POLITECNICO
MILANO 1863

Politecnico di Milano
A.A. 2016 – 2017

Software Engineering 2: "PowerEnJoy"

Integration Test Plan Document

Marco Festa
January 15, 2017

Contents

1 Introduction

1.1 Purpose and Scope

The Integration Test Plan Document (**ITPD**) purpose is to provide a fully detailed approach to the integration testing procedure. Testing is fundamental to ensure proper interoperability between different components and subsystems. Different tools and strategies are used to ensure each module's integrity and functionality.

PowerEnJoy aim is to provide a car-sharing service for multiple cities with the peculiar characteristic of deploying exclusively electric cars. The scope of our project is to build a new digital management system to the company in order to support the service operations. A better description of the different goals to achieve is available in previous redacted documents (**RASD** and **DD**.)

1.2 Definitions, Acronyms, Abbreviations

Definitions

- *User (or Registered User):* a person registered to the system. The driving license has been verified and the customer info is correctly added into the database. Registered users are the only entities eligible for car riding.
- *Driver:* the user who physically unlocks an electric car using his own credentials and starts driving it becomes automatically the driver. Registered users are actually the only possible persons to potentially become drivers.
- *Guest:* a person that is not necessarily registered to the system.
- *Safe area:* geographical area where cars are authorized to be parked giving the user the chance to end the ride.
- *Power Grid Station:* inside the safe areas are located several power stations where electric cars can be plugged in and have the central battery recharged.
- *Free Car:* a car which is not being used by any registered user and is not under any pending reservation is considered available or free.

- *Reserved Car:* each registered user has the ability to choose a free car from the smartphone app or web interface and have it reserved for at most one hour. During this phase the car disappears from the list of free cars and can only be opened and unlocked by the user who made the reservation. The reservation state of a car ends either when the car is unlocked by the user who automatically becomes the driver, the one-hour limit is reached, or the state is ended by the user itself.
- *Opening procedure:* a user can open a free car, or one he had reserved, by using the dedicated feature on his smartphone app. An opened and locked car has to be considered a reserved car.
- *Unlocking procedure:* car unlocking is achieved via a PIN code entered through the touch screen display inside the vehicle.
- *PIN code (or PIN):* a 4-digit secret code chosen by the user on his first car ride.
- *In-use Car:* a car that has been unlocked by a registered user and is now able to be turned on by the driver.
- *CAN bus:* the physical network connecting each present electronic device: sensors, micro-controllers, actuators and instruments.

Acronyms

- **RASD**: Requirement Analysis and Specification Document
- **DD**: Design Document
- **ITPD**: Integration Test Plan Document
- **DBMS**: Database Management System
- **DB**: Database
- **PGS**: Power Grid Station
- **API**: Application Programming Interface: a common way to communicate with other systems.
- **EUCARIS**: European Car and Driving License Information System

1.3 Reference Documents

- PowerEnJoy RASD
- PowerEnJoy DD
- Specification document: "Assignments AA 2016-2017.pdf"
- Example document: "Integration Testing Example Document.pdf"

1.4 Document Overview

- **Section 1** – **Introduction**: presentation of the document and product.
- **Section 2** – **Integration Strategy**: this section explains all of the testing strategies adopted. It's divided in:
  - *Entry Criteria*
  - *Elements to be Integrated*
  - *Integration Testing Strategy*
  - *Component / Subsystem Integration*
- **Section 3** – **Individual Steps and Test Description**: in this section is are specified the types of tests performed for each step proposed in section 2.
- **Section 4** – **Tools and Test Equipment Required:** this section lists all of the tools used to perform tests.
- **Section 5** – **Program Stubs and Test Data Required**
- **Section 6** – **Appendix**:
  - Tools used to write the document
  - Working hours division.
  - Revisions

2 Integration Strategy

2.1 Entry Criteria

To ensure a safe and correct integration testing procedure we need to satisfy some basic prerequisites.

All of the design and requirements documents have to be fully written and reviewed (**RASD** and **DD**) to provide a detailed overview of all the components features.

Each component must reach a state of at least 80% of their completion or expose the critical modules to test. Full component completion is not required for testing neither for deployment.

Test units have to target all of the critical component functions with a code coverage of at least 90%. To further detect bugs and possible unexpected vulnerabilities code inspection has to be performed by third party experts.

2.2 Elements to be Integrated

Referring to the component view proposed in *section 2.2* of the DD, the components to include in the testing process are:

- Mobile App
- Web Application
- Car Subsystem
- Application Subsystem

All of the other components are external and thus assumed to be correctly functioning.

