DOCUMENTATION

ORDER MANAGEMENT

3RD ASSIGNMENT

STUDENT NAME: IAZ ANIA MEDEEA

GROUP:30424

Content

1.Theme objective

Main objective: design and implement an application for managing the client orders for a shop. Consider an application Orders Management for processing client orders for a shop. Relational databases should be used to store the products, the clients, and the orders. The application should be designed according to the layered architecture pattern and should use (minimally) the following classes:

- Model classes - represent the data models of the application
- Business Logic classes - contain the application logic
- Presentation classes – GUI related classes
- Data access classes - classes that contain the access to the database

Some sub-objectives are: analyzing the problem and identifying the requirements, designing the orders management application, implementing the orders management application and testing the orders management application.

2.Problem analysis, modeling, scenarios, use cases

2.1 Problem analysis

In this problem the user can introduce products, clients and orders as well, with the help of a database to which the project is connected. In the database I have 3 tables: product, client and orders. Because a client can buy many products and a product can be bought by many clients as long as the product it is not out of stock, I needed a table to link the product and clients. This table is the orders table itself where I introduce the id of the product, the id of the client and the quantity required by the client.

In order to add, remove, create and update different fields, some simple operations are implemented by communicating with the database. In my case I have chosen postgres because I have worked with it the most. It was easy to make connections between the tables and add constrains, for example we cannot have clients that are younger than eighteen.

Another thing to consider it that the user may enter incorrect data intro the table. The application should stop the user from doing that and signal what went wrong or which field should be modified in order for the application to work properly and fulfill the required operation.

2.2 Modeling

Since the application is not very complex the data is pretty simple to represent having few entities with either integer or String types of data. I have three main tables: product, client and orders. The table orders represent a many to many relationship between the products and the clients. This relationship is compulsory, since we want to represent things as they happen in the real life.

