POLITECNICO
MILANO 1863

Computer Science and Engineering

A.A. 2016/2017

Software Engineering 2 Project:

“PowerEnJoy”

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

January 15, 2017

Prof.Luca Mottola

Matteo Michele Piazzolla Matr. 878554

Andrea Millimaggi Matr. 876062

Revision History

Revision	**Date**	**Author(s)**	**Description**
1.0	13/11/16	Piazzolla Millimaggi	First document issue

Contents

Todo list

List of Figures

1 INTRODUCTION

1.1 Purpose and Scope

1.1.1 Purpose

This is the Integration Test Plan Document for the PowerEnJoy service. Its aim is to describe the integration tests that have to be done by the people in charge of testing in the development team, in order to verify that all the components work correctly and interact with each other properly as well.

1.1.2 Scope

PowerEnJoy is a digital system for the management of a car-sharing service that only employs electric cars. In particular the aim is to develop a mobile application that allows the user to pick up and use electric cars in the areas reached by the service.

1.2 List of Definitions and Abbreviations

1.2.1 Acronyms

- RASD: Requirement Analysis and Specification Document.
- DD: Design Document.
- DBMS: Database Management System.
- DB: Database.
- EJB: Enterprise Java Bean
- GUI: Graphical User Interface
- GPS: Global Positioning System

1.2.2 Definitions

1.2.3 Abbreviations

- IT-n: Integration Test n.

1.3 List of Reference Documents

- PowerEnJoy RASD.
- PowerEnJoy DD. **NOTE:** The reader must refer to the version 1.1 of the DD.

2 INTEGRATION STRATEGY

2.1 Entry Criteria

Before the integration testing can begin, both the RASD and the DD must be completed. Furthermore at least the 80% of the lines of the code of all components must have been tested with a Unit Test. We strongly suggest the Junit testing framework for Unit Testing.

2.2 Elements to be Integrated

As we have described in the DD, the main components of the system are:

- Web Client GUI
- Mobile Client GUI
- Driver Client GUI
- Service Manager
- HTTP Server
- Request Handler
- DBMS

In turn, the Service Manager is composed of:

- Car Manager
- User Manager
- Position Manager
- Fee Manager

For a further description of the components refer to the Design Document.
Of course, because before the Service Manager can be integrated with other components, the components that compose it have to be integrated.

2.3 Integration Testing Strategy

We have chosen for the Integration Testing a **bottom up** approach. This means that the integration will regard first the low-level components. For this type of testing, because we start from the low level components, we will need some **drivers**. These are temporary modules that simulate the behaviour of the higher level components and do the calls of the functions of lower level components.
That reason why we have chosen the bottom-up approach are:

- The lowest level component of our system is the DBMS, impletemented with MySQL 5.6. MySQL is a well-known software component that is available on the market. So the DBMS is ready to use and is a robust starting point for the Integration Testing. Starting from this we can begin to integrate the components of the Backend of our system, that essentialy is the Service Manager component.
- The low level components that compose the Backend of our service are the biggest part of our system and all the other components depends on them. Because of this importance of the low level components, the best choice is to start from them and, when it's sure that they work well together, integrate the other components.

When the integration testing regards the Service Manager, the bottom up approach will be mixed with the **functional testing** approach. With functional testing, not the entire components are tested, but only some functionalities of them. In our case, when the components that compose the Service Manager will be tested together with the Service, only some functionalities of the Service Manager will be tested, that are the ones that call the specific subcomponent that is being tested.

2.4 Sequence of Component/Function Integration

2.4.1 Software Integration Sequence

Because we have chosen a bottom up approach, the first components to be tested are the low level ones, so the ones that compose the Service Manager. After we have completed the integration for the subcomponents of the Service Manager, we will integrate the components that are directly interfaced with that subsystem, that are the Car Manager and the Request Manager. At the end, we will integrate the remaining front-end components, that are the Web Client and the Mobile Client. At each step of the integration, a driver for the new component to be integrated has to be constructed. When the integration testing regarding that component stops, the driver is replaced with the actual component.

2.4.1.1 Integration of the components of the Service Manager The strategy that we have chosen to follow for the integration testing between the subcomponents of the Service Manager and the Service Manager itself is a functional one. So it won't constructed a driver for all the functionalities of the Service Manager, but at every step a driver for the single functionality regarding the subcomponent that is being tested is constructed.

Request Manager and User Manager The first components to be tested with the Service Manager are the ones that have a relation with the DBMS: the User Manager and the Reservation Manager. The first one has to manipulate data regarding the users and their accounts and the second one has

to manipulate data regarding the cars. As we mentioned in the previous section, the DBMS is a component ready to use, so the User Manager and the Reservation Manager will be directly tested with the DBMS, without the construction of drivers. The tests with the DBMS have to precede those with the Service Manager
The order in which the Reservation Manager and the User Manager are chosen doesn't matter.

