Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

Semester 2021-2021

Fundamental Programming Techniques

- Laboratory Assignment no. 3 -

Orders Management

Student: Maria Gliga-Hambeț

Group: 30423

Supervising teacher: Cristina Pop

# TABLE OF CONTENTS

[1. Objectives 3](#_Toc97399318)

[2. Problem analysis, modelling, use cases 4](#_Toc97399319)

[3. Design 5](#_Toc97399320)

[4. Implementation 8](#_Toc97399321)

[5. Results 15](#_Toc97399322)

[6. Conclusions 17](#_Toc97399323)

[7. Bibliography 17](#_Toc97399324)

1. **Objectives**

The main purpose of this project is to design and later on implement an order management system for a warehouse, with a user-friendly graphical interface, which could be easily used in order to simulate real life scenarios. The user can do a various number of actions, relating the clients and products of the warehouse, as well as the orders that can be placed. What the project manages to achieve is work with a database in which all of the information is kept.

In order to achieve the desired result, the whole project can be divided into the following, smaller tasks:

* Analyze the problem and identify requirements
* Design the orders management application
* Implement the orders management application
* Test the orders management application

1. **Problem analysis, designing, scenarios, use cases**
   1. **Problem analysis**

The orders management application simulates a scenario that is met in various fields in everyday life, the management of a warehouse, for example, or any sort of domain that requires the placement of orders, done by individuals. What should be achieved in the end are the options to add, update and delete clients and products. At the same time, the user should be able to see the data concerning the clients and products and be able to place an order. The order is processed when selecting a client, a product and the desired quantity.

In order to make a closer to life experience, we must take into account the following facts:

* A check must be present to see whether or not there exists the desired amount of a product. In the latter case, the client must be informed of the situation and not be allowed to place the order.
* Certain inserted fields must be unique in the whole database, beside the id, which is automatically incremented, and therefore certainly unique. For the clients, it is the case of the email, whilst for the products, it is the name.

Because of the fact that the project establishes a connection with a database in order to store and manipulate information which is similar among all of our three tables (Client, Product, Orders), we will make use of reflection techniques when in comes to the creation and running of statements. This way, using abstract types and implementing a general solution, the project can further on expand without it being a problem as to rewrite specific code for each new table.

* 1. **Designing**

For bringing a close to life experience to our simulation, the application uses in the background a database. So, what is important to always keep in mind is to constantly update what the user is seeing as well. As stated before, a more general approach on the data manipulation code is desirable as in assures easier future developments. Another thing which could be considered is for reflection to be used also in the GUI, as, for example, the client and product windows are almost identical.

* 1. **Scenarios and use cases**

When the simulation is started, the user chooses what part of the application he wants to interfere with: the management of clients, products or orders. At any point, he can go back to the main page and choose other actions as well. From than on, depending on the initial selection, the user can do various operations on the data. In the case of clients and products, he can add, update or delete, whilst for orders, he is shown the current existing ones and is able to place another. Therefore, the whole application allows the user to do plenty of actions, all of which are presented below, in the use cases:

**Use case description**

Since the product and client windows and operations that can be performed are more or less the same (the only difference being the fields which need to be completed), I will further on only present the cases for the client part, but the content and flow goes the same way for the product cases as well.

**Title:** Add client

**Primary actor:** The user

**Main success steps:**

1. The user is presented with the main, starting window and presses on the “Manage clients” button

2. The user can see a table containing the current clients of the database

3. The user presses on the “Add Client” button

4. A new window is opened where the user inserts the wanted data

5. The user clicks once again on the “Add Client” button and is brought back to the main clients page, where he can now see the recently added client as part of the table.

**Alternative sequences:**

* From step 3: the user decides that he wanted to do an action not related to the clients, so he pushed the home button and is brought back to the main, starting page
* From step 5: the user decides that he no longer wants to add a new client, so he presses on the back button, which is represented by a back arrow icon
* From step 5: the user inserts the data incorrectly or incompletely so the addition will not be completed and the window remains the current one

**Title:** Delete client

**Primary actor:** The user

**Main success steps:**

1. The user is presented with the main, starting window and presses on the “Manage clients” button

2. The user can see a table containing the current clients of the database

3. The user selects one of the rows from the table by clicking it

4. The user pushed the “Delete Client” button

5. The user will now see the updated version of the table, not containing the previously selected client

What also happens in the back of the application is that if the deleted client was also part of an order, that particular order will be deleted as well. Otherwise, there would be errors in the database because of the way the tables are linked.

