Assignment\_3

**DATABASE MANAGEMENT SYSTEM**

**Teacher**: Viorica Rozina Chifu

**Student**: Furcovici Teodora-Melania

**Group**:30421

1. **Assignment objective**

* **Main objective**

The purpose of this assignment was to simulate the process of  customer orders for a warehouse.

* **Sub-objective** 
  1. Analyze the problem and identify the requirements
  2. Design the relation between database and application
  3. Implement relation between database and application
  4. Test the relation between database and application

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

* **Overview**

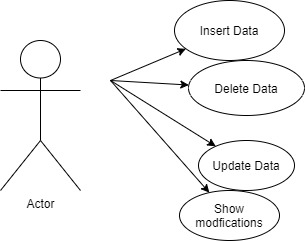
This application should be able to fulfil all the requirements in order to print, modify and keep track of orders, clients and products. These are stored in a relational MySQL database, along with the information about the users which have access to the system. This way, all the data is easier to retrieve and access from different computers.

* **Modelling**

The user will be able to use the INSERT button in order to insert a new client, order or product in the database. Also, if the user presses the DELETE button it deletes certain row, and if the UPDATE button is pressed then it will modify a certain row from the database.

After the SHOW MODIFICATIONS button is pressed, the result will appear in the interface.

* **Scenarios and use cases**

****

1. **First use case (Insertion)**

**Normal scenario**

The user inputs all the data in a correct manner and it presses the button INSERT and the SHOW MODIFICATION button is activated, then updated database will be displayed in the interface.

**Alternative scenario**

The user doesn`t fill in all the compulsory fields or in some fields the data introduced doesn`t respect the imposed format. In this case, a message will issue informing the user that the values are not correct.

1. **Second use case (Deletion)**

**Normal scenario**

The user inputs all the data in a correct manner and it presses the button DELETE and the SHOW MODIFICATION button is activated, then updated database will be displayed in the interface.

**Alternative scenario**

The user doesn`t fill in all the compulsory fields or in some fields the data introduced doesn`t respect the imposed format. In this case, a message will issue informing the user that the values are not correct.

1. **Third use case (Update)**

**Normal scenario**

The user inputs all the data in a correct manner and it presses the button UPDATE and the SHOW MODIFICATION button is activated, then updated database will be displayed in the interface.

**Alternative scenario**

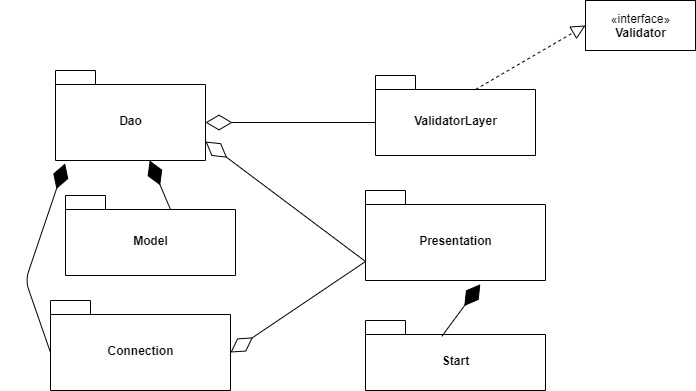
The user doesn`t fill in all the compulsory fields or in some fields the data introduced doesn`t respect the imposed format. In this case, a message will issue informing the user that the values are not correct.

1. **Fourth use case (Show)**

**Normal scenario**

The user presses the SHOW MODIFICATIONS button and the table is displayed in the interface.

1. **Design (design decisions, UML diagrams, data structures, class design, interfaces, relationships, packages, algorithms, user interfaces)**

* **Black Box**
* **O imagine care conține text

  Descriere generată automatPackages**

In develop my application after the Layered Architecture Design Pattern. I believe that the data becomes more re-useable and easy to follow.

My application is divided into 6 parts because it respects the Layered Architecture Design Pattern.

