MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICA SI CALCULATOARE
SPECIALIZAREA CALCULATOARE SI TEHNOLOGIA INFORMATIEI

TEHNICI DE PROGRAMARE

- ORDER MANAGEMENT -

MURESAN GEORGIANA ROBERTA
GRUPA 30223
2020-2021

CUPRINS

1. OBIECTIVUL TEMEI

Cerinta temei: Considerati o aplicatie “Order Management” pentru a procesa comenzile unor clienti dintr-un depozit, inserarea de clienti si produse noi, dar si stergerea lor. O baza de date este folosita pentru a stoca informatiile despre produse, clienti si comenzi. Aplicatia ar trebui sa permita procesarea comenzilor dintr-un fisier text dat ca argument, sa rezolve operatiile solicitate, sa salveze datele in baza de date si sa genereze rapoarte in format pdf.

Obiectivul acestei teme este de a realiza anumite operatii specifice unui depozit(realizarea comenzilor, adaugare/ stergere clienti, adaugare/stergere produse). Aplicatia trebuie sa fie capabila de a interpreta o anumita operatie citita din fisier si de a o realiza. Baza de date trebuie sa stocheze datele si sa fie capabila de a suporta operatiile respective. In baza de date se vor retine informatii despre clienti, produse si comenzi. Despre un client se vor retine doua date: numele si adresa, produsele vor fi definite prin nume, pret si cantitate. O comanda se va realiza sau nu, in functie de datele anterior stocate despre clienti si comenzi.

2. Analiza problemei, modelare scenarii, cazuri de utilizare

Cerința aplicației este de a dezvolta o aplicație de gestiune a unui depozit pentru a procesa comenzile clienților si care utilizează o baza de date relațională pentru stocarea produselor, clienților si a comenzilor.

Provocarea acestei aplicații o reprezintă conectarea la o baza de date MySQL care stochează datele necesare gestionarii depozitului si oferă utilizatorului posibilitatea sa vadă si sa acceseze aceste date prin intermediul interfeței grafice ale aplicației Java.

Accesarea unei baze de date dintr-o aplicatie Java se realizeaza prin intermediul unui program de comanda (driver) specific unui anumit sistem de gestiune a bazelor de date. Un driver intermediază legătura dintre aplicatii si baze de date.

Java DataBase Connectivity JDBC reprezinta un API care permite lucrul cu baze de date relationale. Prin intermediul JDBC sunt transmise comenzi SQL la un server de baze de date. Folosind JDBC, nu este necesara dezvoltarea mai multor aplicatii pentru a accesa servere de baze de date care utilizeaza sisteme diferite de gestiune a bazelor de date (Oracle, MySQL, Sybase). Este suficienta o singura aplicatie, care sa utilizeze API-ul JDBC, pentru a transmite comenzi SQL la serverul de baze de date dorit. In felul acesta este asigurata portabilitatea aplicatiei.

Driver-ul JDBC ofera acces uniform la bazele de date de tip relational. JDBC include clase si interfete, scrise in Java, care furnizeaza o interfata SQL standard pentru proiectantii de aplicatii cu baze de date. Clasele si interfetele necesare in API-ul JDBC sunt disponibile prin intermediul pachetului java.sql.

Accesarea unei baze de date din Java, via JDBC, presupune realizarea urmatorilor pasi:

- stabilirea unei conexiuni la serverul de baze de date si selectarea unei baze de date;
- transmiterea si rularea unor secvente SQL;
- preluarea si prelucrarea rezultatelor obținute.

