MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Facultatea de Automatică și Calculatoare

Departamentul de Calculatoare

Documentație Tema 3

Orders management

Student:

Mălăescu Bianca

Grupa 30238

Cuprins

1. Obiectivul temei

Obiectivul temei constă în proiectarea și implementarea unei aplicații care gestionează comenzile unui depozit.

Aceasta va stoca clienții depozitului, toate produsele disponibile și toata comenzile efectuate.

Interfața aplicației va conține 3 ferestre: fereastra dedicată produselor, cea dedicată clienților și fereastra corespunzătoare comenzilor. Utlizatorul va putea efectua operații de tip CRUD pe produse și pe clienți și poate adauga noi comenzi de produse. Crearea comenzilor va fi restricționata de stocul isponibil al produsului care urmează să fie adăugat în comanda.

Limbajul utilizat pentru realizarea aplicației este Java, unul dintre cele mai populare limbaje de programare care utilizează programarea orientată pe obiecte. Aceasta este folosit cel mai des pentru aplicații desktop, web sau de mobil.

Scopul acestui proiect este de a gestiona legăturile dintre cele 3 modele folosite: comanda, produs și client pentru o bună funcționare a aplicației și pentru a facilita sistemul de creare comenzi.

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

Analiza problemei

Observăm că cele 3 tipuri de obiecte cu care vom lucra sunt: client, produs și comandă.

Persistența datelor va trebui să fie asigurată. Ca soluție voi folosi baza de date MySQL și MySQL Workbench, pentru a gestiona mai ușor datele stocate.

MySQL este un sistem de management al bazelor de date relaționale bazat pe SQL - Structured Query Language. Aplicația este utilizată pentru o gamă largă de scopuri, inclusiv depozitarea datelor, comerțul electronic și aplicațiile de înregistrare în jurnal. Cu toate acestea, cea mai comună utilizare pentru MySQL este în scopul unei baze de date web.

Obiectele care vor fi stocate în baza de date au nevoie de unele legături între ele. O comandă va conține un client și un produs, astfel va trebui sa realizăm aceste legături între entitățile bazei de date. Având aceste legături, unele acțiuni pe care le poate face utilizatorul vor fi restricționate de anumite condiții. Daca se încearcă ștergerea unui client care are deja asociată o comanda, va trebui afișat un mesaj corespunzător și clientul nu va putea fi șters. De asemenea, dacă se încearcă ștergerea unui produs care este inclus în cel puțin o comandă, se va afișa în interfață un mesaj corespunzător și produsul nu va putea fi șters. În ambele cazuri descrise, este necesară ștergerea comenzilor corespunzătoare înaintea ștergerii clienților/produselor.

De asemenea, fiecare produs stocata are o anumită cantitate disponibilă. Dacă se încearcă efectuarea unei comenzi cu produsul respectiv într-o cantitate mai mare decât cea disponibilă, în interfață se va afișa un mesaj corespunzător și comanda nu va putea fi adăugata. Se poate reîncerca adăugarea comenzii în limita stocului disponibil.

Odată ce un produs a fost adăugat într-o comandă cu o anumită cantitate, va trebui să recalculăm cantitatea rămasă. Produsul respectiv din baza de date va fi actualizat cu o nouă cantitate, rezultată din diferența cantității inițiale și cantității care a fost adaugată la comandă.

Interfața va trebui să cuprindă cele 3 ferestre pentru fiecare obiect: client, produs și comandă. În fereastra corespunzătoare comenzilor va trebui sa avem câmpuri dedicate pentru adăugarea unei comenzi noi, folosind datele pe care deja le știm despre produsul care va fi adăugat în comandă și clientul asociat acesteia.

Ferestrele corespunzătoare clienților și produselor vor avea câmpuri și etichete care să permită realizarea operațiilor de tip CRUD (create, read, update, delete) pe obiectele de tip Client și Produs.

