DOCUMENTATIE

TEMA *3*

NUME STUDENT: Stefan Razvan-Mihai
GRUPA: 30224

CUPRINS

1. Obiectivul temei

Obiectivul principal al acestei teme este dezvoltarea unei aplicatii prin care sa se poata realiza actiunile principale ale unei firme. Angajatii trebuie sa poata introduce clienti, produse si comenzi intr-o baza de date. Acestia vor indeplini aceste cereri utilizand o interfata grafica.
In baza de date vor exista 3 tabele: Client, Produs si Comanda. Pentru fiecare dintre acestea va aparea o fereastra pentru efectuarea anumitor operatii. Pentru Produs si Client, angajatul va putea adauga, sterge, vizualiza tabelul, precum si a edita un camp la alegere. Pentru Comanda, angajatul va putea adauga (selectand un client si un produs) si vizualiza comenzile efectuate pana atunci.

Obiective secundare:

-analizarea problemei și identificarea cerinţelor
-proiectarea unei aplicatii (diagrame de entitati, diagrame cu cazuri de utilizare etc.)
-implementarea aplicatiei
-testarea aplicatiei

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

O baza de date este o colectie structurata si organizata de informatii stocate electronic intr-un computer. Bazele de date sunt controlate de obicei folosind un sistem de gestiune a bazelor de date (SGBD). In cazul acestui proiect, sistemul de gestiune a bazelor de date ales a fost MySQL. Cu ajutorul acestuia, au fost create tabelele necesare pentru a stoca date despre un depozit si despre tranzactiile efectuate cu produsele din acesta.

Cerintele functionale sunt:
- posibilitatea de a introduce, edita, sterge si produse si clienti,
posibilitatea de a vizualiza tabele cu toate produsele, clientii si comenzile si posibilitatea de a introduce comenzi in mod intuitiv.
-crearea documentatiei javadoc in urma descrierilor adaugate in cod.
-stocarea datelor in mod constant (nu doar cand programul ruleaza).
-tratarea unor exceptii, fara a cauza intreruperea programului (ex: adaugare produse cu pret negativ, editarea unui client/ produs care nu mai exista, plasarea unei comenzi cu cantitate negativa etc.).

Utilizatorul destinat acestei aplicatii este un angajat al unei firme care poate efectua livrari de diverse produse dintr-un depozit. Acest angajat trebuie sa se ocupe de gestionarea unei baze de date care sa stocheze informatii despre clienti, a stocului de produse si a comenzilor realizate de clienti.

Utilizatorul trebuie sa cunoasca informatiile despre clientii si produsele pe care doreste sa le introduca (pentru clienti: nume, adresa, email; pentru produse: nume, pret, cantitate iar pentru comenzi, acesta nu trebuie sa cunoasca informatii specifice, deoarece acestea se vor crea odata cu efectuarea lor, cu ajutorul interfetei grafice).

Cazuri de utilizare:

Adaugarea unui client/ produs: se introduc in casuta corespunzatoare urmatoarele informatii, in aceasta ordine si cu spatiu intre ele: nume, adresa, adresa de email/nume, pret, cantitate , dupa care se apasa butonul “ Add Client/Product ”.

Editarea unui client/ produs: se introduce in campul corespunzator id-ul clientului/ produsului care se doreste a fi editat, se selecteaza atributul care trebuie editat, se introduce intr-un alt camp noua valoare a acelui atribut, dupa care se apasa pe butonul “ Edit Client/Product with ID: ”.

Stergerea unui client/ produs: se introduce in campul corespunzator id-ul produsului/ clientului care se doreste a fi sters, dupa care se apasa butonul “ Delete Product/Client with ID: ”.

