DOCUMENTATION

ASSIGNMENT *3*

STUDENT NAME: Hoza Violeta Maria

GROUP: 30424

CONTENTS

1. Assignment Objective

i) ***Main Objective***:

The primary goal of this project is to build and implement an order management application for customer orders at a warehouse. The user has several options, including adding, deleting, editing, and viewing a specific client or product, as well as creating an order. The program will use a database to hold client, product, and order information. Some basic information regarding customers, items, and orders is stored. Not all orders will be allowed; a filter will be developed to ensure that orders that do not meet the criteria are not accepted. This filter determines whether the warehouse has at least as many products as the type of product being ordered.

ii) Sub-objectives:

- Analyze the problem and identify requirements.
- Design the orders management application.
- Implement the orders management application.
- Create a bill file that contains information about each order.
- Use a layered architecture.
- Create generic methods.
- Test the orders management application.

2. Problem Analysis, Modeling, Scenarios, Use Cases

i) Functional requirements

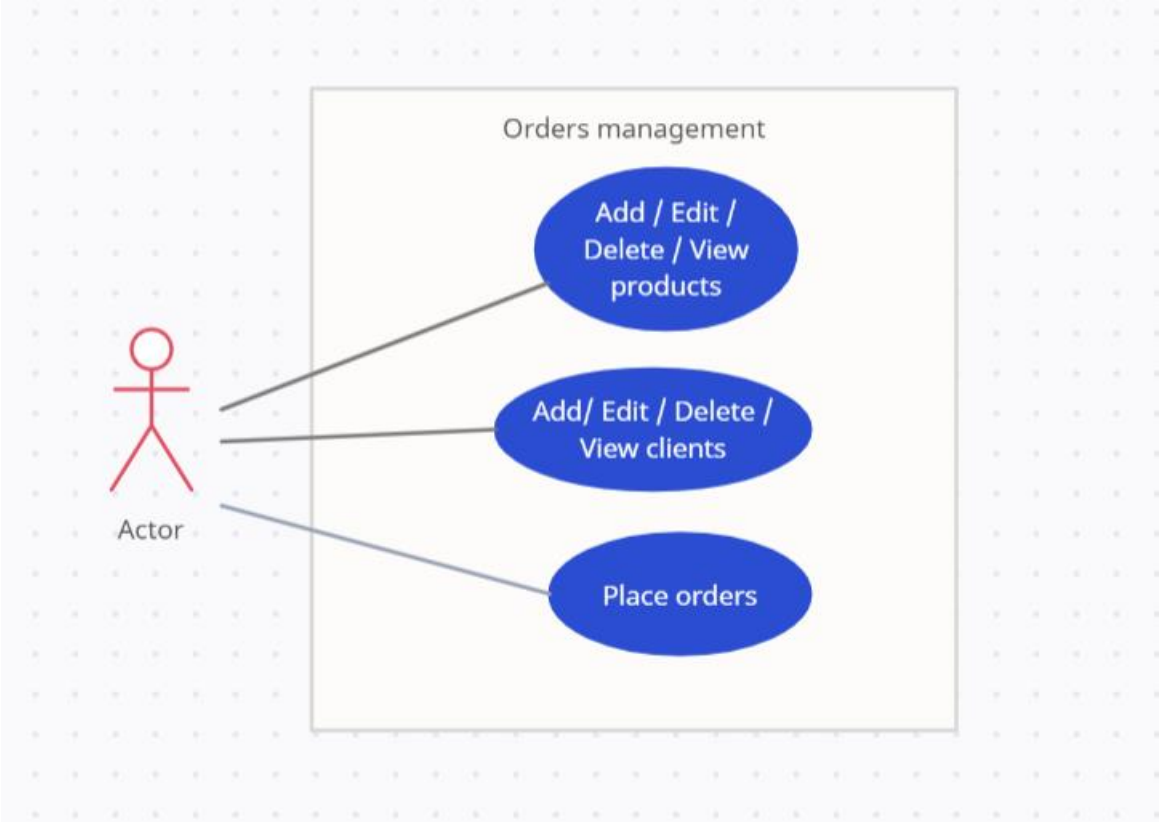
- Must have: A connection to a database which stores all the relevant information about each client, product, and order. A graphical user interface that allows the user to interact with the application and the database implicitly. CRUD operations on the database.
- Should have: Generic classes and methods that allow the use of the same methods on different types of objects, without duplicate code.
- Could have:
- Won't have: Login page for client/employee. Lack of ability for the client to perform only client operations makes the app useable only by the employees.

ii) Non-functional requirements:

- Ease of use: the user interface should be simple and intuitive, with labels that indicate what the application is expecting from the user.
- Compact design: the application's graphical interface should not have a greater size than necessary but should be big enough to display all the information that has to be shown.

iii) ***Use cases:*** The employee can choose between client or product operations or create an order.

