



UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

DOCUMENTAȚIE GESTIONAREA COMENZILOR



Iepure Denisa-Alexandra

grupa: 30223



CUPRINS:

1.Obiectivele proiectului

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

3.Proiectare

4.Implementare

5.Rezultatele testării

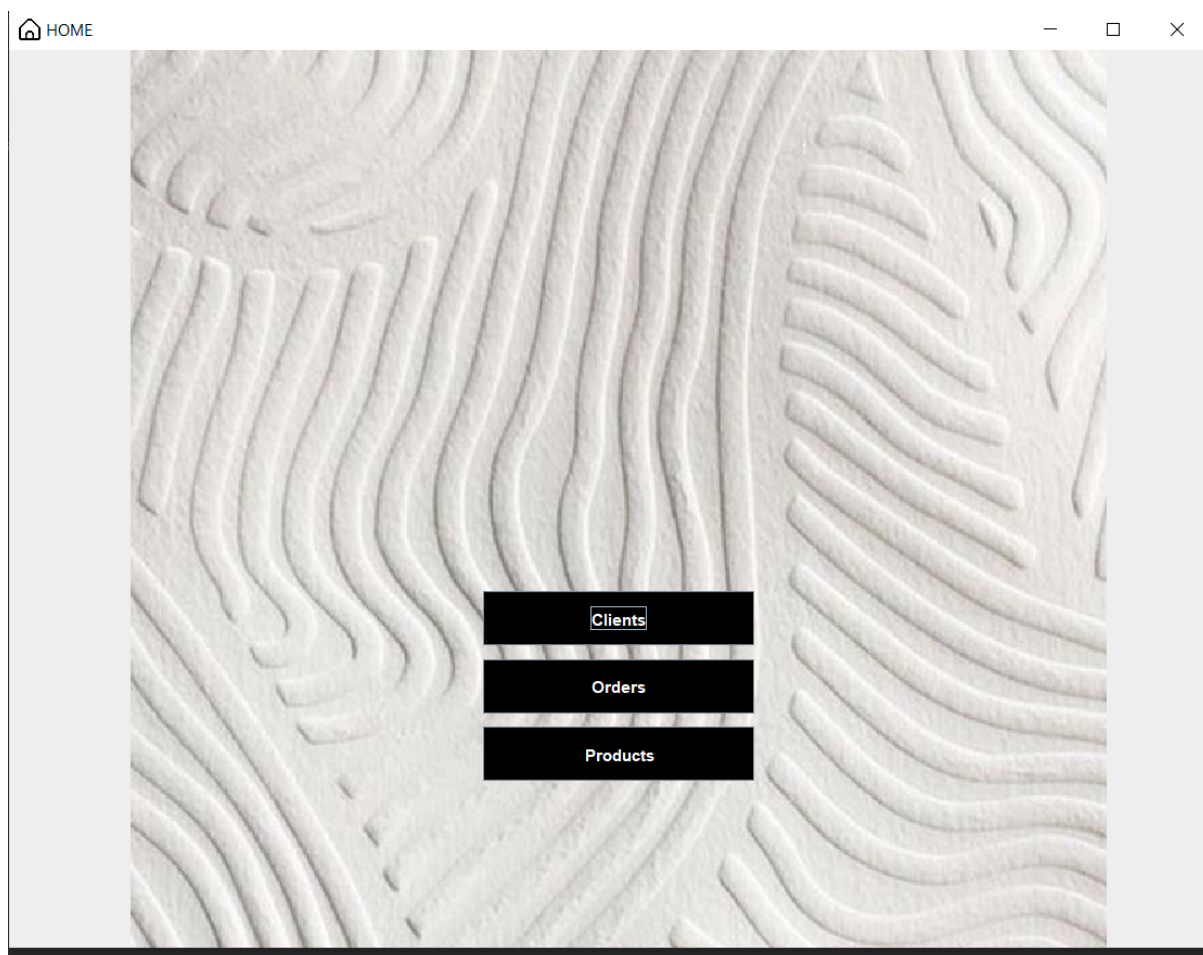
6.Concluzii

7.Biografie




UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

1.OBIECTIVELE PROIECTULUI





 Clients

BACK

ADD client

ADD client

UPDATE client

DELETE client

VIEW client

APPLY



 Clients



BACK

VIEW client

id	lastName	firstName	email
6	Iepure	Diana	denisaiepure3...
13	Gradinariu	Robert	roby_g@yaho...
15	Pavel	Andreea	pavel_andrei...
17	Ciofu	Dina	123
18	Sangeorzan	Elena	s_elena@yah...
25	Popescu	Andreea	pop_andre@y...

