**DOCUMENTATION**

ASSIGNMENT 3

STUDENT NAME: LACATUS ARPAD ALEX

GROUP: 30423

# CONTENT

[1. Objective of the assignment 3](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297885)

[2. Problem analysis, modeling, scenarios, use cases 3](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297886)

[3. Design 4](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297887)

[4. Implementation 5](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297888)

[5. Results 6](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297889)

[6. Conclustions 6](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297890)

[7. Bibliography 6](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297891)

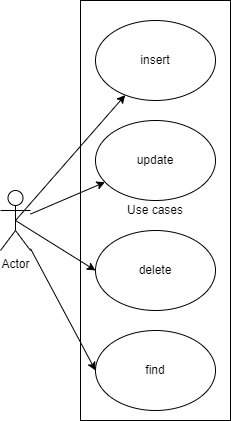
1. The Objective of the assignment - The main objective of this assignment is to design and implement an application for managing the client orders for a warehouse by also storing all the valuable data in interconnected responsive tables using MySQL , and providing receipt for each order . Some secondary objectives are:

-analyzing the problem and identifying the requirements

-designing the orders management application that works for our problem

-implementing that said orders management application as simple and efficient as possible

-testing the orders management application for various cases

1. Problem Analysis, modeling, scenarios, use-cases

First of all we need to consider all the possible steps our app could go through and what methods we require in order to implement this app as efficient as we could and as user friendly as possible. We will need a lot of classes and methods to make the implementation easy to understand, work with, and in order to solve better the bugs that may occur. Another thing to consider is that our implementation of the modeling classes should mirror the tables in our database in order to create a proper connection.

**Analyzing** the problem, we have the following requirements:

**For inserting:**

-the orders management application should allow users to insert a new client with data they wish for (with some restrictions)

-the orders management application should allow users to insert a new product with data they wish for (with some restrictions)

-the orders management application should allow users to insert a new order with data they wish for (with some restrictions) , after checking that the client and product associated are valid, also should provide a new recipe for this order

**For updating:**

-the orders management application should allow users to update an old client with data they wish for (with some restrictions)

-the orders management application should allow users to update an old product with data they wish for (with some restrictions)

-the orders management application should allow users to update an old order with data they wish for (with some restrictions) , after checking that the client and product associated are valid and making the required changes in the other tables too, also should provide a new receipt for this order

**For finding:**

-the orders management application should allow users to find a client either by providing an id or by choosing the option to list all of them

-the orders management application should allow users to find a product either by providing an id or by choosing the option to list all of them

-the orders management application should allow users to find an order either by providing an id or by choosing the option to list all of them

**For deleting:**

-the orders management application should allow users to delete a client either by providing an id or by choosing the option to delete all of them

-the orders management application should allow users to delete a product either by providing an id or by choosing the option to delete all of them

-the orders management application should allow users to delete an order either by providing an id or by choosing the option to delete all of them

For **modeling** our data, we use mainly the Client class, Order class and the Product class. Alongside them we have the other classes that work with these two and orchestrate the actual application and provide accessibility for these 3, mainly they work in trios (e.g., Client, ClientLogic, ClientDAO). These three were chosen to represent the tables in our database, the mirror our tables in order for the connection between our app and the database to be possible.

**Use cases**: add product/order/client

The primary actor is the user, and the **main scenario** would be:

1. The user chooses the table he wants to work with from the tables provided by the GUI (client, order, product)
2. The user inserts the values for the fields of the table (i.e., name, address/category, etc.) and makes sure the data matches the patterns and for the orders make sure that the client and product parameters are valid
3. The user clicks on the “insert” button to insert the wanted element in the database
4. The application updates the table and the database with the desired input, if we work with orders also creates a new receipt

**Alternative Sequence:** Invalid values for the insert parameters

- The user inserts incorrect numbers for the insert method.

- The application displays a message describing the error and request the user to insert valid values

**Use cases**: update product/order/client

The primary actor is the user, and the **main scenario** would be:

1. The user chooses the table he wants to work with from the tables provided by the GUI (client, order, product)
2. The user inserts the values for the updated fields of the table (i.e., name, address/category, etc.) and makes sure the data matches the patterns and for the orders make sure that the client and product parameters are valid
3. The user clicks on the “update” button to update the wanted element in the database
4. The application updates the table and the database with the desired change, in case we are working with the order a receipt will also be provided

**Alternative Sequence:** Invalid values for the update parameters

- The user inserts incorrect numbers for the update method.

