

DOCUMENTATIE

TEMA 3 *ORDERS MANAGEMENT*

Nume: Moga Eduard Mihai
Grupa: 4

CUPRINS

1.	Obiectivul temei	3
2.	Analiza problemei, modelare, scenarii, cazuri de utilizare	3
3.	Proiectare.....	5
4.	Implementare.....	Error! Bookmark not defined.
5.	Rezultate.....	Error! Bookmark not defined.
6.	Concluzii	Error! Bookmark not defined.
7.	Bibliografie.....	20

1. Obiectivul temei

Scopul proiectului constă în crearea unei aplicații pentru gestionarea comenzilor, menită să simplifice procesul de prelucrare a comenzilor clienților pentru un depozit. Se va folosi o bază de date relațională pentru a stoca informațiile despre produse, clienți și comenzile acestora, garantând o gestionare eficientă a datelor.

- Procesul începe cu analiza nevoilor utilizatorilor și stabilirea cerințelor (se va detalia în capitolul: 2. Analiza problemei, modelare, scenarii, cazuri de utilizare).
- Urmează proiectarea detaliată a arhitecturii software și a interfeței. (se va detalia în capitolul: 3. Proiectare)
- Implementarea se concentrează pe scrierea codului și integrarea funcționalităților. (se va detalia în capitolul: 4. Implementare)
- Testarea este esențială pentru asigurarea corectitudinii și stabilității aplicației. (scenariile pentru testare vor fi prezentate în capitolul: 5. Concluzii)

Scopul final este dezvoltarea unei aplicații robuste și eficiente, care să permită gestionarea optimă a comenzilor clienților pentru depozit, utilizând baza de date relațională pentru stocarea și manipularea datelor.

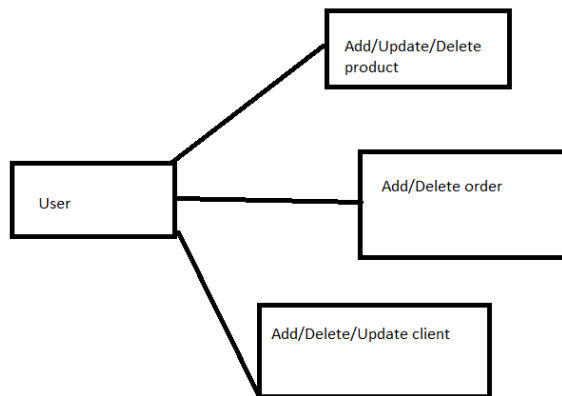
2. Analiza problemei, modelare, scenarii, cazuri de utilizare

Cerințe Funcționale:

- Insert / Update / Delete Product: Utilizatorul poate introduce un nou produs, sa modifice unul existent sau sa stearga un produs.
- Insert / Update / Delete Client: Utilizatorul poate introduce un nou client, sa modifice unul existent sau sa stearga un client.
- Insert / Delete Order: Utilizatorul poate introduce o noua comanda, sa modifice una existenta sau sa stearga o comanda.

- Interfață Grafică: O interfață grafică intuitivă care permite utilizatorului să manipuleze informații din baza de date și să vizualizeze modificările făcute în diferitele tabele.

Cerințe Non-Funcționale:



- Performanță: Aplicația trebuie să fie rapidă și responsive la interacțiunile utilizatorului.
- Ușurință de Utilizare: Interfața grafică trebuie să fie prietenoasă și ușor de înțeles pentru utilizatori de toate nivelurile de experiență.
- Fiabilitate: Aplicația trebuie să fie robustă și să gestioneze corect diversele scenarii de utilizare.

Use-case-urile sunt prezentate mai jos, actorul principal al acestora fiind utilizatorul:

- Add / delete / update client:

Scenariu de succes:

- Utilizatorul accesează interfața grafică a aplicației de gestionare a clientilor.
- Utilizatorul introduce datele corespunzătoare clientului
- Utilizatorul confirmă operația asupra clientului, făcând clic pe butonul respectiv operației dorite.

d) Sistemul validează / înregistrează / modifica clientul în baza de date.