Modelare

Pentru a modela problema propusă vom avea în vedere 3 modele: Client, Produs și Comandă. Clientul va avea ca atribute un id pentru identificare facilă, numele, adresa de e-mail și adresa de domiciliu. Produsul va conține id-ul specific, numele său, cantitatea de produs disponibilă (care se poate comanda) și prețul unui produs. Comanda va avea ca atribute id-ul comenzii, id-ul corespunzător clientului asociat comenzii, id-ul corespunzător produsului care va fi comandat, cantitatea dorită de produs și prețul total.

Conexiunea cu baza de date va fi realizată într-o clasa dedicată. Pentru a nu instanția și a deschide mai multe conexiuni, voi folosi design pattern-ul Singleton, care permite o singură instanțiere a clasei respective. Obținerea conexiunii cu baza de date o voi asigura prin crearea unei metode getter.

Baza de date va avea 3 tabele: clients, orders și products. Pentru fiecare tabel, id-ul corespunzător obiectelor descrise va fi primary key pentru tabel. În tabelul dedicat comenzilor, id-ul clientului asociat comenzii și id-ul produsului comandat vor fi foreign key-uri ale tabelului și vor referenția tabelul de clienți și tabelul de produse.

Partea de interfață grafică va avea 3 clase diferite, câte una pentru fiecare tip de model. Fiecare clasă va descrie o fereastră.

Scenariu

Scenariul aplicației cuprinde partea de modelare a produselor, clienților și comenzilor, implementarea operațiilor și interacțiunea utilizatorului cu aplicația prin interfața grafică. Prin intermediul acesteia, utilizator poate sa realizeze operațiile de tip CRUD pe clienți și pe produse, să vizualizeze comenzile și să adauge comenzi noi.

În ferestrele dedicate clienților și produselor putem vizualiza tabelele corespunzătoare obiectelor prin apăsarea butonului “show”. Ferestrele vor fi actualizate și tabelele vor fi populate cu date despre fiecare client/ produs. În partea de jos a ferestrelor se regăsesc textbox-uri corespunzătoare pentru adăugarea și actualizarea clienților/produselor. Pentru a adăuga un nou client/produs se vor completa textfield-urile corespunzătoare cu valori potrivite și se va apăsa butonul “Add”. Pentru a face refresh la datele din tabel și a vizualiza modificarea este necesară apăsarea butonului “Show”.

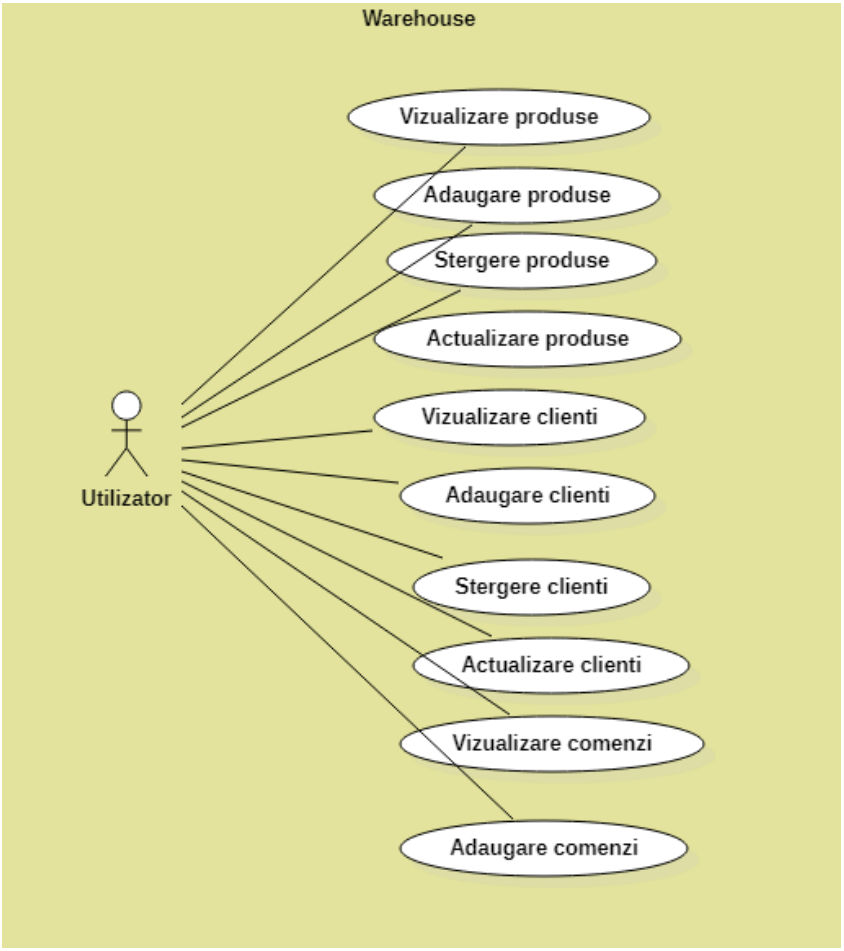
Actualizarea produselor/clienților se face urmând aceiași pași, singura diferență fiind necesitatea selectării unei linii din tabel, corespunzătoare clientului/produsului care se dorește a fi actualizat. După completarea câmpurilor cu valori potrivite se va apăsa butonul “Update” și se va face refresh la datele din tabel prin apăsarea butonului “Show”.