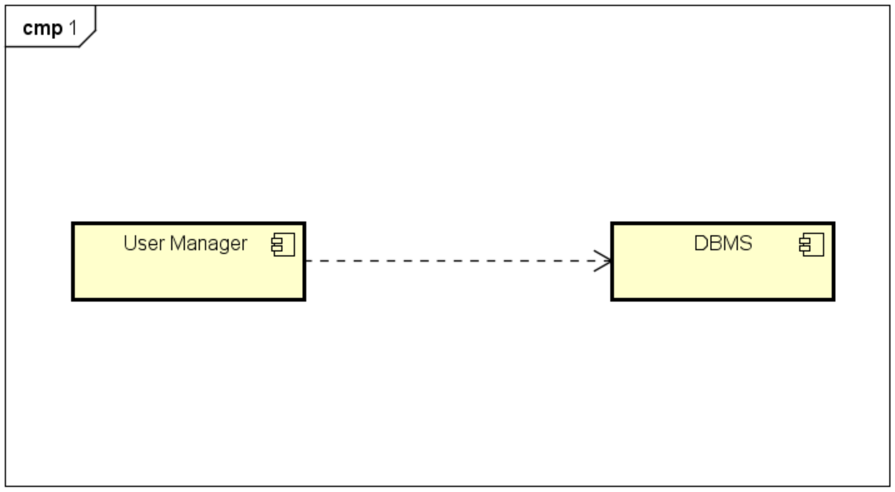

Figure 1: Integration of User Manager and DBMS

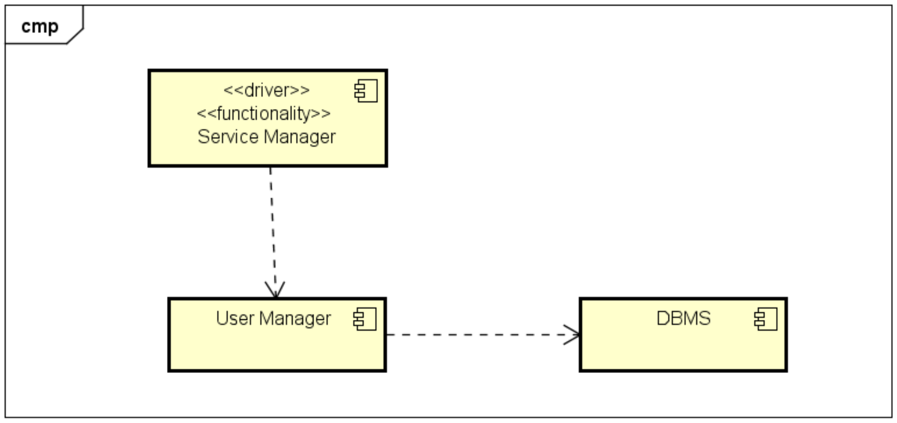

Figure 2: Integration of User Manager and DBMS with Service Manager

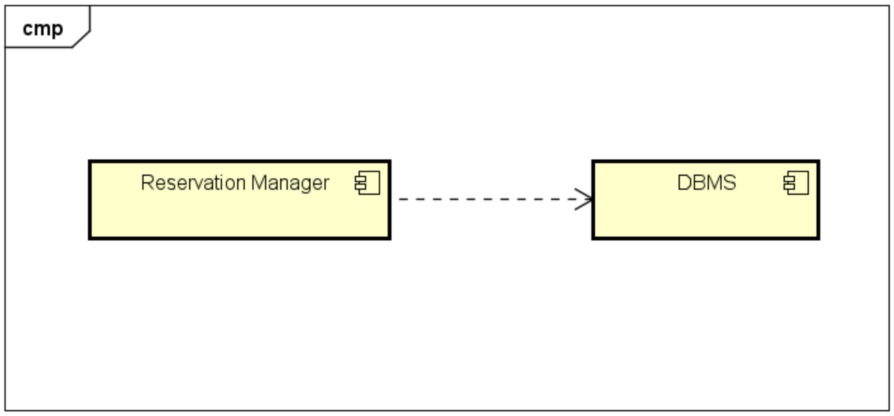

Figure 3: Integration of Reservation Manager and DBMS

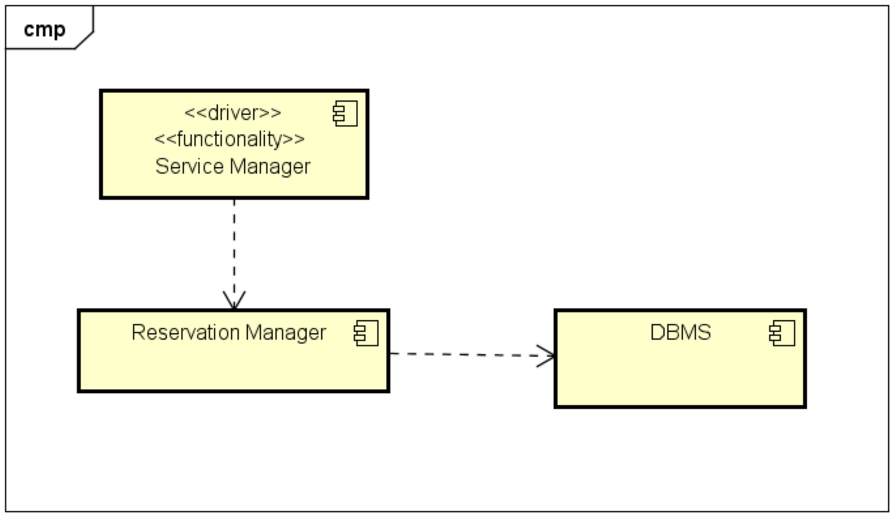

Figure 4: Integration of Reservation Manager and DBMS with Service Manager

Fee Manager The Fee Manager has to interact with payment systems, because it has the task of calculating the fee and charging the user. Because it would be hard and expensive using a real payment system, a stub simulating its functionalities is used. The tests with the Payment System stub have to precede the tests with the Service Manager.

