**Documentation**

Assignment 3

Orders management

Name: Știrb Călin-Alexandru

Group: 30422

**Contents:**

1. Assignment Objective............................................................................................3
2. Problem Analysis, Modeling, Scenariosm, Use Cases..........................................3
3. Design....................................................................................................................6
4. Implementation......................................................................................................9
5. Results....................................................................................................................13
6. Conclusions............................................................................................................13
7. Bibliography...........................................................................................................13
8. **Assignment Objective**

The main objective of this assignment is to learn how to create connections between java applications and external databases, in this case a postgreSQL database. We also learn how to write and then generate javaDOC files for our classes.

**Sub-objectives:**

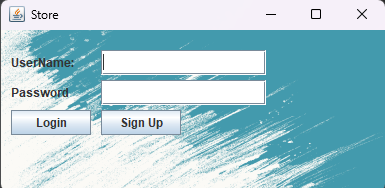
* Implementing the functional classes ( see the **design** section for more info):
  + AbstractDao: class provides a generic implementation for database operations using JDBC.
    - ClientDAO
    - OrderDAO
    - ProductDAO
  + ClientLogic: class represents the logic for client-related operations in the application.
  + OrderLogic class represents the logic for order-related operations in the application.Gui: the visual element of the application
  + ProductLogic class represents the logic for product-related operations in the application.
  + Client class represents a client in the application
  + Order class represents an order in the application.
  + Product class represents a product in the application.

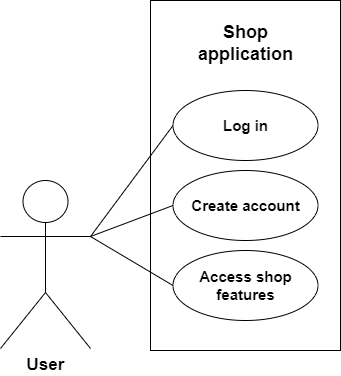
A screenshot of a computer

Description automatically generated with medium confidence

1. **Problem analysis, Modeling, Scenarios, Use cases**

For the functional part of the application, the user is required to enter a series of numbers into their respectable text fields, which represent the control values of the simulation:

The screenshots to the left are the user-interactive part of the GUI, the first gui you see when you open the application, the login screen, and the one you see after login, the main menu.

 The use-case diagram illustrates the direct functionality of the application

The primary actor for all the use cases is the ***user.***

Main success scenario for any of the illustrated use cases:

1. The user successfully registers and logs into their account.
2. The user selects what kind of operations they want to execute: client oriented, product oriented or order oriented.
3. The user successfully uses the implemented features.
4. The user uses the list function to see the changes they made.

Alternative sequence: Incorrect login data

* The user does not have an account or introduces the wrong account data.
* An error message is displayed.

A picture containing screenshot, circle, black and white, spring

Description automatically generated

**Functional requirements:**

* The app should allow the user to create and log into an account
* The app should allow the user to see the changes they have made
* The app should allow the user to commit changes to any of the 3 tables in the database: Client,Product,Order.

**Non-functional requirements:**

* The app should be easy to use by the user
* The output data should be formatted in such a way that it’s easy to understand: a Jtable format

