PowerEnJoy

INTEGRATION TEST PLAN DOCUMENT

Flavio Primo, Hootan Haji Manoochehri
POLITECNICO DI MILANO | SOFTWARE ENGINEERING 2

Index

1 Introduction

1.1 Purpose

The is the Integration Test Plan Document (ITPD), it describes how integration tests are to be performed. The described tests focus on the information flow between different modules rather than on the modules themselves. It describes the adopted methodologies, the sets of all tests to be performed and the used tools during the whole process.

1.2 Scope

This system, from now on called PowerEnJoy, is about a digital management system for a car-sharing service that exclusively employs electric cars. PowerEnJoy provides classical functionality found in similar services such as: user registration, search for an available car and renting a car.
Since PowerEnJoy is about electric cars, it will manage facilities to park and recharge the cars and special discounts for users with virtuous behavior in respect to the environment and other users.

1.3 List of definitions and abbreviations

- **JUnit:** The tool used for unit testing. See Section 4 for more information.
- **Mockito:** A mocking framework used in conjunction with JUnit.
- **Arquillian:** The tool used for the actual integration testing. See Section 4 for more information.
- **RASD:** Requirements & Analysis Specification Document
- **DD:** Design Document
- **ITPD:** Integration Test Plan Document
- **DBMS:** Database Management System
- **GPS:** Global Positioning System
- **API:** Application Programming Interface
- **UI:** User Interface

2 Integration Strategy

2.1 Entry Criteria

Several entry criteria must be met before Integration Testing phase.

The whole architecture must be designed with the Design Document in mind. Following this phase, each component that will be integrated must be coded.

After development phase, each module must successfully pass a thorough unit testing, which guarantees the absence of critical bugs in the codebase.

As stated in Section 4, JUnit will be the privileged tool for this phase.

In addition to this, the following must have been produced:

- A thorough Javadoc documentation covering public methods
- The Requirements Analysis and Specification Document (RASD) [1]
- The Design Document (DD) [2]

2.2 Elements to be Integrated

The list of all elements to be integrated is reported in Table 1. This closely parallels with [2] and more in particular with the composite structure diagram and the main components diagram. For

more information on the order of the integration tests, see Section 2.4.1; for more information about the tests themselves, see Section 3.

1. **Helpers**

ID	Component	Subsystem
1	PaymentProviderHelper	Payment
2	NotificationHelper	Reservation
3	MapsHelper	Address
4	PluggedCar	Payment
5	FarFromSpecialSafeArea	Payment
6	MoreThanHalfBattery	Payment
7	MoreThanThreePassenger	Payment
8	DiscountHelper	Payment

Table 1: List of all helper components to be integrated.

2. **Models**

ID	Component	Subsystem
9	User	User
10	PaymentInformation	Payment
11	Ride	Ride
12	Reservation	Reservation
13	Car	Car
14	Address	Address
15	SafeArea (Normal & Special)	SafeArea
16	UserService	User
17	PaymentInformationService	Payment
18	RideService	Ride
19	ReservationService	Reservation
20	CarService	Car
21	AddressService	Car
22	SafeAreaService	Car

Table 2: List of all model components to be integrated.

3. **Controllers**

ID	Component	Subsystem
23	UserController	User
24	PaymentController	Payment
25	RideController	Ride
26	ReservationController	Reservation
27	CarController	Car
28	MaintenanceServiceController	Maintenance

Table 3: List of all controller components to be integrated.

It is not necessary to test the Discount class since it is an interface. It has to be programmed prior to the classes that implements it.

2.3 Integration Testing Strategy

The strategy that will be adopted is called bottom-up approach.
An incremental approach is fundamental, to prevent all shortcomings that are typical in Big Bang approach (need to wait until all modules get complete, identifying the faulty components can be difficult, some interfaces could be missed easily during the test, etc.).

We decided to choose bottom-up rather than top-down because there are many components at the lower levels (see Fig 1); this means that none or few drivers are needed.