All of the listed elements are tested both internally, covering the lower integration between their sub-components, and externally considering the possible interaction between each other. This much more high-level testing procedure is also to be extended to the external components:

- Mobile App/Web App ⇔ Application Subsystem
- Car Subsystem ⇔ Application Subsystem
- EUCARIS API ⇔ Application Subsystem
- Payment API ⇔ Application Subsystem
- PGS Subsystem ⇔ Application Subsystem

All of the internal sub-components are listed and shown in *figure 3 and 4* of the DD.

For performance and integrity reasons the testing process has to focus on the Application Subsystem and its internal components interactions. The request dispatcher component along with the model component are surely two of the most critical elements to consider.

2.3 Integration Testing Strategy

A bottom-up testing strategy is the only reasonable choice in our scenario: with this approach we are able to start testing the core Application Subsystem while the other components or API interfaces are still in development state (with a requirement limit of 80% completion as stated in *section 2.1* of this document.) The integration with the Car Subsystem will be covered right after the two interacting components have finished their internal testing process. To avoid a third parallel testing unit to be deployed at the same time, user applications (both Mobile and Web) can be excluded from this approach and tested after the two main component are covered. The user part is in fact less critical and more subject and open to future changes (GUI updates etc.) allowing us to focus mainly on the interactive modules and functions.

2.4 Component / Subsystem Integration

In this section the integration order is further detailed and extended to each single component. As a notation, an arrow going from component C1 to component C2 means that C1 is necessary for C2 to function and so it must have already been implemented.

Software Integration Sequence

As we mentioned in the upper section the bottom-up approach is split in two parallel procedures, respectively covering the Application Subsystem and the Car Subsystem. The testing procedure will later merge once the single components are fully tested.

Application Subsystem

Data Components

The first two elements to be tested are the components responsible for the representation and storage of the application data structure. The DBMS component, accessed through the DBMS interface, is and external component considered to work as expected (we don't need to test if queries are working properly). The analysis is mainly restricted on the Model Component.

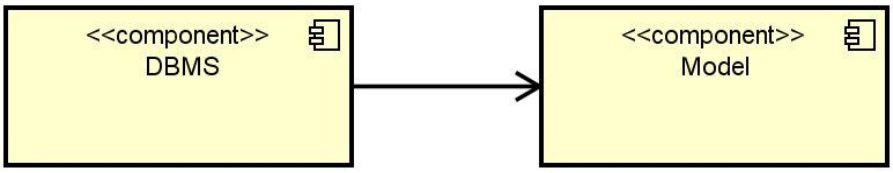


Figure 1: Data Components

Account Management

The diagrams below show the integration testing of all the components in charge of handling user-data and account settings. In this specific phase only the user-data related types and methods are tested.

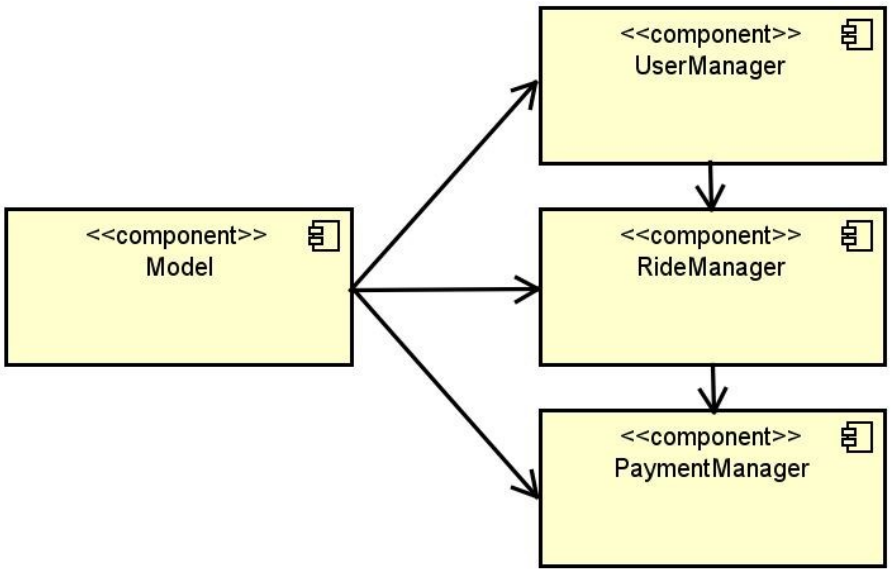


Figure 2: Account Management

Car Management

CarManager in the component receiving and exposing all of the car signals and functionalities. While direct connection with the CarSubsystem is tested later in the process we are still able to integrate all of its other methods and procedures related to Ride management and data structure representation.