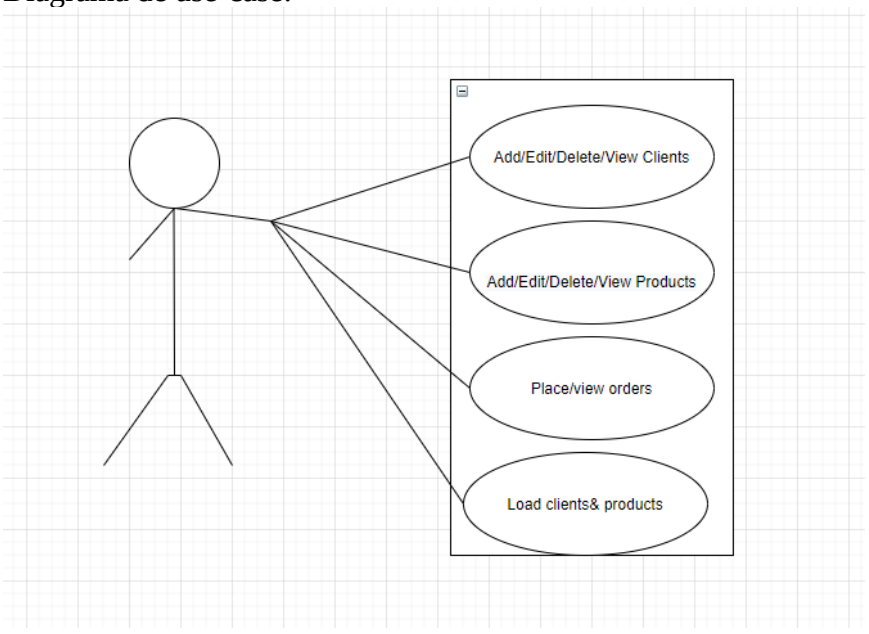
Vizualiarea clientilor/ produselor/ comenzilor: se apasa butonul “ View Products/Clients/Orders ”, dupa care se va afisa tabelul corespunzator in partea de jos a interfetei grafice.

Adaugarea unei comenzi: se apasa butonul “ Load Clients and Products ”, pentru a incarca valorile acestor tabele in 2 combobox-uri. Se selecteaza clientul si produsul corespunzator din acele 2 combobox-uri, se introduce cantitatea, dupa care se apasa butonul “ Place Order ”. Rezultatul comenzii se va afisa intr-un camp pentru text sub forma unui mesaj care poate fi “Comanda realizata cu succes ” sau “ Stoc insuficient. Comanda nu a fost realizata”, in cazul in care cantitatea introdusa depaseste stocul disponbil.

Pentru operatiile de stergere sau editare se recomanda vizualizarea tabelelor prin apasarea butonului predestinat, pentru introducerea corecta a id-ului.

De asemenea, pentru a vizualiza tabelele actualizate dupa efectuarea operatiilor de adaugare/ editare/ stergere, trebuie apasat butonul pentru vizualizare.

Diagrama de use-case:



3. Proiectare

Pentru proiectarea acestei aplicatii s-a folosit o arhitectura de tip Layered Architecture, care contine urmatoarele pachete:

-businesslogic: care contine clase cu functiile pentru efectuarea operatiilor mentionate la punctele anterioare, aplicand si unele verificari a corectitudinii datelor, folosind pachetul validators. De asemenea, contine clasa main, de unde incepe executia.

-connection: aici se afla clasa care contine clasa ConnectionFactory, unde se afla metodele folosite pentru conectarea aplicatiei Java la baza de date MySQL.

