POLITECNICO DI TORINO

Master of Science in Computer Engineering

Master Degree Thesis

# Web platform for Eco-Pack observatory


**Supervisor:**
Ing. Alfredo Benso

**Assistant Supervisor:**
Arc. Silvia Barbero

**Candidate:**
François-Xavier Renna
matricola: s220351

October 2018

# Summary

The thesis consists in creating a web platform for the Observatory of the Eco-Pack, making very divergent skills converge and dialogue to bring back a complex project. More specifically: The IT Engineering (of the Polytechnic of Turin) will play the part of back-end (database, authentication, etc ...) while those of Design and Visual Communication (of the Department of Architecture and Design of the Polytechnic of Turin) will deal with the part of front-end (graphical interface, etc ...).

# Acknowledgements

Un sincero ringraziamento a tutti coloro che in questi anni hanno intrecciato le loro strade con la mia ed hanno reso possibile questo traguardo. In particolare:

- alla mia famiglia senza la quale non sarebbe stato possibile alcun traguardo;
- alla defunta Fausta;
- ad un amico e non solo, Cristiano Cavo con il quale ho condiviso la maggior parte degli studi universitari, progetti, vita collegiale e altro. Sebbene nelle mille diversitá e nei molteplici problemi abbiamo sempre trovato la retta via scalando pian piano la montagna. Senza dimenticare di ringraziare anche la sua gentile famiglia;
- ad una particolare amica, Sabrina Camurati, conosciuta alla specialistica e compagna di avventure tra le quali Verigraph insieme a Cristiano Cavo;
- Inoltre va ricordata la singolare amicizia con Rosa Bianca.

In aggiunta non vanno dimenticate tutte quelle persone che nel bene o/e nel male hanno interagito nella mia vita poiché con il se si metterebbe Parigi in bottiglia...

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

The chapter wishes to describe the thesis' goal and giving also a summary for each chapter.

## 1.1 Goal and roles

The goal of the thesis is to create a web platform for the Observatory of the Eco-Pack, making very divergent skills converge and dialogue to bring back a complex project. More specifically:

- The IT Engineering (of Turin's Polytechnic) will play the part of back-end and the most part of front-end (totally written and in part created without mock-up).
- The Department of Architecture and Design of the Turin's Polytechnic has produced the mock-up of the main views of the packagings giving high-level specifications for back- end development.

In other words, the work consists to create a web site in order to insert, modify and delete the packagings (e.g milk packaging). These packet cards (or packagings) contents the information about the packaging (e.g packaging's name, production year and so on).

This information is for example the packaging's name, production's year, etc etc. In addition it is possible to do some search in order to pick up the right cards. There is also a registration and login to split the authorized persons from others. The first can manipulate the database (to insert, delete and update cards) while the others can only see the cards and do the search.

The project has been created using the KISS principle[1] approach in order to give others programmers the possibility to update the code. In particular some parts have been built like black box and these (black boxes) have been used to build the project and can be use for a future to grow the project.

## 1.2 High view of the chapters

The work of the thesis is articulated in the following chapters:

- **Chapter 2** Chapter 2 describes the mock-up;
- **Chapter 3** Chapter 3 describes MySQL database with the respective implementation choices;
- **Chapter 4** Chapter 4 describes the back-end with the respective implementation choices;
- **Chapter 5** Chapter 4 describes the front-end with the respective implementation choices.

# Chapter 2

# Mock-up

The chapter provides a description of the mock-up specification. The mock-up has been the starting point because it represents the specifications or in others terms it outlines both the back-end part and the front-end part.

It is not necessary to describe the mock-up because it is easier to see the images of it in order to understand better, but it is necessary to know some things:

- it has been divided in 3 parts (2 pages, 1 page and 2 pages);
- the first part and the last have two pages but the different about the pages is an image that changes color (from color to green and vice versa). The images show the packaging;
- each of them (parts) contain certain information about the packaging.

## 2.1 Pre mock-up

It is important to know some points in order to better understand the elements:

- certain fields can be empty because the user that has created the packaging card did not know this information;
- one field (name of packaging) must be present because it is necessary to have an intelligible user-unique link between the database and the final user;
- if a field is empty, it has replaced from the follow symbol "-";
- certain names, that have been used in the database is not immediately understandable but they will have a sense when reading the next chapters;
- almost all fields have own table, in order to have a list of elements, but some others have not a table;
- there is a big table in the database that contains the packaging's information, the table is called *Packets*.

## 2.2 First part of mock-up

The first part has the follow information:

- **Nome packaging:** it represents the name of the packaging and it must be unique in all the database (each packaging has own name) and each packaging must have a name. It is named by *PacketName* in database. It has not a table but it is memorised in *Packets* table. The maximum length is 50 characters and it doesn't change if the web site changes language. The packaging has only one name.
- **Azienda:** it represents the company that produces the final packaging. It is named by *Company* in the database and it has own table named *Companies*. The maximum length is 50 characters and a packaging cannot have this field. It (packaging) has only one.
- **Designer:** it presents the designers that have created the packaging. Because a packaging can be created by one or multiple designers, this field can be repeated until five times. So a packaging can be make by one designer, two designers until to five designers. The designer's field only has this property. The field cannot translate in others language, because it contains a proper name of person. In the database is called *Designer* and it has own table named *Designers*. The maximum length is 50 characters. And all the designers can be empty.
- **Azienda produtrice del packaging:** it represents the packaging manufacturer and in the database it is named *Manufacturer* and it has own table named *Manufacturers*. It can be empty and the maximum length is 50 characters. The packaging only has one and it is stored with an id in *Packets* table.
- **Settore merceologico:** it represents the commodity sector like for example pharmaceutical. In the database it is named *ProductSector* and it has own table named *ProducSectors*. It can be empty and the maximum length is 50 characters. The packaging has only one and it is stored with an id in *Packets* table.
- **Anno di produzione:** it represents the production year of the packaging. In the database it is named *ProductionYear* and it has own table named *ProductoinYears*. It can be empty and it is an integer like 1990. The packaging has only one and it is stored with an id in *Packets* table.
- **Nazione:** it represents the nation where the packaging has been built. In the database it's named *Nation* and it has own table named *Nations*. It can be empty and the maximum length is 50 characters. The packaging has only one and it's stored with an id in the *Packets* table.
- **Tipologia:** it represents the typology of the packaging like for example bottle. In the database it's named *Typology* and it has own table named *Typologies*. It can be empty and the maximum length is 50 characters. The packaging has only one and it is stored with an id in *Packets* table.

- **Volume:** it represents the *PacketVolume* percent of the packaging (e.g 50%). In the database it's named Volume and it has not an own table. It can be empty and the maximum length two integer from 0 to 99 (e.g. 9% or 99%). The packaging has only one and it's stored with an id in *Packets* table.

- **Altezza:** it represents the height in millimeters (mm). In the database it's named *PacketHeight* and it has not an own table. It can be empty and the maximum length is 4 integer from 0 to 9999. The packaging has only one and it's stored with an id in *Packets* table.

- **Larghezza:** it represents the width in millimeters (mm). In the database it's named *PacketWidth* and it has not an own table. It can be empty and the maximum length is 4 integer from 0 to 9999. The packaging has only one and it's stored with an id in *Packets* table.

- **Peso:** it represents the weight in grams (g). In the database it's named *PacketWeight* and it has not an own table. It can be empty and the maximum length is 4 integer from 0 to 9999. The packaging has only one and it's stored with an id in *Packets* table.

- **Funzionalitá:** it represents the functionality of the packaging. It is a rating of 5 points from 1 to 5. It has automatically computed using another parameters that stay on the second part of mock-up where is the functionality rating table. In the database it is called *PacketFunctionalityAverage* field. The packaging has only one.

- **Sostenibilitá:** it represents the sustainability of the packaging. It is a rating of 5 points from 1 to 5. It has automatically computed using another parameters that stay on the second part of mock-up where is the sustainability rating table. In the database it is called *PacketSustainabilityAverage* field. The packaging has only one.

- **Comunicazione preventiva:** it represents the communication of the packaging. It is a rating of 5 points from 1 to 5. It has automatically computed using others parameters that stay on the second part of mock-up where is the communication rating table. In the database it's called *PacketCommunicationAverage* field. The packaging has only one commodity.

- **Giudizio riassuntivo:** it represents the final rating calculates on all the rating that describe the packaging. It is memorised in table with *TotalRating* in *Packets* table. The packaging has only one.


Figure 2.1 – Mock-up part 1 page 1

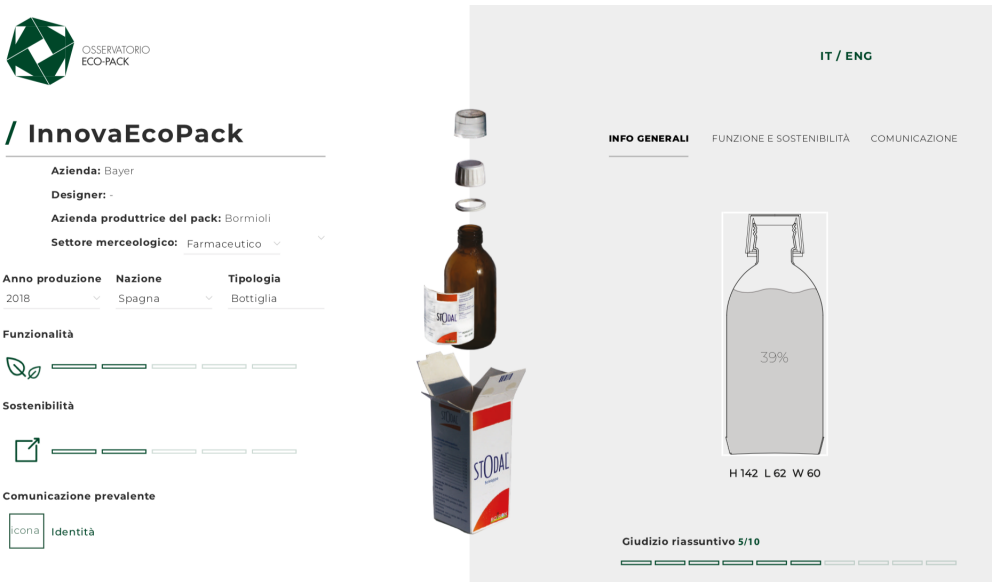


Figure 2.2 – Mock-up part 1 page 2

## 2.3 Second part of mock-up

The second part of mock-up has the follow information:

- **Funzionalitá** has composed with some ratings:

  - **Ottimizzazione degli spazi:** saved in *Packets* table with *ARatingSpaceOptimization* item; It can empty and it is a foreign key *AListRatingSpaceOptimizations*.
  - **Protezione e conservazione del prodotto:** saved in *Packets* table with *ARatingProtectionConservation* item; It can empty and it is a foreign key *AListRatingSpaceOptimizations*.
  - **Praticitá di utlizzo:** saved in *Packets* table with *ARatingPracticalityUse* item; It can empty and it is a foreign key *AListRatingSpaceOptimizations*.

- **Sostenibilitá** has composed with some ratings:

  - **Presenza di sovraimballaggi:** saved in *Packets* table with *BRatingPresenceOverpacking* item; It can empty and it is a foreign key *BListRatingPresenceOverpackings*.
  - **Composizione e separabilitá dei materiali:** saved in *Packets* table with *CRatingCompositionSeparability* item; It can empty and it is a foreign key *CListRatingCompositionSeparabilities*.
  - **Rapporto pesi e volumi:** saved in *Packets* table with *ARatingWeightVolume* item; It can empty and it is a foreign key *AListRatingSpaceOptimizations*.

- On the middle of the page there is an image of the packaging, that contains its elements and material's element. For instance polyethylene stopper. This image is saved on the database under *DescriptionImages* table. For each language that the packaging has been translated, there is an image with right term written in this tongue.

- The table shows which components and which materials has a packaging with own weight. More in details, each packaging is made up of one or more components and each component is made up of only one material. An element(s) cannot be used in the sum in order to have the total weight or it can be multiplied for x because in the product there are some of these elements. So, near the weight can be a string of characters in order to specific the case.

  The string can be:

  - **xN** where x means multiply and N for how many (e.g. in a tea pack there are more filters).
  - **<N** means that this weight cannot use for the total weight (e.g. the package contains a feather).

  It is not possible to have the two strings. All the weights are in grams.

  All this information has been saved in some tables:

  - **Materials** table for the material with own specific attribute language.
  - **Components** table for the material with own specific attribute language.

– **TypeWeights** table where have been saved the string (xN or <N).


Figure 2.3 – Mock-up part 1 page 2

## 2.4 Third part of mock-up

The last part or third part has the following information:

- **Free text:** where the user can write about the packaging. It has been stored in *Phrases* table with own language attribute and the name of field is named *Phrase*; The maximum length is 500 characters. The field can be empty.The packaging has only one.
- **Font:** represents the font of the packaging (e.g. serif), it has stored in *Fonts* table and the specific field is *Font*. The field can be empty; the maximum length is 20 characters. The packaging has only one.
- **Immagini:** represents the typology of image (e.g. photography, iconic illustration). It stores in *ComunicationImages* tables.
- **Sensorialitá:** called *PacketSensoriality*, it is a boolean and it has stored in *Packets* table.
- **Fine vita:** called *EndLive)* it is a boolean and it has stored in *EndLives* table with own language. The field can be empty and the maximun length is 20 characters.
- **Palette colori:** represents the list of colors of the packaging. It can have 0 and the maximum is 4. There is a main color and the rest are secondary colors with

a fix position (a rating position). The colors are stored in *Colors* table and the maximum length is 30 characters. In order to choose the properly color there is also the correspond on hexadecimal color (e.g White #FFFFFF).

- **Dettagli immagini:** it represents the detail of packaging's image and they have stored in *Images* table.

- **Marchi ambientali:** it represents the environmental trademarks and they have been stored in *EnvironmentalTrademarks* table like image with own properly number.

- **Rapporto informazione/comunicazione:** divided in three parts:

  - **Informazione** stored in *Packets* table with a integer like *PacketPercentInformation*.

  - **Comunicazione** stored in *Packets* table with an integer like *PacketPercentCommunication*.

  - **Area neutra** stored in *Packets* table with an integer like *PacketPercentNeutralArea*.

- **Comunicazione:** has divided in third indexes:

  - **Appeal** stored in *Packets* table with a integer like *ARatingAppeal*.

  - **Identitá** stored in *Packets* table with a integer like *ARatingIdentity*.

  - **Messaggio** stored in *Packets* table with a integer like *ARatingMessage*.

  - **Informazione** stored in *Packets* table with a integer like *ARatingInformation*.

  - **Affordance** stored in *Packets* table with a integer like *ARatingAffordance*.


Figure 2.4 – Mock-up part 3 page 1

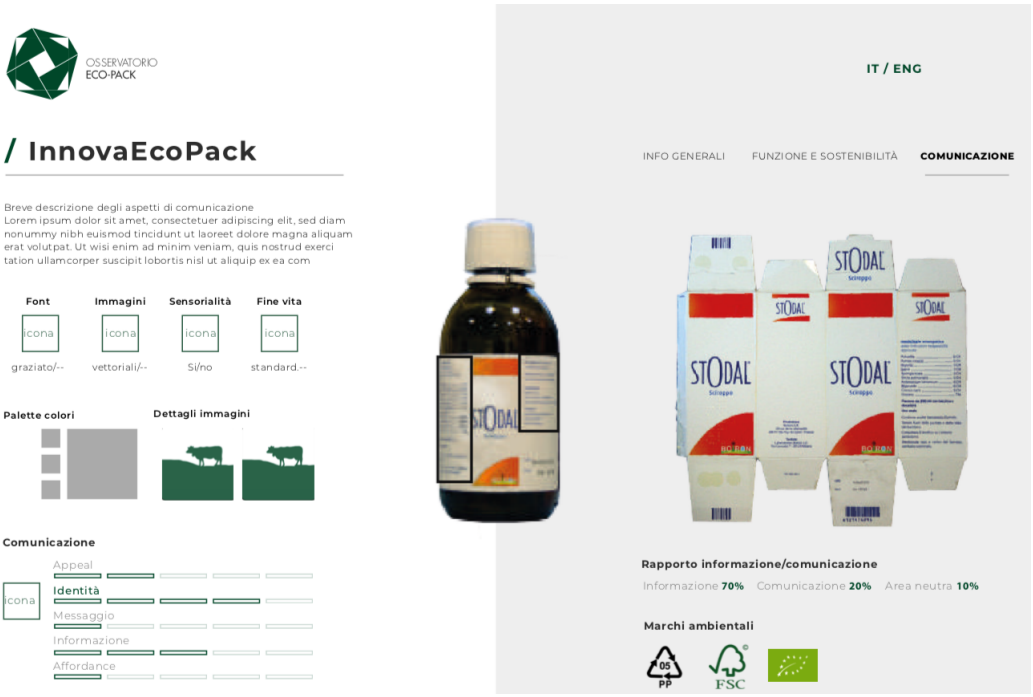


Figure 2.5 – Mock-up part 3 page 2

## 2.5 Rating table

The third rating tables have the evaluation phrases, in this way the user can decide the right valuation:

- **AListRatingFunctionalityAndVolumeWeights** table, the rating are:

  1. poor
  2. discreet
  3. good
  4. great
  5. excellent

  It is used by

  1. Appeal
  2. Identitá
  3. Messaggio
  4. Informazione
  5. Affordance
  6. Ottimizzazione degli spazi stoccaggio e trasporto
  7. Protezione e conservazione del prodotto
  8. Praticitá di utilizzo

- **BListRatingOverpacks** table, the rating are:

  1. more overpack for aesthetic purposes
  2. an overpack for aesthetic purposes
  3. more overpacks for protective purposes
  4. an overpack for protective purposes
  5. no overpackaging

  It has used by Presenza di sovraimballaggi

- **CListRatingCompositionSeparability** table, the rating are:

  1. not separable
  2. different materials, difficult to separate
  3. different materials, easy to separate
  4. few materials, easy to separate
  5. monomaterial

  It has used by Composizione e separabilitá dei materiali

These phrases have been stored for each language that the project can use.

