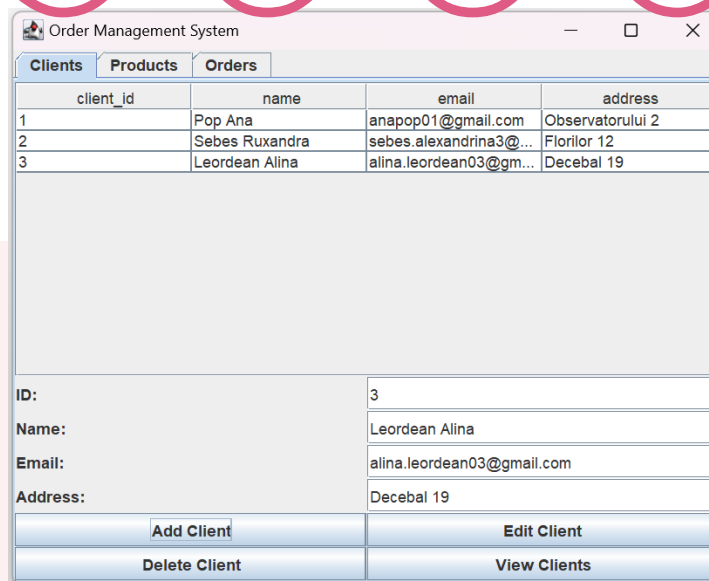


DOCUMENTATIE

TEMA 3

Order Management System



The screenshot displays the 'Order Management System' window. It features three tabs: 'Clients', 'Products', and 'Orders'. The 'Clients' tab is active, showing a table with the following data:

client_id	name	email	address
1	Pop Ana	anapop01@gmail.com	Observatorului 2
2	Sebes Ruxandra	sebes.alexandrina3@...	Florilor 12
3	Leordean Alina	alina.leordean03@gm...	Decebal 19

Below the table, there is a form for editing a client. The form fields are as follows:

- ID: 3
- Name: Leordean Alina
- Email: alina.leordean03@gmail.com
- Address: Decebal 19

At the bottom of the form, there are four buttons: 'Add Client', 'Edit Client', 'Delete Client', and 'View Clients'.

NUME STUDENT: Leordean Alina-Anuta
GRUPA: 30228

CUPRINS

1. Obiectivul temei	Error! Bookmark not defined.
2. Analiza problemei, modelare, scenarii, cazuri de utilizare	Error! Bookmark not defined.
3. Proiectare.....	4
4. Implementare.....	5
5. Rezultate.....	6
6. Concluzii	8
7. Bibliografie.....	8

1. Obiectivul temei



Obiectivul acestui proiect este dezvoltarea unei aplicații de gestionare a comenzilor pentru un depozit, utilizând baze de date relaționale pentru stocarea informațiilor despre produse, clienți și comenzi. Aplicația va permite utilizatorilor să efectueze operațiuni esențiale de gestionare a comenzilor, cum ar fi adăugarea, actualizarea și ștergerea produselor și comenzilor, precum și gestionarea informațiilor despre clienți. Scopul principal al proiectului este de a crea un sistem eficient și ușor de utilizat pentru administrarea stocurilor și a comenzilor, asigurându-se că datele sunt gestionate în mod corect și actualizat în timp real.

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



Cazuri funcționale

- Sistemul trebuie să permită adăugarea, actualizarea și ștergerea clienților.
- Sistemul trebuie să permită adăugarea, actualizarea și ștergerea produselor.
- Sistemul trebuie să permită crearea, actualizarea și ștergerea comenzilor.
- Sistemul trebuie să gestioneze stocul produselor și să actualizeze stocurile în funcție de comenzile plasate.
- Sistemul trebuie să afișeze listele de clienți, produse și comenzi.

Cazuri non-funcționale

- Sistemul trebuie să fie ușor de utilizat, cu o interfață grafică intuitivă.
- Sistemul trebuie să fie fiabil și să gestioneze corect datele.
- Performanța sistemului trebuie să fie adecvată, răspunzând prompt la acțiunile utilizatorului.
- Sistemul trebuie să fie scalabil, permițând adăugarea de noi funcționalități în viitor.

