DOCUMENTAȚIE

TEMA 3

Orders Management

Nume student: Matis Oana-Antonia

Grupa: 30228

# CUPRINS

[1. Obiectivul temei 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297886)

[3. Proiectare 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297887)

[4. Implementare 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297888)

[5. Rezultate 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297889)

[6. Concluzii 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297890)

[7. Bibliografie 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297891)

**1. Obiectivul temei**

Principalul obiectiv al temei este proiectarea și implementarea unei aplicații pentru gestionarea comenzilor dintr-un depozit, aplicație ce are rolul de a procesa comenzile clienților. Trebuie folosite bazele de date relaționale pentru a stoca produsele, clienții și comenzile.

Pentru a îndeplini obiectivul principal, trebuie urmați câțiva pași esențiali, și anume obiectivele secundare:

* **Analiza problemei și identificare cerințelor:** constă în identificarea principalelor obiecte și atributelor acestora (se va detalia în capitolul: *2*. *Analiza problemei, modelare, scenarii, cazuri de utilizare*)
* **Proiectare**: constă în proiectarea unei soluții care să se plieze pe toate cazurile pe care le introduce utilizatorul și se referă la structura, arhitectura proiectului (se va detalia în capitolul: *3. Proiectare)*
* **Implementare:** se referă la clasele și metodele utilizate pentru implementarea simulatorului de cozi și crearea unei interfațe user-friendly (se va detalia în capitolul: *4. Implementare)*
* **Testare:** constă în verificarea corectitudinii implementării realizate și observarea rezultatelor finale, concluziile (se va detalia în capitolul: *5. Concluzii)*

# 2. Analiza problemei, modelare, scenarii, cazuri de utilizare

1. Cerințele funcționale ale proiectului sunt:

* Aplicația trebuie să permită unui angajat să adauge un client nou în baza de date
* Aplicația trebuie să permită unui angajat să adauge un produs nou în baza de date
* Aplicația trebuie să permită unui client adăugarea unei comenzi
* Aplicatia ar trebui sa permita cautarea unui client dupa diferite criterii.
* Aplicatia ar trebui sa permita cautarea unui produs dupa diferite criterii.
* Aplicatia ar trebui sa permita cautarea unei comenzi dupa diferite criterii.
* Aplicatia ar trebui sa permita modificarea unui client in functie de un criteriu.
* Aplicatia ar trebui sa permita modificarea unui produs in functie de un criteriu.
* Aplicatia ar trebui sa permita modificarea unei comenzi in functie de un criteriu.
* Aplicatia ar trebui sa permita stergerea unui client in functie de un criteriu.
* Aplicatia ar trebui sa permita stergerea unui produs in functie de un criteriu.
* Aplicatia ar trebui sa permita stergerea unei comenzi in functie de un criteriu.
* Aplicatia ar trebui sa permita afisarea unui tabel cu clienti.
* Aplicatia ar trebui sa permita afisarea unui tabel cu produse.
* Aplicatia ar trebui sa permita afisarea unui tabel cu comenzi.

1. Cerințele non-funcționale ale proiectului sunt:

* Aplicația de simulare trebuie să fie intuitivă și ușor de folosit de către utilizator

1. Diagrama Use Case

O imagine care conține diagramă

Descriere generată automat

Descrierea Use case-urilor:

**1. Use Case**: add client  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to add a new client

2. The application will display a form in which the client details should be inserted

3. The employee inserts the client id, name and email

4. The employee clicks on the “Insert” button

5. The application stores the client data in the

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

- The user inserts an incorrect email address form

- The application displays an error message and requests the user to insert a valid email address

- The scenario returns to step 3

**2. Use Case**: add product  
 **Primary Actor:** user  
 **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 id and name of the product, its price and quantity

4. The employee clicks on the “Insert” button

5. The application stores the product data in the database

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

- The user inserts a negative value for the quantity 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. Use Case**: add order  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to add a new order

2. The application will display a form in which the product details should be inserted

3. The employee inserts the id, clientId, productId, price, quantity

4. The employee clicks on the “Insert” button

5. The application stores the order data in the database

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

- The user inserts a greater value for the quantity of the product

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

- The scenario returns to step 3

**4. Use Case**: add bill  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to add a new bill

2. The application will display a form in which the product details should be inserted

3. The employee inserts the id, clientId, productId, price, quantity

4. The employee clicks on the “Insert” button

5. The application stores the bill data in the database

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

- The user inserts a greater value for the quantity of the product

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

- The scenario returns to step 3

**5. Use Case**: edit client  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to edit a client

2. The application will display a form in which the client details should be inserted and modified

3. The user enters the necessary data: the column we want to modify, the new value, the column to search for, and the value to search for.