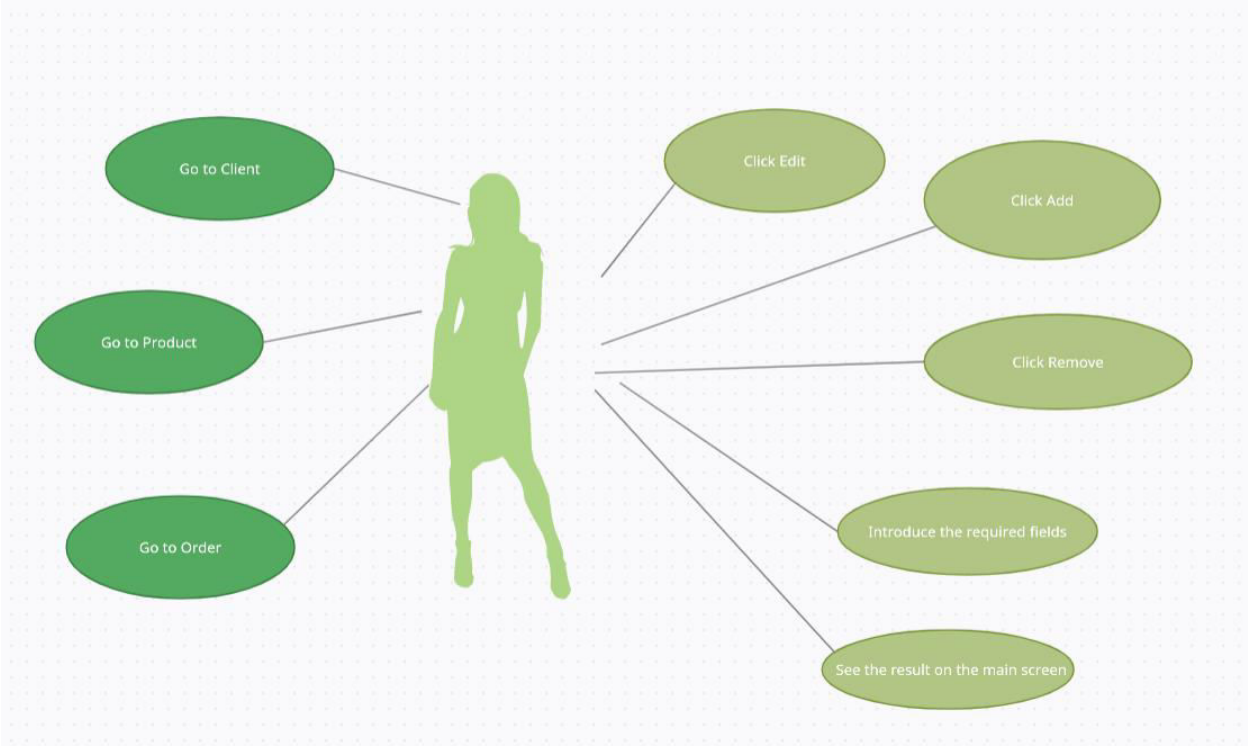
2.3 Scenarios

Because it is not sure how the user will interact with the graphical interface, I needed to make sure that he or she cannot introduce incorrect data or leave fields uncompleted. As long as the input it is not correct, the application will not do anything with the data, but some messages will be displayed. The message will indicate the error that was made so that the user can easily identify it. For example if the user does not complete all the fields a message saying to insert all the fields then press on the ‘Add’ button.

After that the only thing that I need to do is to illustrate the result on the graphical user interface after every operation performed.

2.4 Use Cases

The use case diagram can be a pleasant way to visualize the interaction between the user and the system that he or she may use. It also shows what the user needs to do in order to get the right results, giving a general description of the system.



As it can be seen in the picture above the actions will take place in the following order:

- The user can choose to which screen she or he wants to go by pressing a button
- After that we have three operations that could be performed:
  - For the Client: we can add, edit or remove
  - For the Product: we can add, edit or remove
  - For the Order: we can add
- Complete the required fields if necessary
- See the results of the operation performed after the data is introduced correctly

3. Design

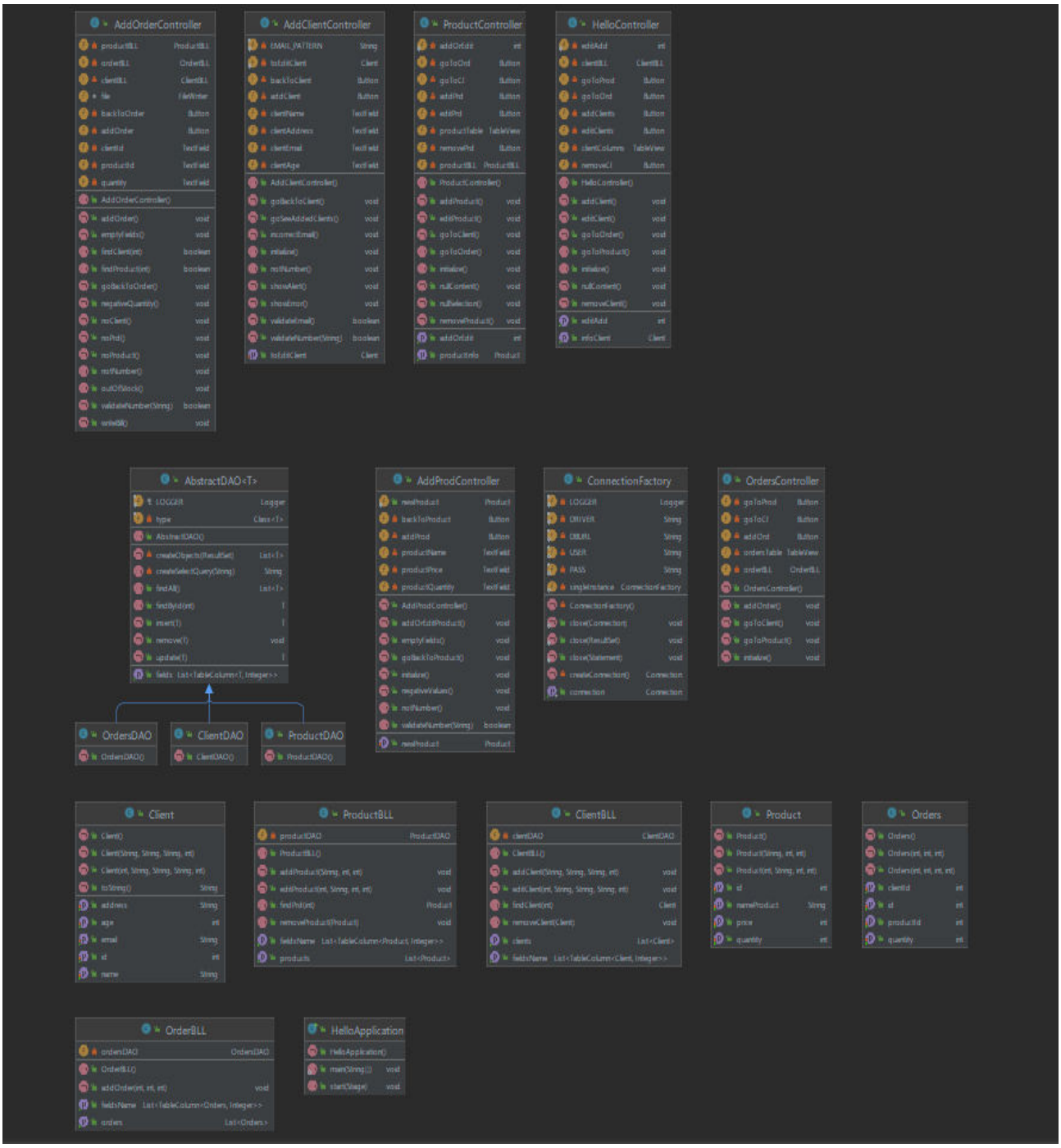
For the design I chose to use the MVC pattern for a better modularization of the project. This design specifies that an application consists of data model, presentation information, and control information. The pattern requires that each of these be separated into different objects.