**Alternative sequences:**

* From step 3: the user decides that he wanted to do an action not related to the clients, so he pushed the home button and is brought back to the main, starting page
* From step 3: the user clicks on the “Delete Client” button without previously selecting a client. Nothing happens in that case and the data is still unchanged

**Title:** Edit client

**Primary actor:** The user

**Main success steps:**

1. The user is presented with the main, starting window and presses on the “Manage clients” button

2. The user can see a table containing the current clients of the database

3. The user selects one of the rows from the table by clicking it

4. The user pushed the “Edit Client” button

5. The user will be shown a new window, in which he has to input the field of the client (with the exception of the id, which cannot be changed)

6. The user pushes the “Edit Client” button and is brought back to the main clients window and can now see that the user previously selected is updated (it can be recognized by means of the id).

**Alternative sequences:**

* From step 3: the user clicks on the “Edit Client” button without previously selecting a client. Nothing happens in that case and the data is still unchanged
* From step 3: the user decides that he wanted to do an action not related to the clients, so he pushed the home button and is brought back to the main, starting page
* From step 6: the user inserts the data incorrectly or incompletely so the addition will not be completed and the window remains the current one

**Title:** View Orders

**Primary actor:** The user

**Main success steps:**

1. The user is presented with the main, starting window and presses on the “Manage orders” button

2. The user can see a table containing the current orders of the database

3. The user from there on can press on the home button and the brought back to the main starting page

**Title:** Add order

**Primary actor:** The user

**Main success steps:**

1. The user is presented with the main, starting window and presses on the “Manage orders” button

2. The user can see a table containing the current clients of the database

3. The user clicks on the “Add Order” button

4. The user is shown a window with a field to be completed and 2 tables: one with the clients and the other one with the products

5. The user presses one row on each table, action through which he selects the client and product

6. The user inserts the desired quantity of the product

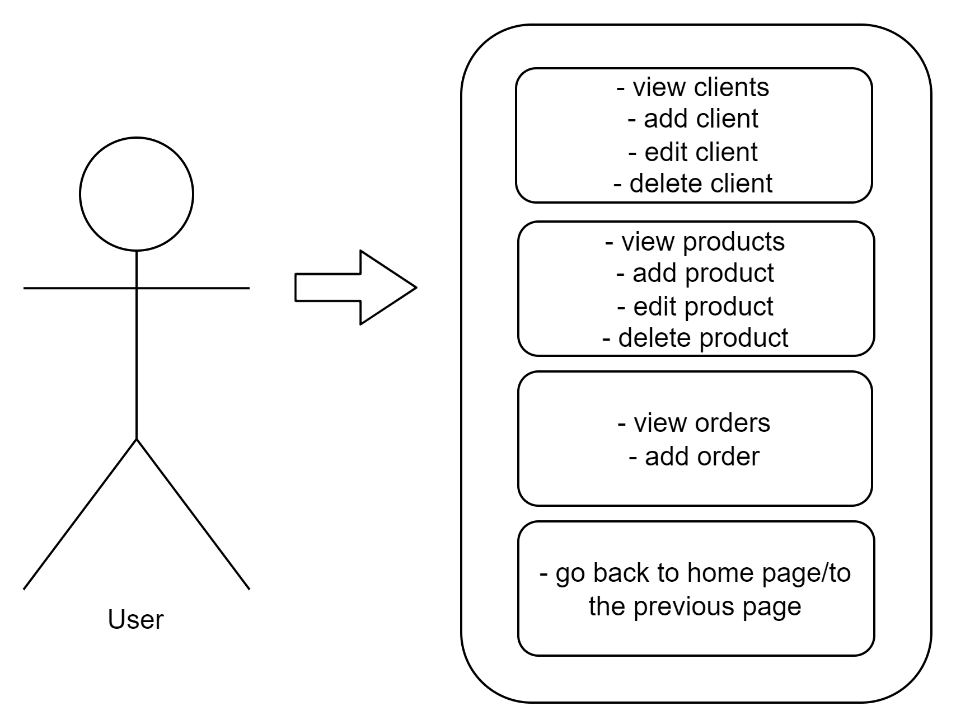
7. The user presses on the “Add order” button

8. The user is brought back to the main orders window and can now see the recently added order

What also happens here after the order is placed is that the bill is also created and can be seen in the files afterwards.

