Politecnico di Milano

AA 2016-2017

Software Engineering 2 project: "*PowerEnJoy*"

Prof. Mottola Luca

ITPD

Integration **T**est **P**lan **D**ocument

15/01/17 - version 1.0

Edited by:

Giulia Pennati **878686**

Paola Sanfilippo **809689**

Marco Siracusa **878083**

Table of Contents

1 INTRODUCTION

1.1 Revision History:

- 09/01/2017 - 1.0 version

1.2 Purpose and Scope:

Purpose

The Design Document's purpose is to give more detailed instructions of how PowerEnJoy system should be tested. It is addressed to the developers that after the development of some components wonder how to test it.

Scope

This project aims to realize an electric car sharing service.
This service makes available to the users an entire fleet of electric cars, power grid stations furnished with plugs to recharge the cars and a maintenance team ready to intervene in case of need.
It also provides both a website and a mobile application, with which the registered users can access the system functionalities.
In order to access these system functionalities, a person should first register, inserting all the required data (that has to be correct), and then wait until the reception of an email containing a password.
The email (s)he provided during the registration with the password received via email are his/her credentials to log into the system and access all the functionalities.
A logged in user can make a research of a car within a certain area, the "Safe Area", choosing the one that most fits his/her needs, and reserve the desired one.
(S)He also can use the car according to the PowerEnJoy company's rules (that will be explained in the next pages) and can both enjoy the different available discounts and be charged with extra fees, limited to a discount of 30% (i.e. +charges-discounts ≤ 30% total discount).
Discounts and charges policy is thought to reward those behaviours that simplify the availableness of usable cars (i.e. in the right place and with a sufficient battery level) and discourage the others.

1.3 List of Definitions and Abbreviations

Definitions:

[D01] **Visitor**: a person who has not already completed the registration phase. An inaccessible account is assigned to him/her.
[D02] **Account**: a record into the system that identifies each visitor/user and contains all his/her data. Each account can be either “Not owing” or “Owing” depending on whether the user has carried out all payments or not.
[D03] **Registration phase**: procedure that starts with a visitor that sends the required data and ends receiving a password back from the system (being allowed to log into that and use all the provided functionalities).
[D04] **User**: any person that, consequently a registration phase, has received back the password and so is able, after the login phase, to access all the system functionalities.
[D05] **Login**: procedure with which a user (already registered), inserting his/her username and password, accesses into the system in order to use all the provide functionalities.
[D06] **Car**: any electric vehicle of the provided fleet. This vehicle is equipped with all the required sensors to provide the information necessary to the system.
The car can be:

- Not reserved: has not been reserved yet, so it’s ready to be reserved
- Reserved: has already been reserved, so there’s another user that has reserved the car or is using it

And

- Plugged (to a power grid station)
- Not plugged (to any power grid station)

Parts of the car that are named throughout the text are:

- Doors
- Trunk
- Battery
- Internal screen
- Terminal

A car provides an interface to communicate with the system.
[D07] **Reservation**: action that makes the cars change their state from “reserved” to “Not reserved” and vice versa. It **starts** when the user receives the confirmation of the requests of the car’s reservation done through the mobile application and **ends** when the user unlocks the car
[D08] **Reservation-time**: time elapsed from the beginning to the end of the reservation.
[D09] **Locking**: action requested by the system that consists in the locking of the car’s doors and trunk too.
[D10] **Unlocking**: action requested by the system that consist in the unlocking of the car’s doors and trunk too.
[D11] **Deploy**: action that intends the effective use of the service that the user is charged for. It **starts** from the unlocking of the car and **ends** on the locking.
[D12] **Deploy-time**: time elapsed from the beginning to the end of the deploy.
[D13] **Ride**: action that **starts** when the engine is switched on and **ends** when the engine is switched off.