4. The employee clicks on the “Update” button

5. The application stores the client data in the database

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

- The user inserts an incorrect email address form

- The application displays an error message and requests the user to insert a valid email address

- The scenario returns to step 3

**6. Use Case**: edit product  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to edit a product

2. The application will display a form in which the product details should be inserted

3. The user enters the necessary data: the column we want to modify, the new value, the column to search for, and the value to search for.

4. The employee clicks on the “Update” button

5. The application stores the product data in the database

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

- The user inserts a negative value for the quantity of the product

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

- The scenario returns to step 3

**7. Use Case**: edit order  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to edit an order

2. The application will display a form in which the product details should be inserted

3. The user enters the necessary data: the column we want to modify, the new value, the column to search for, and the value to search for.

4. The employee clicks on the “Insert” button

5. The application stores the product data in the database

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

- The user inserts a greater value for the quantity of the product

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

- The scenario returns to step 3

**8. Use Case**: delete client  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to delete a client

2. The application will display a form in which the client details should be inserted

3. The user enters the necessary data: the column to search by and the value to search for the client we want to delete.

4. The employee clicks on the “Delete” button

5 The application deletes the client data from the database

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

- The user inserts data for a client that does not exist

- The application displays an error message and requests the user to insert a valid client data

- The scenario returns to step 3

**9. Use Case**: delete product  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to delete a product

2. The application will display a form in which the order details should be inserted

3. The user enters the necessary data: the column to search by and the value to search for the product we want to delete.

4. The employee clicks on the “Delete” button

5. The application deletes the product data from the database

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

- The user inserts data for a product that does not exist

- The application displays an error message and requests the user to insert a valid product data

- The scenario returns to step 3

**10. Use Case**: delete order  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to delete an order

2. The application will display a form in which the order details should be inserted

3. The user enters the necessary data: the column to search by and the value to search for the product we want to delete.

4. The employee clicks on the “Delete” button

5. The application deletes the order data from the database

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

- The user inserts data for an order that does not exist

- The application displays an error message and requests the user to insert a valid order data

- The scenario returns to step 3

**11. Use Case**: view all clients  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1) The user selects the “View all” option  
 2) The app displays the clients table

**12. Use Case**: view all products  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1) The user selects the “View all” option  
 2) The app displays the products table

**13. Use Case**: view all orders  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1) The user selects the “View all” option  
 2) The app displays the orders table

**14. Use Case**: find client by ID  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to find a client by ID

2. The application will display a form in which the client details should be inserted

3. The user enters the ID for the searched client

4. The employee clicks on the “Find” button

5. The application searches the client data in the database

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

- The user inserts an invalid ID

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

- The scenario returns to step 3

**15. Use Case**: find client by name  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to find a client by name

2. The application will display a form in which the client details should be inserted

3. The user enters the name for the searched client

4. The employee clicks on the “Find” button

5. The application searches the client data in the database

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

- The user inserts an invalid name

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

- The scenario returns to step 3

**16. Use Case**: find client by email  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to find a client by email

2. The application will display a form in which the client details should be inserted

3. The user enters the name for the searched client

4. The employee clicks on the “Find” button

5. The application searches the client data in the database

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

- The user inserts an invalid email

- The application displays an error message and requests the user to insert a valid email address

- The scenario returns to step 3

**17. Use Case**: find product by ID  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to find a product by ID

2. The application will display a form in which the product details should be inserted

3. The user enters the ID for the searched product

4. The employee clicks on the “Find” button

5. The application searches the client data in the database

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

- The user inserts an invalid ID

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

- The scenario returns to step 3

**18. Use Case**: find product by name  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to find a product by name

2. The application will display a form in which the product details should be inserted

3. The user enters the name for the searched product

4. The employee clicks on the “Find” button

5. The application searches the product data in the database

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

- The user inserts an invalid name

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

- The scenario returns to step 3

**19. Use Case**: find product by price   
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to find a product by price

2. The application will display a form in which the product details should be inserted

3. The user enters the price for the searched product

4. The employee clicks on the “Find” button

5. The application searches the product data in the database

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

- The user inserts an invalid price

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

- The scenario returns to step 3

**20. Use Case**: find product by quantity   
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to find a product by quantity

2. The application will display a form in which the product details should be inserted

3. The user enters the quantity for the searched product

4. The employee clicks on the “Find” button

5. The application searches the product data in the database

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

- The user inserts an invalid price

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

- The scenario returns to step 3

**21. Use Case**: find order by ID  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to find an order by ID

2. The application will display a form in which the order details should be inserted

3. The user enters the ID for the searched order

4. The employee clicks on the “Find” button

5. The application searches the order data in the database

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

- The user inserts an invalid ID

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

- The scenario returns to step 3

**22. Use Case**: find order by clientID  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to find an order by clientID

2. The application will display a form in which the order details should be inserted

3. The user enters the clientID for the searched order

4. The employee clicks on the “Find” button

5. The application searches the order data in the database

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

- The user inserts an invalid clientID

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

- The scenario returns to step 3

**23. Use Case**: find order by productID  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to find an order by productID