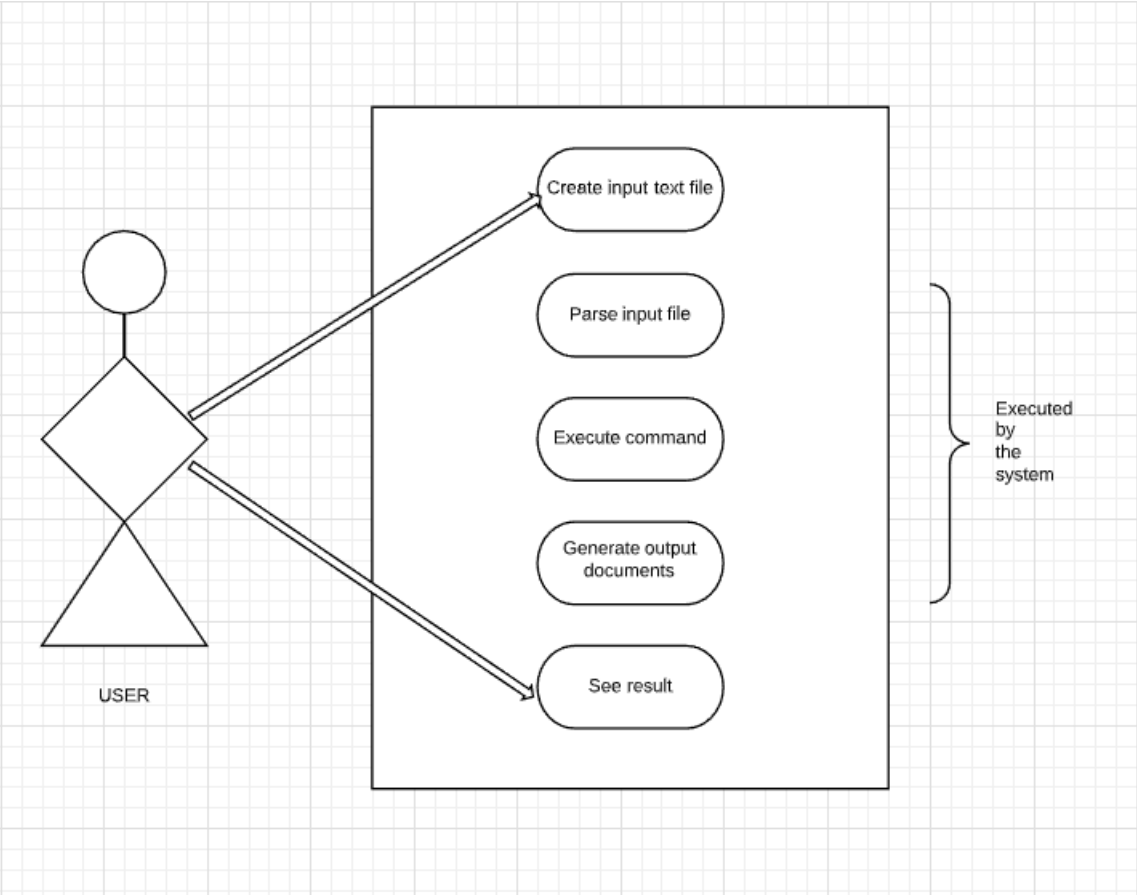
Cu ajutorul acestei aplicatii utilizatorul poate interactiona cu aplicatia intr-un mod simplu si eficient, fara a avea in prealabil cunostinte de programare sau de lucru cu baze de date relationale. Modul de stocare al datelor si organizarea acestora in tabele prin intermediul bazei de date este transparent pentru utilizator, acesta putand vizualiza si modifica aceste date prin intermediul interfetei grafice.

Utilizatorul poate sa efectueze urmatoarele operatii:

- asupra clientilor: adaugarea unui nou client, editarea datelor unui client existent, stergerea unui client si vizualizarea unei liste a tuturor clientilor.
- asupra produselor: adaugarea unui nou produs, editarea datelor unui produs, stergerea unui produs, vizualizarea unei liste a tuturor produselor.

• sa plaseze o comanda noua cu detaliile aferente comenzii (datele clientului, data si ora comenzii, produsele comandate)

Dupa plasarea comenzii, trebuie sa se genereze factura clientului care contine detaliile comenzii sub forma unui fisier text sau pdf.



Organizare structurata(tabelar) a cerintelor intr-o baza de date:

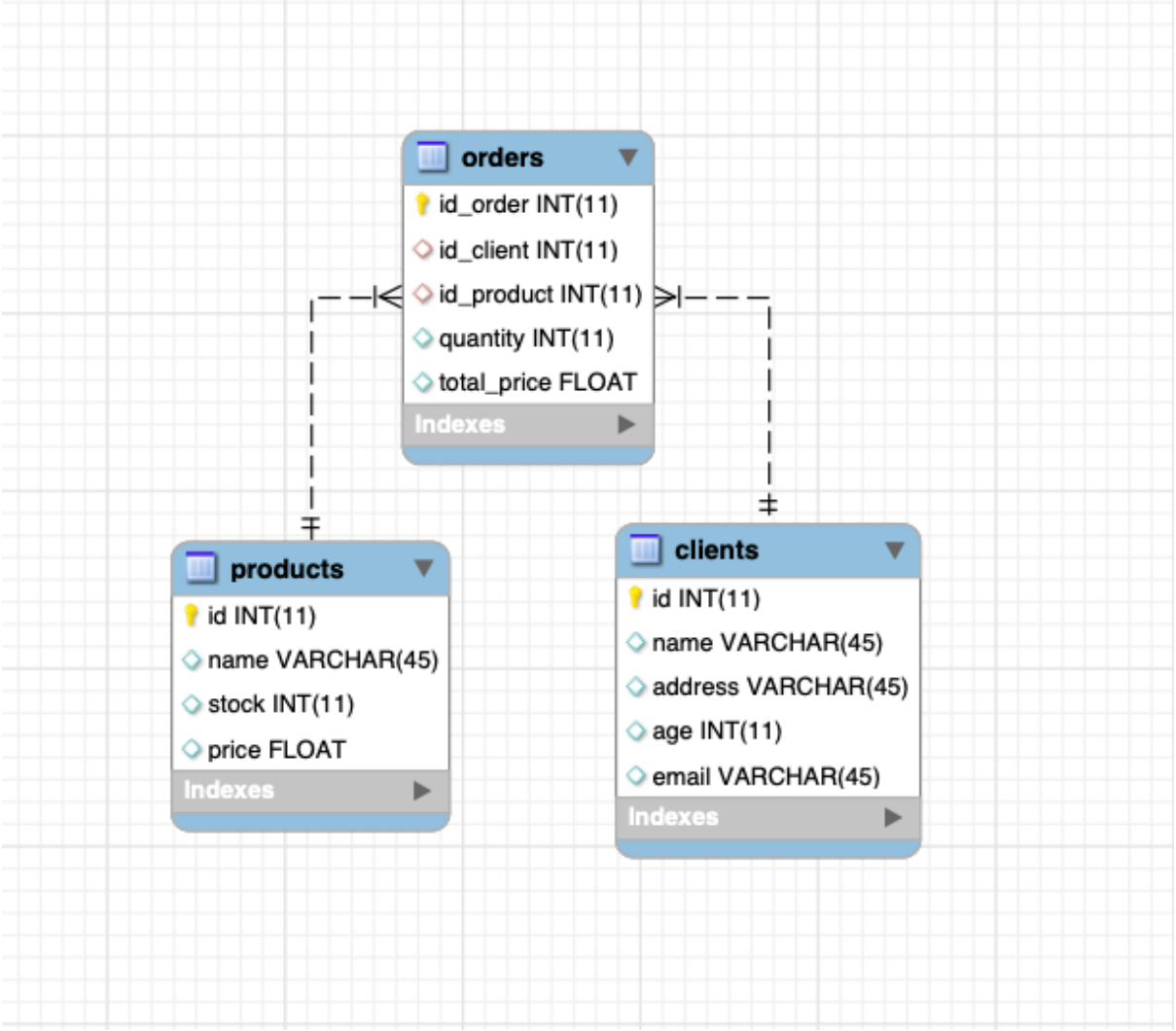
Baza de date trebuie sa stocheze următoarele informatii:

-informatii despre un client: id, nume, adresa, varsta, email

-informații despre un produs: id, nume, pret, cantitate in stoc

-informații despre o comanda: id, id-ul clientului care plasează comanda, id-urile produselor din comanda respectiva si cantitățile comandate din fiecare produs

Modelul relational ce sta la baza acestei aplicații este prezentat in figura următoare si este stocat intr-o baza de date MySQL:



Tabelul clients conține si ID-uri, care sunt de fapt chei străine din celelalte tabele. Toate acestea sunt construite pe cascadă de ștergere, deci la ștergerea unui client, de exemplu, ordinea corespunzătoare acestuia și toate articolele comenzii sunt șterse, de asemenea.

3. PROIECTARE

3.1. Decizii de proiectare

In construirea acestei aplicatii, si dupa cum specificau cerintele am ales sa folosesc o arhitectura pe mai multe niveluri (eng. multitier architecture).

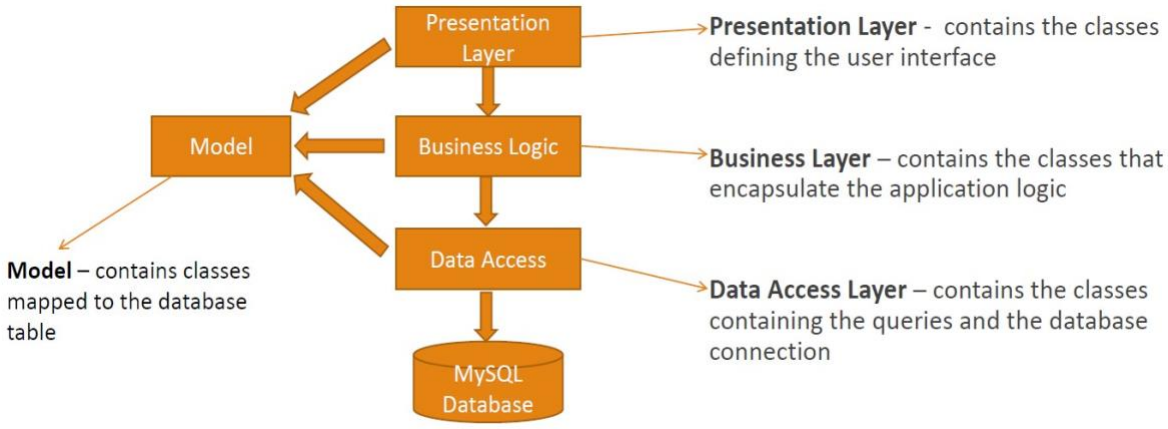
Arhitectura de aplicații N -tier oferă un model prin care dezvoltatorii pot crea aplicații flexibile și reutilizabile. Prin separarea unei aplicații în niveluri, dezvoltatorii dobândesc opțiunea de a modifica sau adăuga un anumit strat, în loc să refacă întreaga aplicație. O arhitectură logică multistratificată pentru un sistem informațional cu un design orientat obiect este compusă de obicei din mai multe straturi dupa cum urmeaza:

• Stratul de prezentare (cunoscut și ca stratul UI, stratul de vizualizare, nivelul de prezentare în arhitectura pe mai multe niveluri)