- The application displays a message describing the error and request the user to insert valid values

**Use cases**: find product/order/client

The primary actor is the user, and the **main scenario** would be:

1. The user chooses the table he wants to work with from the tables provided by the GUI (client, order, product)
2. The user inserts the values for the id to be searched for
3. The user clicks on the “find” button to find the wanted element in the database
4. The application updates the table with the desired item

**Alternative Sequence:** Invalid values for the find parameters

- The user inserts an inexistent id

- The application will display an empty table

**Use cases**: findall product/order/client

The primary actor is the user, and the **main scenario** would be:

1. The user chooses the table he wants to work with from the tables provided by the GUI (client, order, product)
2. The user clicks on the “findall” button
3. The application shows all the elements in the chosen table

**Alternative Sequence:** There is no alternative sequence.

**Use cases**: delete product/order/client

The primary actor is the user, and the **main scenario** would be:

1. The user chooses the table he wants to work with from the tables provided by the GUI (client, order, product)
2. The user inserts the values for the id of the item to be deleted
3. The user clicks on the “delete” button to find the wanted element in the database and delete it
4. The application updates the table and the database with the desired change

**Alternative Sequence:** Invalid values for the delete parameters

- The user inserts an inexistent id

- Nothing will change

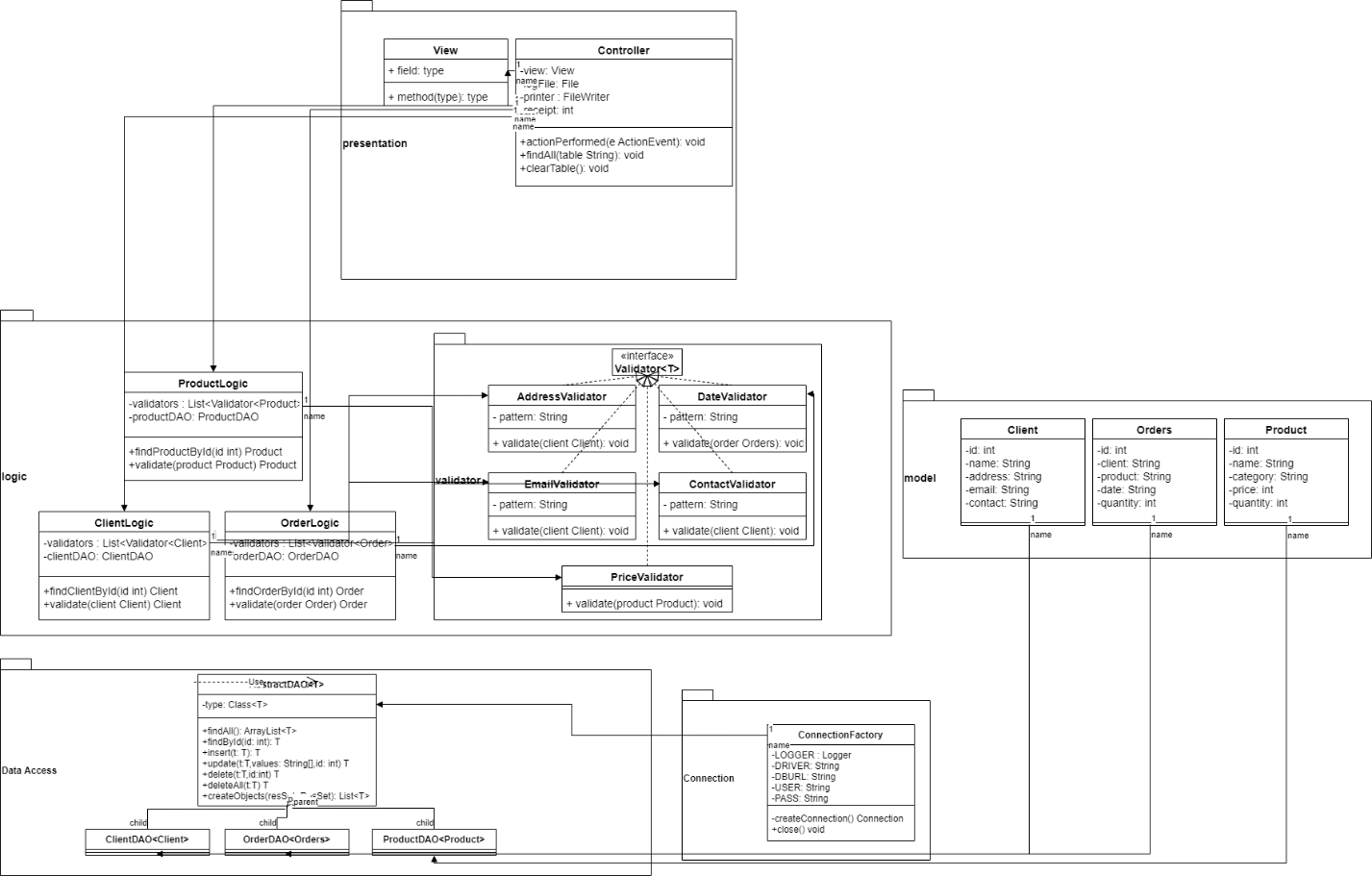
**Use cases**: deleteall product/order/client

