**DOCUMENTATION**

**ASSIGNMENT #3**

STUDENT NAME: Bărăian Adrian-Călin

GROUP: 30422

**CONTENTS**

[**1.** **Assignment Objective** 3](#_Toc166688558)

[**2.** **Problem Analysis, Modeling, Scenarios, Use Cases** 4](#_Toc166688559)

[**3.** **Design** 8](#_Toc166688560)

[**4.** **Implementation** 15](#_Toc166688561)

[**6.** **Bibliography** 16](#_Toc166688562)

1. **Assignment Objective**

**Main Objective**

* Design and implement an application for managing the client orders for a warehouse.

**Sub-objectives**

* Analyze the problem and identify requirements;
* Design the orders management application;
* Implement the orders management application;
* Test the orders management application.

1. **Problem Analysis, Modeling, Scenarios, Use Cases**

**Use Cases**

A diagram of a company

Description automatically generated

Figure 1. Use Case Diagram

1. **Use Case**: Add Client

**Primary Actor**: User

**Main Success Scenario**:

1. The user selects the client window from the main window of the application
2. The user completes the form fields inside the window with client data
3. The user presses the “Add” button
4. The application stores the client data in the database and displays a message of success

**Alternative Sequence**: Invalid values for the client’s data

* The user inserts invalid email format
* The application displays an error message and requests the user to insert valid values
* The scenario returns to step 2

1. **Use Case**: Edit Client

**Primary Actor**: User

**Main Success Scenario**:

1. The user selects the client window from the main window of the application
2. The user selects a record from the table of clients
3. The user completes the form fields inside the window with client data
4. The user presses the “Edit” button
5. The application stores the new client data in the database and displays a message of success

**Alternative Sequence**: Invalid values for the client’s data

* The user inserts invalid email format
* The application displays an error message and requests the user to insert valid values
* The scenario returns to step 2

1. **Use Case**: Delete Client

**Primary Actor**: User

**Main Success Scenario**:

1. The user selects the client window from the main window of the application
2. The user selects a record from the table of clients
3. The user presses the “Delete” button
4. The application deletes the client data from the database and displays a message of success

**Alternative Sequence**: No record selected

* The user does not select a record from the client table
* The application displays a message with the text “Please select a row”
* The scenario returns to step 2

1. **Use Case**: View Clients

**Primary Actor**: User

**Main Success Scenario**:

1. The user selects the client window from the main window of the application
2. The application displays the table of clients in the client window
3. **Use Case**: Add Product

**Primary Actor**: User

**Main Success Scenario**:

1. The user selects the product window from the main window of the application
2. The user completes the form fields inside the window with product data
3. The user presses the “Add” button
4. The application stores the product data in the database and displays a message of success

**Alternative Sequence**: Invalid values for the product’s data

* The user inserts invalid price or quantity
* The application displays an error message and requests the user to insert valid values
* The scenario returns to step 2

1. **Use Case**: Edit Product

**Primary Actor**: User

**Main Success Scenario**:

1. The user selects the product window from the main window of the application
2. The user selects a record from the table of products
3. The user completes the form fields inside the window with product data
4. The user presses the “Edit” button
5. The application stores the new product data in the database and displays a message of success

**Alternative Sequence**: Invalid values for the product’s data

* The user inserts invalid price or quantity
* The application displays an error message and requests the user to insert valid values
* The scenario returns to step 2

1. **Use Case**: Delete Product

**Primary Actor**: User

**Main Success Scenario**:

1. The user selects the product window from the main window of the application
2. The user selects a record from the table of products
3. The user presses the “Delete” button
4. The application deletes the product data from the database and displays a message of success

**Alternative Sequence**: No record selected

* The user does not select a record from the product table
* The application displays a message with the text “Please select a row”
* The scenario returns to step 2

1. **Use Case**: View Products

**Primary Actor**: User

**Main Success Scenario**:

1. The user selects the product window from the main window of the application
2. The application displays the table of products in the product window
3. **Use Case**: Create Order

**Primary Actor**: User

**Main Success Scenario**:

1. The user selects the order window from the main window of the application
2. The user selects a client
3. The user selects a products
4. The user selects a quantity
5. The user presses the “Add” button
6. The application creates an order with the selected data and inserts it in the database along with a bill of the order