**Alternative sequences:**

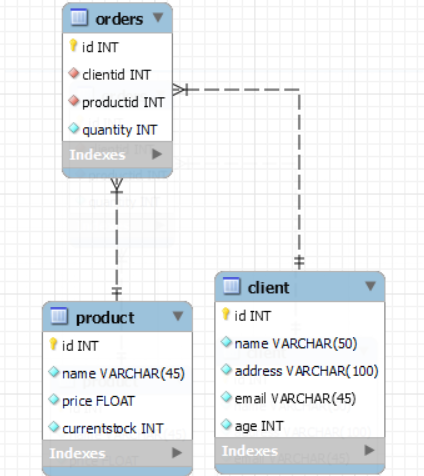
* From step 4: the user decides that he wanted to do an action not related to the orders, so he pushed the home button and is brought back to the main, starting page
* From step 8.1: the desired quantity was greater than the one present on stock so an error window appears
* From step 8.2.1: the user can press the home button to return to the main starting page
* From step 8.2.2: the user can shut down the application



1. **Design**

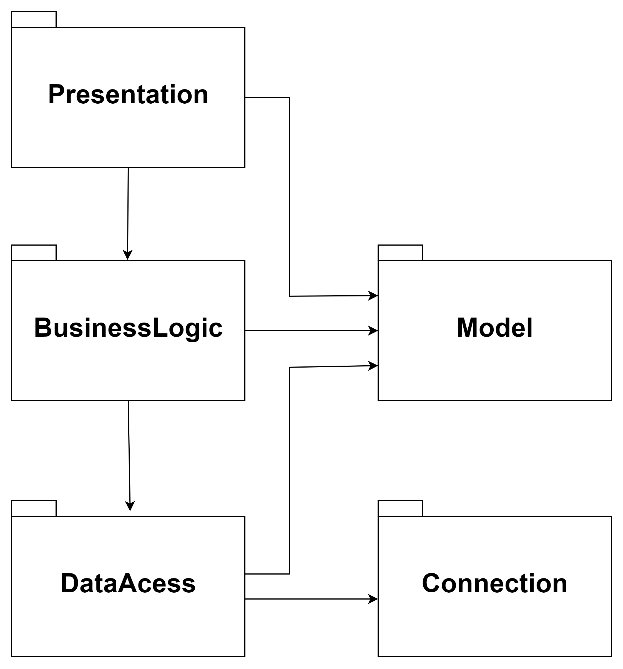
When talking about the design of the project, there are 2 main area we can mention: the package + class organization and the database. Both of them are as important in the clear organization of the project, as they contain both the organization of the code and the data storage.