**Validator Layer** package represents the Business Logic of the application, and it contains all the classes necessary to validate the input data.

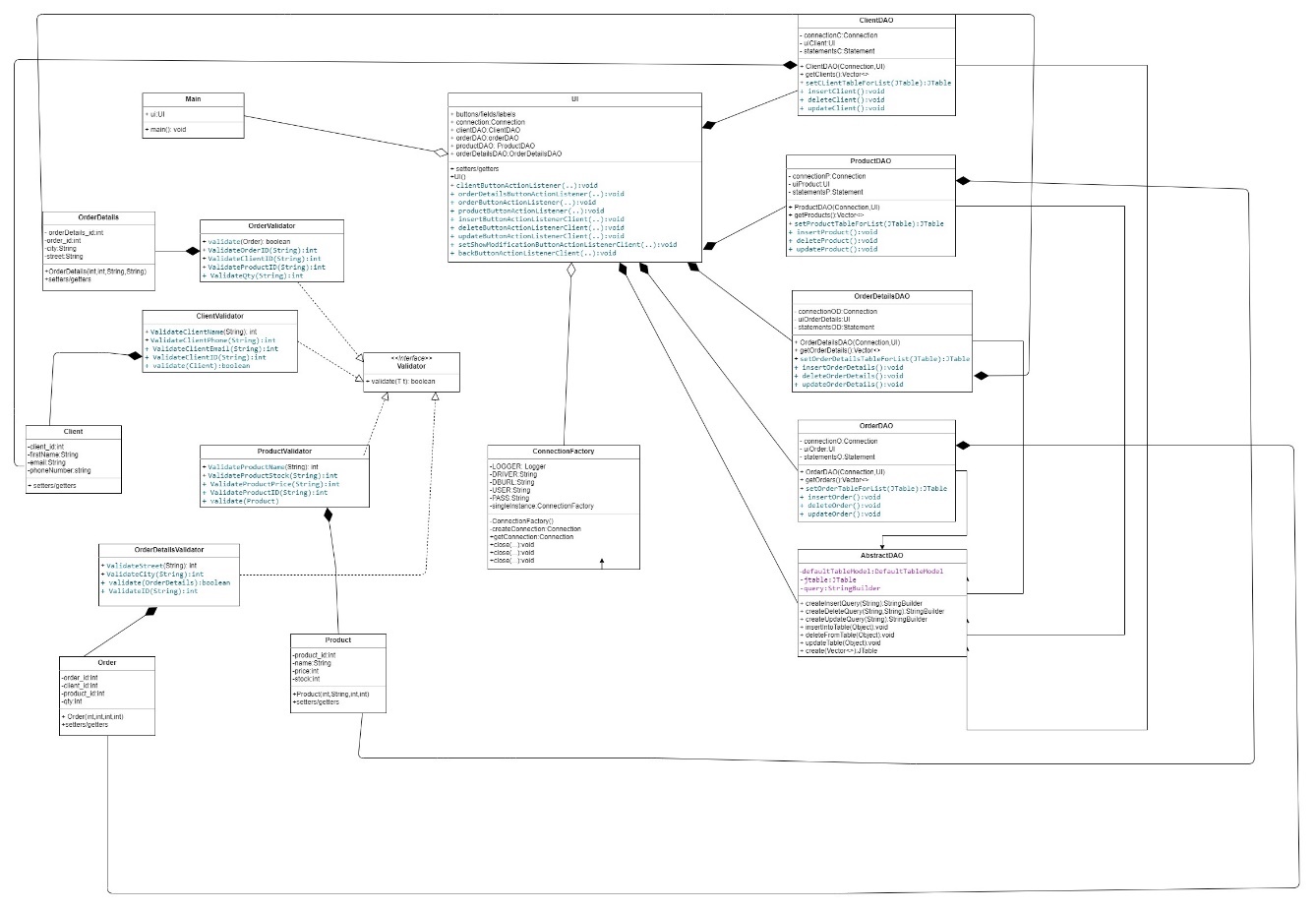
**Connection** package represent the Database Connection, where all the needed connections between Java and MySQL are made.