For the model part of the pattern I have encapsulated the classes Product, Client and Orders that contain information about each of them, id, name, quantity and others. My controller consists of more than one class that acts like a controller. Because javafx uses .fxml files for each screen I needed a controller that corresponds to each one of them. Another thing to consider is that in this classes I use validators for the data that the user introduces. If a problem occurs then a message will be displayed telling the user what went wrong and what he or she should do. Form example if the email is not provided in the well know form and something gibberish is introduced the user will be notified that the email should be reintroduced.

The most important class is AbstractDAO found in the package called DataAccess. In this class I have implemented the operations performed on the database, for example insert, update, find by id and others. This class uses T type as parameters so that the other classes, ProductDAO, ClientDAO and OrdersDAO will only need to extend this class and its methods. The purpose of this is to use reflection and have as little code written as possible. Each query will be generated through reflection using the fields from the classes of the Model package. For this every class encapsulated in the package Model has an empty constructor and getters and setters for each field. Another thing to consider is that the name of the attributes in this classes has to be exactly the same as the one from the database created in order for the application to work.

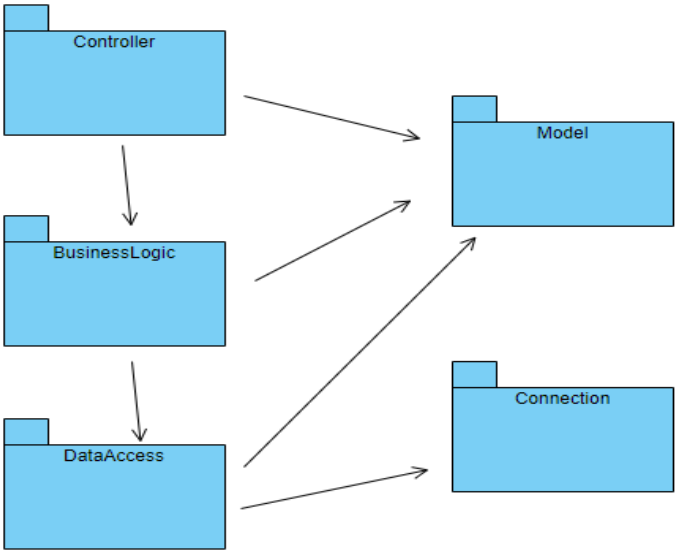
In the package BusinessLogic we can observe three classes: ClientBLL, ProductBLL and OrdersBLL. The purpose of these classes is to link the controller, as well as the view, to the AbstractDAO class that performs the operations and vice versa.