1. **Design**

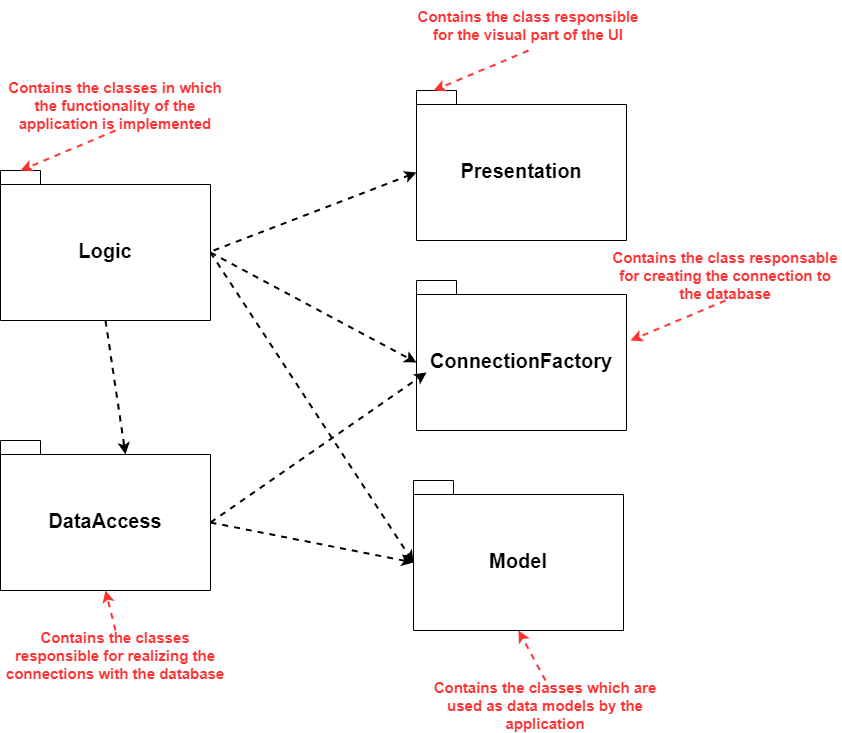
This section will provide a more in-depth view of the design choices taken during the development of the application.

* The data models used by this app are the classes: Client, Product and Order which are used to store elements from their respective tables in the database

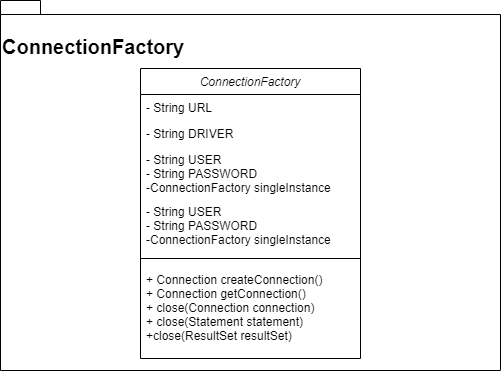
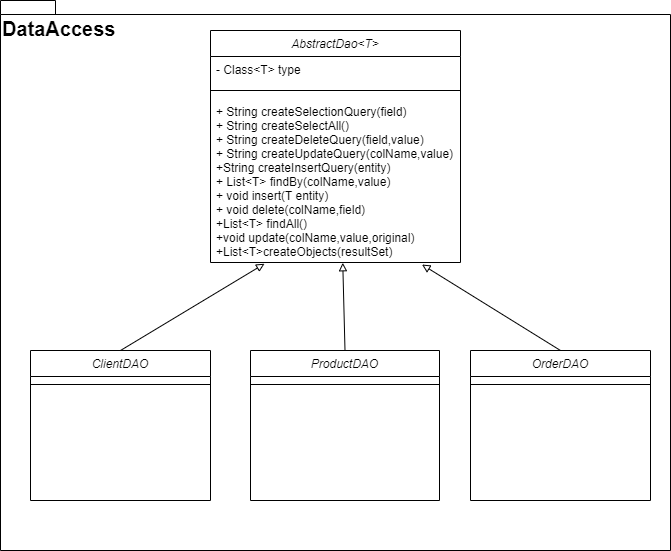
Level 1: Overall system design

