Managementul unui magazin utilizand o baza de date

**1.Cerinta:**

Se cere implementarea unei aplicatii software pentru gestionarea si procesarea comenzilor unor client. Gestionarea si procesarea comenzilor trebuie realizate cu ajutorul unei baze de date care monitorizeaza toate schimbarile facute. Pentru realizarea aplicatiei se cer cel putin urmatoarele clase:

* Clasele de acces la date (ConnectionFactory)
* Bussines Logic (Procesarea comenzii, Administrarea clientilor si a produselor)
* Presentation classes (Clasele ce implementeaza interfata grafica)
* Clasele specific (Client, Produs, Comanda)

Desigur ca orice alte clase realizate in plus fata de cele mentionate mai sus sunt bine venite atata timp cat aduc un plus aplicatiei.

**2.Obiectivul temei:**

Obiectivul acestei teme este proiectarea si implementarea unui program ce va fi folosit in procesarea unor comenzi . Aceste comenzi vor veni de la anumiti clienti , care vor dori sa cumpere produse in anumite cantitati de la magazin . Produsele se pot gasi in stocul magazinului in cantitati suficiente sau insuficiente . Comenzile se definesc printr-un anumit client care le plaseaza , denumirea produsului , cantitea care doreste sa fie cumparata si localitatea unde va fi livrata comanda . Clientii se definesc prin nume , numar de telefon si localitate .

Dupa efectuarea unei comenzi si realizarea acesteiea , produsele cumparate vor fi scazute din stocul magazinului . De asemenea , avem si posibilitatea de a improspata stocul. Toate aceste informatii vor fi stocate intr-o baza de date , care va fi reimprospatata in timp real la fiecare operatie asupra tabelelor si anume , inserare , stergere si update .