# Chapter 3

# Database design

The chapter describes the database that has been developed from the mockup's specifications.

## 3.1 Design choices

In order to have a good and understandable database there are some rules that have been used.

### 3.1.1 Name attributes and name tables

There are some conventions for the names:

- All tables have their own names and end up with an "s", in this way it is simpler to distinguish the name of the tables to the attributes because they don't finish with "s".
- All the attribute names start with lowercase, but the ids start with uppercase in order to better identifies.
- All the names of tables start with uppercase.
- The relation tables have a concatenates name with the tables that create the relation. The name is in alphabetic order. In one case, they start with Packet when the relation is among or between *Packet* table.
- It has used the camelcase for all the names of database.
- Every table has own id and the name of id is "id" plus the name of the table without the "s". All the attributes that are not important to have a list has been stored inside the *Packet* table like volume. On the contrary, where there is the necessity to have a list of name, a table has been created like *Nation* table in order to maintain a list like for the nations. In the first case, the value has stored in *Packet* table while in the second case it has divided in two cases:

  - if there is a single value for the packaging, the link between the packet and the table is a table's id memorized in *Packet* table;
  - if there is not a single value like for designers, there is a relation table.

  Obviously like all the tables (apart from those of relationships) there is an id belonging to the table itself.

- The relation tables have more id and the id are represented by the id of the tables that form the relation: For example:

```
1          CREATE TABLE PacketsTrademarks (
2          IdPacket INTEGER NOT NULL,
3          IdTrademark INTEGER NOT NULL,
4          PRIMARY KEY (IdPacket, IdTrademark)
5          );
```

  This is the relation between packet and its trademarks, the ids of this table are: *IdPacke* and *IdTradeMarks*.

### 3.1.2 Char, Text, Varchar

Char,Text and Varchar are ways to memorise a string in the database and each has pro and cons. Text is not good to memorise single or few words because it has been thought for storing text so the two remains are: Char and Varchar[2].

#### Pro and cons of char and varchar

- CHAR
  - Used to store the string value of fixed length characters.
  - The maximum n. of characters that the data can contain is 255 characters.
  - It is 50% faster than VARCHAR.
  - Use static memory allocation.
- VARCHAR
  - Used to store alphanumeric data of variable length;
  - The maximum that this type of data can contain is up to:
    - Pre-MySQL 5.0.3: 255 characters.
    - In MySQL 5.0.3+: 65,535 characters shared by line.
  - It is slower than CHAR.
  - Use dynamic memory allocation.

Considering the pro and cons the best way to store the information is char, because is faster than varchar and it is not important to lose space in database because nowdays the hard disks have a lot of memory. Notwithstanding, for some fields it has been preferred to store using varchar because the memory lost could be too big. It has been used this type for the SHA field.

### 3.1.3 SHA

The SHA field is over-allocated in order to allow, one day, to change the way to create the digest without changing the database.

### 3.1.4 Image storage

The images have been saved in the database and not in the file system, although the application is not a crucial application (such as those of hospitals) but in this way it is possible to use all the transaction properties that the databases can give us. The image are stored using the MEDIUMBLOB type. For MEDIUMBLOB is not possible to use unique like attribute so it has been used the SHA512 in order to distinguish an image from another one.

### 3.1.5 User table

The user table contains the user's information: the user name that should not exceed 50 characters, the password, which is also not longer than 50 characters but must have at least one number, upper and lower case letter and finally the salt for the creation of the digest itself (more in details the random salt). The password has been stored in the database as a digest composed of the password chained to a random salt (randomly calculated for each new password) and a fixed digest that is stored in the PHP code. In which way it's more complex to crack the user password, because it's necessary to get the PHP code and the database in order to have all the elements for starting to search the real password (with a dictionary attack).

### 3.1.6 Language

The platform contains packaging cards translated in different languages. In order to do it, the tables that have to translate its elements contain specific attributes.

The attributes are:

- *IdLanguage*
- *CodeLanguage* plus name of table (without the "s")

*IdLanguage* is a foreign key on language table in order to know the effective language. The other term (*CodeLanguage*) is used to link a term with others terms inside the table itself.

The tables 3.1 and 3.2 (or the fig 3.1)) are an example of the "code language" use. More in detail, if the color is inserted for a packaging in automatically it is possible to translate this color and to know which language belongs to.

| IdColor | Name field | IdLanguage | CodeLanguage |
|---|---|---|---|
| 1 | Bianco | 1 | 1 |
| 2 | White | 2 | 1 |
| 3 | Blanco | 3 | 1 |
| 4 | Black | 2 | 2 |
| 5 | Nero | 1 | 2 |

Table 3.1 – Code language's example

| IdLanguage | Language |
|---|---|
| 1 | IT |
| 2 | GB |
| 3 | ES |
| 4 | FR |
| 5 | HU |

Table 3.2 – Language's example

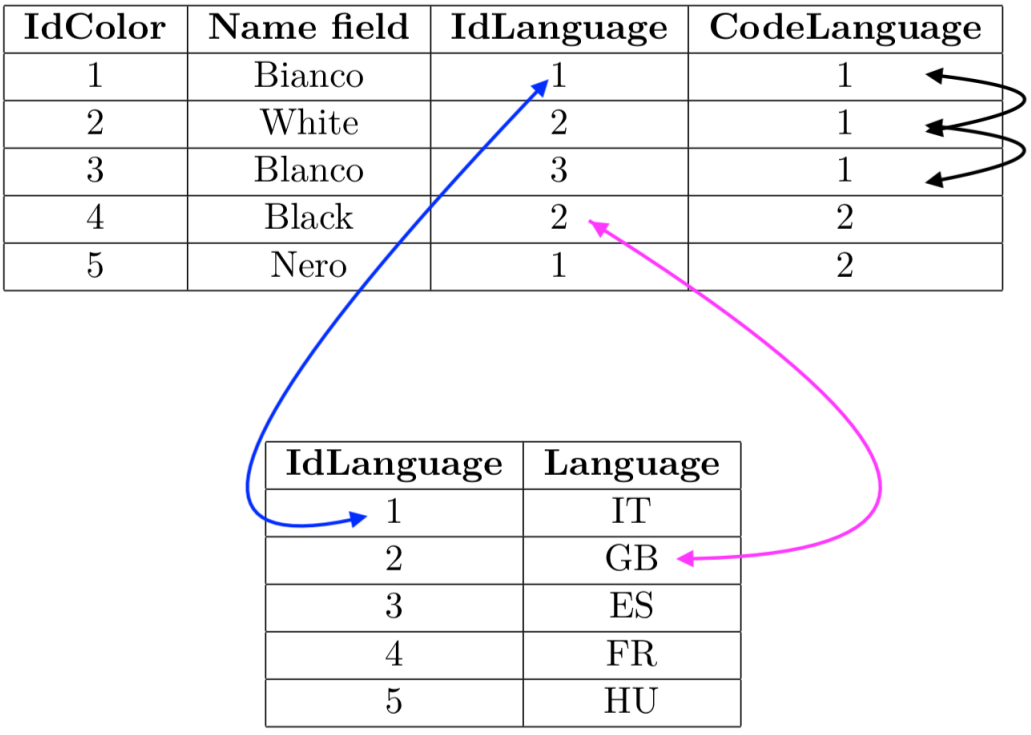


Figure 3.1 – Example of code language

## 3.2 List of tables

### 3.2.1 The tables

The tables are:

- Rating table for
  - Optimization of storage spaces.
  - Practicality of use.
  - Protection and conservation of the product.

  (table's name is *AListRatingFunctionalityAndVolumeWeights*).
- Rating table for presence of overpacking (table's name is *BListRatingOverpacks*).
- Rating table for Composition and separability of materials (table's name is *CListRatingCompositionSeparabilities*).
- Color table (table's name is *Colors*).
- Image detail table (table's name is *(*CommunicationImage)).
- Company table (table's name is *Companies*).
- Component table (table's name is *Components*).
- Description image table (table's name is *DescriptionImages*).
- Designer table (table's name is *Designers*).
- End live table (table's name is *EndLives*).
- Font table (table's name is *Fonts*).
- Hexadecimal colors live table (table's name is *HexadecimalColors*).
- Image table (table's name is *Images*).
- Language table (table's name is *Languages*).
- Manufacturer table (table's name is *Manufacturers*).
- Material table (table's name *Materials*).
- Nation table (table's name is *Nations*).
- Packet table (table's name is *Packets*).
- Phrase table (table's name is *Phrases*).
- Product sector table (table's name is *ProductSectors*).

- Production year table(table's name is *ProductionYears*).
- Trademark table (table's name is *Trademarks*).
- Type images table (table's name is *TypeImages*).
- Type weights table(table's name is *TypeWeights*).
- Typology table (table's name is *Typologies*).
- User table (table's name is *Users*).

### 3.2.2 The relation tables

The relation tables are:

- Relation table between packet table and color table (table's name is *PacketsColors*).
- Relation among packet table, image table and type image table(table's name is *PacketCommunicationImage*).
- Relation table among packet table, component table and material table (table's name is *PacketsComponentsMaterials*).
- Relation between packet table and designer table (table's name is *PacketsDesigners*).
- Relation among packet table, image table and type image table(table's name is *PacketsImagesTypes*).
- Relation between packet table and trademark table(table's name is *PacketsTrademarks*).

## 3.3 ER-UML diagram

In this section there are some images of database to understand in a better way. For pagination (layout)'s problems, there are some images that focus on some different aspects. Under each image there is a description for it.