The UML diagram can be seen below:



It cand be easily seen that the view is missing due tot the fact that it is a .fxml file and not a java class. Also the class that contains the main is missing, its only purpose being to launch the application and set the title, width and length of the screen that will be displayed.

The package diagram will be presented below:



- **Data Structures**

For this application I used as data structure Lists, more specific ArrayLists. Some advantages of this data structure are: they are resizable, elements can be inserted and deleted at or from a specified indexes, they have many methods to manipulate the objects stored and they can store any type of object that we define. Another reason for which I have chosen them is that they are very compatible with the database and tableView from the view that helps to display every product, client or order that is present in the tables.

4. Implementation

○ Model Classes

As mentioned before, I have three classes in this package: Product, Client and Orders, the latest linking the first two. In the Client class I have five private attributes: id, name, address, email and age. As constrains I have added from the database that the email should be unique, the client should be older than 17 and all the fields must not be null. Also, the id is a primary key, unique and not null. For the reflection purpose I have one empty constructor and other two used to initialize different attributes. As methods I have only getters and setters for all five private fields.

In the class Product I have four private fields: id, nameProduct, price and quantity. The price and quantity are of integer type. As constrains for these fields I have set the nameProduct to be unique, the price to be greater than zero and the quantity to be greater or equal to zero. Also, all the fields have to be not null. As in the case of the Client class I have one empty constructor and other two. The methods are represented by the getters and setters of the private fields.

In the class Orders I have four fields: id, clientId, productId and quantity. All attributes are positive integer. The id refers to the id of the order while the clientId and productId that must correspond to one of the id present in the tableView of the Client and Product, otherwise an error message will be displayed.

○ Connection Class

This package contains only one class, ConnectionFactory, that has the purpose to link the application to the database created. It does this with the help of five final strings: LOGGER, DRIVER, DBURL, USER and PASS.

```
try {
    connection = DriverManager.getConnection(DBURL, USER, PASS);
} catch (SQLException e) {
    LOGGER.log(Level.WARNING, "An error occured while trying to connect to
the database");
    e.printStackTrace();
}
```

- ## DataAccess Classes

As presented before, this package contains four classes. The main class, AbstractDAO, contains methods for adding, removing, updating and others, in connection with the database, using as parameters data of type T. The other three classes will only extend this one, specifying the object that should replace the T.

- ## BusinessLogic Classes

The three classes mentioned before from this package have as purpose to establish a connection between the controllers and the DataAcces classes. In each class I have a private attribute of type DAO through which I access the methods needed.