Fereastra corespunzătoare comenzilor permite vizualizarea tabelului populat cu date despre toate comenzile prin apăsarea butonului “Show”. În această fereastra este permisă doar funcționalitatea de adăugare a unei noi comenzi, completând textfield-urile cu valori potrivite si apăsând butonul “Add”.

Interfața reprezintă un mediu facil de utilizare și este foarte intuitivă.

La fiecare adăugare cu succes a unei noi comenzi, aplicația generează o factură sub forma unui fișier text, în care apar datele despre comandă. După fiecare adăugare de comanda putem observa în directorul aplicației noul fișier text corespunzător comenzii adăugate. Numele fișierului conține id-ul comenzii, pentru a putea fi identificat mai ușor.

Diagrama cazurilor de utilizare



Cazurile de utilizare cuprinde cele 8 funcționalități ale aplicației : operațiile de tip CRUD pe clienți și pe produse, alături de vizualizarea comenzilor existente și adăugarea unei noi comenzi.

Diagrama conține un singur tip de actor, utilizatorul aplicației.

3.Proiectare

Arhitectura proiectului este bazată pe layered architecture style. Stilul arhitectural stratificat este unul dintre cele mai comune stiluri arhitecturale. Ideea din spate este că modulele sau componentele cu funcționalități similare sunt organizate în straturi orizontale. Ca rezultat, fiecare strat îndeplinește un rol specific în cadrul aplicației. Stilul de arhitectură stratificat nu are o restricție privind numărul de straturi pe care aplicația le poate avea, deoarece scopul este de a avea straturi care promovează conceptul de separare.

Aplicația construită conține 4 nivele/pachete: dataAccessLayer, businessLayer, model and presentation.

Pachetul dataAccessLayer (DAO) se ocupa cu modelarea datelor stocate în baza de date. În acest pachet avem clase care conțin query-uri ce vor fi aplicate bazei de date pentru a o modifica/pentru a extrage informațiile din ea.