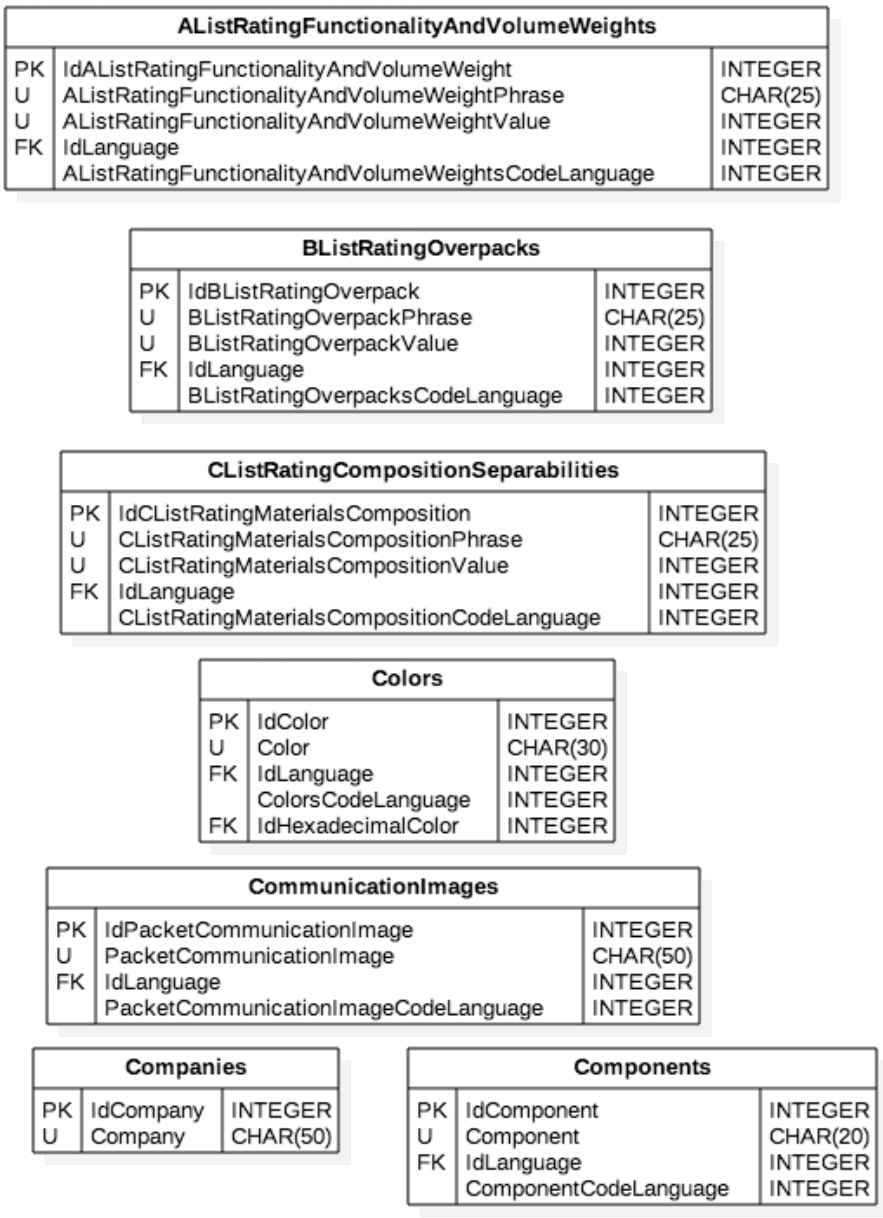


Figure 3.2 – Simple tables part one

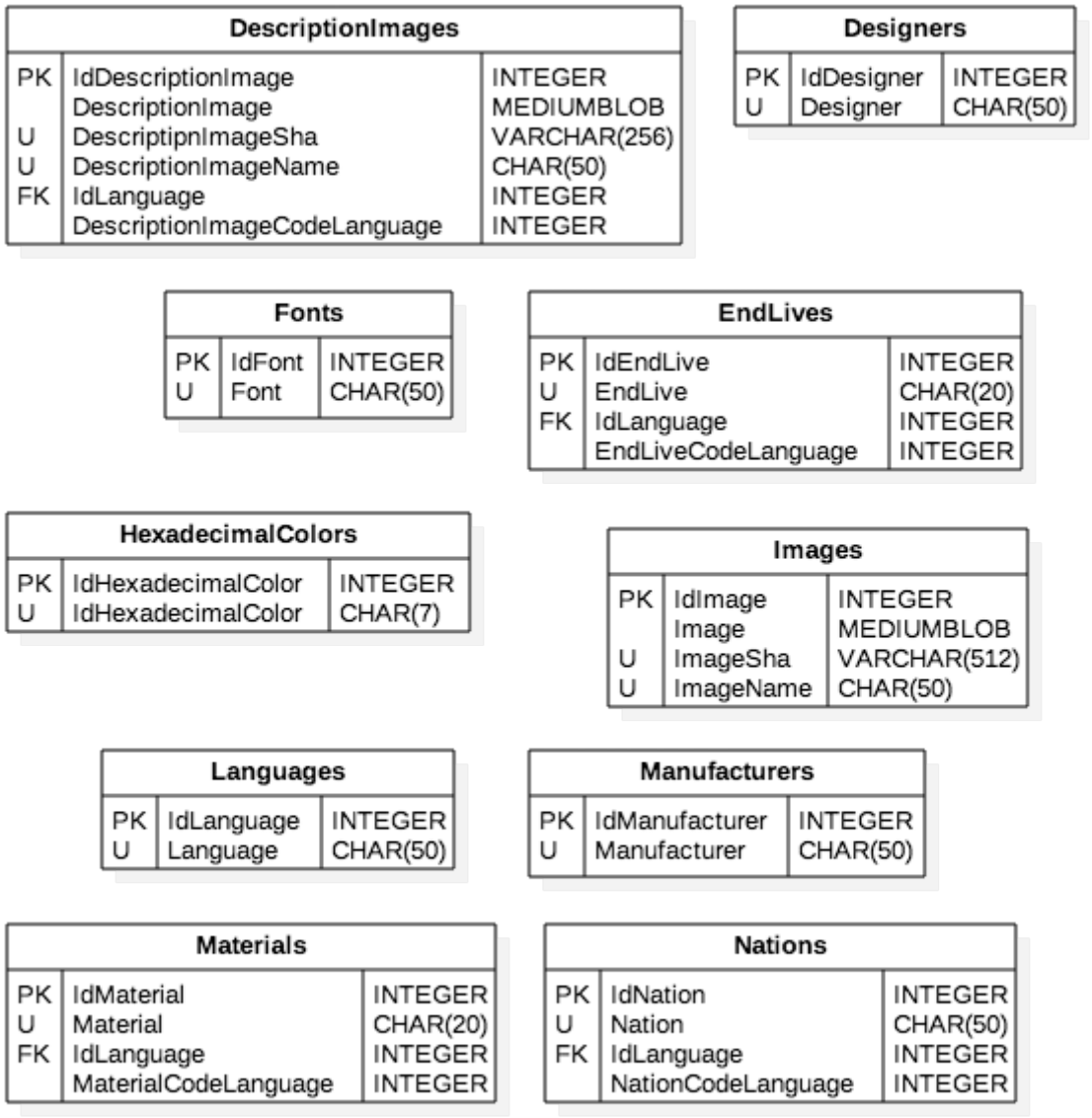


Figure 3.3 – Simple table part 2

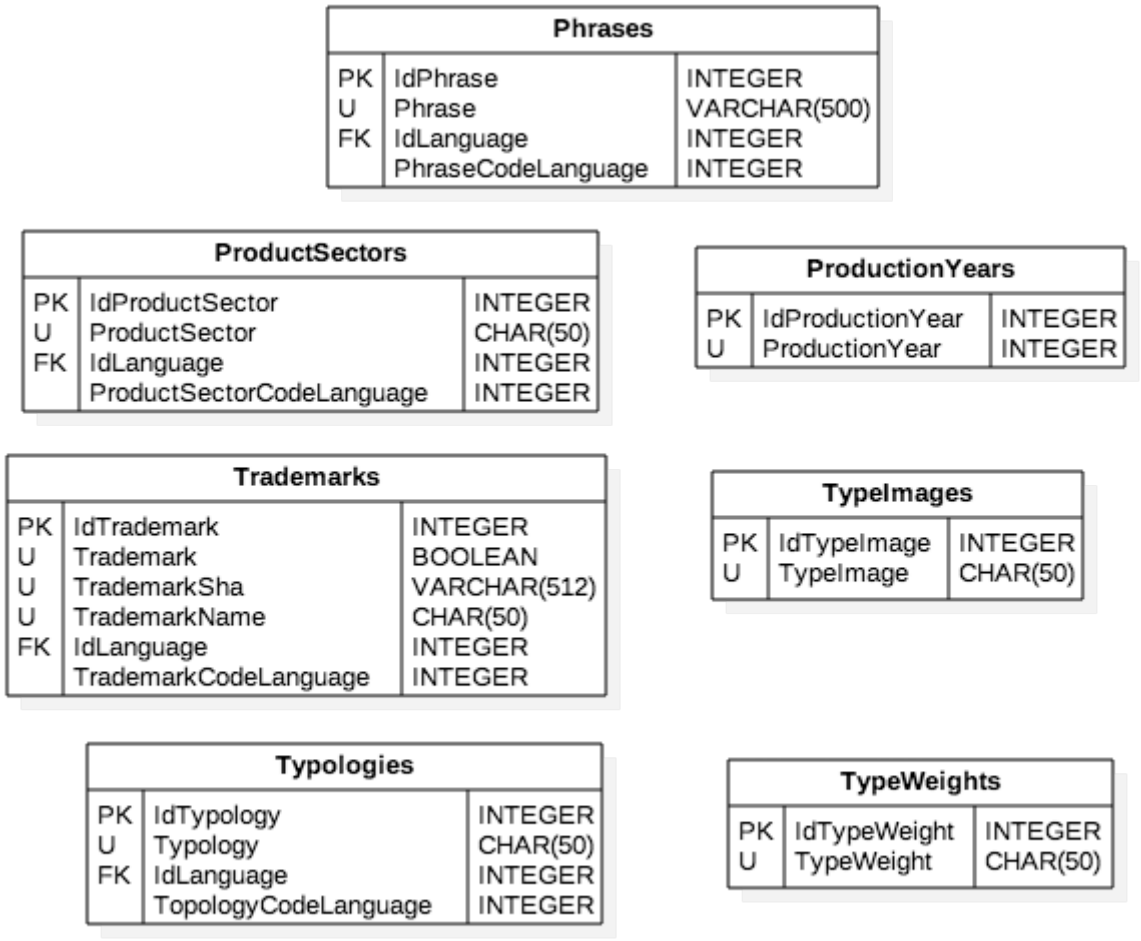


Figure 3.4 – Packet table

| Packets | | |
|---|---|---|
| PK | IdPacket | INTEGER |
| U | PacketName | CHAR(50) |
| FK,N | IdCompany | INTEGER |
| FK,N | IdManufacturer | INTEGER |
| FK,N | IdNation | INTEGER |
| FK,N | IdProductSector | INTEGER |
| FK,N | IdProductionYear | INTEGER |
| FK,N | IdTypology | INTEGER |
| N | PacketFunctionalityAverage | INTEGER |
| N | PacketSustainabilityAverage | INTEGER |
| N | PacketCommunicationAverage | INTEGER |
| N | PacketLength | INTEGER |
| N | PacketHeight | INTEGER |
| N | PacketWeight | INTEGER |
| N | PacketVolume | INTEGER |
| N | PacketPrevalentCommunicationInteger | INTEGER |
| N | PacketPrevalentCommunicationValue | INTEGER |
| N | PacketTotalWeightComponent | INTEGER |
| FK,N | ARatingSpaceOptimization | INTEGER |
| FK,N | ARatingProtectionConservation | INTEGER |
| FK,N | ARatingPracticalityUse | INTEGER |
| FK,N | BRatingPresenceOverpacking | INTEGER |
| FK,N | CRatingCompositionSeparability | INTEGER |
| FK,N | ARatingWeightVolume | INTEGER |
| FK,N | ARatingAppeal | INTEGER |
| FK,N | ARatingIdentity | INTEGER |
| FK,N | ARatingMessage | INTEGER |
| FK,N | ARatingInformation | INTEGER |
| FK,N | ARatingAffordance | INTEGER |
| N | PacketPercentInformation | INTEGER |
| N | PacketPercentCommunication | INTEGER |
| N | PacketPercentNeutralArea | INTEGER |
| N | PacketSensoriality | BOOLEAN |
| FK,N | IdDescriptionImage | INTEGER |
| FK,N | IdEndLife | INTEGER |
| FK,N | IdFont | INTEGER |
| FK,N | IdPhrase | INTEGER |
| N | PacketTotalRating | INTEGER |
| | PacketDraft | BOOLEAN |

Figure 3.5 – Packet table

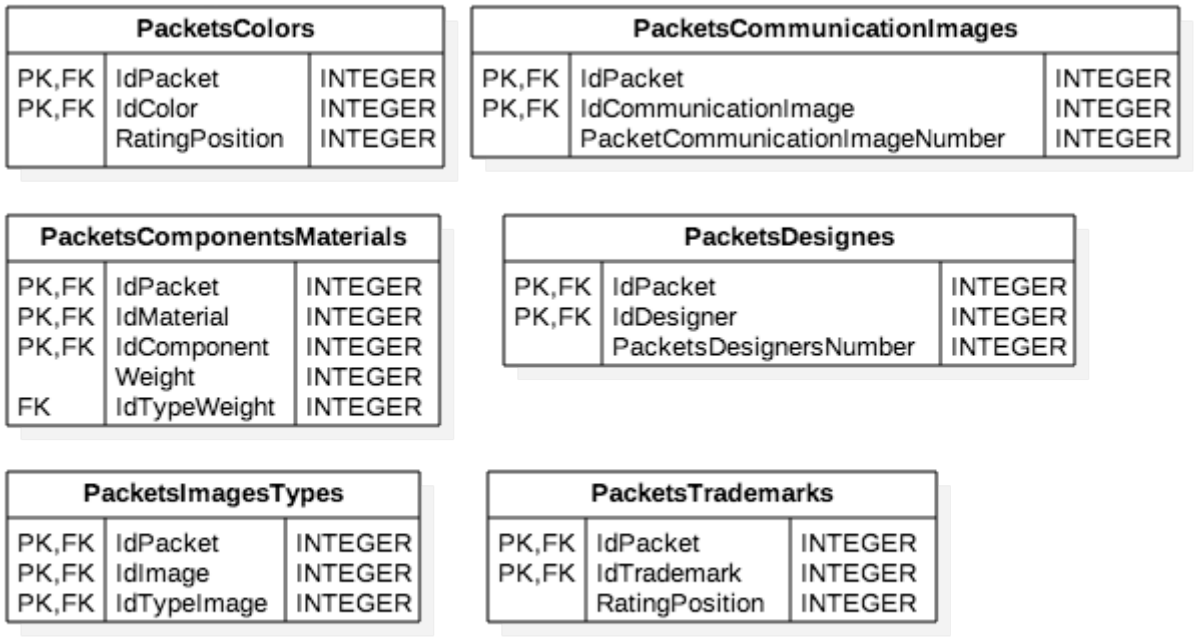


Figure 3.6 – Relation tables

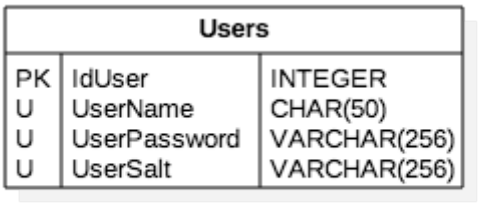


Figure 3.7 – User table

## 3.4 SQL CODE

The section wishes to describe the SQL code using in order to create the tables.

### 3.4.1 AListRatingFunctionalityAndVolumeWeights

The *AListRatingFunctionalityAndVolumeWeights* table memorises the possible rating phrases that the packet can have about its weight and pack / product volume optimization. The SQL code is:

```
1  CREATE TABLE AListRatingFunctionalityAndVolumeWeights (
2   IdAListRatingFunctionalityAndVolumeWeight INTEGER NOT NULL
       AUTO_INCREMENT,
3   AListRatingFunctionalityAndVolumeWeightPhrase CHAR(50) NOT NULL,
4   AListRatingFunctionalityAndVolumeWeightValue INTEGER NOT NULL,
5   IdLanguage INTEGER NOT NULL,
6   AListRatingFunctionalityAndVolumeWeightsCodeLanguage INTEGER NOT
       NULL,
7   PRIMARY KEY (IdAListRatingFunctionalityAndVolumeWeight),
8   UNIQUE (AListRatingFunctionalityAndVolumeWeightPhrase,
       AListRatingFunctionalityAndVolumeWeightValue)
9  );
10
11    ALTER TABLE AListRatingFunctionalityAndVolumeWeights ADD FOREIGN
         KEY (IdLanguage) REFERENCES Languages(IdLanguage);
```

### 3.4.2 BListRatingOverpacks

The table memorises the possible rating phrases that packet can have about its overpack. The SQL code is:

```
1  CREATE TABLE BListRatingOverpacks (
2   IdBListRatingOverpack INTEGER NOT NULL AUTO_INCREMENT,
3   BListRatingOverpackPhrase CHAR(50) NOT NULL,
4   BListRatingOverpackValue INTEGER NOT NULL,
5   IdLanguage INTEGER NOT NULL,
6   BListRatingOverpacksCodeLanguage INTEGER NOT NULL,
7   PRIMARY KEY (IdBListRatingOverpack),
8   UNIQUE (BListRatingOverpackPhrase, BListRatingOverpackValue)
9  );
10
11 ALTER TABLE BListRatingOverpacks ADD FOREIGN KEY (IdLanguage)
      REFERENCES Languages(IdLanguage);
```

### 3.4.3 CListRatingCompositionSeparabilities

The table memorises the possible rating phrases that packet can have about its composition and separability. The SQL code is:

```
1  CREATE TABLE BListRatingOverpacks (
2   IdBListRatingOverpack INTEGER NOT NULL AUTO_INCREMENT,
3   BListRatingOverpackPhrase CHAR(50) NOT NULL,
4   BListRatingOverpackValue INTEGER NOT NULL,
5   IdLanguage INTEGER NOT NULL,
6   BListRatingOverpacksCodeLanguage INTEGER NOT NULL,
7  PRIMARY KEY (IdBListRatingOverpack),
8  UNIQUE (BListRatingOverpackPhrase, BListRatingOverpackValue)
9  );
10
```

```
11 ALTER TABLE CListRatingCompositionSeparabilities ADD FOREIGN KEY (
       IdLanguage) REFERENCES Languages(IdLanguage);
```

### 3.4.4 Colors

The table stores the names of colors by their hexadecimal color number (this value is a link to hexadecimal table) The SQL code is:

```
1  CREATE TABLE Colors (
2   IdColor INTEGER NOT NULL AUTO_INCREMENT,
3   Color CHAR(30) NOT NULL,
4   IdLanguage INTEGER NOT NULL,
5   ColorsCodeLanguage INTEGER,
6   IdHexadecimalColor INTEGER NOT NULL,
7   PRIMARY KEY (IdColor),
8   UNIQUE (Color)
9  );
10
11 ALTER TABLE Colors ADD FOREIGN KEY (IdLanguage) REFERENCES
       Languages(IdLanguage);
12
13 ALTER TABLE Colors ADD FOREIGN KEY (IdHexadecimalColor) REFERENCES
       HexadecimalColors(IdHexadecimalColor);
```

### 3.4.5 CommunicationImages

The table stores all elements about the communication images, that stay in the third part of mock-up. The SQL code is:

```
1  CREATE TABLE CommunicationImages  (
2   IdCommunicationImage  INTEGER NOT NULL AUTO_INCREMENT,
3   CommunicationImage CHAR(50) NOT NULL,
4   IdLanguage INTEGER NOT NULL,
5   CommunicationImageCodeLanguage INTEGER NOT NULL,
6   PRIMARY KEY (IdCommunicationImage),
7   UNIQUE (CommunicationImage)
8  );
9
10 ALTER TABLE CommunicationImages ADD FOREIGN KEY (IdLanguage)
       REFERENCES Languages(IdLanguage);
```

### 3.4.6 Companies

The table stores all the possible companies. The SQL code is:

```
1  CREATE TABLE Companies (
2   IdCompany INTEGER NOT NULL AUTO_INCREMENT,
3   Company CHAR(50) NOT NULL,
4   PRIMARY KEY (IdCompany),
```

```
5   UNIQUE (Company)
6  );
```

### 3.4.7 Components

The table stores all possible components like a cap of bottle. The SQL code is:

```
1  CREATE TABLE Components (
2   IdComponent INTEGER NOT NULL AUTO_INCREMENT,
3   Component CHAR(20) NOT NULL,
4   IdLanguage INTEGER,
5   ComponentCodeLanguage INTEGER,
6   PRIMARY KEY (IdComponent),
7   UNIQUE (Component)
8  );
9
10 ALTER TABLE Components ADD FOREIGN KEY (IdLanguage) REFERENCES
       Languages(IdLanguage);
```

### 3.4.8 DescriptionImages

The table memorises the images that have some captions, that stay on the third part of mock-up. The SQL code is:

```
1  CREATE TABLE DescriptionImages (
2   IdDescriptionImage INTEGER NOT NULL AUTO_INCREMENT,
3   DescriptionImage MEDIUMBLOB,
4   DescriptionImageSha VARCHAR(512) NOT NULL,
5   DescriptionImageName CHAR(50),
6   IdLanguage INTEGER,
7   DescriptionImageCodeLanguage INTEGER,
8   PRIMARY KEY (IdDescriptionImage),
9   UNIQUE (DescriptionImageName)
10 );
11
12 ALTER TABLE DescriptionImages ADD FOREIGN KEY (IdLanguage)
       REFERENCES Languages(IdLanguage);
```

### 3.4.9 Designers

The table memorises all the name designers. The SQL code is:

```
1  CREATE TABLE Designers (
2   IdDesigner INTEGER NOT NULL AUTO_INCREMENT,
3   Designer CHAR(50) NOT NULL,
4   PRIMARY KEY (IdDesigner),
5   UNIQUE (Designer)
6  );
```

### 3.4.10 EndLives

The table stores the possible end lives of the packagings (e.g standard). The SQL code is:

```
CREATE TABLE EndLives (
 IdEndLife INTEGER NOT NULL AUTO_INCREMENT,
 EndLife CHAR(20) NOT NULL,
 IdLanguage INTEGER,
 EndLifeCodeLanguage INTEGER,
 PRIMARY KEY (IdEndLife),
 UNIQUE (EndLife)
);

ALTER TABLE EndLives ADD FOREIGN KEY (IdLanguage) REFERENCES
    Languages(IdLanguage);
```

