DOCUMENTATION

ASSIGNMENT *ASSIGNMENT_3*

STUDENT NAME: Crăciunaș Victor
GROUP: 30425

CONTENTS

1. Assignment Objective

The goal of this project is to design and implement a computer application that manages customer orders for a warehouse. This involves creating a system that helps warehouse staff efficiently handle orders from start to finish. The application will allow employees to input new orders, update existing ones, and track order status all in one place, ensuring that the warehouse operates more smoothly and effectively. This will replace older, manual methods that are slower and more error-prone, making the warehouse's overall operations more efficient.

The following table lists the sub-objectives, each describing necessary steps to achieve the main objective, along with the sections where these will be addressed:

Sub-Objective	**Description**	**Section where it will be addressed**
Analyze the problem and identify requirements	The first step in our project involves a thorough analysis of the current problems and challenges faced by the warehouse in managing client orders. This will include identifying the specific needs that the application must meet and the issues it needs to resolve. Based on this analysis, we will list all the essential requirements for the application, such as features needed to streamline order processing, improve accuracy, and enhance data accessibility.	Problem addressed at section 2: Problem Analysis, Modeling, Scenarios ,Use cases
Design the simulation application	This phase involves creating a detailed blueprint of how the application will function and its structure. We will decide on the layout of the user interface, ensuring it is user-friendly and effective for	Problem addressed at section 3: Design

	warehouse staff. Additionally, we'll outline how the different components of the application—like the database, user interfaces, and processing logic—will interact with each other. This design will also include specifying the technology and tools to be used in developing the application. The aim is to create a comprehensive design that will serve as a roadmap for the developers to follow during the implementation stage.	
Implement the simulation application	Then, we build the application according to our plans. This involves writing code, using the right programming tools, and making sure everything works together as intended. The goal here is to turn our designs into a working app.	Problem addressed at section 4: Implementation
Test the simulation application	Lastly, we check that the app works correctly by testing it. We'll look for any mistakes and make sure it does what it's supposed to do under different conditions. This step ensures the app is ready to be used without problems.	Problem addressed at section 5: Results

2. Problem Analysis, Modeling, Scenarios, Use Cases

In the problem analysis phase, I carefully examined the requirements and identified the functional and non-functional ones for the computer application, aiming to give the most efficient way to handle the clients, products and the orders of a shop.

For functional requirements, the following functions describe what my simulation application can do:

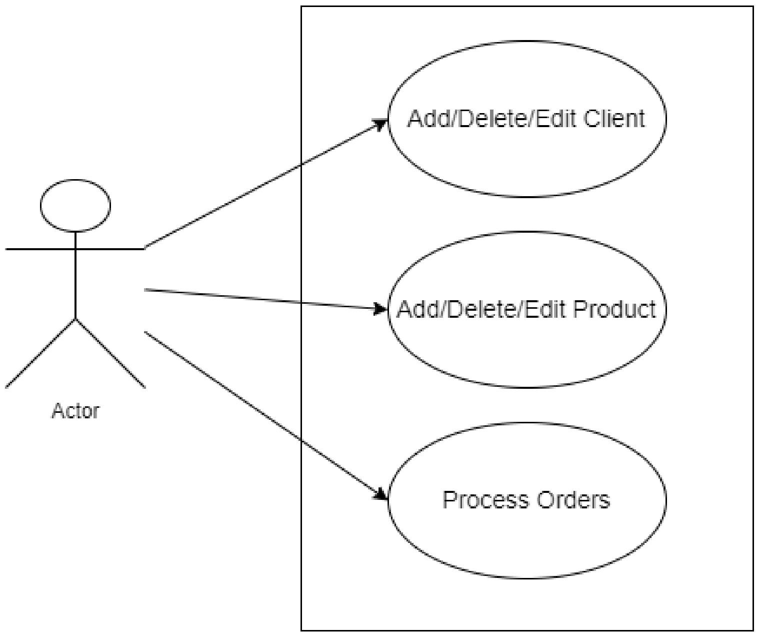
- Add a new client - The application should make it straightforward for employees to add new clients. This feature should include a detailed form where employees can enter essential information such as the client's name, address, phone number, email, and any specific notes or preferences. This helps ensure that all relevant client information is captured accurately and can be accessed whenever needed.
- Add a new product - Employees should have the capability to easily add new products to the system's inventory database. The interface should allow them to input the product name, provide a detailed description, set the price, and enter the quantity in stock. This feature should also include error checking to prevent incorrect data entry, such as typing letters where numbers should be.
- Edit Client Information - The application must include functionality for updating existing client information. This will allow employees to make necessary changes such as modifying contact details, updating addresses, or changing noted preferences. It's important that this process is user-friendly to encourage timely and accurate updates to client profiles.
- Edit product information - There should be an option for employees to update product information as needed. This could include changing the pricing, updating the product description to reflect any new features, or adjusting the stock levels based on recent inventory checks. Changes should be reflected in real-time across the system to ensure all employees have the latest information.
- View and Manage Inventory - The application should provide a comprehensive tool for viewing and managing the warehouse inventory. This feature should include a dashboard that displays current stock levels, alerts for items that are running low, and quick options to update inventory quantities.
- Process Orders: It is essential for the application to streamline the process of creating and managing orders. Employees should be able to select products, choose a client, and specify quantities using a simple interface. The system should automatically check the availability of products and notify the user if the quantity requested exceeds the stock level.
- Delete Client or Product - The application should also allow for the deletion of client or product records when necessary. This feature should include safeguards to prevent accidental deletions and require confirmation before removing any data.