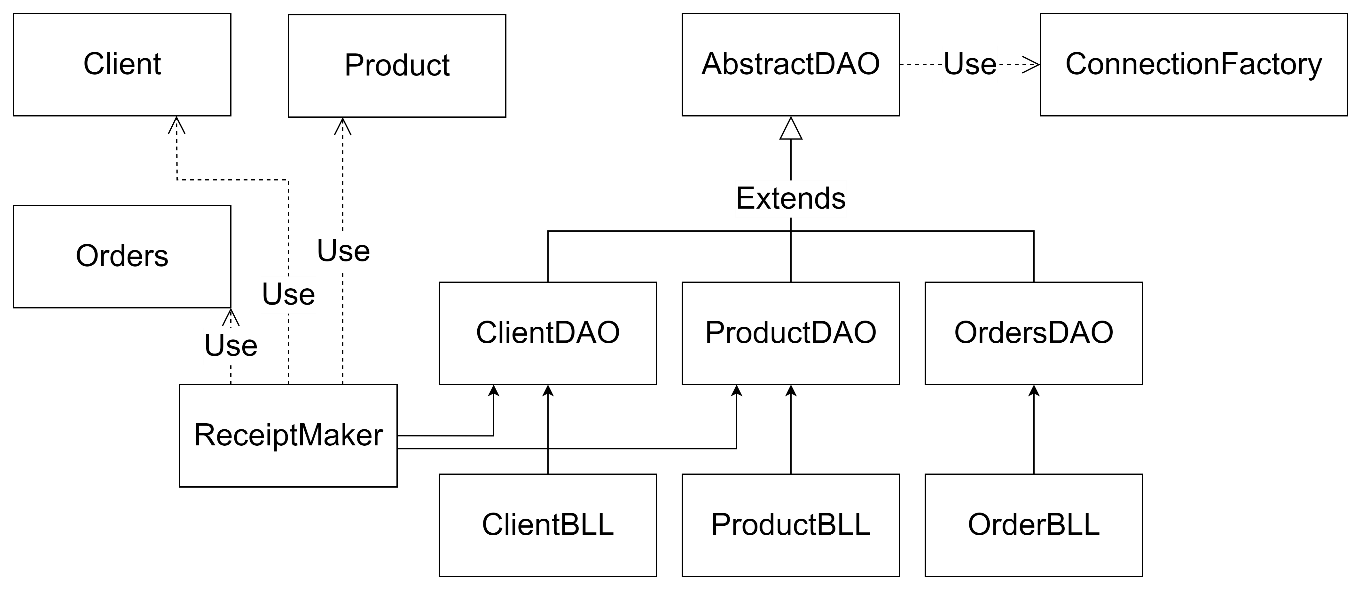
* 1. **Database**

****The relational database consists of 3 tables: Client, Product and Orders. Intuitively, they are linked through the ids of the product and the client.

* 1. **Packages and relationship between them**

The architecture used in this project is a layered architecture, made out of five packages: BusinessLogic, DataAccess, Model, Connection and Presentation (this package is, as well, divided into smaller packages in order to keep things more organized. Therefore, it consists of the Client, Order and Window packages and a standalone class, the StartingWindow one). Their roles are as it follows:

* The BusinessLogic creates the connection between our application and the DataAccess classes. It basically contains part of the logic, together with the controllers.
* The DataAccess contains an AbstractDAO class and 3 other classes that extends it. The main functionality of it is creating the queries of adding, deleting, selecting and updating instances of tables.
* The Connection handles the connection between the application and the database
* The Presentation contains the frames of the graphical interface and their controllers. The controllers, as above mentioned, also hold part of the decision-making process of the application. The smaller packages of the Presentation package are the Client, Order and Product one, present in order group windows and controllers which are related in their purpose.
* The Model holds the basic classes we will be working with in the whole application, those being, in our case: Client, Product and Orders.
  1. **Division into classes – without the use interface part**



1. **Implementation**

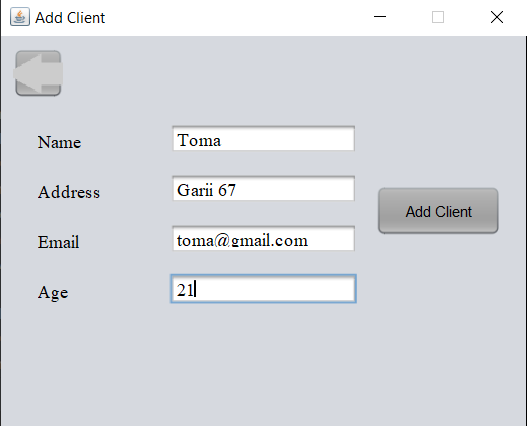
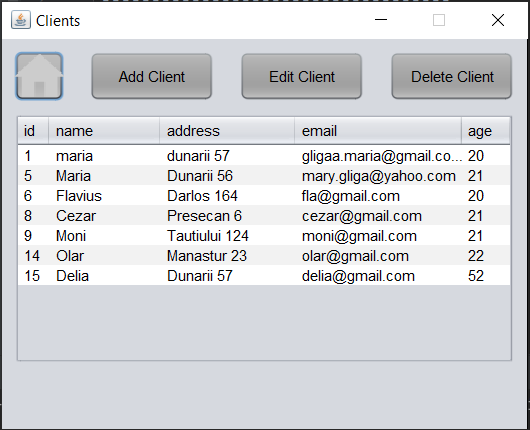
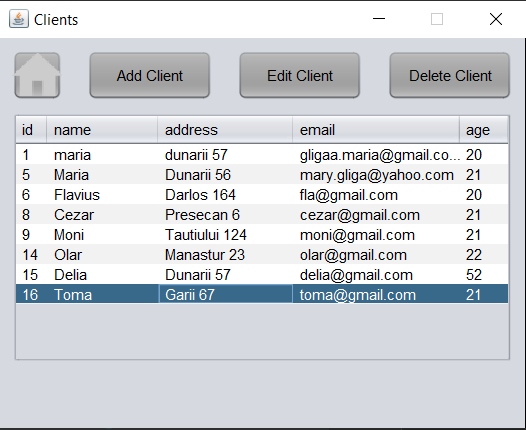
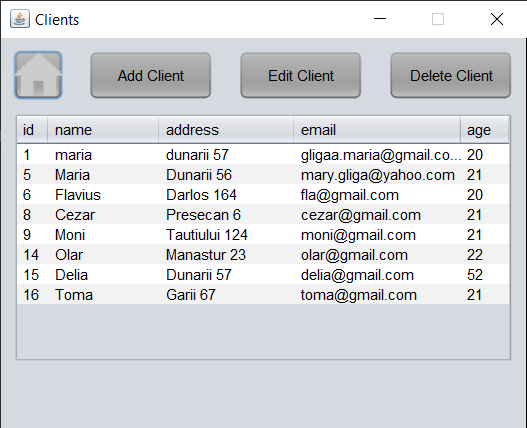
One of the most important classes of the project is the AbstractDAO class, as it manages with the queries needed in order to interact with the database and therefore save, edit and receive data. For each of the queries, there are queries firstly prepared which are later on filled in with values in the methods to come. I also added a getData method which returns the data of the tables in the form of a matrix of type String, in order to interact more easily with the graphical user interface (the JTable requires this particular type for the received data). Besides this, it can also provide the header of the columns.