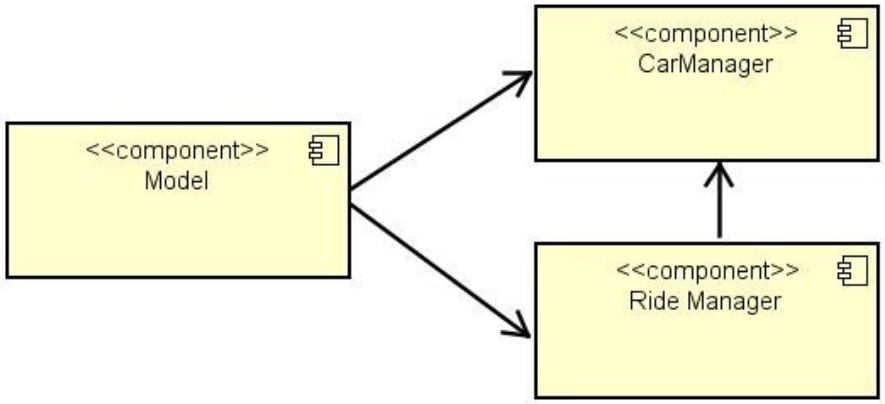


Figure 3: Car Management

Search Management

SearchManager directly connects to the Model component in order to operate user custom searches.

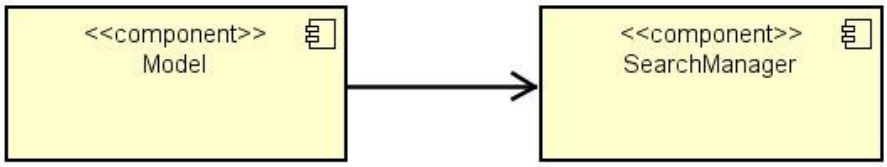


Figure 4: Search Management

PGS Management

Basic PGS data representation tests are performed in this unit. Connection to the PGS external interface is considered a higher level interaction and thus integrated later on.

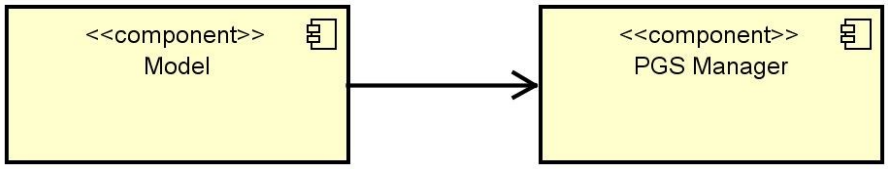


Figure 5: PGS Management

Request Dispatcher

The request dispatcher communicates with all of the modules shown below therefore needs an accurate testing procedure in order to guarantee all of the main system functionalities.

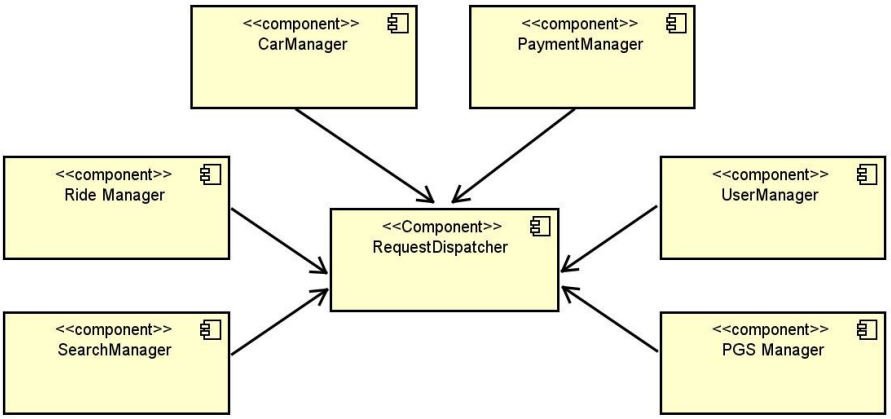


Figure 6: Request Dispatcher

Interfaces

All of the interfaces are briefly tested to ensure the API documentation was carefully followed and no errors were made during component development.