### 3.4.11 Fonts

The table stores all possible fonts that the packagings can have. The SQL code is:

```
CREATE TABLE Fonts (
 IdFont INTEGER NOT NULL AUTO_INCREMENT,
 Font CHAR(50) NOT NULL,
 PRIMARY KEY (IdFont),
 UNIQUE (Font)
);
```

### 3.4.12 HexadecimalColors

The table memorises all possible the hexadecimal color numbers. The SQL code is:

```
CREATE TABLE HexadecimalColors(
 IdHexadecimalColor INTEGER NOT NULL AUTO_INCREMENT,
 HexadecimalColor CHAR(7) NOT NULL,
 PRIMARY KEY (IdHexadecimalColor),
 UNIQUE (HexadecimalColor)
);
```

### 3.4.13 Images

The table stores all those images that do not need to be translated in other languages. The SQL code is:

```
CREATE TABLE Images (
IdImage INTEGER NOT NULL AUTO_INCREMENT,
Image MEDIUMBLOB,
ImageName CHAR(50),
ImageSha VARCHAR(512) NOT NULL,
PRIMARY KEY (IdImage),
UNIQUE (ImageSha, ImageName)
```

```
8 );
```

### 3.4.14 Languages

The table stores the language names (e.g IT for italian, and so on...). The SQL code is:

```
1 CREATE TABLE Languages (
2  IdLanguage INTEGER NOT NULL AUTO_INCREMENT,
3  Language CHAR(50) NOT NULL,
4  PRIMARY KEY (IdLanguage),
5  UNIQUE (Language)
6 );
```

### 3.4.15 Manufacturers

The table stores the names of all the manufacturers. The SQL code is:

```
1 CREATE TABLE Manufacturers (
2  IdManufacturer INTEGER NOT NULL AUTO_INCREMENT,
3  Manufacturer CHAR(50) NOT NULL,
4  PRIMARY KEY (IdManufacturer),
5  UNIQUE (Manufacturer)
6 );
```

### 3.4.16 Materials

The table stores the materials. The SQL code is:

```
1 CREATE TABLE Materials (
2  IdMaterial INTEGER NOT NULL AUTO_INCREMENT,
3  Material CHAR(20) NOT NULL,
4  IdLanguage INTEGER NOT NULL,
5  MaterialCodeLanguage INTEGER NOT NULL,
6  PRIMARY KEY (IdMaterial),
7  UNIQUE (Material)
8 );
9 ALTER TABLE Materials ADD FOREIGN KEY (IdLanguage) REFERENCES
     Languages(IdLanguage);
```

### 3.4.17 Nations

The table stores the nations. The SQL code is:

```
1 CREATE TABLE Nations (
2  IdNation INTEGER NOT NULL AUTO_INCREMENT,
3  Nation CHAR(50)NOT NULL,
4  IdLanguage INTEGER,
5  NationCodeLanguage INTEGER,
6  PRIMARY KEY (IdNation),
```

```
 7   UNIQUE (Nation)
 8  );
 9
10  ALTER TABLE Nations ADD FOREIGN KEY (IdLanguage) REFERENCES
       Languages(IdLanguage);
```

### 3.4.18 Packets

The table stores all the packagings with their name, their dimensions and all those ids that allow to get the right elements that are stored in others tables. The SQL code is:

```
 1  CREATE TABLE Packets (
 2   IdPacket INTEGER NOT NULL AUTO_INCREMENT,
 3   PacketName CHAR(50),
 4   IdCompany INTEGER,
 5   IdManufacturer INTEGER,
 6   IdNation INTEGER,
 7   IdProductionYear INTEGER,
 8   IdProductSector INTEGER,
 9   IdTypology INTEGER,
10   PacketFunctionalityAverage INTEGER,
11   PacketSustainabilityAverage INTEGER,
12   PacketCommunicationAverage INTEGER,
13   PacketLength INTEGER,
14   PacketHeight INTEGER,
15   PacketWeight INTEGER,
16   PacketVolume INTEGER,
17   PacketPrevalentCommunicationInteger INTEGER,
18   PacketPrevalentCommunicationValue INTEGER,
19   PacketTotalWeightComponent INTEGER,
20   ARatingSpaceOptimization INTEGER,
21   ARatingProtectionConservation INTEGER,
22   ARatingPracticalityUse INTEGER,
23   BRatingPresenceOverpacking INTEGER,
24   CRatingCompositionSeparability INTEGER,
25   ARatingWeightVolume INTEGER,
26   ARatingAppeal INTEGER,
27   ARatingIdentity INTEGER,
28   ARatingMessage INTEGER,
29   ARatingInformation INTEGER,
30   ARatingAffordance INTEGER,
31   PacketPercentInformation INTEGER,
32   PacketPercentCommunication INTEGER,
33   PacketPercentNeutralArea INTEGER,
34   PacketSensoriality BOOLEAN,
35   IdDescriptionImage INTEGER,
36   IdEndLife INTEGER,
37   IdFont INTEGER,
38   IdPhrase INTEGER,
```

```
39   PacketTotalRating INTEGER,
40   PacketDraft BOOLEAN NOT NULL,
41   PRIMARY KEY (IdPacket),
42   UNIQUE (PacketName)
43  );
44
45  ALTER TABLE Packets ADD FOREIGN KEY (IdCompany) REFERENCES
       Companies(IdCompany);
46
47  ALTER TABLE Packets ADD FOREIGN KEY (IdManufacturer) REFERENCES
       Manufacturers(IdManufacturer);
48
49  ALTER TABLE Packets ADD FOREIGN KEY (IdNation) REFERENCES Nations(
       IdNation);
50
51  ALTER TABLE Packets ADD FOREIGN KEY (IdProductionYear) REFERENCES
       ProductionYears(IdProductionYear);
52
53  ALTER TABLE Packets ADD FOREIGN KEY (IdProductSector) REFERENCES
       ProductSectors(IdProductSector);
54
55  ALTER TABLE Packets ADD FOREIGN KEY (IdTypology) REFERENCES
       Typologies(IdTypology);
56
57  ALTER TABLE Packets ADD FOREIGN KEY (ARatingSpaceOptimization)
       REFERENCES AListRatingFunctionalityAndVolumeWeights (
       IdAListRatingFunctionalityAndVolumeWeight);
58
59  ALTER TABLE Packets ADD FOREIGN KEY (ARatingProtectionConservation)
        REFERENCES AListRatingFunctionalityAndVolumeWeights (
       IdAListRatingFunctionalityAndVolumeWeight);
60
61  ALTER TABLE Packets ADD FOREIGN KEY (ARatingPracticalityUse)
       REFERENCES AListRatingFunctionalityAndVolumeWeights (
       IdAListRatingFunctionalityAndVolumeWeight);
62
63  ALTER TABLE Packets ADD FOREIGN KEY (BRatingPresenceOverpacking)
       REFERENCES BListRatingOverpacks(IdBListRatingOverpack);
64
65  ALTER TABLE Packets ADD FOREIGN KEY (CRatingCompositionSeparability
       ) REFERENCES CListRatingCompositionSeparabilities (
       IdCListRatingMaterialsComposition);
66
67  ALTER TABLE Packets ADD FOREIGN KEY (ARatingWeightVolume)
       REFERENCES AListRatingFunctionalityAndVolumeWeights (
       IdAListRatingFunctionalityAndVolumeWeight);
68
```

```
69 ALTER TABLE Packets ADD FOREIGN KEY (ARatingAppeal) REFERENCES
       AListRatingFunctionalityAndVolumeWeights (
       IdAListRatingFunctionalityAndVolumeWeight);
70
71 ALTER TABLE Packets ADD FOREIGN KEY (ARatingIdentity) REFERENCES
       AListRatingFunctionalityAndVolumeWeights (
       IdAListRatingFunctionalityAndVolumeWeight);
72
73 ALTER TABLE Packets ADD FOREIGN KEY (ARatingMessage) REFERENCES
       AListRatingFunctionalityAndVolumeWeights (
       IdAListRatingFunctionalityAndVolumeWeight);
74
75 ALTER TABLE Packets ADD FOREIGN KEY (ARatingInformation) REFERENCES
        AListRatingFunctionalityAndVolumeWeights (
       IdAListRatingFunctionalityAndVolumeWeight);
76
77 ALTER TABLE Packets ADD FOREIGN KEY (ARatingAffordance) REFERENCES
       AListRatingFunctionalityAndVolumeWeights (
       IdAListRatingFunctionalityAndVolumeWeight);
78
79 ALTER TABLE Packets ADD FOREIGN KEY (IdDescriptionImage) REFERENCES
        DescriptionImages (IdDescriptionImage);
80
81 ALTER TABLE Packets ADD FOREIGN KEY (IdEndLife) REFERENCES EndLives
       (IdEndLife);
82
83 ALTER TABLE Packets ADD FOREIGN KEY (IdFont) REFERENCES Fonts(
       IdFont);
84
85 ALTER TABLE Packets ADD FOREIGN KEY (IdPhrase) REFERENCES Phrases(
       IdPhrase);
```

### 3.4.19 Phrases

The table stores the phrases that stay on the last part of the mock-up. The SQL code is:

```
1  CREATE TABLE Phrases (
2   IdPhrase INTEGER NOT NULL AUTO_INCREMENT,
3   Phrase VARCHAR(500) NOT NULL,
4   IdLanguage INTEGER NOT NULL,
5   PhraseCodeLanguage INTEGER NOT NULL,
6   PRIMARY KEY (IdPhrase),
7   UNIQUE (Phrase)
8  );
9
10 ALTER TABLE Phrases ADD FOREIGN KEY (IdLanguage) REFERENCES
       Languages(IdLanguage);
```

### 3.4.20 ProductSectors

The table stores all the product sectors (e.g pharmaceutical sector). The SQL code is:

```
1  CREATE TABLE ProductSectors (
2   IdProductSector INTEGER NOT NULL AUTO_INCREMENT,
3   ProductSector char(50) NOT NULL,
4   IdLanguage INTEGER,
5   ProductSectorCodeLanguage INTEGER,
6   PRIMARY KEY (IdProductSector),
7   UNIQUE (ProductSector)
8  );
9
10 ALTER TABLE ProductSectors ADD FOREIGN KEY (IdLanguage) REFERENCES
       Languages(IdLanguage);
```

### 3.4.21 ProductionYears

The table stores all the production years (e.g 2009). The SQL code is:

```
1  CREATE TABLE ProductionYears (
2   IdProductionYear INTEGER NOT NULL AUTO_INCREMENT,
3   ProductionYear INTEGER NOT NULL,
4   PRIMARY KEY (IdProductionYear),
5   UNIQUE (ProductionYear)
6  );
```

### 3.4.22 Trademarks

The table stores all the images of trademarks. The SQL code is:

```
1  CREATE TABLE Trademarks (
2   IdTrademark INTEGER NOT NULL AUTO_INCREMENT,
3   Trademark MEDIUMBLOB,
4   TrademarkSha VARCHAR(512) NOT NULL,
5   TrademarkName CHAR(50) NOT NULL,
6   IdLanguage INTEGER NOT NULL,
7   TrademarkCodeLanguage INTEGER NOT NULL,
8   PRIMARY KEY (IdTrademark),
9   UNIQUE (TrademarkSha, TrademarkName)
10 );
11
12 ALTER TABLE Trademarks ADD FOREIGN KEY (IdLanguage) REFERENCES
       Languages(IdLanguage);
```

### 3.4.23 TypeImages

The table stores all the image's types that are used in relation between packet and image table in order to distinguish an image from another image . The SQL code is:

```
1  CREATE TABLE TypeImages (
2   IdTypeImage INTEGER NOT NULL AUTO_INCREMENT,
3   TypeImage CHAR(50) NOT NULL,
4   PRIMARY KEY (IdTypeImage),
5   UNIQUE (TypeImage)
6  );
```

### 3.4.24 TypeWeights

The table stores all weight' s types that are used in relation among packet, component and material table. The SQL code is:

```
1  CREATE TABLE TypeWeights (
2   IdTypeWeight INTEGER NOT NULL AUTO_INCREMENT,
3   TypeWeight CHAR(50) NOT NULL,
4   PRIMARY KEY (IdTypeWeight),
5   UNIQUE (TypeWeight)
6  );
```

### 3.4.25 Typologies

The table stores all the typologies (e.g bottle). The SQL code is:

```
1  CREATE TABLE Typologies (
2   IdTypology INTEGER NOT NULL AUTO_INCREMENT,
3   Typology CHAR(50) NOT NULL,
4   IdLanguage INTEGER,
5   TypologyCodeLanguage INTEGER,
6   PRIMARY KEY (IdTypology),
7   UNIQUE (Typology)
8  );
9
10 ALTER TABLE Typologies ADD FOREIGN KEY (IdLanguage) REFERENCES
       Languages(IdLanguage);
```

### 3.4.26 Users

The table stores all users. The SQL code is:

```
1  CREATE TABLE Users(
2   IdUser INTEGER NOT NULL AUTO_INCREMENT,
3   UserName CHAR(50) NOT NULL,
4   UserPassword VARCHAR(256) NOT NULL,
5   UserSalt VARCHAR(256) NOT NULL,
6   PRIMARY KEY (IdUser),
7   UNIQUE (UserName)
8  );
```

### 3.4.27 PacketsColors

The SQL code is:

```
1  CREATE TABLE PacketsColors (
2   IdPacket INTEGER NOT NULL,
3   IdColor INTEGER NOT NULL,
4   PacketsColorRatingPosition INTEGER NOT NULL,
5   PRIMARY KEY (IdPacket, IdColor)
6  );
7
8  ALTER TABLE PacketsColors ADD FOREIGN KEY (IdPacket) REFERENCES
       Packets(IdPacket);
9
10 ALTER TABLE PacketsColors ADD FOREIGN KEY (IdColor) REFERENCES
       Colors(IdColor);
```

### 3.4.28 PacketsCommunicationImages

The table memorises the relation between packet and communication image table. The SQL code is:

```
1  CREATE TABLE PacketsCommunicationImages (
2  IdPacket INTEGER NOT NULL,
3  IdCommunicationImage INTEGER NOT NULL,
4  PRIMARY KEY (IdPacket, IdCommunicationImage)
5  );
6
7  ALTER TABLE PacketsCommunicationImages ADD FOREIGN KEY (IdPacket)
       REFERENCES Packets(IdPacket);
8
9  ALTER TABLE PacketsCommunicationImages ADD FOREIGN KEY (
       IdCommunicationImage) REFERENCES CommunicationImages(
       IdCommunicationImage);
```

### 3.4.29 PacketsComponentsMaterials

The table memorises the relation among packet, component and material table. The SQL code is:

```
1  CREATE TABLE PacketsComponentsMaterials (
2   IdPacket INTEGER NOT NULL,
3   IdMaterial INTEGER NOT NULL,
4   IdComponent INTEGER NOT NULL,
5   PacketsComponentsMaterialWeight INTEGER NOT NULL,
6   IdTypeWeight INTEGER NOT NULL,
7   PRIMARY KEY (IdPacket, IdMaterial, IdComponent)
8  );
9  ALTER TABLE PacketsComponentsMaterials ADD FOREIGN KEY (IdPacket)
       REFERENCES Packets(IdPacket);
```

```
10
11 ALTER TABLE PacketsComponentsMaterials ADD FOREIGN KEY (IdComponent
       ) REFERENCES Components(IdComponent);
12
13 ALTER TABLE PacketsComponentsMaterials ADD FOREIGN KEY (IdMaterial)
        REFERENCES Materials(IdMaterial);
14
15 ALTER TABLE PacketsComponentsMaterials ADD FOREIGN KEY (
       IdTypeWeight) REFERENCES TypeWeights(IdTypeWeight);
```

### 3.4.30 PacketsDesigners

The table memorises the relation between packet and designer table. The SQL code is:

```
1  CREATE TABLE PacketsDesigners (
2   IdPacket INTEGER NOT NULL,
3   IdDesigner INTEGER NOT NULL,
4   PacketsDesignersNumber INTEGER NOT NULL,
5   PRIMARY KEY (IdPacket, IdDesigner)
6  );
7
8  ALTER TABLE PacketsDesigners ADD FOREIGN KEY (IdPacket) REFERENCES
       Packets(IdPacket);
9
10 ALTER TABLE PacketsDesigners ADD FOREIGN KEY (IdDesigner)
       REFERENCES Designers(IdDesigner);
```

### 3.4.31 PacketsImagesTypes

The table memorises the relation among packet. The SQL code is:

```
1  CREATE TABLE PacketsImagesTypes (
2   IdPacket INTEGER NOT NULL,
3   IdImage INTEGER NOT NULL,
4   IdTypeImage INTEGER NOT NULL,
5   PRIMARY KEY (IdPacket, IdImage, IdTypeImage)
6  );
7  ALTER TABLE PacketsImagesTypes ADD FOREIGN KEY (IdPacket)
       REFERENCES Packets(IdPacket);
8
9  ALTER TABLE PacketsImagesTypes ADD FOREIGN KEY (IdImage) REFERENCES
        Images(IdImage);
10
11 ALTER TABLE PacketsImagesTypes ADD FOREIGN KEY (IdTypeImage)
       REFERENCES TypeImages(IdTypeImage);
```

### 3.4.32 PacketsTrademarks

The table memorises the relation between packet and trademark table. The SQL code is:

```
1 CREATE TABLE PacketsTrademarks (
2 IdPacket INTEGER NOT NULL,
3 IdTrademark INTEGER NOT NULL,
4 PRIMARY KEY (IdPacket, IdTrademark)
5 );
6
7 ALTER TABLE PacketsTrademarks ADD FOREIGN KEY (IdPacket) REFERENCES
      Packets(IdPacket);
8
9 ALTER TABLE PacketsTrademarks ADD FOREIGN KEY (IdTrademark)
    REFERENCES Trademarks(IdTrademark);
```

# Chapter 4

# Back-end

The chapter gives an explanation of the back-end part of the project that has been written in PHP without framework.