3.Diagrama UML + diagrama USE CASE

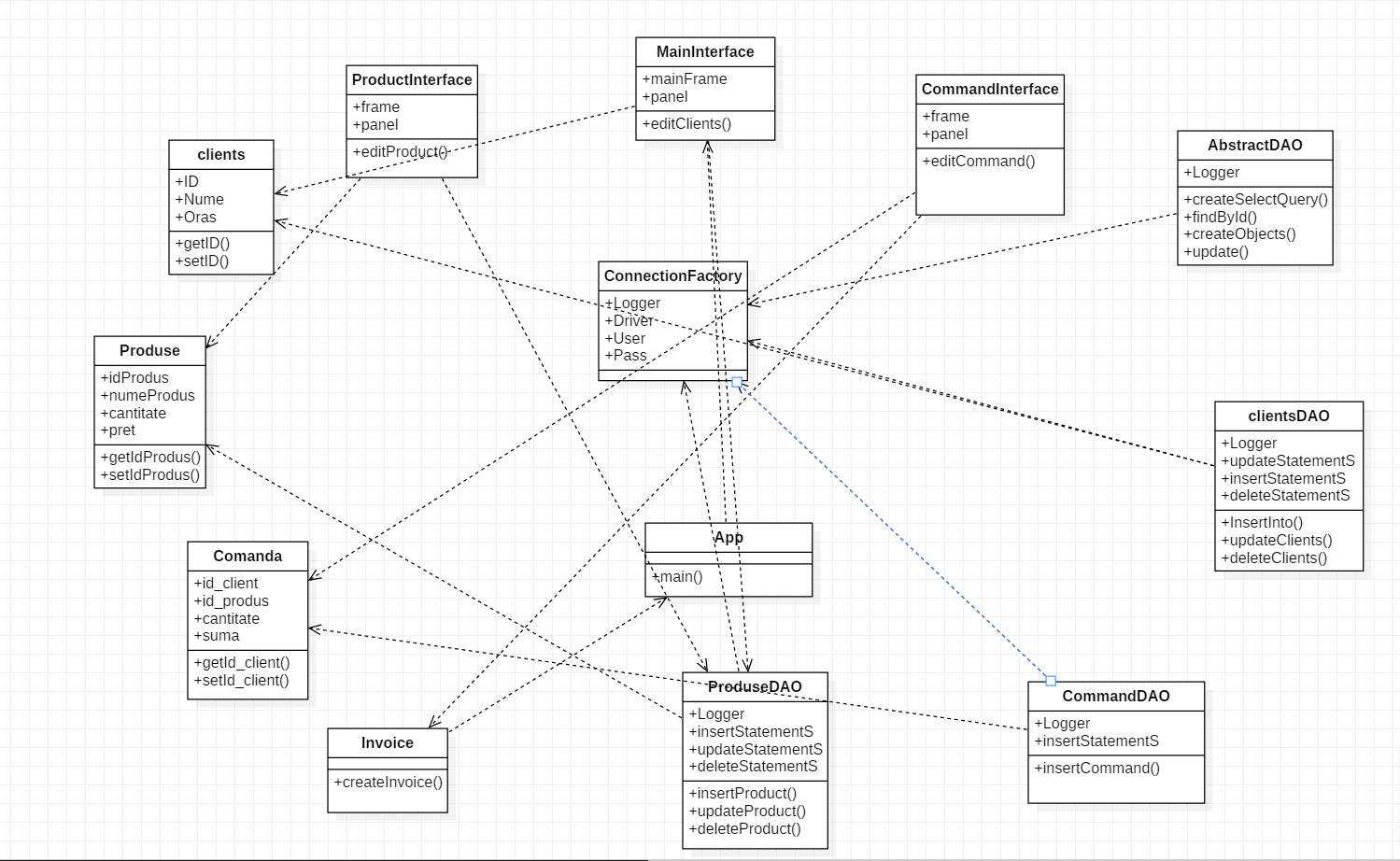
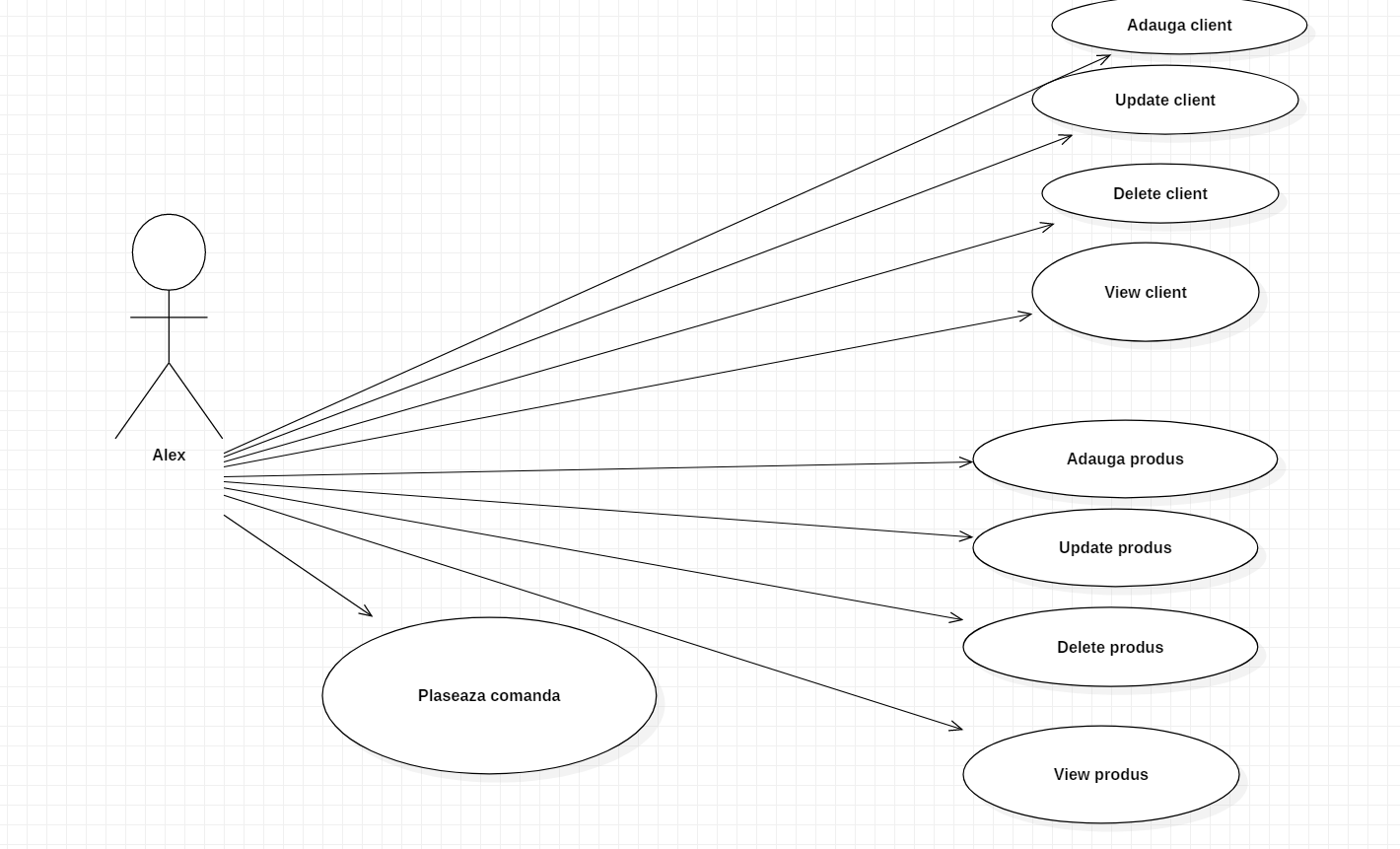