2. The application will display a form in which the order details should be inserted

3. The user enters the productID for the searched order

4. The employee clicks on the “Find” button

5. The application searches the order data in the database

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

- The user inserts an invalid productID

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

- The scenario returns to step 3

**24. Use Case**: find order by price  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to find an order by price

2. The application will display a form in which the order details should be inserted

3. The user enters the price for the searched order

4. The employee clicks on the “Find” button

5. The application searches the order data in the database

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

- The user inserts an invalid price

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

- The scenario returns to step 3

**25. Use Case**: find order by quantity  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to find an order by quantity

2. The application will display a form in which the order details should be inserted

3. The user enters the price for the searched order

4. The employee clicks on the “Find” button

5. The application searches the order data in the database

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

- The user inserts an invalid qunatity

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

- The scenario returns to step 3

**26. Use Case**: find bill by clientID  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to find a bill by clientID

2. The application will display a form in which the bill details should be inserted

3. The user enters the clientID for the searched bill

4. The employee clicks on the “Find” button

5. The application searches the bill data in the database

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

- The user inserts an invalid clientID

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

- The scenario returns to step 3

**27. Use Case**: find bill by productID  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to find an bill by productID

2. The application will display a form in which the bill details should be inserted

3. The user enters the productID for the searched bill

4. The employee clicks on the “Find” button

5. The application searches the bill data in the database

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

- The user inserts an invalid productID

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

- The scenario returns to step 3

**28. Use Case**: find bill by price  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to find a bill by price

2. The application will display a form in which the bill details should be inserted

3. The user enters the price for the searched bill

4. The employee clicks on the “Find” button

5. The application searches the bill data in the database

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

- The user inserts an invalid price

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

- The scenario returns to step 3

**29. Use Case**: find bill by quantity  
 **Primary Actor:** user  
 **Main Success Scenario**:  
 1. The employee selects the option to find a bill by quantity

2. The application will display a form in which the product details should be inserted

3. The user enters the price for the searched bill

4. The employee clicks on the “Find” button

5. The application searches the order data in the database

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

- The user inserts an invalid quantity

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

- The scenario returns to step 3

**3. Proiectare**

Diagrama de pachete:

O imagine care conține text

Descriere generată automat

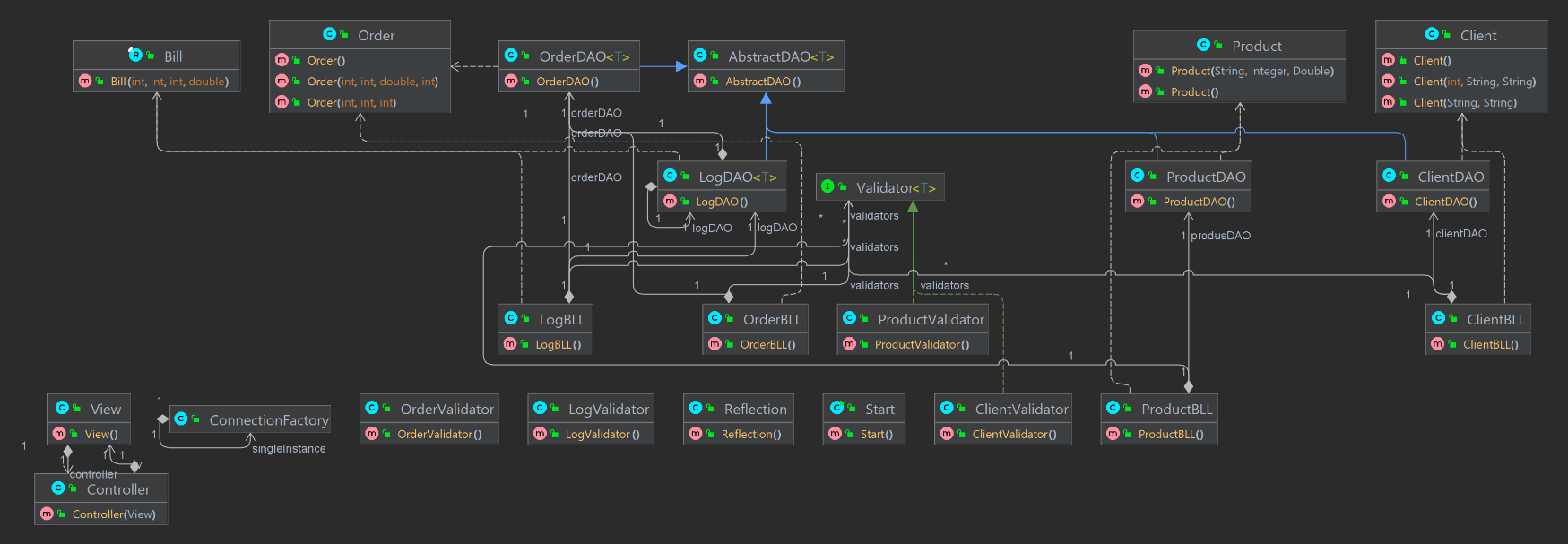
Pentru realizarea aplicației depozitului, am folosit arhitectură structurală. Asta înseamnă că am creat mai multe pachete, care reprezintă mai multe straturi ale aplicației. Pachetele create sunt următoarele:

* **Pachetul Model:** conține clasele mapate la tabelele din baza de date: Client, Produs, Orders, Bill.
* **Pachetul Connection:** conține clasa care realizează conexiunea claselor la baza de date: ConnectionFactory
* **Pachetul DAO:** conține clasele care vor efectua interogările asupra bazei de date : AbstractDAO, ClientDAO, ProdusDAO, OrderDAO, LogDAO.
* **Pachetul BLL (Business Logic Layer):** conține clasele care vor apela metodele din clasele pachetului DAO: ClientBLL, OrdersBLL, ProdusBLL, LogBLL. Acest pachet conține un al pachet, și anume **Validator**. Acest pachet conține niște clase care sunt validatori pentru datele introduse de utilizator, adică împiedică introducerea datelori invalide (ex. preț mai mic sau egal decât 0, cantitate mai mică decât 0 etc.)
* **Pachetul Presentation:** conține clasele care au rolul în realizarea interfeței aplicației: View, Controller, Reflection
* **Pachetul Main:** conține clasa Main a proiectului

În această arhitectură structurală se regăsește și arhitectura MVC (Model View Controller), care împarte aplicația în trei părți:

* **Componentele Model:** încapsulează datele de bază și funcționalitățile
* **Componentele View:** afișează informații utilizatorului, obține datele pe care le afișează de la Model
* **Controller:** Fiecărui View îi este asociată o componentă Controller. Controllerele primesc intrarea de obicei sub forma unor evenimente care denotă mișcarea mouse-ului, activarea butoanelor mouse-ului sau intrarea de la tastatură. Evenimentele sunt traduse în cereri de service, care sunt trimise fie la Model sau la View

Diagrama de clase:



**4. Implementare**

Pentru înțelegerea corectă și bună a funcționalității aplicației de procesare a comenzilor unui depozit, în acestă secțiune se vor descrie detaliat clasele menționate în secțiunea anterioară.

* **Clasa Client:** Această clasă este utilizată pentru a stoca datele referitoare la un client, date

existente în baza de date. Atributele clasei sunt: ID, name, email.

Clasa conține constructori, gettere și settere, și metoda toString, care ajută la afișarea clienților din baza de date.

* **Clasa Product:** Această clasă este utilizată pentru a stoca datele referitoare la un produs, date existente în baza de date. Atributele clasei sunt: ID, name, quantity, price. Clasa conține contructori, gettere și settere.
* **Clasa Order:** Această clasă este utilizată pentru a stoca datele referitoare la o comandă, date existente în baza de date. Atributele clasei sunt: ID, idClient, idProdus, price, quantity. Clasa conține constructori, gettere și settere.
* **Clasa Bill:** Această clasă imutabila este utilizată pentru a stoca datele referitoare la o comandă, date existente în baza de date. Atributele clasei sunt: ID, idClient, idProdus, price, quantity. Clasa conține constructori, gettere și settere.
* **Clasa AbstractDAO:** Aceasta clasa contine operatiile de baza pentru fiecare tabel (findById, findAll, delete). Pe lângă aceste metode, se regăsesc și metodele care creează interogările de SELECT și DELETE. Metodele mai semnificative:

**-List<T> findAll():** Aceasta metoda returneaza o lista de obiecte ce contine toate rândurile unui tabel. Prima dată se realizează conexiunea la baza de date, se creează un statement cu textul interogării, și dacă rezultatul nu e null, acesta se returnează.

**-List<T> createObjects(resultSet):** Acesastă metodă creează lista de rezultat, conținând elemente de tipul obiectului care se află în resultSet.

-**T findById(int id):** Această metodă returnează obiectul din baza de date cu ID-ul primit ca și argument. Prima dată se realizează conexiunea la baza de date, se creează un statement cu textul interogării, și dacă rezultatul nu e null, acesta se returnează.

* **Clasa ClientDAO:** această clasă conține operațiile de baza specifice unui client (findByName, findByEmail, deleteByNameOrMail, insert, updateByID, updateByNameOrMail). Pe lângă aceste metode, se regăsesc și metodele care creează interogările de SELECT, INSERT și UPDATE. Metodele mai semnificative:

**-List<Client> findByName(String name):** Această metodă returnează o listă de clienți care au numele dat ca și argument. Prima dată se realizează conexiunea la baza de date, se creează un statement de SELECT, și dacă rezultatul nu e null, acesta se returnează.