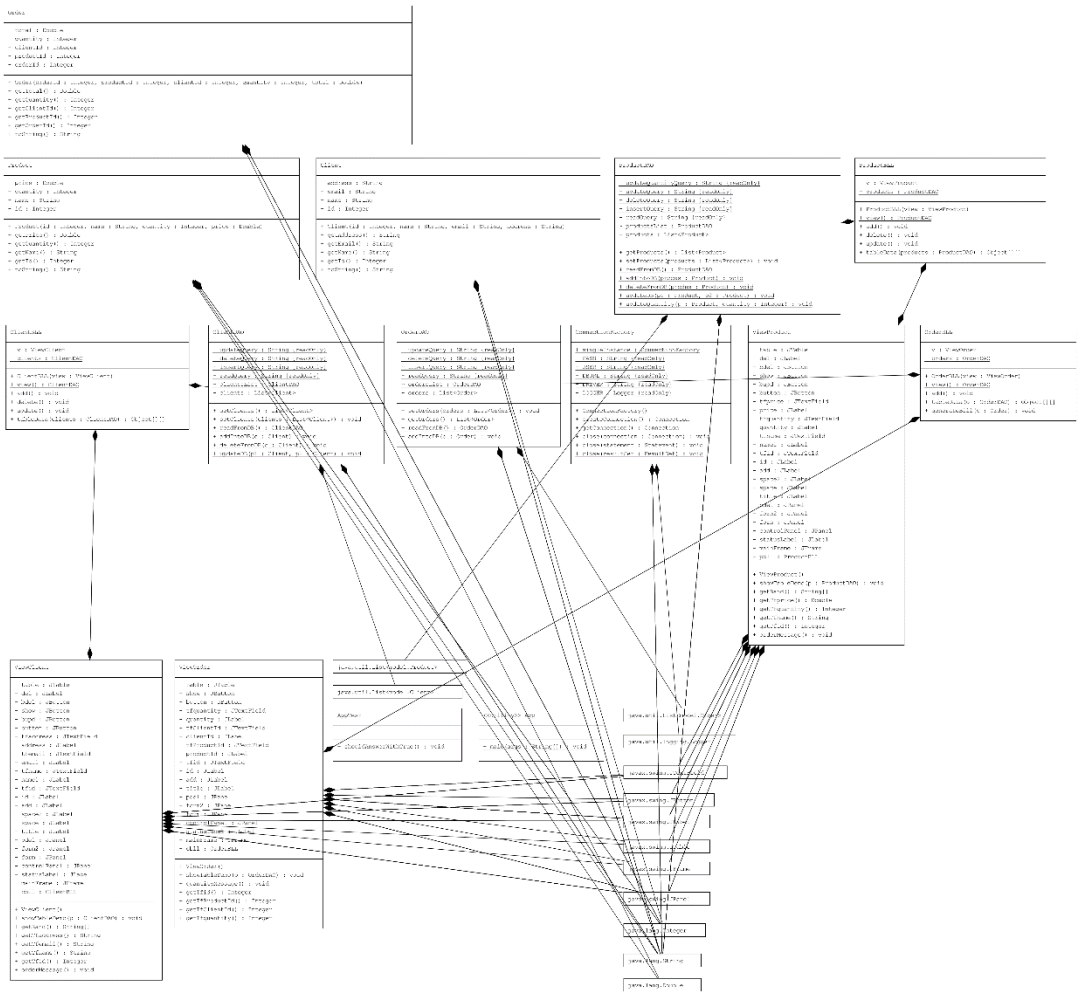
Pachetul de businessLayer conține logica funcționalităților care vor fi implementate. Informațiile sunt extrase din nivelul DAO, iar acest pachet face conexiunea cu parte de presentation a aplicației.

Pachetul model conține cele 3 modele ale aplicației: client, produs și comandă.

Pachetul presntation conține view-urile corespunzătoare celor 3 ferestre din interfața grafică. Fiecare view conține butoane, care la apăsare apelează metode implementate în partea de businessLogic.

Pentru documentarea claselor și facilitarea înțelegerii codului am folosit JavaDoc. Clasele, atributele și metodele implementate sunt explicate pe larg.

Diagrama UML de clase



În schema de mai sus este reprezentată diagrama UML de clase. Clasele cuprinse în proiect sunt:

-Clasa Cient: modelează clienții care efectuează comenzi. Ca atribute regăsim id-ul clientului, numele său, adresa de e-mail și adresa de domiciliu. În clasă regăsim un constructor si 4 metode de tip getter pentru a obține atributele unui obiect, cât si o metodă care transformă un obiect de tip client într-un String. Ultima metodă a fost implementată pentru a facilita verificarea funcționalităților pe parcursul construirii aplicației.

