Politecnico di Milano

Computer Science and Engineering's master degree course

Department of Electronics and Information

PowerEnjoy
Design Document

Version 1.0

Release Date: 15/01/2017

Customer: Eng. Elisabetta DI NITTO

Authors:

Eng. Marco FERNI	Id. 877712
Eng. Angelo Claudio RE	Id. 877808
Eng. Gabriele TERMIGNONE	Id. 877645

Contents

Chapter 1

Introduction

1.1 Revision History

Version 1.0 - First Release Version

1.2 Purpose and Scope

PowerEnjoys is a Car Reservation professional software whose main goal is to help people move around easier, without having to rely on their personal transport; a secondary goal is the reduction of cities' pollution and acoustic noise.

A user can look for a car by entering either his current position (detected by using their smartphone's GPS) or a specific location, chosen on the map, that he'll need to reach by himself.

The system policies encourage a smarter use of our service, by offering discounts to those who share a trip together.

This is The Integration Test Plan Document (ITPD) for the PowerEnjoy Project.

The ITPD's purpose is to document and define the approach to be used in the Integration Testing of all the project's components.
In addition, a brief description about testing software and tools to be used will be given.

1.3 List of Definitions and Abbreviations

- *Cost of the Trip:* raw price of the service calculated only on the base of the duration of the car's usage, before discounts or additional charges are applied.
- *Virtuosness Coefficient:* the factor by which to multiply the cost of the trip to get the amount of the bill. Its initial value is 1.
- *Supervisor:* a company employee who work at the Car hub controller
- *Recharge on site:* a company procedure to send a worker to recharge a low car that was parked detached from the power grid
- *Car recovery:* a worker is sent to retrieve a car that has been forgotten outside a safe area and move that car into in one of these lot.
- *Guest:* a person who is not already registered to the system.
- *User:* a registered customer.
- *DD:* Design Document
- *DB:* DataBase
- *RASD:* Requirement Analysis and Specification Document
- *ITPD:* Integration Test Plan Document

1.4 List of Reference Documents

- The PowerEnjoy's RASD
- PowerEnjoy's DD
- The project's Assignments PDF
- Old years projects
- The ITPD standards
- Software Engineering course slides
- Specific tools' tutorials and documentation

Chapter 2

Integration Strategy

2.1 Entry Criteria

This section describes the prerequisites that need to be met before integration testing can be started.
First of all, code inspection has to be performed on all the code in order to find possible issues and to ensure maintainability and respect of conventions.
Taking as reference the component diagram (2.2 from DD), the following component must be unit-tested before our integration tests:

- Notifications displayer (Supervisor)
- Sensor/actuator manager (Car)
- Info displayer (Car)
- Mobile App (User)
- DB
- Email gateway
- Push gateway

The DB component represents the database: because the database run on the same system of the application server, in the component diagram it is represented

as a simple component; following this convention, we will continue to represent it as a component and to consider it as part of the *"central server"* subsystem.

In addition, there are present in the diagram 2 components which represent external services (Google maps and bank service), which can be considered already reliable and do not need to be unit tested.

We should test all non-trivial methods, getter and setter methods can be skipped. The components *"User"* and *"Safe park slot"*, which contain only getter and setter methods, are considered as already tested.

2.2 Elements to be Integrated

The elements to be integrated are all the elements represented in the already mentioned component diagram, considering that our system must cooperate with 2 external services (google maps and the bank service).

