DOCUMENTATION

ASSIGNMENT *ASSIGNMENT_3*

STUDENT NAME: ALEXANDRU-ANDREI AVRAM
GROUP: 30422

CONTENTS

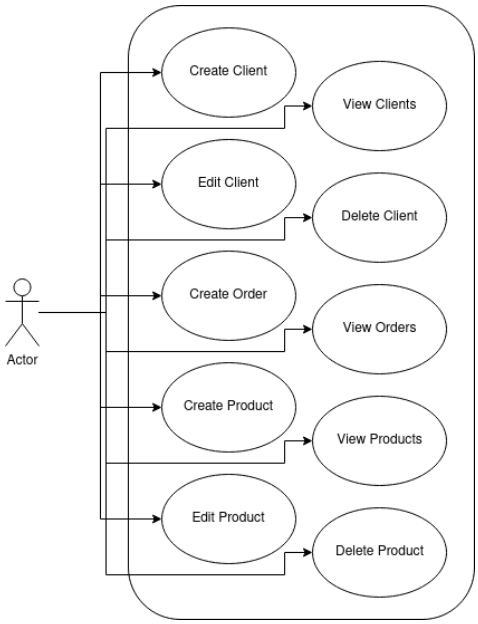
1. Assignment Objective

Main objective: Develop an Orders Management application using a layered architecture pattern and a relational database for storing clients, products, and orders.
Sub-objectives:

1. Design the data model for the application and implement model classes.
2. Implement data access classes for accessing the database.
3. Implement business logic classes to contain the application logic.
4. Implement presentation classes to create a user-friendly GUI.
5. Integrate the layers and classes to build a functional Orders Management application.

2. Problem Analysis, Modeling, Scenarios, Use Cases

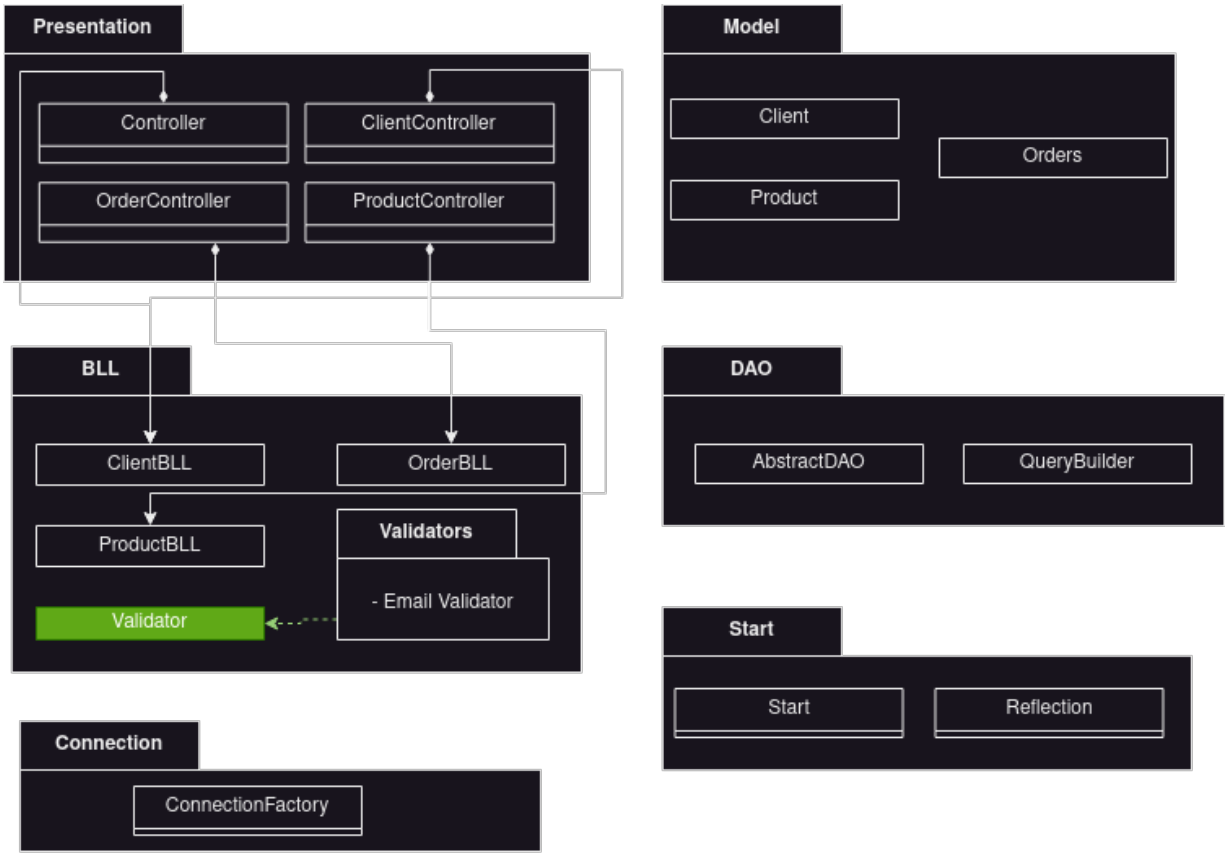
For the Orders Management application, we can identify the following use cases:



1. Create Client: This use case describes the process of creating a new client in the system. The user provides the client's details, such as name, address, email, and age. The system validates the information and saves the client in the database.
2. View Clients: This use case describes the process of viewing a list of all clients in the system. The user can select a client from the list to view their details.
3. Edit Client: This use case describes the process of editing a client's details. The user selects a client from the list and modifies their details. The system validates the changes and updates the client in the database.
4. Delete Client: This use case describes the process of deleting a client from the system. The user selects a client from the list and confirms the deletion. The system removes the client from the database.
5. Create Order: This use case describes the process of creating a new order in the system. The user selects a client and product and adds a quantity of said product to the order. The system validates the information and saves the order in the database.
6. View Orders: This use case describes the process of viewing a list of all orders in the system. The user can select an order from the list to view its details.
7. Create Product: This use case describes the process of creating a new product in the system.

8. View Products: This use case describes the process of viewing a list of all products in the system. The user can select a product from the list to view its details.
9. Edit Product: This use case describes the process of editing a product. The user selects a product from the list and modifies its details. The system validates the changes and updates the product in the database.
10. Delete Product: This use case describes the process of deleting a product from the system. The user selects a product from the list and confirms the deletion. The system removes the product from the database.

3. Design



The Product, Client, and Order classes will represent the data models of the application, and will contain the necessary fields and methods for each entity.
The ProductBLL, ClientBLL, and OrderBLL classes will contain the application logic for their respective entities. They will be responsible for creating, updating, deleting, and retrieving products, clients, and orders from the database.

The Controller class will serve as the main entry point of the application and will contain the necessary menus and buttons to navigate between the GUI pages, which will in turn be controlled by ProductController, ClientController, and OrderController respectively.

An AbstractDAO class can be a useful way to provide a common implementation for all the DAOs in an application. The AbstractDAO class defines the common CRUD (create, read, update, delete) methods.

4. Implementation

The application uses the layered architecture pattern and all Business Logic Layer (BLL) classes, namely ClientBLL, OrderBLL, and ProductBLL, utilize the AbstractDAO class for database access. The AbstractDAO class contains methods to execute queries and interact with the database.
The ClientController, OrderController, and ProductController classes serve as the controllers for their respective models in the Model-View-Controller (MVC) pattern. They handle requests from the presentation layer and call the appropriate BLL methods to perform the required operations.

The Client, Orders, and Product classes represent the data models of the application. They define the attributes of their respective entities and provide methods to manipulate the data.

The QueryHelper class contains methods to generate SQL queries for the BLL classes to execute.

The ConnectionFactory class provides a connection to the database for the AbstractDAO class to use.

The EmailValidator class provides methods to validate email addresses.

5. Conclusions

In this assignment, we have designed an Orders Management application for processing client orders, using a layered architecture pattern.

Through this project, I have learned about the importance of using a layered architecture pattern to create a scalable and maintainable application. I have also gained practical experience in designing and implementing a system that integrates with a relational database and uses the reflection technique.

For future developments, the application can be extended to include additional features such as search functionality, sorting, filtering, and pagination to improve the user experience.

6. Bibliography

The references that were consulted by the student during the implementation of the homework will be added.

Example:

1. *Bruce Eckel, Thinking in Java (4th Edition), Publisher: Prentice Hall PTRUpper Saddle River, NJUnited States, ISBN:978-0-13-187248-6 Published:01 December 2005.*
2. *JavaDoc Baeldung documentation - https://www.baeldung.com/javadoc*
3. *What are Java classes? - www.tutorialspoint.com*
4. *Reflection in Java - https://www.youtube.com/watch?v=bhhMJSKNCQY&t=284s*
5. *Layered Architecture - https://www.youtube.com/watch?v=-EiyyCA0GPk*