-Clasa Product: modelează produsele disponibile în depozit și are ca atribute id-ul produsului, denumirea sa, cantitatea de produs disponibilă în depozit și prețul unui produs. În clasă regăsim un constructor si 4 metode de tip getter pentru a obține atributele unui obiect, cât si o metodă care transformă un obiect de tip produs într-un String. Ultima metodă a fost implementată pentru a facilita verificarea funcționalităților pe parcursul construirii aplicației.

-Clasa Ordrer: modelează comenzile generate și are ca atribute id-ul comenzii, id-ul produsului inclus în comandă, id-ul clientului asociat comenzii, cantitatea de produs comandată și prețul total al produselor din comandă. În clasă regăsim un constructor si 5 metode de tip getter pentru a obține atributele unui obiect, cât si o metodă care transformă un obiect de tip comandă într-un String. Ultima metodă a fost implementată pentru a facilita verificarea funcționalităților pe parcursul construirii aplicației.

-Clasa ConnectionFactory: Realizează conexiunea cu baza de date . Instanțiază un singur obiect al clasei care poate fi accesat prin metoda de get.

-Clasa ClientDAO: se ocupă de persistența datelor din baza de date. Are ca atribut o lisă de clienți care va fi construită pe baza înregistrărilor din tabela clients din baza de date. Conține query-uri sub formă de String-uri care vor fi aplicate pe baza de date și metode care preiau/adaugă înregistrări în tabelul clients din baza de date.

-Clasa ProductDAO: se ocupă de persistența datelor din baza de date. Are ca atribut o lisă de produse care va fi construită pe baza înregistrărilor din tabela products din baza de date. Conține query-uri sub formă de String-uri care vor fi aplicate pe baza de date și metode care preiau/adaugă înregistrări în tabelul products din baza de date.

-Clasa OrderDAO: se ocupă de persistența datelor din baza de date. Are ca atribut o lisă de comenzi care va fi construită pe baza înregistrărilor din tabela orders din baza de date. Conține query-uri sub formă de String-uri care vor fi aplicate pe baza de date și metode care preiau/adaugă înregistrări în tabelul orders din baza de date.

-Clasa ClientBLL: implementează logica de business pentru partea de clienți. Are ca atribute un obiect de tip ClientDAO și un obiect de tip ViewCient. Metodele implementate fac legătura dintre aceste 2 clase care aparțin layerelor diferite din arhitectură.

-Clasa ProductBLL: implementează logica de business pentru partea de produse. Are ca atribute un obiect de tip ProductDAO și un obiect de tip ViewProduct. Metodele implementate fac legătura dintre aceste 2 clase care aparțin layerelor diferite din arhitectură.

-Clasa OrderBLL: implementează logica de business pentru partea de comenzi. Are ca atribute un obiect de tip OrderDAO și un obiect de tip ViewOrder. Metodele implementate fac legătura dintre aceste 2 clase care aparțin layerelor diferite din arhitectură.

-Clasa ViewClient: implementează fereastra corespunzătoare clienților. Descrie partea de interfață grafică unde se afișează tabelul de clienți și se pot efectua operații de tip CRUD.