## 4.1 Language

This section wants to explain the reason for certain choices.

### 4.1.1 PHP Vs Java

There are more or less two ways to write back-end:

- Java
- PHP

Both options are good. However, not knowing who will have to take care of the project in the future and being the project in working progress has been chosen PHP. The PHP is easier to learn and to develop with it, so it is conceivable, that everybody could continue to develop the project without strong difficulty. In addition there is a very good documentation for PHP with a lot of examples on the ufficial web site `http://php.net/` and in others web site.

### 4.1.2 PHP framework

There are several PHP frameworks for back-end, like for example:

- Slim[4]
- Laravel[5]
- CakePHP[6]
- Phalcon[7]

- ecc . .

Each framework has its potentials and its problems but all must always be maintained by third parties and updated on the deployment server. It's true that is easier to adopt a framework like Laravel in order to write faster, simpler and in secure way, nevertheless for first two reasons (framework's maintenance and updating on the server) it has been chosen to develop the back-end without framework in order to be free and to have the maximum of liberty to adopt the own structures and choices without depending by anybody. For instance, in this way it is possible to change, update (or downgrade) the code in relationship with the version of PHP that the server uses.

## 4.2 High view back-end design

The back-end is formed by PHP classes and PHP files. More in details the structure is the following:

- *database-iteraction* folder;
- *page-do-class* folder;
- *select-request* folder;
- *switch-ajax-listening* folder;
- *utility* folder;
- *include-const-verb-ajax.php* file;
- *include-files-database-and-generate-class.php* file;
- *require-file.php* file;
- *return-json.php* file.

In every folder there are only classes, except for *switch-ajax-listening* folder where there is an unique file PHP.

All the names of the classes that are in the folders (like *database-iteraction* folder ) starts with the first part of the name's folder. And each folder starts with a different letter. In order to understand this choice, it has to watch the autoload of PHP in the back-end (Require-file.php).

The back-end's structure is very very simple for tow main reasons:

- all the project has been done following the KISS principle[1];
- the project will be continued and maintained by several programmers that don't have contributed to create so in this way it is simpler to learn/understand.

### 4.2.1 How back end works

The back-end and front-end only communicate using POST http's verb sending a JSON that contains some fields. The back-end has one entry point for receiving the JSON. In other words all the messages that the client sent to server go to *switch-ajax-listening* folder where there is a PHP file (*switch-ajax-listening.php*. In this file there are the first checks on the JSON field and if packet (JSON message) passes the checkpoints, the file builds a specific class in relationship that the client wishes and can do. If the JSON doesn't pass the controls the file return to client an JSON where inside there is an error code/message.

### 4.2.2 Why JSON?

It has been considered JSON and XML in order to send and to receive the data. Each data's format has own properties, advantages and disadvantages. At the end the best choice is to use the JSON because it uses less bandwidth and it's easier to parse and use (in according to KISS approach[14]).

## 4.3 Database-iteraction folder

The *database-iteraction* folder contains all the simplest element that the back-end can use in order to interact with the database. More in details the folder contains the following classes:

- *Database_connection.php*
- *Database_list_query_read.php*
- *Database_list_query_write.php*
- *Database_list_table_and_attribute.php*
- *Database_read_write.php*

Theses classes have been created in order to give the possibility to change the database without changing PHP code, because if the database changes (within certain limits) it is sufficient to change some of them.

### 4.3.1 Database_connection class

The class has the simple method in order to

- connect with the database with*dbCConnectToDatabase()* method;
- set rollback with*dbCSetAutoCommit($flag)* method;
- commit with *dbCCommit()* method;
- roolback with *dbCRoolBack()* method;

- launch a query and to get the result with *dbCQueryToDatabase($query, &$result)* method;
- close connection with *dbCCloseConnection()* method;
- ping the database with *dbCPingToDatabse()* method.

But it allows another configuration that it could use in the future:

- get and set the host with *getHost()* and *setHost(string $host)* method;
- get and set the name of database with *getDbName()* and *setDbName(string $dbName)* method;
- get and set the user of database with *getDbUserRoot()* and *setDbUserRoot(string $dbUserRoot)* method;
- get and set the database's password with *getPassword()* and *setPassword(string $password)* method;
- get and set the connection with the database with *getConn()* and *setConn(string $connection)* method.

All these methods cannot be directly used by back-end but they are incapsulated in others methods of others classes *Database_read_write()* in order to have very very few and simply methods.

### 4.3.2 Database_list_query_read class

The *Database_list_query_read class* class is able to return all queries that the back-end can do in order to read the database.

The methods can be divided in three big groups:

- constructor of the class:
  - *___construct*
- list query
  - *getGeneralQueryResultTable*
  - *getGeneralQueryResultTable2*
- dynamic query
  - *createQueryToGetValueAroundValue*
  - *creaateQuery*

More in details:

- *___construct()* method. This method creates the object itself and a new object from *Database_list_table_and_attribute()* class that has inside all the attributes and tables' names of the database.

- *getGeneralQueryResultTable($whichQuery, $elementToSearch, &$result, $idLanguage)* method. The method contains a swicth/case block that is used to select the right query by *whichQuery* in this way the method returns right query . In addition there are some parameters that it receives to have a minimum of dynamic. They are: *elementToSearch, result and idLanguage.*

  – *elementToSearch* is used when the select has to search some things (or it has a where clause);

  – *result* is a variable where is saved the result;

  – *idLanguage* is used in order to get the right language from the database.

  In order to better understand, the mains parts of the method and see the utilization of three variables:

```
1    public function getGeneralQueryResultTable($whichQuery,
         $elementToSearch, &$result, $idLanguage)
2  {
3    switch ($whichQuery)
4    {
5      case(self::idPacketFromPackets):
6      {
7        $result =
8        "SELECT " . $this->dbConst::IdPacket . " " .
9        " FROM " . $this->dbConst::Packets . " " .
10       "WHERE '" . $elementToSearch . "' = " .
11        $this->dbConst::PacketName . ";";
12       return true;
13     }
14   [...]
15   case(self::idNationFromNations):
16     {
17       $result =
18       "SELECT " . $this->dbConst::IdNation . " " .
19       " FROM " . $this->dbConst::Nations . " " .
20       "WHERE '" . $elementToSearch . "' = " .
21        $this->dbConst::NationsNation .
22       " AND " . $idLanguage . " = " .
23       $this->dbConst::NationsIdLanguage . ";";
24       return true;
25     }
26     [...]
27     default:
28     {
29       return false;
30     }
31   }
32 }
```

- *getGeneralQueryResultTable2($whichQuery, $elementToSearch1, $elementToSearch2, &$result)* method. This method is the same of *getGeneralQueryResultTable* but there are two differences:

  1. There are two elements inside the clause. There are two variables: *$elementToSearch1* and *$elementToSearch2*.
  2. There is not language field (*$idLanguage*).

  The main parts of the method:

```
1  public function getGeneralQueryResultTable2($whichQuery,
       $elementToSearch1, $elementToSearch2, &$result)
2  {
3    switch ($whichQuery)
4    {
5      case(self::getImageNameFromImagesByImageNameAndSha):
6      {
7        $result =
8        "SELECT " . $this->dbConst::ImageName . " " .
9        " FROM " . $this->dbConst::Images . " " .
10       "WHERE '" . $elementToSearch1 . "' = " . $this->
             dbConst::ImageName .
11       " AND  " . "'" . $elementToSearch2 . "' = " . $this->
             dbConst::ImageSha . ";";
12       return true;
13     }
14     [...]
15 }
```

- *createQueryToGetValueAroundValue($select, $table, $fieldWhere, $searchValue, $limit)* method. It is able to create a query in order to limit the result from the database. It is simpler to see the code:

```
1  public function createQueryToGetValueAroundValue($select, $table,
       $fieldWhere, $searchValue, $limit)
2  {
3    $query = "SELECT ". $select . " FROM " . $table . " WHERE " .
         $fieldWhere  . " LIKE '%". $searchValue ."%' LIMIT " .
         $limit;
4    return $query;
5  }
```

- *creaateQuery(array $select, array $tables, array $join, $orderBy, &$query)* method. The method receives some vectors:

  – for select *$select*. It identifies all select fields;
  – for tables *$tables*. It identifies all tables that the query has to use;
  – for join *$join*. It identifies all joins action that the query has to do.

  and orderBy element and the future result of the method. The method use another one (*part1Part2)*to don't repait the same code.

```
1  public function creaateQuery(array $select, array $tables, array
       $join, $orderBy, &$query)
2  {
3    $query = "SELECT ";
4    if ($select == null)
5    {
6      $query = $query . "*";
7    }
8    else if (count($select) == 0)
9    {
10     $query = $query . "*";
11   }
12   else
13     $this->part1Part2($select, $query);
14
15   if ($tables == null) return false;
16   if (count($tables) == 0) return false;
17
18   $query = $query . "FROM ";
19   $this->part1Part2($tables, $query);
20
21   if ($join == null)
22   {
23     $query = $query . ";";
24     return true;
25   }
26   if (count($join) == 0)
27   {
28     $query = $query . ";";
29     return true;
30   }
31   $query = $query . "WHERE ";
32
33   foreach ($join as $value)
34   {
35     $query = $query . $value . " ";
36   }
37
38   if ($orderBy == null)
39   {
40     $query = $query . ";";
41     return true;
42   }
43     $query = $query . "ORDER BY " . $orderBy . ";";
44   return true;
45   }
46
47 private function part1Part2(array $part, &$query)
48 {
49     $arrLength = count($part);
50   for ($i = 0; $i < $arrLength - 1; $i++)
51   {
52     $query = $query . $part[$i] . ", ";
53   }
```

```
54        $query = $query . $part[$arrLength - 1] . " ";
55    }
```

The classes also have some constants. These constants have been used in all the methods that return a "static" query ( *getGeneralQueryResultTable* *getGeneralQueryResultTable2*).

Example of constants:

```
1  //Get id from table
2  const idPacketFromPackets = "idPacketFromPackets";
3  const idCompanyFromCompanies = "idCompanyFromCompanies";
4  const idDesignerXFromDesigners = "idDesignerXFromDesigners";
5  const idImageFromImages = "idImageFromImages";
6  const idNationFromNations = "idNationFromNations";
7  const idManufacturerFromManufacturers = "idManufacturerFromManufacturer
       ";
8  const idSelectorYearFromProductionYears = "
       idSelectorYearFromProductionYears";
9  const idProductSectorFromProductSectors = "
       idProductSectorFromProductSectors";
10 const idTypologyFromTypologies = "idTypologyFromTypologies";
11 const idTypeImageFromTypeImage = "idTypeImageFromTypeImage";
12 [...]
```

### 4.3.3 Database_list_query_write class

The *Database_list_query_write* class is the same of *Database_list_query_read* class, but the last one (class) is able to create query in order to read the database, this class is able to create query for writing on the database.

The constructor of the class is *___construct()* and it has to create the object itself and a new object from *Database_list_table_and_attribute()* class has inside all the attributes and tables name of the database:

```
1  public function __construct()
2  {
3    $this->dbConst = new Database_list_table_and_attribute();
4  }
```

The others methods can be divided in three groups in relationship their actions on the database:

- Insert
- Delete
- Update

More in details:

**Insert method:** The insert method is *constQueryInsert($whichQuery, $value, &$query)* and the result of it is:

```
INSERT INTO table\_name (column1, column2, column3, ...) VALUES (
    value1, value2, value3, ...);
```

The method receives three variables *($whichQuery, $value, &$query)*, the first two contain the information in order to create the query and the last is the one where it saves the result. More in details the method has a list of inserts, it choses the right element (of the list) by *($whichQuery* and it inserts into the element of the list some fields that have given by *($value)*.

```
public function constQueryInsert($whichQuery, $value, &$query)
{
  switch($whichQuery)
  {
    case(self::insertIntoPacketsDesignes):
      {
      $query = "INSERT INTO " . $this->dbConst::PacketsDesigners . " ("
          .
      $this->dbConst::IdPacket . ", ".
      $this->dbConst::IdDesigner . ", ".
      $this->dbConst::PacketsDesignersNumber . ") VALUES ( ".
      $value[0] . ", " .
      $value[1] . ", " .
      $value[2] . ") ";
      return true;
    }
    [...]
    default:
    {
      return false;
    }
  }
}
```

**Delete method:** The method to delete is *genereteGeneralQueryForDeleteTuplaWithoutQuotationMarks* and it receives the name of the table *$table* and the values *$values*(with the corresponding names of the tuple *$field*) in vector form that has to delete. The method returns a query like this:

```
DELETE FROM Packet WHERE PacketName='Fox';
```

The method's code is:

```
public function genereteGeneralQueryForDeleteTuplaWithoutQuotationMarks
    ($table, $field, $values)
{
  $query = "DELETE FROM " . $table;
  $quer1 = $query . "  WHERE  ";
  $this->insertFieldEqualValueWithoutQuotationMarks( $field, $values,
      $query1);
  $query = $query1 . " ;";
```

```
7     return $query;
8   }
```

The method uses *insertFieldEqualValueWithoutQuotationMarks* in order to read and to insert the element that are inside the vector.

The method code is:

```
1   private function insertFieldEqualValueWithoutQuotationMarks ($field,
        $value, &$query)
2   {
3     $query1= $query;
4     for($i=0; $i<count($value)-1; $i++)
5     {
6       $query = $query1 . " " . $field[$i] . " = " . $value[$i];
7       $query1= $query;
8     }
9     $query1 = $query . " " . $value[count($value)-1];
10    $query=$query1;
11  }
```

**Update method:** The method for updating is *genereteGeneralQueryForUpdateTuplaWithoutQuotationMarks*. The function is similar to *genereteGeneralQueryForDeleteTuplaWithoutQuotationMarks* for exception that the result is different. For instance the result could be:

```
1   UPDATE PacketName SET PacketName = 'Fox'  WHERE PacketName = '
        Charly';
```

The code of the method is:

```
1   public function genereteGeneralQueryForUpdateTuplaWithoutQuotationMarks
        ($table, $field, $values, $fieldWhere, $valuesWhere)
2   {
3     $query = "UPDATE  " . $table . " SET ";
4     $this->insertFieldEqualValueWithoutQuotationMarks( $field, $values,
          $query);
5
6     $quer1 = $query . "  WHERE  ";
7     $this->insertFieldEqualValueWithoutQuotationMarks( $fieldWhere,
          $valuesWhere, $query1);
8     $query = $query1 . " ;";
9     return $query;
10  }
```

Also in this case the method uses *insertFieldEqualValueWithoutQuotationMarks* in order to read and insert the elements that are inside the vector.

**Other information:** The class also has some constants. These constants have been used in *constQueryInsert* in order to chose the right query.

Please pay attention that all these methods don't insert automatically the quotation marks, so for the value/field that have the necessity to have it, the programmers have to insert in the variables that he passes to the them (method).

### 4.3.4 Database_list_table_and_attribute.php

The class is a list of constants or in others words it contains all the tables' names with their attributes' names. In addition there are some methods that are able to concatenate some of these constants.