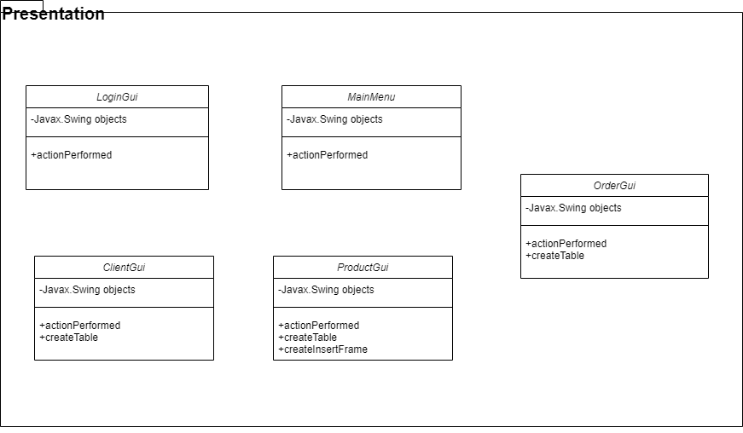
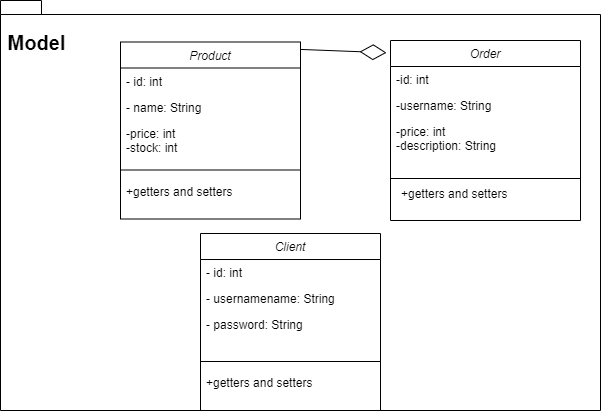
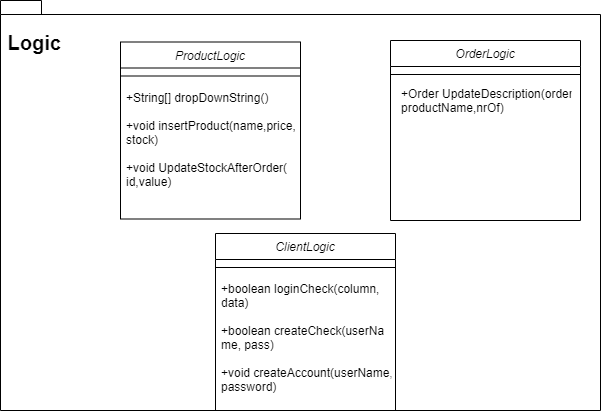


* In the diagram above are represented the main inputs and outputs of the app
* The series of inputs on the left represent the user’s own written input
* The input below is the drop down menu for the selection of different products
* The input above named “Action buttons” represent the buttons which are used to give the graphical interface of the application it’s functional aspect
* The output are the JTables which represent the modified tables.

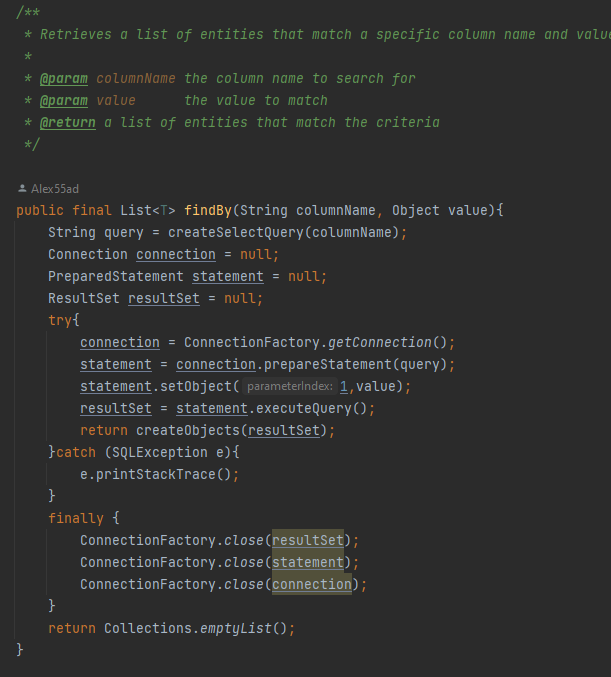
Level2: Division into packages

Level 3: Division into classes





1. **Implementation**
2. **AbstractDAO**
   * This class contains 1 objects of type Class<T>: type
   * This class contains multiple methods for access to the database, which are passed to it’s child classes to perform access to the database for their respective table in the database:
     + For example the following method which returns a List of type T that respect the slection query by a columnName of value value