Diagrama USE CASE



**4. Notiuni privind bazele de date**

**Ce este o baza de date ?**

O bază de date, uneori numită și bancă de date (abreviat BD), reprezintă o modalitate de stocare a unor [informații](https://ro.wikipedia.org/wiki/Informa%C8%9Bie) și [date](https://ro.wikipedia.org/wiki/Dat%C4%83) pe un suport extern (un dispozitiv de stocare), cu posibilitatea extinderii ușoare și a regăsirii rapide a acestora. La prima vedere sarcina poate părea banală. Totuși, în condițiile în care este vorba de a lucra cu milioane de elemente, fiecare putând consta din cantități de date care trebuie accesate simultan prin [Internet](https://ro.wikipedia.org/wiki/Internet) de către mii de utilizatori răspândiți pe întreg globul; și în condițiile când disponibilitatea aplicației și datelor trebuie să fie permanentă (de ex. pentru a nu pierde ocazia de a încheia afaceri), soluțiile bune nu sunt de loc simple.

De obicei o bază de date este memorată într-unul sau mai multe [fișiere](https://ro.wikipedia.org/wiki/Fi%C8%99ier). Bazele de date sunt manipulate cu ajutorul [sistemelor de gestiune a bazelor de date](https://ro.wikipedia.org/wiki/Sistem_de_gestiune_a_bazelor_de_date).

Cel mai răspândit tip de baze de date este cel relațional, în care datele sunt memorate în [tabele](https://ro.wikipedia.org/wiki/Tabel). Pe lânga tabele, o bază de date relațională mai poate conține: indecși, proceduri stocate, declanșatori, utilizatori și grupuri de utilizatori, tipuri de date, mecanisme de securitate și de gestiune a tranzacțiilor etc.

Alte tipuri de baze de date sunt modelul ierarhic, modelul orientat pe obiecte și, mai nou, modelul [XML](https://ro.wikipedia.org/wiki/XML).

Despre MySql:

MySQL este un [sistem de gestiune a bazelor de date](https://ro.wikipedia.org/wiki/Sistem_de_gestiune_a_bazelor_de_date) [relaționale](https://ro.wikipedia.org/w/index.php?title=Baze_de_date_rela%C8%9Bionale&action=edit&redlink=1), produs de compania suedeza [MySQL AB](https://ro.wikipedia.org/wiki/MySQL_AB) și distribuit sub [Licența Publică Generală GNU](https://ro.wikipedia.org/wiki/Licen%C8%9Ba_Public%C4%83_General%C4%83_GNU). Este cel mai popular [SGBD](https://ro.wikipedia.org/wiki/SGBD) [open-source](https://ro.wikipedia.org/wiki/Open-source) la ora actuală, fiind o componentă cheie a stivei [LAMP](https://ro.wikipedia.org/wiki/LAMP) ([Linux](https://ro.wikipedia.org/wiki/Linux), [Apache](https://ro.wikipedia.org/wiki/Apache), MySQL, [PHP](https://ro.wikipedia.org/wiki/PHP)).