2.4 Sequence of components / Function Integration

Since, the system consists of several different parts interrelated one to another,

It has been decided to plan integration testing under two different points of view:

- subsystem
- class-level testing

In particular, Section 2.4.1 will explain how the main software components will be integrated, as they were defined in Design Document [2] in the High-Level Components section, whilst the following section will explain how integration will take part on an upper level by considering subsystems only.

2.4.1 Software Integration Sequence

First, all transactions with the DB should work properly in order to proceed

with all other integrations: for this reason, all modules which interacting with the DBMS are tested first (DAOs and Entity Beans).

After that, we test the controllers to make sure that they behave as expected, the main controller in our system are **ReservationController**, **RideController**, **PaymentController** which has dependency on **CarController**, **UserController** so, we first make sure that the **CarController**, **UserController** are working as we expect them and after that we test **ReservationController**, **RideController**, **PaymentController** consequently.

Eventually, the core is complete; the last integration test is done on the mobile app (for example, to check if the *getCarListByRange()* method is working fine). And we make sure about connectivity of PowerEnJoy car and its interaction with system (since the PowerEnJoyBox is a hardware and has its own special test such as power consumption, circuit protection, noise cancelling etc. the internal test of the PowerEnJoyBox is done by the provider with respect to the RASD assumption section).

List of all components to be integrated

Test ID	Component 1	Subsystem 1	Component 2	Subsystem 2
A	DAOs	DataAccessLayer	DBMS	DBMS
B	EntityBeans	CrossCuttingLayer	DAOs	DataAccessLayer
1	PaymentProviderHelper	Payment	Stripe	External
2	NotificationHelper	Reservation	PushNotificationService	External
3	MapsHelper	Address	GoogleMapsAPI	External
4	PluggedCar	Payment	-	-
5	FarFromSpecialSafeArea	Payment	-	-
6	MoreThanHalfBattery	Payment	-	-
7	MoreThanThreePassenger	Payment		-
8	DiscountHelper	Payment	TestID: 4,5,6,7	-
9	User	User	DBMS	DBMS
10	PaymentInformation	Payment	DBMS	DBMS
11	Ride	Ride	DBMS	DBMS
12	Reservation	Reservation	DBMS	DBMS
13	Car	Car	DBMS	DBMS
14	Address	Address	DBMS	DBMS
15	SafeArea (Normal & Special)	SafeArea	DBMS	DBMS
16	UserService	User	User	User
17	PaymentInformationService	Payment	PaymentInformation	Payment
18	RideService	Ride	Ride	Ride
19	ReservationService	Reservation	Reservation	Reservation
10	CarService	Car	Car	Car
21	AddressService	Car	Address MapsHelper	Address
22	SafeAreaService	Car	SafeArea	SafeArea
23	UserController	User	UserService	User
24	CarController	Car	CarService RideService AddressService	Car
25	ReservationController	Reservation	ReservationService UserService CarService	Reservation
26	RideController	Ride	RideService	Ride
27	PaymentController	Payment	PaymentInformationService	Payment
28	MaintenanceServiceController	Maintenance	AddressService	Maintenance

Table 4: List of all components to be integrated.

Bottom up testing of Entity -> Service -> Controller

2.4.2 Subsystem Integration Sequence

Three main subsystems make up the whole architecture, listed in Table 5. All components described in Design Document [2] make up the System, which contains the main logic of the service; as stated in the previous section, the first important integration to be done is the one with the DBMS. When the System is ready, Mobile App and the PowerEnJoyBox can be integrated.

Integration order of subsystems

N	Subsystem	Integrate with
1	System (back-end)	DBMS
2	Mobile App	System (back-end)
3	PowerEnJoyBox	System (back-end)