- Add / delete / update product:

Scenariu de succes:

- Utilizatorul accesează interfața grafică a aplicației de gestionare a produselor.
- Utilizatorul introduce datele corespunzătoare produsului
- Utilizatorul confirmă operația asupra produsului, făcând clic pe butonul respectiv operației dorite.
- Sistemul validează / înregistrează / modifica produsul în baza de date.

- Add / delete order:

Scenariu de succes:

1. Utilizatorul accesează interfața grafică a aplicației de gestionare a comenzilor.
2. Utilizatorul introduce datele corespunzătoare comenzii
3. Utilizatorul confirmă operația asupra comenzii, făcând clic pe butonul respectiv operației dorite.
4. Sistemul validează / înregistrează / modifica comanda în baza de date și generează un bill de tip pdf cu datele comenzii.

3. Proiectare

Am ales să implementez un model arhitectural Multi Layered Architecture precum și un design arhitectural MVC. Acest model împarte aplicația în șapte componente distincte:

1. Model (Order, Client, Product, Log) Fiecare clasă descrie entitățile de bază implicate în gestionarea comenzilor pentru depozit. Clasa Order reprezintă o comandă plasată de un client pentru produsele din depozit și include informații despre produsele comandate, cantități și detalii despre clientul care a plasat comanda. Clasa Client modelează un client care plasează comenzi în depozit și conține informații precum nume, adresă și alte detalii relevante pentru identificarea și contactarea clientului. Clasa Product reprezintă un produs disponibil în depozit și conține detalii despre nume, descriere, preț și cantitatea disponibilă în stoc. Clasa Log reprezintă o factură generată pentru o comandă plasată de un client și conține informații despre comanda asociată, sumele totale pentru produsele comandate.
2. GUI: HomeScreen, clasa care se ocupa cu navigarea printre interfețe prin ajutorul butoanelor; HomeScreenController, clasa care implementează ActionListener și implementează logica butoanelor din HomeScreen; GenericPanel clasa care se ocupa cu interacionarea butoanelor de agaudare, modificare, stergere a elementelor din tabele, afisarea tabelelor și o comunicare cu alte paneluri; InsertController, clasa care

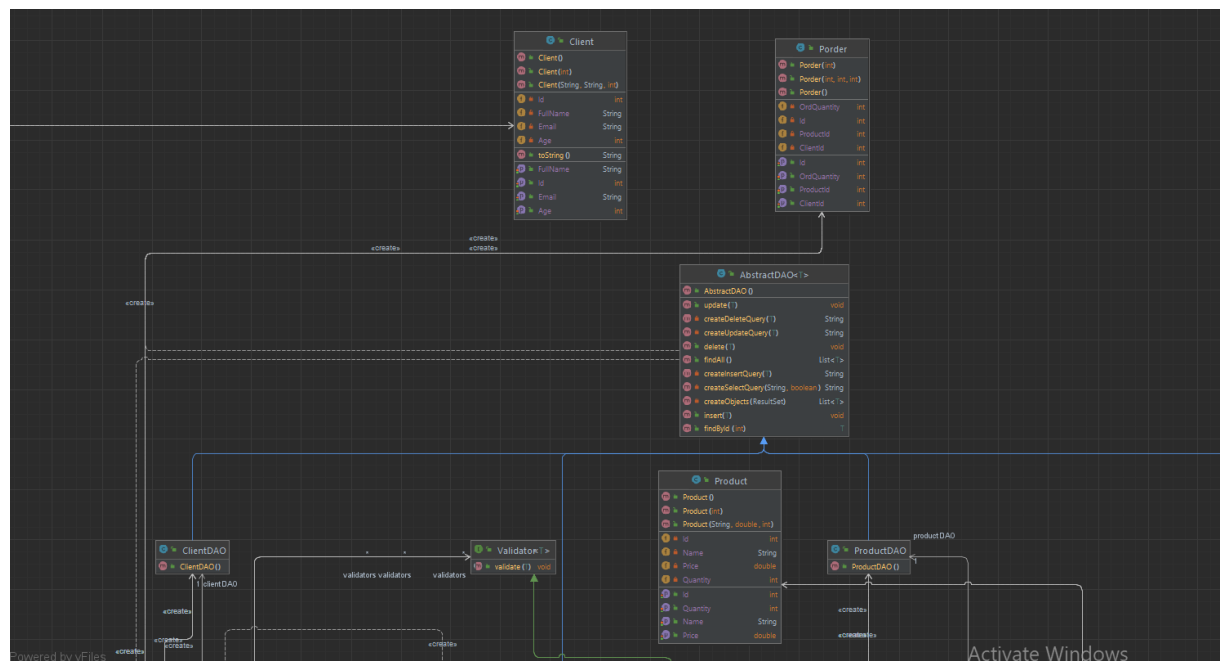
se ocupa de interfata de inserare a datelor in table; ButtonController clasa care se ocupa de actionarea butoanelor din Panelurile Client, Product, Porder, Bill.