**5.Analiza problemei**

Primul pas in dezvoltarea aplicatiei a fost sa instalez MySql server. Cu ajutorul MySql server am creat 3 baze de date pentru fiecare clasa principala din Java. De precizat faptul ca numele tabelelor din baza de date trebuie sa fie acelasi cu numele claselor care reprezinta tabele, astfel ca am creat urmatoarele 3 tabele: clients, Produse, Comanda. Modul in care am creat tabelele va fi descris la capitolul proiectare.

**6.Proiectare**

Inainte de a crea tabelele cu care vom lucre in MySql a trebuit sa cream o baza de date. Am procedat asa cum ni se spunea in indrumatorul de laborator astfel ca am creat o baza de date numita schooldb. Aceasta baza de date este populate cu 3 tabele ele fiind clients, Produse, Comanda.

Pentru crearea tabelei clients am folosit urmatoarea sintaxa:

CREATE TABLE clients(

ID INT(11) NOT NULL,

Nume VARCHAR(45),

Oras(VARCHAR(45),

PRIMARY KEY(ID));

Pentru crearea tabelei Produse am folosit urmatoarea sintaxa:

CREATE TABLE Produse(

idProdus INT(11),

numeProdus(VARCHAR45),

pret INT(11),

cantitate INT(11),

PRIMARY KEY(idProdus));

Pentru crearea tabelei Comanda am folosit urmatoarea sintaxa:

CREATE TABLE Comanda(

Id\_client INT(11),

Id\_produs INT(11),

Cantitate INT(11),

Suma INT(11));

Pentru a implementa aplicatia in Java, am folosit 5 pachete:

1. Pachetul connection
   1. Contine clasa ConnectionFactory cu ajutorul careia ne conectam la baza de date
2. Pachetul DAO
   1. Contine clasele AbstractDAO, clientsDAO, commandDAO, ProduseDAO, clase in interiorul carora realizam interogarile Java pentru baza de date
3. Pachetul model
   1. Contine clasele clients, Produse, Comanda, clase care descriu obiectele noastre si pe langa acestea mai avem si clasa Invoice cu ajutorul careia reusim sa realizam un fisier .txt echivalentul unei facturi pentru fiecare client care face o comanda
4. Pachetul Presentation
   1. Acest pachet contine 3 clase, CommandInterface, MainInterface si ProductInterface, clase care realizaza interfata grafica si functionalitatea ei
5. Pachetul Start
   1. Pachetul contine doar o clasa, App, care este de fapt main-ul aplicatiei

**7.Implementare**

Pentru a fi mai usor de implementat, de scris codul si de facut debugging am impartit aplicatia in 5 pachete dupa cum am explicat mai sus.

Printre cele mai importante clase se afla clasa ConnectionFactory cu ajutorul careia reusim sa ne conectam la baza noastra de date.

private static final Logger ***LOGGER*** = Logger.*getLogger*(ConnectionFactory.class.getName());

private static final String ***DRIVER*** = "com.mysql.cj.jdbc.Driver";

private static final String ***DBURL*** = "jdbc:mysql://localhost:3306/schooldb?useTimezone=true&serverTimezone=UTC";

private static final String ***USER*** = "root";

private static final String ***PASS*** = "root123";

Driver-ul reprezinta numele driverului unde trebuie sa ne conectam. DBURL reprezinta locatia bazei noastre de date, iar USER si PASS reprezinta id-ul si parola cu care ne putem conecta la baza de date unde putem face modificari, update-uri, creare de tabele, etc.

Clasa ConnectionFactory mai contine metode pentru realizarea unei conexiuni, obtinerea unei conexiuni si inchiderea acesteia.