[D14] **Ride-time**: time elapsed from the beginning to the end of the ride.
[D15] **Passenger**: person who, for a single ride, travels with the user inside the reserved car.
[D16] **Power grid station**: place equipped with company's plugs whereby the user can put the car on charge. These are placed only in the safe area and well distributed. Each one has a total number of plugs.
[D17] **Safe area**: places within the urban city area where the user is allowed to park the car in. It has been already defined before the deploy phase.
[D18] **Payment system**: external payment system that is supposed to carry out the transactions of the due amounts of money of the several users. It communicates with the developed system through an interface.
[D19] **Pledge**: Amount of money (100€) used as security for the fulfillment of the reservation debt. It is liable to forfeiture the damage of the car or the inability to afford the payment of the reservation.
[D20] **Data checking system**: system that checks the validity and completeness of the user's data. The feedback is positive if and only if the inserted data are completed with respect to the required ones and valid with respect to the laws applied in the country where the system is running.
[D21] **Money saving option**: option that, when selected, drives the user to the power grid station that is nearest to the destination (s)he selected but with not less than half of the plugs available. If the user leaves and plugs the car in the suggested place, (s)he receives the discount.
[D22] **Maintenance team**: team that manages the cars that has been left with a percentage battery level less than 20%
[D23] **Research**: procedure that allows the user to see which cars (s)he can reserve. The showed cars have battery charge level more or equal 20%, are not reserved and within a 500m range from the user's current provided location or from an address specified.
[D24] **2 minutes wait**: time elapsed from the locking of the car and the moment in which the system checks whether the car has been plugged or not and computes the total amount of discounts and recharges

Abbreviations:

DB : Database
DCS : Data Checking System
MSO : Money Saving Option
MT : Maintenance Team
PS : Payment System

1.4 List of Reference Documents:

These are the documents we referred to during the composition of the ITPD

- The project description
- The RASD (Requirements Analysis and Specification Document)
- The DD (Design Document)

2 INTEGRATION STRATEGY

2.1 Entry Criteria

Starting the integration of two components, it has been assumed that each of them (only the required functionalities) had already been developed and unit tested on its own.
Furthermore, a documentation of the components development is expected and supposed coherent with what written down in the RASD and DD (considered as completed yet).
The interaction with

- the Database and the DBInterface
- the Car and the Car Unlock Command Dispatcher
- the Car and the Ride Cost Dispatcher

are considered as already tested by the (third-parties) producers.

2.2 Elements to be Integrated

The components covered by the integration test are listed here and are scheduled by their identifier that has scope in this document only:

1. Visitor interface (web)
2. User interface (web + mobile application)
3. Car (third party component)
4. Maintenance team system (third party component)
5. Data checking system (third party component)
6. Payment system (third party component)
7. Payment Handler
8. Database
9. DB interface
10. Visitor’s registration handler
11. Visitor’s registration manager
12. Data checking system handler
13. Email Dispatcher
14. User’s interaction handler
15. Login Manager
16. Car’s interaction handler
17. Maintenance team handler
18. Session Manager (subsystem composed by)
    a. Reservation Manager
    b. Ride Manager
    c. Push Notification Dispatcher
    d. Car Lock Command Dispatcher
    e. Ride Cost Dispatcher
    f. Money Saving Option Manager
    g. Discount Computation
    h. Ride Cost Dispatcher (third party component)

2.3 Integration Testing Strategy

The chosen integration test approach is the one that follows the functional thread testing, supposing that a subsystem (Session Manager) would be tested apart.
This choice lies on the fact that the whole system may be seen as several functionality threads that can be mainly tested on their own. Therefore, these four main threads are:

- Registration phase, tested with a bottom-up approach, considers all the events due to a triggering of a new account registration registration request by a visitor
- Login phase, tested with a bottom-up approach, intends the trivial login procedure that is considered as standard and then so straightforward
- Reservation phase, tested with a bottom-up approach, cares about the reservation of a car by a user
- Drive phase, tested with a bottom-up approach, concerns about the actual driving of a car by a user

These three phases have some components in common (see DBInterface as an example) that, being those functionalities partitioned with respect to the several threads, if the test covers all the three phases listed above, then each component can be considered tested at all.
Here the "Session Manager" subsystem can be considered and tested on its own in order to integrate all its components seen just as a block during the testing on the whole system. In the following diagrams of this document, he components of the "Session Manager" subsystem will be wrapped in dashed frames.