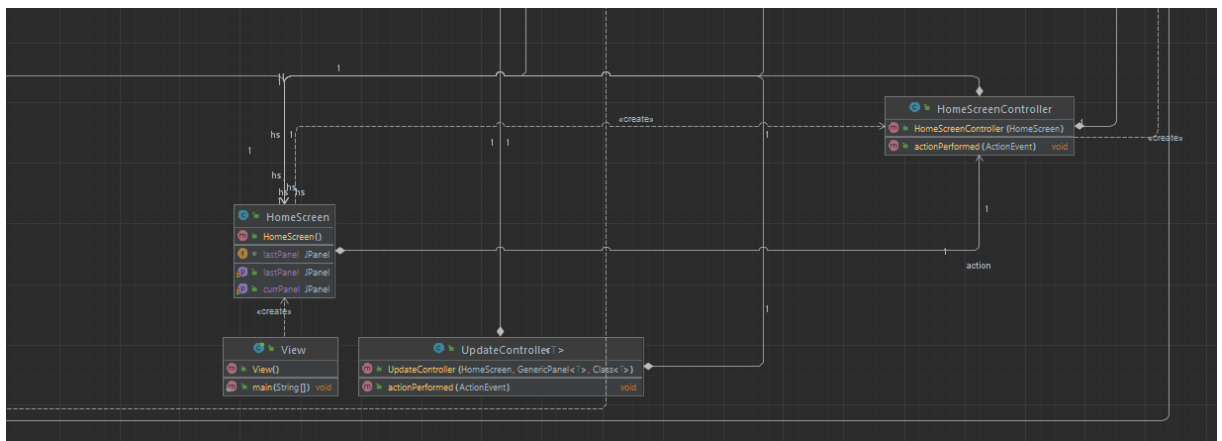
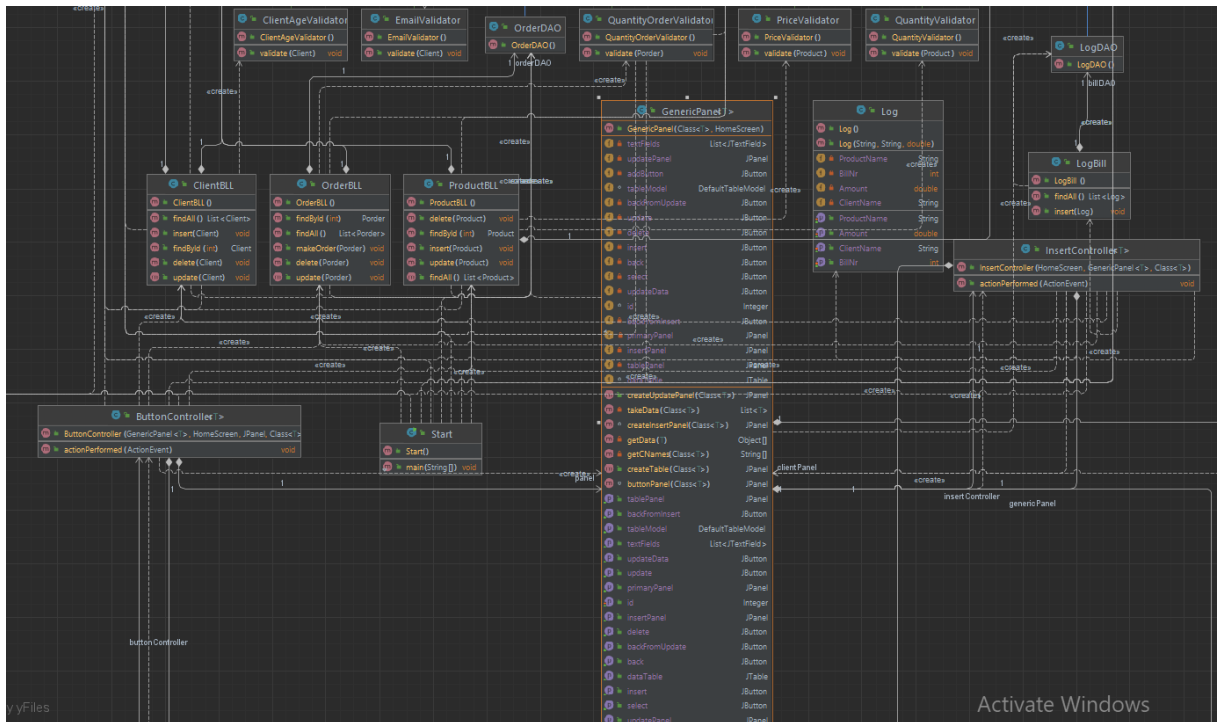
3. DAO(data access object): AbstractDAO clasa care implementeaza functii pentru prelucrearea datelor si comunicarea cu baza de date accesul backend a programului;