And as for the non-functional functions:

•Intuitive User Interface: The system is user-friendly, with a well-organized interface that's easy to navigate. Users will be able to perform all functions intuitively, without the need for in-depth tutorials or guidance.

•System Performance: The application will be responsive, with minimal lag between user actions and system responses, even when handling a large number of simultaneous clients in the simulation.

For the use case, I'm going to setup a simulation below:



Use case: add product;

Primary Actor: employee;

Main Success Scenario:

1. The employee selects the option to add a new product
2. The application will display a form in which the product details should be inserted;
3. The employee inserts the the name of the product, its price and price and its current stock;
4. The employee clicks on the "Add" button;

5. The application stores the product into the database and displays an aknowledge message

Alternative Sequence: Invalid values for the product's data

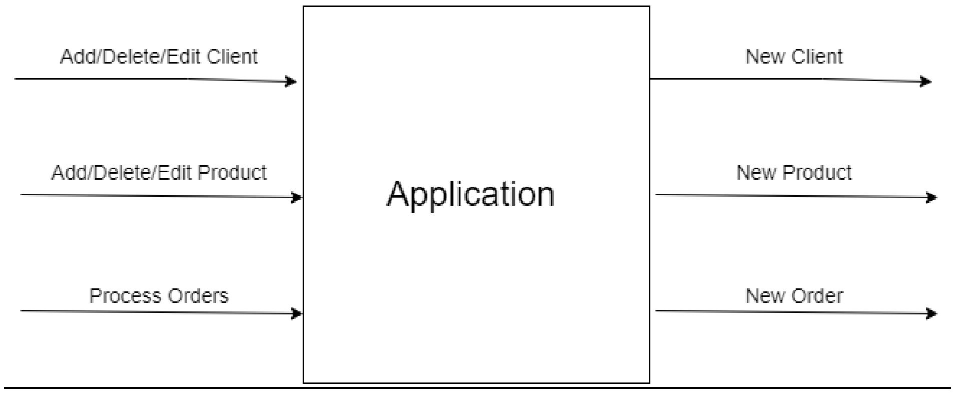
-the user inserts a negative value for the stock of the product

-the application displays an error message and requests the user to insert a valid stock

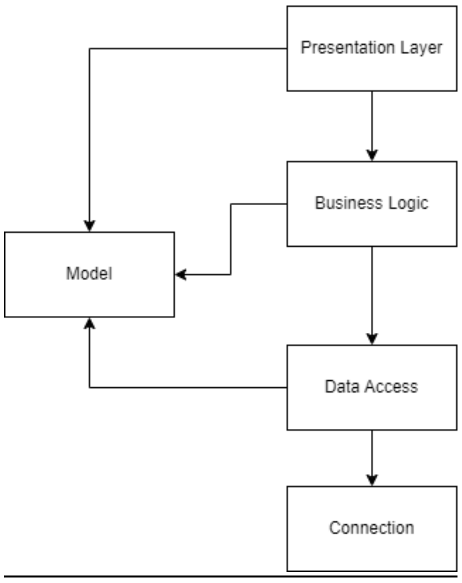
-the scenario returns to step 3

3. Design

Overall Design:

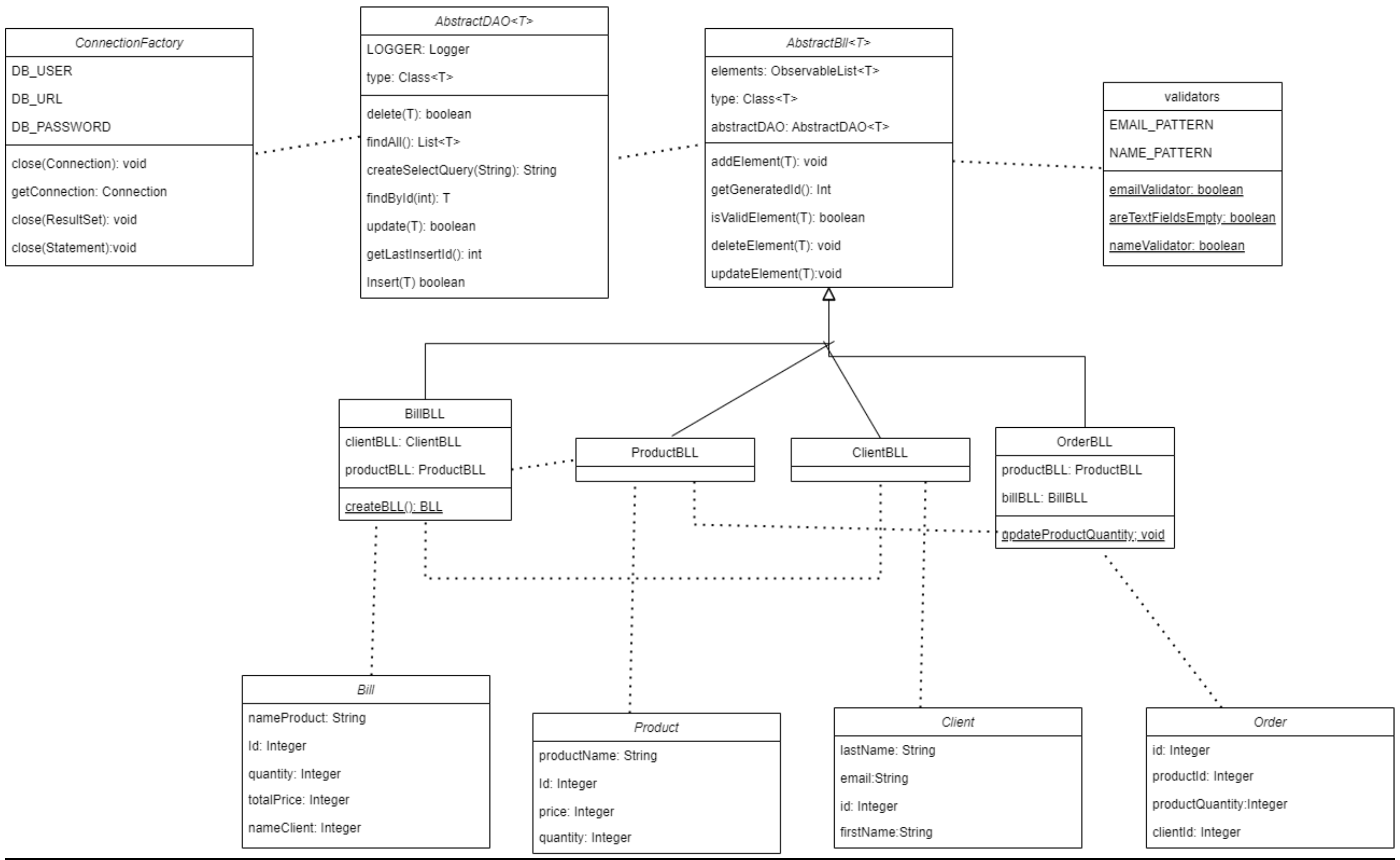


Overall Design:



- Model: the package where we keep the models from the database(each table in the data base is a model class in this package)
- Presentation Layer: the package which contains the controllers for each view. The GUI package
- Business Logic: the package which contains the logic of the app
- Data Access: the package for classes which are related to queries to data base
- Connection: Here we make the connection for the data base

Class Diagram



Data structuresused:

- ObservableList: I use ObservableList to keep the models in order to see make the change on the UI in real time;
- List: I also use List to retrieve the objects from the data base

The following should be presented: OOP design of the application, UML package and class diagrams, used data structures, defined interfaces and used algorithms (if it is the case).