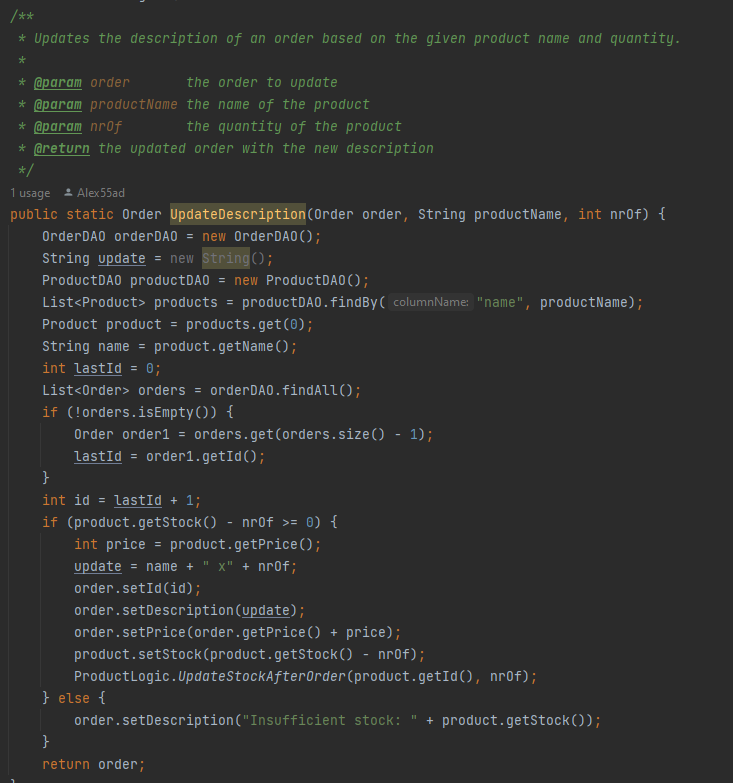


1. **ClientDAO**
   * This class extends the AbstractDAO<Client>for the type client.
2. **ProductDAO**
   * This class extends the AbstractDAO<Product> for the type product.
3. **OrderDAO**
   * This class extends the AbstractDAO<Order> for the type order.
4. **Client Logic**
   * This class does not contain any objects.
   * This class contains multiple methods which handle the logic of the operations on clients:
     + For example the method: loginCheck

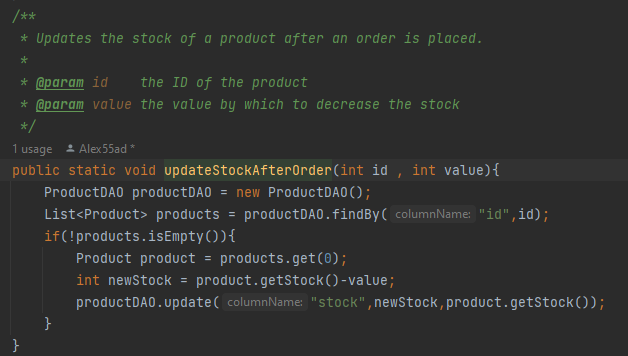
A picture containing text, screenshot, font

Description automatically generated

1. **OrderLogic class**
   * This class does not contain any objects
   * This class implements one method that deal with the logic of operations on orders, UpdateDescription:



1. **ProductLogic class**
   * This class does not contain any objects.
   * This class contains multiple methods which handle the logic of the operations on products:
     + For example the method: updateStockAfterOrder