ClientDAO, LogDAO, OrderDAO, ProductDAO clasa care extinde AbstractDAO

4. ConnectionFactory: Clasa care face conexiunea propriu zisa cu severul de baze de date creata
5. BLL(Bussiness logic layer): ClientCLL, LogBLL, OrderBLL, ProductBLL clasa care se ocupa de logica, acesta clasa valideaza date iar apoi le trimite catre DAO pentru a le introduce in baza de date
6. Validators: ClientAgeValidator, valideaza varsta; EmailValidator valideaza emailul, PriceValidator valideaza preturile, QuantityOrderValidator valideaza comenzi valide in functie de id-ul Clientilor a Produselor si daca acestea exista precum si cantitatile comandate, QuantityValidator valideaza cantitatea in momentul crearii unui query de produs

Diagrama UML a proiectului este urmatoarea:





4. Implementare

AbstractDAO():

Este un constructor care folosește reflexia Java pentru a obține tipul generic al clasei. Folosește `getClass().getGenericSuperclass()` pentru a obține clasa generică a superclasei și `getActualTypeArguments()[0]` pentru a obține primul tip argument al clasei generice, adică tipul T.

Metoda `createSelectQuery(String field, boolean all)`:

Construiește și returnează o interogare SELECT SQL.

Dacă parametrul all este true, returnează un SELECT * FROM tableName.

Dacă parametrul all este false, returnează un SELECT * FROM tableName WHERE field = ?.

Metoda createInsertQuery(T field):

Construiește și returnează o interogare INSERT SQL pentru un obiect specificat de tipul T.

Utilizează reflexia pentru a parcurge câmpurile obiectului și a obține valorile acestora.

Generează o interogare care adaugă valorile câmpurilor obiectului într-un rând nou în tabelul corespunzător.

Metoda createUpdateQuery(T field):

Construiește și returnează o interogare UPDATE SQL pentru un obiect specificat de tipul T.

Folosește reflexia pentru a parcurge câmpurile obiectului și a obține valorile acestora.

Generează o interogare care actualizează valorile câmpurilor obiectului în rândul tabelului corespunzător bazat pe identificatorul obiectului.

Metoda createDeleteQuery(T field):

Construiește și returnează o interogare DELETE SQL pentru un obiect specificat de tipul T.

Folosește reflexia pentru a parcurge câmpurile obiectului și a obține identificatorul acestuia.

Generează o interogare care șterge rândul corespunzător din tabel bazat pe identificatorul obiectului.