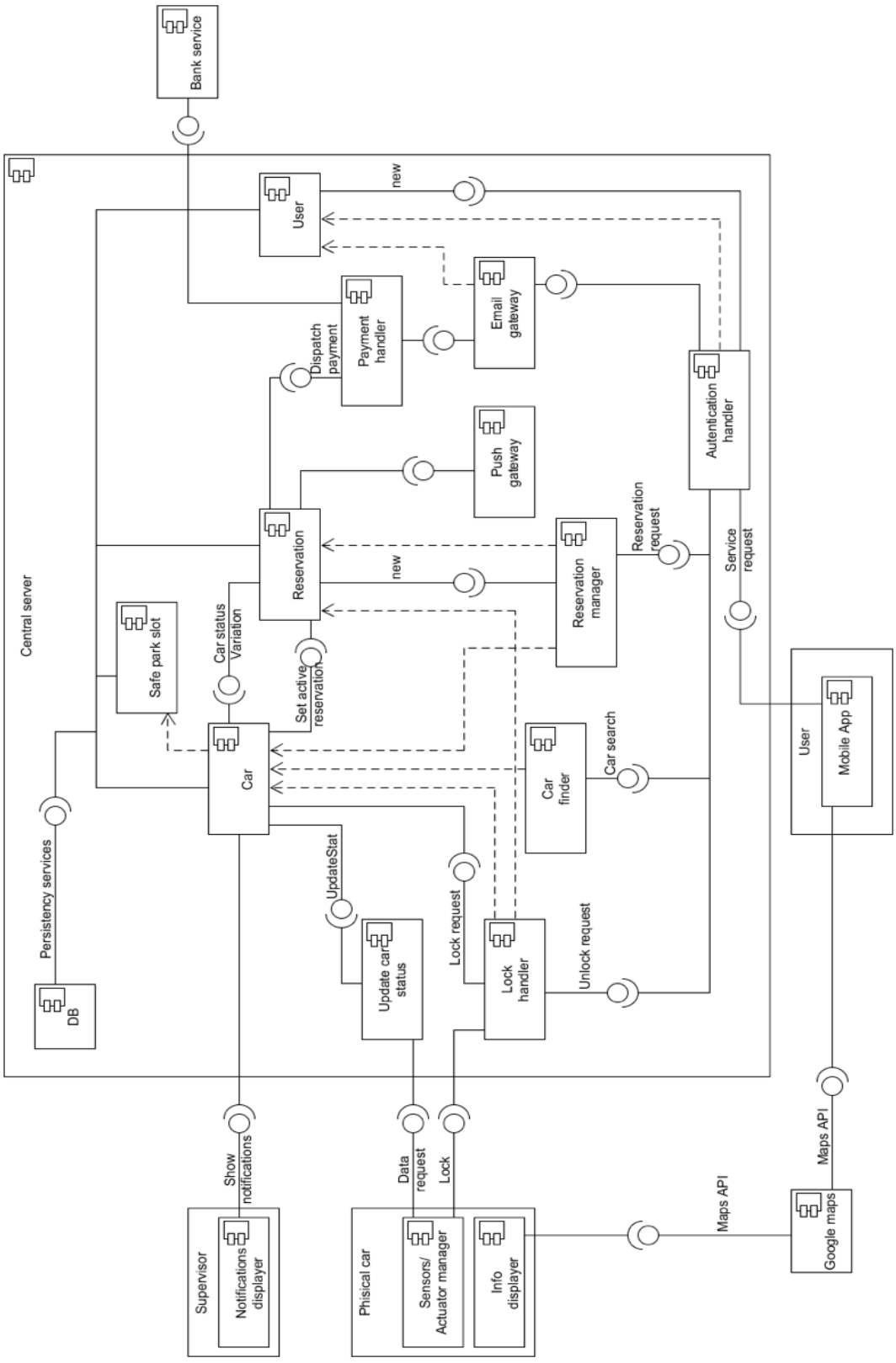

Figure 2.1: *Component Diagram*

2.3 Integration Testing Strategy

The integration strategy that we decided to follow is the bottom-up approach. The main reason for this choice is that we assume that we already have the unit-test for some of the simplest components, so we can proceed from the bottom.

Moreover, except for the central server, the other high-level components are composed by simple parts that we consider as atomic; the high level components communicate through well defined interfaces (REST API), so the integration of each of them will not be hard in a later time.

In addition, this approach has other intrinsic advantages: we can limit the usage of stubs, the errors are well located and, if the realization of the components follows a bottom-up approach too, the testing of lower level modules can take place earlier.

2.4 Sequence of Component/Function Integration

2.4.1 Software Integration Sequence

In this section it will be shown the sequence of the components' integration. As mentioned above, we consider the high-level components *"User"*, *"Supervisor"* and *"Physical car"* as atomic subsystems, and therefore they are not consider in this section; in the following figure we describe in detail the integration sequence of the components of the *"Central server"* subsystem. The bottom-up approach has been respected in the majority of the cases, with only two exception: it's present a circular dependency between the *"Car"* and *"Reservation"* components, that forces us to take a choice on which one we implement first, and the push gateway, which have to send information both to the physical car and to the mobile app: we will need a stub for the latter one.