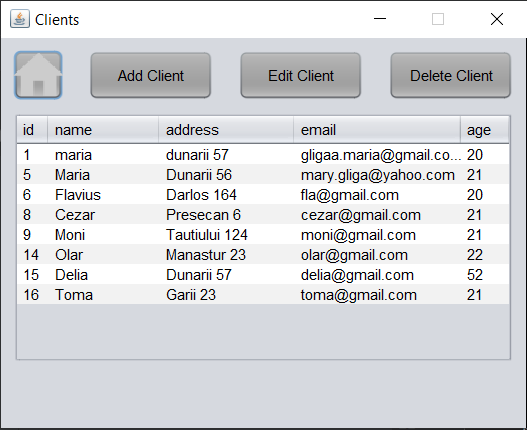
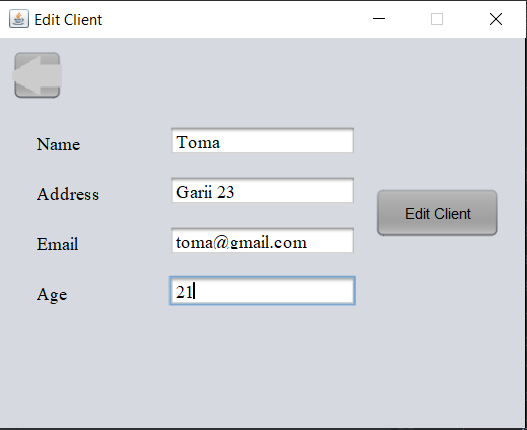
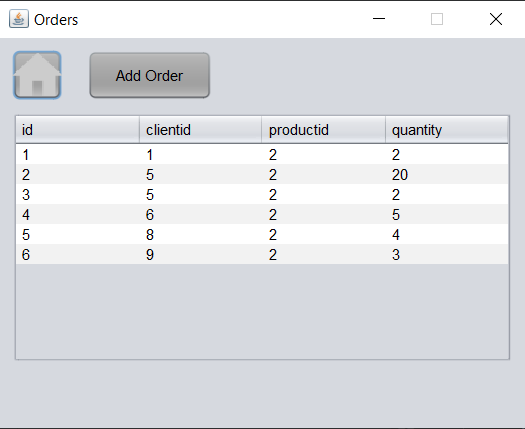
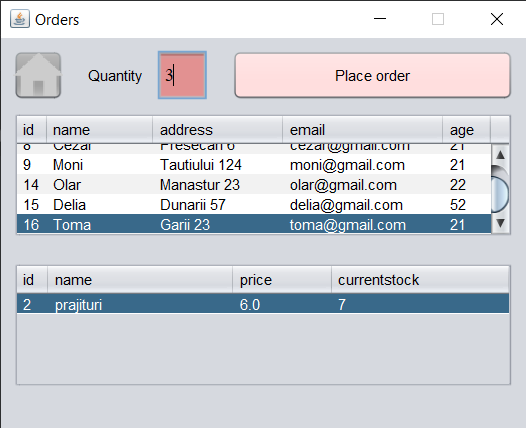
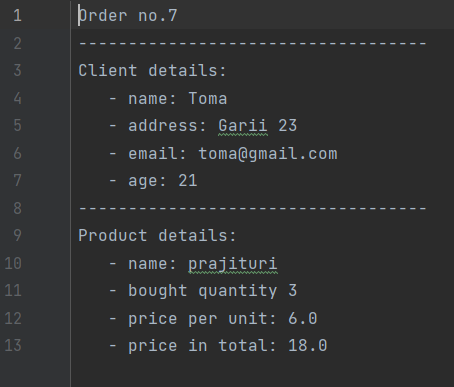
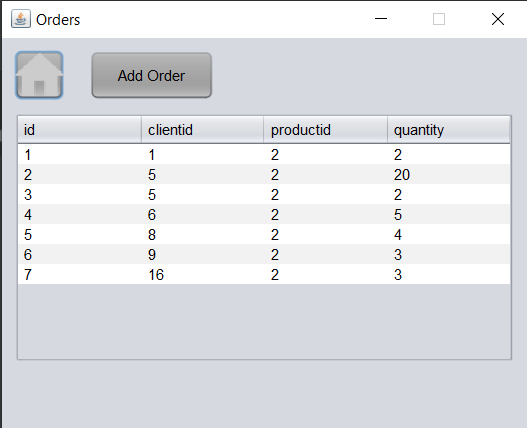
Later on, the ClientDAO, ProductDAO and OrdersDAO extend the previously mentioned class, making it specific for their own type. All the methods created inside these classes are later on called by the ones in the BusinessLogic packages. The BusinessLogic also contains the ReceiptMaker class which, when called by one of the interface controllers, creates a receipt based on the order that it is given as a parameter.