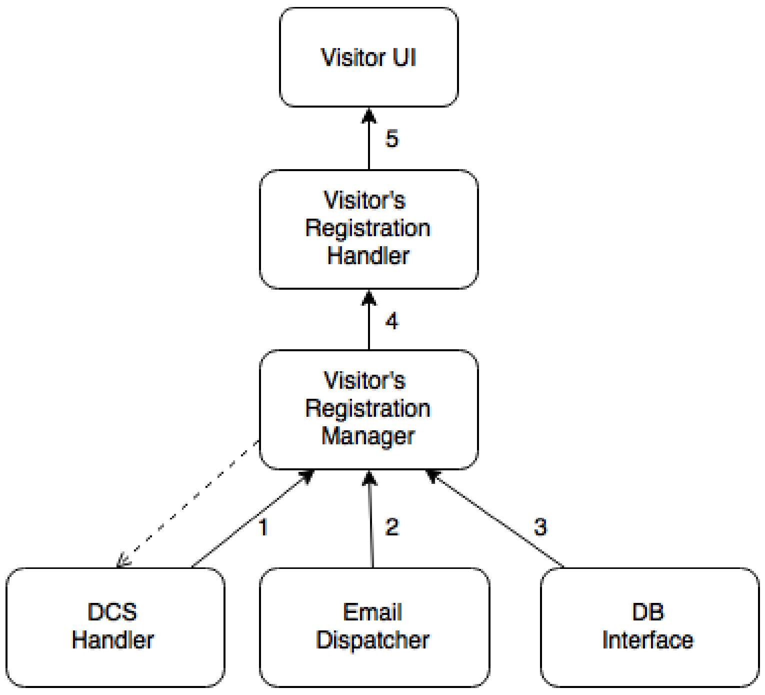
The bottom-up approach testing has been preferred because the easiness of building a driver (compared to a stub) and because of the great amount of stubs that would have been implemented if a top-down approach had been chosen.

A further advance of the thread testing approach is that each test could actually be tested in parallel with the others.

2.4 Sequence of Component/Function Integration

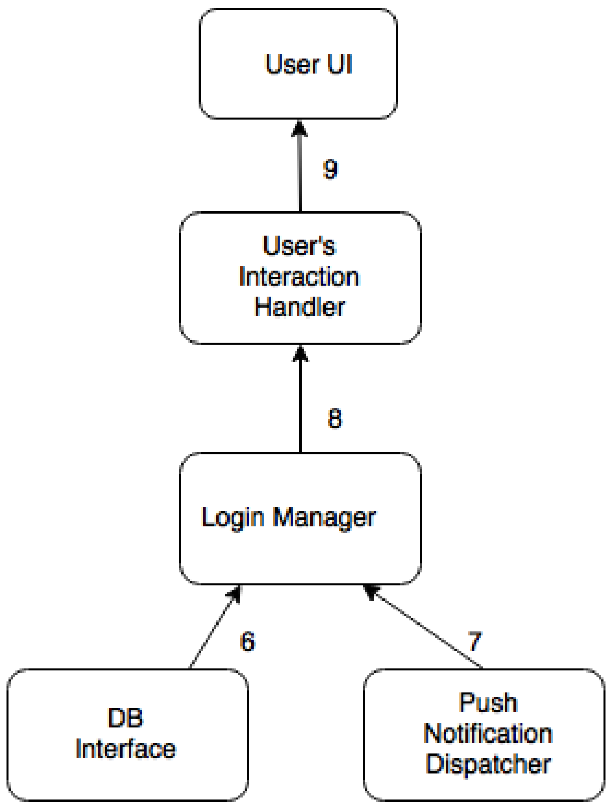
Here it is shown in which order the components are going to be integrated.
First there is a description on how the outermost components are paired, then we go into the details of the integrations in the session manager subsystem.