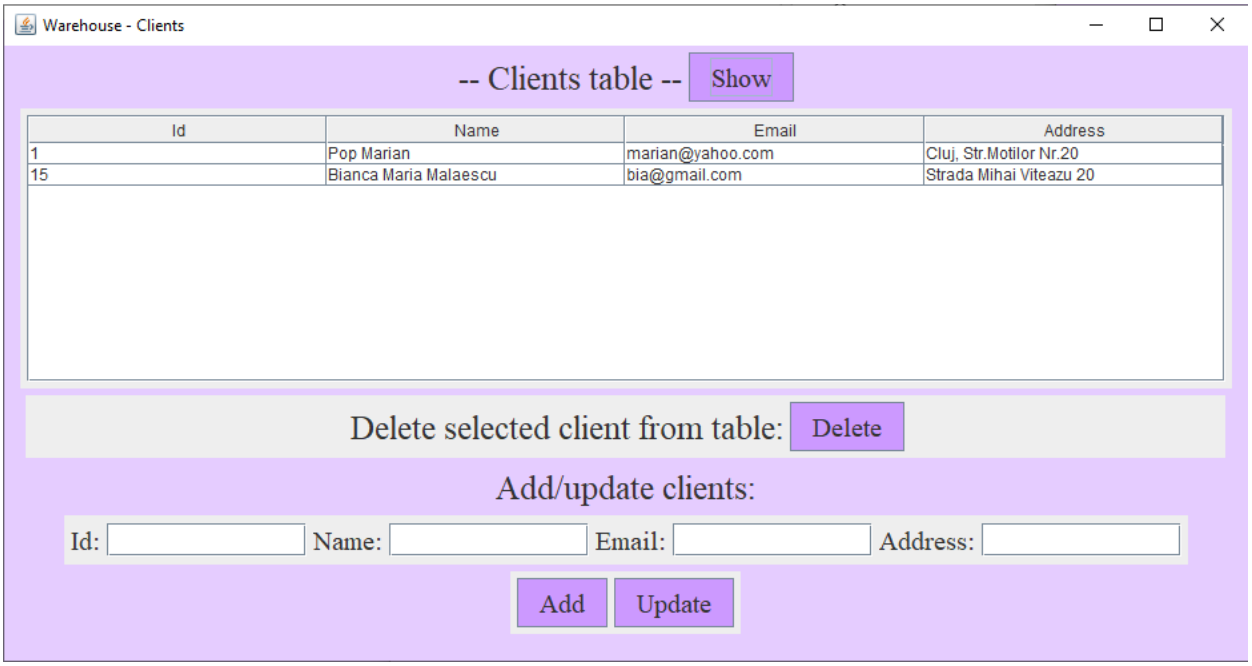
-Clasa ViewProduct: implementează fereastra corespunzătoare produselor. Descrie partea de interfață grafică unde se afișează tabelul de produse și se pot efectua operații de tip CRUD.

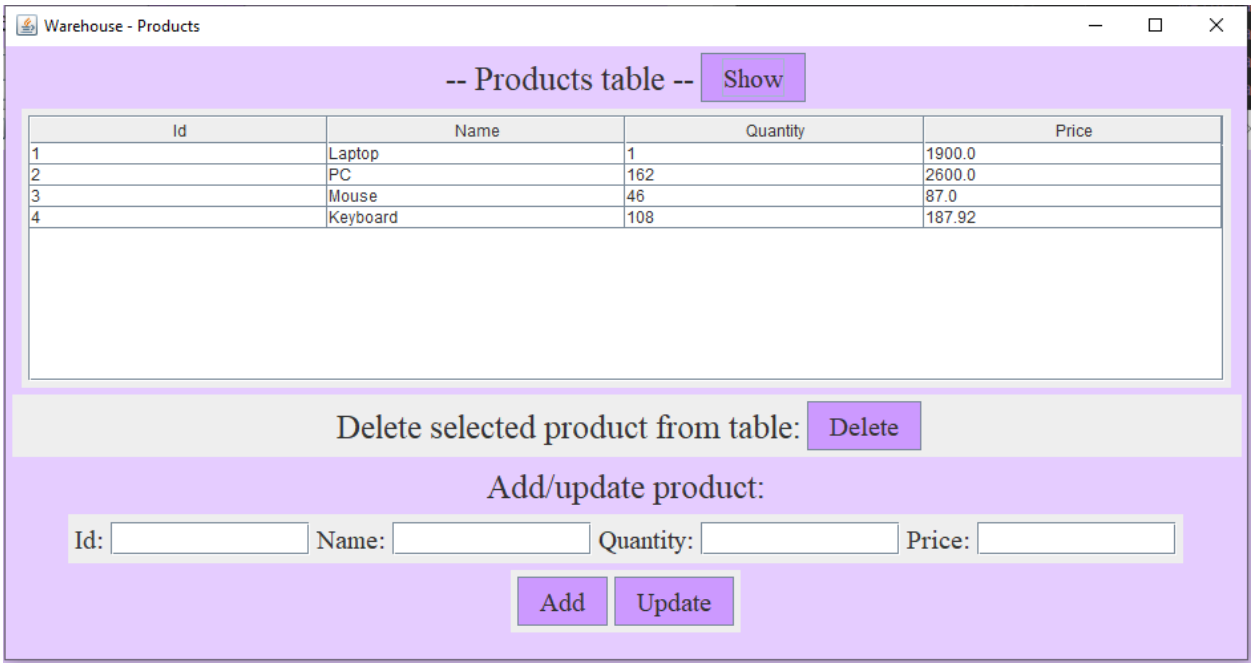
-Clasa ViewOrder: implementează fereastra corespunzătoare comenzilor. Descrie partea de interfață grafică unde se afișează tabelul de comenzi și se poate efectua operația de adăugare.