Pentru a reusi sa extragem elementele din baza de date trebuie ca numele tabelei din baza noastra de date sa fie identic cu numele clasei. Astfel am creat cele 3 clase din pachetul model.

Clasa clients:

public class clients {

private int ID;

private String Nume;

private String Oras;

public clients(int iD, String nume, String oras) {

super();

ID = iD;

Nume = nume;

Oras = oras;

}

Clasa Produse:

public class Produse {

private int idProdus;

private String numeProdus;

private int cantitate;

private int pret;

public Produse(int idProdus, String numeProdus, int cantitate, int pret) {

super();

this.idProdus = idProdus;

this.numeProdus = numeProdus;

this.cantitate = cantitate;

this.pret = pret;

Clasa Comanda:

public class Comanda {

private int id\_client;

private int id\_produs;

private int cantitate;

private int suma;

public Comanda(int id\_client, int id\_produs, int cantitate, int suma) {

super();

this.id\_client = id\_client;

this.id\_produs = id\_produs;

this.cantitate = cantitate;

this.suma = suma;

}

Clasa ProduseDAO:

public class ProduseDAO {

protected static final Logger ***LOGGER*** = Logger.*getLogger*(clientsDAO.class.getName());

private static final String ***insertStatementS*** = "INSERT INTO Produse (idProdus, numeProdus, cantitate, pret)"

+ " VALUES (?,?,?,?)";

private static final String ***updateStatementS*** = "UPDATE Produse SET numeProdus = ?, cantitate = ?, pret = ? WHERE idProdus = ?";

private static final String ***deleteStatementS*** = "DELETE FROM Produse WHERE idProdus = ?";

public void insertProduct(Produse produs) {

Connection dbConnect = ConnectionFactory.*getConnection*();

PreparedStatement insertStatement = null;

int x = 0;

try {

insertStatement = dbConnect.prepareStatement(***insertStatementS***);

insertStatement.setInt(1, produs.getIdProdus());

insertStatement.setString(2, produs.getNumeProdus());

insertStatement.setInt(3, produs.getCantitate());

insertStatement.setInt(4, produs.getPret());

insertStatement.executeUpdate();

} catch (SQLException e) {

***LOGGER***.log(Level.***WARNING***, "ERROR INSERT PRODUCTS" + e.getMessage());

x = 1;

} finally {

ConnectionFactory.*close*(insertStatement);

ConnectionFactory.*close*(dbConnect);

if (x == 0) {

JOptionPane.*showMessageDialog*(null, "SUCCES");

} else

JOptionPane.*showMessageDialog*(null, "FAIL");

}

}

Imediat dupa numele clasei sunt definite string-urile cu ajutorul carora vom reusi sa facem interogarile necesare. Stringul insertStatementS va fi folosit pentru inserarea unor noi produse in tabela deja existenta Produse. Stringul updateStatementS este folosit pentru update-ul unor produse din table iar stringul deleteStatementS va fi folosit pentru stergerea anumitor produse din tabela, acelea care nu vor mai exista in stoc.

Dupa declararea acestor stringuri cu ajutorul carora vom realiza interogarile respectiv inserturi, updateuri si deleturi este implementata metoda de insert produs in tabela.

Cu ajutorul variabilei dbConnect de tipul Connection se va realiza conexiunea la baza de date. Pe urma se declara un insertStatement care este initializat la NULL. Cu ajutorul unui bloc de tipul try -> catch se incearca realizarea insertului in tabela. Pe prima linie dupa try se initializeaza interogarea. Pe urmatoarele 4 linii se seteaza valorile variabilelor idProdus, numeProdus, cantitate, pret, valori care vor fi introduse mai tarziu prin intermediul interfetei grafice si mai apoi preluate de metoda insert. Dupa ce aceste valori au fost preluate se executa update-ul.

Cu ajutorul clauzei catch putem vedea daca a aparut vreo eroare in momentul in care am dorit sa facem interogarea. Clauza finally este clauza care se va executa indifferent daca interogarea este facuta sau nu cu success iar in finally inchidem conexiunea la baza de date. In cazul in care interogarea a avut success vom afisa pe ecran un buton care ne va indica SUCCES, iar in caz contrar FAIL.