4. Implementation

Models:

Client Model: This model is crucial for storing and managing data about individuals or companies that do business with the warehouse. It records vital information such as the client's first and last name, email, and potentially other contact details. This allows the system to maintain a comprehensive client database, essential for customer relationship management and targeted communications .

Product Model: Central to inventory management, the Product model keeps track of all items available in the warehouse. It includes detailed descriptions, pricing, and stock levels for each product. This model is fundamental for processing transactions as it ensures that product availability and details are up to date, which helps in preventing stock shortages and facilitating efficient order fulfillment .

Order Model: The Order model links the Client and Product models, detailing which products are being ordered by which clients, in what quantities, and at what price. It tracks the lifecycle of each order from creation to delivery, including any updates or modifications. This model is essential for ensuring orders are processed accurately and clients receive their goods on time. It also plays a role in inventory control, as it adjusts stock levels based on order quantities .

Bill Model: After an order is processed, the Bill model generates an invoice that details the transaction, including the client involved, the products sold, the quantities, and the total price. This model is designed to be immutable, meaning once a bill is created, it cannot be altered, ensuring the accuracy and integrity of financial records. Bills are stored separately in a log table for security and auditing purposes, providing a reliable financial history of transactions .

The AbstractDAO class in the Data_Access package is a generic class designed to work with different types of data from the database. It simplifies how the application interacts with the database by handling common actions like adding, updating, reading, and deleting data.

Key Features:

-Creating SQL Queries: This class can automatically create the right SQL (a type of database language) queries needed to get or change data based on the type of data it's dealing with. It does this using Java's reflection feature, which allows it to examine and modify the properties of classes and objects.

-Handling Data: The class offers functions to insert new data, update existing data, remove data, and find specific data by its ID. These operations use prepared statements, which are a way of writing SQL queries that are safer and often run faster.

-Creating Objects from Data: When the class retrieves data from the database, it uses reflection to turn this data into objects of the type it manages. This means creating new instances of a class and filling them with data from the database.

The ConnectionFactory class in the connection package handles all the necessary actions to manage database connections for the application. It provides methods to open and close connections, ensuring the application can communicate with the database efficiently.

Key Features:

-Get Connection: This function establishes a connection to the database using specific settings like the database URL, user, and password. It helps the application talk to the database safely and quickly.

-Close Connection: After the application is done using the database, this method ensures that the connection is closed properly. This is important to free up resources and avoid potential issues with too many open connections.

-Close Statement and Result Set: The class also has methods to close SQL statements and result sets. Closing these properly after use is crucial to prevent memory leaks and ensure the database performs well.

The AbstractBll class in the Business_Logic package is designed to manage the business logic associated with database entities using an AbstractDAO for database operations. It serves as a foundation for managing different types of data and includes various methods to manipulate this data efficiently.

Key Features:

-Initialization and Data Loading: When an instance of AbstractBll is created, it initializes the data access object (DAO) for the specified type T and loads all current records from the database into an observable list. This list is useful for keeping the UI in sync with the data.

-Data Manipulation: This class provides methods to add, delete, and update elements in both the observable list and the database. This ensures that changes are reflected immediately in the application's user interface.

-Data Validation: It includes an abstract method, isValidElement(T t), which should be implemented to define custom validation rules for the elements managed by the class. This helps in maintaining data integrity.

-ID Management: The getGenerated1Id method retrieves the most recent ID generated for an element, useful in scenarios where an ID is assigned automatically by the database.