2.4.1 Registration thread

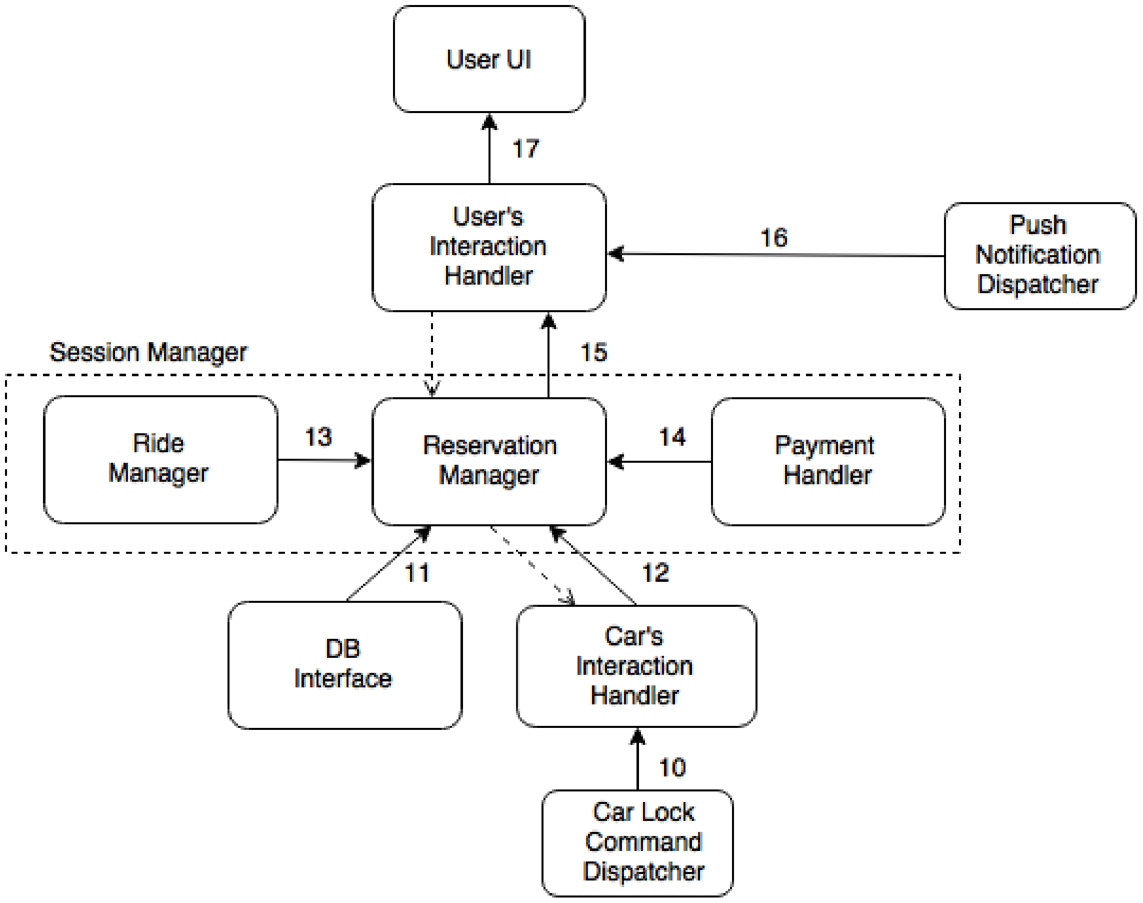


Visitor UI
5
Visitor's Registration Handler
4
Visitor's Registration Manager
1
2
3
DCS Handler
Email Dispatcher
DB Interface