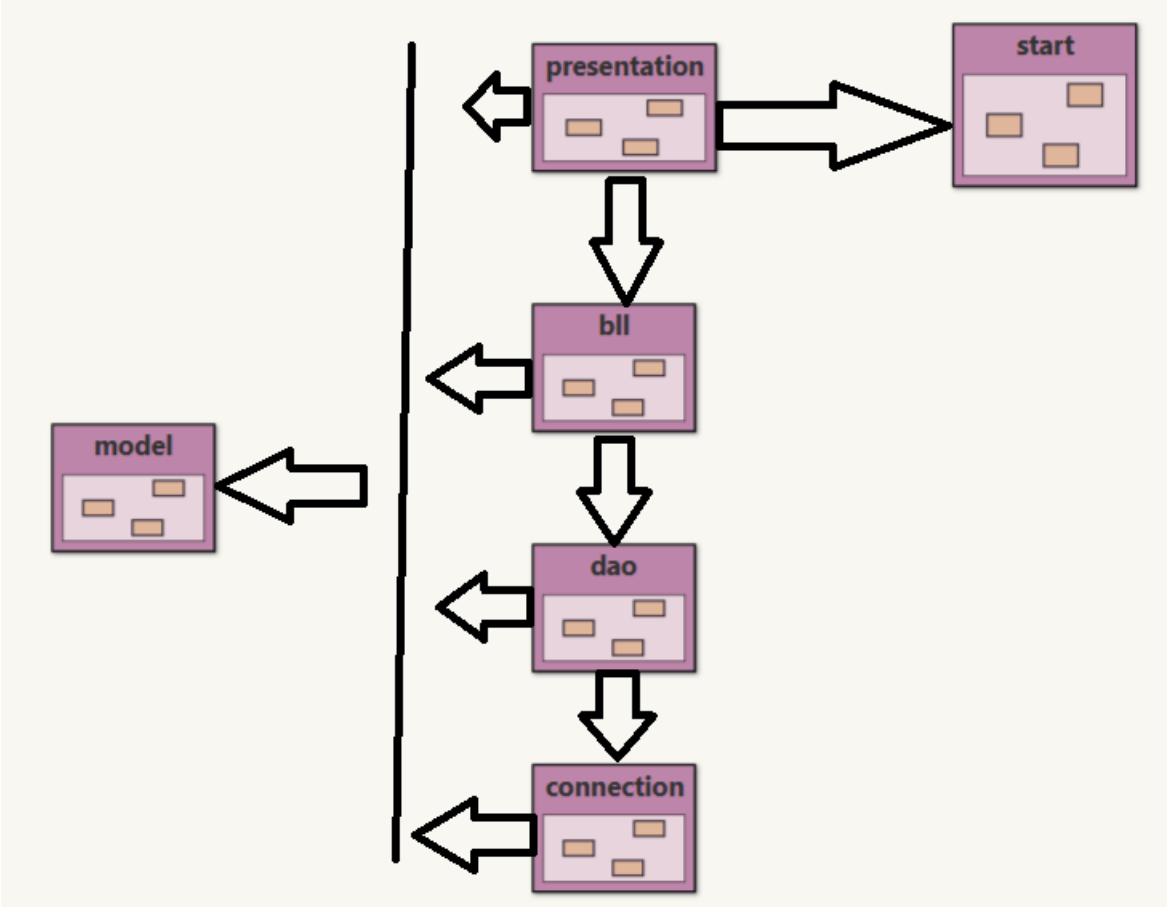
• Stratul de aplicație (cunoscut și ca strat de serviciu sau stratul de control GRASP)

• Stratul de afaceri (alias strat de logică de afaceri (BLL), strat de domeniu)

• Stratul de acces la date (cunoscut și ca strat de persistență , jurnalizare, rețea și alte servicii care sunt necesare pentru a sprijini un anumit strat de afaceri)



3.2. Diagrame UML, clase, pachete



Chiar baza proiectului este pachetul model, care conține la fel de multe clase ca baza de date, și anume trei, fiecare fiind echivalentul unui tabel. Acestea sunt utilizate pentru a simula practic o înregistrare a bazei de date și instanțele de obiecte ale acestor clase sunt utilizate în toate celelalte clase pentru a menține o ordine și un nivel de organizare în cod.

a) **Pachetul Model** contine urmatoarele clase: Client, Product si Orders si mapeaza tabelele din baza de date in clase Java.

Fiecare dintre aceste clase are in baza de date un tabel corespondent, iar variabilele sale de instanta se portrivesc cu numele coloanelor din acel tabel.

b) **Pachetul connection –** realizeaza conexiunea principala la baza de date.

c) **Pachetul DAO** contine urmatoarele clase: AbstractDAO, ClientsDAO, ProductDAO, OrdersDAO si si reprezinta nivelul in care se realizeaza operatiile cu baza de date (interogarile).

Clasa AbstractDAO

Contine implementari generice pentru operatiile CRUD (insert, find, update si delete) folosind tehnici de inspectare a claselor- Java Reflection.