From this AbstractBll class, 4 subclasses are inherited:

- -BillBLL
- -ClientBLL
- -OrderBLL
- -ProductBLL

These classes have specific method in order to manipulate their type of model.

he GenericController class in the Presentation.controllers package is an abstract class that serves as a foundation for all controllers managing different types of entities in your application. It integrates with the JavaFX framework to handle user interactions efficiently, providing core functionalities like setting up table views, and managing the addition, deletion, and editing of items through the user interface.

Key Features:

-Table Setup and Data Binding: Upon initialization, the class configures table columns and binds items from the AbstractBll class to the table view, enabling real-time data display and updates.

-Edit Handling: It enables the double-click on table rows to edit entries directly in the table, enhancing user interaction and data management efficiency.

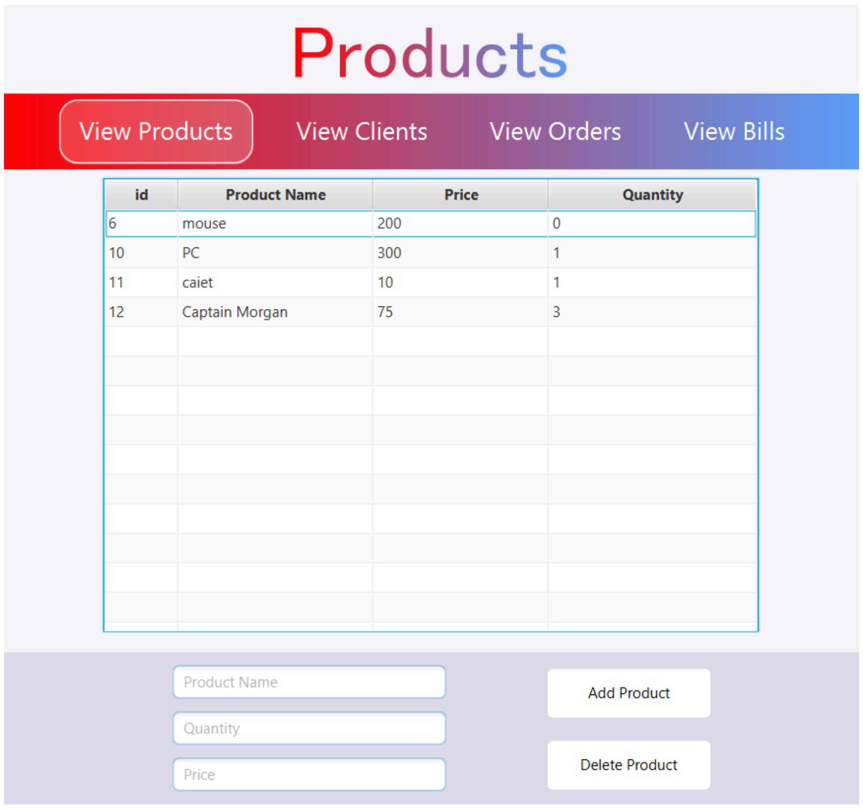
-Adding Items: The class provides functionality to validate and add new items to the database through the business logic layer. It includes checks to ensure inputs are not empty and the data is valid before addition.

-Deleting Items: Users can select items from the table and delete them. The class includes confirmation prompts to prevent accidental deletions, ensuring user actions are deliberate.

