

# DOCUMENTAȚIE

## TEMA 3

NUME STUDENT: Marcu Ariana-Mălina  
GRUPA: 30222

# CUPRINS

1.	Obiectivul temei.....	3
2.	Analiza problemei, modelare, scenarii, cazuri de utilizare .....	3
3.	Proiectare .....	5
4.	Implementare .....	6
5.	Rezultate .....	13
6.	Concluzii.....	13
7.	Bibliografie .....	14

# 1. Obiectivul temei

## Obiectiv principal:

Proiectarea si implementarea unei aplicatii de Management al Comenzilor utilizând un model arhitectural stratificat, în care bazele de date relaționale sunt utilizate pentru a stoca produsele, clienții și comenzile.

## Obiective secundare:

- ❖ Definirea claselor model pentru produse, clienți și comenzi. (Clase Model)
- ❖ Implementare clase de logica de business care conțin logica aplicației pentru procesarea comenzilor clienților și gestionarea depozitului. (Clase Logica de Business)
- ❖ Dezvoltarea claselor de prezentare pentru interfața grafică (GUI), permițând utilizatorilor să interacționeze cu aplicația și să plaseze comenzi. (Clase de Prezentare)
- ❖ Crearea claselor de acces la date pentru a gestiona operațiile de baze de date, inclusiv operațiile CRUD pentru produse, clienți și comenzi. (Clase Acces Date)
- ❖ Proiectarea si crearea schemei bazei de date relaționale pentru a stoca produsele, clienții și comenzile.
- ❖ Implementarea de interogări necesare pentru a extrage și manipula datele din baza de date.
- ❖ Integrarea diferitelor straturi ale aplicației utilizând mecanisme adecvate de comunicare (ex. apeluri de metode, evenimente etc.).
- ❖ Testarea și depanarea aplicației pentru a ne asigura că funcționează corect și îndeplinește cerințele.(Rezultate)
- ❖ Documentarea detaliile de proiectare și implementare ale aplicației, inclusiv diagrame, explicații ale codului și instrucțiuni de utilizare.

Prin atingerea acestor obiective secundare, se va realiza obiectivul principal de proiectare și implementare a unei aplicații de Management al Comenzilor utilizând un model arhitectural stratificat cu baze de date relaționale.

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

**Use Case:** add product

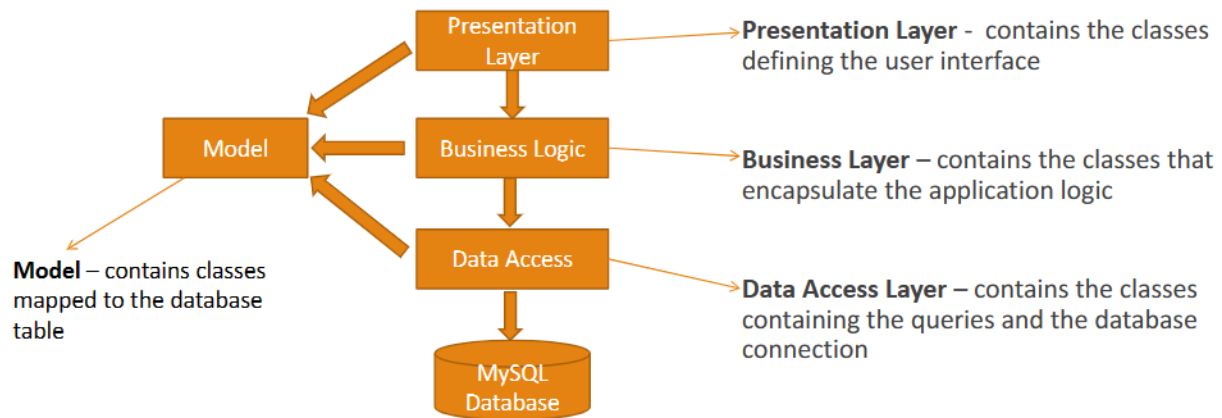
**Primary Actor:** employee

**Main Success Scenario:**

1. The employee selects the option to add a new product
2. The application will display a form in which the product details should be inserted
3. The employee inserts the name of the product, its price and current stock
4. The employee clicks on the "Add" button
5. The application stores the product data in the database and displays an acknowledge message

**Alternative Sequence:** Invalid values for the product's data

- The user inserts a negative value for the stock of the product
- The application displays an error message and requests the user to insert a valid stock
- The scenario returns to step 3