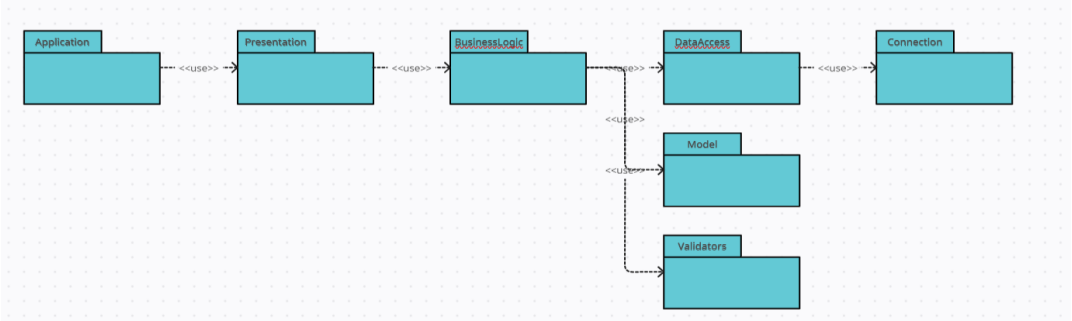
- For the first two options, the user can: add a new client / product, delete them from the database, edit or update them. When the option view object is chosen, a table with all the entries appears.
- For the create order operation, with the help of two JComboBox, one for clients and the other one for the products, the employee will have the ability to choose an option (a client/product from the list). Another field must be completed where the user is asked for the quantity the user wants to buy. The submit option will finish the order and save it into the database.



3. Design

- Package diagram:

The application is divided into several packages, each containing related classes. These packages include Application, Presentation, Model, BusinessLogic, Validators, Connection and DataAccess.



There are four layers in this architecture:

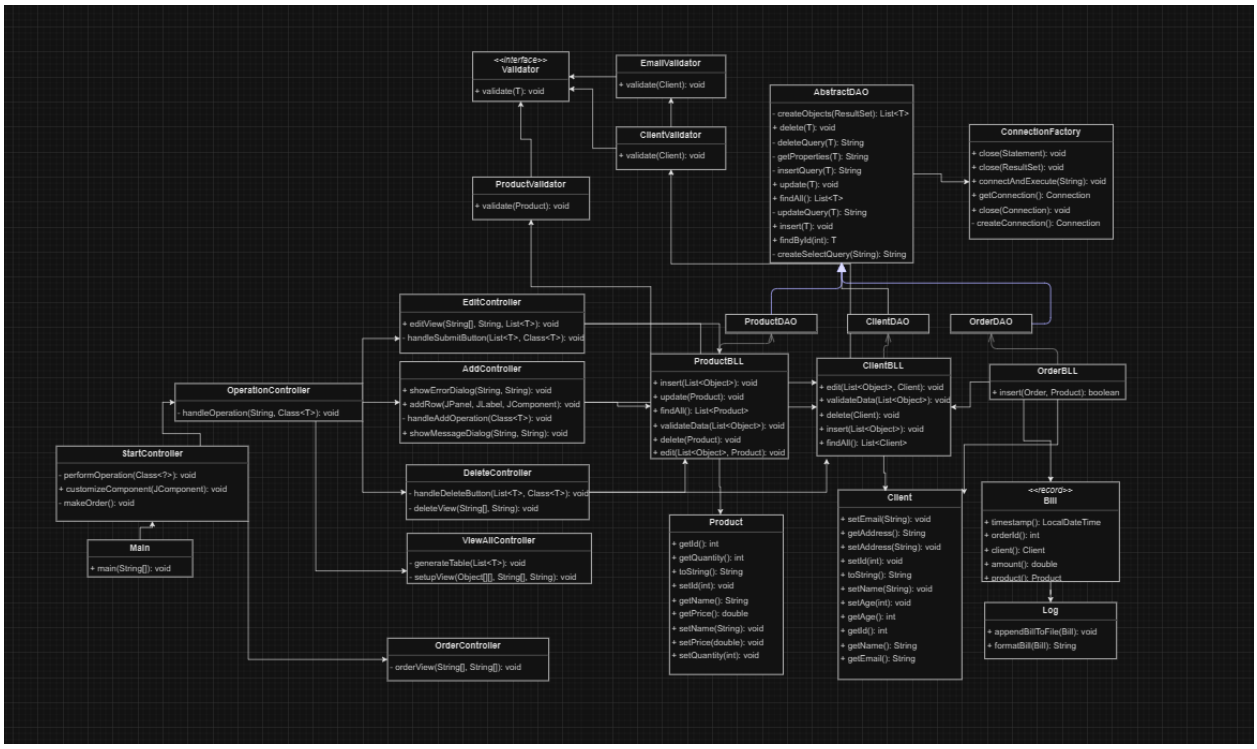
• The presentation layer - the application's user interface and communication layer, through which the end user interacts. Its primary aim is to display and collect information from the user. The current application receives the user's desired next step as well as the data input (if applicable). It also opens new windows with the information the user requested.