Prin intermediul acestei clase am furnizat principalele implementari de care vom avea nevoie in aceasta aplicatie pentru operatiile efectuate asupupra clientilor, produselor si comenzilor.

Dupa cum am specificat si inainte, accesarea unei baze de date se realizează printr-un driver.

Astfel de operatii care lucreaza cu baza de date, urmaresc pasii de mai jos:

> Stabilire conexiune server: o conexiune la un server de baze de date reprezinta un canal de comunicatii prin care sunt transmise cereri SQL si sunt returnate raspunsuri corespunzatoare. Stabilirea unei conexiuni dintr-o aplicatie Java presupune inregistrarea (incarcarea) unui driver si realizarea conexiunii propriu-zise prin intermediul clasei DriverManager din pachetul java.sql.

Utilizand clasa DriverManager, stabilirea efectiva a unei conexiuni la un server de bazede date necesita specificarea unui URL prin intermediul unei cereri care are urmatoarea forma:

```
Connection conn = DriverManager.getConnection(url, nume_utilizator, parola);
```

> Rulare comenzi SQL: Dupa stabilirea unei conexiuni la serverul de baze de date si selectarea unei baze de date active este necesara crearea unei instante de tip PreparedStatement prin metoda prepareStatement() a clasei Connection. Instanta de tip Statement permite manipularea unor comenzi SQL.

Un obiect de tip Statement este un container care permite transmiterea si rularea comenzilor SQL, si obtinerea rezultatelor corespunzatoare prin intermediul conexiunii asociate. Pe langa interfata Statement, mai pot fi utilizate doua subinterfete ale acesteia: PreparedStatement si CallableStatement.

```
Connection connection = DriverManager.getConnection(url);

PreparedStatement stmt = connection.prepareStatement();
```

Rularea unei comenzi SQL poate fi realizata prin trei metode:

• **executeQuery():** este utilizata pentru a rula comenzi SQL care returneaza un obiect de tip ResultSet;

```
public Product findProductsByName(String name){
.............................some code here.....................................
String query="SELECT * FROM product WHERE name LIKE '" +name+ "%'";'
PreparedStatement statement = connection.prepareStatement(query);
ResultSet resultSet = statement.executeQuery();
...........................another code here...................................
}
```

• **executeUpdate():** este utilizata pentru a rula comenzi SQL care permit manipularea datelor (INSERT, UPDATE, DELETE – returneaza numarul de inregistrari afectate de rularea comenzii SQL) sau modificarea structurii unui tabel din baza de date (CREATE, ALTER, DROP – returneaza valoarea 0);

```
public boolean update(T t) {
.............................some code here.....................................
PreparedStatement statement = connection.prepareStatement(query); int changedRows
= statement.executeUpdate(query);
...........................another code here...................................
}
```

• **execute():** poate fi utilizata pentru a rula orice tip de comanda SQL.

Manipulare si prelucrare rezultate: De cele mai multe ori rularea unei comenzi SQL pe o baza de date are ca si rezultat un set de date care apar sub forma tabelara. Pentru a obtine date dintr-o baza de date pot fi rulate comenzi SQL prin intermediul metodei executeQuery(). Aceasta metoda returneaza informatia sub forma unor linii de date (inregistrari), care pot fi accesate prin intermediul unei instante de tip ResultSet.

```
Connection connection = DriverManager.getConnection(url);
Statement stmt = connection.createStatement();
String sql = "SELECT id, firstName, lastName, address FROM client";
ResultSet rs = stmt.executeQuery(sql);
```

Inregistrarile dintr-un obiect de tip ResultSet pot fi parcurse cu ajutorul metodei next(). De asemenea, accesarea valorilor corespunzatoare anumitor coloane poate fi realizata prin metode de tipul get<Type>() (getString(), getInt()). Coloanele dintr-un tabel pot fi referite prin intermediul numelui sau prin intermediul pozitiei coloanei in interiorul tabelului (prima coloana din tabel are pozitia 1).

```
while(rs.next()) {
    int id = rs.getInt("id");
    String firstName = rs.getString("firstName");
    String lastName = rs.getString("lastName");
}
```

Toate tabelele unei baze de date detin meta-date care descriu denumirile si tipurile de date specifice fiecarei coloane. In acest fel poate fi utilizata clasa ResultSetMetadata pentru a obtine numarul de coloane dintr-un tabel sau denumirile coloanelor.

```
ResultSetMetaData meta = rs.getMetaData(); for(int
i = 0; i < meta.getColumnCount(); i++)
    System.out.print(meta.getColumnName(i + 1) + "\t");
```

Clasa ClientsDAO

Contine metode specifice de care am avut nevoie in realizarea aplicatiei:

```
public static Clients findByName(String clientName)
public static String insert(Clients client)
public static String delete(Clients client)
public static ArrayList<String> selectColumnByName()
public static void editClient(Clients client)
public static void view()
```

Clasa ProductsDAO

Contine metode specifice de care am avut nevoie in realizarea aplicatiei:

```
-   public static Products findByName(String productName)
-   public static String insert(Products product)
-   public static String delete(Products product)
-   public static ArrayList<String> selectColumnByName()
-   public static void editProduct(Products product)
-   public static void view()
```

d) **Pachetul BLL** - contine clasa: ClientBLL, EmailValidator, ClientAgeValidator si interfata Validator. La nivelul acestui pachet sunt implementate cerintele functionale ale aplicatiei – operatii CRUD pe clienti, produse respectiv plasarea unei noi comenzi.