The constants have been divided in three parts:

1. In the first part there only are the tables' names:

```
1  const AListRatingFunctionalityAndVolumeWeights = "
       AListRatingFunctionalityAndVolumeWeights";
2
3  const BListRatingOverpacks = "BListRatingOverpacks";
4
5  const CListRatingCompositionSeparability = "
       CListRatingCompositionSeparability";
6  const Colors = "Colors";
7  const Companies = "Companies";
8  const Components = "Components";
9      [...]
```

2. In the second part there only are the attributes' names of the tables:

```
1  [...]
2  const IdImage = "IdImage";
3  const Image = "Image";
4  const ImageName = "ImageName";
5  const ImageSha = "ImageSha";
6  [...]
```

3. In the last part there only are the concatenation among tables' names, dot and attributes' names that the tables can have:

```
1  [...]
2  const ImagesIdImage = "Images.IdImage";
3  const ImagesImageName = "Images.ImageName";
4  const ImagesImageSha = "Images.ImageSha";
5  const ImagesImage = "Images.Image";
6  [...]
```

The class's methods are for example like this:

```
1  public function arrayDesigners($flagGetId)
2  {
3    $arrayX = [];
4    if ($flagGetId) array_push($arrayX, $this::IdDesigner,
         $this::Designer);
5    else
6      array_push($arrayX, $this::Designer);
7    return $arrayX;
8  }
```

It has been created this class in order to simplify the life of the programmers, because by the memory tool of programming is easy to remember and chose the right table's name or attribute's name. In other words, when the programmers are writing, the tool is able to suggest some possible right choice, when the person only has to choose the best.

### 4.3.5 Database_read_write.php

The class is an extension of *Database_connection*.

The goal is to have a easy way to use the other class. For instance: when the programmers wish to connect or to close the database he has to call some *Database_connection*'s method in order to do it, as he has to choose if the opening is roolback or not or if one day he wishes to launch new method during the closure.

So it's a bit tricky, to have in some points of the code some lines in order to make it. So this class is able to perform this job:

This is class's code:

```
1  class Database_read_write extends Database_connection
2  {
3    public function __construct()
4    {
5
6    }
7
8    public function connectToDbSetAutocommit($flagAutocommit)
9    {
10
11     if($this->dbCConnectToDatabase())
12     {
13       if($this->dbCSetAutoCommit($flagAutocommit));
14       else
15       return false;
16     }
17     else
18     return false;
19   }
20
21   public function closeConnectionRollbackOptional($flagRoolBack)
22   {
23     if($flagRoolBack)
24     {
25       try
26       {
27         if($this->dbCRoolBack())
28         {
29           $this->dbCCloseConnection();
30           return true;
31         }
32         else
33         return false;
34       }
35       catch (Exception $e)
```

```
36        {
37          return false;
38        }
39      }
40      else
41      {
42        if($this->dbCCommit())
43        if($this->dbCConnectToDatabase())
44        return true;
45        return false;
46      }
47
48    }
49
50    public function launchQuery($query, &$result)
51    {
52      if ($this->dbCQueryToDatabase($query, $result))
53      return true;
54      else
55      return false;
56    }
57
58  }
```

In order to (for example) close a connection, it is necessary to call one method (passing a flag) instead to launch ones of *Database_connection* class:

- before rollbback with *dbCRoolBack()*;
- after close the connection with *dbCCloseConnection()*.

## 4.4 Page-do-class folder

The folder contains PHP classes. Each front-end page has a particular set of PHP classes. These classes are able to manage/validate/send and/or to save or update information.

In other words every front-end page has 4 PHP classes that are able to do a specific job in relationship with the database:

- Get job: to take the information from the database and to send to front end using a JSON. The front-end can ask or a particular or a random page in relation to what it wish to do.
- Save job: to receive the information from the front-end and it convalidates them, for after (if there is not an error) to save in the database. In any case it returns a JSON to indicate if the operation has been completed without error or if there has been an error (the JSON indicates where there has been the error).
- Update job: the job is similar to the job saves but in this case it updates the database with new information.

- Delete job: the job consists to delete the information from the database.

In each operation the back-end before to convalidate the front-end input and after it will go to update the database and in every case it returns with a JSON in order to indicate the success or the failure. In the second case it also gives the point of error, if it can do it (for instance in the login it cannot say if the password or username is/are fault in order don't give relevant information to alleged attacker).

In order to better recognize the classes they have a concatenation names: *PageNumber_JOB.php*

Where Number is the number of the page of mock-up and the JOB can be:

- get to get the page;
- save to save the page;
- update to update the page;
- delete to delete the page.

Summerise: *Page1_get.php*, *Page2_delete.php* and so on.

## 4.5 Select-request folder

The folder only contains one class that is able to do a particular work: It has to reply to all front-end AJAX selector. In few words in front-end part there are several selectors that send in realtime the input of user and it wishes to show to user some choices that the user can chose.

In order to better understand see this: `https://select2.org/data-sources/ajax`.

The class receives to parameters:

- *$field* identifies which select the front-end is using;
- *$searchValue* is the field that the query has to search in the database.

The class starts with a switch/case constructor in order to know which selector the front-end is working and in this case the back-end understands how have to change the database.

The main part of the class is:

```
1  class Select_request
2  {
3
4    function getQueryFromDb($field, $searchValue)
5    {
6      $dblr = new Database_list_query_read();
7      $dbc = new Database_list_table_and_attribute();
8
9      switch ($field)
10     {
```

```
11        case(cp1packetName):
12        {
13          $query = $dblr->createQueryToGetValueAroundValue(
                $dbc::PacketName, $dbc::Packets, $dbc::PacketName,
                $searchValue, 6);
14          $this->launchQueryAndGetResultQuery($query, $dbc::PacketName);
15          break;
16        }
17        case(cp1company):
18        {
19          $query = $dblr->createQueryToGetValueAroundValue($dbc::Company,
                 $dbc::Companies, $dbc::Company, $searchValue, 6);
20          $this->launchQueryAndGetResultQuery($query, $dbc::Company);
21          break;
22        }
23      [...]
24        case(cp1productionYear):
25        {
26          $query = "";
27          $dblr->creaateQuery
28          (
29          [$dbc::ProductionYear],
30          [$dbc::ProductionYears],
31          [$dbc::ProductionYear],
32          $dbc::ProductionYear . " ASC ",
33          $query
34          );
35          $this->GetResultQuery($query, $dbc::ProductionYear,
                $searchValue );
36          break;
37        }
38        [...]
39        default:
40        {
41          echo "defualt";
42          return null;
43        }
44      }
45    }
46
47    public function launchQueryAndGetResultQuery($query, $fieldSearch)
48    {
49      $resultQuery= "";
50      $this->launchQuery($query,$resultQuery);
51
52      $i = 0;
53      $data = [];
54      $row = array();
55      $return_arr = array();
56      $row_array = array();
57      [...]
58      while ($row = $resultQuery->fetch_array())
59      {
60
```

```
61          $row_array['id'] = utf8_encode($row[$fieldSearch]);
62          $row_array['text'] = utf8_encode($row[$fieldSearch]);
63          array_push($return_arr, $row_array);
64          $i++;
65        }
66        echo json_encode($return_arr);
67
68        return true;
69
70      }
71    }
```

There are some important parts:

- Query: prototype/method to create the query is:

```
1  public function createQueryToGetValueAroundValue($select, $table,
       $fieldWhere, $searchValue, $limit)
2  {
3    $query = "SELECT ". $select . " FROM " . $table . " WHERE " .
         $fieldWhere  . " LIKE '%". $searchValue ."%' LIMIT " .
         $limit;
4    return $query;
5  }
```

  These parts can be find reading database class (*Database_list_query_read.php*).

- The class must be return a JOSN like this:

```
1  $json[] = ['id'=>"NamePacket1", 'text'=>"NamePacket1"];
```

  Where *NamePacket1* is for example the name of packet or the year of production. . . . In this way the front-end immediately can use the JSON. In order to create this JSON there is a particular code's portion in *launchQueryAndGetResultQuery* method:

```
1  while ($row = $resultQuery->fetch_array())
2  {
3
4    $row_array['id'] = utf8_encode($row[$fieldSearch]);
5    $row_array['text'] = utf8_encode($row[$fieldSearch]);
6    array_push($return_arr, $row_array);
7    $i++;
8  }
```

## 4.6 Switch-ajax-listening folder

The folder contains an unique file named *switch-ajax-listening.php* and it is the unique entry point of the back end.

Before to explain the PHP file, it is better to see the main part of the code (the code has been simplified from the real code):

```
1
2  <?php
3  require("./../require-file.php");
4  requireFile(1, "switch-ajax-listening");
5  requireFile(1, "return-json");
6
7  header('Content-Type: text/plain');
8
9  if (session_status() == PHP_SESSION_NONE)
10 {
11   session_start();
12 }
13
14 if (!isset($_POST[constActionId]))
15 {
16   statusErrorJsonSendReply("System error");
17   return;
18 }
19
20 // Don't forget the encoding
21 $actionId = utf8_encode($_POST[constActionId]);
22
23 //Page action
24 switch ($actionId)
25 {
26
27   case(deletePage):
28   {
29     if(isLogin()===0)
30     {
31       return;
32     }
33     if (!issetEmptyFieldX(constJsonField, "Error", true)) return;
34
35     $page1Delete = new Page1_delete();
36     $page1Delete->runClass(getJsonFromFrontEnd($_POST[constJsonField]))
           ;
37     break;
38   }
39
40 [...]
41   case(cp1SavePage1):
42   {
43     if(isLogin()===0)
44     {
45       return;
46     }
47   [...]
48     $userClass = $_SESSION["user"];
49     if(strcmp( $userClass->getIsUpdate(), "yes")===0)
50     {
51       if (!issetEmptyFieldX(constJsonField, "Error", true)) return;
52       $page1Save = new Page1_save();
```

```
53        $page1Save->databaseProcedure(getJsonFromFrontEnd($_POST[
              constJsonField]), 1);
54      }
55      else
56      {
57        if (!issetEmptyFieldX(constJsonField, "Error", true)) return;
58        $page1Save = new Page1_save();
59        $page1Save->runClass(getJsonFromFrontEnd($_POST[constJsonField]))
              ;
60      }
61      break;
62    }
63    case(cp1GetPage1NoSelect):
64    {
65      $page1Get = new Page1_get();
66      $page1Get->runClassNoSelect();
67      break;
68    }
69    case(cp1GetPage1Select):
70    {
71      if (!issetEmptyFieldX(constJsonField, "Error", true)) return;
72      $page1Get = new Page1_get();
73      $page1Get->runClassSelect($_POST[constJsonField]);
74      break;
75    }
76    case(constGetSelector):
77    {
78      if (!issetEmptyFieldX(constJsonField, "Error", true)) return;
79      if (!issetEmptyFieldX(constSearchTerm, "Error", true)) return;
80
81      $select_request = new Select_request();
82      $select_request->runClass(utf8_encode($_POST[constJsonField]),
            utf8_encode($_POST[constSearchTerm]));
83      break;
84    }
85    case(doUserAction):
86    {
87      $userClass="";
88      if (!issetEmptyFieldX(constJsonField, "Error", true)) return;
89      if(!isset($_SESSION["user"]))
90      {
91        $userClass = new User();
92      }
93      else
94      {
95        $userClass = $_SESSION["user"];
96      }
97      $userClass->runUser(getJsonFromFrontEnd($_POST[constJsonField]));
98      $_SESSION["user"] = $userClass;
99      return;
100     break;
101   }
102   [...]
103   default:
```

```
104     {
105
106     }
107   }
108
109   function isLogin()
110   {
111     if(!isset($_SESSION["user"]))
112     {
113       $userClass = new User();
114     }
115     else
116     {
117       $userClass = $_SESSION["user"];
118     }
119     $_SESSION["user"] = $userClass;
120     if(strcmp($userClass->getFlagLogin(), "si")===0)
121     return 1;
122     statusErrorJsonSendReply("Operazione non consentita perchÃ© non si Ã¨
          loggati");
123     return 0;
124   }
125
126   function issetEmptyFieldX($toTestField, $phrase, $flagPhrase)
127   {
128     if (isset($_POST[$toTestField])) return true;
129     if ($flagPhrase) statusErrorJsonSendReply($phrase);
130     return false;
131   }
132
133
134   function getJsonFromFrontEnd($postDeconde)
135   {
136     return (json_decode(utf8_encode($postDeconde), true));
137   }
```

In the Flowchart (fig: 4.1) there are the main steps. This file is able to require all the possible file using a special PHP file that has this specific job.

The file has two others particularities:

- Every-time that an operation (like save, update, ecc...) necessities that the user is logged, this check is effected by such file. Obviously this happens before creating the right class.

- Before making something the back-end must know in advance what the front-end wishes to do. In this case it is possibile to recycle the front-end's code in very well way. So there are some checks in back-end in order to understand if the client can do it, as the same (client) had before got the authorization for it.

In order to be able to manage as above, the file uses *User* class PHP, that contains certain specific fields.

In this way it is very easy to change a lot of things without destroing the original code. For instance, if an action can be effected without login, it is sufficient to change some code's rows.

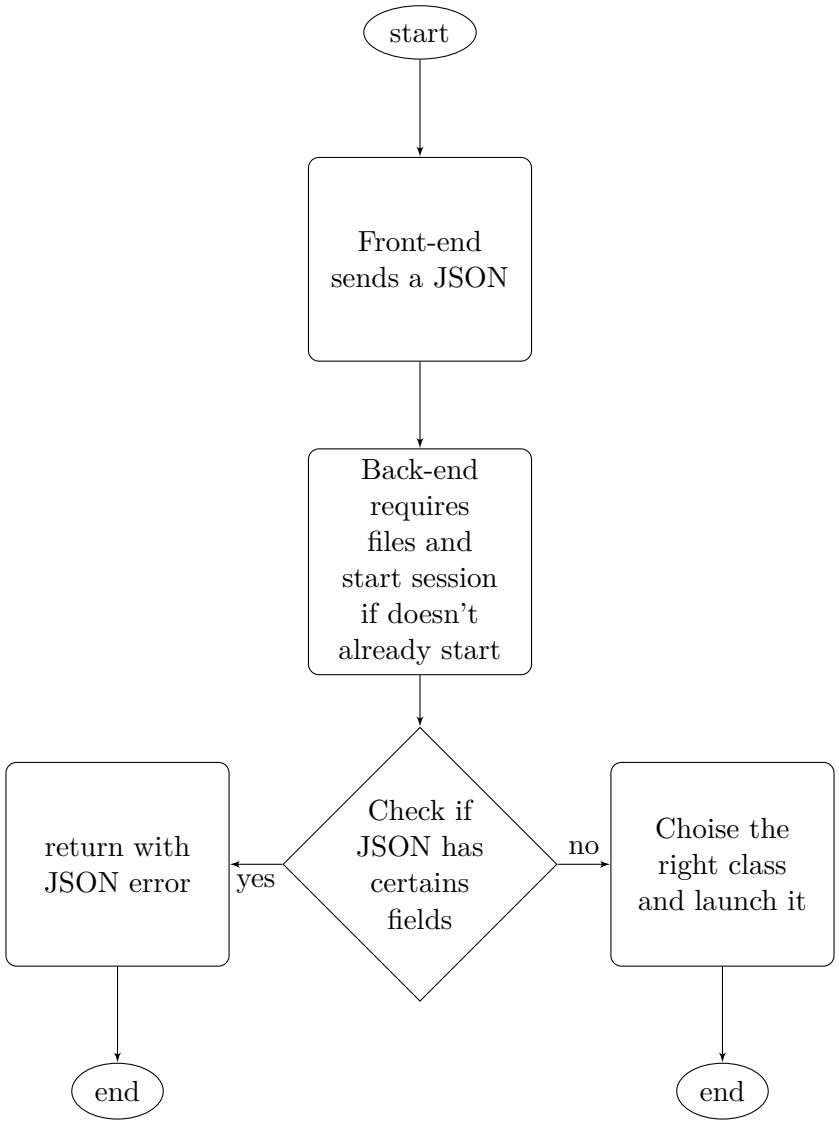


Figure 4.1 – How file works

## 4.7 Utility folder

The folder contains two class:

- User class
- Utility class

### 4.7.1 User class

The *User* calss is able to manage the registration, login and logout but also to memorize the next action that the client wishes to do. For login and registration it checks the input before to send the query to database.

In few words all the management of the user-session has done by it. Consequently the user class has been created with each new session and little by little the fields as user name or password will be populated when it will be necessary.

### 4.7.2 Utility class

The Utility class is a particular class that is able to concentrate all those methods that others classes wish to execute. Therefore, it is not a good idea to repeat the same code in several classes. So it was born the Utility class in order to cancel the repetition code.

More in details the class contains some methods in order to better convalidate all the JSON input during the save and update packet-cards.

## 4.8 Include-const-verb-ajax file

The file contains the constants that are used by JSON as field, like *database_list_table_and_attribute* class.

## 4.9 Include-files-database-and-generate-class file

The file contains all the constants of the JSON can have and the error or success code. In this way it is possible to do some actions:

- to change an action name or attribute name in all the code by refrector of the tool or editor (in few seconds);
- it is possible to add or to remove a name and to check if it is used or not used by refrector of the editor tool;
- it is easier for the programmers to remember/see/write as the editor is able to give a big help.