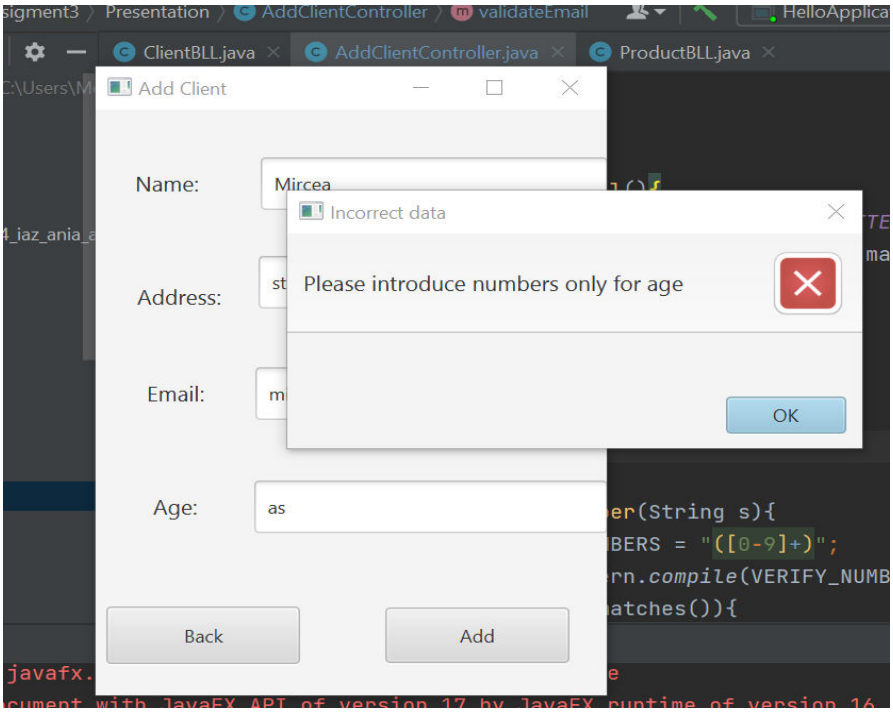
```
private ClientDAO clientDAO;

public ClientBLL(){
    clientDAO = new ClientDAO();
}
```

- ## Presentation Classes

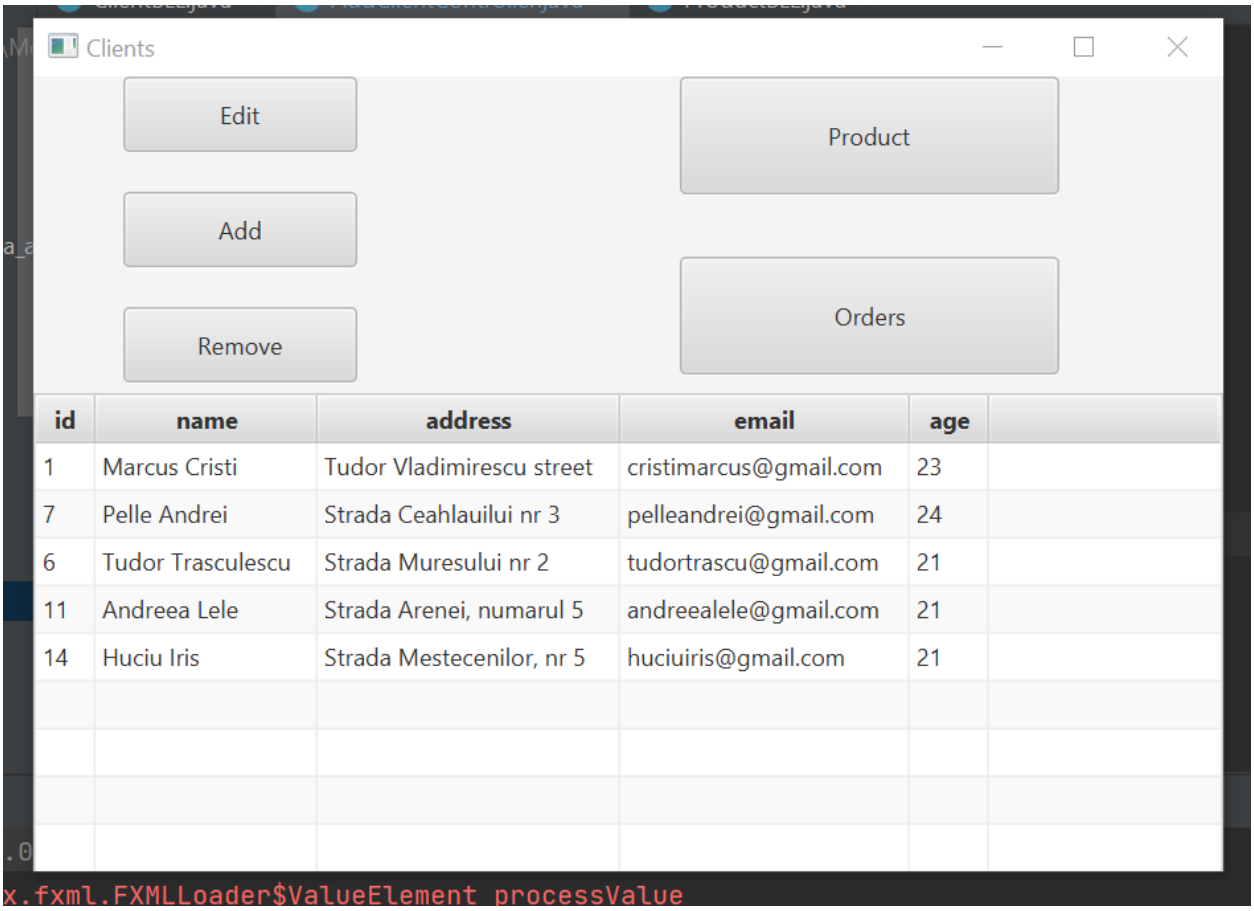
This package contains all the controllers for the view. I created the classes here because these are the only classes that can access the view and modify or display different textFields. Also, in these classes I have methods that validate the data. The methods return a boolean value that indicates if the data introduced by user is valid or not. For the data validation I have used regex.

```
public boolean validateNumber(String s){
    final String VERIFY_NUMBERS = "([0-9]+)";
    Pattern pattern = Pattern.compile(VERIFY_NUMBERS);
    if(pattern.matcher(s).matches()){
        return true;
    }
    return false;
}
```



