

Orders MANAGEMENT

DOCUMENTATION - ASSIGNMENT 3

Programming Techniques

STUDENT: DOMȘA-STÎNĂ ANDA

GROUP: 30423

# TABLE OF CONTENTS

[1. Objectives 4](file:///C:\Users\User\Downloads\PT2021-2022_Documentation_Template.doc#_Toc95297885)

[2. Analizing the requirements 4](file:///C:\Users\User\Downloads\PT2021-2022_Documentation_Template.doc#_Toc95297886)

[3. Design 7](file:///C:\Users\User\Downloads\PT2021-2022_Documentation_Template.doc#_Toc95297887)

[4. Implementation 11](file:///C:\Users\User\Downloads\PT2021-2022_Documentation_Template.doc#_Toc95297888)

[5. Results 14](file:///C:\Users\User\Downloads\PT2021-2022_Documentation_Template.doc#_Toc95297889)

[6. Conclusions 14](file:///C:\Users\User\Downloads\PT2021-2022_Documentation_Template.doc#_Toc95297890)

[7. Bibliography 15](file:///C:\Users\User\Downloads\PT2021-2022_Documentation_Template.doc#_Toc95297891)

# Objectives

1. Main objective

*Project specification:*

Consider an application Orders Management for processing client orders for a warehouse. 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

1. Sub-objectives:

*Analyze the problem and identify requirements*

The first step in implementing the program is to understand the problem, to identify the tasks it should perform and to model its functionalities using an object-oriented approach. This will be forward detailed in chapter 2.

*Design the simulation application*

The next step in solving this project’s requirements is to design the orders management application by dividing it into smaller components using structural diagrams. This will be further discussed in chapter 3.

*Implement the simulation application*

Now that the requirements are understood and modeled, the next step is to implement the actual java program by writing code for the needed classes in Intellij. This step will be covered in chapter 4.

*Test the simulation application*

The last step of the project development is to test the application, a process which I have described in chapter 5 of the documentation.

# Analyzing the requirements

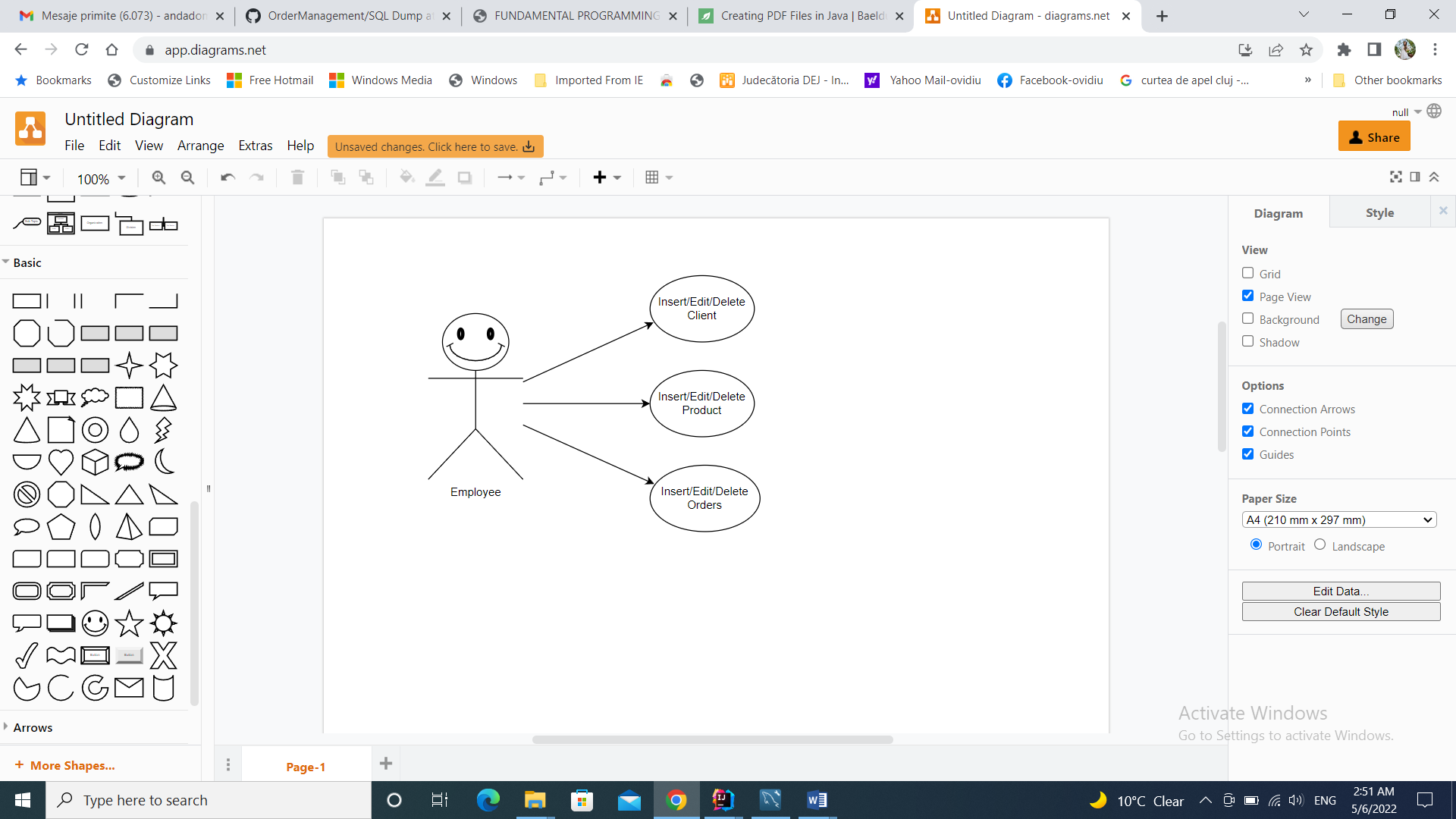
The first step in implementing the program is to understand the problem, to identify the tasks it should perform and to model its functionalities using an object-oriented approach.

PROBLEM: “Managing the products, the clients and the orders for a warehouse using handwritten registries is difficult and time consuming ”.

SOLUTION: Orders Management Application.

This application should be able to fulfil all the requirements in order to display, 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.

**Use-Cases:**



The use case diagram presents the actor (the employee), which in our case is the user that interacts with the application. He can choose to perform several actions on the warehouse database tables, such as inserting, editing and deleting clients, products or orders, and also viewing the data from the tables of the relational database.

*Requirements:*

1. Functional requirements of the project:

* The warehouse application should allow the user to insert clients in the database.
* The warehouse application should allow the user to insert products in the database.
* The warehouse application should allow the user to insert orders in the database.
* The warehouse application should allow the user to edit clients’ data from the database.
* The warehouse application should allow the user to edit products’ data from the database.
* The warehouse application should allow the user to edit orders’ data from the database.
* The warehouse application should allow the user to delete clients from the database.
* The warehouse application should allow the user to delete products from the database.
* The warehouse application should allow the user to delete orders from the database.

1. Non-Functional requirements of the project:
   * The warehouse application should be intuitive and easy to use by the user
   * The application should have a straight-forward interface to facilitate the interaction with the database.
2. UI Requirements:

The I/O mechanism for the Orders Management Application must contain:

• One Java compatible device which can compile and run Java standalone applications

• A keyboard from which the user can input some data.

• A screen onto which he can see the input / output for the application.

*Possible scenarios:*

1. Use case: inserting a client, actor: employee.

Main success scenario:

* Step 1. The user opens the application and selects the Client button which will open the Client interface.
* Step 2. The user selects the task that he wants to perform by pressing the button corresponding to it, choosing form the 3 available ones: “Insert”, “Edit”, “Delete” (in our case we choose insert). That will open another window in which the employee will have to input the client’s details.
* Step 3. The Orders managing application performs the task and adds the new client in the table “Client” of the database.

Alternative scenario:

* The user inserts incorrect client details
* The program displays a warning pop-up
* The program returns to step 1

1. Use case: editing the details of a product, actor: employee.

Main success scenario:

* Step 1. The employee opens the application and selects the “Product” button which will open the Product interface.
* Step 2. The user selects a product he wants to alter and the task that he wants to perform by pressing the button corresponding to it, choosing form the 3 available ones: “Insert”, “Edit”, “Delete” (in our case we choose “Edit”). That will open another window in which the employee will have to input the product’s new details.
* Step 3. The Orders managing application performs the task and alters the selected product’s details in the table “Product” of the database.

Alternative scenario:

* The user inserts incorrect product details
* The program displays a warning pop-up
* The program returns to step 1

1. Use case: deleting an order, actor: employee.

Main success scenario:

* Step 1. The employee opens the application and selects the “Order” button which will open the Orders interface.
* Step 2. The user selects the Order that he wants to delete and the task that he wants to perform by pressing the button corresponding to it, choosing form the 3 available ones: “Insert”, “Edit”, “Delete” (in our case we choose “Delete”).
* Step 3. The Orders managing application performs the task and deletes the selected order’s details from the table “Order” of the database.

Alternative scenario:

* The user doesn’t select any order and just presses the “Delete” button
* The program displays a warning pop-up
* The program returns to step 1

1. Use case: inserting an order, actor: employee.

Main success scenario:

* Step 1. The employee opens the application and selects the “Order” button which will open the Orders interface.
* Step 2. The user selects the task that he wants to perform by pressing the button corresponding to it, choosing form the 3 available ones: “Insert”, “Edit”, “Delete” (in our case we choose insert). That will open another window in which the employee will have to input the order’s details.
* Step 3. The Orders managing application performs the task and adds the new order in the table “Order” of the database.