Clasa MainInterface:

public class MainInterface extends JFrame {

public static JFrame *mainFrame*;

private JPanel panel;

public void createMainInterface() {

panel = new JPanel();

*mainFrame* = new JFrame("Magazin");

panel.setLayout(null);

*mainFrame*.setSize(500, 400);

*mainFrame*.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

JButton buton = new JButton("Optiuni clienti");

buton.setFont(new Font("Courier", Font.***BOLD***, 12));

buton.setBounds(50, 50, 150, 50);

JButton buton2 = new JButton("Optiuni produse");

buton2.setFont(new Font("Courier", Font.***BOLD***, 12));

buton2.setBounds(280, 50, 150, 50);

JButton buton3 = new JButton("Comandati");

buton3.setFont(new Font("Courier", Font.***BOLD***, 12));

buton3.setBounds(160, 200, 150, 50);

panel.add(buton);

panel.add(buton2);

panel.add(buton3);

*mainFrame*.add(panel);

*mainFrame*.setVisible(true);

buton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

*mainFrame*.setVisible(false);

*editClients*();

}

});

buton2.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

ProductInterface prod = new ProductInterface();

prod.editProduct();

}

});

buton3.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

CommandInterface comanda = new CommandInterface();

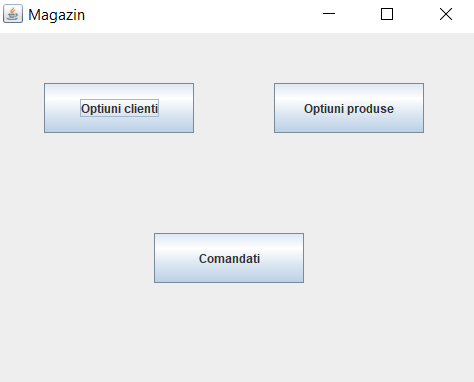
comanda.editCommand();

}

});

}

Codul de mai sus este doar o mica parte din clasa MainInterface. In MainInterface ne realizam ecranul principal al aplicatiei, mai exact ecranul de pornire.



Cel de mai sus este ecranul principal de pornire. In momentul in care se apasa unul dintre cele trei butoane se vor deschide alte panel-uri cu alte butoane si alte functionalitati. Fiecare buton are in spatele sau un ascultator care in momentul in care este apasat apeleaza una dintre functiile sale. Spre exemplu, in momentul in care butonul Optiuni produse este apasat se va apela metoda editProduct din clasa ProductInterface care la randul sau creeaza o noua alta interfata.

public void editProduct() {

final JFrame clientFrame = new JFrame("Produse");

JPanel panelClient = new JPanel();

clientFrame.setSize(500, 400);

panelClient.setLayout(null);

clientFrame.add(panelClient);

clientFrame.setDefaultCloseOperation(***EXIT\_ON\_CLOSE***);

clientFrame.setVisible(true);

*frame*.setVisible(false);

JButton backButton = new JButton("BACK");

panelClient.add(backButton);

backButton.setBounds(375, 275, 70, 50);

backButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

clientFrame.setVisible(false);

//frame.setVisible(true);

}

});

JButton viewProducts = new JButton("Vizualizare produse");

panelClient.add(viewProducts);

viewProducts.setBounds(50, 50, 150, 50);

JButton addProduct = new JButton("Adauga produs");

addProduct.setBounds(280, 50, 150, 50);

panelClient.add(addProduct);

JButton updateProduct = new JButton("Update produs");

panelClient.add(updateProduct);

updateProduct.setBounds(50, 150, 150, 50);

JButton deleteProduct = new JButton("Delete produs");

panelClient.add(deleteProduct);

deleteProduct.setBounds(280, 150, 150, 50);

Metoda editProduct creeaza un alt frame cu 4 butoane , Vizualizare produse, Adauga produs, Update produs si delete produs.

Fiecare interfata are implementat cate un buton suplimentar si anume butonul de back care ne ajuta sa mergem inapoi.