Scenariul de succes:

- Utilizatorul accesează interfața aplicației și navighează la secțiunea dorită (Clienți, Produse, Comenzi).
- Utilizatorul selectează opțiunea de adăugare a unui nou client.
- Utilizatorul completează informațiile necesare și confirmă adăugarea.
- Sistemul validează datele și adaugă clientul în baza de date.
- Utilizatorul primește un mesaj de confirmare a succesului operațiunii.

Scenariul alternativ:

- Utilizatorul accesează interfața aplicației și navighează la secțiunea dorită (Clienți, Produse, Comenzi).
- Utilizatorul selectează opțiunea de adăugare a unui nou client.
- Utilizatorul completează informațiile necesare, dar introduce date incorecte sau incomplete.
- Sistemul validează datele și detectează erorile.
- Utilizatorul primește un mesaj de eroare și este solicitat să corecteze informațiile.
- Utilizatorul corectează datele și reîncearcă adăugarea clientului.
- Sistemul validează datele corecte și adaugă clientul în baza de date.
- Utilizatorul primește un mesaj de confirmare a succesului operațiunii..

3. Proiectare

Aplicația de gestionare a comenzilor pentru depozit este proiectată conform unui model de arhitectură stratificată (layered architecture pattern), care separă responsabilitățile în mai multe straturi distincte. Această abordare permite o mai bună organizare a codului, facilitând întreținerea, scalabilitatea și testarea aplicației. Fiecare strat are un rol specific și interacționează cu celelalte straturi pentru a asigura funcționarea corectă a aplicației.

Pachetele:

- BusinessLogic: conține clasele care implementează logica de afaceri a aplicației. Aceste clase gestionează procesele și regulile de afaceri care trebuie aplicate datelor de model
- Connection: Pachetul conține clasele necesare pentru gestionarea conexiunilor la baza de date. DataAccess: Pachetul conține clasele care se ocupă de interacțiunea directă cu baza de date. Aceste clase sunt responsabile pentru operațiunile CRUD (Create, Read, Update, Delete) și pentru gestionarea conexiunilor la baza de date.
- Model: Pachetul conține clasele care reprezintă obiectele de date ale aplicației. Aceste clase corespund entităților din baza de date și definesc structura obiectelor utilizate în cadrul aplicației.
- Presentation: Pachetul conține clasele responsabile pentru interfața cu utilizatorul (GUI). Aceste clase definesc ferestrele, formularele și elementele grafice ale aplicației.

