DOCUMENTATIE

TEMA *3*

NUME STUDENT:
Buzila Maria-Alexandra
GRUPA:
30226

CUPRINS

1. Obiectivul temei

Obiectivul principal se referă la proiectarea și implementarea unei aplicații de management a comenzilor pentru un depozit. Obiectivele secundare ce ajuta la realizarea celui principal sunt prezentate succint in lista de mai jos.

- Analizarea problemei și identificarea cerințelor (obiectiv descris in capitolul 2)
  - Analiza poate fi definita prin impartirea problemei in etape mai mici
  - Se determina cerintele functionale si cele non-functionale
- Proiectarea aplicatiei de gestionare a comenzilor (obiectiv descris in capitolul 3)
  - Utilizarea resurselor pe care le avem la dispozitie pentru a realiza legaturile importante, care ne vor ajuta cel mai mult la realizarea propriu-zisa a acestuia.
  - Realizarea unui plan/sistem eficient, corect si cat mai usor de implementat: determinam clasele, legaturile dintre ele si organizarea lor in pachete
- Implementarea aplicatiei de gestionare a comenzilor (obiectiv descris in capitolul 4)
  - Aceasta etapa este reprezentata de scrierea de cod in limbaj Java a claselor corespunzatoare, impreuna cu metodele acestora.
- Testarea aplicatiei de gestionare a comenzilor (obiectiv descris in capitolul 5)
  - Testarea cea mai importanta etapa in realizarea unui astfel de proiect.
  - Realizarea unor operații prin intermediul interfeței și observarea modificării tabelelor din baza de date

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

In continuare, voi explica cateva din cerintele functionale (ce descriu ce trebuie sa faca programul) si non-functionale (ce descriu calitatile pe care ar trebui sa le aiba programul) ce trebuie bine intelese.

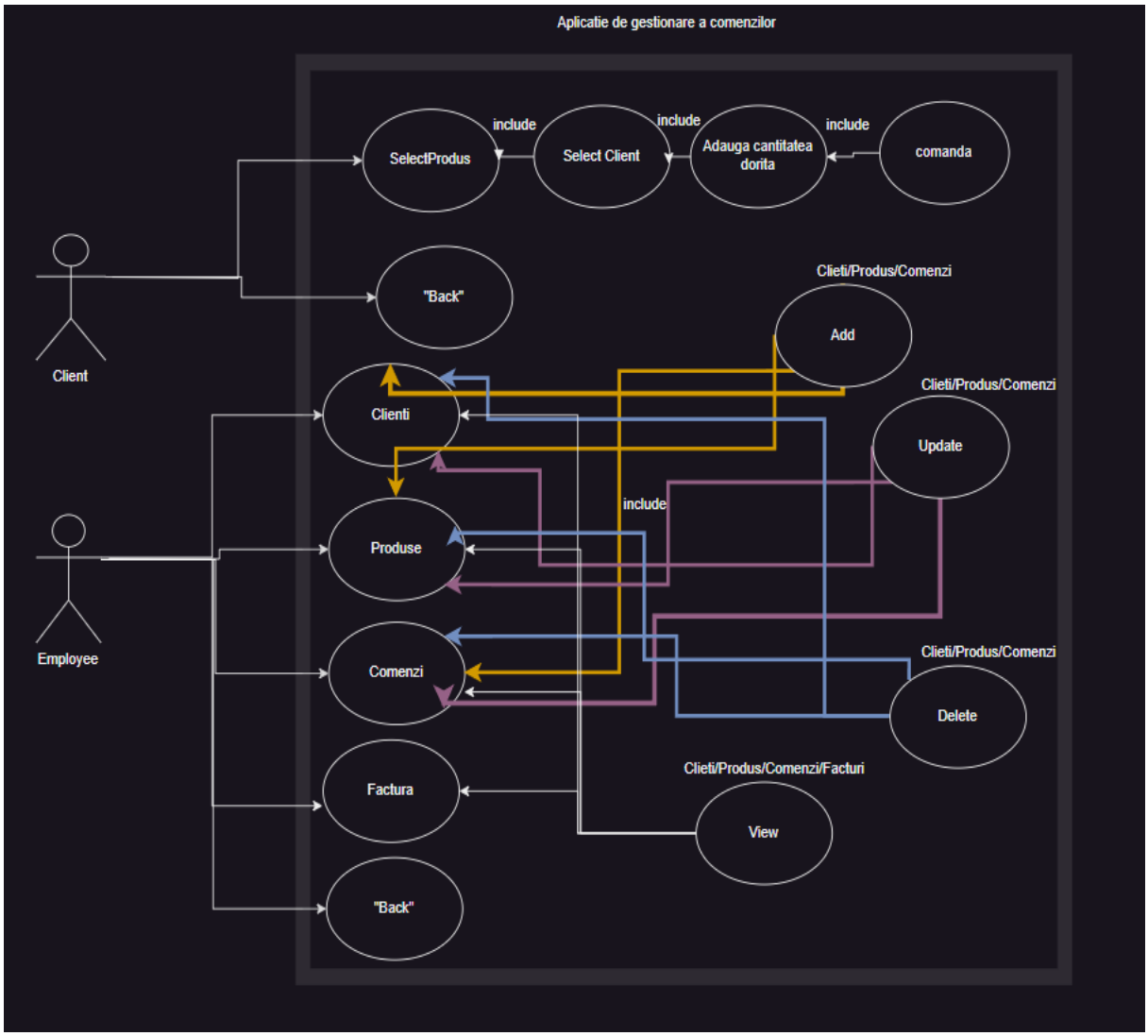
Cerinte functionale:

- Aplicația ar trebui sa-i permita angajatului să selecteze un/o produs/client/comanda și să îl/o șteargă din baza de date
- Aplicația ar trebui sa-i permita angajatului să selecteze un/o produs/client/comanda și să îl/o editeze in baza de date
- Aplicația ar trebui sa-i permita angajatului să adauge un/o produs/client/comanda in baza de date

- Aplicația ar trebui sa-i permita clientului să selecteze un produs pe care sa-l comande

Cerinte non-functionale:

- Aplicația ar trebui să fie intuitivă și ușor de utilizat utilizare de către utilizator
- Aplicatia ar trebui sa informeze clientul sau angajatul de fiecare data cand introduce date invalide.



