Facultatea de Automatică și Calculatoare

**Documentație**

**Order Management**

-Tema III-

Nume student: Ivan Alexandra-Ștefania

Grupa: Grupa 30227

**CUPRINS**

[1. Obiectivul temei 3](file:///C:\Users\Stefi\Downloads\PT2023_Template_Documentatie_RO.doc#_Toc95297885)

* 1. Obiectiv principal …………………………………………...3

1.2 Obiective secundare……………………………………........3

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 4](file:///C:\Users\Stefi\Downloads\PT2023_Template_Documentatie_RO.doc#_Toc95297886)

[3. Proiectare 5](file:///C:\Users\Stefi\Downloads\PT2023_Template_Documentatie_RO.doc#_Toc95297887)

3.1 Diagrama UML de clase ……………………………………...6

[4. Implementare 6](file:///C:\Users\Stefi\Downloads\PT2023_Template_Documentatie_RO.doc#_Toc95297888)

4.1 Clasele ClientBll, ProductBll, OrderBll.………………………6

4.2 Clasa AbstractDAO.…………………………………………...8

4.3 Clasele Client, Product, Orders .………………………………8

4.4 Clasa ConnectionFactory…….…..…………………..………10

4.5 Clasa Controller …………………………………..…………10

4.6 Clasa View…… …………………………………..…………11

[5. Rezultate 12](file:///C:\Users\Stefi\Downloads\PT2023_Template_Documentatie_RO.doc#_Toc95297889)

[6. Concluzii 12](file:///C:\Users\Stefi\Downloads\PT2023_Template_Documentatie_RO.doc#_Toc95297890)

[7. Bibliografie 13](file:///C:\Users\Stefi\Downloads\PT2023_Template_Documentatie_RO.doc#_Toc95297891)

1. **Obiectivul temei**

1.1 **Obiectivul principal** al aplicației de gestionare a comenzilor pentru un depozit este de a permite procesarea comenzilor clientului prin utilizarea unei baze de date relaționale. Aceasta implică stocarea informațiilor despre **produse**, **clienți** și **comenzile** plasate de clienți într-o bază de date relațională.

1.2 **Obiectivele secundare** ale aplicației de gestionare a comenzilor pentru un depozit pot include:

**Analiza problemei și identificarea cerințelor** - Obiectivul este de a înțelege problema și cerințele asociate gestionării comenzilor într-un depozit.

**Proiectarea aplicației de gestionare a comenzilor** - Obiectivul este de a proiecta arhitectura și structura aplicației de gestionare a comenzilor. Aceasta implică definirea componentelor, proiectarea interfețelor necesare pentru a îndeplini cerințele identificate anterior.

**Implementarea aplicației de gestionare a comenzilor** - Obiectivul este de a implementa componentele și funcționalitățile definite în faza de proiectare.

**Testarea aplicației de gestionare a comenzilor** - Obiectivul este de a verifica și valida funcționalitatea și performanța aplicației.

**Documentarea și mentenanța aplicației de gestionare a comenzilor** - Obiectivul este de a documenta în mod corespunzător aplicația și procesele asociate, astfel încât aceasta să poată fi înțeleasă cu ușurință de utilizatori.

1. **Analiza problemei**

**Cerințe funcționale**:

* Aplicația trebuie să permită înregistrarea și gestionarea informațiilor

despre clienți, inclusiv nume, adrese și alte detalii relevante.

* Aplicația trebuie să permită înregistrarea și gestionarea informațiilor despre produse, inclusiv numele produsului, preț, stoc disponibil, etc. Trebuie să existe funcționalități pentru adăugarea, actualizarea și ștergerea produselor din sistem.
* Utilizatorii (clienții) trebuie să poată plasa comenzile prin intermediul aplicației. Aceasta poate implica selectarea produselor dorite și specificarea cantității.
* Sistemul trebuie să verifice disponibilitatea produselor în stoc în momentul plasării comenzii și să afișeze utilizatorului informații actualizate privind disponibilitatea și stocurile.
* Aplicația trebuie să ofere funcționalități pentru procesarea comenzilor, cum ar fi generarea facturii.

**Cerințe non-funcționale:**

* Aplicația trebuie să ofere o performanță rapidă și eficientă, astfel încât să permită procesarea comenzilor în timp real și să ofere o utilizare ușoară.
* Sistemul trebuie să fie sigur și să ofere protecție pentru datele clienților, comenzilor și produselor.
* Interfața aplicației trebuie să fie intuitivă și ușor de utilizat, astfel încât să se poată plasa comenzile și gestiona informațiile fără dificultate.