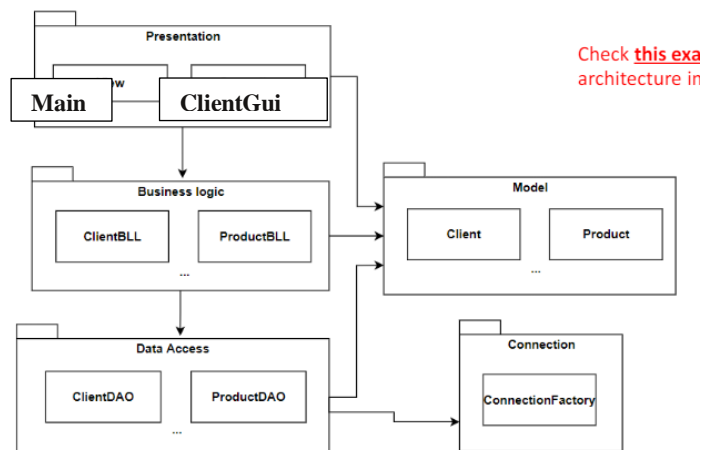
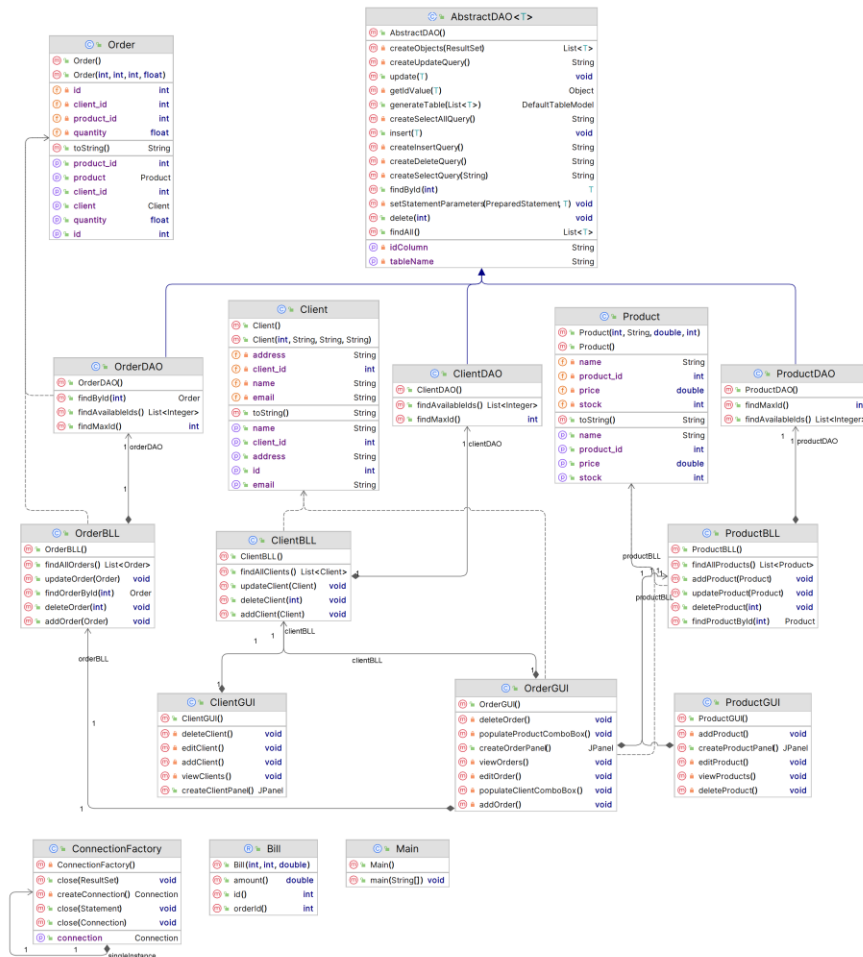


Diagrama UML a claselor:

Diagrama UML a claselor oferă o imagine grafică a structurii și relațiilor dintre clasele din aplicație. Scopul acestei diagrame este de a ilustra cum sunt organizate clasele în pachetele definite și cum interacționează între ele pentru a realiza funcționalitățile sistemului.



4. Implementare

- Clasa **Client** din pachetul Model:

Client.java: Reprezintă entitatea Client, cu atribute precum client_id, name, și address. Este utilizată pentru a stoca și manipula informațiile despre clienți în cadrul aplicației.

- Clasa **Product** din pachetul Model:

Product.java: Reprezintă entitatea Product, cu atribute precum product_id, name, price, și stock. Este utilizată pentru a stoca și manipula informațiile despre produse.

- Clasa **Order** din pachetul Model:

Order.java: Reprezintă entitatea Order, cu atribute precum id, client_id, product_id, și quantity. Este utilizată pentru a stoca și manipula informațiile despre comenzi.

- Clasa **Bill** din pachetul Model:

Bill.java: Reprezintă entitatea Bill utilizând Java Records, cu attribute precum id, orderId, și amount. Este utilizată pentru a stoca informații despre facturile generate pentru comenzi.

- Clasa **ClientBLL** din pachetul BusinessLogic:

ClientBLL.java: Contine logica de afaceri pentru gestionarea clienților. Metodele acestei clase permit adăugarea, actualizarea, ștergerea și vizualizarea clienților. Utilizează ClientDAO pentru interacțiunea cu baza de date..

- Clasa **ProductBLL** din pachetul BusinessLogic:

ProductBLL.java: Contine logica de afaceri pentru gestionarea produselor. Metodele acestei clase permit adăugarea, actualizarea, ștergerea și vizualizarea produselor. Utilizează ProductDAO pentru interacțiunea cu baza de date.