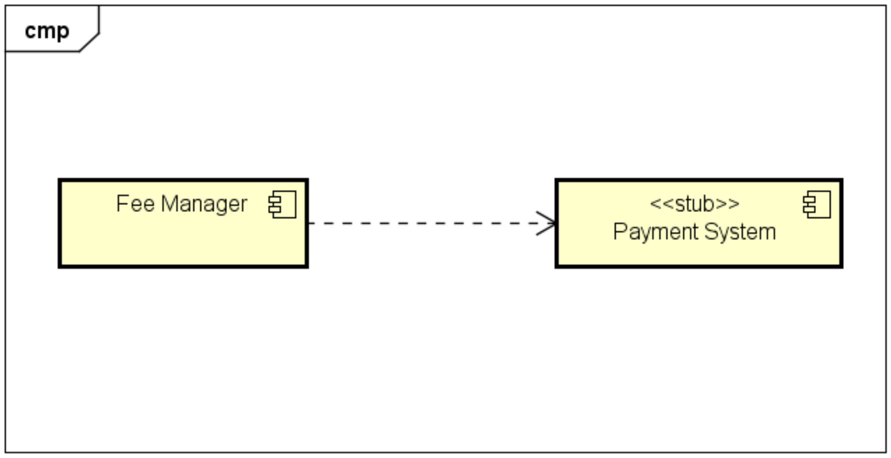

Figure 5: Integration of Fee Manager and Payment System

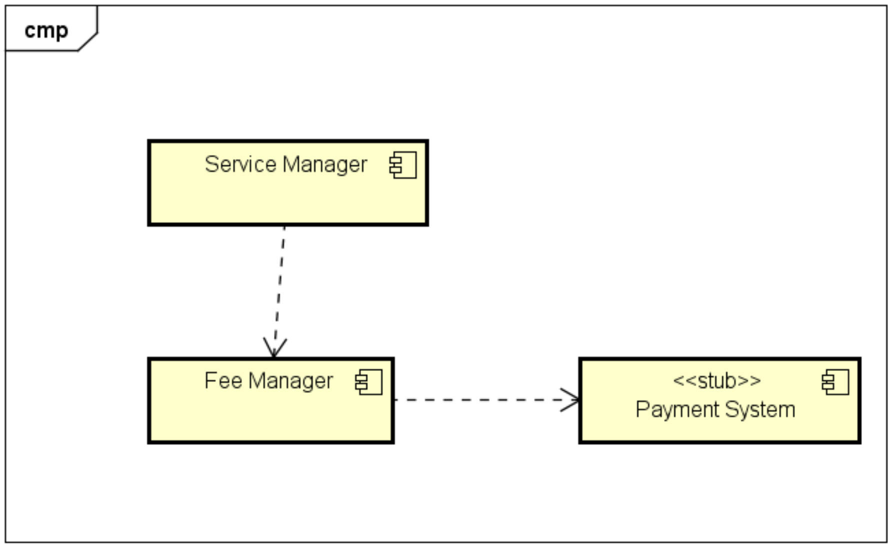

Figure 6: Integration of Fee Manager and Payment System with Service Manager

Position Manager In addition to the Service Manager, the Position Manager has to interact also with the GPS software. Like the DBMS, the GPS software is a component found on the market and ready to use and so it can be tested directly with the Position Manager, without the construction of a driver. The tests with the GPS component have to precede the tests with the Service Manager.