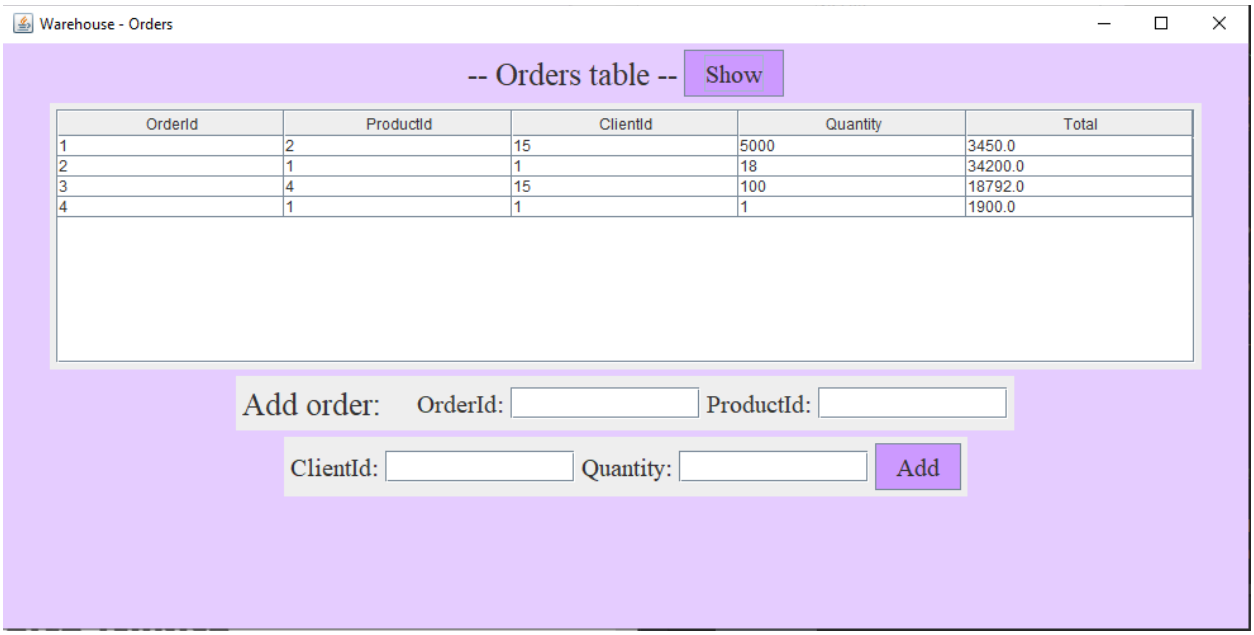
Interfața grafică

Prin intermediul interfeței grafice construite, utilizatorul poate efectua operații de tip CRUD pe tabela de clienți si cea de produse, și poate vizualiza comenzile și adăuga comenzi noi .

Mai jos sunt ilustrate cele 3 ferestre ale interfeței grafice cu utilizatorul.