**-Client findByEmail(String name):** Această metodă returnează clientul care are emailul dat ca și argument. Prima dată se realizează conexiunea la baza de date, se creează un statement de SELECT, și dacă rezultatul nu e null, acesta se returnează.

-**void deleteByNameOrMail(String field, String value):** Această metodă se folosește atât la ștergerea clienților după nume, cât și la ștergerea după email. Metoda realizează ștergerea unui client, care are un câmp (field) cu valoarea primită ca și argument (value).

-**Client insert(String name, String email):** Această metodă inserează un client în baza de date, cu numele și mailul primit ca și argument. ID-ul nu trebuie introdus de către utilizator, deoarece acela se auto-incrementează în baza de date. În metodă, prima dată se realizează conexiunea la baza de date, se creează un statement de INSERT cu metoda createInsertStatement(), și dacă clientul inserat nu e null, acesta se returnează.

- **Client updateByID(String field, Integer value, String name, String email):** Această metodă modifică un client, care are câmpul field cu valoarea value în baza de date. Acest client primește valorile din argument. ID-ul nu trebuie introdus de către utilizator, deoarece acela se auto-incrementează în baza de date. În metodă, prima dată se realizează conexiunea la baza de date, se creează un statement de UPDATE cu metoda createInsertStatement(), și dacă clientul inserat nu e null, acesta se returnează.

- **Client updateByNameOrEmail(String field, String value, String name, String email):** Această metodă modifică un client, care are câmpul field cu valoarea value în baza de date. Acest client primește valorile din argument. ID-ul nu trebuie introdus de către utilizator, deoarece acela se auto-incrementează în baza de date. În metodă, prima dată se realizează conexiunea la baza de date, se creează un statement de UPDATE cu metoda createUpdateStatement(), și dacă clientul inserat nu e null, acesta se returnează.

* **Clasa ProdusDAO:** această clasă conține operațiile de baza specifice unui produs (findByName, findByQuantity, findByQPrice, deleteByName, deleteByPrice, insert, updateByIDorQuantity, updateByName, updateByPrice). Pe lângă aceste metode, se regăsesc și metodele care creează interogările de SELECT, INSERT si UPDATE. Metodele mai semnificative:

**-produs findByName(String name):** metoda returnează produsul cu numele name. Prima dată se realizează conexiunea la baza de date, se creează un statement de SELECT, și dacă rezultatul nu e null, acesta se returnează.

- **produs findByPrice(Double price):** returnează produsul cu prețul primit ca și argument. Prima dată se realizează conexiunea la baza de date, se creează un statement de SELECT, și dacă rezultatul nu e null, acesta se returnează.

- **produs findByQuantity(Integer quantity):** returnează produsul cu prețul primit ca și argument. Prima dată se realizează conexiunea la baza de date, se creează un statement de SELECT, și dacă rezultatul nu e null, acesta se returnează.

-**void deleteByName(String field, String value):** șterge produsul care are la câmpul field valoarea value.

- **void deleteByPrice(String field, Double value):** șterge produsul care are la câmpul field valoarea value.

-**Produs insert(String name, int quantity, double price):** Această metodă inserează un produs în baza de date, cu valorile primite ca și argument. ID-ul nu trebuie introdus de către utilizator, deoarece acela se auto-incrementează în baza de date. În metodă, prima dată se realizează conexiunea la baza de date, se creează un statement de INSERT cu metoda createInsertStatement(), și dacă produsul inserat nu e null, acesta se returnează

**-Produs updateByIDOrQuantity(String field, integer value, String name, Integer quantity, Double price):** Această metodă mdofică produsul cu câmpul de valoarea value, primind noile date primite ca și argument. În metodă, prima dată se realizează conexiunea la baza de date, se creează un statement de UPDATE cu metoda createUpdateStatement(), și dacă produsul modificat nu e null, acesta se returnează

**-Produs updateByName(String field, String value, String name, Integer quantity, Double price):** Această metodă modifică produsul cu câmpul de valoarea value, primind noile date primite ca și argument

**-Produs updateByPrice(String field, Double value, String name, Integer quantity, Double price):** Această metodă mdofică produsul cu câmpul de valoarea value, primind noile date primite ca și argument

* **Clasa OrdersDAO:** această clasă conține operațiile de baza specifice unei comenzi (findByClientID, findByProdusID, updateByID, insert). Pe lângă aceste metode, se regăsesc și metodele care creează interogările de SELECT, INSERT si UPDATE. Metodele mai semnificative:

**-List<Orders> findByClientID(int id):** returnează comanda care are id-ul de client ca și id-ul primit ca parametru.

**-List<Orders> findByProductID(int id):** returnează comanda care are id-ul de produs ca și id-ul primit ca parametru.

**-Orders updateByID(String field, int value, int idClient, int idProdus, double price, int quantity):** modifică comanda care are fieldul cu valoarea value, primind noile valori din argument.