Table 5: Integration order of the subsystems

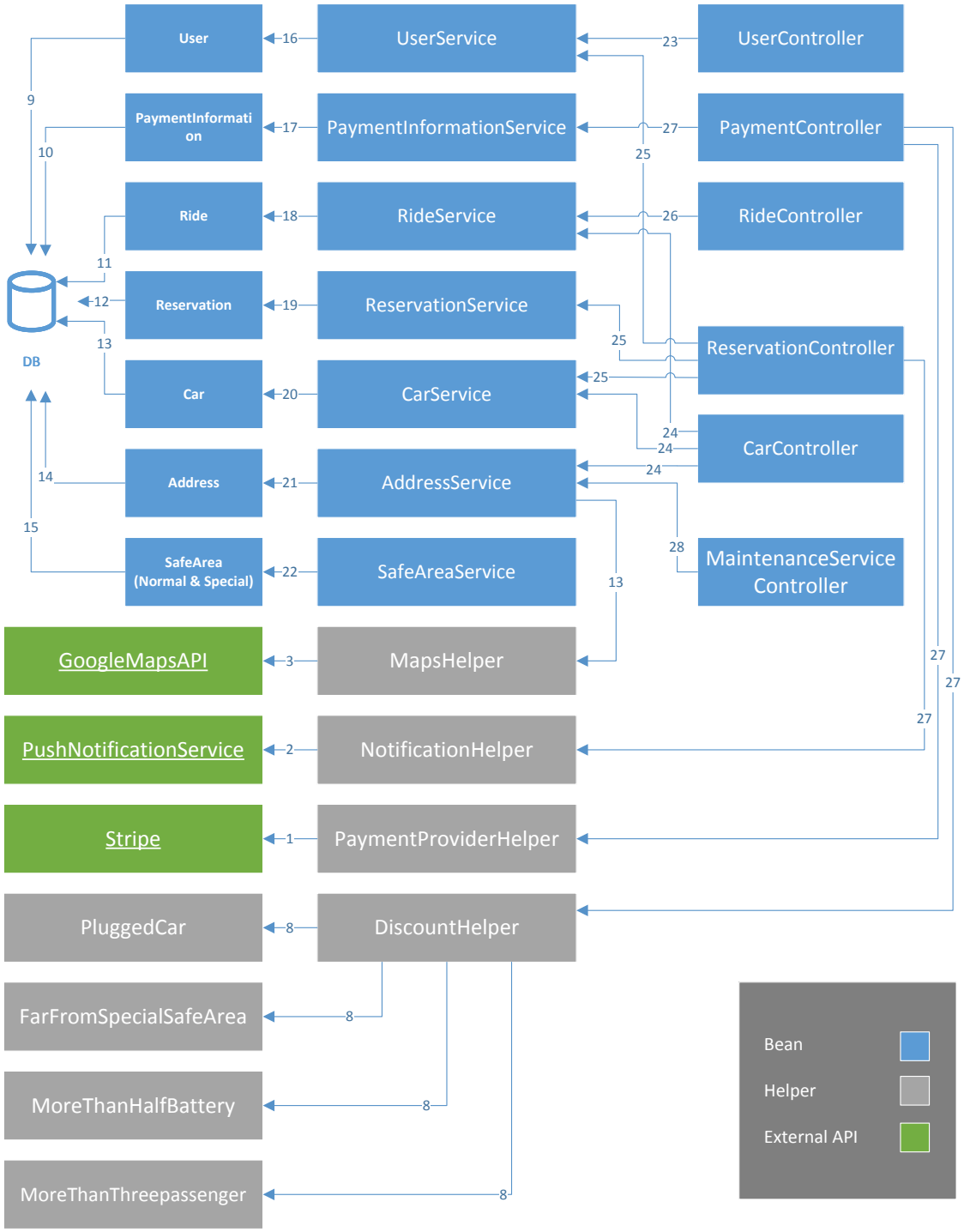

Figure 1: The elements to be integrated; the number refer to Table 2 and description of the tests in the section 3.

3 Individual Steps and Test Description

3.1 Component Tests

Please note that the actual access to the DB is done by the corresponding DAO for each Entity Bean. In order to avoid further complexity, the following tests are simplified by considering the entity as the object doing operations on the DB.

3.1.1 Helpers

3.1.1.1 Test 1: PaymentProviderHelper – Payment functionality with Stripe

Test Case Identifier	15
Test Item(s)	PaymentProviderHelper->Stripe
Input Specification	StripeToken, Total
Output Specification	A string in JSON format which specifies the status of the payment
Environmental Needs	Working Glassfish Server, Working DB Server, working Internet Connection, well configured Stripe API
Test Description	Verify that the helper works properly and we have the payment ability.