1. **Proiectare**

Proiectul trebuie să respecte arhitectura stratificată (layered architecture). Este important să separăm responsabilitățile și să organizăm componentele aplicației în straturi distincte. Iată cum ar putea fi structurată aplicația în cadrul arhitecturii stratificate:

* **Presentation Layer:**

Responsabil pentru interfața utilizatorului și interacțiunea cu acesta.

Include componente precum interfețe grafice, pagini web, servicii web sau interfețe de linie de comandă.

Se concentrează pe prezentarea datelor și interacțiunea cu utilizatorul.

* **Business Logic Layer:**

Responsabil pentru implementarea logicii aplicației.

Include componente precum servicii, clase de procesare a datelor sau reguli de validare.

Se ocupă de manipularea și prelucrarea datelor în conformitate cu regulile.

* **Data Access Layer:**

Responsabil pentru interacțiunea cu baza de date.

* 1. **Diagrama UML de clase**

Diagrama UML de clase a proiectului este prezentată în imaginea de mai

jos:

A screenshot of a computer

Description automatically generated with medium confidence

1. **Implementare**

**4.1 ClientBll, ProdusBll, OrderBll**

**Clasele ClientBll, ProdusBll și OrderBll** sunt componente ale Business Logic Layer în cadrul arhitecturii stratificate (layered architecture) și sunt responsabile de implementarea logicii și gestionarea operațiilor legate de clienți, produse și comenzi.

**ClientBll:** Această clasă este responsabilă de gestionarea operațiilor legate de clienți.

**ProdusBll:** Această clasă se ocupă de operațiile legate de produse.

**OrderBll:** Această clasă se ocupă de gestionarea operațiilor legate de comenzi.

Fiecare clasă implementează următoarele metode:

* Delete: Această metodă este responsabilă pentru ștergerea unui obiect din sistem. Prin intermediul obiectului DAO, se apelează metoda delete pentru a efectua operația de ștergere în sursa de date.
* findById(int id): Această metodă are rolul de a căuta un obiect în sistem în funcție de un identificator specificat. Prin intermediul obiectului DAO, se apelează metoda findById pentru a obține obiectul corespunzător identificatorului dat. Dacă nu există un obiect cu identificatorul respectiv, se aruncă o excepție de tip NoSuchElementException.
* updateById(): Această metodă se ocupă de actualizarea informațiilor unui obiect existent în sistem. Prin intermediul obiectului DAO, se apelează metoda update pentru a actualiza datele în baza de date.
* insert(): Această metodă este responsabilă de adăugarea unui nou obiect în sistem. Se apelează metoda insert a obiectului DAO pentru a adăuga obiectul în baza de date. În caz contrar, dacă una din validări eșuează, se aruncă o excepție corespunzătoare.
* findAll(): Această metodă returnează o listă cu toate obiectele de un anumit tip din sistem. Prin intermediul obiectului DAO, se apelează metoda findAll pentru a obține lista din baza de date.
* getDataForTable(): Această metodă returnează un obiect de tip DataForTable, care conține informațiile necesare pentru construirea unei tabele cu obiecte de un anumit tip. Prin intermediul metodei findAll a obiectului DAO, se obține lista, iar apoi se apelează metoda getDataForTable a obiectului DAO pentru a crea obiectul DataForTable pe baza listei.

Aceste clase Bll (Business Logic Layer) reprezintă un strat intermediar între prezentare (Presentation Layer) și accesul la date (Data Access Layer). Prin separarea logicilor de celelalte straturi ale aplicației, se obține reutilizare și testabilitate.

**4.2 AbstractDao**

**Clasa AbstractDAO** este o clasă generică în Java, care servește drept clasă de bază pentru alte clase DAO (Data Access Object) specializate. Scopul său principal este de a furniza metode comune pentru operațiile de bază de acces la date, cum ar fi selectarea, inserarea, actualizarea și ștergerea datelor în baza de date.

**4.3 Client, Product, Orders**

**A screen shot of a computer

Description automatically generated with medium confidence**

**A screen shot of a computer

Description automatically generated with medium confidence**

**A screen shot of a computer program

Description automatically generated with low confidence**

**4.4 ConnectionFactory**

Clasa ConnectionFactory este responsabilă de stabilirea unei conexiuni la baza de date folosind JDBC (Java Database Connectivity).