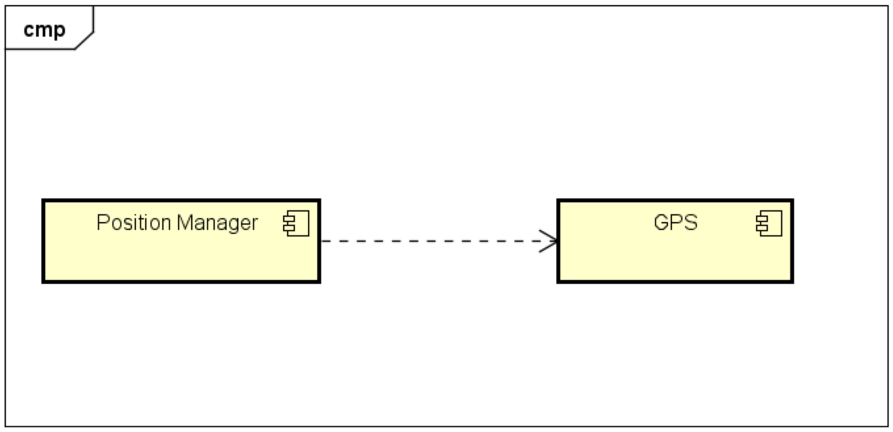

Figure 7: Integration of Position Managere and GPS

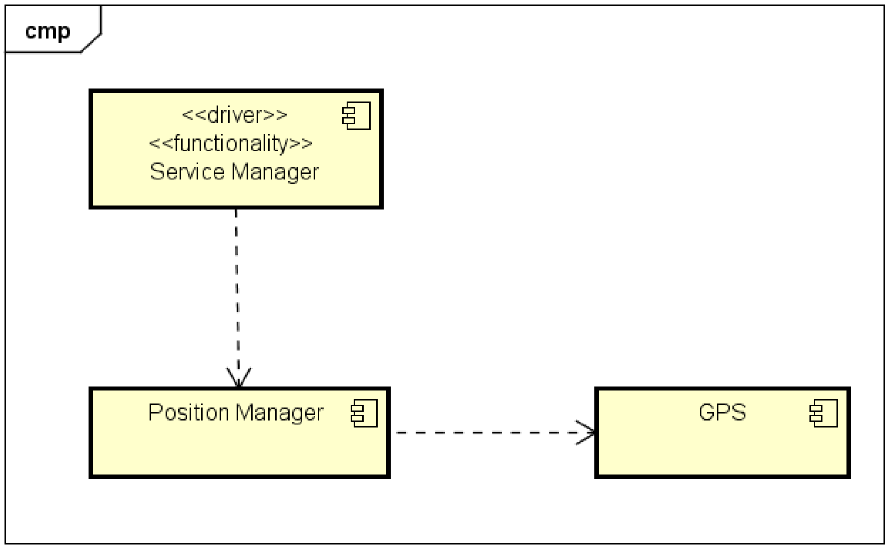

Figure 8: Integration of Position Manager and GPS with Service Manager

Car Manager The Car Manager is interfaced only with the Service Manager, so it is tested only with the driver of that component:

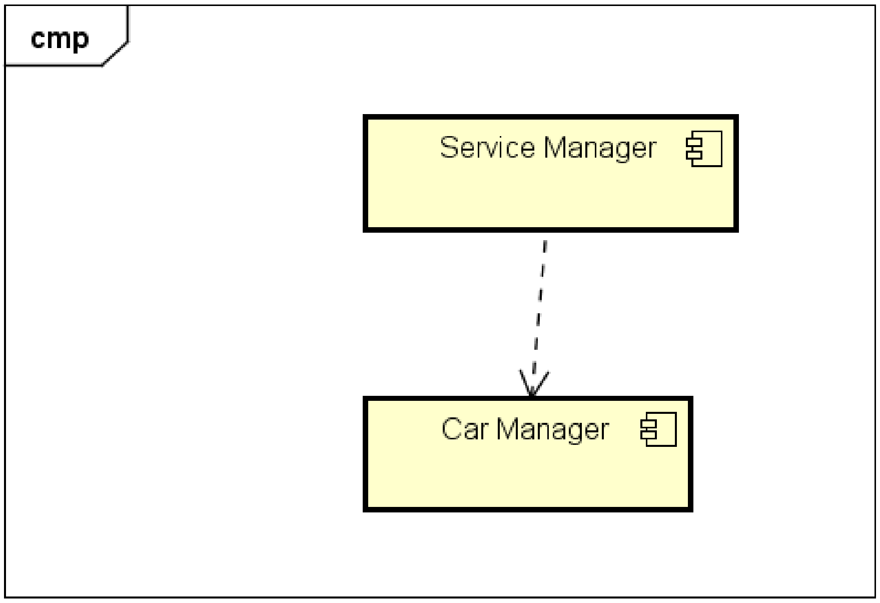

Figure 9: Integration of Car Manager and Service Manager

2.4.1.2 Components interacting with the Service Manager

After the integration tests on the subcomponents of the System Manager are completed, the Service Manager is ready to be integrated with higher level components. The testing of the subcomponents of the Service Manager ends here because those subcomponents implement functionalities that are independent with each other and so is sufficient to test them singularly together with the Service Manager and the external components they need.
So, after the completion of the test on the Service Manager, its driver is substituted by the actual component and this component will have to be tested with the Request Handler and the Car Manager, that are the components directly interfaced with the Service Manager. First we choose to test the Service Manager with the Request Handler driver, because the Request Handler is the most critical component:

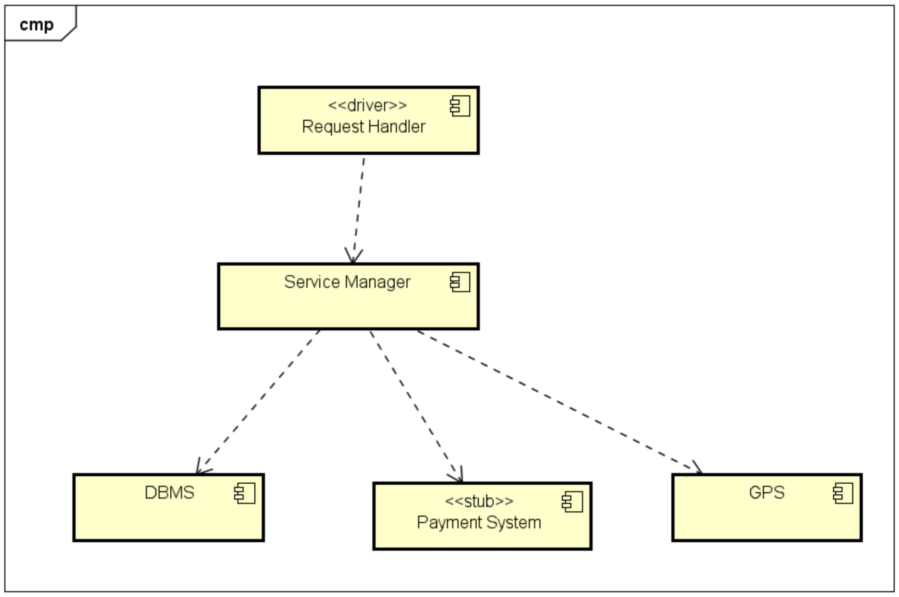

Figure 10: Integration of Request Handler with Service Manager, DBMS, GPS and Payment System