3.1.1.2 Test 2: NotificationHelper – Notification functionality with notification providers

Test Case Identifier	16
Test Item(s)	NotificationHelper-> PushNotificationServer providers
Input Specification	NotificationSettings, Title, Content
Output Specification	String, status (delivered Successfully, connected to server, …)
Environmental Needs	Working Glassfish Server, Working DB Server, working Internet Connection, well configured Notification service
Test Description	Verify that the helper works properly and we can push notification to the client.

3.1.1.3 Test 3: MapsHelper – Map functionality with Google Maps

Test Case Identifier	17
Test Item(s)	MapsHelper -> GoogleMapAPI
Input Specification	Coordinate (Latitude: 40.7734933, Longitude: 73.96567111111112)
Output Specification	List of Addresses (ex. 927 Fifth Avenue, New York, NY 10021, USA)
Environmental Needs	Working Glassfish Server, Working DB Server, working Internet Connection, well configured googleMapAPI
Test Description	Verify that the helper works properly and we can translate coordinates to addresses and viceversa.

3.1.1.4 Test 4: PluggedCar – Check if discount applies correctly

Test Case Identifier	18
Test Item(s)	PluggedCar -> Reservation, Car
Input Specification	Get a Reservation on which it takes information about the Car (specifically if it has been plugged or not).
Output Specification	Percentage of the discount to apply, can be positive or zero.
Environmental Needs	Working Glassfish Server.
Test Description	Verify that it applies a discount when the Car has been plugged when payment is being performed.

3.1.1.5 Test 5: FarFromSpecialSafeArea – Check if discount applies correctly

Test Case Identifier	19
Test Item(s)	FarFromSpecialSafeArea -> Reservation, Car, AddressService, Address
Input Specification	Get a Reservation on which it takes information about the Car (specifically its location).
Output Specification	Percentage of the discount to apply, can be positive or negative.
Environmental Needs	Working Glassfish Server.
Test Description	Verify that it applies a negative discount when the Car has been parked far from the nearest Special Safe Area.

3.1.1.6 Test 6: MoreThanHalfBattery – Check if discount applies correctly

Test Case Identifier	20
Test Item(s)	MoreThanHalfBattery -> Reservation, Car
Input Specification	Get a Reservation on which it takes information about the Car (specifically its battery percentage).
Output Specification	Percentage of the discount to apply, can be positive or zero.
Environmental Needs	Working Glassfish Server.
Test Description	Verify that it applies a discount when the Car has been plugged when payment is being performed.

3.1.1.7 Test 7: MoreThanThreePassenger – Check if discount applies correctly

Test Case Identifier	21
Test Item(s)	MoreThanThreePassenger -> Reservation, Ride
Input Specification	Get a Reservation on which it takes information about the Ride (specifically how many people are inside the car when the Ride starts).
Output Specification	Percentage of the discount to apply, can be positive or zero.
Environmental Needs	Working Glassfish Server.
Test Description	Verify that it applies a discount when the Car has carried more than three passengers.

3.1.1.8 Test 8: DiscountHelper –Discounts calculation

Test Case Identifier	23
Test Item(s)	DiscountHelper -> Reservation
Input Specification	Reservation on which it takes necessary information to apply discounts.
Output Specification	The final price for the reservation with discounts applied.
Environmental Needs	Working Glassfish Server.
Test Description	Verify that it returns the correct price based on the applicable discounts on the reservation.

3.1.2 Models

3.1.2.1 Test 9: User - Access to DB

Test Case Identifier	1
Test Item(s)	User -> DBMS
Input Specification	-
Output Specification	The User corresponding table on the DB.
Environmental Needs	Working Glassfish Server, Working DB Server.
Test Description	Verify that table is created w.r.t. entity attributes and constraints.

3.1.2.2 Test 10: PaymentInformation - Access to DB

Test Case Identifier	2
Test Item(s)	PaymentInformation -> DBMS
Input Specification	-
Output Specification	The PaymentInformation corresponding table on the DB.
Environmental Needs	Working Glassfish Server, Working DB Server.
Test Description	Verify that table is created w.r.t. entity attributes and constraints.