Cazurile de utilizare ale aplicației:

Use case: afisare clienti/produse/comenzi/facturi
Actor principal: manager/angajat
Principalul scenariu de succes:
1. Utilizatorul apasă din fereastra principală butonul Employee

2. Utilizatorul selectează butonul cu obiecte pe care doreste sa le vizualizeze (Clients/Products/Orders/Bills)
3. Utilizatorul apasă butonul view

Secvențe secundare

- Utilizatorul apasa butonul „Back to login”
- Se revine la pasul 1

Use case: inserare clienti/produse/comenzi
Actor principal: manager/angajat
Princpalul scenariu de succes:
1. Utilizatorul apasă din fereastra principală butonul Employee
2. Utilizatorul selectează butonul cu obiecte pe care doreste sa le vizualizeze (Clients/Products/Orders)
3. Utilizatorul apasă butonul add
4. Utilizatorul introduce id-ul, numele, adresa si email-ul pentru clienti/ id-ul, denumirea, pretul si cantitatea pentru produse/ id-ul, id-ul clientului, id-ul produsului si cantitatea pentru comenzi
5. Utilizatorul apasă pe butonul *Add*

Secvențe secundare

▪ Utilizatorul introduce id deja folosit.
▪ Utilizatorul introduce email invalid.
▪ Utilizatorul introduce cantitate sau pret ca fiind un numar negativ
▪ Utilizatorul nu introduce toate datele cerute
▪ Utilizatorul va fi informat in legatura cu greseala facuta, apoi poate introduce datele corecte
▪ Utilizatorul poate apasa butonul „Back”

Use case: editare clienti/produse/comenzi
Actor principal: manager/angajat
Princpalul scenariu de succes:
1. Utilizatorul apasă din fereastra principală butonul Employee
2. Utilizatorul selectează butonul cu obiecte pe care doreste sa le vizualizeze (Clients/Products/Orders)
3. Utilizatorul apasă butonul Update
4. Utilizatorul trebuie sa selecteze un rand din tabelul care ii este prezentat (selecteaza obiectul pe care doreste sa-l modifice)

4. Utilizatorul introduce numele, adresa si email-ul pentru clienti/ denumirea, pretul si cantitatea pentru produse/ id-ul clientului, id-ul produsului si cantitatea pentru comenzi
5. Utilizatorul apasă pe butonul *Update*

Secvențe secundare

▪ Utilizatorul introduce email invalid.
▪ Utilizatorul introduce cantitate sau pret ca fiind un numar negativ
▪ Utilizatorul nu introduce toate datele cerute
▪ Utilizatorul nu selecteaza un obiect
▪ Utilizatorul va fi informat in legatura cu greseala facuta, apoi poate introduce datele corecte
▪ Utilizatorul poate apasa butonul „Back”

Use case: stergere clienti/produse/comenzi
Actor principal: manager/angajat
Princpalul scenariu de succes:
1. Utilizatorul apasă din fereastra principală butonul Employee
2. Utilizatorul selectează butonul cu obiecte pe care doreste sa le vizualizeze (Clients/Products/Orders)
3. Utilizatorul apasă butonul Delete
4. Utilizatorul trebuie sa selecteze un rand din tabelul care ii este prezentat (selecteaza obiectul pe care doreste sa-l stearga)
5. Utilizatorul apasă pe butonul *Delete*

Secvențe secundare
▪ Utilizatorul nu selecteaza un obiect
▪ Utilizatorul va fi informat in legatura cu greseala facuta
▪ Utilizatorul poate apasa butonul „Back”

Use case: vizualizare clienti/produse/comenzi/facturi
Actor principal: manager/angajat
Princpalul scenariu de succes:
1. Utilizatorul apasă din fereastra principală butonul Employee
2. Utilizatorul selectează butonul cu obiecte pe care doreste sa le vizualizeze (Clients/Products/Orders/Bills)
3. Utilizatorul apasă butonul View

Secvențe secundare
▪ Utilizatorul poate apasa butonul „Back”

Use case: realizare comanda
Actor principal: client
Princpalul scenariu de succes:
1. Utilizatorul apasă din fereastra principală butonul Client
2. Utilizatorul selectează un rand din tabelul cu produse ce ii este prezentat

3. Utilizatorul apasă butonul Select Client
4. Utilizatorul trebuie sa selecteze un rand din tabelul ce contine clienti (se alege pe el)
4. Utilizatorul introduce cantitatea pe care doreste sa o comande
5. Utilizatorul apasă pe butonul *Order*

Secvențe secundare

▪ Utilizatorul nu selecteaza un produs sau un client
▪ Utilizatorul introduce cantitate numar negativ sau un numar mai mare decat cantitatea ce se afla pe stoc
▪ Utilizatorul nu introduce toate datele cerute
▪ Utilizatorul va fi informat in legatura cu greseala facuta, apoi poate introduce datele corecte
▪ Utilizatorul poate apasa butonul „Back”

3. Proiectare

Sistemul de management este divizat în 5 pachete.

Presentation - conține clasele care definesc interfața grafică cu controllerul asociat, pentru a gestiona semnalele date de client prin butoane si datele introduse de acesta.

