify_error(exception,cookie_session)
Input	Effect
A null parameter	A NullArgumentException is raised
An exception which is not between the following ones: NullArgumentException, InvalidClientFormatException, InvalidCarFormatException, InvalidAreaFormatException, InvalidRideFormatException, InvalidBookingFormatException, InvalidBillFormatException.	A InvalidArgumentValueException is raised
An exception which is between the following ones: NullArgumentException, InvalidClientFormatException, InvalidCarFormatException, InvalidAreaFormatException, InvalidRideFormatException, InvalidBookingFormatException, InvalidBillFormatException.	Parses into text the kind of exception that has been produced and sends them to the client mobile.

	submit_gps_coordinates(coordinates)
Input	Effect
A null parameter	A NullArgumentException is raised
A non null coordinate but not correctly formatted	An InvalidFormatException is raised
Valid coordinates	Sends the coordinates to to the Controller, where the show_vehicles function inside the ResourceManager is called. Returns the result of that function.

	ask_showAccountInfo(userid)
Input	Effect
A null parameter	A NullArgumentException is raised
Non null parameters	The information is sent to the Controller, where the get_account_info function inside the ResourceManager is called. Returns the output of that function.

	ask_booking(userid, car_id)
Input	Effect
A null parameter	A NullArgumentException is raised
Non null parameters	The information is sent to the Controller, where the book function inside the ServiceManager is called. Calls show_notify_ack if the operation was successfully completed or show_notify_error if it wasn't.

	ask_cancel_booking(userid)
Input	Effect
A null parameter	A NullArgumentException is raised
Non null parameters	The information is sent to the Controller, where the cancel_booking function inside the ServiceManager is called. Calls show_notify_ack if the operation was successfully completed or show_notify_error if it wasn't.

	ask_OpenVehicle(userid)
Input	Effect
A null parameter	A NullArgumentException is raised
Non null parameters	The information is sent to the Controller, where the open_vehicle function inside the ServiceManager is called. Calls show_notify_ack if the operation was successfully completed or show_notify_error if it wasn't.

	ask_Conclude(userid)
Input	Effect
A null parameter	A NullArgumentException is raised
Non null parameters	The information is sent to the Controller, where the end_ride function inside the ServiceManager is called. Calls show_notify_ack if the operation was successfully completed or show_notify_error if it wasn't.

	ask_showCarInfo(car_id)
Input	Effect
A null parameter	A NullArgumentException is raised
Non null parameters	The information is sent to the Controller, where the get_car_information function inside the ResourceManager is called. Returns the output of that function.

	submit_gps_coordinates(coordinates)
Input	Effect
Nothing	Sends the coordinates to to the Controller, where the show_vehicles function inside the ResourceManager is called. Returns the result of that function.

	submit_registration(registration_form)
Input	Effect
A null parameter	A NullArgumentException is raised
Non null parameters	The information is sent to the Controller, where the register function inside the Registration / Log in Manager is called. Calls show_notify_ack if the operation was successfully completed or show_notify_error if it wasn't.

	submit_login_info(login_form)
Input	Effect
A null parameter	A NullArgumentException is raised
Non null parameters	The information is sent to the Controller, where the login function inside the Registration / Log in Manager is called. Calls show_notify_ack if the operation was successfully completed and builds the new session with the cookies obtained. It calls show_notify_error if the operation was not successfull.

4 Tools and Required Equipments

In this section we are going to explain which are the tools that are required to perform the testing activities, why they were selected and how they should be used.

4.1 Testing Frameworks

Since the project is written in JEE we are going to exploit two powerfull tools that are designed for Java verification: JUnit and the Arquillion framework. The former, that is widely used in unit testing as well, offers the possibility to check that method's calls from one component to the other work in the right way, returning what expected and producing the desired effect. The latter, instead, is a tool that is specifically design to support the developer in the integration testing activity and it can be used in order to verify the correct functioning of the dependency injections and of the interactions of the components between themselves and with the database. This two components can be considered the main software equipments needed for the testing activity of the control-layer that we have described above. For what the customer application is concerned the tools that we are going to use in order to perform the testing activities are the ones that are included in the IDEs used for the application development (respectively XCode for iOs and Android Studio for Android).

4.2 Support Tools

In some cases the testing activity can be hard or it can fail not because of problems with the code but because of the impossibility to have control on some events (e.g.: network's problems). For this reason and in order to be sure that is possible to test also some conditions that appear only rarely we decided to employ Mockito. Mockito is a testing tool that can allow the developer to define the behaviour of an object and to use the mocked object instead of the real one while testing. In the integration testing this can also be usefull because if the behaviour of single units has already been tested and is given for sure. In addition we are going to use GreenMail as a mocked email service in all those test cases in which an email messaging system is needed.

4.3 Testing Equipment

The testing activity requires some physical equipments as well. In particular we need one Android device and one iOs device, one web-server where all the business-logic components are installed and where the mocked version of the Existing System, DBMS and Billing System has been set up.