**DAO** package contains the database access classes, where the actual queries to perform the basic operations on a table in sql are implemented.

**Model** package represents an object for carrying data. This is the “invisible” layer of the application, because with the help of the classes contained in this package the application is fulfilling the requirements.

**Presentation** package represents the visualization of the data that model contains. Practically, this package has the classes that are used for constructing the interface of the application.

**Start** package contains the class from where the entire application is turned on.

* **UML Diagrams**

* **Data structures**

Data Structures I have used in this project are basic, such as primitive data types (integer and double, mostly) and reference data types. A complex data structure would be the Vector.

The reference type I used developing this application was Client, Product, Order and OrderDetails.

Vector are preferred here due to the fact that they are very easy to use and have a lot of methods, which makes the life of the programmer easier. In this case we don`t have to worry about always keeping the length(as it would have happened for simple arrays) and also it easier to add to an Vector rather than to an array.

* **Class Design**

I choose not to write all the code in one class, because it is more readable for persons who see the application for the first time and because if it happens to appear an error, I have to search it in the entire program and it is loss of time.

Fora better understanding of the application and to be more accessible, I choose to divide the program in 6 major parts: **MODEL,** **CONNECTION**, **DAO, PRESENATION, VALIDATOR LAYER**  and **START**. Each of this parts in the application is represented as a separated package, which contains one or more classes.

For the PRESENTATION package, I designed a class where I take care of all problems that may arise when designing the GUI.

For the MODEL package, I designed one class for each table from database where I all the data needed for each client, order, product or order detail.

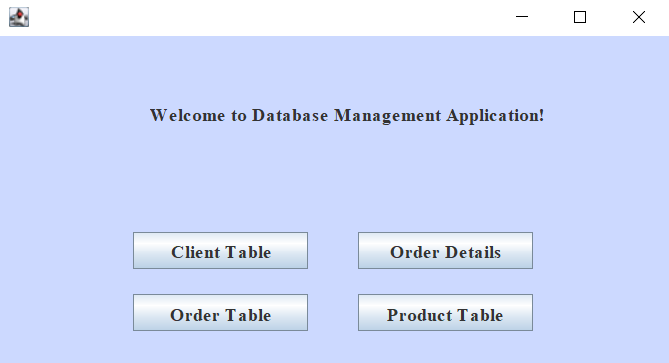
For the VALIDATOR package, I designed one class for each table from database where it validates the data given by the user.