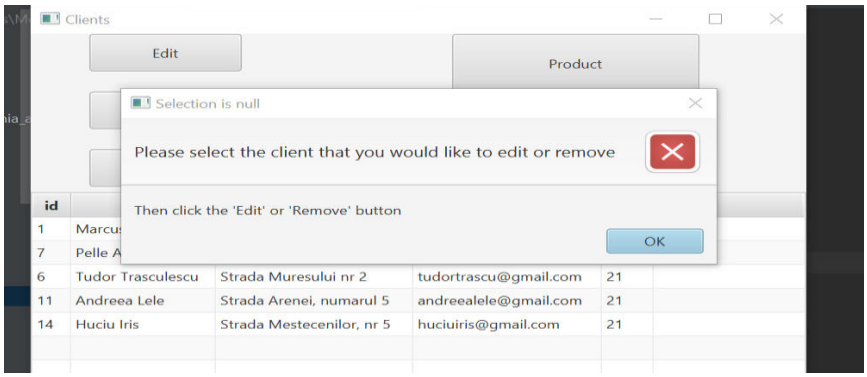
- **Graphical user interface**

For the graphical user interface I have six screens that will be displayed, depending on the button that is pressed. The main screen that is displayed when the application starts is called Clients. Here we can easily observe the clients that already exist in the database, as well as five buttons: Add, Remove, Edit, Orders and Product. The first three buttons refers to the clients while the other two will display a different screen, similar with the main one.



5. Results

The application has been implemented in Java Programming language in the IntelliJ IDE. The database part was implemented using DataGrip IDE with Postgres. For the testing part I intentionally introduced wrong data in the textFields from the view to verify if the application displays the appropriate error messages for each mistake in order to be easier for the user to see where the problem occurred.



Above you can see what happens when the user wants to remove a row without selection anything.

6. Conclusions

From this application I have learnt how to work with database and become more comfortable with the queries. In addition, it is the first time when I have worked with reflection, which is commonly used, and generic parameters. These parameters come with a lot of advantages, such like: better performance, less code written, type safety and streamline dynamically generated code, meaning that when you use generics with dynamically generated code you do not need to generate the type.

A limitation of the application is that the user has to know the id of the client and product. This could be a very useful further development: to be able to add the client by his or her name, as well as the product. Another further development could be to have less views for the operations.

7. Bibliography

- https://regexr.com/
- https://stackoverflow.com/
- https://app.creately.com/d/pfulq18TWJw/edit
- https://online.visual-paradigm.com/diagrams/features/package-diagram-software/
- https://gitlab.com/utcn_dsrl/pt-reflection-example