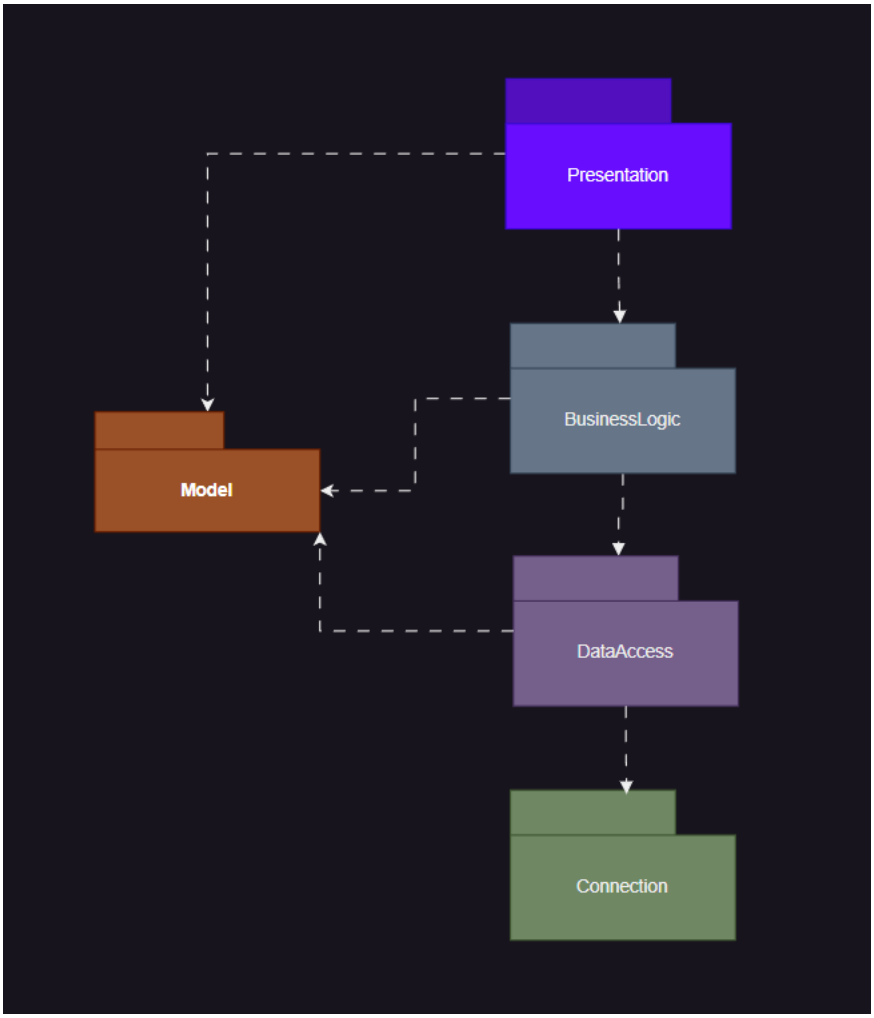
BusinessLogic – conține clasele care încapsulează logica aplicației. Prin intermediul lui se cer claselor ce manipulează direct baza de date să facă anumite modificări în raport cu modelul. Această clasă conține și un „subpachet” numit *validators* ce conține validatorii pentru obiectele aflate in baza de date. (EmailValidator, PriceValidator, QuantityValidator)

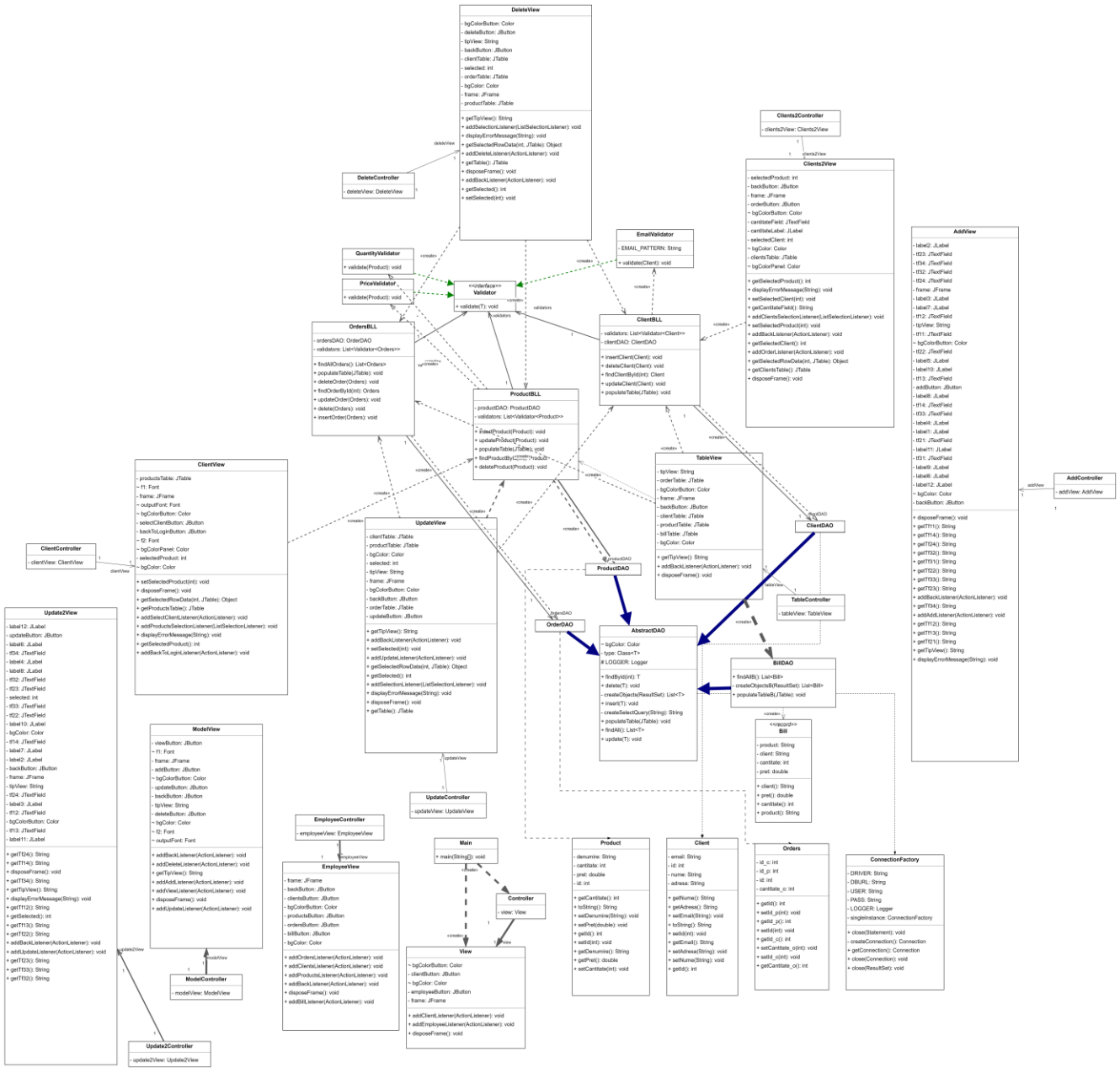
Model- conține clasele modelelor de date (client, comanda, produse, factura)

DataAccess - conține clasele ce contin interogările, cererea conexiunii la baza de date și prin intermediul cărora se fac actualizările bazei de date. Principalele interogări sunt cele de tipul CRUD.