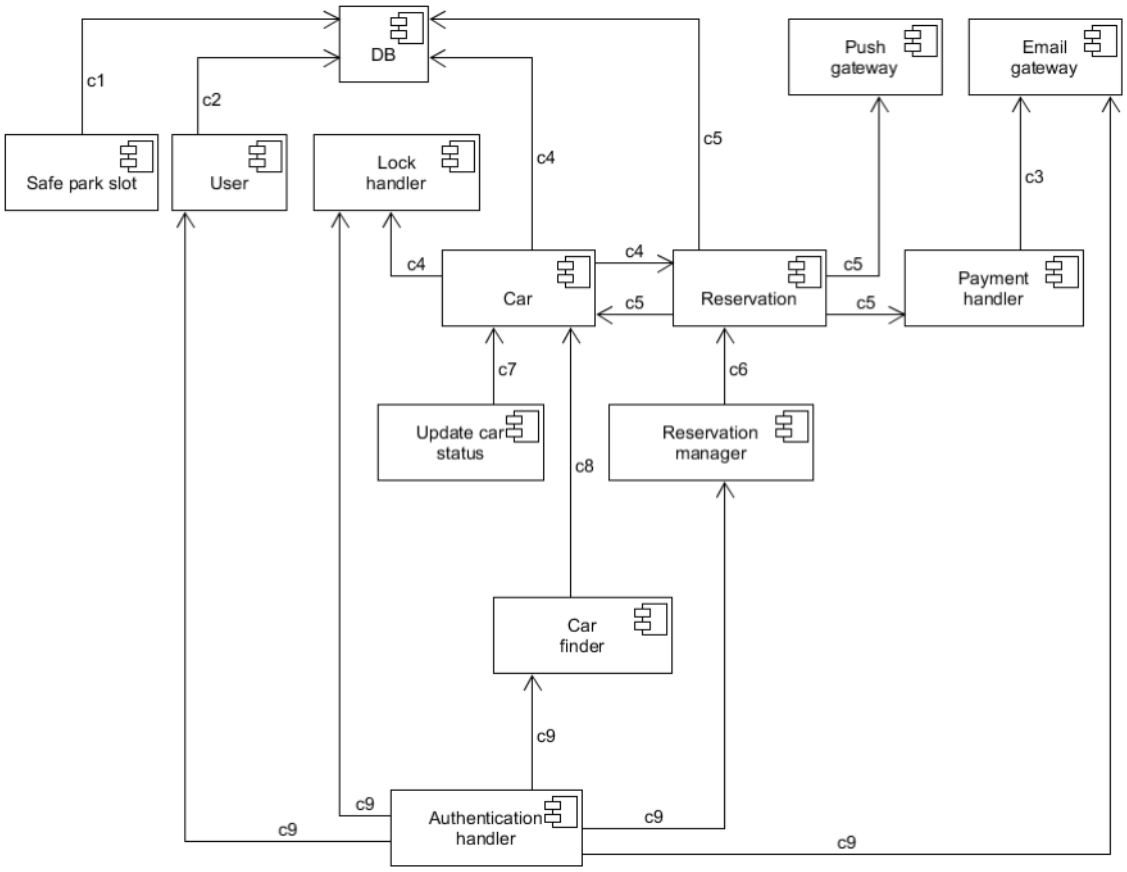

Figure 2.2: *Central server integration sequence*

$N°$	**Component**	**Integrates with**
c1	Safe park slot	DB
c2	User	DB
c3	Payment handler	Email gateway
c4	Car	DB, Reservation, Lock handler
c5	Reservation	DB, Car, Push gateway, Payment handler
c6	Reservation manager	Reservation
c7	Update car status	Car
c8	Car finder	Car
c9	Authentication handler	User, Lock handler, Car finder, Reservation manager, Email gateway

2.4.2 Subsystem Integration Sequence

For subsystem integration we follow the bottom-up approach too. The reason to do so is that it's simpler to implement driver for testing the low level components, instead of making coherent stubs of the subsystems. Moreover, a working central server is needed to test in a profitable way the mobile app (User subsystem).

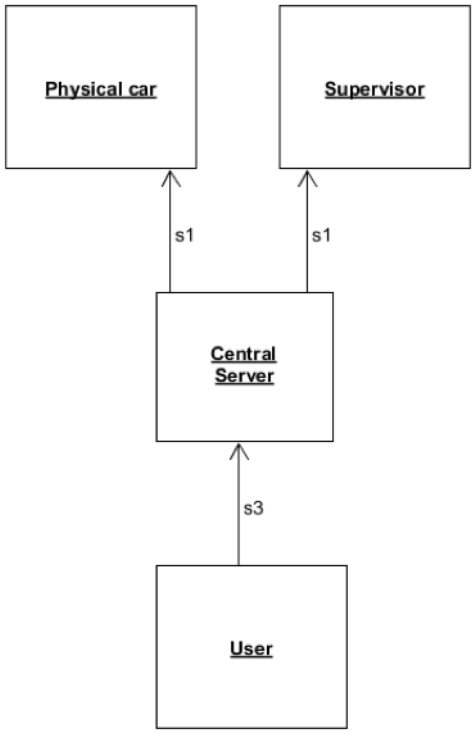

Figure 2.3: *Subsystems integration sequence*

N°	Subsystem	Integrates with
s1	Central server	Physical car, Supervisor
s3	User	Central server

Chapter 3

Individual Steps and Test Description

3.1 Integration test case c1

Test Case Identifier	C1T1
Test Item(s)	Safe park slot → DB
Input Specification	Typical queries on table Safe park slot
Output Specification	Correct result returned
Environmental Needs	DB
Description	The DB should contain the corret informations related to the parking slots. Some views of known test data must be done.