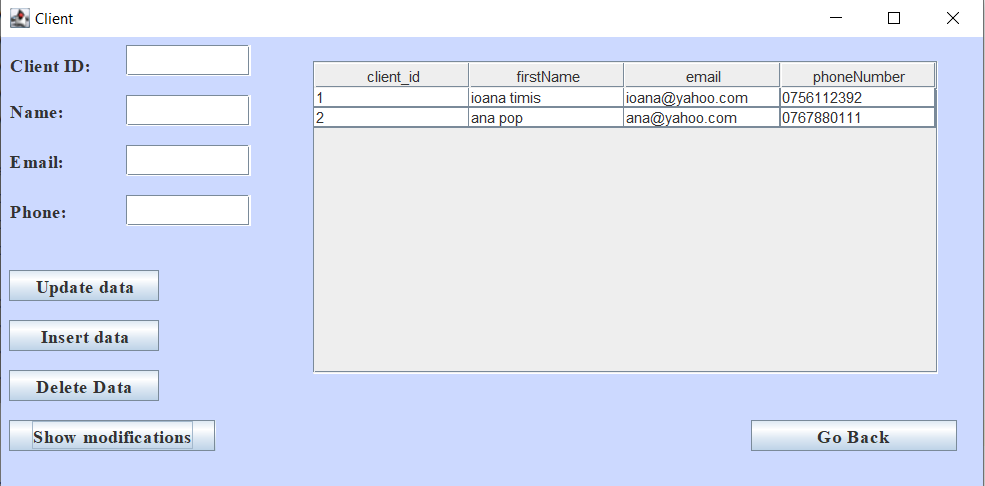
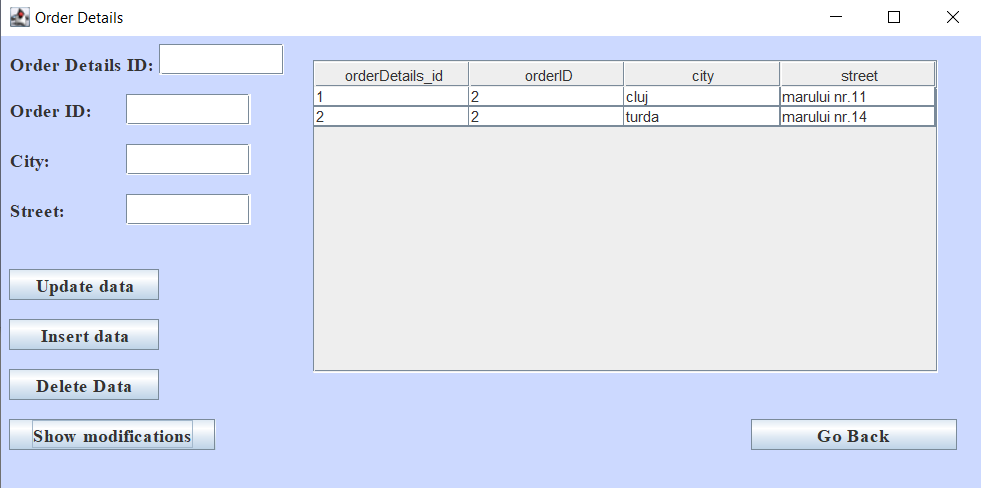
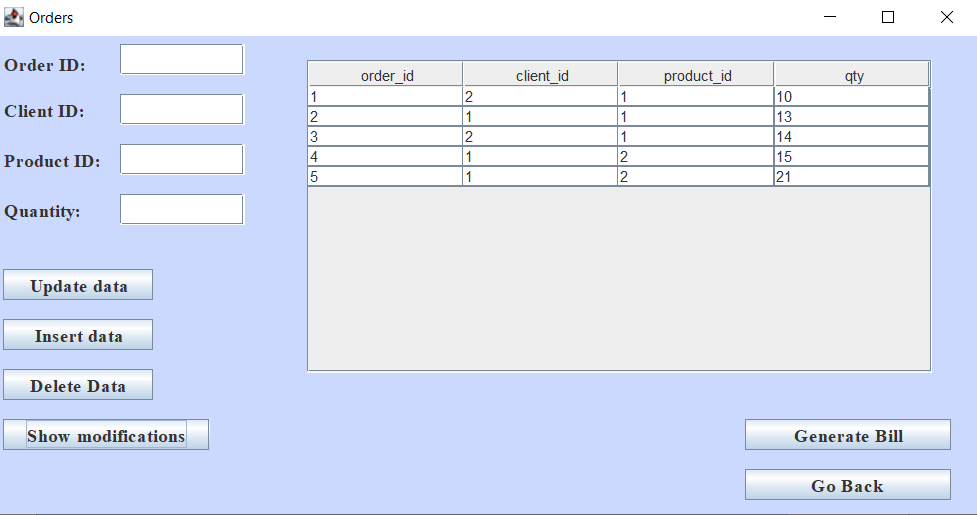
For the CONNECTION package, I implement a class that keeps track of all the needed connections between the application and the database.

For the DAO package, it has one class for each table from the database where it performs the basic operations.