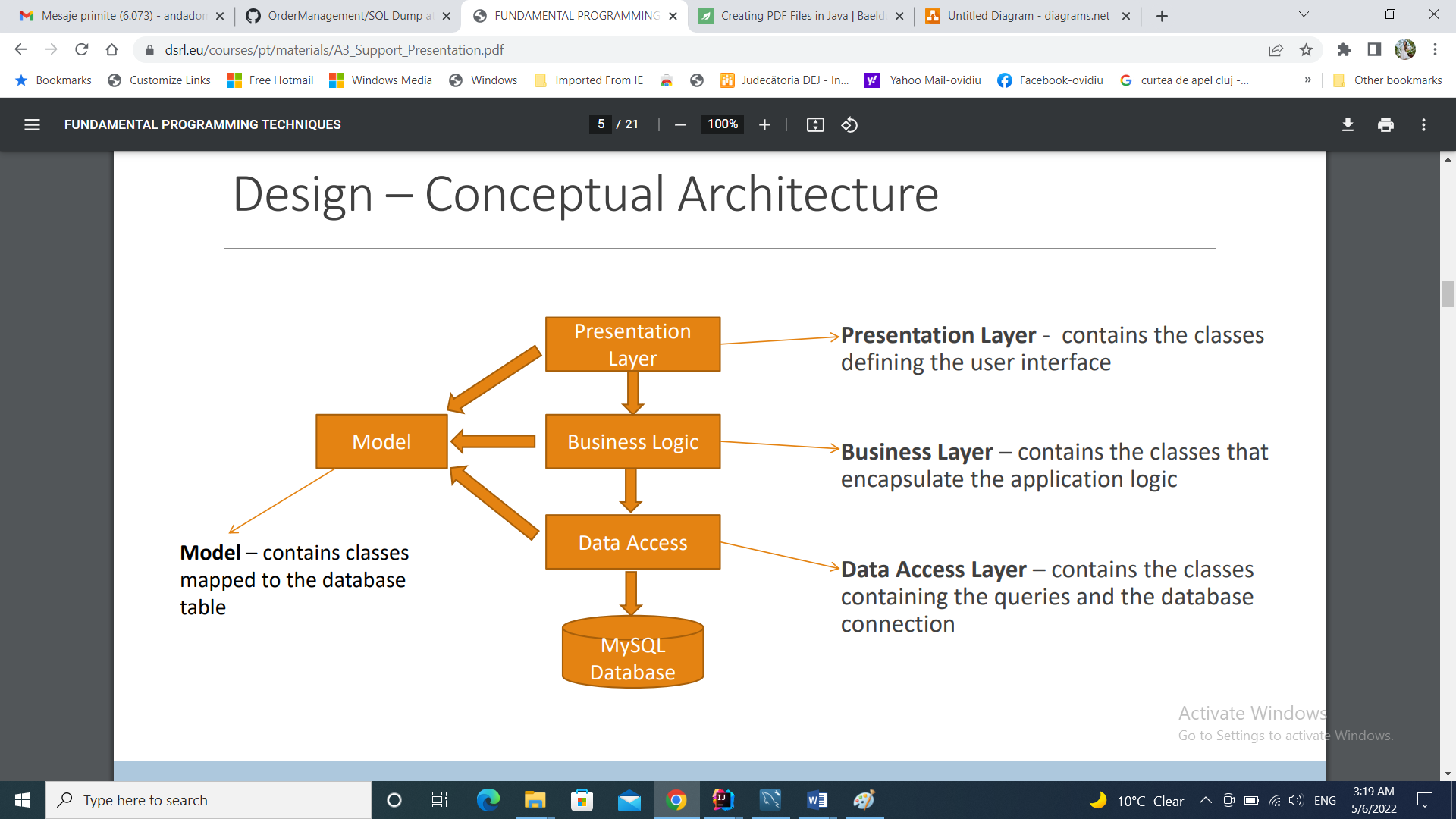
Alternative scenario:

* The user inserts incorrect order details
* The program displays a warning pop-up
* The program returns to step 1

*Modelling the problem:*

The most important objects that had to be modeled were of course the Client, Product and Order which are the classes that mirror the tables of the relational database we are working with.

# Design



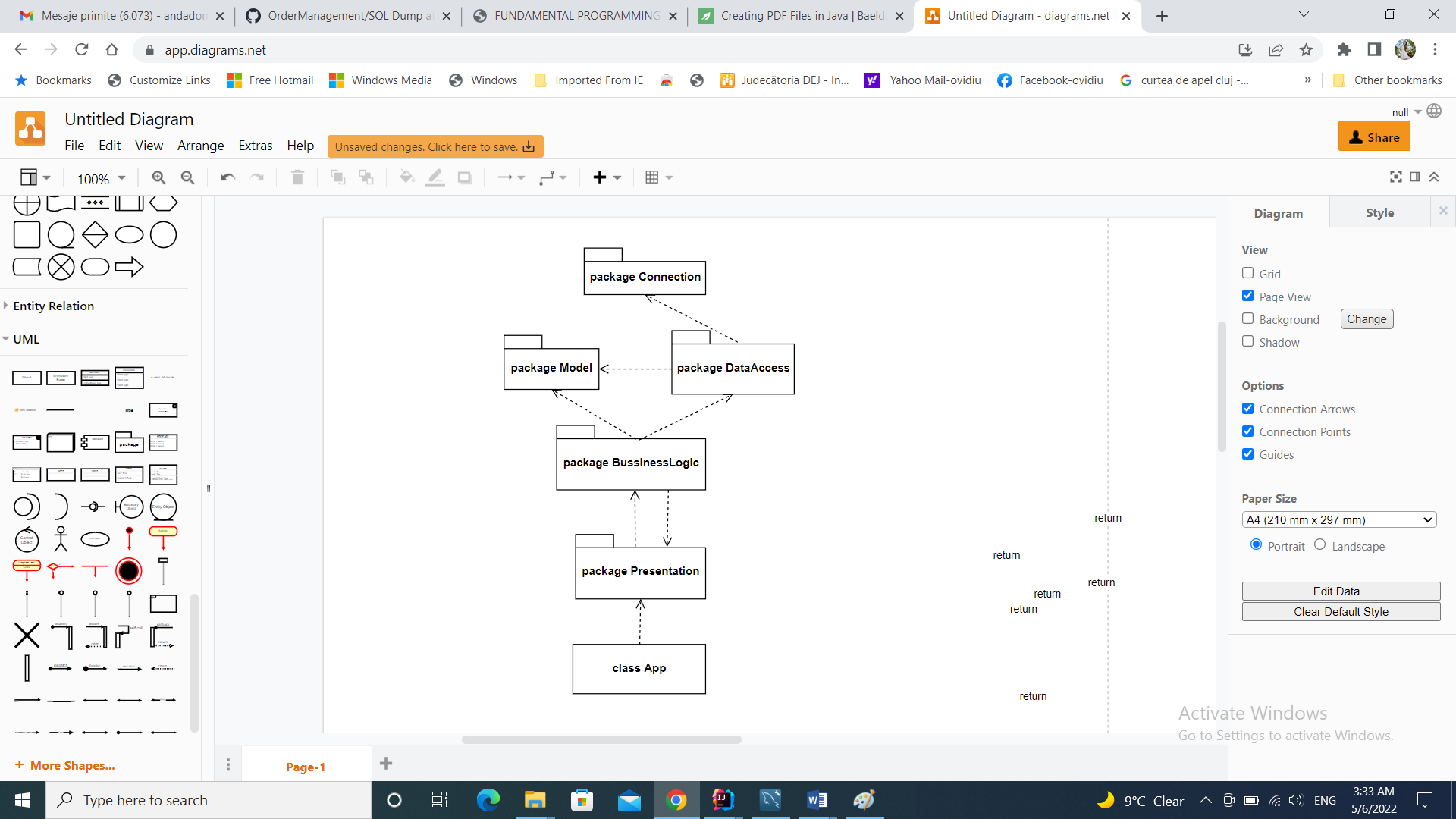
The next step in solving this project’s requirements is to design the Orders Management Application by dividing it into smaller components using structural diagrams.

*Data Structures*

The data structures that I have been working with in this problem are primarily primitive data types (especially integers and doubles) and a more complex one.

So I have decided to use ArrayList and List instead of the classic arrays because I think that they are more efficient from the point of view of memory management, performance and provide a faster access to their content. Also, the size of an ArrayList is not fixed and adding new elements to the list is very easy because we do not have to worry about exceeding the length of the list (array).

Package Diagram:



Java packages help in organizing multiple modules and group together related classes and interfaces.

**The App class** is the class that start the application and makes it run the user interfaces needed for interacting with the database.