The primary actor is the user, and the **main scenario** would be:

1. The user chooses the table he wants to work with from the tables provided by the GUI (client, order, product)
2. The user clicks on the “deleteall” button
3. The application shows an empty table and the table chosen will be cleared in the database

**Alternative Sequence:** There is no alternative sequence

To note that for this application to consider the inputs valid the address is of the format “\* str”, contact contains only numbers, email of the format [\*@\*. \*](mailto:*@*.%20*) ,price>0,date of the format “\*\*/\*\*/\*\*”, and for the orders client and product should already be existent and the quantity of the product should exceed the quantity of the order.

1. Design

Leve 1: System design:

At this level we have the application itself that has as the inputs the parameters inserted in the fields that we want to work with and the chosen table by the user and the resulted table as the output in the GUI , to note that for working with orders we also have the output files that are the receipts.

Level 2: Design into sub-systems/packages:

We will need 3 main packages for this assignment:

-Graphical User Interface (presentation) – the package that contains the classes used for implementing the graphical user interface, we only have two (View, Controller) each ones role being fully described by their name

-Data Models (models) -the package that contains the classes modeling the application data (Client, Order, Product), here we have the main classes used for implementing the tables form the database so they need to be mirroring the tables with their attributes

-Business Logic(logic) – the package that contains the classes used for running the application. These classes are used at the higher level and they contain all the required information, and methods for working with the database and validating the inputs thanks to their components. Here we have the classes ClientLogic, OrderLogic, ProductLogic. We also have an extra package here containing all the validators used for validating the inputs, used by the logic classes.

-Data Access(data,access) - the package that contains the classes responsible with implementing the methods for working with the database( insert, update, delete, find). Here we have the AbstractDAO the class that implements the methods mentioned above with generics and eventually the ClientDAO, OrderDAO, ProductDAO will inherit these methods with the right types.

-Connection(connection) – the package containing the class tasked with making the connection with our database. ConnectionFactory makes the connection by logging in and after all the operations are performed closing the necessary connections.

Level 3: Division into classes

The 3 main classes used are Client, Order, Product used for modeling data used in trio with their helpers ClientDAO, OrderDAO, ProductDAO (for data access),ClientLogic, OrderLogic, ProductLogic(for logic operations). Other classes used are the ones used for validating input(the validators), for managing the connection ConnectionFactory, for display the GUI(View) and for getting the input and preparing the next operation(Controller)

Level 4: Division into routines

All the classes implemented are divided in routines in order to facilitate our work and make the code much easier to understand, the methods are implemented in such a way so we don’t have unnecessary long routines with hundreds lines of code in order to not make things too complicated.

Level 5: Internal routine design

In each routine our code is well delimited in pieces of code with different functionalities, which combined give us a final routine easy to follow and comprehend.