Then, the Service Manager is integrated with the OnBoard Device driver:

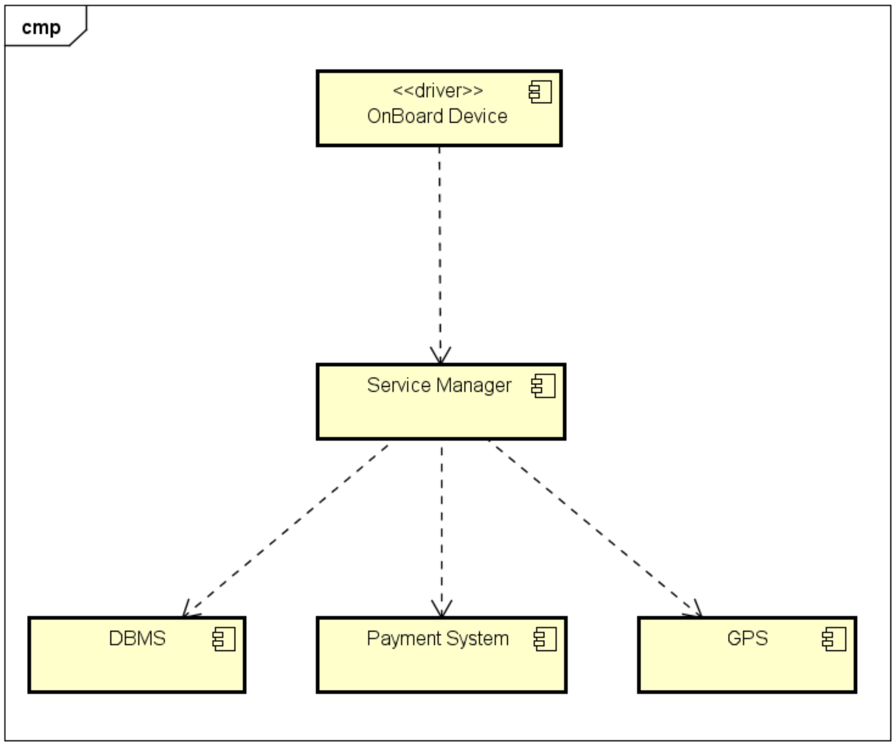

Figure 11: Integration of OnBoard Device with Service Manager, DBMS, GPS and Payment System

2.4.1.3 HTTP Server

After the integration of the Request Handler and OnBoard Device, we can test the resulting subsystem with the HTTP Server driver. In particular the focus is on the interaction between the Request Handler and the HTTP Server:

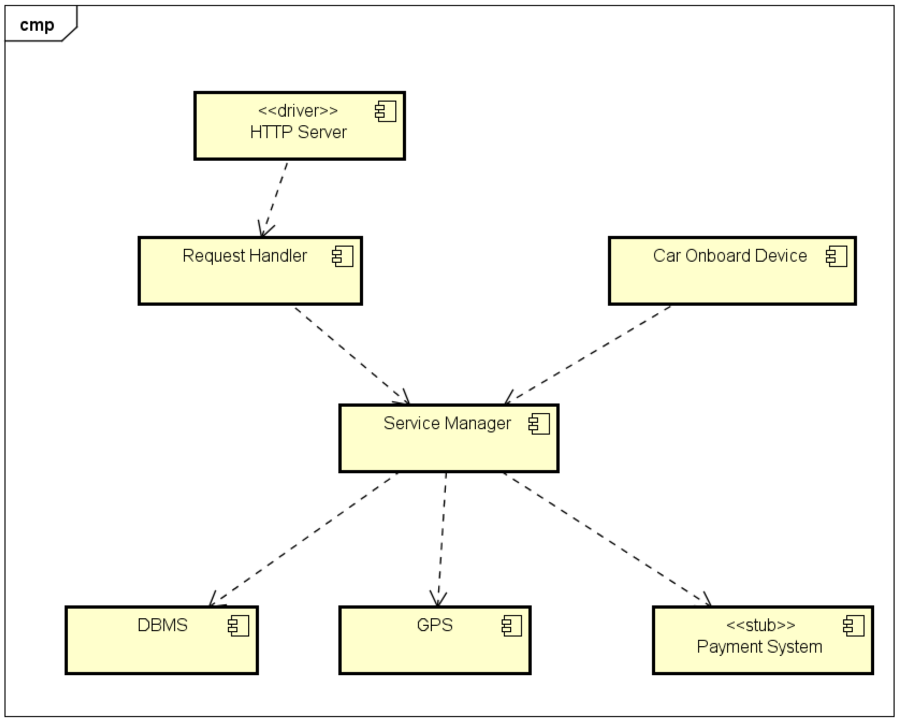

Figure 12: Integration of HTTP Server and Subsystem including Request Handler, Service Manager, OnBoard Device, DBMS, GPS and Payment System

2.4.1.4 Frontend Components

In the end, when the HTTP Server driver is substituted by the real component and this is tested with the drivers of the remaining two frontend components: the Mobile Client and the Web Client. The order in which this two components are chosen doesn't matter:

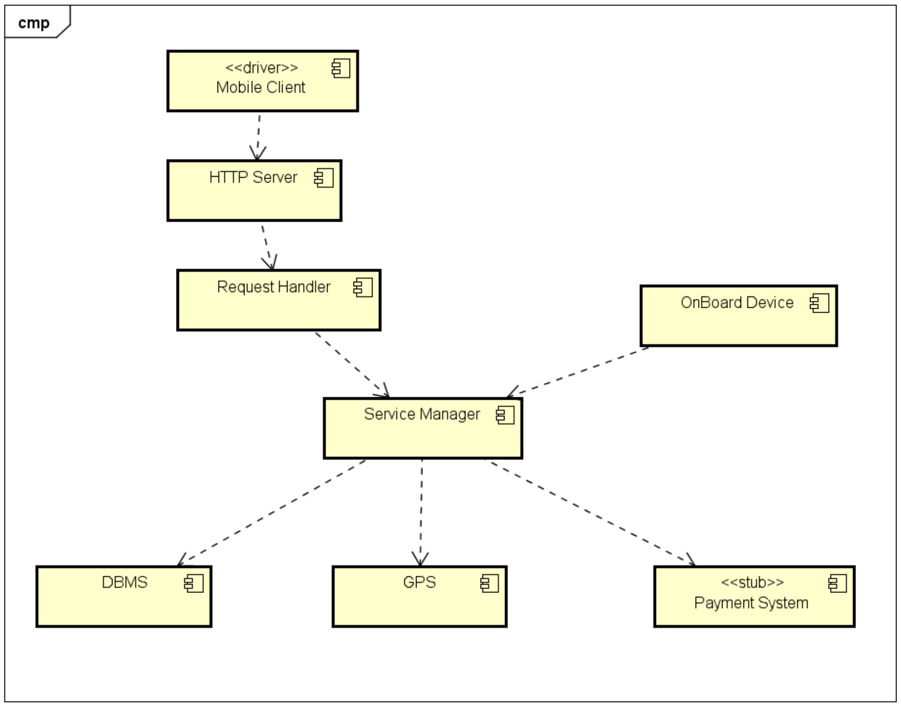

Figure 13: Integration of Mobile Client and Subsystem including HTTP Server, Request Handler, Service Manager, OnBoard Device, GPS, DBMS and Payment System

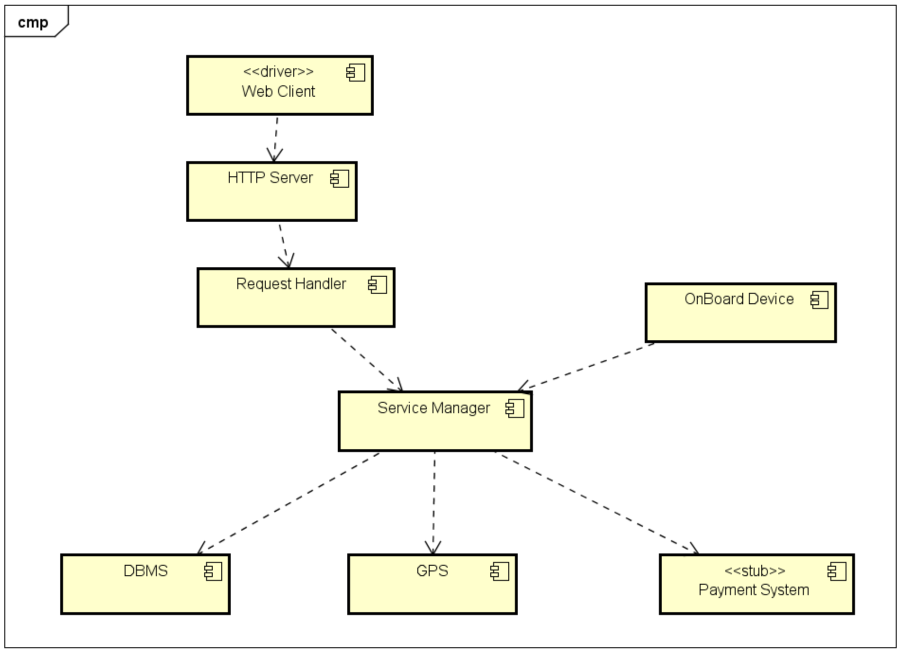

Figure 14: Integration of Web Client and Subsystem including HTTP Server, Request Handler, Service Manager, OnBoard Device, DBMS, GPS and Payment System

After these two integration tests, the all components of the system can be tested together.

3 INDIVIDUAL STEPS AND TEST DESCRIPTION

All this tests can be split in different and more specific tests.