**The Presentation package** is the package containing the classes defining the interface (views + controllers).

**The BussinessLogic package** is the package that contains the classes describing the application’s logic.

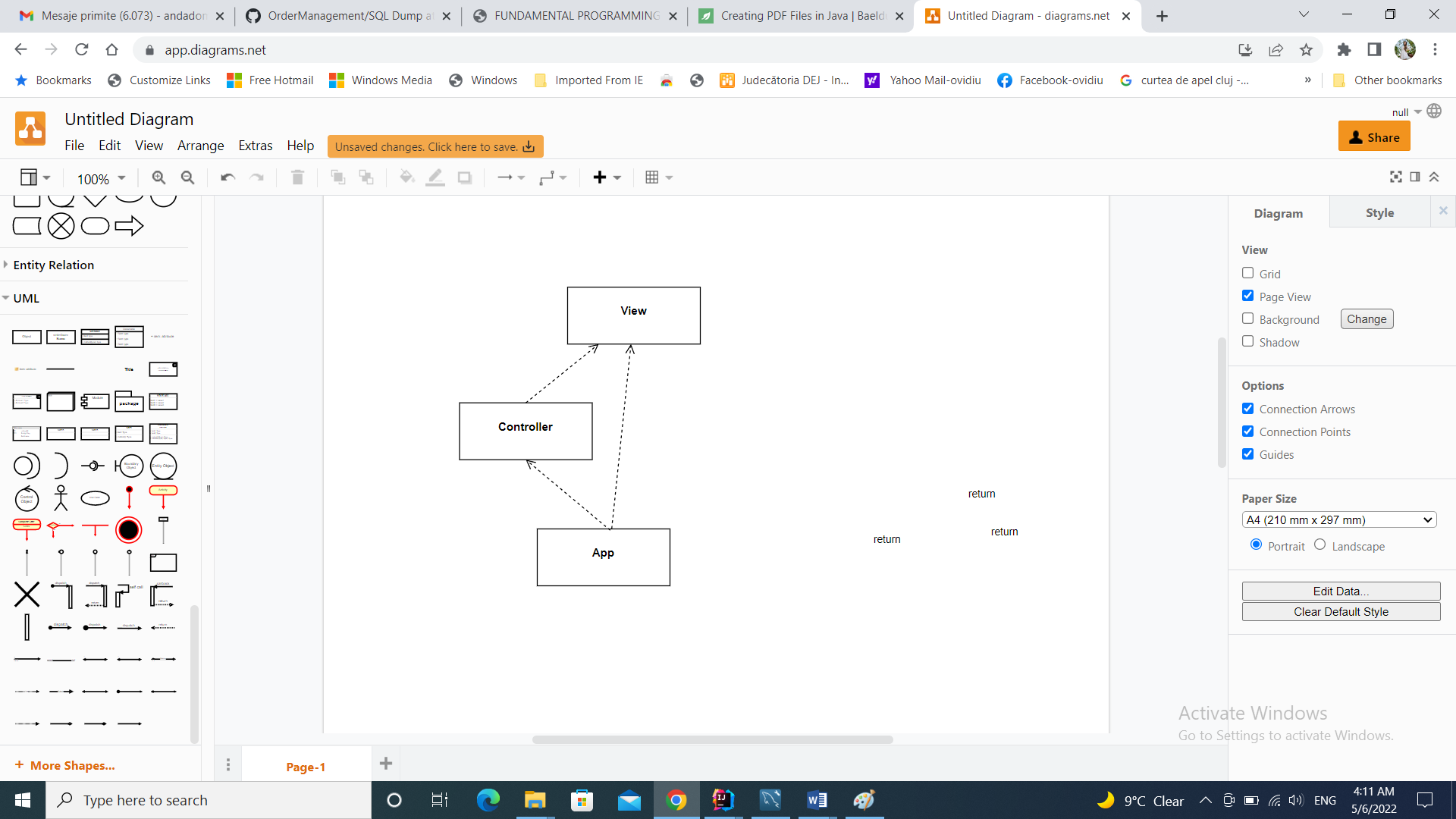
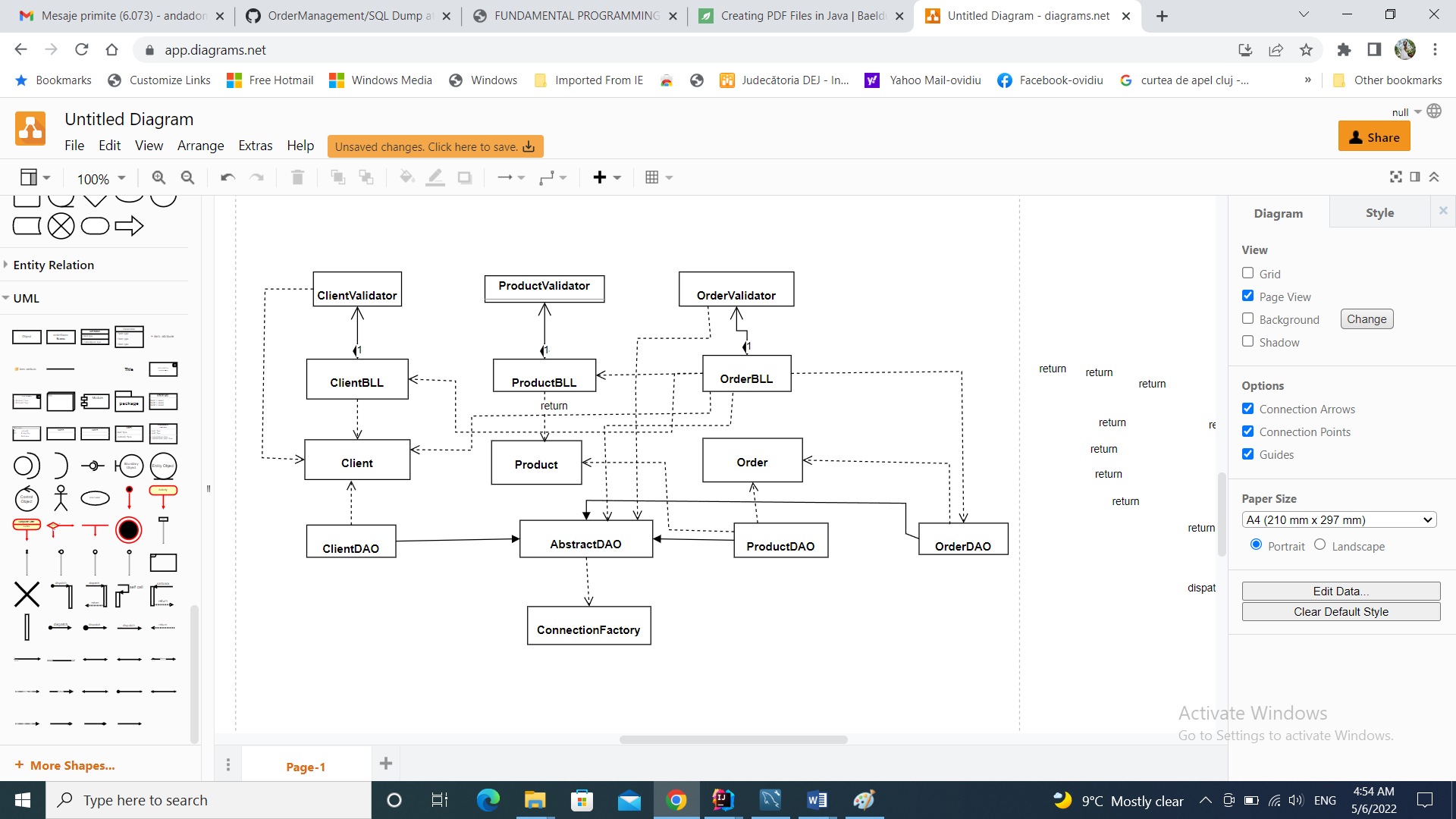
**The Model package** is the package containing the classes that mirror the database’s tables: “Client”, “Product”, “Order”.

**The DataAccess package** is the package containing the classes that implement the queries used to interact with the relational database.

And finally, **the Connection** package is the package that contains the classes which make the effective connection between the database and the application and allow the user to alter the tables through the warehouse application.

Class Diagram:

-without the presentation classes:

-generated by Intellij:

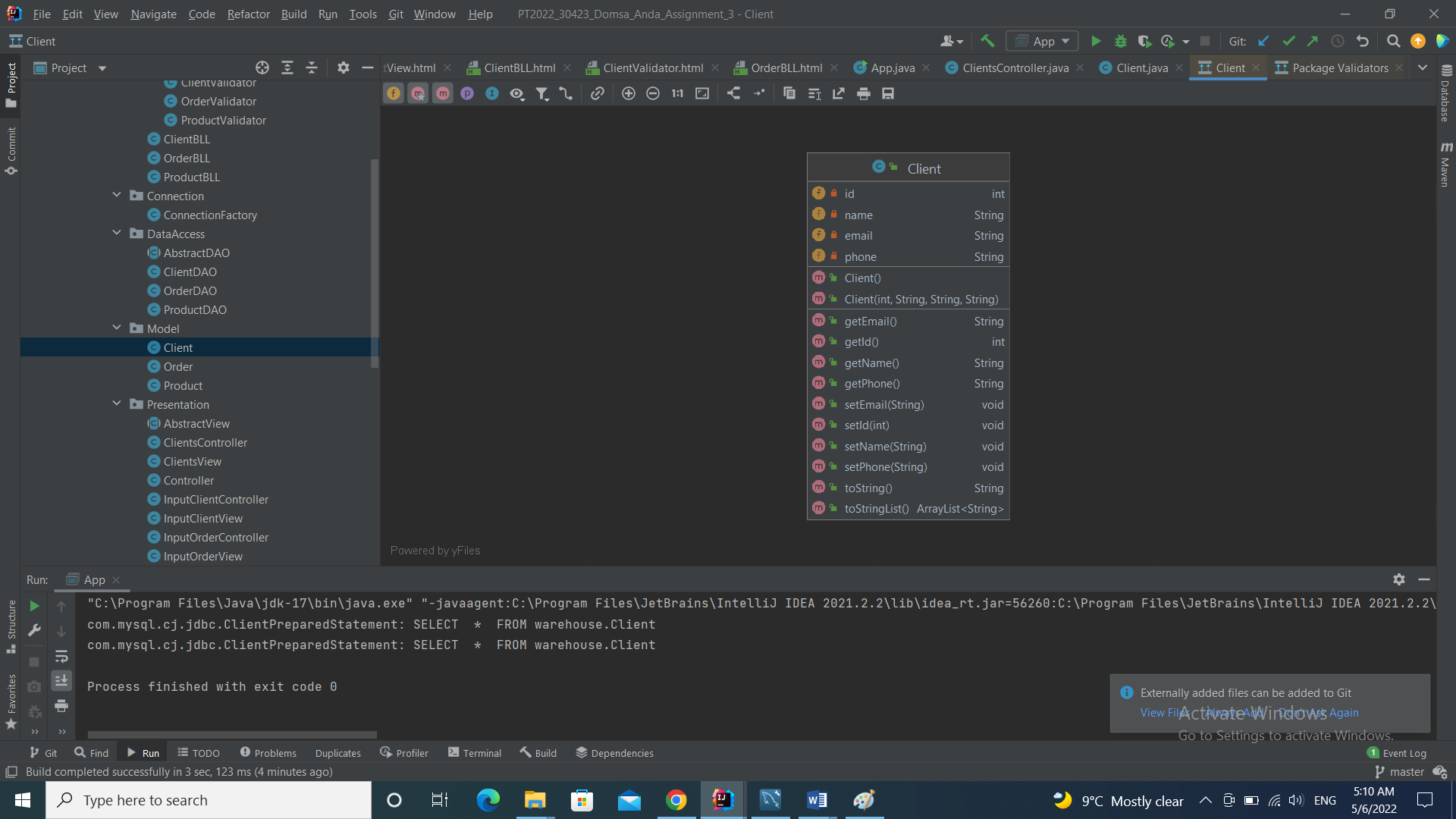


# Implementation

* Model package:

I started the implementation by working on the “Client”, ”Product” and “Order” classes, all three of them being found in the “Model” package.