1. Implementation

Data Model classes:

**Client:** Client is the class used for modeling a client from the database. It mirrors a row from the client table and has 3 constructors along with getters and setters needed in order to better manipulate our data.

**Order:** Orders is the class used for modeling an order from the database. It mirrors a row from the orders table and has 3 constructors along with getters and setters needed in order to better manipulate our data. Note that the class is called "Orders" instead of "Order" because "order" is a reserved word in sql language.

**Product:** Product is the class used for modeling a product from the database. It mirrors a row from the product table and has 3 constructors along with getters and setters needed in order to better manipulate our data

Logical classes:

**ClientLogic:** ClientLogic class represents the class that interacts directly with the controller providing the DAO part and validating the inputs. Its constructor selects the required validators and creates a new DAO for the clients.It has the function findClientById that with the DAO help returns the client searched for after validating it. Validate method is used for validating a client.

**OrderLogic:** OrderLogic class represents the class that interacts directly with the controller providing the DAO part and validating the inputs. Its constructor selects the required validators and creates a new DAO for the orders .It has the function findOrderById that with the DAO help returns the order searched for after validating it. Validate method is used for validating an order.

**ProductLogic:** ProductLogic class represents the class that interacts directly with the controller providing the DAO part and validating the inputs. Its constructor selects the required validators and creates a new DAO for the products.It has the function findProductById that with the DAO help returns the product searched for after validating it. Validate method is used for validating a product.

**Validators:** These classes implement the Validator interface. They have a pattern with which they check if a client/order/product is valid in the validate function by matching their attributes mith the required pattern.

Data Access classes:

**AbstractDAO:** AbstractDAO is used for implementing our DAOs.This class has methods used for creating specific queries(select, insert, delete, update) that return a specific string written in sql language for each operation and table.Here we have the important methods that process the string mentioned before and provide them to the databse and receive its response(findById, insert, update, delete, deleteAll, findAll. The function createObjects is responsible with creating the objects by choosing the right constructor and the right fields for each item then creating it and adding to a list that we’ll be eventually returned;

**ClientDAO, OrderDAO, ProductDAO:** Classes that inherit all the methods from the AbstractDAO and have no methods of their own, only give the required type for the methods already implemented.

Connection classes:

**ConnectionFactory:** ConnectionFactory is the class that manages the connection with the database. It has the credentials used for logging and uses them to do so. Has the methods createConnection that tries to create the connection using the credentials, a getter for the connection, and 3 close methods that close a connection/statement/resultset..

GUI classes:

**View:** View class has all the components that are displayed on the screen (buttons, labels, text fields, combo box, table) and is connected with the controller. In the constructor it initializes all its jcomponents.

**Controller:** Controller class is the action listener and when it receive a stimuli it prepares with the help of some methods the operation that follows. Depending on the button pressed and the table chosen its functionality differs. All the operations use the logic components to access the dao components and to interact with the database and validate the inputs if needed then the result is given to the output table and the output file if needed. For the insert snd update order the implementation also has the part when we checjk if the client and product are viable and update the data for the product when needed.We also have the method getUpdateParams for easily access to some fields and clearTable used for clearing the table in the GUI before each operation.

Other things to be considered:

The id for the items will not start from 1 because of the testing that has been done beforehand and the autoincrementing function of the data base. The receipt number is not correlated with the id of the order, being incremented for 1 each time we start the application.

1. Results

The testing was done manually by introducing some random various inputs in order to test the functionality of the program. It was discovered that the program works as intended and there were no major bugs reported.

.

1. Conclusions

To conclude, I found this homework quite enjoyable and a good practice for working with databases. I consider that the final application meets most if not all the requirements presented.

I think this assignment really helped me gain some experience working with bot GUI and databases and I am thankful for this opportunity.

1. Bibliography

For this project I mainly use only the resources provided at the class, I found them easy enough to grasp and implement so I did not need the help of other resources whatsoever. The links of the resources are:

<https://dsrl.eu/courses/pt/materials/A3_Support_Presentation.pdf>

<https://dsrl.eu/courses/pt/materials/PT2021-2022_Assignment_3.pdf>

For the documentation the documentation template provided was used