- Clasa **OrderBLL** din pachetul BusinessLogic:

OrderBLL.java: Contine logica de afaceri pentru gestionarea comenzilor. Metodele acestei clase permit adăugarea, actualizarea, ștergerea și vizualizarea comenzilor. Utilizează OrderDAO pentru interacțiunea cu baza de date.

- Clasa **ConnectionFactory** din pachetul DataAccess:

ConnectionFactory.java: Oferă metode pentru gestionarea conexiunilor la baza de date. getConnection() returnează o conexiune la baza de date, iar close() închide conexiunile, declarațiile și seturile de rezultate.

- Clasa **AbstractDAO** pachetul DataAccess:

AbstractDAO.java: Clasa abstractă generică pentru operațiuni CRUD (Create, Read, Update, Delete). Utilizează tehnica reflexion pentru a genera interogări SQL și a maparea rezultatelor seturilor de date la obiectele model. Reflexion este utilizată pentru a obține numele coloanelor și valorile acestora din obiectele generice T.

- Clasa **ClientDAO , ProductDAO , OrderDAO** din pachetul DataAccess:

ClientDAO.java, ProductDAO.java, OrderDAO.java: Extind AbstractDAO pentru a oferi funcționalități specifice entităților Client, Product, și Order.

- Clasa **ClientGUI** din pachetul Presentation:

ClientGUI.java: Oferă interfața grafică pentru gestionarea clienților. Utilizatorii pot adăuga, edita, șterge și vizualiza clienți. Interacționează cu ClientBLL pentru logica de afaceri și utilizează JTable pentru afișarea datelor.

- Clasa **ProductGUI** din pachetul Presentation:

ProductGUI.java: Oferă interfața grafică pentru gestionarea produselor. Utilizatorii pot adăuga, edita, șterge și vizualiza produse. Interacționează cu ProductBLL pentru logica de afaceri și utilizează JTable pentru afișarea datelor.

- Clasa **OrderGUI** din pachetul Presentation:

OrderGUI.java: Oferă interfața grafică pentru gestionarea comenzilor. Utilizatorii pot adăuga, edita, șterge și vizualiza comenzi. Interacționează cu OrderBLL pentru logica de afaceri și utilizează JTable pentru afișarea datelor. De asemenea, permite selecția produselor și clienților din listele derulante (JComboBox).

Utilizarea Tehnicii Reflexion

Tehnica reflexion este utilizată în principal în clasa AbstractDAO pentru a permite operațiuni generice pe diferite entități de model. Reflexion permite:

Crearea dinamică a interogărilor SQL pe baza numelor de coloane obținute din câmpurile clasei model.

Maparea rezultatelor seturilor de date la obiectele model prin accesarea și setarea valorilor câmpurilor prin reflexie.

Generarea antetelor de tabel și popularea datelor din tabelele grafice (JTable) în mod dinamic.

Rolul reflexion este de a simplifica și generaliza operațiunile CRUD, evitând astfel codul redundant și specific fiecărei entități, făcând codul mai flexibil și ușor de întreținut.

5. Rezultate



Implementarea aplicației de Management al Comenzilor a fost realizată cu succes, folosind o arhitectură stratificată care a asigurat o separare clară a responsabilităților. Pachetul Model a definit structura datelor, BusinessLogic a gestionat logica de afaceri, DataAccess a facilitat interacțiunea cu baza de date folosind tehnica reflexion, iar Presentation a oferit o interfață grafică intuitivă pentru utilizatori. Aceste elemente au permis realizarea eficientă a operațiunilor CRUD și gestionarea comenzilor, produselor și clienților în mod organizat și robust.

6. Concluzii



Proiectul a demonstrat eficiența unei arhitecturi bine structurate în dezvoltarea aplicațiilor complexe, evidențiind beneficiile utilizării reflexiei pentru operațiuni generice și reducerea redundanței codului. Interfața grafică a îmbunătățit experiența utilizatorului, facilitând gestionarea datelor.

7. Bibliografie



1. JavaDoc, <https://www.baeldung.com/javadoc>
2. OracleHelpCenter, <https://docs.oracle.com/javase/8/docs/api/java/sql/package-summary.html>
3. MySQL, <https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html>
4. GeeksforGeeks, <https://docs.oracle.com/javase/8/docs/api/java/sql/package-summary.html>