3.1.2.3 Test 11: Ride - Access to DB

Test Case Identifier	3
Test Item(s)	Ride -> DBMS
Input Specification	-
Output Specification	The Ride corresponding table on the DB.
Environmental Needs	Working Glassfish Server, Working DB Server.
Test Description	Verify that table is created w.r.t. entity attributes and constraints.

3.1.2.4 Test 12: Reservation - Access to DB

Test Case Identifier	4
Test Item(s)	Reservation -> DBMS
Input Specification	-
Output Specification	The Reservation corresponding table on the DB.
Environmental Needs	Working Glassfish Server, Working DB Server.
Test Description	Verify that table is created w.r.t. entity attributes and constraints.

3.1.2.5 Test 13: Car - Access to DB

Test Case Identifier	5
Test Item(s)	Car -> DBMS
Input Specification	-
Output Specification	The Car corresponding table on the DB.
Environmental Needs	Working Glassfish Server, Working DB Server.
Test Description	Verify that table is created w.r.t. entity attributes and constraints.

3.1.2.6 Test 14: Address - Access to DB

Test Case Identifier	6
Test Item(s)	Address -> DBMS
Input Specification	-
Output Specification	The Address corresponding table on the DB.
Environmental Needs	Working Glassfish Server, Working DB Server.
Test Description	Verify that table is created w.r.t. entity attributes and constraints.

3.1.2.7 Test 15: SafeArea (Normal & Special) - Access to DB

Test Case Identifier	7
Test Item(s)	SafeArea (Normal & Special) -> DBMS
Input Specification	-
Output Specification	The Normal and Special SafeArea corresponding tables on the DB.
Environmental Needs	Working Glassfish Server, Working DB Server.
Test Description	Verify that tables are created w.r.t. entity attributes and constraints.

3.1.2.8 Test 16: UserService - Access to DB

Test Case Identifier	8
Test Item(s)	UserService -> User, DBMS
Input Specification	Frequent queries on table User (crud, getUserByMail, getUserByUserToken, …).
Output Specification	The corresponding tuples to the query.
Environmental Needs	Working Glassfish Server, Working DB Server.
Test Description	Verify that all entities in the DB are correctly mapped.

3.1.2.9 Test 17: PaymentInformationService - Access to DB

Test Case Identifier	9
Test Item(s)	PaymentInformationService -> PaymentInformation, DBMS
Input Specification	Frequent queries on table PaymentInformation (crud, …).
Output Specification	The corresponding tuples to the query.
Environmental Needs	Working Glassfish Server, Working DB Server.
Test Description	Verify that all entities in the DB are correctly mapped.

3.1.2.10 Test 18: RideService - Access to DB

Test Case Identifier	10
Test Item(s)	RideService -> Ride, DBMS
Input Specification	Frequent queries on table Ride (crud, …).
Output Specification	The corresponding tuples to the query.
Environmental Needs	Working Glassfish Server, Working DB Server.
Test Description	Verify that all entities in the DB are correctly mapped.

3.1.2.11 Test 19: ReservationService - Access to DB

Test Case Identifier	11
Test Item(s)	ReservationService -> Reservation, DBMS
Input Specification	Frequent queries on table Reservation (crud, …).
Output Specification	The corresponding tuples to the query.
Environmental Needs	Working Glassfish Server, Working DB Server.
Test Description	Verify that all entities in the DB are correctly mapped.

3.1.2.12 Test 20: CarService - Access to DB

Test Case Identifier	12
Test Item(s)	CarService -> Car, DBMS
Input Specification	Frequent queries on table Car (crud, getCarByAddress, getCarListByRange, …).
Output Specification	The corresponding tuples to the query.
Environmental Needs	Working Glassfish Server, Working DB Server.
Test Description	Verify that all entities in the DB are correctly mapped.