3.2 Integration test case c2

Test Case Identifier	C2T1
Test Item(s)	User → DB
Input Specification	Typical queries on table User
Output Specification	Correct result returned / correct data insertion
Environmental Needs	DB
Description	The "new" interface of the User Component will be used to insert and modify some User data into the DB. The tester should then verify that the DB contains the correct data.

3.3 Integration test case c3

Test Case Identifier	C3T1
Test Item(s)	Payment handler → Email gateway
Input Specification	Typical payment handler input
Output Specification	Correct request for bank services / Correct email generation
Environmental Needs	Email gateway
Description	Some test payment will be executed. An e-mail, sent via the Email Gateway component, is expected to be received. A test g-mail account will be used for such purpose. Some wrong amount (like a negative value) must be tested.

3.4 Integration test case c4

Test Case Identifier	C4T1
Test Item(s)	Car → DB
Input Specification	Typical queries on table Car
Output Specification	Correct result returned / correct data insertion
Environmental Needs	DB
Description	Some standard DB interactions.

Test Case Identifier	C4T2
Test Item(s)	Car → Reservation
Input Specification	Typical input for *"car status variation"* method of reservation
Output Specification	Check if the correct method of reservation is called
Environmental Needs	Update car status driver, C1, Reservation stub
Description	Knowing that the *"Car status variation"* interface is used to notify the reservation object about the changing of specific attributes on the state of the Associated car, a correct detection of when the first engine ignition occurs (so the system can register the number of passenger, or when a car is parked (in a safe area or not, in order to correctly launch the countdown, or ultimately, when a parked car is turned back on, in order to delete the running countdown) must occur. Since the integration of the *"Reservation"* component is not done at this step, we need to use a stub to test the interaction.

Test Case Identifier	C4T3
Test Item(s)	Car → Lock handler
Input Specification	Typical input for *"Lock handler"*
Output Specification	Check if the request is sent under the correct conditions, check if the physical car correctly locks
Environmental Needs	Physical car, Update car status driver, Reservation driver
Description	All the different interactions between these components are to be tested (lock/timeout)

3.5 Integration test case c5

Test Case Identifier	C5T1
Test Item(s)	Reservation → DB
Input Specification	Typical queries on table Reservation
Output Specification	Correct data insertion
Environmental Needs	DB
Description	Some views of the stored reservations' parameters are to be generated and then cross checked with the test data.

Test Case Identifier	C5T2
Test Item(s)	Reservation → Car
Input Specification	Typical input *"Set active reservation"* method reservation
Output Specification	Check if the correct method of car is called
Environmental Needs	Reservation manager driver, C4
Description	The *"set active reservation"* method, used by a new reservation to associate itself to the reserved car, is also used by a reservation when it is about to expire, to set the *"actualReservation"* attribute of the associated car to null.

Test Case Identifier	C5T3
Test Item(s)	Reservation → Push gateway
Input Specification	Typical *"push gateway"* input
Output Specification	Check if the correct data are dispatched
Environmental Needs	Mobile app (from user subsystem) stub, Physical car
Description	The *"push gateway"* component is used to send information about the reservation status to the user's app and to the car's monitor. Some changes in the Reservation are to be forced (directly hardcoded and then re-executed after the implementation of the Reservation Manager to allow such notifications to occur.

Test Case Identifier	C5T4
Test Item(s)	Reservation → Payment handler
Input Specification	Typical *"Payment handler"* input
Output Specification	Check if the correct method of car is called
Environmental Needs	C3
Description	

3.6 Integration test case c6

Test Case Identifier	C6T1
Test Item(s)	Reservation Manager → Reservation
Input Specification	Typical *"Reservation manager"* input
Output Specification	Check if a new reservation is correctly created, observing the requirements criteria
Environmental Needs	C4, C5
Description	All the test required for such stated Output to occur; in addition, the push notification component integration should be re-tested.