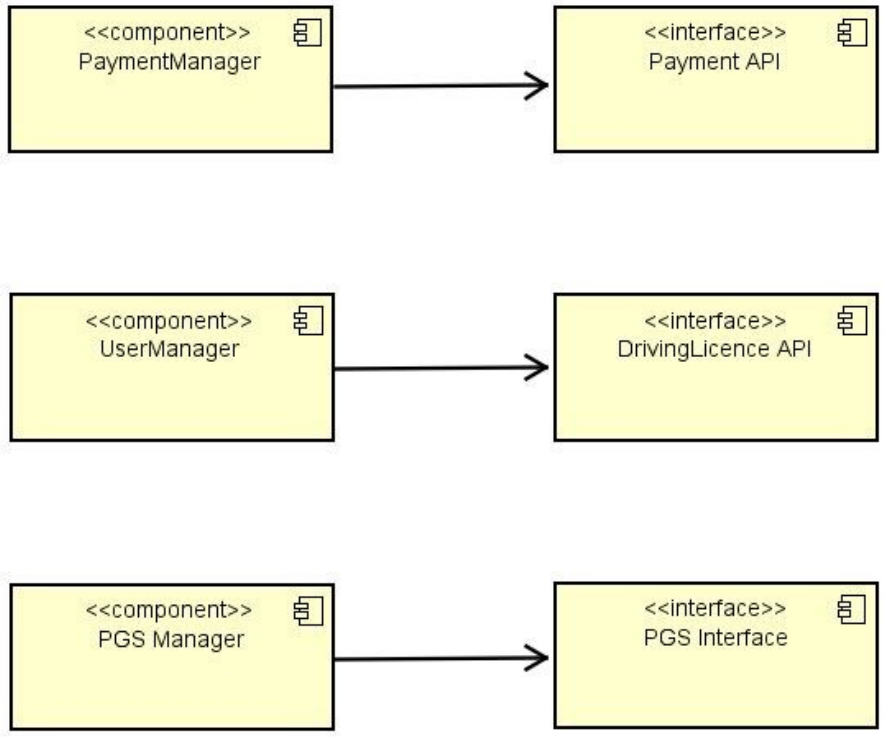


Figure 7: Application Subsystem Interfaces

Car Subsystem

General Controller

This stage tests all of the car sub-components interaction with the General Controller and the various car devices and sensors.