Aceste clase folosesc implementarile din stratul de acces la date, si in plus contin clase de tip <**Validator**> pentru a valida informatia care este transmisa din interfata grafica inspre baza de date, in acest fel eliminandu-se orice fel de erori sau inconsistente la nivelul datelor ce sunt stocate.

e) **Pachetul Presentation** contine clasele de GUI si WriteToFile si imbraca intr-un mod frumos modelul de date care sta la baza aplicatiei, facilitand interactiunea utilizatorului cu acesta prin ferestre dedicate fiecare operatii in parte.

Clasa WriteToFile - este contine metoda :

```
    printBill(String bill)
```

Si este folosita la generarea unui fisier text care contine pentru fiecare comanda nou creata si contine detaliile comenzii (informatiile clientului, produsele comandate, totalul de plata, data).

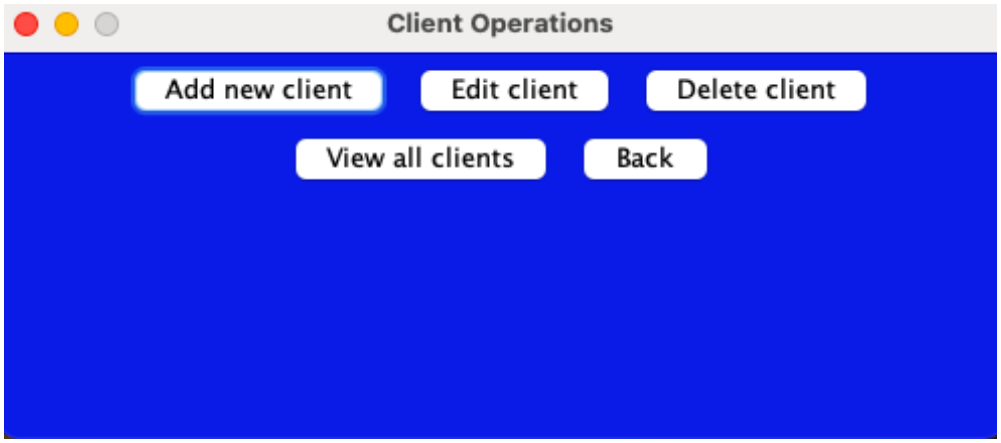
f) **Pachetul start –** contine metoda main.

3.3. Interfata utilizator

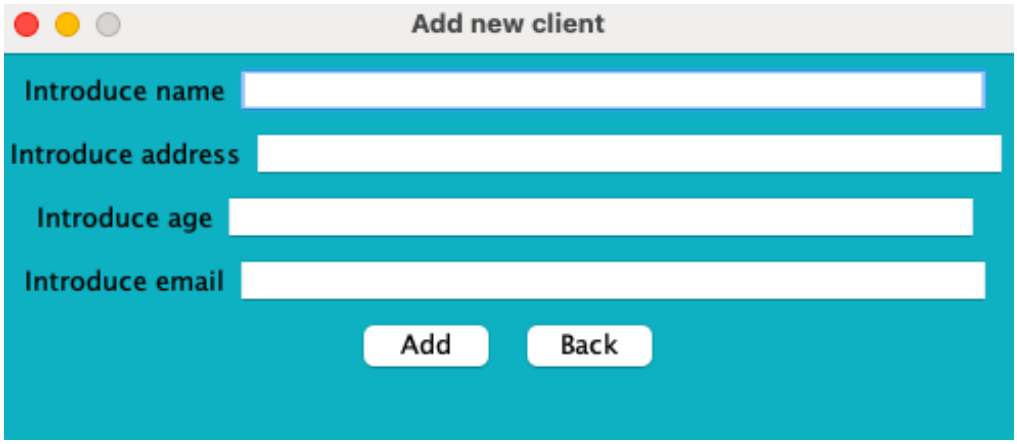
Am ales sa profit de flexibilitatea frame-urilor si de clasele Random si Color, cu scopul de a crea o interfata user-friendly care sa ii capteze atentia utilizatorului.