• Business logic - handles the business rules, calculations, and logic within an application which dictate how it behaves. It validates data and determines the next step based on the user's input. It also interacts with the database by calling the data access layer's methods to perform certain database operations.

• Data Access - contains methods for accessing the underlying database data. It implements CRUD operations and supports any class due to the generic implementation of the methods. Another important aspect is the connection between the application and the database. It uses a singleton class that makes sure that only one connection is made to the same database.

• Model - this is where all the data structures are implemented. There is a class for clients, products, and orders respectively.

- Class Diagram:



4. Classes Implementation

- Application: The Main class is used to start the application.
- Model

Client – This class represents a client entity in the system. It contains fields for the client's unique identifier, age, name, address, and email address. The class provides constructors to initialize a client object, as well as getter and setter methods for accessing and modifying the client's properties. Additionally, it overrides the toString() method to provide a string representation of the client object.

Product - This class represents a product entity in the system. It contains fields for the product's unique identifier, name, price, and quantity available in stock. The class provides constructors to initialize a product object, as well as getter and setter methods for accessing and modifying the product's properties. Additionally, it overrides the toString() method to provide a string representation of the product object.

Order - This class represents an order entity in the system. It contains fields for the order's unique identifier, client ID, product ID, and quantity. The class provides a constructor to initialize an order object with the given client ID, product ID, and quantity, and it generates a

random order ID within a specified range. Getter methods are provided to access the order's properties.

Bill - This record represents a bill generated for a purchase in the system. It includes fields for the order ID, client information, product information, total amount, and timestamp of the bill. It serves as a convenient and immutable way to store bill data.

Log - This class represents a utility for logging bill information to a file. It provides methods for appending bill information to a log file and formatting bill data into a string. The log file path is specified as a constant. The appendBillToFile method appends bill information to the log file, while the formatBill method formats bill data into a readable string format.

- Presentation

StartController - the initial window of the application where users can choose different operations to perform.

OrderController - represents the window for placing new orders in the application. It allows users to select a client, a product, and specify the quantity for the order.

OperationController - represents the window for performing CRUD (Create, Read, Update, Delete) operations on a specific type of object.

AddController - represents the window for adding new objects of a specified type. It provides fields for entering object details and a submit button to perform the addition operation.

DeleteController - represents the window for deleting objects of a specified type. It provides a combo box listing all objects of the specified type and a button to delete the selected object.

EditController - EditController class represents the window for editing objects of a specified type. It provides a combo box listing all objects of the specified type, text fields for editing object attributes, and a button to submit the changes.

ViewAllController - represents the window for viewing all entities of a specified type in a table format.

- BusinessLogic

ClientBLL - Business Logic Layer (BLL) class for performing operations on Client objects. This class serves as an intermediary between the Presentation layer and the DataAccess layer, handling tasks such as inserting, updating, and deleting clients from the database.

OrderBLL - Business Logic class for handling operations related to orders. This class encapsulates the logic for inserting new orders into the database and managing associated tasks such as creating bills and logging them. It interacts with the data access layer (OrderDAO) to

perform database operations and the presentation layer (AddController) to display messages and errors.

ProductBLL - Business Logic class for handling operations related to products. This class encapsulates functionalities for inserting, updating, deleting, and finding products in the database.