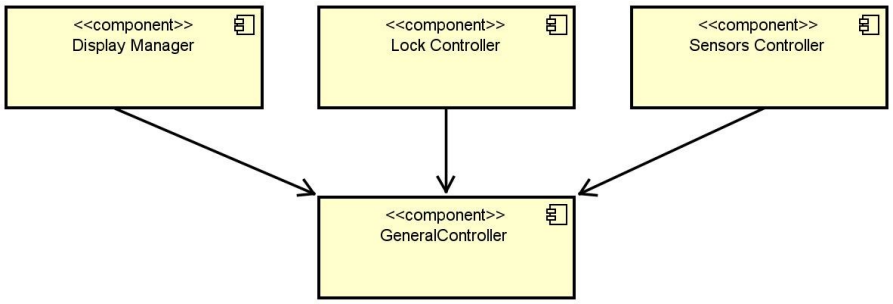


Figure 8: Car General Controller

Request Dispatcher

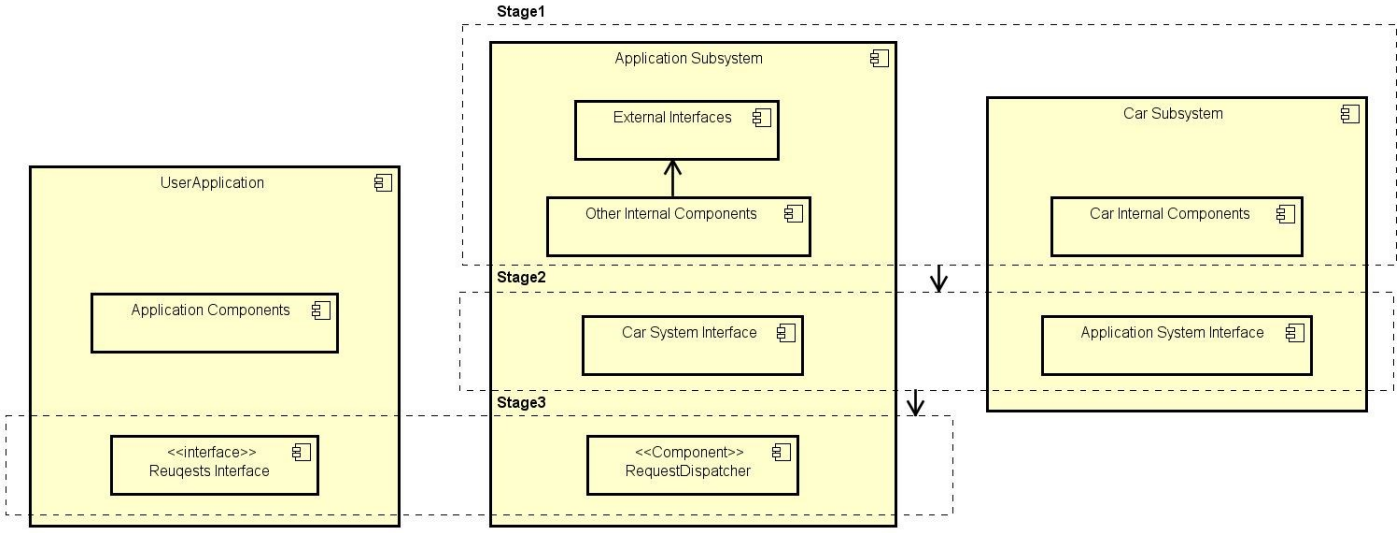
The Car Subsystem Request Dispatcher component is tested right after all of the above components are.

User Application

Both user applications are tested later in the procedure. Their internal components are not listed since they still have not been fully detailed in the project documents. For our integration procedure we only need to consider the interface responsible for the direct communication with the Application Subsystem.

Subsystems Integration Sequence

This section shows how all of the high level components combines during integration testing procedure.



3 Individual Steps and Test Description

3.1 Software Integration

For each step of the integration plan are here listed the most critical functions to test. In the below tables each possible input is followed by the expected effect.

Data Components

This step shows the possible requests generated by the **Model Component** querying the DBMS.

New User	
Input	*Effect*
Unique DB entry for: username, mail, driving license id. Payment method and user personal data.	New user inserted in database.
One of the 3 entry is not unique.	An exception is raised.
Any of the listed data is missing.	An exception is raised.
User Update	
Input	*Effect*
Existing username, new user data.	User update performed.
Non existing username, new user data.	An exception is raised.
Insert Ride	
Input	*Effect*
Ride data, with a valid car ID and ride status set to Reservation. Also ride minutes set to 0.	New ride inserted in database.

Invalid car ID, missing ride data or ride status different from Reservation.	An exception is raised.
Update Ride	
Input	*Effect*
Present ride ID, new ride data car ID excluded.	The ride is updated.
Non existing ride ID, ride status set to Reservation or inconsistent ride data (minutes set to a value lower than 0).	An exception is raised.
Insert Car	
Input	*Effect*
Unique integer ID, availability set to false, valid position provided.	New car inserted in database.
Missing plate, existing ID or position outside SafeArea.	An exception is raised.
Update Car	
Input	*Effect*
Existing Car ID, new valid car data.	Car DB update.
Non existing Car ID, invalid position, negative battery charge value.	An exception is raised.
Get User	
Input	*Effect*
Existing username	A user is returned.
Non existing username	An exception is raised.

Get Ride	
Input	*Effect*
Existing Ride ID.	A Ride element is returned.
Non existing Ride ID.	An exception is raised.
Get Car	
Input	*Effect*
Existing Car ID.	A Car element is returned.
Non existing Car ID.	An exception is raised.

Account Management

User Manager Methods

User Registration	
Input	*Effect*
Username, mail, valid driving license id. Payment method and user personal data.	Return Add User Model request response.
Missing one of above data.	An exception is raised.
User Update	
Input	*Effect*
User ID, new valid user data, valid driving license ID.	Return Update User Model request response.
No user ID provided.	An exception is raised.
Get User Rides	
Input	*Effect*
User ID.	Return ridesHistory[ ] of Get User response.
Null input.	An exception is raised.

Get User Status	
Input	*Effect*
Valid username.	Return the user status.
	An exception is raised.
User Update	
Input	*Effect*
Valid username, new user data.	User is updated.
	An exception is raised.
Check User Credentials	
Input	*Effect*
Correct username, password.	Auth Token is returned.
	An exception is raised.

Ride Manager Methods

Start Ride	
Input	*Effect*
User data, car data.	A ride object is created and returned
	An exception is raised.
Update Ride	
Input	*Effect*
Valid Ride ID, ride data updates.	The ride object is updated.
	An exception is raised.