* **User Interfaces**

The user interface is the connection between the user and the programmer, because with it the application can executes the commands.



O imagine care conține masă

Descriere generată automat

The user inputs the necessary data for performing one certain operations and see the results. The result will be visible in the Table area, just after he/she pressed the button with the operation she/he wants to perform.

Beside this, the user has to respect some conditions in order to see the result:

* + The values introduced in the destinated fields must contain only digits and be a positive integer value, in case an ID is required, or a String otherwise.
  + Then another compulsory condition that must be fulfilled is that all the fields must contain a certain number/name, otherwise the application won`t work.

1. **Implementation**

As I said above, my application follows the “rules” of the Layered Architecture pattern, and so I decided to split my program into more classes, which, practically, is the scope of Java Programming. The application is divided as follows:

* 1. **MODEL**
     1. **Client**

This is class is meant for storing client id, name of the client, email and unique phone number for each client. Here I have 4 instances for the information above mention and a method for listing each client information.

* + 1. **Product**

This is class is meant for storing product id, name of the product, price and stock for each product. Here I have 4 instances for the information above mention and a method for listing each client information.

* + 1. **Order**

This is class is meant for storing order id, the id of the client which puts a certain order , the id of the ordered product and the quantity for each order. Here I have 4 instances for the information above mention and a method for listing each client information.

* + 1. **OrderDetails**

This is class is meant for storing order details id, the id of the order, city and street where to send the order. Here I have 4 instances for the information above mention and a method for listing each client information.

* 1. **DAO**
     1. **Abstract DAO**

In this class are implemented all the basic operations that can be performed on a table in MySQL.

In the **insertIntoTable** method I insert a generic object in a certain table which is mentioned with the help of the parameter **Object obj**. Then, using Java Reflection, it accesses all the fields of the table and based on its type it inserts into the table from MySQL.[1][2]

O imagine care conține text

Descriere generată automat

O imagine care conține text

Descriere generată automat

Then, another important method is the one for updating a certain row from any table. The logic behind is approximately the same as for insert, the only difference is when it adds the where clause, because there the update is done based only on the id.[3][4]

* + 1. **ClientDAO**

This class contains 3 instances, one for realizing the connection between database and application, one for the interface part and the last one is for creating a query that will be execute later from the application.

Inside the **insertClient** method, it inserts a client in the Client table from MySQL. Also, in this method I fetch the information about the client given by the user and then validate the data in order to verify that it can be successfully inserted into table.

O imagine care conține text

Descriere generată automat

O imagine care conține text

Descriere generată automat In the **updateClient** method, I fetch the data from the user interface, the it is validated and afterwards we search the client with the id input by the user. When the program finds the client, it updates its data and also update the specified row from the table.

* + 1. **Product DAO**

This class contains the same thing as the above one, the only difference is that is destinated for the product table. Also, it has the same instance variables.

* + 1. **OrderDAO**

This class contains the same thing as the above one, the only difference is that is destinated for the order table. Also, it has the same instance variables.

O imagine care conține text

Descriere generată automat

In the above class, I create the query and based on the result stored in ResultSet, it iterates through the columns of the table and with the help of the create() method it adds the columns to the table.

* + 1. **OrderDetailsDAO**

This class contains the same thing as the above one, the only difference is that is destinated for the order details table. Also, it has the same instance variables.

* 1. **CONNECTION**
     1. **ConnectionFactory**

The purpose of this class was to link the application to a certain database, which will be specified in a form of a link. In this class, all the methods are implemented to connect or to close the connection between database and application.

* 1. **VALIDATOR LAYER**
     1. **Client Validator**

This class is meant for validate the user`s data, for each property of a client (i.e name ,id, email or phone).

* + 1. **Order Validator**

This class is meant for validate the user`s data, for each property of a order (i.e id, id of the client, id of the product or quantity).