*Client:*



The class that mirrors the table Client from the warehouse database. Its variables are the same as the columns of the table “Client” from the warehouse database.

Methods:

* getters + setters
* toString() method
* toStringList() : returns a list containing the client’s details

*Order:*

The class that mirrors the table Order from the warehouse database. Its variables are the same as the columns of the table “Order” from the warehouse database.

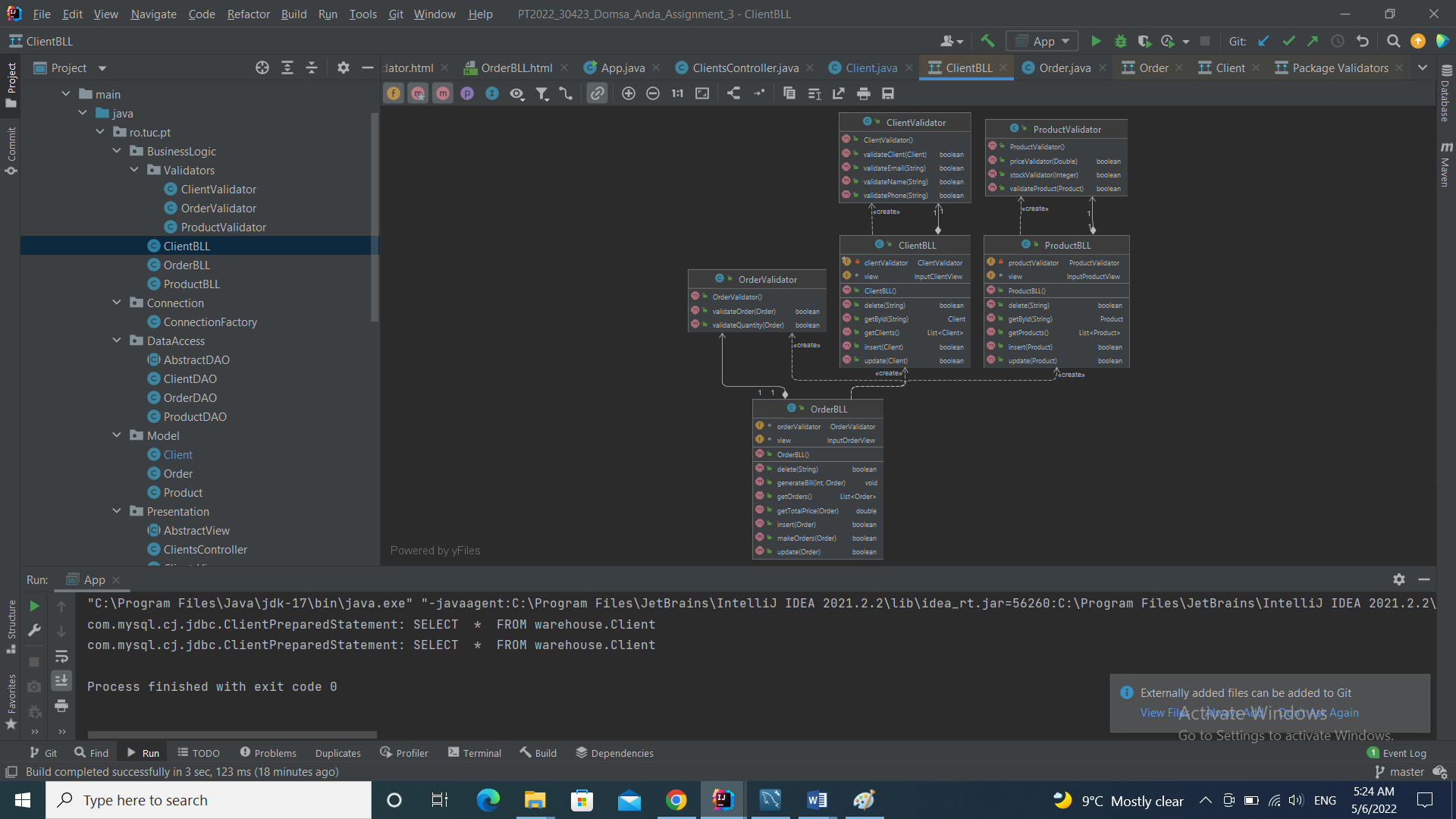
Methods:

* getters + setters
* toString() method
* toStringList() : returns a list containing the order’s details

*Product:*

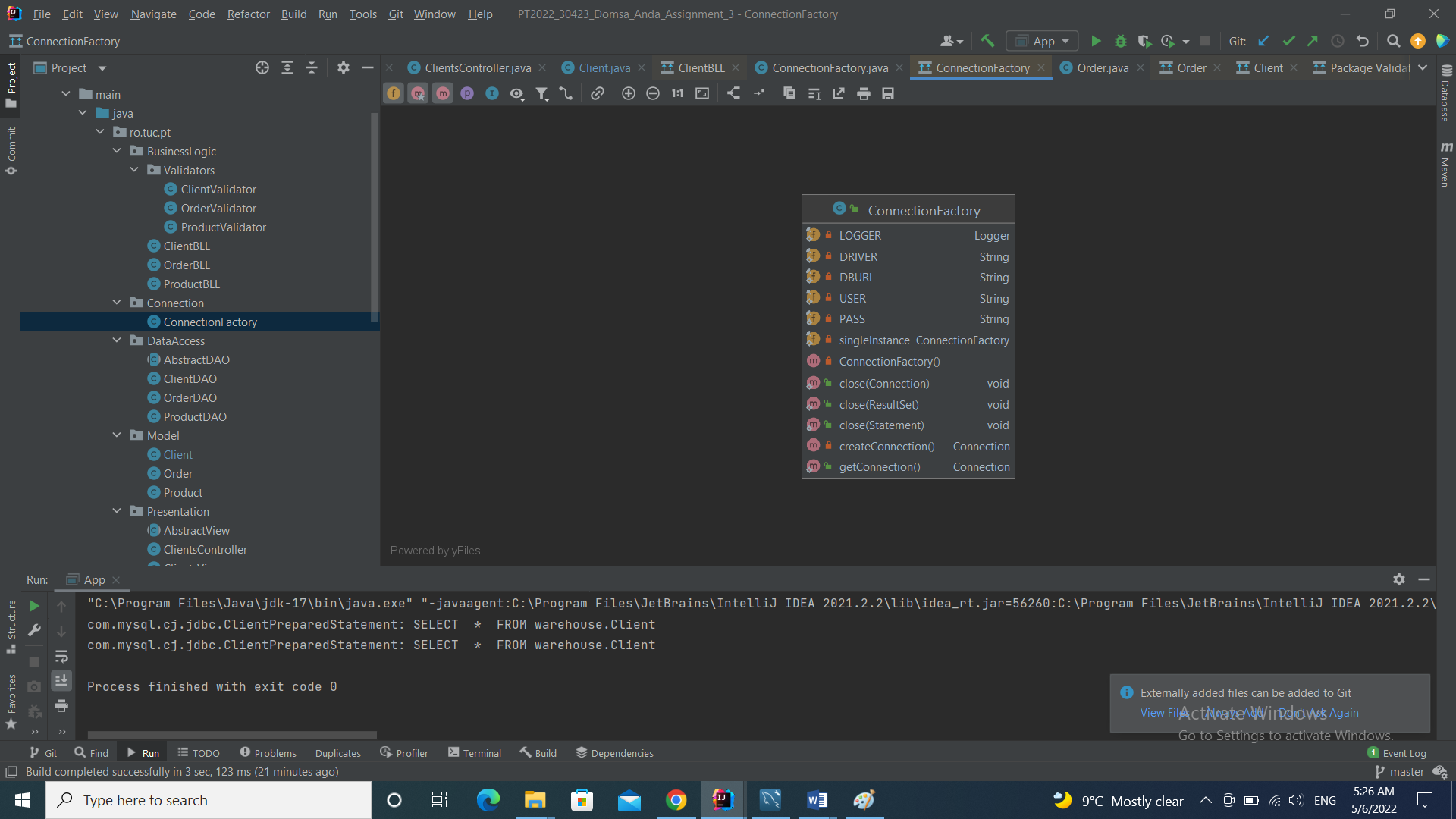
The class that mirrors the table Product from the warehouse database. Its variables are the same as the columns of the table “Product” from the warehouse database.

Methods:

* getters + setters
* toString() method
* toStringList() : returns a list containing the product’s details
* *BussinessLogic package:*
* 

I continued the implementation with the classes from the BussinessLogic package: the BLLs and the validators.