APPLY

Această aplicație are rolul de a simula gestionarea unor comenzi în cadrul unui “magazin online”. Ceea ce este important este utilizarea bazelor de date relaționale pentru a stoca clienții, produsele, comenzile, dar și facturile generate în urma comenzilor. Astfel, se pot efectua diferite operații: delete, update, insert. Interfața grafică ne permite să vizualizăm aceste operații, cât și să le efectuăm într-un mod cât mai natural și ușor de folosit.

Obiectivele secundare(Pașii urmăți)

- Analiza problemei



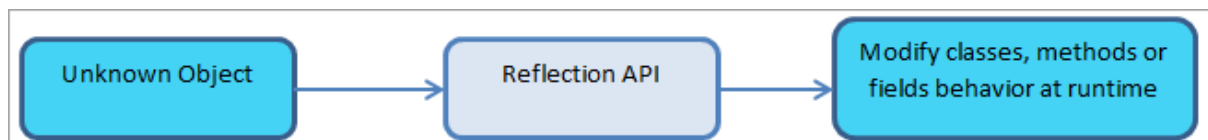
O bază de date reprezintă o colecție de informații (date); informații ce pot fi căutate și care sunt stocate electronic. Funcția unei baze de date nu este doar aceea de a stoca date, ci și de a le stoca într-un format care permite căutarea eficientă și recuperarea rapidă a informațiilor și de a asigura securitatea datelor. Funcțiile bazei de date în sine sunt proceduri care efectuează anumite operațiuni asupra datelor dintr-o bază de date. Funcțiile bazei de date includ operațiuni CRUD de bază, CRUD fiind un acronim pentru crearea, citirea, actualizarea și ștergerea datelor. În acest mod, după o primă analiză a cerinței se dorește efectuarea acestor operațiuni pe CLIENT, PRODUCT. Mai mult, în cerință observăm termenul de reflecție. Reflecția în Java este de a inspecta și schimba comportamentul unui program în timpul rulării. Cu ajutorul acestui API de reflecție, puteți inspecta clase, constructori, modificatori, câmpuri, metode și interfețe în timpul rulării. De exemplu, puteți obține numele clasei sau puteți obține detalii despre membrii privați ai clasei.

- **Proiectare**

După identificarea cerințelor, se proiectează interfața grafică a aplicației, astfel încât utilizatorul să poată introduce și gestiona această aplicație. Această etapă a inclus designul, testFieldurile, label-urile, tot ce vede utilizatorul.

- **Implementarea**

Cea mai importantă clasă, de unde știe programul să efectueze operațiile CRUD se numește **AbstractDAO**. Practic în această clasă implementăm în mod generic, adică pentru clasele Bill, Client, Order, Product ne folosim de metodele generice din AbstractDAO. Pe lângă asta, în implementare am avut mare grijă la denumirile atributelor claselor, la tipul acestora deoarece trebuia să coincidă cu denumirile și tipurile din MYSQL.



- **Testarea**

Această etapă presupune introducerea de clienți, de produse, efectuarea unor comenzi și vizualizarea tabelor rezultate în urma apelării metodei generice având ca parametru o listă de acel tip și care returnează un tabel actualizat.

-> folosim și un validator de email care înainte de adăugarea unui client este apelată.



2. ANALIZA PROBLEMEI, MODELARE, SCENARIU, CAZURI DE UTILIZARE

Cerințe funcționale : este nevoie de o aplicație pentru gestionarea comenzilor unui depozit și stocarea datelor în MySQL pentru o utilizare a acestora mult mai simplificată. Aplicația trebuie structurată în pachete și clase care utilizează o arhitectură specifică Layered Architecture.