3.7 Integration test case c7

Test Case Identifier	C7T1
Test Item(s)	Update car status → Car
Input Specification	Typical *"Update car status"* input
Output Specification	Check if the car's data are correctly updated
Environmental Needs	C4, Physical car
Description	

3.8 Integration test case c8

Test Case Identifier	C8T1
Test Item(s)	Car finder → Car
Input Specification	Typical *"Car finder"* input
Output Specification	Check if the position of the car which respects the search criteria are returned
Environmental Needs	C4
Description	Some tests regarding the representation of such cars in the map are strongly suggested, to check if the position of the car is consistent with the real world and with the car own perceived position.

3.9 Integration test case c9

Test Case Identifier	C9T1
Test Item(s)	Authentication handler → User, Email gateway
Input Specification	Typical *"Authentication handler"* input for the creation of a new user
Output Specification	Correct creation of a new user, email containing the password correctly sent
Environmental Needs	C2, Email gateway
Description	A test user has to be created: the *"authentication handler"* calls the ''new" method of the component *"user"*, and uses the *"email gateway"* component to send the confirmation email containing the password to the new user. All this data should be correct and a user creation can't occur if the inpout fileds are invalid.

Test Case Identifier	C9T2
Test Item(s)	Authentication handler → Lock handler
Input Specification	Typical *"Authentication handler"* input for an unlock request
Output Specification	Check if the physical car correctly unlocks under the right conditions
Environmental Needs	C4, C2, Mobile app driver
Description	The *"authentication handler"* component must check if the user who sent the request is correctly logged and has the right permissions, then redirect the request to the right component.

Test Case Identifier	C9T3
Test Item(s)	Authentication handler → Car finder
Input Specification	Typical *"Authentication handler"* input for the research of a car
Output Specification	Check if the *"car finder"* method is correctly called
Environmental Needs	C4, C2, Mobile app driver
Description	

Test Case Identifier	C9T4
Test Item(s)	Authentication handler → Reservation manager
Input Specification	Typical *"Authentication handler"* input for the creation of a new reservation
Output Specification	Check if the *"Reservation manager"* method is correctly called
Environmental Needs	C6, C2, Mobile app driver
Description	

Chapter 4

Tools and Test Equipment Required

The way the component interacts with the system is just as important as the work it performs. It's then vital that every aspect of the integration procedure is tested with the right consideration and the right tools. The tools used to automate the testing are the following:

- **Apache JMeter:** a load testing tool, built in Java, for analyzing and measuring performance; can be used to simulate heavy load on a server, network or object.
- **Junit:** it's a framework used to write repeatable tests. Will be used in synergy with Arquillian and Mockito.
- **Arquillian:** it's an integration testing framework for containers. Arquillian brings the test to the runtime so there's not need to manage the runtime from the test.
- **Mockito:** Mockito provides a framework for interaction testing, supporting the scaffolding by defining stubs when needed.

Sometimes, after an integration between two components, new vulnerabilities are introduced. (Think about a new registration module replacing or augmenting

a previous one. That may allow the creation of unwanted superusers through some exploit). A short Vulnerability and then PenTest (Penetration Testing) is so executed to check for vulnerabilities. The software and tools used will not be discussed here (Kali Linux, Sqlmap, Nexpose,...).

Chapter 5

Program Stubs and Test Data Required

5.1 Stub

Since we decided to follow a bottom-up approach, we only have 2 stubs to be developed during the test phase:

5.1.1 Reservation Stub

Usage: C4T2

Desciption: This stub is required to test the correctness of the interaction between the *"Car"* and the *"Reservation"* components. In particular, when Car calls the *"Car status variation"* method, under certain conditions the reservation of the car can change status from active to expired: this cause the call of the *"Set active reservation"* method of car by the reservation.

5.1.2 Mobile app stub

Usage: C5T2

Desciption: This stub is required to test the correctness of the information sent through the push gateway; it is not necessary to emulate the entire mobile app, but the only functionality needed is the receipt of the pushed notifications.

5.2 Data for tests

We will initially populate the database with fake data regarding Cars and Safe area (including safe park slots); these are the only information strictly needed by some components before starting the test. Other information (such as users, reservations) could be inserted during the various test phases.

Chapter 6

Effort Spent

Date	Marco (h)	Angelo (h)	Gabriele (h)
13/12/2016		0.5	
27/12/2016			2
28/12/2016			2
29/12/2016			6
30/12/2016			3
31/12/2016	2		
02/01/2017		1.5	
03/01/2017	1		
05/01/2017	1		
07/01/2017	4	1	
08/01/2017		4	
13/01/2017	1	1	1
14/01/2017	1	1	1
TOTAL	10	9	15