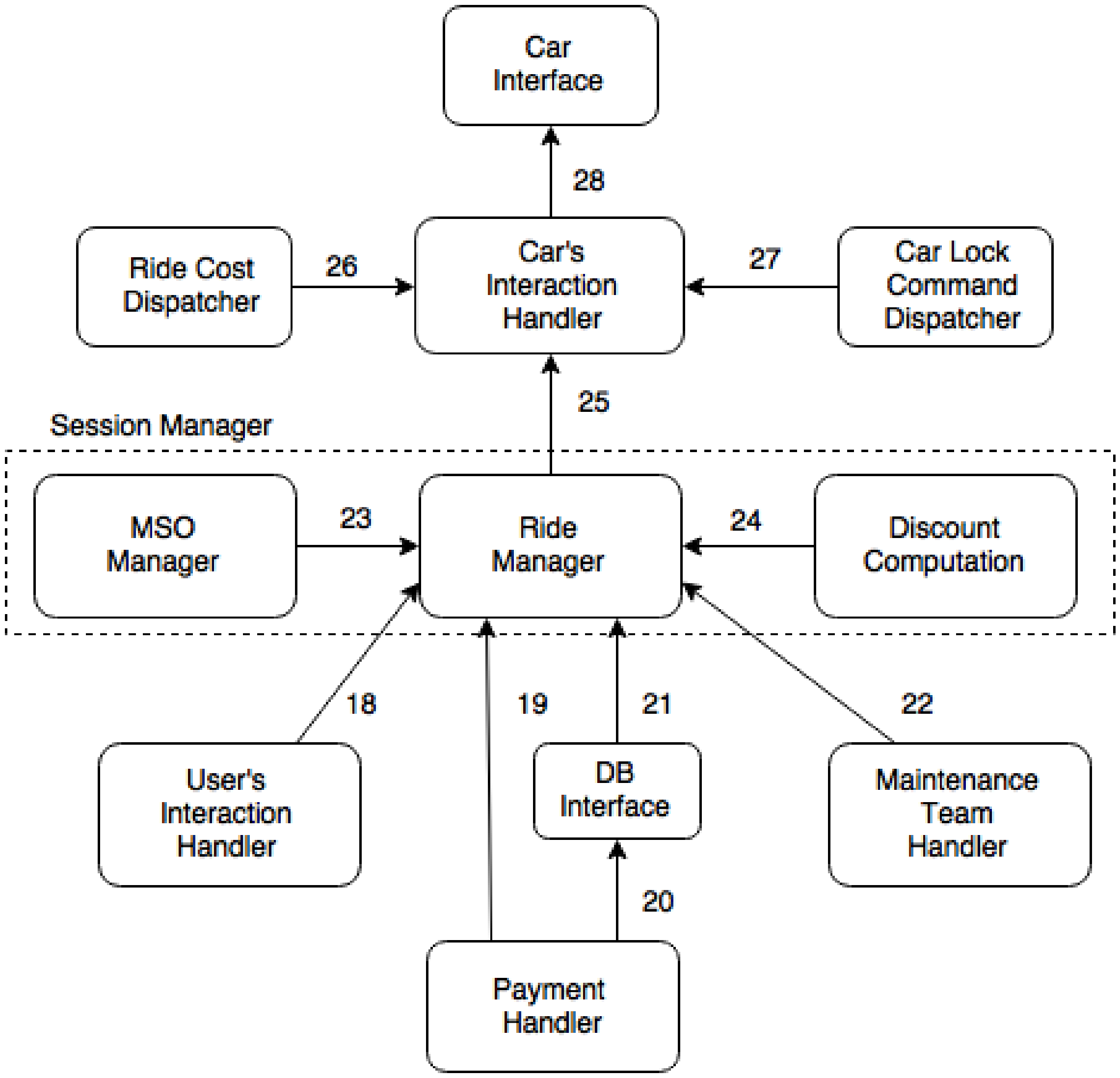
2.4.2 Login thread



2.4.3 Reservation thread

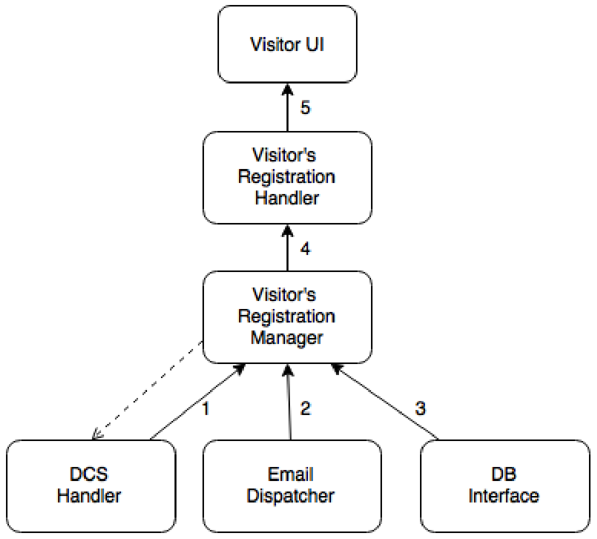

User UI
17
User's Interaction Handler
16
Push Notification Dispatcher
Session Manager
15
Ride Manager
13
Reservation Manager
14
Payment Handler
11
12
DB Interface
Car's Interaction Handler
10
Car Lock Command Dispatcher

2.4.4 Drive thread



3 Individual Steps and Test Description

3.1 Visitor's registration thread


Visitor UI
5
Visitor's Registration Handler
4
Visitor's Registration Manager
1
2
3
DCS Handler
Email Dispatcher
DB Interface

Test Case Identifier	I1
Test Item	DCS Handler ➔ Visitor's Registration Manager
Environmental Needs	-
Input Specification	**Output Specification**
Generate valid visitor's data	Check that all the required methods have been correctly called and so the data checking procedure has produced a positive feedback
Generate an instance of data that contains at least one wrong field	Check that all the required methods have been correctly called and that the feedback is negative
Generate not all the required data	Check that all the required methods have been correctly called and that the feedback is negative

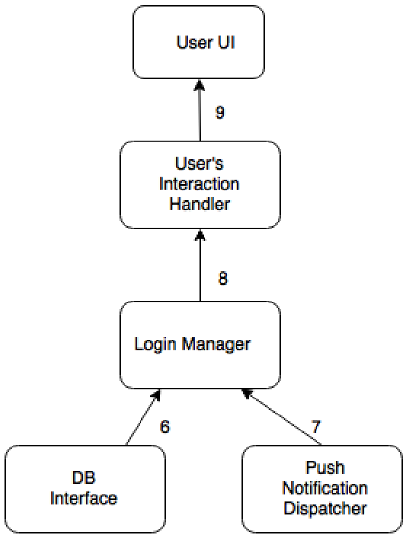
Test Case Identifier	I2
Test Item	Email Dispatcher ➔ Visitor's registration Manager
Environmental Needs	-
Input Specification	**Output Specification**
Trigger an error/password email forwarding to a specified user	Check the Email Dispatcher behavior (the actually sending of the email to the correct user account)