**Alternative Sequence**: No both data selected

* The user does not select a client or a product
* The user presses the “Add” button
* The application displays a message that the user did not select rows from both tables
* The scenario returns to step 2

1. **Use Case**: View Orders

**Primary Actor**: User

**Main Success Scenario**:

1. The user selects the order window from the main window of the application
2. The user presses the “View Orders” button from the order window
3. The application displays the table of orders

1. **Use Case**: View Bills

**Primary Actor**: User

**Main Success Scenario**:

1. The user selects the view bills window from the main window of the application
2. The application displays the table of bills

**Functional Requirements**

* The simulation application should allow users to add a new client
* The simulation application should allow users to add a new product
* The simulation application should allow users to create a new order
* The simulation application should allow users to edit a client
* The simulation application should allow users to edit a product
* The simulation application should allow users to delete a client
* The simulation application should allow users to delete a product
* The simulation application should allow users to visualize the clients
* The simulation application should allow users to visualize the products
* The simulation application should allow users to visualize the orders
* The simulation application should allow users to visualize the bills of orders

**Non-Functional Requirements**

* Intuitiveness:
  + The GUI should be intuitive and easy to use by the user;
* Stability:
  + The application should be stable and reliable;
* Input validation:
  + The application should validate user inputs to prevent erroneous computations.
* Security:
* The application should include a login system

1. **Design**

**Level 1**: Overall System Design

A diagram of a data access

Description automatically generatedFigure 2. Conceptual Architecture of the Application

**Level 2**: Division into sub-systems/packages

A screenshot of a computer

Description automatically generated

Figure 3.Classes of bll Package

A screenshot of a computer

Description automatically generated

Figure 4. Classes Of connection Package

A screenshot of a computer

Description automatically generated

Figure 5. Classes of dao Package

A screenshot of a computer

Description automatically generated

Figure 6. Classes Of model Package

A screenshot of a computer screen

Description automatically generated

Figure 7. Classes of presentation Package

**Level 3.** Division into classes

A diagram of a company

Description automatically generatedFigure 8. Class Diagram of the Application

**Data Structures and Types used**:

**- Java ArrayList:** The ArrayList class is a resizable array, which can be found in the java.util package.

**- Connection** - public interface: A connection (session) with a specific database. SQL statements are executed and results are returned within the context of a connection.

**JDBC:** I have used Postgresql in the implementation of the database, as well as in the implementation of and execution of the CRUD operations.

The database dump file has been included in the project files inside the main/resources folder

1. **Implementation**

* Business Logic classes:
* Used to implement the business logic of the corresponding model classes.
* Methods: Insert, Update, Delete methods which are calling validator methods in addition to the corresponding methods from the DAO classes.
* Model classes:
* Classes which implement the entities of the database, having the same fields as the database tables.
* Methods: Getters and setters of the corresponding fields
* Connection class:
* Class used to create a connection to the database, using a PostgreSQL driver.
* Methods: createConnection(), getConnection() and methods for the closing of the connection, statements and ResultSet objects.
* DAO classes:
* Classes used to implement the CRUD operation which are specific to a database related application.
* Methods:
* Include methods which create the queries for SELECT, INSERT, DELETE, UPDATE operations.
* Include methods which create a connection and execute the corresponding queries in order to apply the operation on the database.
* AbstractDAO<T>: has been implemented with the use of the reflection technique, resulting in a generic class in which the queries are constructed through reflection. I am first retrieving the generic type of the class T and then based on the fields of that class, I am constructing the query statements.
* Presentation classes:
* Represent the Graphical User Interface and is composed of View classes and corresponding Controller classes.
* View Classes: Implement the visual part of the application
* Controller Classes: Manages the action required to be done when the user interacts with the visual interface
* Details about the implementation and code samples are documented with the help of JavaDoc API. Documents have been generated in the projects files and thus provided in the project folder.

1. **Conclusions**

The assignment has presented the opportunity to effectively solve a real-life problem by means of creating an application the user can interact with, in order to manage a database.

The assignment has helped me fixate the concepts of generic classes and reflection techniques through the development of the application as well as learning the means of database integration by developing an application which consists of front-end elements and back-end elements .

1. **Bibliography**
2. [Fundamental Programming Techniques (dsrl.eu)](https://dsrl.eu/courses/pt/) – Laboratory Assignment Guide + Lectures