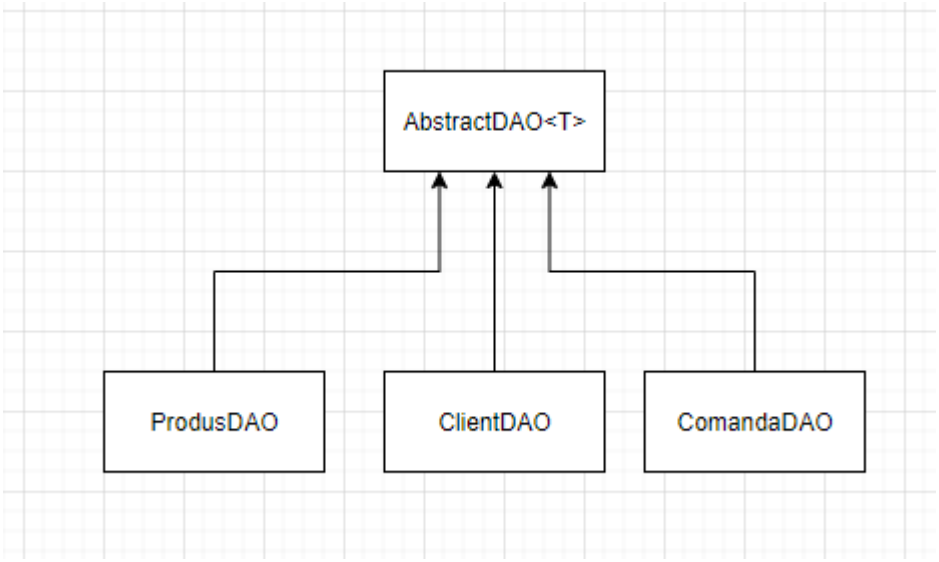
-dataaccess: in acest pachet se afla clasele care se ocupa cu realizarea operatiilor pe cele 3 tabele din baza de date.

-model: aici se afla clasele necesare pentru "traducerea" unui rand din tabelele corespunzatoare din baza de date intr-un obiect care se poate manipula in java. De asemenea, gasim si o clasa auxiliara, care se foloseste pentru realizarea transmiterii argumentelor "prin referinta" unei anumite metode.

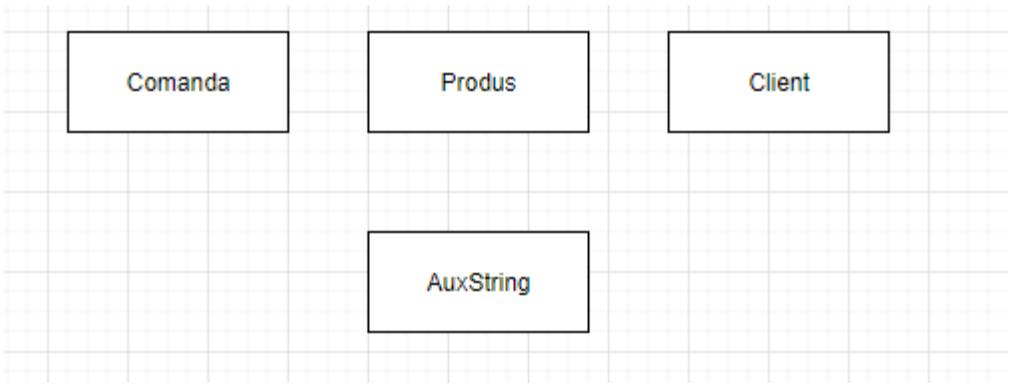
-presentation: aici se afla clasa view, unde este creata interfata grafica, precum si clasa controller unde se realizeaza comunicarea intre interfata grafica si aplicatia din spate.

Diagrama de clase si de pachete:

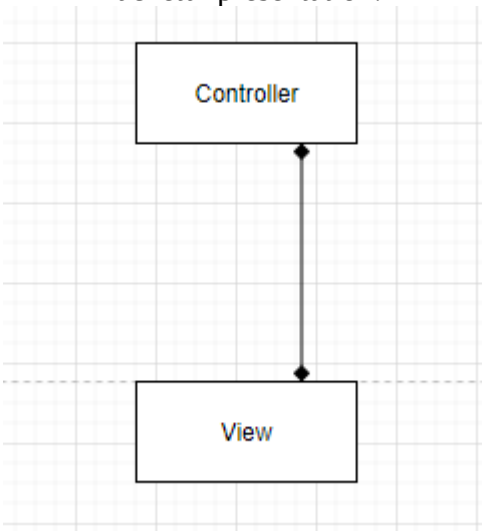
Pachetul DATAACCESS:



Pachetul Model:



Pachetul presentation:



Pachetul validators:

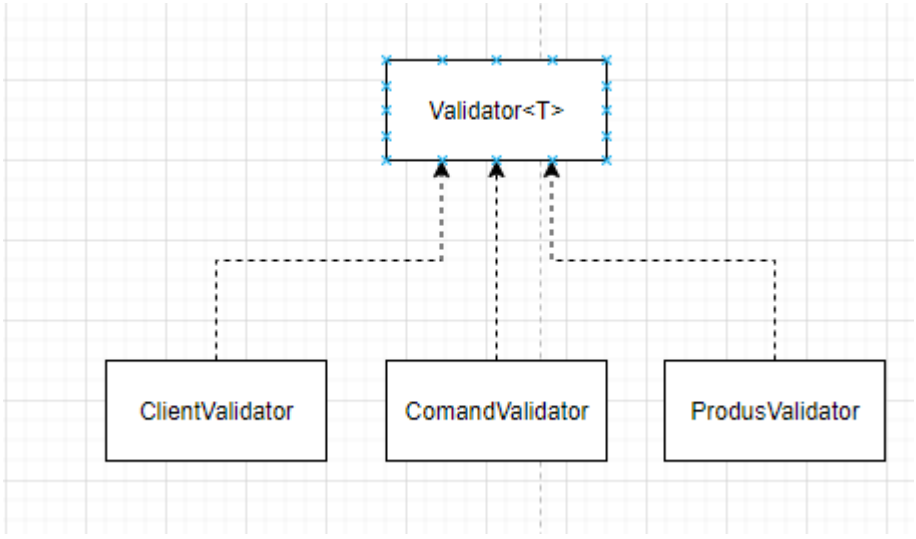
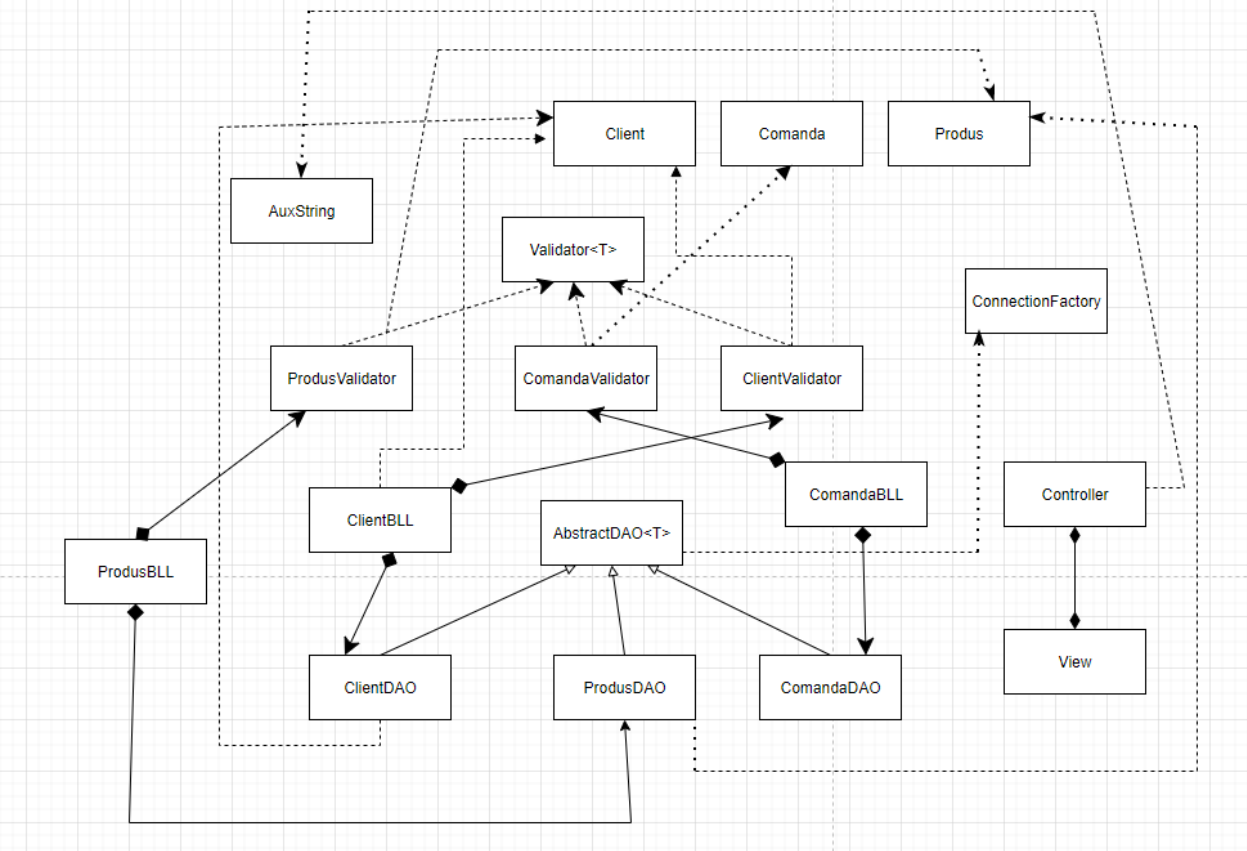