- **Pagina de start :**

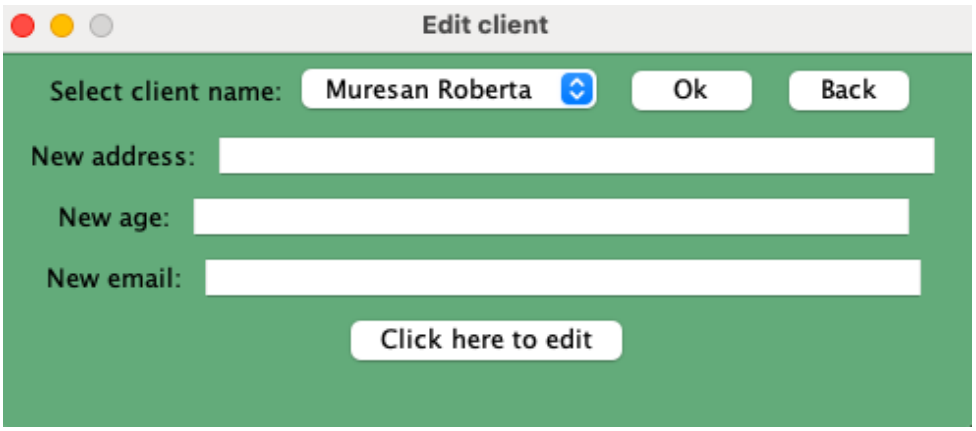
- **Operatii clienti:**



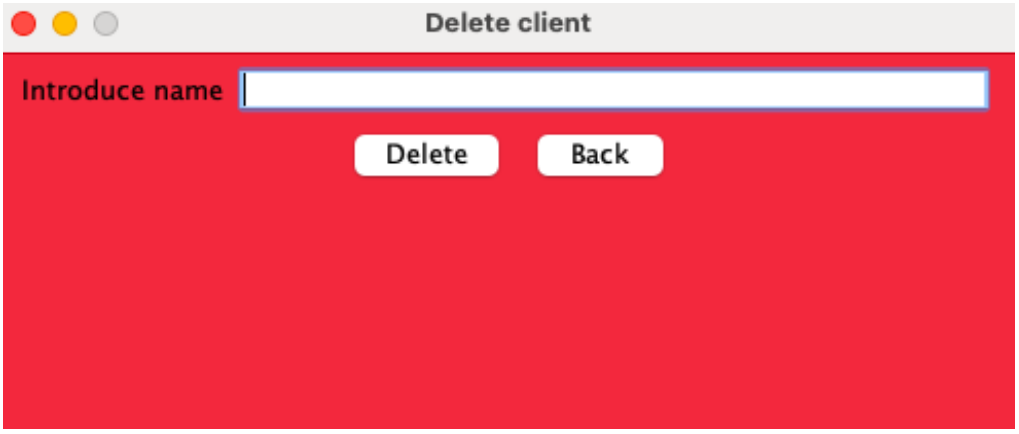
Adaugarea unui nou client:



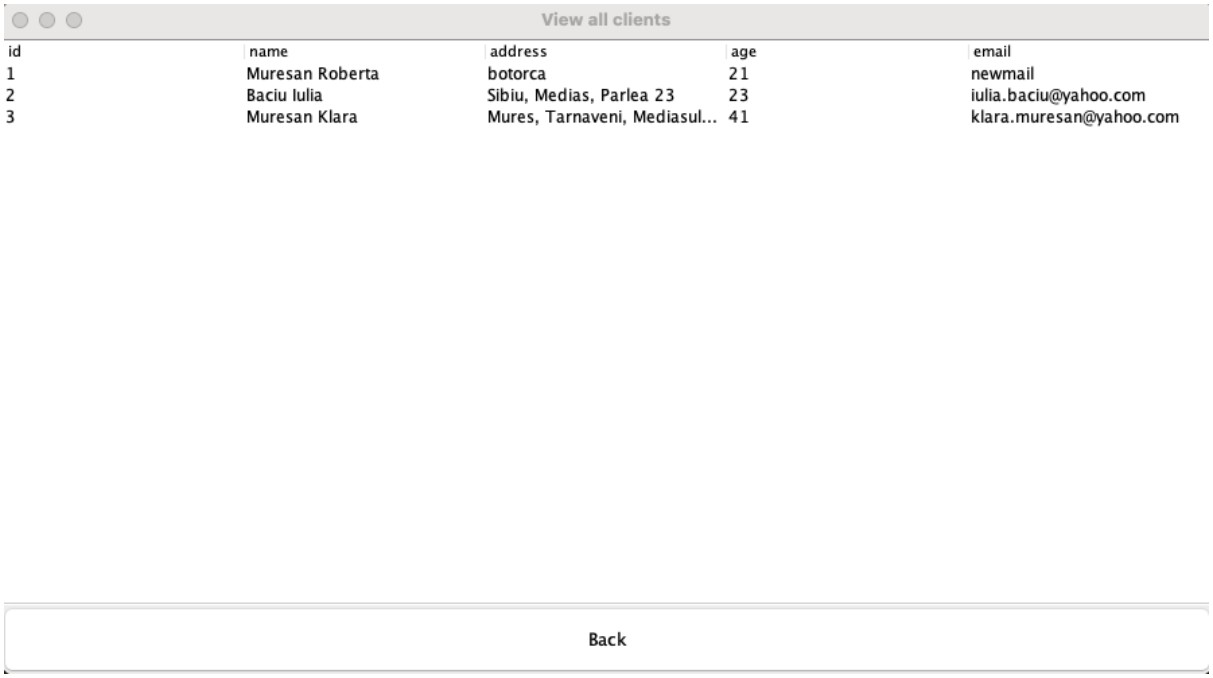
Editarea unui client:



Stergerea unui client:



Vizualizarea tuturor clientilor:



- **Operatii produse:**

Adaugarea unui nou produs.