- Connection

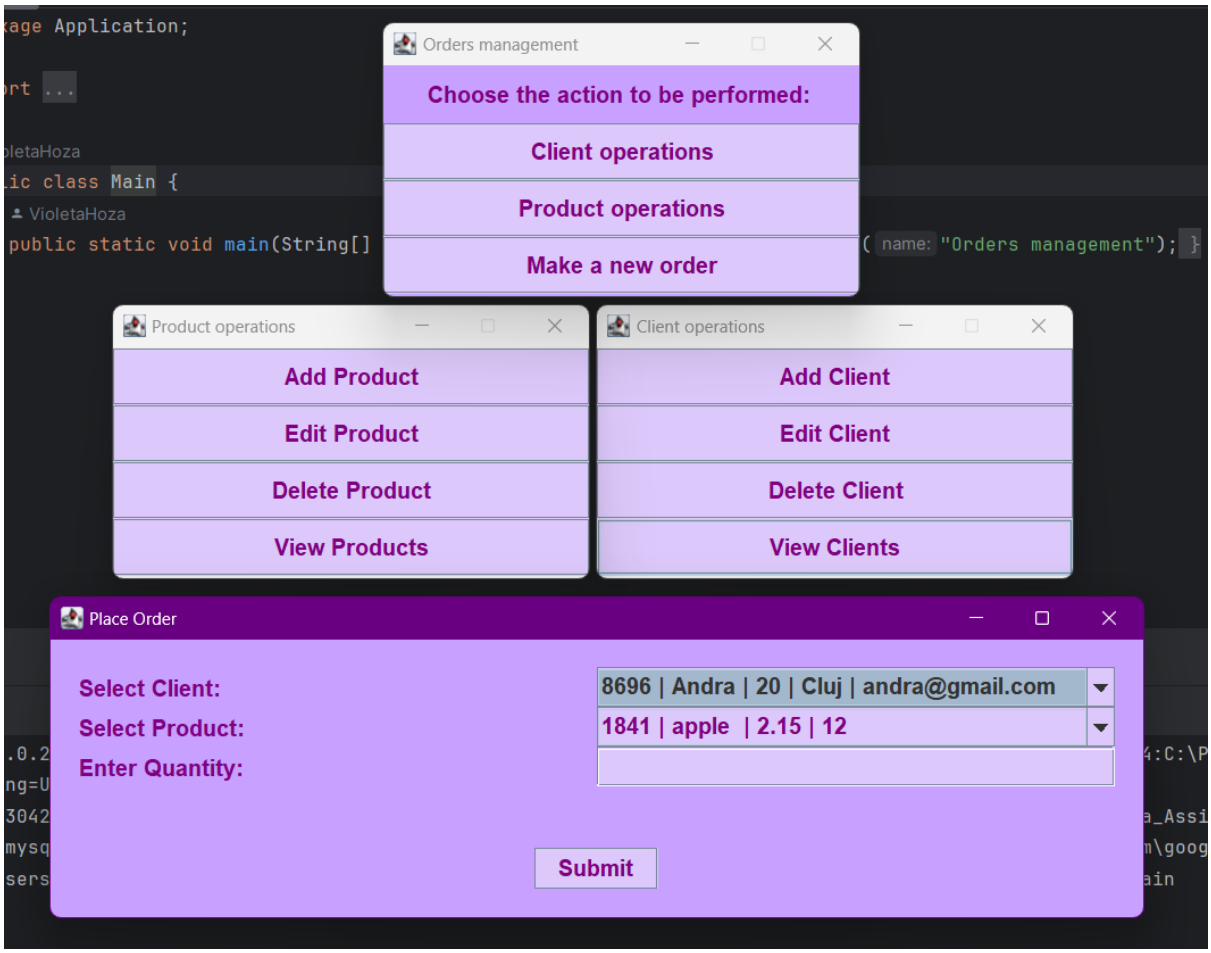
ConnectionFactory - This class manages the database connection for the application. It provides methods for obtaining a connection to the database, closing the connection, statement, and result set, and executing SQL queries. It uses the JDBC driver to establish the connection to the MySQL database specified by the DBURL constant. The class also handles exceptions related to database connectivity.