Test Case Identifier	IT-1
Test Item(s)	User Manager -> DMBS
Input Specification	Create a method call that create users
Output Specification	Users data correctly present in the DB
Environmental Needs	Mock of some users data
Test Description	Test that verify users creation and check if the user is already in the DB

Test Case Identifier	IT-2
Test Item(s)	User Manager -> DMBS
Input Specification	Create a method call that create a new user
Output Specification	Users data presence in the DB
Environmental Needs	Mock of some user data
Test Description	Test that verify user data insert and error handling for already present user

Test Case Identifier	IT-3
Test Item(s)	User Manager -> DMBS
Input Specification	Create a method call that update for user
Output Specification	Users data presence in the DB
Environmental Needs	Mock of some user data
Test Description	Test that verify user data update and error handling for unregistered user

Test Case Identifier	IT-4
Test Item(s)	User Manager -> DMBS
Input Specification	Create a method call that delete user
Output Specification	User information no longer in the DB
Environmental Needs	Mock of some user data
Test Description	Test that verify user deletion and error handling for unregistered user

Test Case Identifier	IT-5
Test Item(s)	ServiceManager -> User Manager -> DBMS
Input Specification	Create a method call that check for registered and unregistered users
Output Specification	Correctness response for the registered user, error for unregistered
Environmental Needs	User data presence in the DB and a mock of the user data
Test Description	Test that verify users login and error handling for the unregistered user

Test Case Identifier	IT-6
Test Item(s)	Reservation Manager -> DMBS
Input Specification	Create a method call that create a reservation for an user
Output Specification	Reservation correctly created
Environmental Needs	Mock of some user and reservation data
Test Description	Test that verify that the reservation is created for an existing user and error handling for incorrect data

Test Case Identifier	IT-7
Test Item(s)	Reservation Manager -> DMBS
Input Specification	Create a method call that delete a reservation
Output Specification	Reservation correctly marked as deleted
Environmental Needs	Mock of some reservation data
Test Description	Test that verify that the reservation is marked as deleted but still present in the DB and error handling for incorrect data

Test Case Identifier	IT-8
Test Item(s)	Service Manager -> Fee Manager -> Payment System «stub»
Input Specification	Create a method call that calculate a fee for a reservation and call the Payment System
Output Specification	Payment accepted
Environmental Needs	Mock of some reservation data, pre-calculated fare and a stub for the Payment System
Test Description	Test check that the fare are correctly calculated and the payment system receive the payment request

Test Case Identifier	IT-9
Test Item(s)	Position Manager -> GPS
Input Specification	Create a method call that update the GPS position for a list of cars
Output Specification	Cars position correctly update
Environmental Needs	Mock of some cars object
Test Description	Test check if the Position Manager handle correctly the various format of GPS coordinate

Test Case Identifier	IT-10
Test Item(s)	Service Manager -> Position Manager -> GPS
Input Specification	Create a method call that return a list of cars inside a given radius list
Output Specification	The returned list are the same of the cars mock lists per radius
Environmental Needs	Mock of some cars object and a list of radius
Test Description	Test check if the Position Manager handle correctly the request

Test Case Identifier	IT-11
Test Item(s)	Service Manager -> Car Manager
Input Specification	Create a method call that request information about a car
Output Specification	Returned information are correct
Environmental Needs	Mock of a car object
Test Description	Test check if the Car Manager correctly handle request for car information

Test Case Identifier	IT-12
Test Item(s)	Service Manager -> Car Manager
Input Specification	Create a method call that update information about a car
Output Specification	Car information are correctly updated
Environmental Needs	Mock of a car object
Test Description	Test check if the Car Manager correctly handle update for car information

Test Case Identifier	IT-13
Test Item(s)	Request Handler -> Service Manager
Input Specification	Create different method calls for all the possible valid requests and an invalid request
Output Specification	Service Manager handle correctly all the requests and return an error for invalid request
Environmental Needs	mock data for all the valid request
Test Description	Test check if the service manager handle all the valid requests

Test Case Identifier	IT-14
Test Item(s)	OnBoard Device -> Service Manager
Input Specification	Phisically or virtually send data from the OnBoard Device to the Service Manager
Output Specification	Service Manager accept the connection
Environmental Needs	OnBoard Device or an OnBoard device simulator
Test Description	The test check the connection between the OnBoard Device and the System

Test Case Identifier	IT-15
Test Item(s)	HTTP Server -> Request Handler
Input Specification	All the possible CRUD REST Requests sent to the HTTP Server
Output Specification	Request Handler correctly parse all the request and return a valid response for all request
Environmental Needs	Mock of all the CRUD Requests and some bad formatted HTTP requests
Test Description	Test check for the REST parser

Test Case Identifier	IT-16
Test Item(s)	Mobile Client -> HTTP Server
Input Specification	Send all the possible REST Requests from the mobile application to the HTTP Server
Output Specification	Mobile application receive a correct response for all the request sent
Environmental Needs	Physical Device or a simulator to run PowerEnJoy mobile application
Test Description	Test check for the connection between the Mobile Application and the HTTP Server

Test Case Identifier	IT-17
Test Item(s)	Web Client -> HTTP Server
Input Specification	Send all the possible REST Requests from the website to the HTTP Server
Output Specification	The website receive a correct response for all the request sent and update the page
Environmental Needs	Web browser
Test Description	Test check for the connection between the website and the HTTP Server

4 TOOLS AND TEST EQUIPMENT REQUIRED

4.1 Testing Tools

For the testing phase of the development of our system, we have chosen four tools to support the work of testers:

- **JUnit:** This is a framework suitable for unit testing. JUnit will be used for the unit tests regarding the single components of the system. We remind that the unit testing must precede the integration testing phase.
- **Mockito:** This is a testing framework that allows the creation of mock components which simulate the behaviour of actual components of the system. Furthermore, it allows testing of the interaction between components. So this tool will be used for the creation of drivers and stubs needed for our integration testing and for testing interactions between components.
- **Arquillian:** This is a testing platform particularly suitable for the testing of EJBs(Enterprise Java Beans) and their interaction with their own Java EE Container. Because the subcomponents of the Service Manager are implemented with EJBs, Arquillian will be used to test them.
- **JMeter:** This is an application that allows to measure the performances of a server and to check its load tolerance. We will apply this tool to the HTTP Server to check its performance and its tolerance to heavy loads.