**4.5 Controller**

**Clasa Controller** este responsabilă de gestionarea evenimentelor și

interacțiunilor utilizatorului cu aplicația de tip warehouse. Aceasta este legată de clasa View și are o referință la obiecte BLL (Business Logic Layer) pentru Client, Product și Order.

Constructorul clasei primește o instanță a clasei View pentru a gestiona interfețele grafice.

Se creează obiectele BLL pentru Client, Product și Order pentru a putea interacționa cu baza de date.

În constructor, se adaugă ascultători pentru diferite evenimente ale interfeței grafice.

Pentru secțiunea Products, se adaugă ascultători pentru adăugarea, editarea, ștergerea și vizualizarea produselor. Acești ascultători preiau datele introduse de utilizator prin interfața grafică și utilizează obiectul BLL ProductBLL pentru a executa operațiile corespunzătoare în baza de date.

Pentru secțiunea Clients, se adaugă ascultători pentru adăugarea, editarea, ștergerea și vizualizarea clienților Acești ascultători preiau datele introduse de utilizator prin interfața grafică și utilizează obiectul BLL ClientBLL pentru a executa operațiile corespunzătoare în baza de date.

Pentru secțiunea Orders, se adaugă ascultători pentru adăugarea, editarea, ștergerea și vizualizarea comenzilor. Acești ascultători preiau datele introduse de utilizator prin interfața grafică și utilizează obiectul BLL OrderBLL pentru a executa operațiile corespunzătoare în baza de date.

Astfel, clasa Controller leagă interfața grafică a aplicației de obiectele BLL și gestionează acțiunile utilizatorului în funcție de evenimentele generate de interfață.

**4.6 View**

Această clasă extinde clasa "JFrame" și reprezintă interfața grafică (GUI) a unei aplicații pentru gestionarea comenzilor.

Constructorul "View()" creează frame-ul principal al aplicației pentru gestionarea comenzilor.

Metoda "addProducts()" permite adăugarea unui ActionListener pentru butonul "Products".

Metoda "addClients()" permite adăugarea unui ActionListener pentru butonul "Clients".

Metoda "addOrders()" permite adăugarea unui ActionListener pentru butonul "Orders".

Metoda "productsFrame()" afișează frame-ul pentru gestionarea produselor.

Metoda "clientsFrame()" afișează frame-ul pentru gestionarea clienților.

Metoda "ordersFrame()" afișează frame-ul pentru gestionarea comenzilor.

Metodele "addClient()", "editClient()", "deleteClient()", "viewClients()", "addProduct()", "editProduct()", "deleteProduct()", "viewProducts()", "addOrder()", "editOrder()", "deleteOrder()" și "viewOrders()" permit adăugarea de ActionListeners pentru butoanele corespunzătoare.

Metoda "setTotal()" afișează un frame separat pentru afișarea unui tabel cu datele furnizate.

Metodele "getClientId()", "getClientName()", "getClientAddress()", "getClientEmail()", "getClientAge()", "getProductId()", "getProductName()", "getProductQuantity()", "getProductPrice()", "getOrderId()", "getOrderProductId()", "getOrderClientId()" și "getOrderProductQuantity()" returnează textul introdus în câmpurile de text corespunzătoare.

1. **Rezultate**

**A screenshot of a computer screen

Description automatically generated with medium confidence**

1. **Concluzii**

Proiectul de management al comenzilor este o aplicație software care are ca scop gestionarea eficientă a comenzilor primite de un depozit.

Aplicația oferă o interfață grafică, editarea, ștergerea și vizualizarea comenzilor, produselor și clienților.

Se folosesc diferite componente grafice, cum ar fi butoane, câmpuri de text și tabele, pentru a afișa și manipula datele într-un mod intuitiv.

Utilizarea reflectiei în cadrul aplicației sugerează că aceasta oferă o abordare dinamică și flexibilă în gestionarea entităților și a acțiunilor asociate acestora.

1. **Bibliografie**

<https://gitlab.com/utcn_dsrl/pt-layered-architecture>

https://gitlab.com/utcn\_dsrl/pt-reflection-example

https://www.jetbrains.com/help/idea/working-with-code-documentation.html