-**Order insertOrder(int idClient, int idProdus, double price, int quantity):** introduce o nouă comandă în baza de date.

* **Clasa LogDAO:** această clasă conține operațiile de baza specifice unei comenzi (findByClientID, findByProdusID, updateByID, insert). Pe lângă aceste metode, se regăsesc și metodele care creează interogările de SELECT, INSERT. Metodele mai semnificative:

**-List<Bill> findByClientID(int id):** returnează factura pentru comanda care are id-ul de client ca și id-ul primit ca parametru.

**-List<Bill> findByProductID(int id):** returnează factura pentru comanda care are id-ul de produs ca și id-ul primit ca parametru.

-**Bill insertBill(int idClient, int idProdus, double price, int quantity):** introduce o nouă factura comandă în baza de date.

* **Clasa ClientBLL:** Metodele acestei clase se folosesc de obiectul ClientDAO prin care se apeleaza metodele din clasa respectiva. Clasa are ca și câmpuri o listă de validatori, și un obiect de ClientDAO.

Clasa are metodele menționate și în clientDAO. În plus, metodele din ClientBLL verifică cu validatorii clienții care au fost inserați/modificați.

* **Clasa ProdusBLL:** Metodele acestei clase se folosesc de obiectul ProdusDAO prin care se apeleaza metodele din clasa respectiva. Clasa are ca și câmpuri o listă de validatori, și un obiect de ProdusDAO.

Clasa are metodele menționate și în ProdusDAO. În plus, metodele din ProdusBLL verifică cu validatorii produsele care au fost inserate/modificate.

* **Clasa OrdersBLL:** Metodele acestei clase se folosesc de obiectul OrderDAO prin care se apeleaza metodele din clasa respectiva. Clasa are ca și câmpuri o listă de validatori, și un obiect de OrderDAO.

Clasa are metodele menționate și în OrderDAO. În plus, metodele din OrderBLL verifică cu validatorii comenzile care au fost inserate/modificate.

* **Clasa LogBLL:** Metodele acestei clase se folosesc de obiectul LogDAO prin care se apeleaza metodele din clasa respectiva. Clasa are ca și câmpuri o listă de validatori, și un obiect de LogDAO.

Clasa are metodele menționate și în LogDAO.

* **Clasa EmailValidator:** Această clasă verifică dacă email-ul unui client are format de email, adică dacă respectă email patternul. Dacă nu, aruncă o excepție. Are o singură metodă de validate (metoda interfetei Validator) care efectuează verificarea.
* **Clasa QuantityValidator:** Această clasă verifică dacă cantitatea unui produs este mai mare decât 0. Dacă nu este, aruncă o excepție. Are o singură metodă de validate (metoda interfetei Validator) care efectuează verificarea.
* **Clasa OrderQuantityValidator:** Această clasă verifică dacă cantitatea produsului unei comenzi este mai mare decât 0. Dacă nu este, aruncă o excepție. Are o singură metodă de validate (metoda interfetei Validator) care efectuează verificarea.
* **Clasa View:** În această clasă este implementată interfața utilizatorului. Clasa extinde JFrame. Clasa are mai multe metode, fiecare efectuând deschiderea unei noi ferestre, în funcție de operație pe care dorim să o efectuăm.
* **Clasa Controller:** În această clasă sunt implementate funcționalitățile butoanelor din GUI. Ea implementează interfața ActionListener. Are o metodă importantă:

- **void actionPerformed(ActionEvent e):** metodă specifică interfeței ActionListener, ce trebuie implementată în mod obligatoriu. În această metodă se verifică cu câte un if, care buton s-a apăsat, și în funcție de butonul apăsat, se execută acțiunile necesare

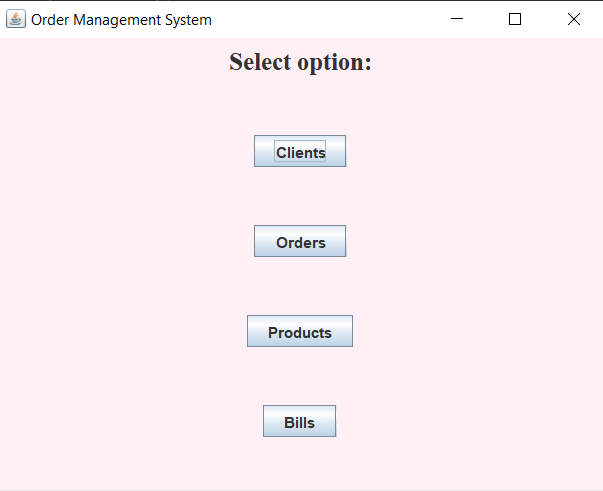
* **Clasa Reflection:** În această clasă implementează formatul de tabel a bazei de date, atunci când se vizualizează datele în GUI. Are o metodă foarte importantă:

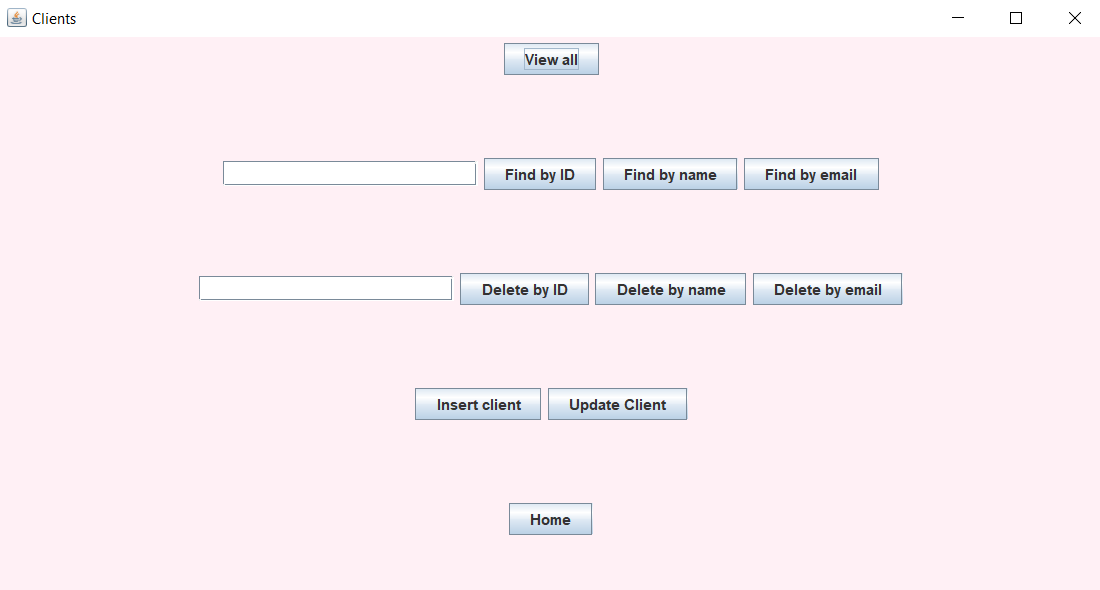
O imagine care conține text

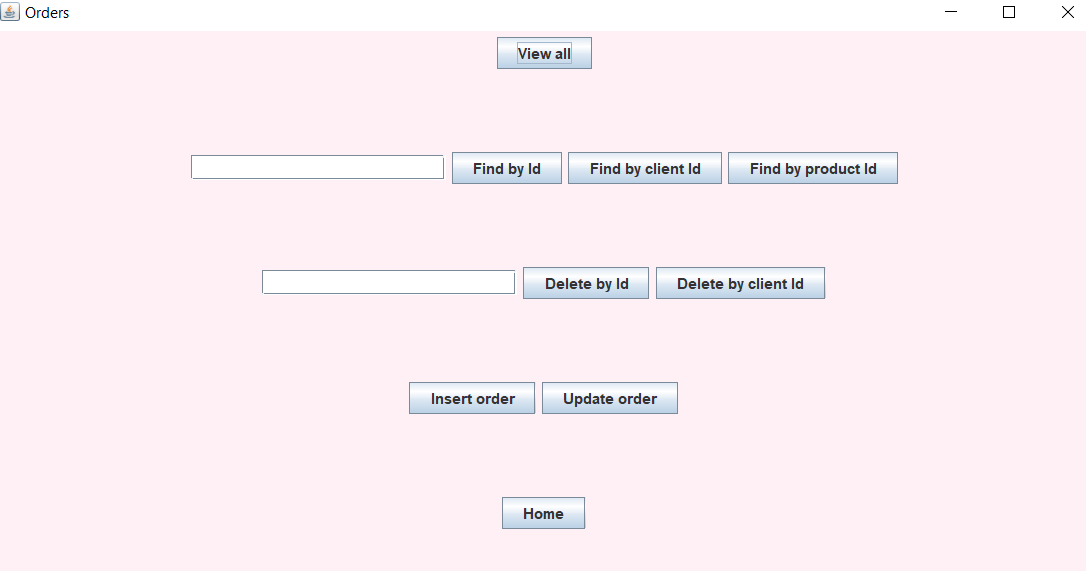
Descriere generată automat**-<T> void getRows(List <T> list) :** această metodă primeste ca si parametru o lista de rezultat și il inserează intr-un tabel. Parcurge cu 2 foruri lista de obiecte, si folosindu-se de o variabila booleana firstIteration verifica daca se afla la primul obiect din lista. Daca da, atunci ia fieldurile lui si le pune in tabel ca si nume de coloane, iar la restul iteratiilor nu mai face acest lucru (variabila firstIteration se face false).