* + 1. **Product Validator**

This class is meant for validate the user`s data, for each property of a product (i.e id ,name, price or stock).

* + 1. **Order Details Validator**

This class is meant for validate the user`s data, for each property of a order detail (i.e id ,id of the order, city or street).

* + 1. **Validator Interface**

This interface contains only one method to be implemented.

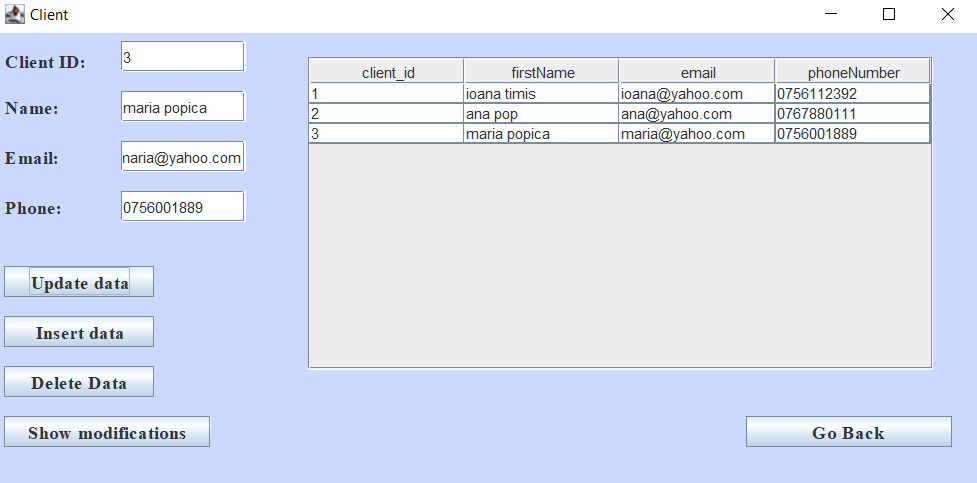
* 1. **PRESENTATION**
     1. **UI**

In the UI class are defined all the elements needed to form the Graphical User Interface and the methods for handling the button actions.

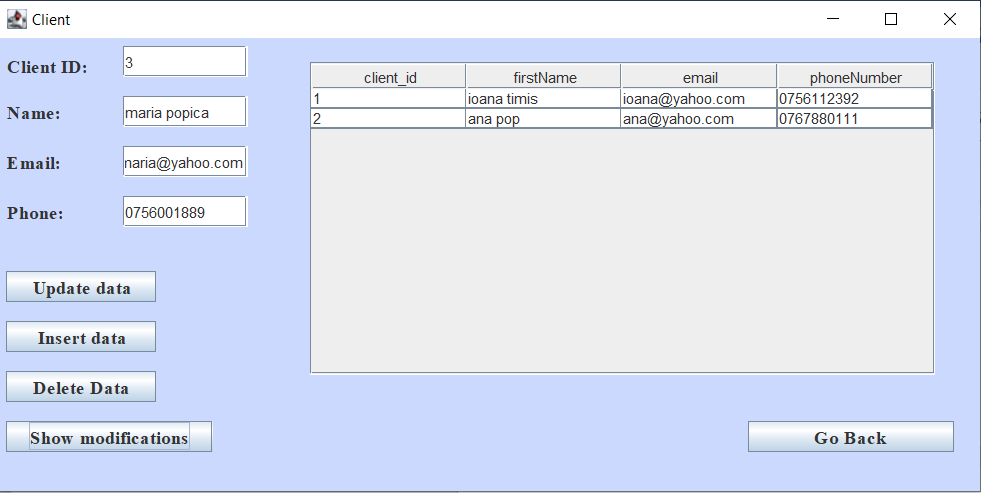
1. **START**
   * 1. **Main**

The class from where it starts the application.

1. **Results**

O imagine care conține masă

Descriere generată automat The results are found in the Table area and in a .txt file also. The results from the user interface are consisted of a table which is identical to the one from database. In the .txt file is the bill for all the orders that have been placed in thus system.