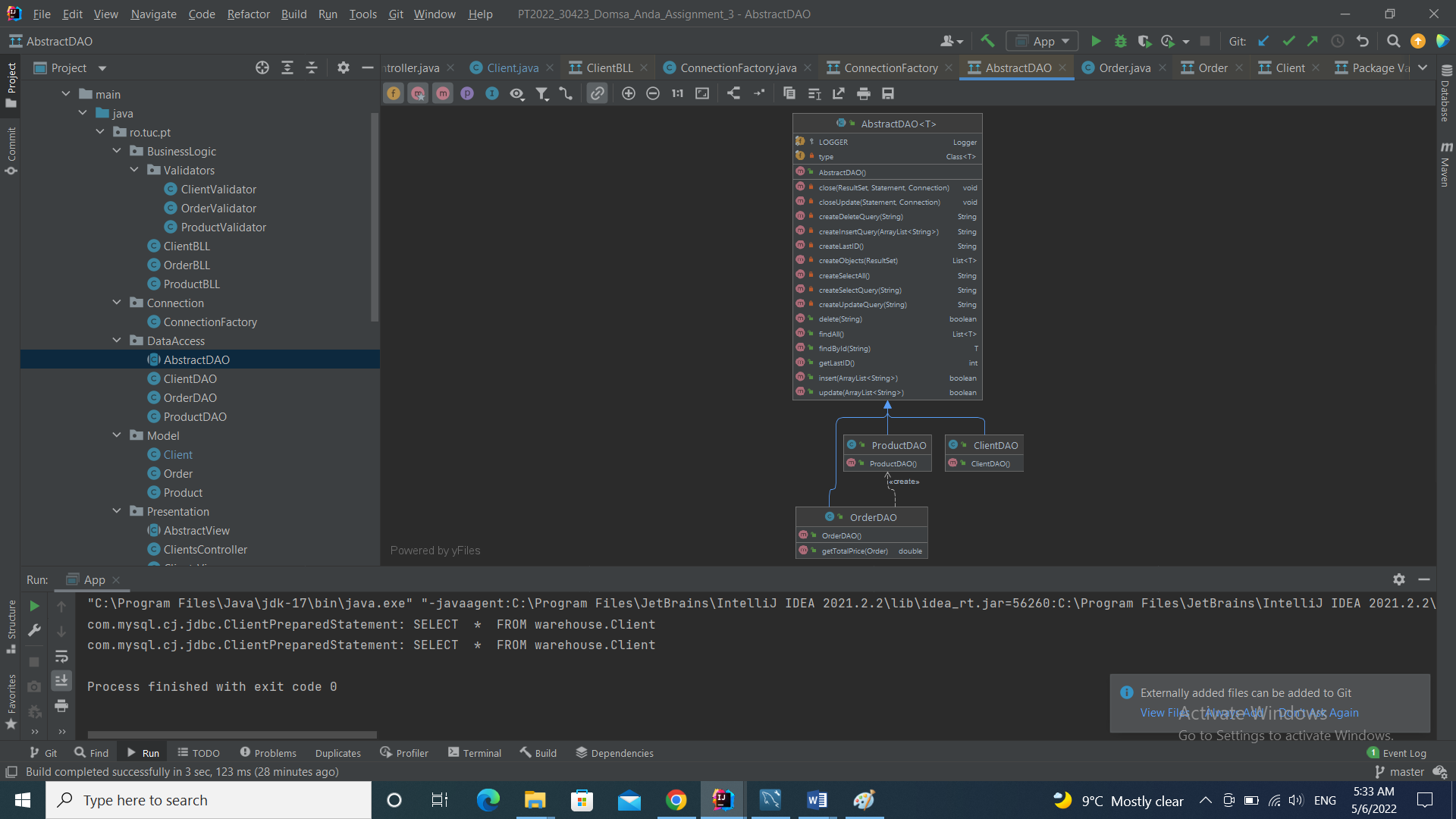
* *Connection package:*



After that, I made sure my database was connected so I implemented the class ConnectionFactory from the Connection package.

Methods:

* public void close(Connection);
* public void close(ResultSet);
* public void close(Statement);
* private Connection createConnection();
* public Connection getConnection();
* *DataAccess package:*

After that, I implemented the classes in the DataAccess package, which make the proper connection between the database and the application, starting with the abstract class “AbstractDAO”, and continuing with the other three: “ClientDAO”, “ProductDAO” and “OrderDAO”.

The “AbstractDAO” class implements all the methods needed for the other three DAOs, primarily the CRUD methods needed to access and alter the database. Also, it implements the methods that are creating the necessary queries that are used to access the data in the tables.

* *Presentation package:*

Then, all I had left to do was the user interface in the Presentation package.

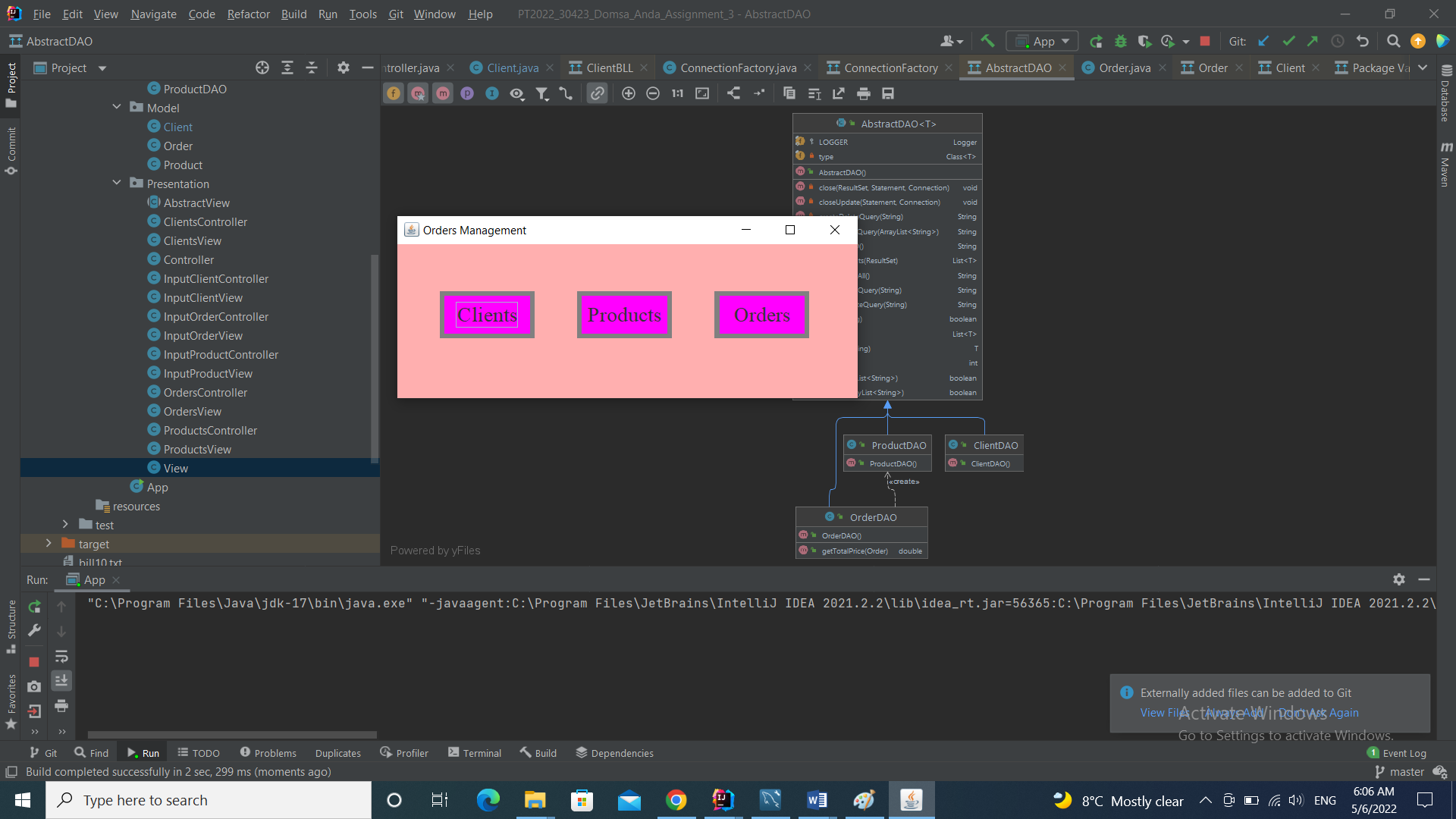
* *View:* The main view which implements the main menu of the application.
* *Controller:* The main controller of the application that connects the other parts of the program to the view.
* *ClientsView, ProductsView, OrdersView :* the three interfaces that extends the abstract class “*AbstractView*” and allow the user to work on each individual table.
* *ClientsController, ProductsController, OrdersController:* the three controllers that connect the data from the database to the tables in the user interfaces.
* *InputClientsView, InputProductsView, InputOrdersView, InputClientsController, InputProductController, InputOrdersController:*  the GUI classes that allows the user to input data or to edit the values in the tables.

The Application class is the last class I implemented and it starts the application opening the main menu.