3. PROIECTARE

Arhitectura de tip layered se referă la organizarea claselor în pachete specifice în funcție de funcționalitatea lor. Aceasta implică împărțirea aplicației în straturi logice, fiecare având un set bine definit de responsabilități. Această abordare facilitează gestionarea și dezvoltarea sistemelor software complexe. Într-o arhitectură de tip layered, clasele sunt împărțite în următoarele pachete principale:

- **Model:** Acest pachet conține clasele care modelează obiectele și entitățile necesare în aplicație. Aceste clase reprezintă structura de date și comportamentul acestora.
- **Presentation:** Aici se află clasele responsabile de interfața grafică a aplicației. Acestea gestionează interacțiunea cu utilizatorul și prezintă datele utilizând elemente de interfață grafică, cum ar fi ferestre, butoane sau casete de text.
- **DataAccess:** Acest pachet conține clasele responsabile de accesul la date. Acestea includ funcționalitatea de comunicare cu baza de date, efectuarea operațiilor CRUD (Create, Read, Update, Delete) și gestionarea conexiunii cu aceasta.
- **BusinessLogic:** Aici se găsesc clasele care conțin logica de afaceri a aplicației. Acestea se ocupă de procesarea datelor, aplicarea regulilor de afaceri și implementarea funcționalităților specifice.

Ele folosesc clasele din pachetul DataAccess pentru a obține și manipula datele. Beneficiile arhitecturii de tip layered includ: Separarea responsabilităților: Clasele sunt organizate în pachete specifice, ceea ce facilitează gestionarea și înțelegerea codului. Reutilizarea și modularitatea: Pachetele permit reutilizarea componentelor și înlocuirea lor cu ușurință, deoarece fiecare pachet reprezintă o unitate logică independentă. Testabilitate: Clasele din fiecare pachet pot fi testate separat, ceea ce facilitează testarea și depanarea aplicației. Scalabilitate: Arhitectura de tip layered permite adăugarea sau înlocuirea de straturi fără a afecta restul sistemului, ceea ce facilitează scalabilitatea și extensibilitatea.



4.IMPLEMENTARE

În pachetul

- **DAO: Clasa AbstractDAO<T>** este o clasă generică care servește ca bază pentru implementarea operațiilor de acces la date (CRUD) pentru obiecte de tip T. Această clasă abstractă este utilizată pentru a simplifica și a standardiza operațiunile de bază ale bazei de date în cadrul altor clase DAO specifice.

Clasele **ClientDAO**, **ProductDAO**, **OrdersDAO** : extind clasa AbstractDAO fiecare pentru tipul descris între<>, astfel ClientDAO pentru Client și așa mai departe.

- **MODEL:** unde sunt implementate clasele Bill, Client, Orders, Product, fiecare cu atributele corespunzătoare duplicate în MySQL pentru capul tabelului.
- **VIEW:** Contine clasele ClientView, MainFr, OrdersView, ProductsView (responsabile pentru crearea ferestrelor interfeței grafice) .
- **BLL:** conține logica aplicației, metodele din AbstractDAO sunt apelate mai elegant pentru fiecare tip
- **connectionDB:** realizarea conexiunii cu baza de date, am ales să o fac prin adăugarea dependenței în **pom.xml**

```
<dependencies>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.32</version>
    <scope>compile</scope>
  </dependency>
</dependencies>
```

5.Concluzii

Tema a implicat utilizarea reflectiei în Java pentru a facilita operațiile de acces la baza de date în cadrul unei aplicații de tip magazin online. Reflectia a permis analizarea și manipularea dinamică a structurii și comportamentului claselor, permițând astfel crearea de interogări și operații de bază pentru obiecte de tip generic. Implementarea arhitecturii de tip layered a permis separarea responsabilităților și organizarea codului în funcție de funcționalitatea acestuia. Pachetele specifice (model, presentation, dataAccess, businessLogic) au fost utilizate pentru a grupa clasele în funcție de rolul lor în aplicație.



6.Rezultate teste

Avem aici rezultatul de la facturi :2 facturi generate pentru cele 2 comenzi efectuate

ORDERS

BACK

Make an order

View orders

Bills

idBill	numeClient	prenumeClie..	numeProdus	cantitate	pret	date	pretBuc
1901	Popescu	Andreea	ciment	120	14400	2023-05-18 ...	120
2106	Gradinariu	Robert	faianta	6	120	2023-05-18 ...	20

6.Bibliografie

<https://ro.myservername.com/java-reflection-tutorial-with-examples>

<https://www.openvision.ro/blog/academia-it/baza-de-date/>