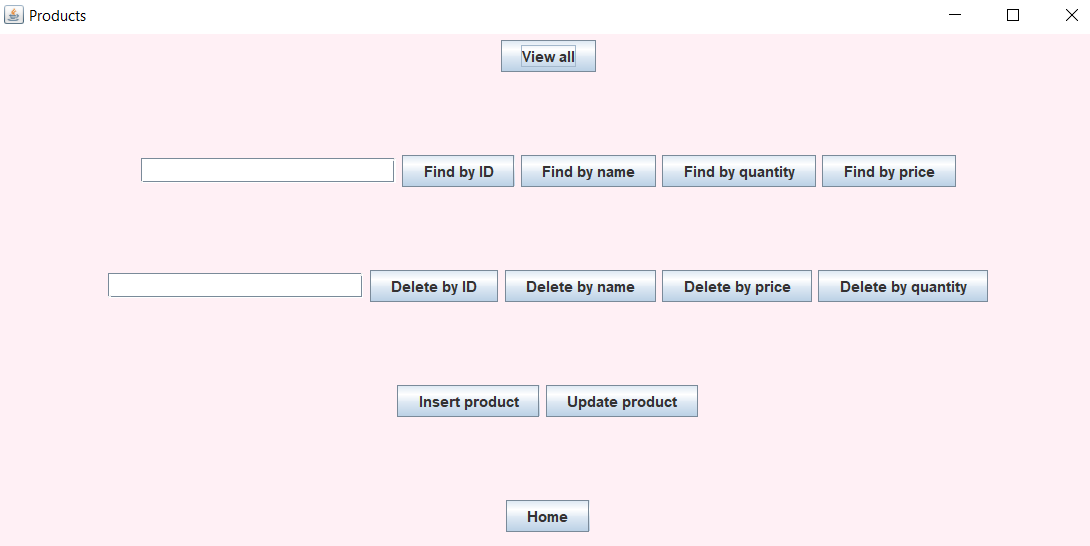
* **Clasa connectionFactory:** aceasta clasa realizeaza conexiunea proiectului la baza de date din MySQL.

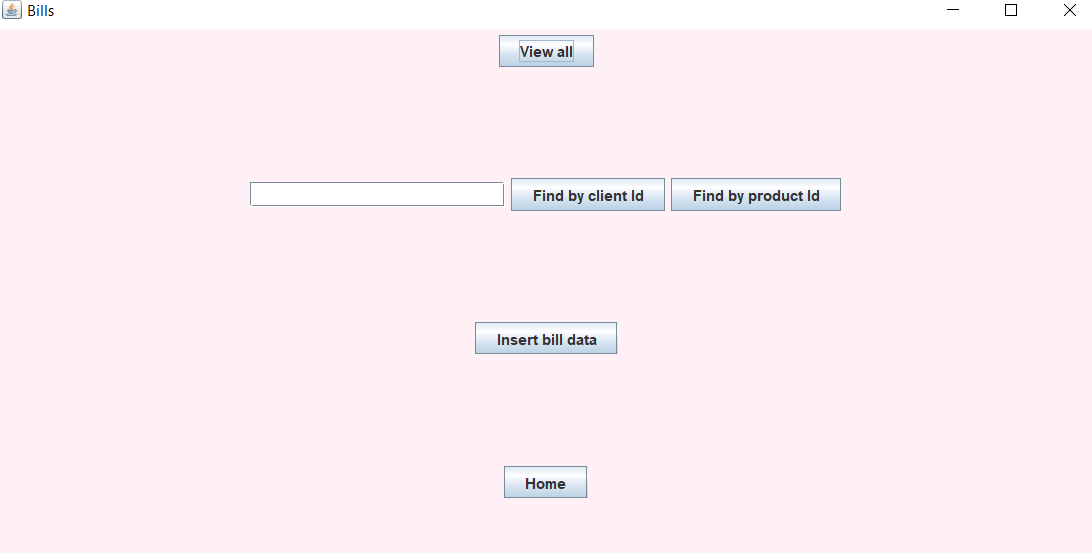
Interfața grafică:











**5. Rezultate:**

O imagine care conține masă

Descriere generată automat

O imagine care conține masă

Descriere generată automat

O imagine care conține masă

Descriere generată automat

**6. Concluzii:**

Ca orice proiect, și acesta a avut dificultățile lui, din care se pot trage concluzii și idei, care ne pot fi utile în implementarea altor proiecte în viitor.

Aceasta tema m-a ajutat sa imi dezvolt cunostintele. Am invatat multe lucruri noi cum ar fi: sa fac conexiunea la o baza de date, sa fac interogari, sa folosesc reflexia si sa generez cu JavaDoc.

**7. Bibliografie:**

<https://docs.oracle.com/javase/tutorial/uiswing/>

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

<https://www.oracle.com/ro/technical-resources/articles/java/javadoc-tool.html>

<https://jenkov.com/tutorials/java-reflection/index.html>