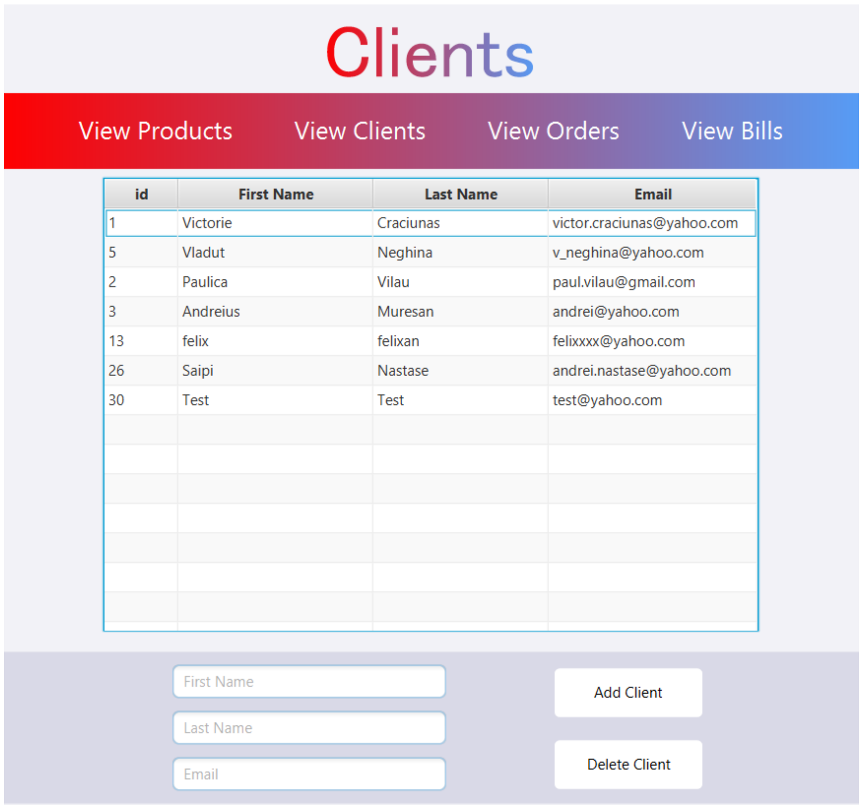
-Alerts and Notifications: It uses alerts to notify users about errors or confirmations, improving the application's usability by providing feedback on their actions.

-Navigation: The class contains methods to navigate between different views of the application, like clients, products, orders, and bills views, supporting a seamless user experience across various parts of the application.