4.Implementare

Aplicația "Orders management" a fost implementată în limbajul de programare Java, utilizând șablonul arhitectural „Layered architecture". Am folosit ca framework Swing pentru a implementa partea de interfață grafică. Swing face parte din clasa de bază Java și este folosit pentru a crea aplicații desktop. Include componente ca butoane, textfield-uri, etichete etc. Compunerea acestor componente rezultă o GUI (graphical user interface).

Metodele claselor

-Clasa Client: are ca metode 4 getere și metoda de transformare a unui obiect de tip Client într-un String, pentru a putea fi vizualizat mai ușor.

-Clasa Product: are ca metode 4 getere și metoda de transformare a unui obiect de tip Product într-un String, pentru a putea fi vizualizat mai ușor.

-Clasa Order: are ca metode 5 getere și metoda de transformare a unui obiect de tip Order într-un String, pemtru a putea fi vizualizat mai ușor.

-Clasa ClientDAO: conține metoda de citire a datelor din tabela clients din baza de date, metoda de adăugare a unui nou client în tabela clients, metoda de ștergere a unei înregistrări din tabela clients și actualizarea unei înregistrări din tabela clients.

-Clasa ProductDAO: conține metoda de citire a datelor din tabela products din baza de date, metoda de adăugare a unui nou produs în tabela products, metoda de ștergere a unei înregistrări din tabela products și actualizarea unei înregistrări din tabela products.

-Clasa OrderDAO: conține metoda de citire a datelor din tabela orders din baza de date și metoda de adăugare a unei noi înregistrări în tabela orders.

-Clasa ClientBLL: conține metodele de citire, adăugare, ștergere și actualizare a listei de clienți obținută din extragerea datelor din baza de date realizată în clasa ClientDAO, cât și o metodă care primește lista de clienți și

returnează o matrice de tip Object populată cu valorile tuturor obiectelor de tip Client, care va fi afișată în tabelul corespunzător din interfață.

-Clasa ProductBLL: conține metodele de citire, adăugare, ștergere și actualizare a listei de produse obținută din extragerea datelor din baza de date realizată în clasa ProductDAO, cât și o metodă care primește lista de produse și returnează o matrice de tip Object populată cu valorile tuturor obiectelor de tip Product, care va fi afișată în tabelul corespunzător din interfață.

-Clasa OrderBLL: conține metodele de citire și adăugare a listei de comenzi obținută din extragerea datelor din baza de date realizată în clasa OrderDAO, o metodă care primește lista de comenzi și returnează o matrice de tip Object populată cu valorile tuturor obiectelor de tip Order, care va fi afișată în tabelul corespunzător din interfață, și metoda de generare a unei facturi sub forma unui fișier text pentru fiecare comandă adăugata cu succes.

-Clasa ViewClient: conține getere pentru textfield-urile din interfață, o metodă care populează tabelul cu datele din ClientBLL, o metodă de get pentru un rând selectat din tabel și o metodă care afișează un mesaj în interfață.

-Clasa ViewProduct: conține getere pentru textfield-urile din interfață, o metodă care populează tabelul cu datele din ProductBLL, o metodă de get pentru un rând selectat din tabel și o metodă care afișează un mesaj în interfață.

-Clasa ViewOrder: conține getere pentru textfield-urile din interfață, o metodă care populează tabelul cu datele din OrderBLL și o metodă care afișează un mesaj în interfață.

5.Rezultate

Aplicația rulează bine, funcționalitățile sunt îndeplinite cu succes și rezultatele obținute în urma operațiilor sunt cele dorite.

6.Concluzii

Realizarea proiectului m-a familiarizat cu șablonul arhitectural „Layered Architecture” și cu framework-ul Swing, foarte util pentru realizarea interfeței. Lucrul cu baza de date MySQL a facilitat realizarea aplicației. Am reușit să implementez toate funcționalitățile cerute.

7.Bibliografie

[1]: https://www.javatpoint.com/java-swing

[2]: https://www.baeldung.com/java-jdbc

[3]: https://www.baeldung.com/javadoc

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

[5]: https://dzone.com/articles/layers-standard-enterprise

[6]: https://dsrl.eu/mantal/pt/