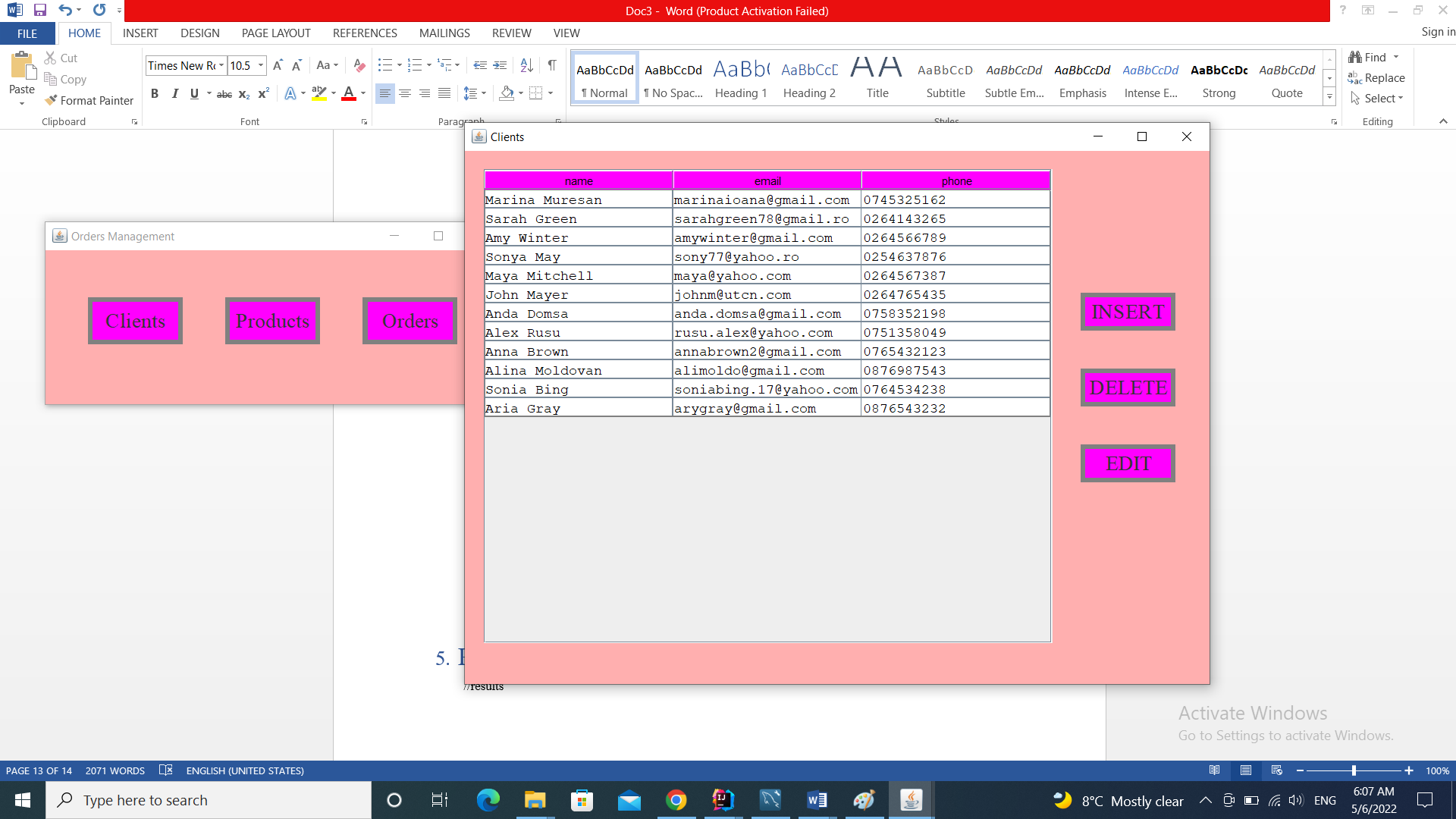
* *User Interface:*

The interface is a very user friendly one, because all the buttons with which the user interacts say very clearly what they are supposed to do.