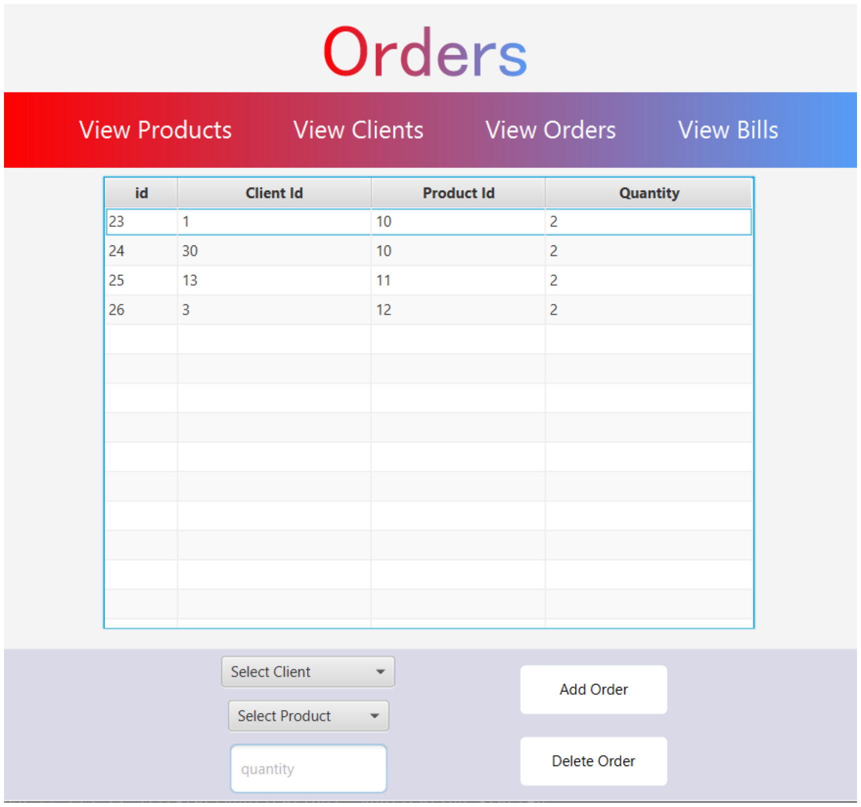
5. Results



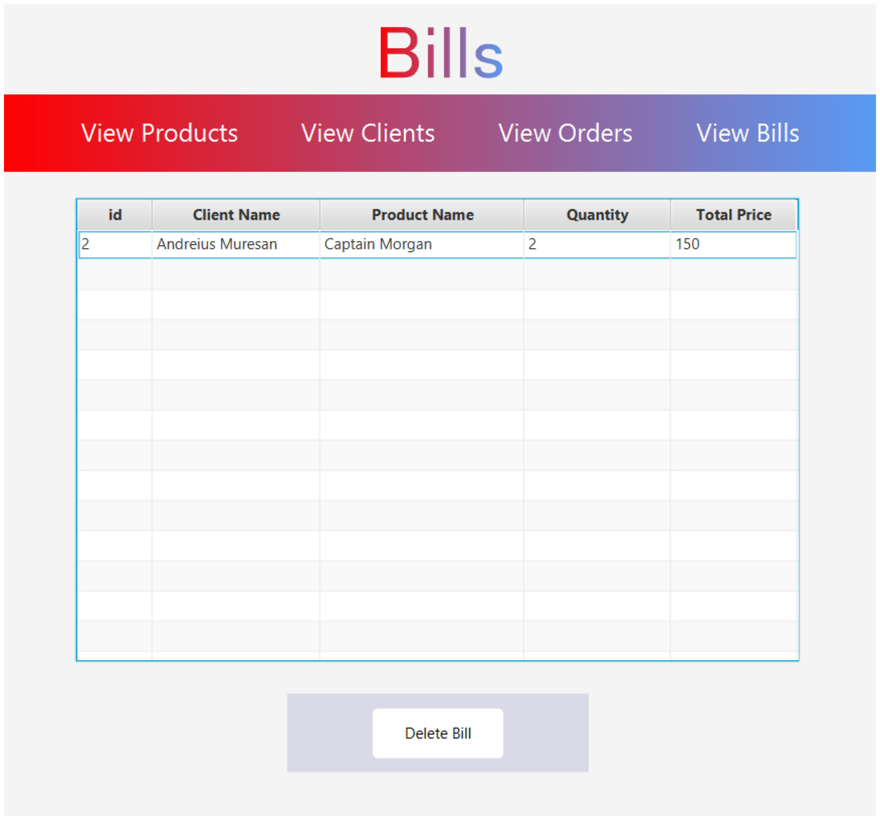
id	Product Name	Price	Quantity
6	mouse	200	0
10	PC	300	1
11	caiet	10	1
12	Captain Morgan	75	3



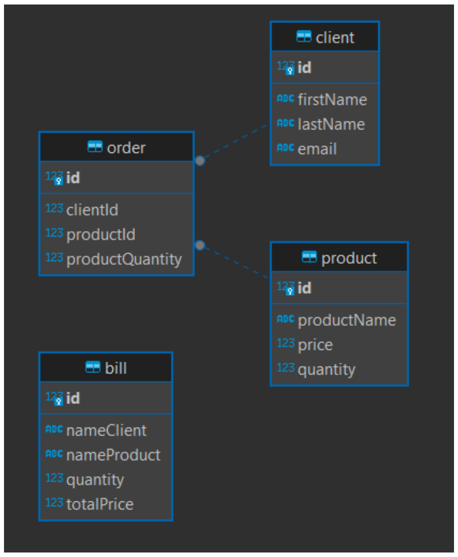
id	First Name	Last Name	Email
1	Victorie	Craciunas	victor.craciunas@yahoo.com
5	Vladut	Neghina	v_neghina@yahoo.com
2	Paulica	Vilau	paul.vilau@gmail.com
3	Andreius	Muresan	andrei@yahoo.com
13	felix	felixan	felixxxx@yahoo.com
26	Saipi	Nastase	andrei.nastase@yahoo.com
30	Test	Test	test@yahoo.com



id	Client Id	Product Id	Quantity
23	1	10	2
24	30	10	2
25	13	11	2
26	3	12	2



id	Client Name	Product Name	Quantity	Total Price
2	Andreius Muresan	Captain Morgan	2	150



6. Conclusions

In conclusion, this project successfully designed and implemented a robust application for managing client orders in a warehouse, integrating various functional components such as data access, business logic, and presentation layers. By leveraging Java technologies and design patterns, the application efficiently handles operations from database management to user interactions, streamlining processes and improving usability. This comprehensive system not only enhances operational efficiency but also supports scalability and maintainability, setting a strong foundation for future enhancements and growth.

7. Bibliography

https://dsrl.eu/courses/pt/materials/PT2024_A3_S1.pdf

https://dsrl.eu/courses/pt/materials/PT2024_A3_S2.pdf