4.2 Test Equipment

The equipments needed for the right execution of integration testing includes all the devices with which users will use the PowerEnJoy service:

- A mobile phone running the latest version of the Android operating system.
- A mobile phone running the latest version of the iOS operating system.
- A computer running the Windows operating system. The latest versions of Google Chrome, Mozilla Firefox and Microsoft Edge browsers must be installed on this machine.
- A computer running the macOS operating system. The latest version of the Safari browser must be installed on this machine.
- One of the OnBoard Devices that will be mounted on the cars of PowerEnJoy

5 PROGRAM STUBS AND TEST DATA REQUIRED

5.1 Drivers and Stubs

Because we have chosen a bottom up approach, for each step of integration a driver is needed for the simulation of the latter component that has been added.

5.1.1 User Manager/Service Manager Integration

For this set of test we need a driver for the functionalities of the Service Manager which calls functions that make the User Manager create, delete or modify the users accounts and retrieve information for the authentication of users

5.1.2 Reservation Manager/Service Manager Integration

For this set of test we need a driver for the functionalities of the Service Manager which calls functions that make the Reservation Manager begin, end or delete a reservation and search cars.

5.1.3 Fee Manager/Payment System Integration

For this set of tests we need a stub for the simulation of the payment system. This stub has to send to the Fee Manager information about payments, like the confirmation of a payment or possible errors.

5.1.4 Fee Manager/Service Manager Integration

For this set of test we need a driver for the functionalities of the Service Manager which calls functions that make the Fee Manager calculate the fee for a travel and charge the user with that fee.

5.1.5 Position Manager/Service Manager Integration

For this set of test we need a driver for the functionalities of the Service Manager which calls functions that make the Position Manager retrieve and return the positions of users and cars.

5.1.6 Car Manager/Service Manager Integration

For this set of test we need a driver for the functionalities of the Service Manager which calls functions that make the Car Manager send an unlock request to a car and to communicate the end of a trip.

5.1.7 Service Manager/Request Handler Integration

For this set of test we need a driver for the Request Handler which calls functions that forwards requests to the Service Manager. So it must have functions that send requests for the creation, deletion or modification of an account, for the search of a car, for the begin or the end of a reservation and the authentication of users.

5.1.8 Service Manager/OnBoard Device Integration

For this set of test we need a driver for the OnBoard Device which call a function that sends the communication of the end of a trip to the Service Manager.

5.1.9 Request Handler/HTTP Server Integration

For this set of test we need a driver for the HTTP Server which calls functions that forward requests coming from the clients to the Request Handler. So it must have functions that send requests for the creation, deletion or modification of an account, for the search of a car, for the begin or the end of a reservation and the authentication of users, similarly to the Request Handler.

5.1.10 HTTP Server/Mobile Client Integration

For this set of test we need a driver for the Mobile Client which call functions that send requests to the HTTP Server. So those functions must send requests for the creation, deletion or modification of an account, the search of a car, the begin or the end of a reservation and the authentication of users to the HTTP Server.

5.1.11 HTTP Server/Web Client Integration

For this set of tests we need a driver for the Web Client which call functions that send requests to the HTTP Server. So it is similar to the Mobile Client driver, but it has less functionalities. The functions this driver must have must send requests for the creation, deletion or modification of an account, for the search of a car and the authentication of users.

5.2 Test Data

In addition to drivers and stubs, to perform integration tests some sets of data to use in those tests are needed:

- A set of account data. This data include name, surname, username, email address, password, telephone number and driving license number. The set must contain:
  - At least a null value for each information

  - At least an invalid value for email address, password, telephone number and driving license number.
  - At least two duplicate values for the couple name/surname and for username, email address, telephone number and driving license number.

  This data will be used for tests regarding components the send requests for or manage the creation, deletion and modification of accounts.

- A set of username/password couples. This set must contain:
  - At least a couple with a null value for the username
  - At least a couple with a null value for the password
  - At least a couple with a null value for both username and password
  - At least a couple with an invalid value for the username
  - At least a couple with an invalid value for the password
  - At least a couple with an invalid value for both username and password

  This data will be used for tests concerning components that request or manage authentication of users.

- A set of reservations. A reservation have fields, that are: username, driving license number, plate number of the car, the time of the beginning of the reservation. This set must contain:
  - At least a null value for each field
  - At least an invalid value for each field

  This data will be used for tests involving components that request the beginning of a reservation and components that manage reservations

- A set of bills. A bill contains the name and the surname of the user, his payment information and the amount to pay. This set must contain:
  - At least a bill with a null value for each field.
  - At least a bill with a invalid value for the name/surname couple and for the payment information
  - At least a bill with a negative value for the amount to pay.

  This data will be used for tests regarding components that manage fees and payments.

6 Appendix

6.1 Used software

1. **TeXstudio:** `http://www.texstudio.org/` to redact this document in LaTeX format.
2. **Astach:** `http://astah.net/` to draw diagrams.

6.2 Time effort

The estimated time spent by us to redact this document:

Andrea Millimaggi	18h
Matteo Michele Piazzolla	10h