JButton backButton = new JButton("BACK");

vPanel.add(backButton);

backButton.setBounds(375, 275, 70, 50);

backButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

vClients.setVisible(false);

clientFrame.setVisible(true);

}

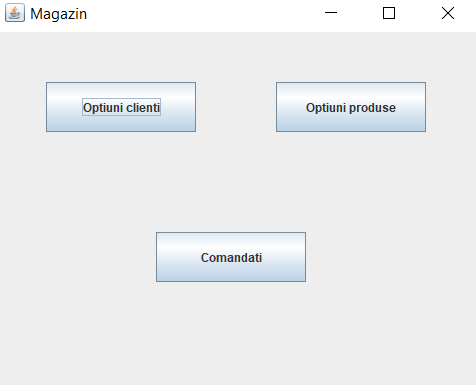
});

}

});

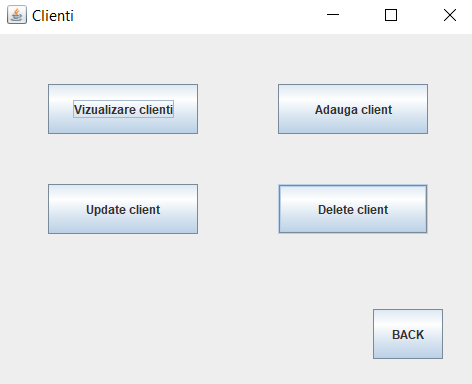
In momentul in care este apasat, butonul de BACK ne ascunde un frame, adica nu ni-l lasa vizibil si ne face alt frame vizibil.

**8.Rezultate:**

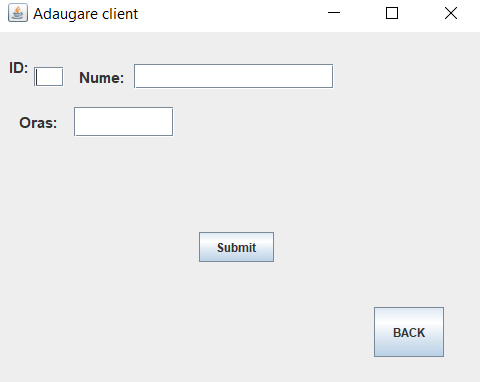


MAIN

In momentul in care se apasa butonul Optiuni client se va afisa un nou frame:

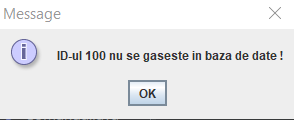


In momentul in care se doreste a se adauga un nou client, se apasa butonul Adauga client:



Dupa ce s-au introdus datele necesare se apasa butonul Submit. In cazul in care totul decurge ok se va afisa un mesaj de SUCCESS, altfel un mesaj de FAIL.

Am incercat sa acopar cat mai multe dintre posibilele bug-uri. Spre exemplu, in momentul in care se doreste stergerea sau update-ul unui client al carui ID nu exista se va afisa un mesaj de eroare de felul acesta :



**9.Concluzii**

Cu fiecare tema realizata reusesc sa invat mai bine sa lucrez in Eclipse in ceea ce priveste interfata grafica: crearea de componente, setarea lor in spatiu, setarea textului acestora, setarea dimensiunilor acestora, implementarea ActionListener-ului pe butoane etc.

Ca si dezvoltari ulterioare as lua in considerare urmatoarele posibilitati:

a) Reprezentarea aplicatiei din 2 perspective: a adminului si a clientului

b) Adaugarea de imagini corespunzatoare fiecarui produs

**10.Bibliografie**

<https://www.daniweb.com/programming/software-development/threads/283156/display-database-values-into-jtables>

<https://itextpdf.com/en/resources/books/zugferd-future-invoicing/5-creating-pdf-invoices-basic-profile>

* <http://www.coned.utcluj.ro/~salomie/PT_Lic/3_Lab/HW3_Tema3/>
* <https://ro.wikipedia.org/wiki/MySQL>
* <https://ro.wikipedia.org/wiki/Baz%C4%83_de_date>
* <http://stackoverflow.com/>