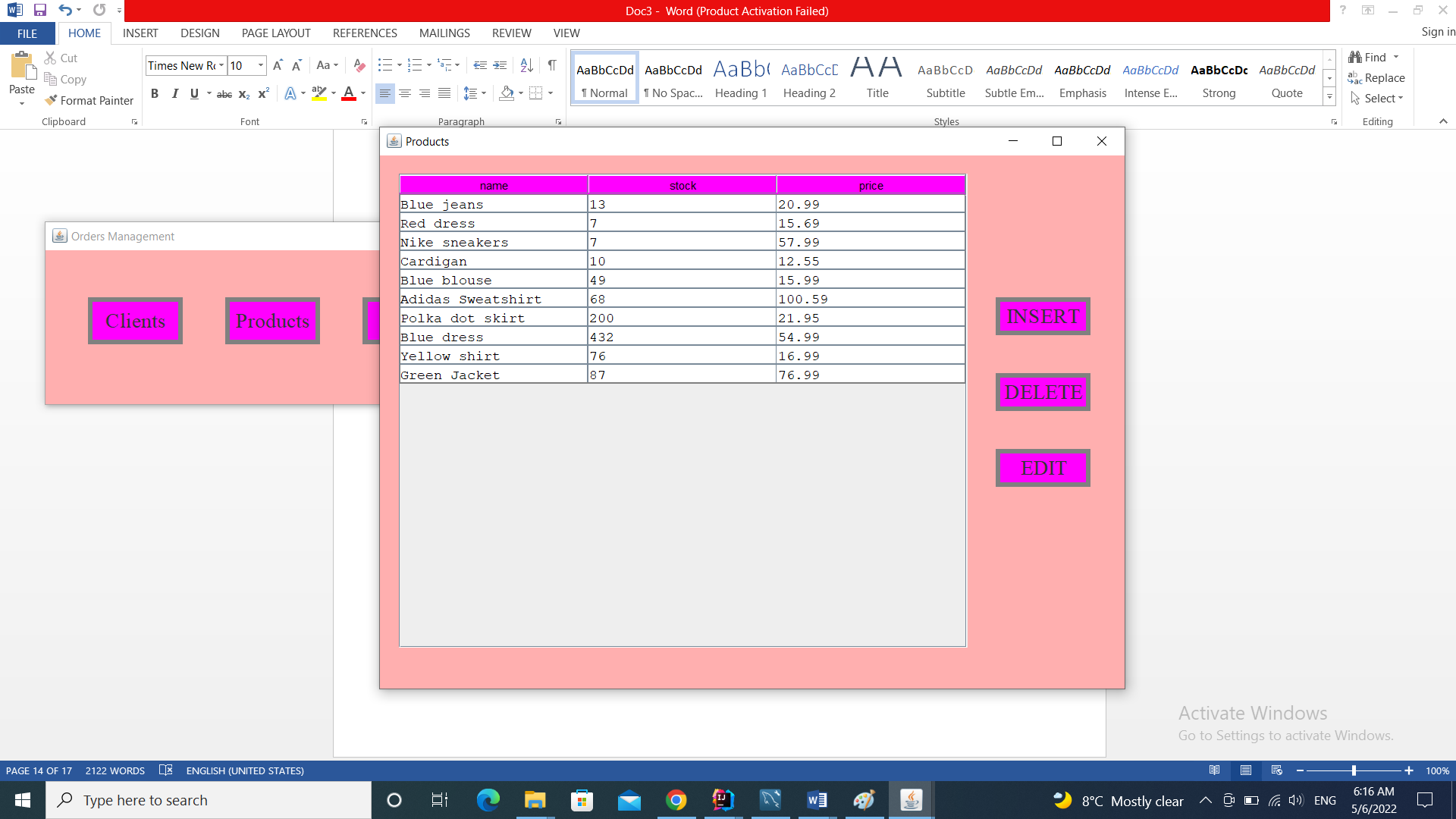
*Main menu:*



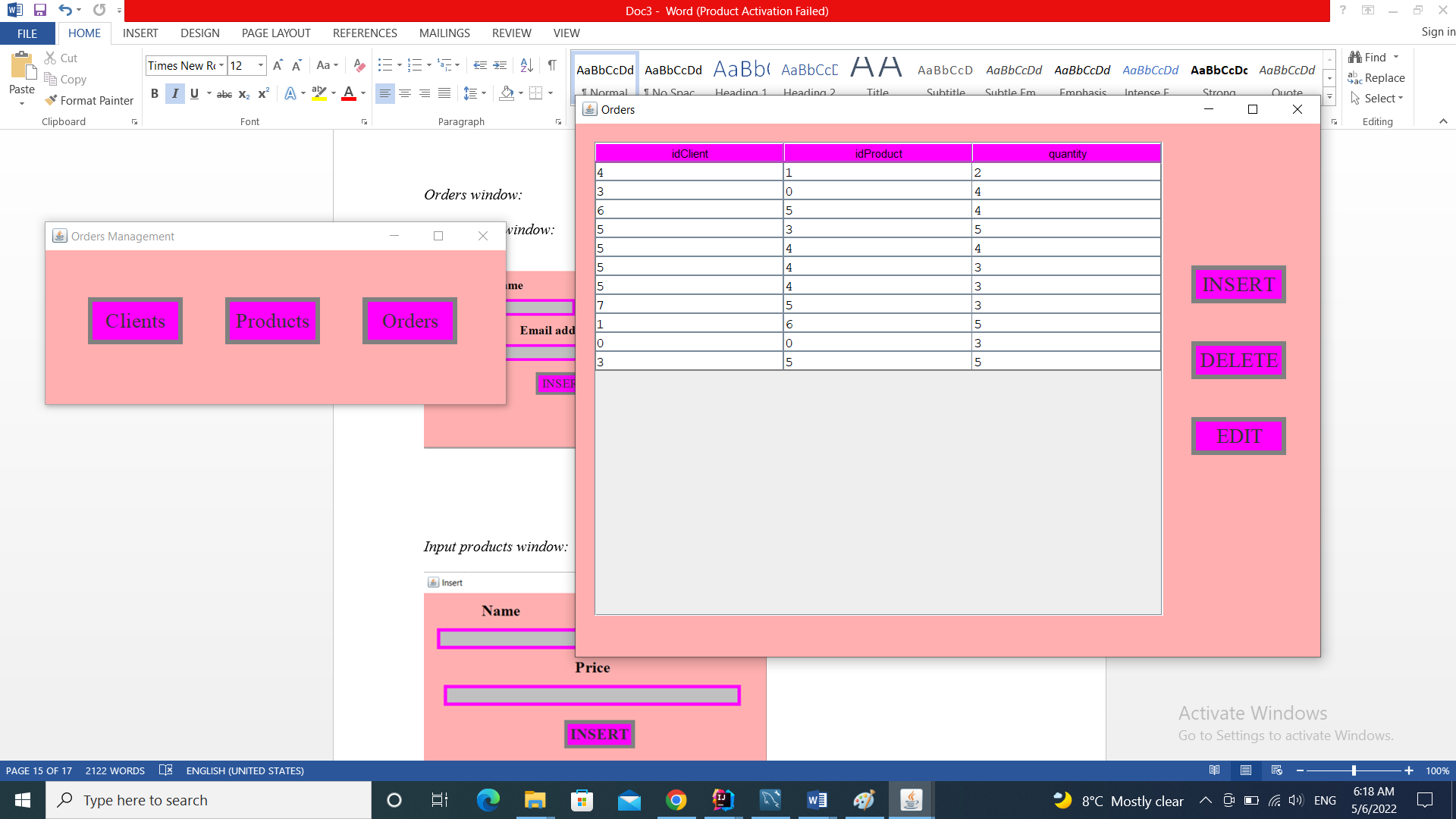
*Clients window:*



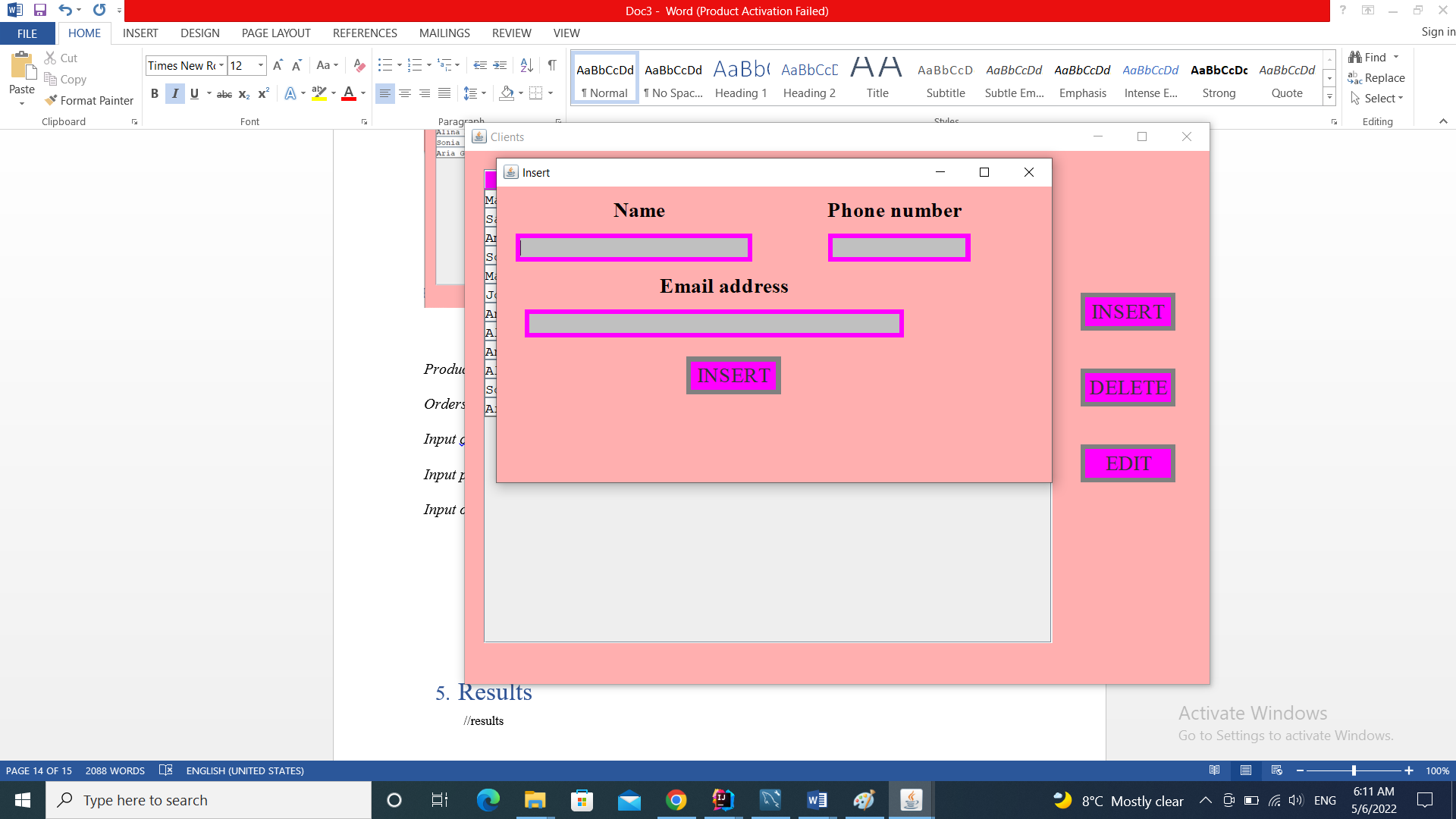
*Products window:*



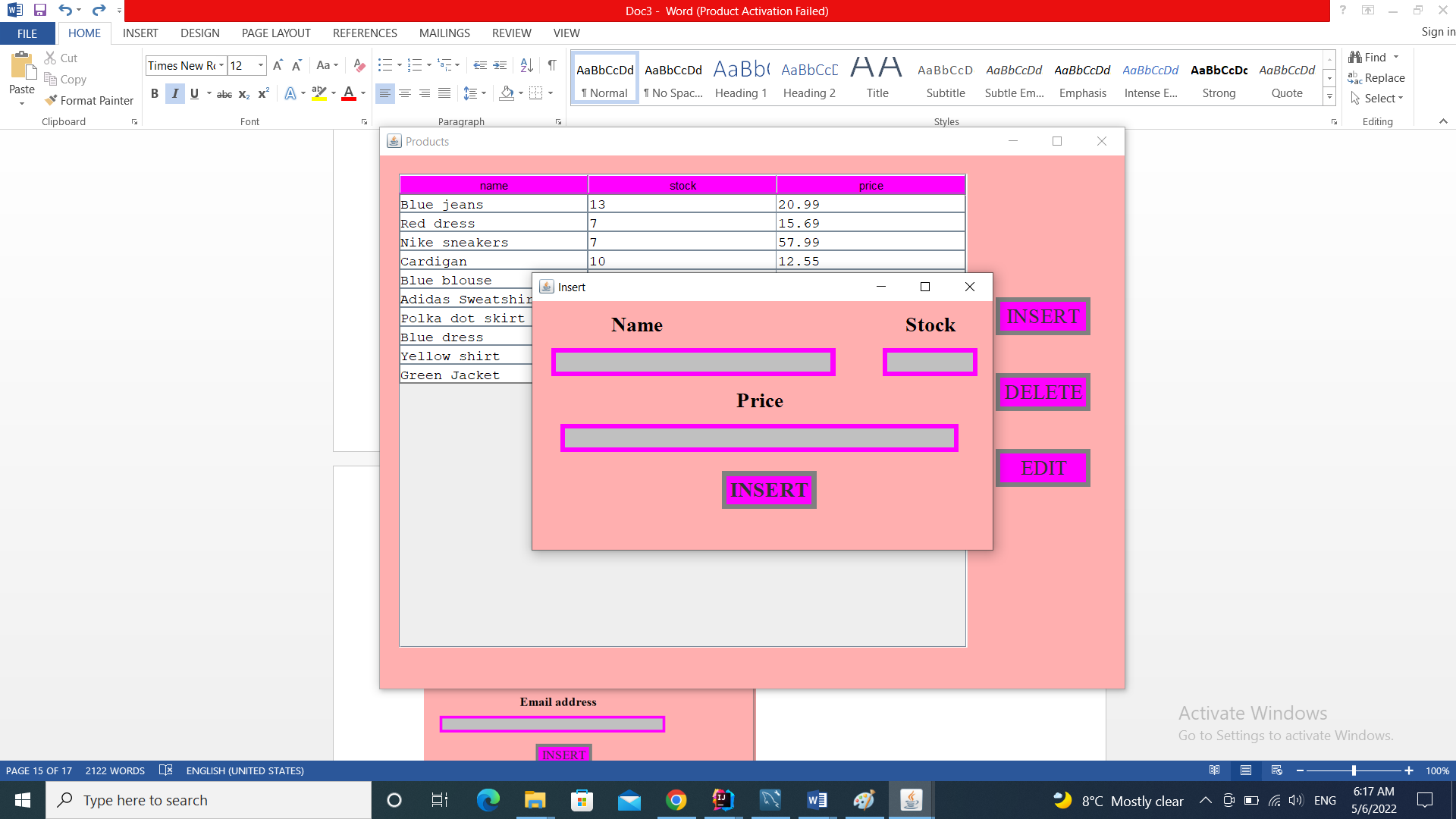
*Orders window:*



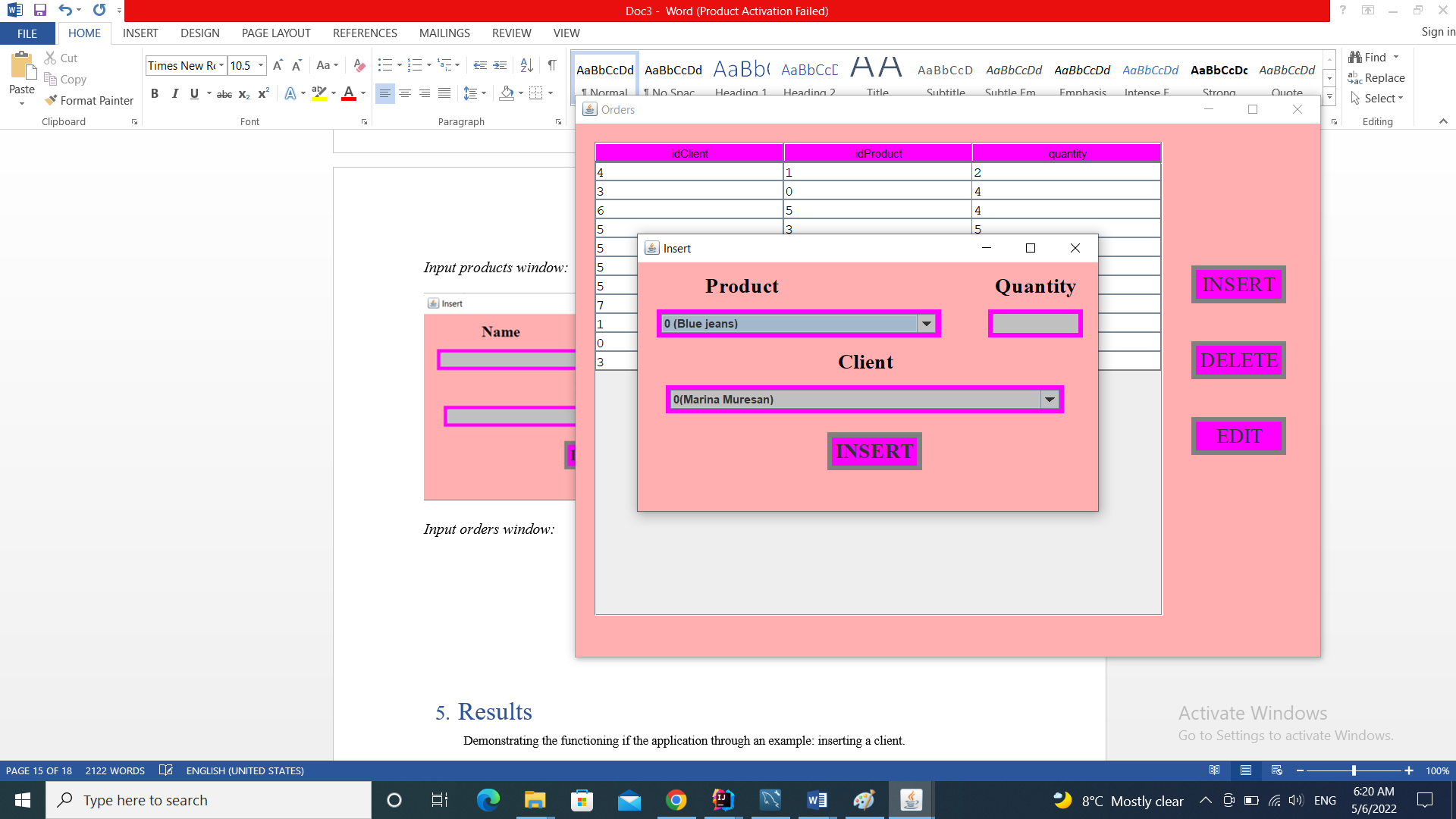
*Input clients window:*



*Input products window:*

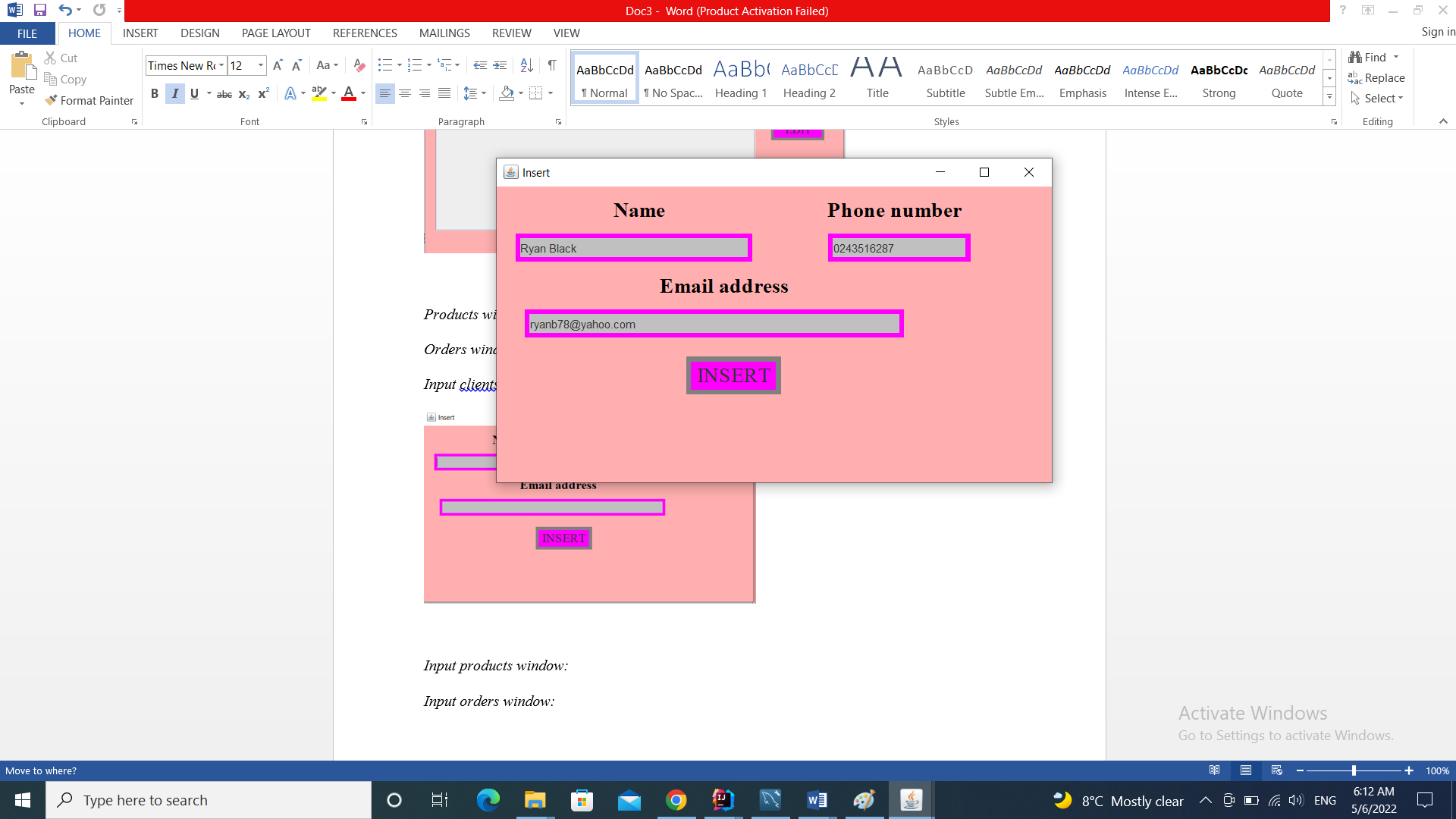


*Input orders window:*

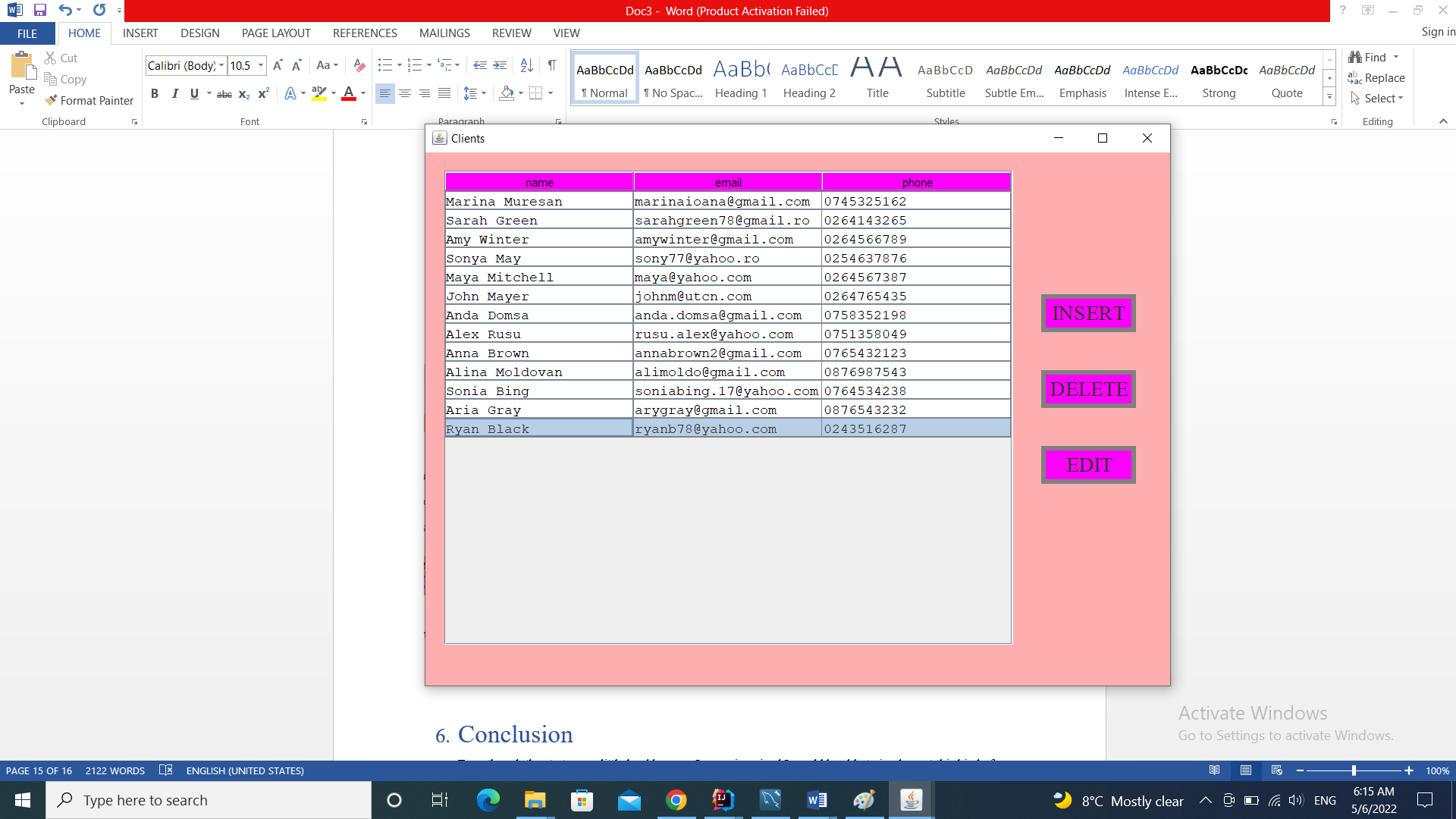


# Results

Demonstrating the functioning if the application through an example: inserting a client.



After I inserted the details and I clicked on the “INSERT” button, the client has been added on the last row of the table.



# Conclusion

Even though the start was a little hard because I never imagined I would be able to implement this kind of an application, after diving in it all started to make sense and towards the end working on it actually became kind of fun and relaxing and made me feel quite accomplished.

Besides renewing my knowledge about databases, the project helped me learn how to work better with OOP in Java. I also learned how to work with databases in an application and I had fun playing with dimensions and colors for the user interface.

# *Possibilities of further improvement:*

To improve the Orders Management Application in the future I could make it more dynamically so that the stock of the products decrement/increment automatically when adding/deleting an order so that you don’t have to reopen the window to see the changes in stock.

# Bibliography:

* <https://app.diagrams.net/>
* <https://dsrl.eu/courses/pt/materials/A3_Support_Presentation.pdf>
* <https://google.github.io/styleguide/javaguide.html>
* <https://sparxsystems.com/enterprise_architect_user_guide/14.0/model_domains/packagediagram.html>
* [www.stackoverflow.com](http://www.stackoverflow.com)
* [www.wikipedia.org](http://www.wikipedia.org)
* <https://www.draw.io/> (for diagrams)
* <https://www.baeldung.com/java-pdf-creation>