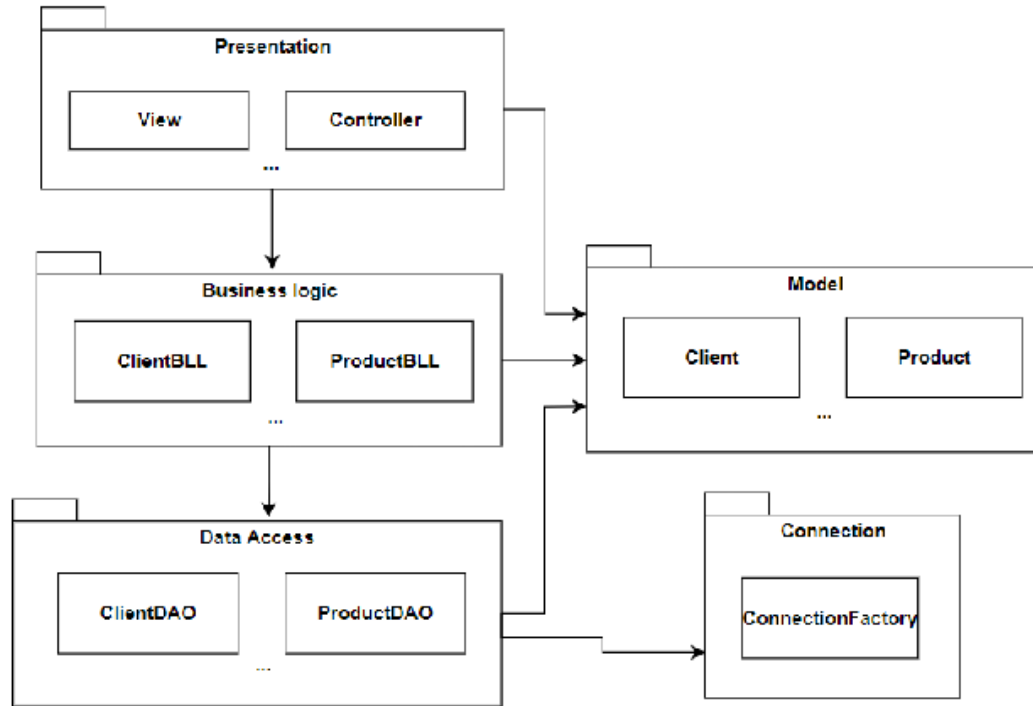
Cerințele funcționale posibile pentru o aplicație de gestionare a comenzilor ar putea include:

- Aplicația trebuie să permită unui angajat să creeze o nouă comandă pentru un client existent.
- Aplicația trebuie să permită unui angajat să actualizeze informațiile unei comenzi existente.
- Aplicația trebuie să permită adaugarea unui nou client și a unui nou produs în depozit.
- Să ofere o funcționalitate de căutare a comenzilor, permițând angajatului să găsească rapid informațiile relevante despre o comandă.
- Aplicația trebuie să ofere o funcționalitate de gestionare a stocurilor, permițând angajatului să verifice disponibilitatea produselor înainte de a plasa o comandă.

Cerințele non-funcționale posibile ar putea include:

- ✚ **Performanță:** Aplicația trebuie să fie rapidă și să ofere o experiență fluidă utilizatorilor, indiferent de numărul de clienți sau de produse existente în sistem.
- ✚ **Securitate:** Aplicația trebuie să fie securizată pentru a proteja informațiile confidențiale ale clienților și ale companiei.
- ✚ **Scalabilitate:** Aplicația trebuie să fie scalabilă, astfel încât să poată gestiona creșterea volumului de date și de utilizatori fără a compromite performanța.
- ✚ **Fiabilitate:** Aplicația trebuie să fie stabilă și să funcționeze fără întreruperi sau erori frecvente.
- ✚ **Interfață utilizator:** Aplicația trebuie să aibă o interfață intuitivă, ușor de utilizat și de navigat de către angajați, fără a necesita o formare extensivă.
- ✚ **Compatibilitate:** Aplicația ar trebui să fie compatibilă cu diferite platforme și dispozitive.
- ✚ **Ușurință în mentenanță:** Aplicația trebuie să fie ușor de întreținut și de actualizat, cu un sistem robust de gestionare a erorilor și de rezolvare a problemelor.

### 3. Proiectare



**Pachetele** in care mi-am organizat codul sunt:

**BUSINESSLOGIC** – (logica aplicatiei) in care se regaseste pachetul Validators cu clasa EmailValidator si interfata Validator; in special, in acest pachet avem clasele ClientBLL si ProductBLL. In fiecare din ele se regasesc metodele CRUD prin care se executa fiecare query in parte.

**CONNECTION** – cu clasa ConnectionFactory, cea prin care se realizeaza de fapt legatura la baza de date.

**DATA ACCESS** – cu clasele AbstractDAO, ClientDAO si ProductDAO, prin care se realizeaza accesul la datele din tabele.

**MODEL** – pachetul de baza in care se regaseste structura si componentele unui client, produs si respectiv comanda: Client, Product, Order.

**MAIN** – in care se apeleaza interfata

**PRESENTATION** – cu cele mai importante clase: GUI, ClientGUI, ProductGUI, OrderGUI, care fac posibila interactiunea cu utilizatorul.