End Ride	
Input	*Effect*
	An exception is raised.
Get Ride Info	
Input	*Effect*
	An exception is raised.

Payment Manager Methods

Pay Ride	
Input	*Effect*
	An exception is raised.
Refund Ride	
Input	*Effect*
	An exception is raised.

Car Management

Car Manager Methods

Open Car	
Input	*Effect*
	An exception is raised.
Unlock Car	
Input	*Effect*
	An exception is raised.
Close Car	
Input	*Effect*
	An exception is raised.
Get Car Data	
Input	*Effect*
	An exception is raised.
Notify Ride End	
Input	*Effect*
	An exception is raised.

Ride Manager Methods

Set Car Unavailable	
Input	*Effect*
	An exception is raised.

Search Management

Search Manager Methods

Get Cars from Position	
Input	*Effect*
	An exception is raised.
Get PGS from position	
Input	*Effect*
	An exception is raised.

Request Dispatcher

The Request Dispatcher acts like a wrapper for all the requests coming from the User Application therefore the only significant method to test is the TrasferRequest one. The only condition to satisfy is a valid request type (specified by an enumerator object), and a correct number of arguments for each of the corresponding one.

4 Tools and Test Equipment Required

4.1 Tools

The following tools are used to test the entire project in the most efficient way.

- **JUnit:** a testing framework for the Java programming language. JUnit framework includes: assertions, exception testing, timed-out tests, test fixtures and runners.
- **Mockito:** useful framework to test not yet developed components. During integration testing phase this will allow us to emulate some interfaces or external modules behaviors.
- **Selenium web-driver:** a software used to automate browser interaction. A selection of different web-drivers allows us to test the Web Application in different software frameworks (e.g. Firefox, Chrome, IE).
- **Espresso/ UI Automator:** android tools to automate application interaction.

4.2 Test Equipment Required

To perform testing we need a certain set of hardware equipment to emulate the PowerEnJoy system.

- A powerful x86-64 machine to run the PowerEnJoy core application.
- A prototype of the embedded board to be installed on each car.
- A secondary machine to run PGS software emulation since the real technology is proprietary and provided by and external company.
- Test machines/laptops to run the Web Application tests carefully selecting each of them with a different screen size and resolution.
- Numerous mobile devices to test the Mobile Application. At least the two more recent model of smartphones and tablets from both Apple (iPhone and iPad) and Google (Nexus models).

5 Program Stubs and Test Data Required

5.1 Program Stubs and Drivers

To pursue the bottom-up approach we need to develop at least two stubs to emulate the Car System Interface and Application Interface components and a set of drivers to support other all of the other components.

- **PGS Manager Driver:** all of the component methods are implemented in this driver in order to complete the corresponding test.
- **Car Manager Driver:** a car manager driver is also developed to structure the corresponding actions implemented by the Model Component (e.g. continuous car position updates).
- **Search Manager Driver:** this driver also tests against the Model all of the possible interactions related to car searching procedures. It's important in this phase to test performance and consistency in order to allow a fast displaying of the available cars to the user.
- **Ride Manager Driver:** ride manager driver apart from testing its interaction with Model, User Manager and Car Manager components tries to emulate the most common ride procedure including ride payment.
- **Payment Interface Stub:** supports the above testing procedure emulating the Payment API Interface.
- **Request Dispatcher Driver:** this fundamental driver is a practical emulation of the User Application interaction with the Application System. Along with the Car System Stub it allows us to simulate and test the whole system.

5.2 Test Data

In this section we enumerate a series of test data used to perform the various scheduled tests in every possible combination.

- **Sign up data:** we try to register to the system providing all sort of corrupted or inconsistent data.

    - Impossible personal information (e.g. special characters in name and surname)
    - Invalid DOB (different date formats)
    - Non existing email addresses
    - Already in-use email addresses
    - Invalid driving license numbers
    - False or deactivated credit cards numbers
- **Sign in data:** we test a vast range of possible username and password combinations to ensure login procedure behave as expected.
- **Search Inputs:**
    - Range of valid and invalid positions
    - Valid but not "reachable" positions.

6 Appendix

7.1 Software and tools

- Astah Professional for integration testing diagrams.
- MS Word for the whole document.
- Git for version control.

7.2 Effort spent

Marco Festa: 60 hours

7.3 Revisions

- ITPD v1.0 published January 15, 2017.
- ITPD v.1.1 missing sections updates, February 2, 2017