Metoda findAll():

Returnează o listă de toate obiectele din tabel.

Construiește o interogare SELECT * FROM tableName și obține rezultatul.

Utilizează metoda createObjects(ResultSet resultSet) pentru a crea obiecte din rezultatul interogării.

Metoda findById(int id):

Returnează un singur obiect din tabel, identificat prin id.

Construiește o interogare SELECT * FROM tableName WHERE id = ? și obține rezultatul.

Utilizează metoda createObjects(ResultSet resultSet) pentru a crea obiectul din rezultatul interogării.

Metoda insert(T t):

Inserează un nou obiect în tabel.

Construiește o interogare INSERT SQL folosind metoda createInsertQuery(T field) și o execută.

Metoda update(T t):

Actualizează un obiect existent în tabel.

Construiește o interogare UPDATE SQL folosind metoda createUpdateQuery(T field) și o execută.

Metoda delete(T t):

Șterge un obiect din tabel.

Construiește o interogare DELETE SQL folosind metoda createDeleteQuery(T field) și o execută.

ClientBLL():

Inițializează lista de validatori pentru clienți, adăugând doi validatori: EmailValidator și ClientAgeValidator.

Metoda findById(int id):

Găsește un client după id.

Folosește obiectul ClientDAO pentru a căuta clientul în baza de date.

Aruncă o excepție IllegalArgumentException dacă clientul nu este găsit.

Metoda findAll():

Returnează o listă de toți clienții din bază de date.

Aruncă o excepție IllegalArgumentException dacă lista este goală.

Metoda insert(Client c):

Validează clientul folosind toți validatorii din lista.

Inserează clientul în bază de date folosind obiectul ClientDAO.

Metoda update(Client c):

Validează clientul folosind toți validatorii din lista.

Actualizează clientul în bază de date folosind obiectul ClientDAO.

Metoda delete(Client c):

Șterge clientul din bază de date, împreună cu toate comenzile asociate acestuia.

Caută toate comenzile asociate clientului dat și le șterge utilizând obiectul OrderDAO.

Apoi șterge clientul în sine folosind obiectul ClientDAO.

LogBill():

Inițializează obiectul LogDAO, utilizat pentru a interacționa cu baza de date.

Metoda findAll():

Returnează o listă de toate facturile din bază de date.

Utilizează obiectul LogDAO pentru a obține toate facturile.

Aruncă o excepție NoSuchElementException dacă lista de facturi este goală.

Metoda insert(Log b):

Inserează o nouă factură în baza de date.

Folosește obiectul LogDAO pentru a insera factura în baza de date.

OrderBLL():

Inițializează lista de validatori pentru obiectele Porder, adăugând un validator pentru cantitatea comenzii (QuantityOrderValidator).

Inițializează obiectul OrderDAO, utilizat pentru interacțiunea cu baza de date.

Metoda findById(int id):

Găsește o comandă după id.

Folosește obiectul OrderDAO pentru a căuta comanda în baza de date.

Aruncă o excepție IllegalArgumentException dacă comanda nu este găsită.

Metoda findAll():

Returnează o listă de toate comenzile din bază de date.

Utilizează obiectul OrderDAO pentru a obține toate comenzile.

Aruncă o excepție IllegalArgumentException dacă lista de comenzi este goală.

Metoda makeOrder(Porder o):

Validează comanda folosind toți validatorii din lista.

Scade cantitatea produsului din comandă din stoc folosind ProductDAO.

Inserează comanda în baza de date folosind OrderDAO.

Metoda update(Porder o):

Validează comanda folosind toți validatorii din lista.

Actualizează comanda în baza de date folosind OrderDAO.

Metoda delete(Porder o):

Șterge comanda din baza de date folosind OrderDAO.

ProductBLL():

Inițializează lista de validatori pentru obiectele Product, adăugând doi validatori:

PriceValidator și QuantityValidator.