Connection-conține clasa prin intermediul căreia se poate face conexiunea la baza de date cu serverul local și cu parola

Diagrama de pachete:





Există o metodă private în clasa generică AbstractDao (createObjects(ResultSet resultSet)), în intermediul căreia se utilizează tehnica de reflection pentru extragerea rezultatelor din cursorul interogării executate și returnează o listă de obiecte din model care au

câmpurile identice cu coloanele din tabele din baza de date. O interfata implementata este Validator, care are o metoda validate, prin care clasele ce o implementeaza trebuie sa verifice constrangerile datelor de intrare.

4. Implementare

1. Pachet ***Model***:

   *Clasa **Client*** -este clasa care descrie modelul real de client. Acesta clasa va avea câmpurile identice (și ca de numire) cu coloanele tabelului asociat din baza de date. Aceasta va descrie caracteristicile necesare pentru definirea unui client printr-un id propriu, nume, adresa, email.

   *Clasa **Product*** – asemanatoare clasei Client. Aceasta va descrie caracteristicile necesare pentru definirea unui produs printr-un id propriu, denumire, cantitate, pret.

   *Clasa **Order*** – asemanatoare clasei Client. Are ca si atribute id propriu, id client, id produs, cantitate.

   Clasa ***Bill*** - *Clasa Bill reprezintă o factură care include numele clientului, numele produsului, cantitatea și prețul total.*
   Aceasta este o înregistrare (record) care simplifică definirea unei clase de date imutabile.

```
public record Bill(String client, String product, int cantitate, double pret) {}  14 usages
```

2. Pachet ***DataAccess***

   Clasa ***AbstractDAO<T>*** - reprezintă o clasă „abstractă” generică pentru operațiile CRUD (Create, Read, Update, Delete)
   pe obiecte de tip T. Această clasă folosește reflexia pentru a interoga baza de date și a crea obiecte de tipul specificat. De asemenea, oferă metode pentru inserarea, actualizarea, ștergerea și găsirea obiectelor din baza de date, precum și pentru popularea unui tabel Swing JTable cu datele obținute din baza de date.

```java
public void update(T t) {
    String updateStatement = "UPDATE " + type.getSimpleName() + " SET ";
    Connection connection = null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    int id = -1;
    try {
        for (Field field : type.getDeclaredFields()) {
            field.setAccessible(true);
            Object obj = field.get(t);
            if (field.getName().equals("id")) {
                id = (Integer) obj;
            }
            if (obj instanceof String) {
                updateStatement += field.getName() + " = '" + obj + "',";
            } else {
                updateStatement += field.getName() + " = " + obj + ",";
            }
```

```java
        updateStatement = updateStatement.substring(0, updateStatement.length() - 1);
        updateStatement += " WHERE id = ?";
        connection = ConnectionFactory.getConnection();
        statement = connection.prepareStatement(updateStatement);
        statement.setInt( parameterIndex: 1, id);
        statement.execute();
    } catch (IllegalAccessException | SQLException e) {
        e.printStackTrace();
    } finally {
        ConnectionFactory.close(resultSet);
        ConnectionFactory.close(statement);
        ConnectionFactory.close(connection);
    }
}
```

Clasele **ClientDAO, OrdersDAO, ProductDAO** – extind AbstractDAO<T> si sunt responsabile pentru accesul la datele tabelului Client/Order/Produc din baza de date și manipularea acestora. Nu adaugă metode suplimentare față de cele moștenite din AbstractDAO, dar specifică tipul generic ca fiind Client/Product/Orders

Clasa **BillDAO -** este responsabilă pentru accesul la datele tabelului Bill din baza de date și manipularea acestora.
Extinde clasa AbstractDAO pentru a beneficia de operația insert definită în aceasta. Această clasă oferă metode pentru a găsi toate înregistrările din tabel și pentru a popula un tabel Swing Jtable cu datele obținute din baza de date**.**

```
public void populateTableB(JTable table) {  1 usage
    DefaultTableModel model = new DefaultTableModel();
    model.addColumn( columnName: "Client");
    model.addColumn( columnName: "Product");
    model.addColumn( columnName: "Cantitate");
    model.addColumn( columnName: "Pret");
    List<Bill> bills = findAllB();
    for (Bill bill : bills) {
        Object[] row = new Object[4];
        row[0] = bill.client();
        row[1] = bill.product();
        row[2] = bill.cantitate();
        row[3] = bill.pret();
        model.addRow(row);
    }
    table.setModel(model);
}
```

3. Pachet **Connection**

Clasa ***ConnectionFactory*** **-** responsabilă pentru gestionarea conexiunilor la baza de date. Aceasta folosește un model Singleton pentru a asigura o unică instanță de conexiune. Include metode pentru crearea și închiderea conexiunilor, a statement-urilor și a result set-urilor.

```java
private static final Logger LOGGER = Logger.getLogger(ConnectionFactory.class.getName());
private static final String DRIVER = "com.mysql.cj.jdbc.Driver";  1 usage
private static final String DBURL = "jdbc:mysql://localhost:3306/warehouse";  1 usage
private static final String USER = "root";  1 usage
private static final String PASS = "AlleDB2003";  1 usage

private static ConnectionFactory singleInstance = new ConnectionFactory();  1 usage

/**
 * Constructorul privat al clasei ConnectionFactory.
 * Încarcă driverul necesar pentru conexiunea la baza de date.
 */
private ConnectionFactory() {  1 usage
    try {
        Class.forName(DRIVER);
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
}
```

4. Pachetul **BusinessLogic**:

   Pachet **Validators**:

   - Interfata ***Validator***: Interfața Validator definește metoda de validare pentru obiectele de tip generic T.
     Această interfață este utilizată pentru a asigura că obiectele respectă anumite reguli de validare.

```java
public interface Validator<T> {  16 usages  3 implementations

    /**
     * Metoda de validare care trebuie implementată pentru a verifica dacă un obiect de
     *
     * @param t obiectul de validat
     */
    public void validate(T t);  6 usages  3 implementations
}
```

- Clasa ***QuantityValidator*** - implementează interfața Validator pentru obiecte de tip Product. Validează cantitatea unui produs asigurându-se că este pozitivă.
- Clasa ***PriceValidator*** - implementează interfața Validator pentru obiecte de tip Product. Validează prețul unui produs asigurându-se că este pozitiv.

```java
public class PriceValidator implements Validator<Product> {  1 usage

    /**
     * Metoda validate verifică dacă prețul unui produs este valid (mai mare sau egal cu zero).
     *
     * @param t obiectul Product de validat
     * @throws IllegalArgumentException dacă prețul produsului nu este valid
     */
    public void validate(Product t) {  6 usages
        if (t.getPret() ⩽ 0) {
            throw new IllegalArgumentException("The price limit is not respected!");
        }
    }
}
```

- Clasa ***EmailValidator*** - *implementează interfața Validator pentru obiecte de tip Client. Validează adresa de email a unui client utilizând un pattern regex.*

Clasele ***ClientBLL, ProductBLL, OrdersBLL*** - gestionează operațiunile de business logic pentru entitatatile din pachetul Model: Client, Product, Orders. Aceasta include validarea și interacțiunea cu baza de date prin intermediul claselor din pachetul DataAccess.