Test Case Identifier	I3
Test Item	DB Interface ➔ Visitor’s registration Manager
Environmental Needs	-
Input Specification	**Output Specification**
Generate all the possible interactions with the database (store, update, read)	Check the feedback (and the DB)

Test Case Identifier	I4
Test Item	Visitor’s Registration Manager ➔ Visitor’s Registration Handler
Environmental Needs	I1, I2, I3 successful
Input Specification	**Output Specification**
Generate a registration request	Check that the request be properly handled

Test Case Identifier	I5
Test Item	Visitor’s Registration Handler ➔ Visitor UI
Environmental Needs	I4 successful
Input Specification	**Output Specification**
Generate a registration request	Check that the request be properly handled

3.2 Login thread



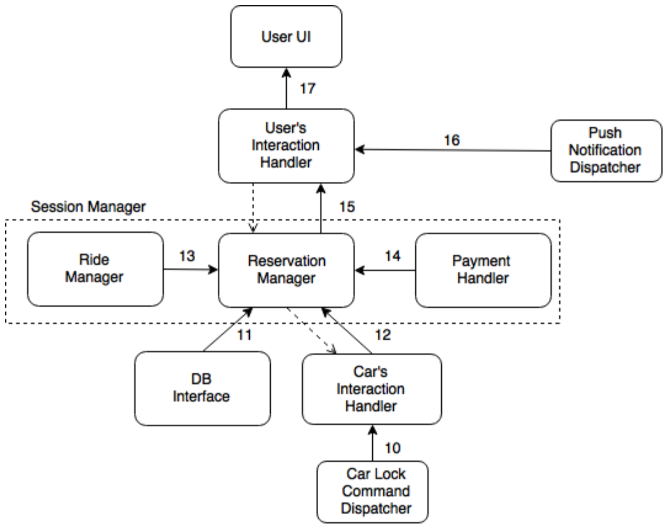
Test Case Identifier	I6
Test Item	DB Interface ➔ Login Manager
Environmental Needs	-
Input Specification	**Output Specification**
Generate all the possible interactions with the database (read)	Check the feedback (and the DB)

Test Case Identifier	I7
Test Item	Push Notification Dispatcher ➔ Login Manager
Environmental Needs	-
Input Specification	**Output Specification**
Generate a push feedback to be sent to a "fake" user	Check that the "fake" user be getting it

Test Case Identifier	I8
Test Item	Login Manager ➔ User's Interaction Handler
Environmental Needs	I6, I7 successful
Input Specification	**Output Specification**
Generate a login request	Check that the request be properly handled

Test Case Identifier	I9
Test Item	User’s Interaction Handler ➔ User UI
Environmental Needs	I8 successful
Input Specification	**Output Specification**
Generate a login request	Check that the request be properly handled

3.3 Reservation thread


User UI
17
User's Interaction Handler
16
Push Notification Dispatcher
Session Manager
15
Ride Manager
13
Reservation Manager
14
Payment Handler
11
12
DB Interface
Car's Interaction Handler
10
Car Lock Command Dispatcher

Test Case Identifier	I10
Test Item	Car Lock Command Dispatcher ➔ Car's Interaction Handler
Environmental Needs	-
Input Specification	**Output Specification**
Trigger an unlock signal for a monitored car	Check that the action be correctly performed

Test Case Identifier	I11
Test Item	DB Interface ➔ Reservation Manager
Environmental Needs	-
Input Specification	**Output Specification**
Generate all the possible interactions with the database (store, update, read)	Check the feedback (and the DB)

Test Case Identifier	I12
Test Item	Car's Interaction Handler ➔ Reservation Manager
Environmental Needs	I10 successful
Input Specification	**Output Specification**
Trigger an unlock signal for a monitored car	Check that the action be correctly received by the Car Lock Command Dispatcher
Generate car data for the beginning of the ride	Check that the data are properly handled

Test Case Identifier	I13
Test Item	Ride Manager ➔ Reservation Manager
Environmental Needs	-
Input Specification	**Output Specification**
Signal the beginning of a new ride	Check that the Ride Manager be reacting properly