3.1.2.13 Test 21: AddressService - Access to DB

Test Case Identifier	13
Test Item(s)	AddressService -> Address, DBMS
Input Specification	Frequent queries on table Address (crud, getAddressByString, getListByRange, …).
Output Specification	The corresponding tuples to the query.
Environmental Needs	Working Glassfish Server, Working DB Server.
Test Description	Verify that all entities in the DB are correctly mapped.

3.1.2.14 Test 22: SafeAreaService - Access to DB

Test Case Identifier	14
Test Item(s)	SafeAreaService -> SafeArea, DBMS
Input Specification	Frequent queries on table SafeArea (crud, …).
Output Specification	The corresponding tuples to the query.
Environmental Needs	Working Glassfish Server, Working DB Server.
Test Description	Verify that all entities in the DB are correctly mapped.

3.1.3 Controllers

3.1.3.1 Test 23: UserController – User related functionalities

Test Case Identifier	24
Test Item(s)	UserController -> UserService
Input Specification	User information.
Output Specification	Query to User entities handled by the UserService.
Environmental Needs	Working Glassfish Server, Working DB Server.
Test Description	Verify that all the high perspective business logic operations concerning the User are correctly handled. A guest should be able to either register or login as a User. A User should be able to delete its profile.

3.1.3.2 Test 24: PaymentController – Payment related functionalities

Test Case Identifier	25
Test Item(s)	PaymentController -> PaymentInformationService, DiscountHelper,PaymentProviderHelper
Input Specification	Ride information.
Output Specification	Invoice (Total, Discount percentage, status of the payment, Ride Id, …)
Environmental Needs	Working Glassfish Server, Working DB Server, Working Internet Connection, already Tested Helpers and Services.
Test Description	Verifies that mentioned components works well as an integrated system to make a payment transaction.

3.1.3.3 Test 25: RideController - Ride related functionalities

Test Case Identifier	26
Test Item(s)	RideController -> RideService
Input Specification	Reservation Information
Output Specification	Create (start) a ride and keep tracks of it, at the end Stop the ride.
Environmental Needs	Working Glassfish Server, Working DB Server
Test Description	Verifies that mentioned components works well as an integrated system to make a payment transaction.

3.1.3.4 Test 26: ReservationController - Reservation related functionalities

Test Case Identifier	27
Test Item(s)	ReservationController -> ReservationService
Input Specification	User id and Car information.
Output Specification	Query to Reservation entities handled by the ReservationService.
Environmental Needs	Working Glassfish Server, Working DB Server.
Test Description	Verify that all the high perspective business logic operations concerning the Reservation are correctly handled. Given User and Car information (respecting the constraints) it should be possible to manage (create, delete, modify) a reservation.

3.1.3.5 Test 27: CarController - Car related functionalities

Test Case Identifier	28
Test Item(s)	CarController -> CarService
Input Specification	Car information and coordinates for localization.
Output Specification	Query to Car entities handled by the CarService.
Environmental Needs	Working Glassfish Server, Working DB Server.
Test Description	Verify that all the high perspective business logic operations concerning the Car (reserve or unlock a car, …) are correctly handled.

3.1.3.6 Test 28: MaintenanceServiceController – Maintenance Service related functionalities

Test Case Identifier	29
Test Item(s)	MaintenanceServiceController -> AddressService, CarService
Input Specification	Car information
Output Specification	Location of the car
Environmental Needs	Working Glassfish Server, Working DB Server.
Test Description	Verify that all the high perspective business logic operations concerning the Maintenance Service (find, or update a car's status, …) are correctly handled.

3.2 Subsystem tests

3.2.1.1 Test 29: Application Server - Main functionalities with DB

Test Case Identifier	29
Test Item(s)	System (back-end) -> DBMS
Input Specification	Standard query by the application server on the DB
Output Specification	Correct tuples, based on the actual component that made the request
Environmental Needs	Working Glassfish Server, Working DB Server.
Test Description	Several components in the application server use DB for their functionalities, by either inserting, updating or deleting information.

3.2.1.2 Test 30: PowerEnJoyApp - Integration with the application server

Test Case Identifier	30
Test Item(s)	Mobile App -> System (back-end)
Input Specification	User inputs, application server notifications and query requests responses
Output Specification	Results on the application server
Environmental Needs	Working Glassfish Server, Working DB Server, Mobile phone with GPS and internet enabled
Test Description	All the system functionalities must be tested through the Mobile App (user registration and login, find a car, reserve for a car, pay for a ride, …).