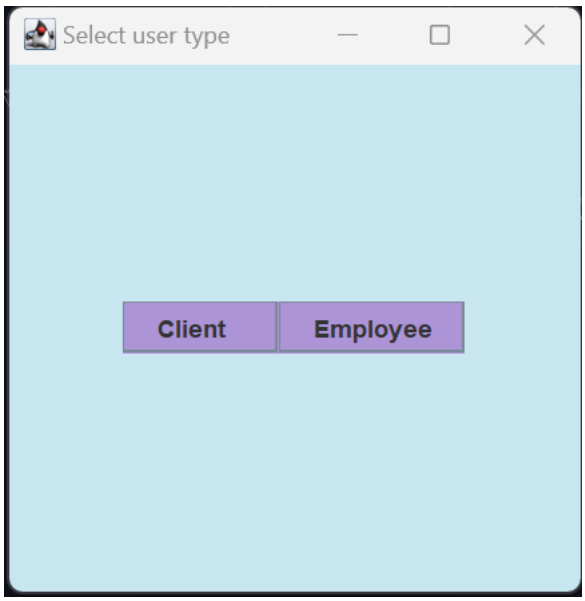
```
private OrderDAO ordersDAO;  8 usages
private List<Validator<Orders>> validators;  3 usages

/**
 * Constructorul inițializează lista de validatori și obiectul DAO pentru comenzi.
 */
public OrdersBLL() {  7 usages
    ordersDAO = new OrderDAO();
    validators = new ArrayList<Validator<Orders>>();
}
```

```
public void updateOrder(Orders order) {  1 usage
    for (Validator<Orders> v : validators) {
        v.validate(order);
    }
    ordersDAO.update(order);
}
```

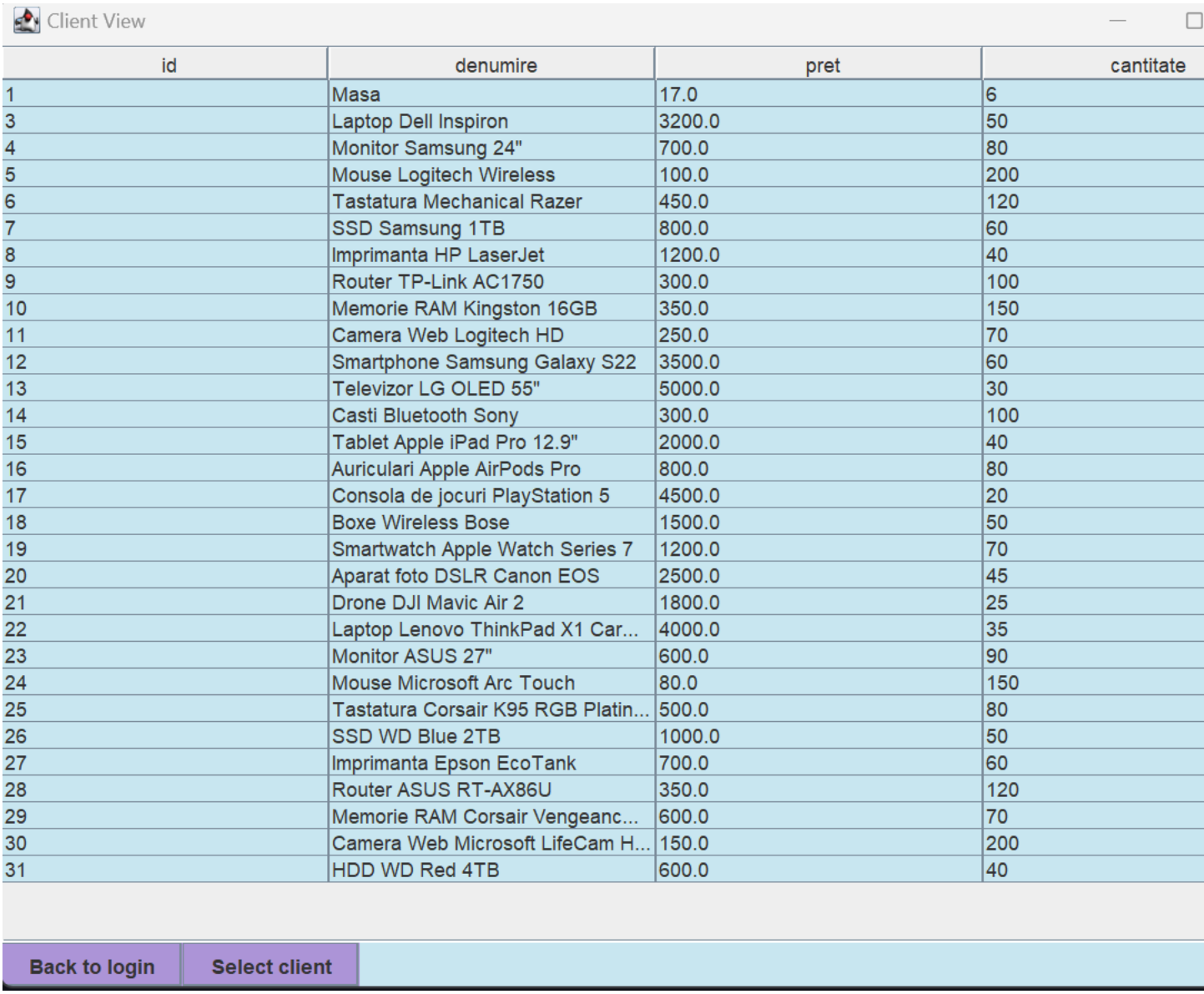
Pachetul ***Presentation***:

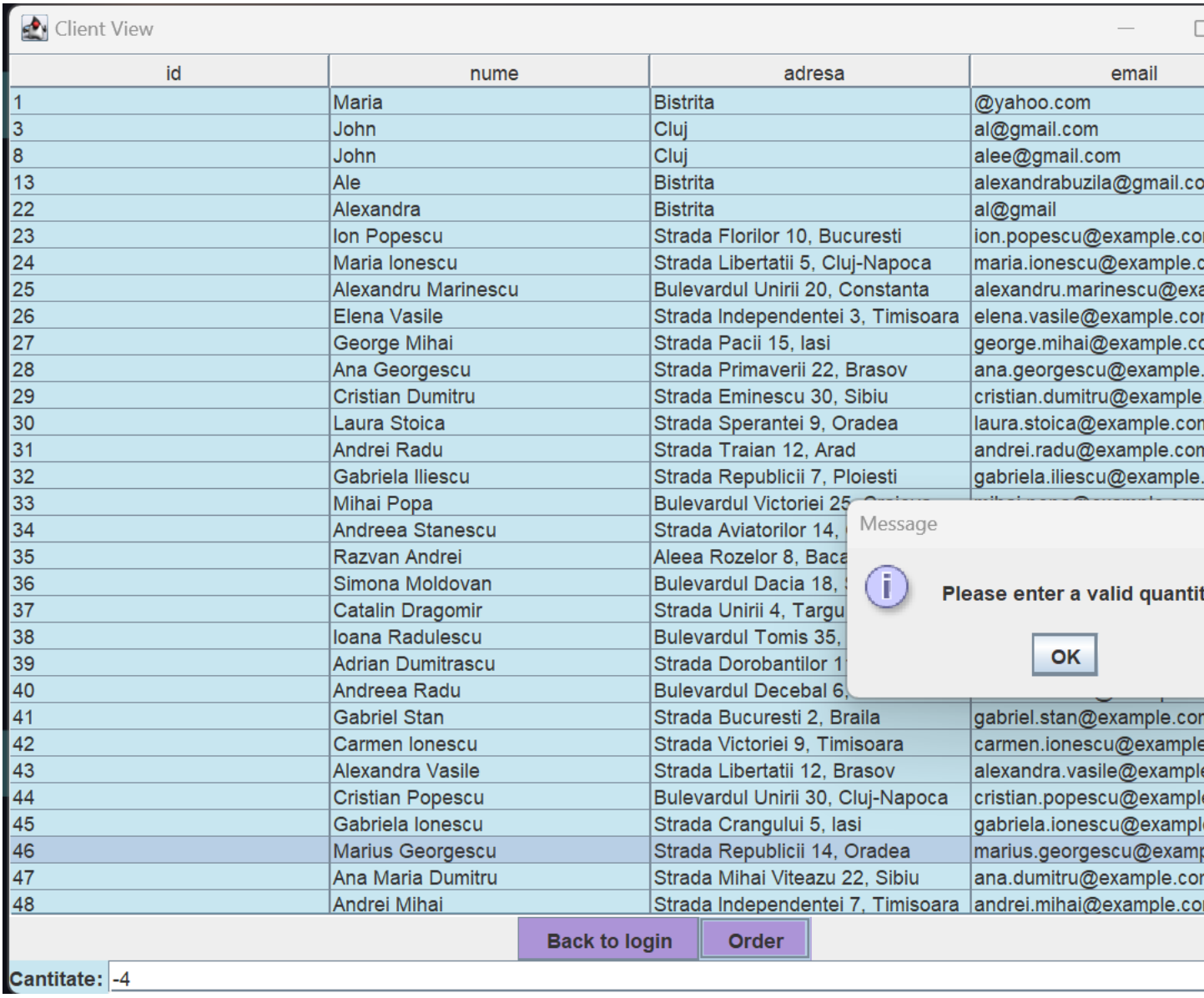
In acest pachet avem 10 clase view, fiecare avand un controller. In total 20 de clase, 10 Jframe-uri intre care putem naviga usor. Cele principale sunt clasele View si Controller, cele de la care incepe aplicatia si singurele clase instantiate in metoda main. In continuare vor fi prezentate aceste interfete utilizand imagini.



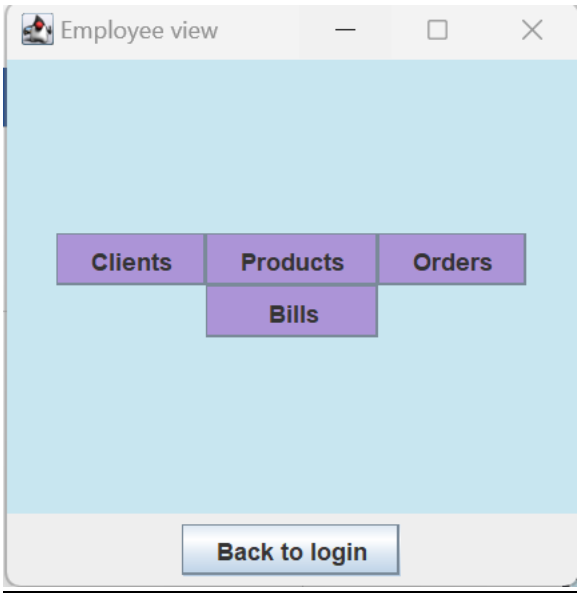
Prima parte a aplicatiei

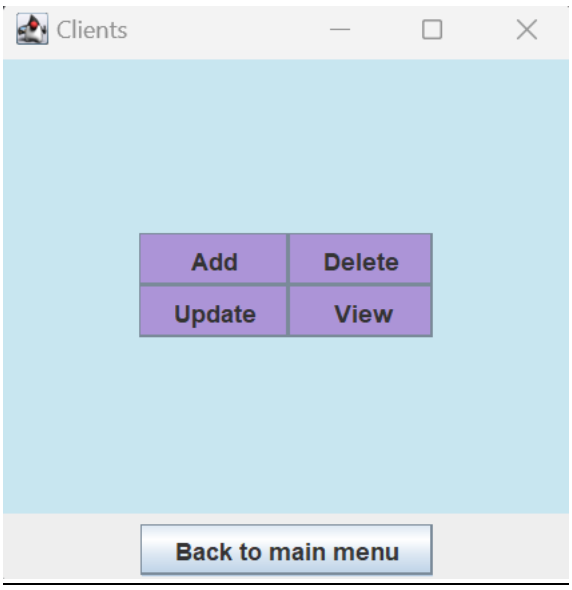
Butonul Client:

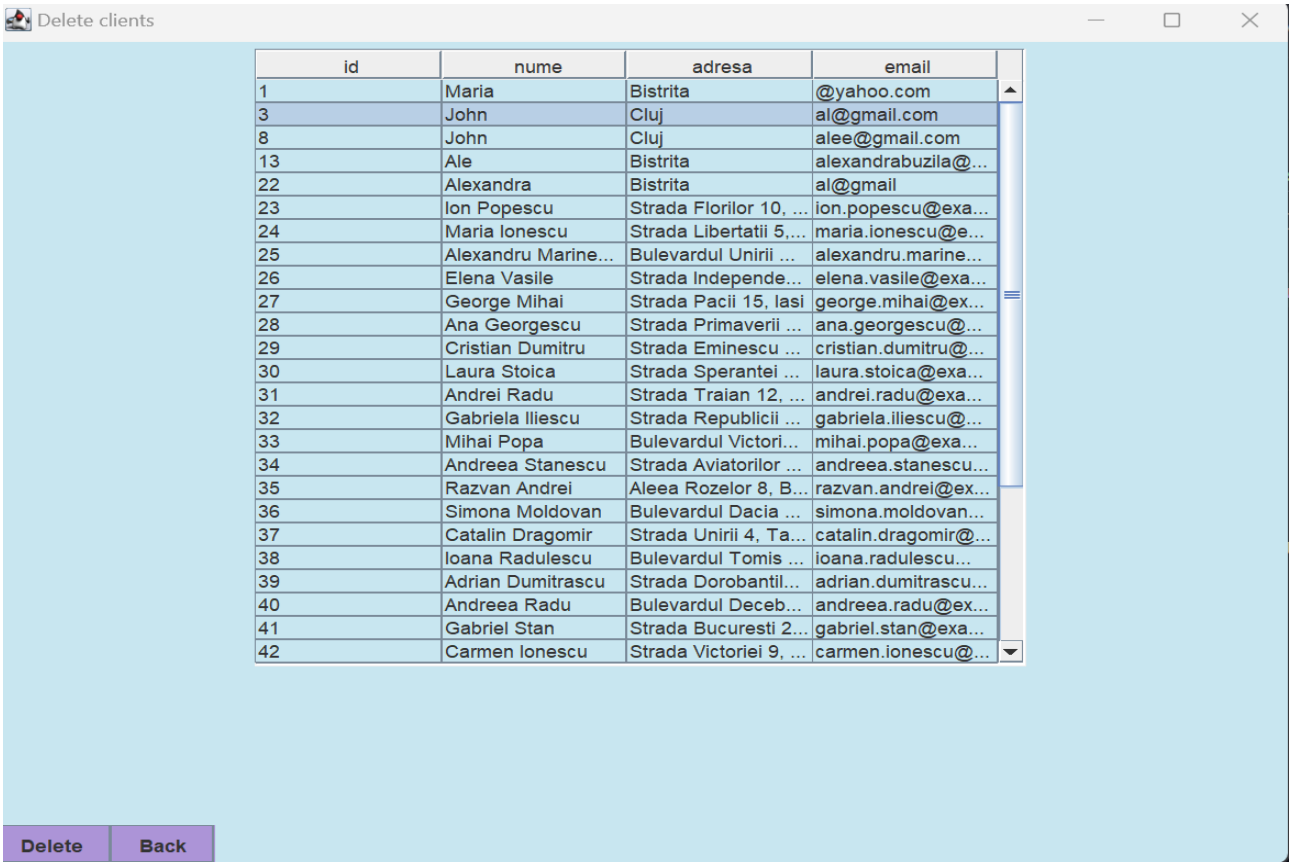

Client View
id
denumire
pret
cantitate
1 Masa 17.0 6
3 Laptop Dell Inspiron 3200.0 50
4 Monitor Samsung 24" 700.0 80
5 Mouse Logitech Wireless 100.0 200
6 Tastatura Mechanical Razer 450.0 120
7 SSD Samsung 1TB 800.0 60
8 Imprimanta HP LaserJet 1200.0 40
9 Router TP-Link AC1750 300.0 100
10 Memorie RAM Kingston 16GB 350.0 150
11 Camera Web Logitech HD 250.0 70
12 Smartphone Samsung Galaxy S22 3500.0 60
13 Televizor LG OLED 55" 5000.0 30
14 Casti Bluetooth Sony 300.0 100
15 Tablet Apple iPad Pro 12.9" 2000.0 40
16 Auriculari Apple AirPods Pro 800.0 80
17 Consola de jocuri PlayStation 5 4500.0 20
18 Boxe Wireless Bose 1500.0 50
19 Smartwatch Apple Watch Series 7 1200.0 70
20 Aparat foto DSLR Canon EOS 2500.0 45
21 Drone DJI Mavic Air 2 1800.0 25
22 Laptop Lenovo ThinkPad X1 Car... 4000.0 35
23 Monitor ASUS 27" 600.0 90
24 Mouse Microsoft Arc Touch 80.0 150
25 Tastatura Corsair K95 RGB Platin... 500.0 80
26 SSD WD Blue 2TB 1000.0 50
27 Imprimanta Epson EcoTank 700.0 60
28 Router ASUS RT-AX86U 350.0 120
29 Memorie RAM Corsair Vengeanc... 600.0 70
30 Camera Web Microsoft LifeCam H... 150.0 200
31 HDD WD Red 4TB 600.0 40
Back to login
Select client