Diagrama finala:



4. Implementare

Clasa Client

Contine atributele: id, name, address,email si este folosita pentru crearea de obiecte echivalente ale unei linii din tabelul Client al bazei de date.

Clasa Produs

Contine atributele: id, name , price, quantity si se foloseste pentru crearea de obiecte echivalente unei linii din tabelul Produs al bazei de date.

Clasa Comanda

Contine atributele: id, idProd, idClient, cantitate, data. idProd si idClient reprezinta id-ul produsului comandat, respectiv id-ul clientului care comanda.

Clasa AuxString

Atribute: M, care este o matrice de siruri de caractere.

Este folosita pentru transmiterea “prin referinta” a unei matrice de String-uri functiei view din AbstractDAO.

Metode: getStrings(int nrAtr): Transforma o matrice de siruri de caractere intr-un sir de String-uri, unde fiecare element este un string care reprezinta o linie din matricea anterior

precizata, concatenata sub un anumit format. Se foloseste pentru popularea combobox-urilor din interfata grafica folosite pentru selectarea clientilor si produselor in cazul plasarii unei comenzi. Parametrul nrAtr ne da numarul de atribute al obiectului pentru care aplicam metoda.

Clasa ConnectionFactory

Contine atributele finale LOGGER, DRIVER, DBURL, USER, PASS.
Mai apoi, clasa contine metodele createConnection(), getConnection() si close, folosite pentru a realiza comunicarea dintre aplicatia in java si baza de date.

Interfata Validator

Contine prototipul metoda validate(T t), care urmeaza a fi implementata in cele 3 clase care implementeaza aceasta interfata;

Clasa ClientValidator

Contine metoda validate(Client c), care primeste un client si verifica daca acesta are id-ul valid (>=0) si daca numele nu contine cumva cifre (pentru evitarea unor erori umane cand se tasteaza datele. In cazul identificarii acestor erori, se va afisa o exceptie care nu va opri executia programului.

Clasa ComandaValidator

Contine metoda validate(Comanda comanda), care verifica daca: comanda are id-ul pozitiv, id-ul produsului este pozitiv, id-ul clientului este pozitiv, si daca cantitatea este pozitiva.

Clasa ProdusValidator

Contine metoda validate(Produs produs) care verifica daca produsul are un id pozitiv si daca pretul acestuia este pozitiv.

Clasa AbstractDAO

Aceasta clasa este de tipul generic, deoarece metodele acesteia vor fi apelate de catre mai multe subclase diferite.

Contine atributele:

-LOGGER care este utilizat in stabilirea conexiunii cu baza de date

-type care este utilizat pentru obtinerea clasei pentru care sunt apelate anumite metode.

Contine metodele:

-getAttributes(Object object, boolean comma): Obtine un string care contine numele atributelor obiectului obj separate cu spatiu daca comma este false, respectiv virgula, daca comma este true.

-getAttributeValues(Object object): returneaza un string care contine valorile atributelor obiectului object, separate prin virgula si inconjurate de paranteze (deoarece acest string se foloseste in interiorul statement-urilor de executie pt mysql).