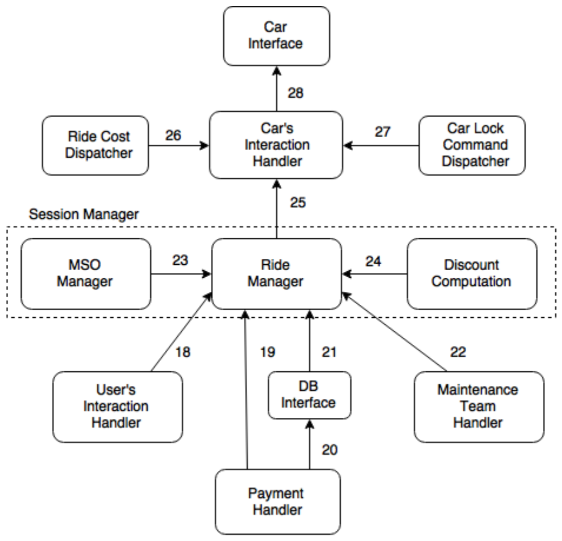
Test Case Identifier	I14
Test Item	Payment Handler ➔ Reservation Manager
Environmental Needs	-
Input Specification	**Output Specification**
Trigger a pledge checking	Check the feedback coherence
Trigger a payment request	Monitor the transaction's feedback

Test Case Identifier	I15
Test Item	Reservation Manager ➔ User's Interaction Handler
Environmental Needs	I11, I12, I13, I14 successful
Input Specification	**Output Specification**
Generates all the possible interactions	Monitor how the system reacts

Test Case Identifier	I16
Test Item	Push Notification Dispatcher ➔ User's Interaction Handler
Environmental Needs	-
Input Specification	**Output Specification**
Generate a push feedback to be sent to a "fake" user	Check that the "fake" user be getting it

Test Case Identifier	I17
Test Item	User's Interaction Handler ➔ User's UI
Environmental Needs	I15, I16 successful
Input Specification	**Output Specification**
Generates all the possible interactions	Monitor how the system reacts

3.4 Drive thread



Test Case Identifier	I18
Test Item	User's Interaction Handler ➔ Ride Manager
Environmental Needs	-
Input Specification	**Output Specification**
Generates all the possible interactions	Monitor how the system reacts

Test Case Identifier	I19
Test Item	Payment Handler ➔ Ride Manager
Environmental Needs	-
Input Specification	**Output Specification**
Trigger a payment request	Monitor the transaction's feedback

Test Case Identifier	I20
Test Item	Payment Handler ➔ DB Interface
Environmental Needs	-
Input Specification	**Output Specification**
Generate the interaction with the database (update)	Check the feedback (and the DB)

Test Case Identifier	I21
Test Item	DB Interface ➔ Ride Manager
Environmental Needs	I20 successful

Input Specification	**Output Specification**
Generate all the possible interactions with the database (store, update, read)	Check the feedback (and the DB)

Test Case Identifier	I22
Test Item	Maintenance Team Handler ➔ Ride Manager
Environmental Needs	-
Input Specification	**Output Specification**
Trigger a Maintenance Team intervention	Check the feedback

Test Case Identifier	I23
Test Item	MSO Manager ➔ Ride Manager
Environmental Needs	-
Input Specification	**Output Specification**
Trigger a Money Saving Option request	Check the feedback

Test Case Identifier	I24
Test Item	Discount Computation ➔ Ride Manager
Environmental Needs	-
Input Specification	**Output Specification**
Trigger a discount computation request	Check the feedback

Test Case Identifier	I25
Test Item	Ride Manager ➔ Car’s Interaction Handler
Environmental Needs	I18, I19, I21, I22 successful
Input Specification	**Output Specification**
Generates all the possible interactions	Monitor how the system reacts

Test Case Identifier	I26
Test Item	Ride Cost Dispatcher ➔ Car’s Interaction Handler
Environmental Needs	-
Input Specification	**Output Specification**
Keep sending cost value	Check that the cost be displayed in the car’s screen

Test Case Identifier	I27
Test Item	Car Lock Command Dispatcher ➔ Car’s Interaction Handler
Environmental Needs	-
Input Specification	**Output Specification**
Trigger a lock signal for a monitored car	Check that the action be correctly received by the Car Lock Command Dispatcher

Test Case Identifier	I28
Test Item	Car’s Interaction Handler ➔ Car Interface
Environmental Needs	I25, I26, I27 successful
Input Specification	**Output Specification**
Generates all the possible interactions	Monitor how the system reacts

4 Tools and Test Equipment Required

Supposing that the software has already been validated, it is being verified in an automated way despite some parts manually tested.
The manual testing could replace the automated one where who executes the test considers that opportune (e.g. where the developing of the whole context for the testing is way too expensive than carrying it out manually).