Inițializează obiectul ProductDAO, utilizat pentru interacțiunea cu baza de date.
Metoda findById(int id):

Găsește un produs după id.
Folosește obiectul ProductDAO pentru a căuta produsul în baza de date.
Aruncă o excepție IllegalArgumentException dacă produsul nu este găsit.
Metoda findAll():

Returnează o listă de toate produsele din bază de date.
Utilizează obiectul ProductDAO pentru a obține toate produsele.
Aruncă o excepție IllegalArgumentException dacă lista de produse este goală.
Metoda insert(Product p):

Validează produsul folosind toți validatorii din lista.
Inserează produsul în baza de date folosind ProductDAO.
Metoda update(Product p):

Validează produsul folosind toți validatorii din lista.
Actualizează produsul în baza de date folosind ProductDAO.
Metoda delete(Product p):

Șterge produsul din baza de date, împreună cu toate comenzile care conțin acest produs.
Caută toate comenzile care conțin produsul dat și le șterge folosind OrderDAO.
Apoi șterge produsul în sine folosind ProductDAO.

ConnectionFactory():

Este privat pentru a asigura că instanța clasei poate fi accesată doar prin metoda statică getConnection().
Metoda createConnection():

Creează și returnează o conexiune la baza de date folosind URL-ul, numele de utilizator și parola specificate în cod.
Metoda statică getConnection():

Returnează o conexiune la baza de date. Este accesată direct de alte clase pentru a obține o conexiune.
Metoda statică close(Connection connection):

Închide o conexiune dată, dacă este deschisă.

Metoda statică close(Statement statement):

Închide un statement dat, dacă este deschis.

Metoda statică close(ResultSet resultSet):

Închide un ResultSet dat, dacă este deschis.

Client:

Variabile membre:

Id: reprezintă identificatorul unic al clientului.

FullName: reprezintă numele complet al clientului.

Email: reprezintă adresa de email a clientului.

Age: reprezintă vârsta clientului.

Constructori:

Client(): constructor implicit.

Client(String FullName, String Email, int Age): constructor care inițializează toate câmpurile clientului.

Client(int Id): constructor care inițializează doar identificatorul clientului.

Metode de acces pentru variabilele membre:

getId(), getFullName(), getEmail(), getAge(): returnează valorile corespunzătoare ale variabilelor membre.

setId(int id), setFullName(String fullName), setEmail(String email), setAge(int age): setează valorile variabilelor membre.

Metoda toString():

Returnează o reprezentare textuală a obiectului Client, care include toate detaliile despre acesta (id, nume complet, email și vârstă).

HomeScreen:

Butonul viewBut:

Acest buton este utilizat pentru a vizualiza tabela selectată din meniul derulant.

Atunci când este apăsat, declanșează o acțiune pentru a deschide tabela corespunzătoare.

Meniul derulant comboBox:

Acest meniu derulant permite utilizatorului să selecteze tipul de tabel pe care dorește să-l vizualizeze (clienți, produse, comenzi sau facturi).

Utilizează un array de string-uri pentru a afișa opțiunile disponibile.

Panoul `displayPanel`:

Acest panou este utilizat pentru a afișa butonul și meniul derulant.

Este format dintr-un `GridBagLayout` pentru a dispune elementele într-un mod organizat.

Variabila `lastPanel`:

Această variabilă reține panoul anterior, utilizată pentru a reveni la starea anterioară după ce un alt panou a fost afișat.

Variabila `insertBackState`:

Această variabilă ar putea fi utilizată pentru a reține panoul anterior în cazul în care este necesară revenirea la o stare anterioară.

`HomeController` action:

Acesta este controllerul asociat ecranului de start, utilizat pentru a gestiona acțiunile butonului `viewBut`.

Constructorul `HomeController()`:

Inițializează și configurează fereastra.

Adaugă butonul și meniul derulant la panoul de afișare.

Setează dimensiunile ferestrei și o face vizibilă.

HomeController

Variabile membre:

`hs`: Reprezintă instanța de `HomeController` asociată cu acest controler.

`clientPanel`, `productPanel`, `orderPanel`, `billPanel`: Reprezintă panourile generice pentru diferitele entități (clienți, produse, comenzi, facturi).