1. **Client class**
   * This class contains 3 objects:
     + Int id
     + String username
     + String password
   * This class does not contain any methods except for getters/setters
2. **Product class**
   * This class contains 3 objects:
     + Int id
     + String name
     + Int price
     + Int stock
   * This class does not contain any methods except for getters/setters
3. **Order class**
   * This class contains 3 objects:
     + Int id
     + String username
     + Int price
     + String description
   * This class does not contain any methods except for getters/setters
4. **LoginGui class**
   * This class contains a lot of javax.swing elements, such as JButtons,JtextAreas,JTables,JLabels,etc.
   * This class contains only one method: actionPerformed
     + This method acts as the controller for this UI
5. **ClientGui class**
   * This class contains a lot of javax.swing elements, such as JButtons,JtextAreas,JTables,JLabels,etc.
   * This class contains only 2 methods: actionPerformed, and createTable
     + The first method acts as the controller for this UI
     + The second method creates the JTable that displays all the elements of the table client
6. **MainMenu class**
   * This class contains a lot of javax.swing elements, such as JButtons,JtextAreas,JTables,JLabels,etc.
   * This class contains only one method: actionPerformed
     + This method acts as the controller for this UI
7. **OrderGui class**
   * This class contains a lot of javax.swing elements, such as JButtons,JtextAreas,JTables,JLabels,etc.
   * This class contains only 2 methods: actionPerformed, and createTable
     + The first method acts as the controller for this UI
     + The second method creates the JTable that displays all the elements of the table order
8. **ProductGui class**
   * This class contains a lot of javax.swing elements, such as JButtons,JtextAreas,JTables,JLabels,etc.
   * This class contains 3 methods: actionPerformed, createTable and createInsertFrame
     + The first method acts as the controller for this UI
     + The second method creates the JTable that displays all the elements of the table product
     + The third method creates a new JFrame that enables the user to input all the data necessary to insert a new Product into the table
9. **ConnectionFactory class**
   * This class contains the following objects:
     + String URL
     + String DRIVER
     + String PASSWORD
     + String USER
     + ConnectionFactory singleInstance
   * This class contains methods that deal with creating/closing direct connections to the database:
     + createConnection()
     + getConnection()
     + close(Connection)
     + close(Statement)
     + close(ResultSet)

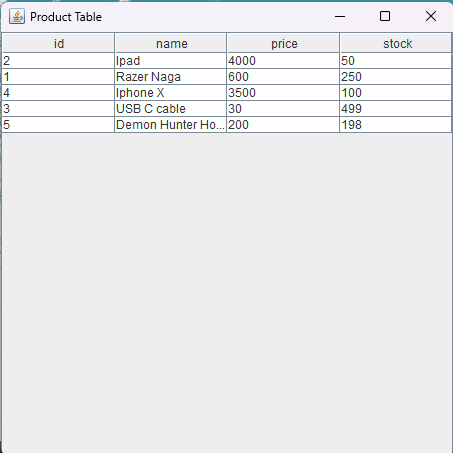
A screenshot of a computer program

Description automatically generated with medium confidence

1. A screenshot of a computer

   Description automatically generated with medium confidenceA screenshot of a computer

   Description automatically generated with low confidence**Results**



A few screenshots with the output in the GUI;

1. **Conclusions**

In conclusion, this assignment helped me hone my skills with java OOP, especially with projects that require database management. It also helped me a lot with creating neat GUI, organizing my code a lot better, and it felt like I was developing an actual application for (almost) public use.

1. **Bibliography**

* Connect to PostgreSQL database:
  + *https://www.postgresqltutorial.com/postgresql-jdbc/connecting-to-postgresql-database/*
* Layered architectures
  + https://dzone.com/articles/layers-standard-enterprise
* Reflection in Java
  + http://tutorials.jenkov.com/java-reflection/index.html
* Creating PDF files in Java
  + <https://www.baeldung.com/java-pdf-creation>
* JAVADOC
  + https://www.baeldung.com/javadoc
* SQL dump file generation
  + https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html