The following testing (software) tools are used:

- *Mockito* (*http://mockito.github.io*) provides functionalities for the integration test beside the ones related to the unit testing.
  It provides a framework that supports the creation of stubs for software scaffolding (for simulating part of software not implemented yet) or mocks (useful to simulate the behavior of an external system) where the integration test strategy requires it.
  Since the Unit testing is supposed to have already been carried out before the integration test starts and since no stubs are needed (the approach is bottom-up and all the modules should have been developed before the testing starts), this software could only be used here to mock external systems during the testing (in order not to involve them just for testing purposes and in order to have a faster response too).
- *JUnit* is a framework used for unit testing but here it is supposed to have been already carried out.
- *Arquillian* (*http://arquillian.org*) is an integration testing framework useful to test the interaction of a component with its container.
  This framework can be used here to test that the resource injection or the interaction with the database is executed correctly.
- *JMeter* (*http://jmeter.apache.org*) is a load testing tool for analyzing and measuring performances and then for assessing nonfunctional requirements.
  It can simulate a load (by means of simulated user requests) on the server and monitor the response. Different test plans can be set and here, for example, the registration phase, the research phase, the reservation phase can be tested modeling the number of user accessing concurrently and at which frequency they would do that.

Note that the drivers (since in this case not so complex) could be written as simple code without using some support framework.

If during the software development some delay is encountered, the constraint of having the component completely developed (over a certain functionality thread) before the testing phase starts some stub or driver could be written with the listed tools to replace the missing code and avoid the delay of the whole testing procedure on that thread.

5 Program Stubs and Test Data Required

Since thread testing and bottom up approach has been followed integrating a subsystem too, only drivers have to be implemented.
Mind you that a component could appear different times as driver but being a thread testing, only the functionalities related to that thread should be implemented.
Here follows the listing of the needed drivers divided by each thread:

Registration thread:

- VisitorUI that is a driver for
  - Visitor’s Registration Handler
- Visitor’s Registration Handler that is a driver for
  - Visitor’s Registration Manager
- Visitor’s Registration Manager that is a driver for
  - DCS Handler
  - Email Dispatcher
  - DB Interface

Note: Actually DCS Handler would be a driver for Visitor’s Registration Manager too because the first calls a function of the latter. This should be managed when the components are coupled and tested.

Login thread:

- UserUI that is a driver for
  - User’s Interaction Handler
- User’s Interaction Handler that is a driver for
  - Login Manager
- Login Manager that is a driver for
  - DB Interface
  - Push Notification Dispatcher

Reservation thread:

- UserUI that is a driver for
    - User’s Interaction Handler
- User’s Interaction Handler that is a driver for
    - Reservation Manager (contained in the Session Manager subsystem)
    - Push Notification Dispatcher

Note: Actually Reservation Manager would be a driver for User’s Interaction Handler too because the first calls a function of the latter. This should be managed when the components are coupled and tested.

- Reservation Manager (contained in the Session Manager subsystem) that is a driver for
    - Ride Manager (contained in the Session Manager subsystem)
    - Payment Handler (contained in the Session Manager subsystem)
    - DB Interface
    - Car’s Interaction Handler

Note: Actually Car’s Interaction Handler would be a driver for Reservation Manager too because the first calls a function of the latter. This should be managed when the components are coupled and tested.

- Car’s interaction Handler that is a driver for
    - Car Lock Command Dispatcher

Drive thread:

- Car Interface that is a driver for
    - Car’s Interaction Handler
- Car’s Interaction Handler that is a driver for
    - Ride Manager (contained in the Session Manager subsystem)
    - Ride Cost Dispatcher
    - Car Lock Command Dispatcher
- Ride Manager (contained in the Session Manager subsystem) that is a driver for
    - MSO Manager (contained in the Session Manager subsystem)
    - Discount Computation (contained in the Session Manager subsystem)
    - DB Interface
    - User’s Interaction Handler
    - Maintenance Team Handler
    - Payment Handler
- Payment Handler that is a driver for
    - DB Interface

6 Effort Spent

- Giulia Pennati – 15 h
- Paola Sanfilippo – 22 h
- Marco Siracusa – 21 h