Constructorul `HomeController(HomeController hs)`:

Inițializează controlerul cu instanța `HomeController` asociată.

Metoda `actionPerformed(ActionEvent e)`:

Este declanșată atunci când utilizatorul interacționează cu un element grafic (de exemplu, butonul sau meniul derulant).

Verifică tipul de tabel selectat și creează un panou corespunzător pentru afișarea datelor.

Înlocuiește panoul curent din HomeScreen cu noul panou creat.

Se ocupă și de gestionarea revenirii la starea anterioară.

GenericScreen<T>

Variabile membre:

data: Lista de obiecte care conține datele afișate în tabel.

type: Clasa tipului de obiect (Client, Product, Porder sau Log).

textFields: Lista de câmpuri de text pentru introducerea datelor noi.

labels: Lista de etichete pentru câmpurile de text.

Buton de navigare înapoi.

Buton de navigare înapoi din fereastra de inserare.

Buton de navigare înapoi din fereastra de actualizare.

Buton de inserare a datelor.

Buton de actualizare a datelor.

Buton de ștergere a datelor.

Buton de adăugare a datelor.

Buton de selectare a datelor.

Panoul de butoane.

Panoul tabelului.

Panoul principal.

Panoul de inserare.

Panoul de actualizare.

Ultimul panou afișat.

Controlerul pentru inserare.

Controlerul pentru butoane.

Controlerul pentru actualizare.

Tabelul de date.

Modelul tabelului.

Id-ul pentru actualizare sau ștergere.

Constructorul GenericPanel(Class<T> type, HomeScreen hs):

Inițializează componenta cu tipul specific de date și cu instanța HomeScreen asociată.

Creează panourile de inserare și actualizare și le asociază controlerii corespunzători.

Creează panoul de butoane și panoul tabelului și le adaugă în panoul principal.

Metoda buttonPanel(Class<T> type):

Creează un panou de butoane care conține butoanele de navigare înapoi, inserare, actualizare și ștergere, în funcție de tipul de date afișat.

Metodele `createInsertPanel(Class<T> type)` și `createUpdatePanel(Class<T> type)`:

Creează panourile de inserare și actualizare, respectiv, cu câmpurile corespunzătoare pentru introducerea datelor.

Metoda `takeData(Class<T> type)`:

Obține datele din baza de date corespunzătoare tipului specific de obiect.

Metoda `getCNames(Class<T> data)`:

Obține numele coloanelor din tabel pe baza atributelor obiectului.

Metoda `getData(T data)`:

Obține datele dintr-un obiect și le returnează sub formă de array de obiecte.

Metoda `createTable(Class<T> type)`:

Creează tabelul care afișează datele obținute din baza de date.

ButtonController<T>

Variabile membre:

`panel`: Panoul generic asociat controlerului.

`cPanel`, `poPanel`, `prPanel`: Panourile specifice pentru Client, Porder și Product.

`hs`: Instanța clasei `HomeScreen` asociată controlerului.

`insertPanel`: Panoul de inserare specific.

`type`: Tipul de obiect gestionat de panoul generic.

Constructorul `ButtonController(GenericPanel<T> genericPanel, HomeScreen hs, JPanel insertPanel, Class<T> type)`:

Inițializează variabilele membre cu valorile primite ca parametri.

Metoda `actionPerformed(ActionEvent e)`:

Gestionează evenimentele declanșate de butoanele din interfața grafică.

Acțiunea butonului "back":

Închide panoul curent și revine la panoul anterior afișat.

Actualizează interfața grafică.

Acțiunea butonului "update":

Identifică rândul selectat din tabel.

Preia datele rândului selectat și le afișează în panoul de actualizare.

Afișează panoul de actualizare în interfață.

Acțiunea butonului "insert":

Afișează panoul de inserare în interfață.

Acțiunea butonului "delete":

Identifică rândul selectat din tabel.

Identifică tipul obiectului și realizează operația de ștergere corespunzătoare în baza de date.

Actualizează lista de date afișate în tabel și interfața grafică.

InsertController<T>

Variabile membre:

hs: Instanța clasei HomeScreen.

genericPanel: Panoul generic asociat controlerului.

type: Tipul de obiect gestionat de panoul generic.