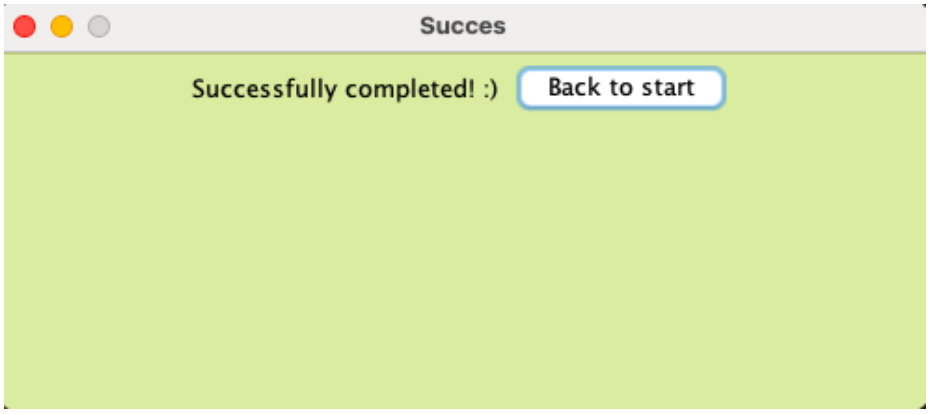
Editarea unui produs.

Stergerea unui produs.

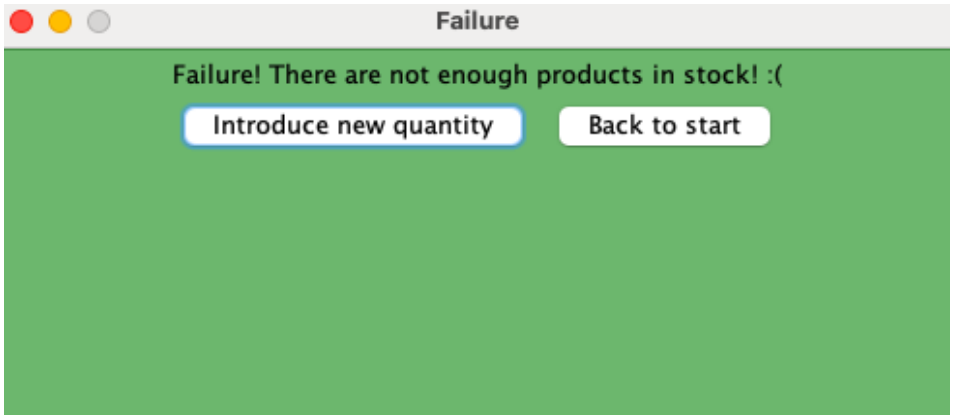
Vizualizarea tuturor produselor.

- **Plasarea unei comenzi:**

- **Success Window:**



- **Failure Window:**



4. Implementare

Proiectul “Orders Management” este o aplicatie desktop realizata in limbajul de programare Java folosind mediul de dezvoltare IntelliJ IDEA. Principalul scop al acesteia este sa gestioneze clientii, produsele si comenzile unui depozit intr-o maniera cat mai usoara si mai prietenoasa pentru utilizator.

Implementarea claselor de tip DAO si Validator se pot observa in cod.

5. Concluzii

Din aceasta tema am invatat rolul important pe care il au bazele de date, si cum imbinand un software cu o baza de date imbunatateste extrem de mult performanta unui serviciu, flexibilitatea, scalabilitatea si procesul de luare a deciziilor deoarece integrarea unei aplicatii care lucreaza cu baze de date simplifica gestionarea datelor, automatizeaza procesele manuale costisitoare si consumatoare de timp, eliberandu-le timp utilizatorilor si permitandu-le sa stocheze datele intr-o forma structurala si apoi sa le acceseze prin intermediul unei interfete grafice extrem de simplu si de eficient.

6. Idei pentru o implementare mai buna a proiectului

Din punctul meu de vedere, acest proiect are o implementara simpla care este abia baza unui program care poate fi folosit in viata de zi cu zi. Pe viitor, se pot adauga: optiunea de a comanda mai multe produse, optiunea de a alege daca te autoidentifici ca si un „client” sau „administrator”, optiunea de a alege modalitatea de a plati comanda si multe altele.

7. Bibliografie

- https://www.baeldung.com/java-jdbc
- http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/
- https://dzone.com/articles/layers-standard-enterprise
- http://tutorials.jenkov.com/java-reflection/index.html
- https://www.baeldung.com/java-pdf-creation
- https://www.baeldung.com/javadoc
- https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html

- ASSIGNMENT_3_SUPPORT_PRESENTATION.pdf
  - Simple layered project: https://gitlab.com/utcn_dsrl/pt-layered-architecture
  - Reflection example: https://gitlab.com/utcn_dsrl/pt-reflection-example