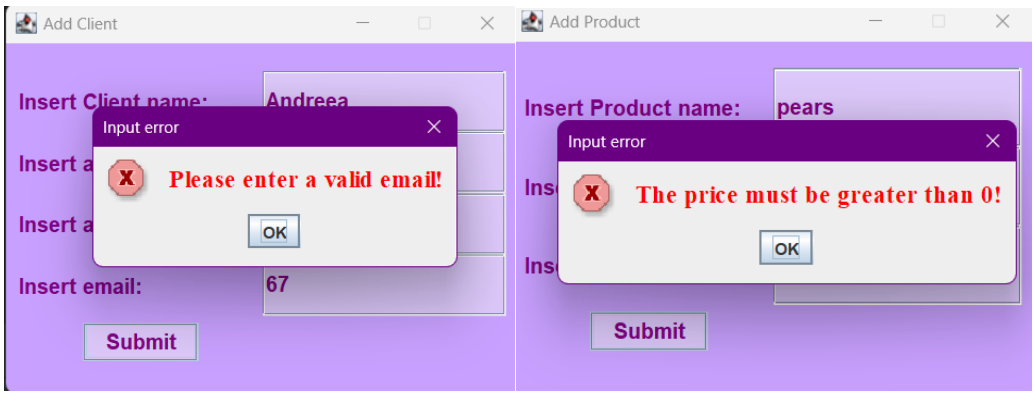
- DataAccess

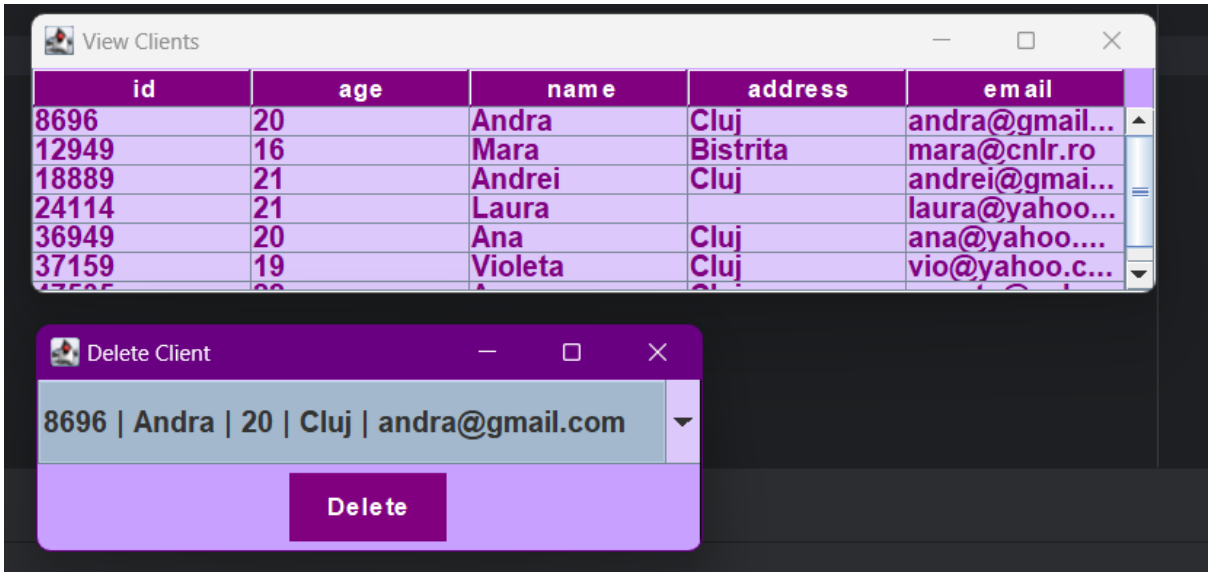
AbstractDAO – This class implements methods that allow CRUD operations on the database. This class contains generic methods that also use reflection techniques such that the same code of the same method is used to perform same operation on the database, but any class is accepted. Reflection techniques assure extractions of the values for each field of the given object and also the names and type of those fields. This technique brings reusability to the code since it can be used for any object type. The methods implemented in this class are used in the

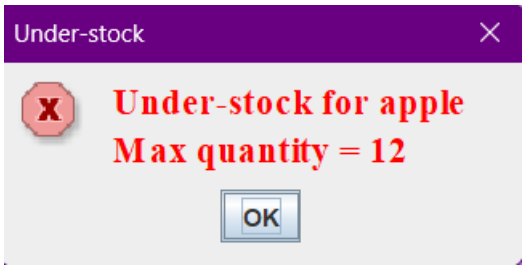
controller package to communicate with the database and perform the tasks requested by the user.

5. Results









6. Conclusions

This project introduced me to new Java programming language features, such as how to use reflection techniques, generic classes, and methods, as well as the layered design, which produces cleaner and more decoupled code. Additionally, I learned about MySQL Workbench and how to use it. All main and secondary requirements have been met. At the completion of the assignment, I believe that my Java programming skills have improved from the last assignment.

7. Bibliography

1. *www.tutorialspoint.com*
2. *Object-Oriented Programming - Lecture Slides of prof. Marius Joldoș*
3. *Programming Techniques - Lectures of prof. Cristina Pop*
4. *app.smartdraw.com – for UML package diagram*
5. *app.creately.com – for the use case diagram*
6. *www.stackoverflow.com*
7. *www.geeksforgeeks.com*
8. *app.diagrams.net – for UML class diagram*