Constructorul InsertController(HomeScreen hs, GenericPanel<T> genericPanel, Class<T> type):

Inițializează variabilele membre cu valorile primite ca parametri.

Metoda actionPerformed(ActionEvent e):

Gestionează evenimentele declanșate de butoanele din interfața grafică.

Acțiunea butonului "Add":

Preia datele introduse în câmpurile de inserare.

Inserează un nou obiect în baza de date în funcție de tipul de obiect gestionat.

Afișează panoul corespunzător în interfață.

Acțiunea butonului "Back" din panoul de inserare:

Revine la panoul anterior afișat în interfață.

Acțiunea butonului "Back" din panoul de actualizare:

Revine la panoul anterior afișat în interfață.

Acțiunea butonului "Update Data" din panoul de actualizare:

Preia datele introduse în câmpurile de actualizare.

Actualizează obiectul selectat în baza de date în funcție de tipul de obiect gestionat.

Afișează panoul corespunzător în interfață.

5. Rezultate

Testarea funcționalităților proiectului a implicat operații pe date predefinite pentru a acoperi diverse scenarii și cazuri de utilizare. Aceste operații au inclus adăugarea și ștergerea comenzilor, gestionarea stocurilor de produse și procesarea clienților.

Fiecare operație a fost proiectată pentru a verifica coerența și corectitudinea funcționalităților în diverse situații de utilizare. De exemplu, în timpul testării adăugării comenzilor, s-au verificat aspecte precum validarea datelor introduse de utilizator, actualizarea corectă a stocurilor de produse și înregistrarea comenzilor în baza de date.

Rezultatele fiecărei operații au fost înregistrate în baza de date pentru a permite o analiză ulterioară și pentru a asigura integritatea datelor. Aceste înregistrări au inclus detalii despre operațiile efectuate, cum ar fi timpul de execuție, parametrii utilizați și eventualele erori sau excepții întâlnite în timpul procesării. Afișarea datelor din baza de date sub forma unui tabel a facilitat procesul de corectare.

back	Id	FullName	Email	Age
insert	25	Edo	edi@yahoo.com	57
delete	26	yusuf	yusuf@yahoo.com	30
update				

FullName


Email

Age

Add

back

Message

 Email is not a valid email!

OK

FullName

Email

Age

Add

back

Experiența acumulată în timpul implementării acestui proiect a adus la lumină câteva concluzii și idei relevante, care pot fi de folos în viitoarele proiecte:

- *Structurarea obiectivelor în sub-obiective a fost esențială pentru progresul eficient și gestionarea adecvată a proiectului. Împărțirea obiectivelor mari în etape mai mici și mai ușor de gestionat ne-a permis să ne concentrăm pe realizarea pașilor necesari pentru atingerea obiectivului final.*

- *Alegerea structurilor de date potrivite a avut un impact semnificativ asupra performanței și eficienței aplicației. Utilizarea adecvată a structurilor de date, cum ar fi bazele de date relaționale pentru stocarea și gestionarea comenzilor și produselor, a dus la o gestionare mai eficientă a informațiilor și a redus timpul necesar pentru procesarea comenzilor.*

- *Abstractizarea codului prin intermediul genericelor a permis o dezvoltare mai flexibilă și extensibilă a aplicației. Definirea unor interfețe clare pentru operațiile de bază și utilizarea genericelor pentru a implementa comportamente comune au facilitat integrarea și extinderea funcționalităților fără a afecta structura existentă a codului.*

- *Menținerea lizibilității codului a fost un aspect crucial în dezvoltarea și întreținerea aplicației. Limitarea dimensiunii metodelor și claselor, precum și respectarea unor convenții de denumire și organizare a codului, au fost esențiale pentru înțelegerea și gestionarea eficientă a acestuia pe parcursul proiectului.*

Bibliografie

1. <https://dsrl.eu/courses/pt/>
2. <https://mvnrepository.com/artifact/org.apache.pdfbox/pdfbox>
3. <https://www.baeldung.com/java-generics>