3.2.1.3 Test 31: PowerEnJoyBox - Integration with the application server

Test Case Identifier	31
Test Item(s)	PowerEnJoyBox -> System (back-end)
Input Specification	AuthenticationKey, notifications
Output Specification	Car status
Environmental Needs	Working Glassfish Server, Working DB Server, Working PowerEnJoyBox
Test Description	Verify that all the high perspective business logic operations concerning the PowerEnJoyBox (correctly receives the authenticationkey to unlock the car, update car's status on the application server, …) are correctly handled.

4 Tools and Test Equipment Required

4.1 Junit

JUnit is a simple framework to write repeatable tests. Each test is done on a single unit, usually composed of one public class. Tests can also be grouped in Suites for multiple instances at once.

Since the system interacts with several external service providers (for example, the interaction with the Stripe payment service or all interactions between a module and the DBMS), a mock framework is necessary.

Among several and similar products (JMock, EasyMock, Powermock, ...), Mockito has been chosen for its simplicity and clearness. These tools are used before Integration Testing happens (described throughout this document).

4.2 Arquillian

Arquillian is an integration testing framework for business objects that are executed inside a container or that interact with the container as a client.

It combines a unit testing framework (JUnit), and one or more supported target containers (Java EE container, servlet container, …) to provide a simple, flexible and pluggable integration testing environment.

Arquillian strives to make integration testing no more complicated than basic unit testing, so it has been decided to use Arquillian as the integration testing framework for its simplicity.

4.3 Smartphone

As described in [1], the mobile application will be available for customers using Android, iOS and Windows Phone. This implies that at least 3 smartphones, one for each OS, must be used for testing:

- Android phones must be updated to at least Android JellyBean
- iOS phones must be updated to v. 9
- Windows Phone must be updated to v. 10
- Each phone must have internet connection, GPS and Bluetooth enabled

5 Program Stubs and Test Data Required

In an undeveloped environment, following the criteria of bottom-up approach, we need to define and use drivers in order to have a complete environment in which we can test the developed parts.

- **Test Server:** A working Glassfish test server is needed in order to properly host the Application server.
- **Test database:** The target environment must have a working and configured DBMS, in which test data and tables must reflect the entities and the relations described in the ER diagram showed in the Design Document [2].
  This mock DB should contain random valid and invalid data about all the entities specified in the ER diagram (User, PaymentInformation, Ride, Reservation, …) for JavaEE: Drivers used to test the proper behaviour of the Java Entity Beans while the Application server is not fully implemented.
  These are placeholders for all the Services and Controllers in the application (ReservationController, ReservationService, UserController, …) and can be eliminated as the upper-level modules are ready.
- **API client:** It is also necessary to emulate a client application with an APIclient which interacts with the server via HTTP requests.
- **Test e-mail confirmation:** An email sender/receiver is needed in order to test and automate the email confirmation process when a user signs up for the service.
- **Fake GPS data:** In order to test position handling for PowerEnJoy cars, a set of random GPS data are generated.

6 References

1. Flavio Primo, Hootan Haji Manoochehri – PowerEnJoy: Requirements Analysis and Specification Document
2. Flavio Primo, Hootan Haji Manoochehri – PowerEnJoy: Design Document
3. AA.VV. - Arquillian Reference Guide (https://docs.jboss.org/author/display/ARQ/Reference+Guide)
4. AA.VV. - Mockito Reference Guide (http://site.mockito.org/mockito/docs/current/org/mockito/Mockito.html)
5. Oracle – JEE Documentation (http://docs.oracle.com/javaee/7/index.html)
6. Markus Eisele - Modern Java EE design patterns (https://www.oreilly.com/ideas/modern-java-ee-design-patterns)

7 Hours Spent

Table describing the time management for the team.

Team member	**Hours**
Flavio Primo	15
Hootan Haji Manoochehri	15
	30 total