**Insert, Delete and Update on Client**

**O imagine care conține masă

Descriere generată automat**

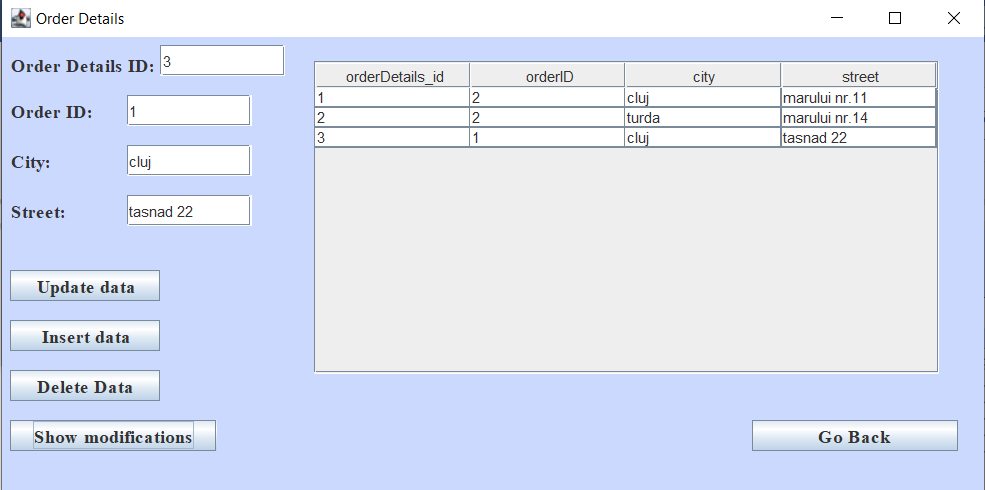
**O imagine care conține masă

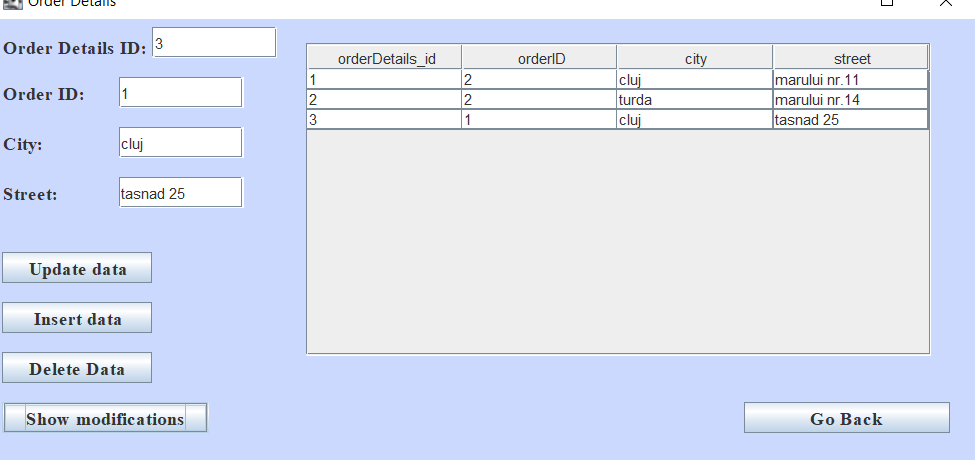
Descriere generată automat**

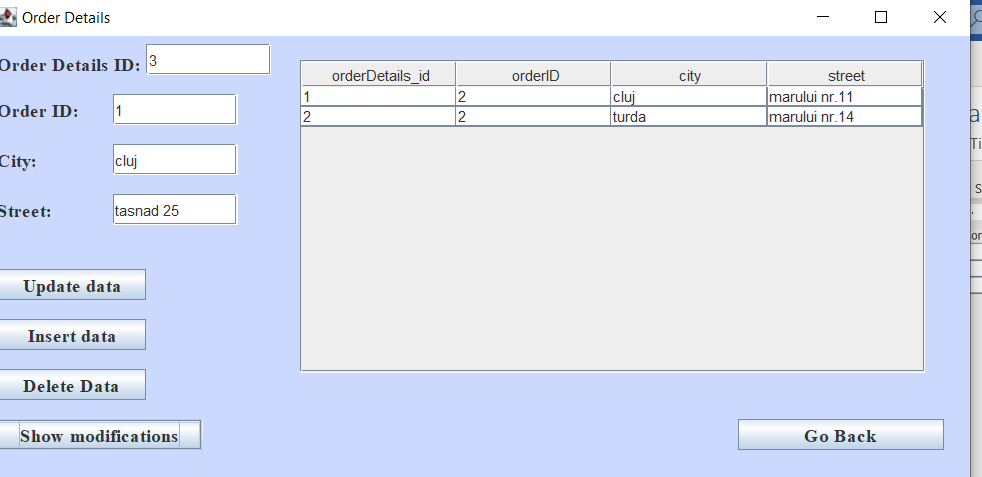
**O imagine care conține masă

Descriere generată automat**

**Insert, Delete and Update on Orders**

****





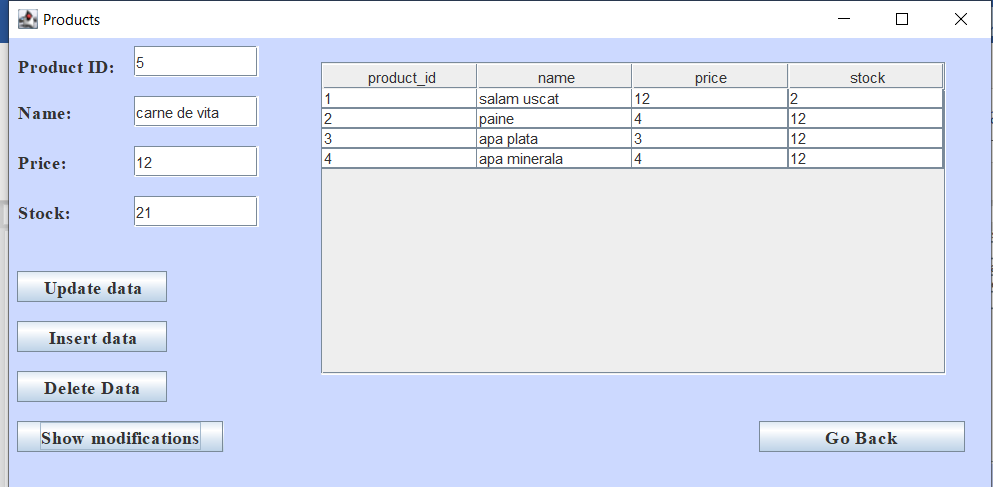
**Insert, Delete and Update on OrderDetails**

**O imagine care conține masă

Descriere generată automat**

**O imagine care conține masă

Descriere generată automat**

****

1. **Conclusions**

For this assignment it was harder to build a strategy and implement it, because the notion of thread was totally new to me. Anyway, I found this very challenging at first, but then I got used to it. I consider that it was useful, due to the fact that we learned how the connectivity with a database working and how we are supposed to modify or list the contents of a table.

I arrived at the conclusion that facing problems with your code and trying to make it work by yourself, through the mean of research, has the benefit of learning new concepts and a better use of the known ones.

About further developments, I think that one advantage would be to have this application implemented using threads.

1. **Bibliography**

[1] <http://tutorials.jenkov.com/java-reflection/private-fields-and-methods.html>

[2] <http://tutorials.jenkov.com/java-reflection/fields.html>

[3] <https://www.javatpoint.com/java-int-to-string>

[4] <https://www.geeksforgeeks.org/vector-setsize-method-in-java-with-example/>