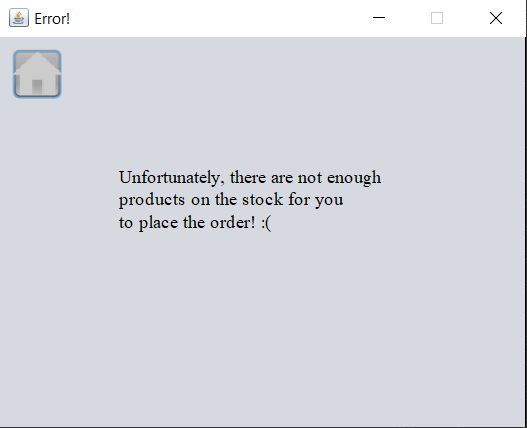
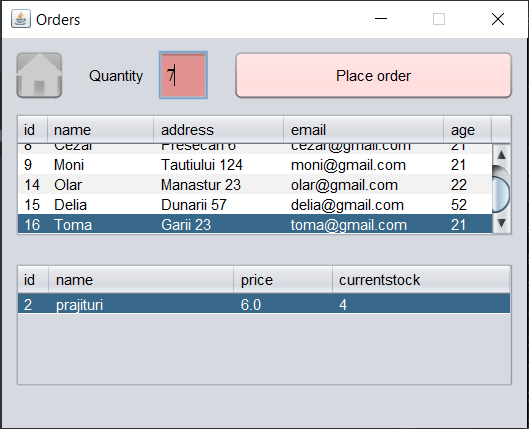
The other part of the project containing decision-making are the controllers of the frames from the graphical interphase. The main trick there was to restart windows or update them between actions such that the information shown would be the latest one.

Important to be mentioned as well is the ConnectionFactory class which deals with the creation of the connection between the code and the relational database.

1. **Results**

In order to demonstrate at least parts of the way the application works, I followed some of the user cases and screenshotted the results:





1. **Conclusions**

The whole project has enabled me to learn how to work together with a database and, moreover, to see the usefulness in working in an as general as possible manner as possible, in order to later on be able to extent the project without a lot of extra work. Compared to the previously realized projects, in terms of the graphical interface, the current project also enabled me to learn how to switch in between windows and update the displayed information.

What is more, I have deepened my knowledge on how to work with a layered architecture.

Future developments for the project:

* Use a general, abstract way for writing the code for the graphical interface, as a lot of the code tends to repeat due to the similar or identical actions that can be performed in certain cases
* Add a functionality which allows the tracking of the order and its status
* I would like to see what ways there are to update the data displayed to the user in order to make the application work faster

1. **Bibliography**

* <https://dsrl.eu/courses/pt/>
* <https://www.youtube.com/watch?v=e8g9eNnFpHQ>
* <https://www.geeksforgeeks.org/java-swing-look-feel/>