Code's example:

```
1  const cp1SavePage1 = "savePage1";
2  const cp1GetPage1NoSelect = "getPage1NoSelect";
3  const cp1GetPage1Select = "cp1GetPage1Select";
4  const cp1InsertOrUpdate = "insertOrUpdate";
5  const constGetSelector = "getSelector";
6  const cp1GetAll = "page1GetAll";
7  const doUserAction = "doUserAction";
8  const deletePage = "deletePage";
```

## 4.10 Require-file file

PHP programming like in C or others languages there is the problem to import or to require (in PHP) others classes or files. This is more or less a problem because including files or classes twice, could have some problems. Nowadays in PHP there is a very very good function called *spl_autoload_register* [8] [9]. In order to have a very powerful auto load there is this class that is able to use this function but also to import the files. Always for simplicity, the programmer has to include the new PHP class/file in this file and after to include a new row only in *switch-ajax-listening.php* file.

To better understand see the main part of the code:

```
1  spl_autoload_register(function ($class)
2  {
3    require_once patchClass($class) . $class . ".php";
4  }
5  );
6
7
8  function patchClass($class)
9  {
10   $firstLetter = substr($class, 0, 1);
11   if ($firstLetter === 'D')
12   {
13     return goBackPath(1) . "database-iteraction/";
14   }
15   if ($firstLetter === 'P')
16   {
17     return goBackPath(1) . "page-do-class/";
18   }
19   if ($firstLetter === 'U')
20   {
21     return goBackPath(1) . "utility/";
22   }
23   if ($firstLetter === 'S')
24   {
25     return goBackPath(1) . "select-request/";
26   }
27 }
28
29
30 function goBackPath($where)
31 {
32   $pathBack = "./";
33   for ($i = 0; $i < $where; $i++)
34   {
35     $pathBack = $pathBack . "../";
36   }
37   return $pathBack;
38 }
39
40
41 function requireFile($wheres, $name)
```

```
42  {
43    switch ($name)
44    {
45      case("switch-ajax-listening"):
46      {
47        require_once(goBackPath($wheres) . "include-const-verb-ajax" . ".
              php");
48        return;
49      }
50      case("return-json"):
51      {
52        require_once(goBackPath($wheres) . "return-json" . ".php");
53        return;
54
55      }
56    [...]
57    }
58
59  }
```

And how to use it (from *switch-ajax-listening* file):

```
1  [...]
2  require("./../require-file.php");
3  requireFile(1, "return-json");
4  [...]
```

The file *switch-ajax-listening* require a file (*require("./../require-file.php");*), inside it there are two entry points and two others functions.

The entry points are:

- *spl_autoload_register* that is used by the PHP engine in order to require the new classes. The function uses two others functions:
  - *patchClass* is called in order to know where is the class. This function is able to identify the right path by the name of the class, because each class has the first name's letter is egual to the first part of folder name that contains itself. So using a switch/case it is simply to identify the location;
  - *goBackPath* is used in the *patchClass* function in order to know how to go back into the relative path.
- *requireFile* is only used in *switch-ajax-listening* to import only PHP file once.

## 4.11 Return-json file

The file is a particular file that allow to code and to return a success JSON or an error JSON depending on the function that is called. Some functions wish to receive a number (a success code or error code) plus a message, others methods only wish to receive a message because the number of the error is general and stored in another file (*include-const-verb-ajax*).

Here there is the code:

```php
<?php
function statusErrorJsonSendReply($messageError)
{
  echo json_encode(array(status => statusError,'message'=>
      $messageError));
  return true;
}

function statusErrorJsonSendReplyWithCode($messageError, $code)
{
  echo json_encode(array(status => statusError,'message'=>
      $messageError, 'code' => $code));
  return true;
}


function statusSuccessJsonSendReply($messageError)
{
  echo json_encode(array(status => statusSuccess,'message'=>
      $messageError));
  return true;
}

function statusSuccessJsonSendReplyWithCode($messageError, $code)
{
  echo json_encode(array(status => statusSuccess,'message'=>
      $messageError, 'code' => $code));
  return true;
}
```

## 4.12 Other information

The PHP code is very very easy. It has been get this direction in order to help the future programmers and to help for debugging, building new items and to manage.

The names of the constants are the same of the content of them. It has been done this choice, for the same motivation: to help the programmers.

It is logic that one day it will be necessary to give more power and more efficiency, the constants could be changed in easier constants to help the computers in order to speed up during computation like switch/case construct.

# Chapter 5

# Front-end

The chapter wishes to describe the front-end and how it interact with the final user.

The front-end has been written using only client language, in this part there is not PHP or another server part and it communicates with the server using Json by AJAX.

In order to have a beautiful and a very powerful interface, the front-end employs several libraries that have been imported using npm tool.

Maybe it is not the best choice to use all these libraries, on the contrary using AngularJS or another big framework could be a better solution. Anyway, in order to give the possibility to the programmers to understand in an easy way this part, it has been written using html, JavaScript and css languages. Because to learn them (html, JavaScript and css) is simple on the contrary to understand and to learn very well a framework like Angular or React is not an easy work. In addition, following this way, there is another advantage: to be completely free and to change several things because there is not a core (like in Angular) but there is simply html, JavaScript and css. At the end by this solution there are not the big problems to update or to manage the client part like as, for instance, with AngularJS or some like that.

The unique downside is that the code is not so hight performate for the browser, but it has been noted that it is not a problem for nowadays.

## 5.1 Organition

The front-end has organized in the following way:

- *cards-insert-update-x* folder
- *cards-list-x* folder
- *cards-view-x* folder
- *footer.html*
- *header* folder
- *icon-image* folder

- *index.css*
- *index.html*
- *index.js*
- *library_modules* folder
- *node_modules* folder
- *package-lock.json*

Each view has three files' typologies: html, css, and JavaScript. For instance, there is the index and for consequence it has three typologies of files: index.html, index.css, index.js. In general there are one file html, one css and one or more js files. But when the programmers use the black boxes he have more than one htlm, css and js file. In this way it is more simply to read and maintain the code.

Every css and js file have the same name of html file, the unique difference is the extension. If there is more than one file js, the name is a concatenation of the html name plus a name that identify the file's job. Moreover the first or root file js is the file that only has the name like html, it is logic that the extension is not html but js.

There are two folder's name: if it finishes with *x* is not a library's folder otherwise it terminates with *x* and it's likely using a library (or more). The goals are to have an unique name for the folders and to reuse the code in others files. The folders' names try to describe the action(s) that the files (inside the folder itself) are able to do.

Another important issue is the *id* in html code. On the whole project all the files have different *id*. In particular the rule is: all the *id* starts with *id* (in order to better identify them with others elements), after *id* there is the initials letter of the folder's name (that indicate what the folder can do). Here there is a difference: if the file is in a library it has not a *x* otherwise if has outside of library, it contains an *x*. For instance: *idClxHeader* is a *id* and it stays in card ("C") list ("l") folder and it has not a library because there is "x". In addition the first letter after *id* always is capital letter and the *x* is lowercase and again the first letter after the *x* is capital letter.

The same aspect is for the name of JavaScript function, the unique difference is for *id* that is not used. All the personal-libraries have a *init* function that contains *init* in its name. When the programmers wish to use a library, he must call its *init* function. When the personal libraries are used it is necessary to do two tasks:

1. to call the *init* function or one of them *init* function;
2. import all necessary libraries. In order to do it, in every personal library there is a *readMe.txt* file where there is the list of all imports libraries in order to use it, maybe one day it's better to improve this item.

All these rules is used in order to import and to manage in better way without any conflict with the personal libraries or others files.

## 5.2 Library

The library has been installed using npm [10]. It has been used npm instead of Bower [11] because npm is more used than Bower [11].

The libraries used are (in installation order):

- jQuery v3.3.1
- Popper.js v1.14.3
- Bootstrap v4.1.1
- Sweet Alert2
- Font Awesome
- Select2
- Bootstrap4 fullscreen modal
- datatables with its extensions:
  - datatables.net-responsive-dt
  - datatables.net-buttons-dt
  - pdfmake
  - jszip

All the libraries are downloaded in *node_modules* folder by the follow commands:

```
1  npm install jquery@3.3.1 --save
2  npm install popper.js@^1.14.3 --save
3  npm install bootstrap@4.1.1 --save
4
5  npm install sweetalert2
6  npm install --save-dev @fortawesome/fontawesome-free
7  npm install --save bootstrap4-fullscreen-modal
8
9  npm install datatables.net
10 npm install datatables.net-dt
11 npm install --save datatables.net-responsive-dt
12 npm install  datatables.net-buttons-dt
13 npm install --save pdfmake
14 npm install --save jszip
```

This is the list where the programmers can find all the documentation:

- jQuery v3.3.1 → `https://jquery.com/`
- Popper.js v1.14.3 → `https://popper.js.org/`
- Bootstrap v4.1.1 → `https://getbootstrap.com/docs/4.0/getting-started/introduction/`

- Sweet Alert2 → `https://sweetalert2.github.io/`
- Font Awesome → `https://fortawesome.com/`
- Select2 → `https://select2.org/`
- Bootstrap4 fullscreen modal → `https://github.com/basilebong/bootstrap4-fullscreen-modal`
- datatables → `https://datatables.net/`
  - datatables.net-responsive-dt → `https://datatables.net/download/`
  - datatables.net-buttons-dt → `https://datatables.net/download/`
  - pdfmake → `https://datatables.net/download/`
  - jszip → `https://datatables.net/download/`

### 5.2.1 Libraries resume

The subsection wishes to give a very very hight view of the frameworks' functionalities in order to better understand for the future programmer that wishes to learn this work:

- jQuery → to have some different easy tricks, for instance to get the value from a button or to have Ajax.
- Popper.js → has been installed for bootstrap. In other words it is not used directly, but it used from other library (Bootstrap).
- Bootstrap → is used in order to have a layout for mobile and desktop; to have a grid where to put the elements. In addition there are several very nice components like modal or buttons that can give a good aspect to the view.
- Sweet Alert2 → is used in order to give an important message to the user after to have interact with the server. It is more or less a modal.
- Font Awesome → in order to have nice icon in pop-up or in others locations.
- Select2 → is used in order to search/select the terms during insert or update. It is able to download the terms close to the input that user has inserted. Look here to better understand the concept `https://select2.org/data-sources/ajax`.
- Bootstrap4 fullscreen modal → In normal Bootstrap there is not the possibility to have a modal fullscreen and sometimes is nicer to have it. So this library can do it.
- datatables with its extensions → it is used in order to have a nice, interact and efficient list. It is able to search in the list, to see some information and using some plug in to have some good tricks.

- datatables.net-responsive-dt → to have an interactive list with buttons (for instance).
- datatables.net-buttons-dt → to show and hide some information from the list.
- pdfmake → in order to export the list in pdf.
- jszip → in order to export the list in others format and to use for others application.

## 5.3 Cards-insert-update-x folder

The folder contains all the files that are able to update the cards.

The code is composed for the majority of import libraries like html body (and JS code). The goal is that the programmer can reuse the libraries in several way, in order to create beautiful page changing for example footer or header and adding new items.

The html of one page code is:

```
1  <!DOCTYPE html>
2  <html>
3  <meta name="viewport;" http-equiv="Content-Type"
4  content="text/html; charset=UTF-16; width=device-width,
       initial-scale=1"/>
5  <!-- <meta name="viewport" content="width=device-width,
       initial-scale=1"> -->
6
7
8  <head>
9  <!-- Import libraries module cards insert update and view cards -->
10 <!-- Jquery -->
11 <script src="./../node_modules/jquery/jquery.min.js"></script>
12
13 <!-- Poppoer -->
14 <script src="./../node_modules/popper.js/dist/umd/popper.min.js">
       </script>
15
16 <!-- Bootstrap -->
17 <script src="./../node_modules/bootstrap/dist/js/bootstrap.min.js">
       </script>
18
19
20 <link href="./../node_modules/bootstrap/dist/css/bootstrap.min.css"
       rel="stylesheet" media="screen">
21 <link href="./../node_modules/bootstrap/dist/css/
       bootstrap-grid.min.css" rel="stylesheet" media="screen">
22 <link href="./../node_modules/bootstrap/dist/css/
       bootstrap-reboot.min.css" rel="stylesheet" media="screen">
23
24 <!-- Bootstrap selector -->
25 <script src="./../node_modules/bootstrap-select/dist/js/
       bootstrap-select.js"></script>
26 <link rel="stylesheet" href="./../node_modules/bootstrap-select/dist/
       css/bootstrap-select.min.css">
```

```
27
28
29      <!-- Selector2 -->
30      <script src="./../node_modules/select2/dist/js/select2.full.min.js">
            </script>
31      <link rel="stylesheet" href="./../node_modules/select2/dist/css/
            select2.min.css">
32
33
34      <!-- Sweet alert 2 -->
35      <script src="./../node_modules/sweetalert2/dist/
            sweetalert2.all.min.js"></script>
36      <link rel="stylesheet" href="./../node_modules/sweetalert2/dist/
            sweetalert2.min.css">
37
38      <!-- font-awesome -->
39      <script src="./../node_modules/@fortawesome/fontawesome-free/js/
            all.min.js"></script>
40      <link rel="stylesheet" href="./../node_modules/@fortawesome/
            fontawesome-free/css/all.min.css">
41
42      <link rel="stylesheet" href="./../node_modules/
            bootstrap4-fullscreen-modal/dist/
            bootstrap4-modal-fullscreen.min.css">
43
44
45      <!--  Import custom libraries module insert update script file -->
46      <script src="./../library_modules/cards-insert-update/
            page1ab-insert-update.js"></script>
47      <script src="./../library_modules/cards-insert-update/
            page1ab-insert-update-decode-response-server.js"></script>
48      <script src="./../library_modules/cards-insert-update/
            page1ab-insert-update-designer-manage.js"></script>
49      <script src="./../library_modules/cards-insert-update/
            page1ab-insert-update-get-page.js"></script>
50      <script src="./../library_modules/cards-insert-update/
            page1ab-insert-update-global-const-and-variable.js"></script>
51      <script src="./../library_modules/cards-insert-update/
            page1ab-insert-update-import-files.js"></script>
52      <script src="./../library_modules/cards-insert-update/
            page1ab-insert-update-init-values.js"></script>
53      <script src="./../library_modules/cards-insert-update/
            page1ab-insert-update-measurement-packet.js"></script>
54      <script src="./../library_modules/cards-insert-update/
            page1ab-insert-update-modal-image.js"></script>
55      <script src="./../library_modules/cards-insert-update/
            page1ab-insert-update-popover.js"></script>
56      <script src="./../library_modules/cards-insert-update/
            page1ab-insert-update-reset-all.js"></script>
57      <script src="./../library_modules/cards-insert-update/
            page1ab-insert-update-resize.js"></script>
58      <script src="./../library_modules/cards-insert-update/
            page1ab-insert-update-selectors-ajax.js"></script>
```

```
59    <script src="./../library_modules/cards-insert-update/
          page1ab-insert-update-send-server.js"></script>
60    <script src="./../library_modules/cards-insert-update/
          page1ab-insert-update-show-hide-center-image.js"></script>
61    <script src="./../library_modules/cards-insert-update/
          page1ab-insert-update-server-const.js"></script>
62
63    <script src="./../library_modules/cards-insert-update/sweet-alert.js"
          ></script>
64
65
66    <!-- Custom style file -->
67    <link rel="stylesheet" href="./../library_modules/cards-insert-update
          /page1ab-insert-update.css">
68
69
70    <link rel="shortcut icon" href="../icon-image/IconEcopack.ico"/>
71
72
73
74    <!-- Import custom  libraries module view cards  -->
75    <link rel="stylesheet" href="../library_modules/cards-view/
          page1ab-view.css">
76    <script src="../library_modules/cards-view/page1ab-view.js"></script>
77    <script src="../library_modules/cards-view/
          page1ab-view-append-element.js"></script>
78    <script src="../library_modules/cards-view/
          page1ab-view-show-hide-center-image.js"></script>
79
80
81
82    <!-- Import custom library -->
83    <script src="./cards-insert-update.js"></script>
84
85    <script src="./../header/header.js"></script>
86
87
88    </head>
89
90    <body>
91    <!-- import header -->
92    <div id="idIuxHeader"></div>
93
94    <div id="idIuxCore"></div>
95
96
97    <div id="idIuxFooter"></div>
98
99    </body>
100
101
102   </html>
```

The JS code of one page is:

```
1    $(document).ready(function()
2    {
3      $("#idIuxHeader").load("/eco-pack-site/front-end/header.html");
4      $("#idIuxFooter").load("/eco-pack-site/front-end/footer.html");
5      $("#idIuxCore").load("/eco-pack-site/front-end/library_modules/
           cards-insert-update/page1ab-insert-update.php");
6
7      iuInitInsertUpdate();
8      hInitHeader();
9
10   } );
```

## 5.4 Cards-list-x folder

The folder contains the list of packagings, in such a way it is possible to do search (on the list) and export (the list).