-getVal(Object object): aceasta metoda obtine un string cu valorile atributelor obiectului, separate prin spatiu.

-createInsertQuery(Object object): aceasta metoda creeaza un string care urmeaza sa fie executat ca fiind cod MySQL, efectul fiind inserarea in baza de date a unei tuple cu

valorile egale cu cele ale obiectului dat ca parametru. In aceasta metoda se folosesc metodele getAttributes si getAttributeValues.

-createUpdateQuery(String field, String newValue, int id): string-ul returnat de aceasta metoda reprezinta codul pentru actualizarea tuplei cu id-ul id, campul fild primind valoarea newValue.

-createDeleteQuery(int id): genereaza codul pentru stergerea din baza de date a tuplei care are id-ul egal cu cel dat ca parametru.

-createSelectQuery(): genereaza codul pentru vizualizarea tuturor datelor din tabelul care are numele egal cu numele clasei care apeleaza.

-insert(Object object): insereaza (in tabelul corespunzator) in baza de date o tupla cu valorile obiectului object. In cazul unui obiect de tip Produs, se apeleaza o procedura in loc sa se foloseasca string-ul pentru insert. Aceasta procedura actualizeaza stocul disponibil de produse in baza de date.
Se foloseste de metoda createSelectQuery.

-edit(String field, String newValue, int id): Se executa comanda creata de createEditQuery pentru a schimba valoarea campului field a tuplei cu id-ul id, cu newValue.

-delete(int id): se executa comanda data de createDeleteQuery pentru a sterget tupla cu id-ul egal cu cel dat ca parametru.

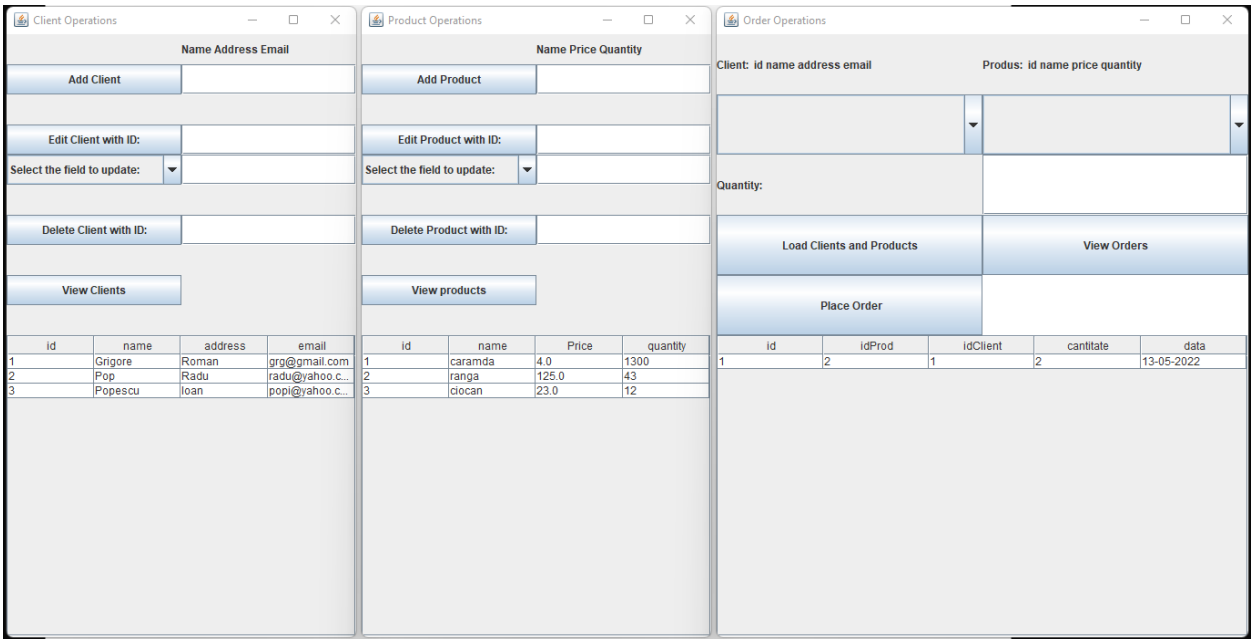
-view(Object obj, boolean tab, AuxString au): Aceasta metoda creeaza un JTable care contine valorile din tabelul cu nume egal cu numele clasei obiectului obj. Aceasta metoda este folosita fie pentru a afisa tabelele in interfata grafica, fie pentru a popula matricea de siruri de caractere continuta de parametrul au. Acea matrice este mai apoi folosita pentru implementarea unor JCombobox-uri in interfara pentru comenzi (pentru a se selecta clientul, respectiv produsul comandat). Pentru obtinerea numelor coloanelor din JTable, este folosita metoda getAttributes. String-ul obtinut este mai apoi impartit intr-un sir de string-uri, folosind metoda predefinita split. Pentru popularea matricii, se foloseste functia createObjects care returneaza un sir de obiecte formate din rezultatele interogarii tabelelor. Pentru fiecare dintre aceste obiecte obtin valorile atributelor cu getAttributeValues si creez un sir de string-uri cu acestea (folosind, din nou, metoda split). Acest sir de string-uri este inserat ca urmatoarea linie in matrice.