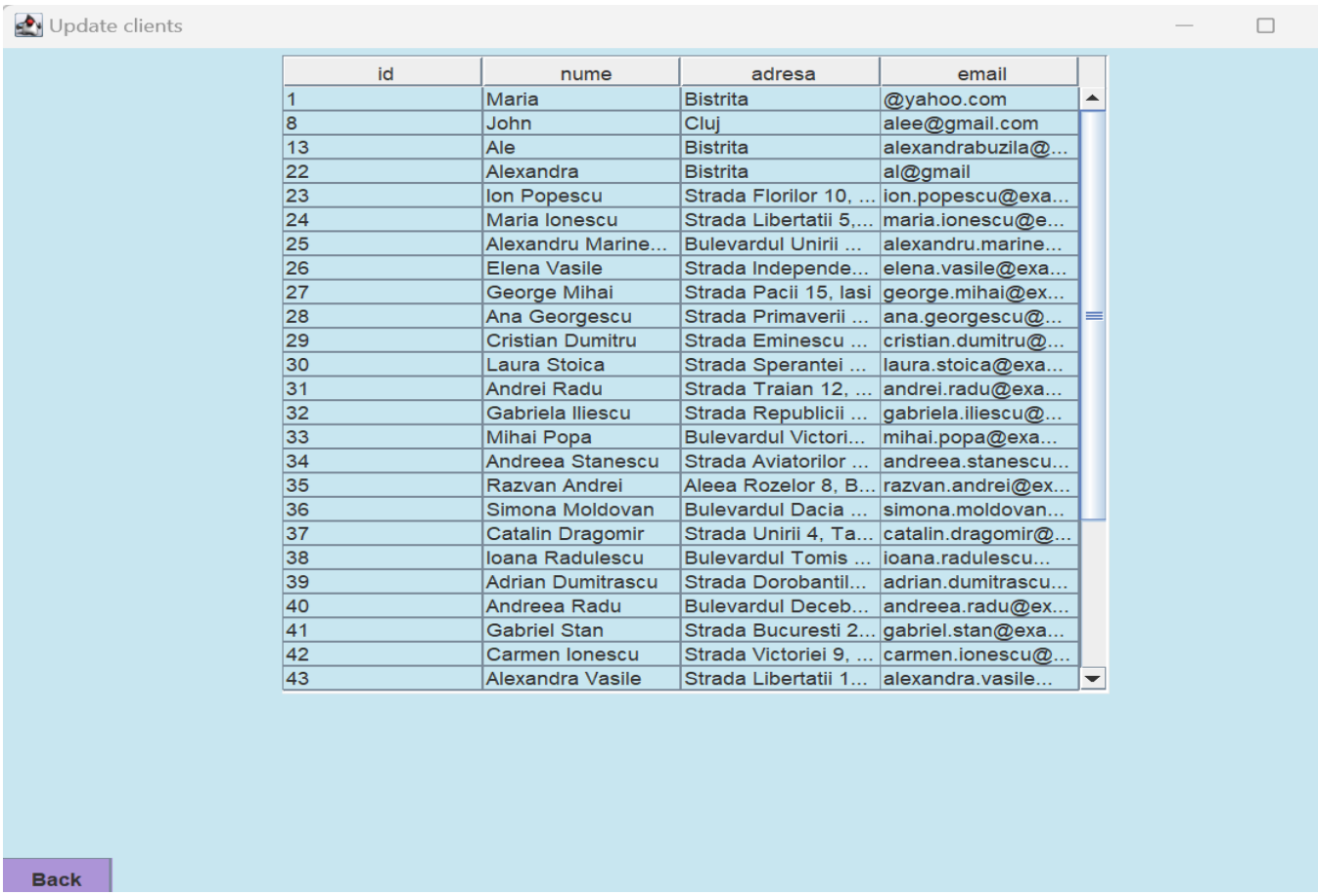


Buton Employee:


Employee view
Clients
Products
Orders
Bills
Back to login


Clients
Add
Delete
Update
View
Back to main menu


Delete clients
id
nume
adresa
email
1 Maria Bistrita @yahoo.com
3 John Cluj al@gmail.com
8 John Cluj alee@gmail.com
13 Ale Bistrita alexandrabuzila@...
22 Alexandra Bistrita al@gmail
23 Ion Popescu Strada Florilor 10, ... ion.popescu@exa...
24 Maria Ionescu Strada Libertatii 5,... maria.ionescu@e...
25 Alexandru Marine... Bulevardul Unirii ... alexandru.marine...
26 Elena Vasile Strada Independe... elena.vasile@exa...
27 George Mihai Strada Pacii 15, Iasi george.mihai@ex...
28 Ana Georgescu Strada Primaverii ... ana.georgescu@...
29 Cristian Dumitru Strada Eminescu ... cristian.dumitru@...
30 Laura Stoica Strada Sperantei ... laura.stoica@exa...
31 Andrei Radu Strada Traian 12, ... andrei.radu@exa...
32 Gabriela Iliescu Strada Republicii ... gabriela.iliescu@...
33 Mihai Popa Bulevardul Victori... mihai.popa@exa...
34 Andreea Stanescu Strada Aviatorilor ... andreea.stanescu...
35 Razvan Andrei Aleea Rozelor 8, B... razvan.andrei@ex...
36 Simona Moldovan Bulevardul Dacia ... simona.moldovan...
37 Catalin Dragomir Strada Unirii 4, Ta... catalin.dragomir@...
38 Ioana Radulescu Bulevardul Tomis ... ioana.radulescu...
39 Adrian Dumitrascu Strada Dorobantil... adrian.dumitrascu...
40 Andreea Radu Bulevardul Deceb... andreea.radu@ex...
41 Gabriel Stan Strada Bucuresti 2... gabriel.stan@exa...
42 Carmen Ionescu Strada Victoriei 9, ... carmen.ionescu@...
Delete
Back



5. Rezultate

Modul de testare a fost să verific dacă modificarea tabelelor din baza de date are loc în conformitatea cu cererile din interfața grafică.

6. Concluzii

În concluzie, prin realizarea acestei teme de laborator, s-a realizat un sistem de management al comenzilor pentru un depozit de produse, prin intermediul căruia un utilizator poate să gestioneze baza de date prin intermediul unei interfețe grafice.

Pentru proiectarea aplicației s-a folosit paradigma orientată pe obiect, folosindu-se layered Architectures, care îi oferă proprietate de reutilizare a codului/ claselor în alte aplicații

Din acest proiect am învățat noțiunile de bază cu privire la conexiunea la o bază de date printr-un server local și de asemenea la execuția de instrucțiuni SQL care au efect asupra tabelelor din baza de date asociată. M-am familiarizat cu tehnica de reflection, tehnică prin care s-a reușit generalizarea unor metode pentru evitarea redundanței.

Ca posibilă dezvoltare ar fi imbunatatirea conceptului de factura. Aceasta ar putea fi realizata intr-un fisier, luand toate datele corespunzatoare din tabela Bill.

7. Bibliografie

1. *https://dsrl.eu/courses/pt/*
2. *http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/*
3. *http://tutorials.jenkov.com/java-reflection/index.html*
4. *https://www.baeldung.com/javadoc*