The important thing is to understand and to see how to use table library, the others code's parts are easy to understand, so it becomes no more necessary to expose it.

Hereunder there is the list of import libraries:

```
1    <!-- data tables core -->
2  <script type="text/javascript" charset="utf8"
3  src="./../node_modules/datatables.net/js/jquery.dataTables.min.js">
       </script>
4
5
6  <!-- data tables responsive plugin -->
7  <script src="./../node_modules/datatables.net-responsive/js/
       dataTables.responsive.min.js"></script>
8
9  <script src="./../node_modules/datatables.net-responsive-dt/js/
       responsive.dataTables.min.js"></script>
10
11 <link rel="stylesheet" href="./../node_modules/
       datatables.net-responsive-dt/css/responsive.dataTables.min.css">
12
13
14 <script src="./../node_modules/datatables.net-buttons/js/
       buttons.colVis.js"></script>
15
16 <script src="./../node_modules/datatables.net-buttons/js/
       buttons.html5.min.js"></script>
17 <script src="./../node_modules/datatables.net-buttons/js/
       buttons.print.min.js"></script>
18
19 <script src="./../node_modules/datatables.net-buttons/js/
       dataTables.buttons.min.js"></script>
20 <link rel="stylesheet" href="./../node_modules/
       datatables.net-buttons-dt/css/buttons.dataTables.min.css">
```

The first important aspect, it is the table list that must be initialized using this function: (clxInitListWithButton). It creates the table and it asks to the server the data.

There are some ways that the server can send the JSON data in others words there are some JSON schema. In this case it has been used this way: a JSON that contains some objects and each object contains the information of one packaging.

```
1  function clxInitListWithButton()
2  {
3    table = $('#idClxListCard').DataTable({
4      "ajax": {
5        type: "POST",
6        url: "/eco-pack-site/back-end/switch-ajax-listening/
             switch-ajax-listening.php",
7        dataType: "json", //it is important be careful!!!
8        data:
9        {
10         actionId: "page1GetAll"
11       }
12
13     },
14     responsive: true,
15
16
17     "columns": [
18     { "data": "idSelectPacketName"},
19     { "data": "idSelectCompany" },
20     { "data": "idSelectDesigner1" },
21     { "data": "idSelectDesigner2" },
22     { "data": "idSelectDesigner3" },
23     { "data": "idSelectDesigner4" },
24     { "data": "idSelectDesigner5" },
25     { "data": "idSelectManufacturer" },
26     { "data": "idSelectorProductSector" },
27     { "data": "idSelectorProductionYear" },
28     { "data": "idSelectorNation" },
29     { "data": "idSelectorTypology" },
30     { "data": "idHeightInput"},
31     { "data": "idLengthInput" },
32     { "data": "idVolumeInput" },
33     { "data": "idWeightInput" },
34     { "data": "nameOutMouseOrImage1" },
35     { "data": "nameOverMouseOrImage2" },
36     {"data":null,"defaultContent":"" +
37       "<div class='container' style='width: 30rem'>" +
38       "<div class='row'>" +
39       "<div class='col-sm-3 pl-0'>" +
40       "<button type='button' class='btn btn-info btn-block'>Vista</
             button>"+
41       "</div>" +
42       "<div class='col-sm-3 pl-0'>" +
43       "<button type='button' class='btn btn-primary btn-block'>Apri</
             button>" +
44       "</div>" +
45       " <div class='col-sm-3  pl-0'>" +
46       "<button type='button' class='btn btn-warning btn-block'>
             Modifica</button>" +
```

```
47          "</div> " +
48          " <div class='col-sm-3  pl-0'>" +
49          "<button type='button' class='btn btn-danger btn-block'>Elimina</
                button>" +
50          "</div> " +
51          "</div> </div>" }
52        ]
53
54    });
```

In order to get the right relationship between JSON fields and the view, the framework uses these piece of code:

```
1     "columns": [
2   { "data": "idSelectPacketName"},
3   { "data": "idSelectCompany" },
4   { "data": "idSelectDesigner1" },
5   { "data": "idSelectDesigner2" },
6   { "data": "idSelectDesigner3" },
7   { "data": "idSelectDesigner4" },
8   { "data": "idSelectDesigner5" },
9   { "data": "idSelectManufacturer" },
10  { "data": "idSelectorProductSector" },
11  { "data": "idSelectorProductionYear" },
12  { "data": "idSelectorNation" },
13  { "data": "idSelectorTypology" },
14  { "data": "idHeightInput"},
15  { "data": "idLengthInput" },
16  { "data": "idVolumeInput" },
17  { "data": "idWeightInput" },
18  { "data": "nameOutMouseOrImage1" },
19  { "data": "nameOverMouseOrImage2" },
```

Over more for each row there are some buttons that are put in the code in this way:

```
1     {"data":null,"defaultContent":"" +
2   "<div class='container' style='width: 30rem'>" +
3   "<div class='row'>" +
4   "<div class='col-sm-3 pl-0'>" +
5   "<button type='button' class='btn btn-info btn-block'>Vista</button>"+
6   "</div>" +
7   "<div class='col-sm-3 pl-0'>" +
8   "<button type='button' class='btn btn-primary btn-block'>Apri</button>"
        +
9   "</div>" +
10  " <div class='col-sm-3  pl-0'>" +
11  "<button type='button' class='btn btn-warning btn-block'>Modifica</
        button>" +
12  "</div> " +
13  " <div class='col-sm-3  pl-0'>" +
14  "<button type='button' class='btn btn-danger btn-block'>Elimina</button
        >" +
15  "</div> " +
16  "</div> </div>" }
17  ]
```

```
18
19 });
```

In addition there is also the initialization of the buttons for exportation in pdf or in others forms. In order to do this, it is necessary to say/give to plug in which column has to export, for doing this there is this code:

```
1 columns: '0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17'
```

Each number represents a column of the table.

Here there is the main part of the code:

```
1
2   var buttons = new $.fn.dataTable.Buttons(table, {
3
4     "buttons": [
5
6     {
7       extend: 'colvis',
8       postfixButtons: [ 'colvisRestore' ],
9       container : '#colvis',
10      columns: '0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17'
11    },
12    'copyHtml5',
13    'excel',
14    'csvHtml5'
15    ],
16  }).container().appendTo($('#idClxShow2'));
```

In order to say to JavaScript which format the final user wishes to export, he have to write this code:

```
1 'copyHtml5',
2 'excel',
3 'csvHtml5'
```

every line correspond to a particular exportation: copy and paste, download in excel and download in csv.

Following there are some JQuery functions that are simply and there is not necessary to explain:

```
1
2   $('#idClxListCard tbody').on('click', '.btn-info', function (e)
3   {
4     var rowInformation = table.row( $(this)).data();
5     var packetName = rowInformation["idSelectPacketName"];
6     $('#idClxViewCardModal').modal("show");
7     $("#idClxViewModalCore").load("/eco-pack-site/front-end/
          library_modules/cards-view/page1ab-view.php");
8     viewInitViewGetPacketFromName(packetName);
9   } );
10
11  $('#idClxListCard tbody').on('click', '.btn-primary', function (e)
12  {
13    var rowInformation = table.row( $(this)).data();
```

```
14      var packetName = rowInformation["idSelectPacketName"];
15      clxSetNamePageAndNextPage(packetName, "no" ,"/eco-pack-site/
            front-end/cards-view-x/cards-view.html");
16    } );
17    var tablex = "";
18    $('#idClxListCard tbody').on( 'click', '.btn-danger', function () {
19      var rowInformation = table.row( $(this)).data();
20      tablex= table.row( $(this));
21
22      var packetName = rowInformation["idSelectPacketName"];
23      var jsonPacket = {};
24
25      jsonPacket["PacketName"] = packetName;
26
27      $.ajax({
28        type: "POST",
29        url: backPath + "/back-end/switch-ajax-listening/
              switch-ajax-listening.php",
30        dataType: "text",
31        data:
32        {
33          jsonField:  JSON.stringify(jsonPacket),
34          actionId: "deletePage"
35        },
36
37        success: function (data)
38        {
39          if(2===tiuAlertXLaunch(data))
40          {
41            tablex.remove().draw(true);
42          }
43
44          return true;
45        },
46        error: function (respond)
47        {
48          return false;
49        }
50      });
51
52    });
53
54    $('#idClxListCard tbody').on( 'click', '.btn-warning', function ()
55    {
56      var rowInformation = table.row( $(this)).data();
57      var packetName = rowInformation["idSelectPacketName"];
58      clxSetNamePageAndNextPage(packetName, "yes" ,"/eco-pack-site/
            front-end/cards-insert-update-x/cards-insert-update.html");
59    });
60
61    $('#idClxNewCard').click(function() {
62      clxNewCardButton();
63    });
64  }
```

## 5.5 Cards-view-x folder

The folder contains the view cards (for seeing a card) with header and footer. In order do to write few lines of code it has been used the personal libraries.

## 5.6 Footer.html file

The footer file is a simple html file that contains the footer code with some links.

## 5.7 Header folder

The folder contains the header files (htmj, JS and css).

## 5.8 Icon-image folder

The folder contains all the images/icons that the web-site uses.

## 5.9 Index.css file

The file contains the css code for the index, nothing else.

## 5.10 Index.html file

The file contains the index html code. It is interesting to read the code because there are few lines of codes because it only uses libraries:

```
1  </head>
2
3  <body>
4  <!-- import header -->
5  <div id="idIxHeader"></div>
6
7  <div id="idIxViewCard"></div>
8
9  <div id="idIxFooter"></div>
10
11 </body>
12 </html>
```

The in the div tag will be put using JavaScript file the code, more in details:

- *idIxHeader* is for the header taking from *header* folder;
- *idIxViewCard* is for view taking from *library_modules* folder;
- *idIxFooter* is for footer taking from *footer.html* file.

## 5.11 Index.js file

The file contains the JavaScript of index. The JavaScript file load the right components and it calls the initialization functions of the items like these:

```
1  $(document).ready(function (e)
2  {
3    $("#idIHeader").load("header.html");
4    $("#idIViewCard").load("./library_modules/cards-view/page1ab-view.php
         ");
5    $("#idIFooter").load("footer.html");
6
7    hInitHeader();
8    viewInitView();
9
10 });
```

## 5.12 Library_modules folder

The folder contains all the personal libraries or in others words all the views and update/insert packaging.

In order don't have too much files that repeat the code, it has been concentrated the code. In others terms, it is not good to have two groups of files with the same code for insertion of packaging and update the data that the unique difference is the initialization function(s). So the better solution is to merge the two groups in one and after to call the right initialization function for the right work. In addition in some parts of the code there are two branch with a flag in order to chose the right jobs (but it is not a difficult thing to understand). In this sense for every mock-up page there are two group: a view that is able to see the packet and another that is able to insert or update (the packet).

There are not relevant parts to describe, it is sufficient to read the code, anyway it is better to be careful with Select2 library because it has some important tricks that is better to see.

Before it is better to read the code in order to understand the future parts:

```
1  [...]
2  $("#idIuSelectPacketName").select2({
3    cache: true,
4    tags: true,
5    placeholder: "Inserire il nome del packet (obbligatorio)",
6    tokenSeparators: [','],
7    ajax: {
8      url: './../../back-end/switch-ajax-listening/
           switch-ajax-listening.php',
9      type: "post",
10     dataType: 'json',
11     delay: 250,
12     data: function (params)
13     {
14       return {
```

```
15          searchTerm: params.term, // search term
16          actionId: "getSelector",
17          jsonField: "idSelectPacketName"
18        };
19      },
20      processResults: function (response)
21      {
22        return {
23          results: response,
24          id: response.term,
25          text: response.term + " (new)",
26          newOption: true
27        };
28      },
29    },
30    createTag: function (params)
31    {
32      var term = $.trim(params.term);
33
34      if (term === '')
35      {
36        return null;
37      }
38      //console.log(term.length);
39      if (term.length > 50)
40      {
41        iuAlertError("The name packet must be less than 50 char");
42        return null;
43      }
44
45      return {
46        id: term,
47        text: term + " (new)",  // add text....
48        newOption: true
49      };
50    },
51  }).on('change select2:close', function (e)
52  {
53    if(pageUpdate===true)
54    {
55      $(this).next('.select2-container').find('.
          select2-selection__rendered').text(function (idx, txt)
56      {
57        if (iuInputNamePacket.localeCompare(txt) !== 0)
58        {
59          $(this).addClass('iuBlueColor');
60          return txt;
61        }
62        else
63        {
64          $(this).removeClass('iuBlueColor');
65        }
66      })
67    }
```

```
68 });
69 [...]
```

This piece of code is very important because is the core of Select2. When the page is initialized this code must be launched. Every time that the user digits some things in the select field, it is executed in order to ask the server if there are some words close to this that the user has inserted so the select can display them (the worlds):

```
1  [...]
2   cache: true,
3  tags: true,
4  placeholder: "Inserire il nome del packet (obbligatorio)",
5  tokenSeparators: [','],
6  ajax: {
7    url: './../../back-end/switch-ajax-listening/
         switch-ajax-listening.php',
8    type: "post",
9    dataType: 'json',
10   delay: 250,
11   data: function (params)
12   {
13     return {
14       searchTerm: params.term, // search term
15       actionId: "getSelector",
16       jsonField: "idSelectPacketName"
17     };
18   },
19   processResults: function (response)
20   {
21     return {
22       results: response,
23       id: response.term,
24       text: response.term + " (new)",
25       newOption: true
26     };
27   },
28 },
29 [...]
```

In order to say the user that word that has inserted is new; The code is:

```
1  [...]
2  processResults: function (response)
3  {
4    return {
5      results: response,
6    id: response.term,
7    text: response.term + " (new)",
8    newOption: true
9    };
10 },
```

In order to check the user has only inserted right char or only number; The code is:

```
1
```

```
2  createTag: function (params)
3  {
4    var term = $.trim(params.term);
5
6    if (term === '')
7    {
8      return null;
9    }
10   //console.log(term.length);
11   if (term.length > 50)
12   {
13     iuAlertError("The name packet must be less than 50 char");
14     return null;
15   }
16
17   return {
18     id: term,
19     text: term + " (new)",  // add text....
20     newOption: true
21   };
22 },
```

If the programmer wishes to change the color during the update packaging, when the user inserts a new word in order to replace the last one; This is code:

```
1  .on('change select2:close', function (e)
2  {
3    if(pageUpdate===true)
4    {
5      $(this).next('.select2-container').find('.
           select2-selection__rendered').text(function (idx, txt)
6      {
7        if (iuInputNamePacket.localeCompare(txt) !== 0)
8        {
9          $(this).addClass('iuBlueColor');
10         return txt;
11       }
12       else
13       {
14         $(this).removeClass('iuBlueColor');
15       }
16     })
17   }
18 });
```

The last important tricky is that every time that a select adds in the DOM (show/hide/append doesn't matter) it is necessary to perform two actions:

- to call this: *$('select').select2();*
- to initialize again all the select like when the page has been opened.

## 5.13 Node_modules folder

The folder contains all the libraries that have been installed by npm.

## 5.14 Package-lock.json

The file has been generate by npm tool[13].

# Bibliography


[1] KISS principle states https://en.wikipedia.org/wiki/KISS_principle

[2] Char Vs VarChar https://stackoverflow.com/questions/1885630/whats-the-difference-between-varchar-and-char Char Vs VarChar Vs Text https://stackoverflow.com/questions/2083791/mysql-text-vs-char-and-varchar https://stackoverflow.com/questions/20958/list-of-standard-lengths-for-database-fields

[3] Join char Vs Integer https://stackoverflow.com/questions/18218687/will-joining-on-integer-be-quicker-than-joining-on-nvarchar

[4] Slim https://www.slimframework.com/

[5] Laravel https://laravel.com/

[6] CakePHP https://cakephp.org/

[7] Phalcon https://phalconphp.com/it/

[8] spl_autoload_register PHP guide http://php.net/manual/en/function.spl-autoload-register.php

[9] spl_autoload_register PHP to better understand https://stackoverflow.com/questions/7651509/what-is-autoloading-how-do-you-use-spl-autoload-autoload-and-spl-autoload-re

[10] NPM https://www.npmjs.com/

[11] Bower https://bower.io/

[12] Difference between NPM and Bower https://stackoverflow.com/questions/18641899/what-is-the-difference-between-bower-and-npm and https://stackoverflow.com/questions/34092060/bower-vs-npm-which-is-better-for-installing-angularjs

[13] NPM tool JSON file https://docs.npmjs.com/files/package-locks

[14] XML vs JSON https://stackoverflow.com/questions/3287060/json-or-xml-or-other-data-format-with-jquery-ajax and https://www.w3schools.com/js/js_json_xml.asp