Clasa View

In interiorul acestei clase este creat designul interfetei grafice cu care interactioneaza utilizatorul. Interfata este formata din 3 JFrame-uri, fiecare continand operatiile specifice pentru clienti, produse si comenzi.

Interfata pentru clienti contine 4 butoane, un combobox, cateva textfield-uri, precum si un JTable daca se apasa butonul de vizualizare. Interfata pentru produse respecta acelasi model, dar are particularitati specifice produselor. Interfata pentru comenzi contine 2 combobox-uri, 3 butoane, un TextField un TextArea, precum si spatiu pentru afisarea JTable-ului corespunzator. Penttru a putea scrie in acel JTextArea, am folosit metoda setText(" ") pentru a goli rezultatul anterior, iar mai apoi functia append, care adauga string- ul dat ca parametru la textarea fara sa rescrie ce se afla deja inauntru. Pentru toate JFrame-urile am folosit GridLayout.

Fiecare JTable este continut de catre un JScrollPane pentru a putea vedea toate inregistrarile, folosind functia de scroll a mouse-ului.



Clasa Controller

Aceasta clasa contine atributul view si metodele actionPerformed, stringToClient si stringToProdus.

Metodele stringToProdus si stringToClient preiau string-urile introduse de utilizator in campurile pentru introdus clienti/produse si le transforma in obiectele corespunzatoare.

Metoda actionPerformed controleaza ce se intampla la apasarea fiecarui buton din interfata grafica.

In cazul butoanelor de tip vizualizare, se sterg si se reconstruiesc componentele din JFrame-uri pentru a putea vedea rezultatele actualizate fara a inchide aplicatia (ci doar reapasant butonul de vizualizare).

In cazul apasarii celorlalte butoane, se apeleaza metodele specifice din clasele de tip BLL pentru efectuarea operatiilor pe baza de date.

In cazul plasarii unei comenzi, pe langa efectuarea comenzii in baza de date, se genereaza si un fisier text cu formatul Order_clientID_time care contine detaliile comenzii.

5. Concluzii

Se vor prezenta concluziile, ce s-a invatat din tema, dezvoltari ulterioare.

Datorita acestei teme am reusit sa ma familiarizez cu driver-ul JDBC si cu modul in care trebuie tratate tipurile de proiecte care contin si baze de date. In plus, datorita multitudinii de operatii de implementat, am devenit mai confortabil cu scrierea de interfete grafice si cu actualizarea in timp real a acestora.

Ca dezvoltari ulterioare, acestui proiect i-ar putea fi adaugate urmatoarele functionalitati:

- Plasarea de comenzi cu produse multiple
- Anularea unor comenzi (cu refacerea stocului)
- Monitorizarea vanzarilor si profitului

6. Bibliografie

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

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

https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html

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