## 4. Implementare

### 1. CLASA CLIENT

Reprezintă o entitate care stochează informațiile despre un client în sistem. Aceasta are câteva variabile de instanță private pentru a stoca id-ul clientului, numele, adresa de email și adresa fizică. Clasa oferă două constructori:

- un constructor cu patru parametri care inițializează toate variabilele instanță cu valorile primite ca argumente.
- un constructor cu trei parametri care inițializează numele, adresa de email și adresa fizică.

Pentru fiecare variabilă instanță, clasa oferă metode de acces (get) și metode de modificare (set) pentru a citi și a actualiza valorile acestora.

```
private int idClient;  
private String nume;  
private String email;  
private String adresa;
```

### 2. CLASA PRODUCT

Reprezintă o entitate care stochează informațiile despre un produs într-un sistem. Aceasta are câteva variabile de instanță private pentru a stoca id-ul produsului, numele produsului, categoria, prețul și cantitatea disponibilă. Clasa oferă mai multe metode de acces (get) și metode de modificare (set) pentru a citi și a actualiza valorile variabilelor de instanță. De asemenea, clasa include doi constructori, ca și cea anterioară, având și o metodă "toString" suprascrisă pentru a returna o reprezentare text a obiectului "Product", care include valorile tuturor variabilelor de instanță (am folosit-o pentru testare la început).

```
private int idProdus;  
private String numeProdus;  
private String categorie;  
private float pret;  
private int cantitate;
```

### 3. CLASA ORDER

Reprezintă o comandă pentru un produs plasată de către o persoană. Aceasta conține câmpuri pentru identificatorul persoanei, identificatorul produsului și cantitatea comandată. Clasa oferă metode pentru a accesa și modifica valorile câmpurilor. **Cele 3 clase prezentate pana acum, pun bazele pachetului Model.**

```
private int idPerson;  
private int idProdus;  
private int cantitate;
```

#### 4. CLASA CONNECTIONFACTORY

Clasa "ConnectionFactory" din pachetul **Connection** este responsabilă de gestionarea conexiunii la o bază de date MySQL (în cazul meu numită **ordersmanagement**). Aceasta conține informații despre driverul MySQL, locația bazei de date, utilizatorul și parola necesare pentru conectare. Clasa oferă metode pentru a inițializa și obține conexiunea la baza de date, pregăti instrucțiuni SQL, executa interogări și închide conexiunea.

Prin intermediul metodei `init()`, clasa încarcă driverul MySQL, stabilește conexiunea la baza de date utilizând informațiile specificate și returnează obiectul de conexiune. Metoda `prepareStatement()` permite pregătirea instrucțiunilor SQL pentru execuție, în timp ce metodele `executeQuery()` și `selectQuery()` permit executarea și obținerea rezultatelor unei interogări. Metodele `close()` sunt folosite pentru închiderea conexiunii la baza 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://127.0.0.1:3306/ordersmanagement";
private static final String USER = "root";
private static final String PASS = "Anca73#@Adi74"; //parola la SQLul propriu
public static Connection connection;
public ConnectionFactory()
{
    init();
}
```

#### 5. CLASA MAIN

Tot ce face Main-ul este să apeleze interfața. Tototdeauna, comentând pasajul, putem vedea și funcționalitatea de la reflection:

```
GUI gui = new GUI();
```

```
ClientBLL clientBLL = new ClientBLL(c);
Client client1 = null;
try {
    client1 = clientBLL.findClientById(10);
} catch (Exception ex) {
    LOGGER.log(Level.INFO, ex.getMessage());
}
Reflection.retrieveProperties(client1);
```

#### 6. CLASA CLIENTBLL

Clasa `ClientBLL` din pachetul `BusinessLogic` gestionează operațiile logice de afaceri legate de clienți. Iată o scurtă descriere a metodelor și funcționalităților oferite:

- \* Metoda `insert(Client client)` inserează un client în baza de date.
- \* Metoda `delete(int id)` șterge un client din baza de date pe baza ID-ului specificat.
- \* Metoda `edit(Client c)` actualizează informațiile unui client în baza de date.

- \* Metoda view() recuperează toți clienții din baza de date și returnează un obiect ResultSet care conține înregistrările clienților.
- \* Metoda readClientNames() recuperează numele tuturor clienților din baza de date și le returnează într-un ArrayList.
- \* Metoda getIdByName(String nume) recuperează ID-ul unui client pe baza numelui acestuia.
- \* Metoda findClientById(int id) găsește un client în baza de date pe baza ID-ului specificat și returnează obiectul de tip Client.

Această clasă folosește și o clasă ConnectionFactory pentru a stabili conexiunea cu baza de date și a executa interogări.

## 7. CLASA ORDERBLL

Clasa OrderBLL din pachetul BusinessLogic gestionează inserarea și ștergerea comenzilor în baza de date. Aceasta utilizează o conexiune la baza de date prin intermediul clasei ConnectionFactory.

```
public void insert(Order o)
{
    String sql = "INSERT INTO orders (idPerson, idProdus, cantitate) VALUES (" + o.getIdPerson() + ", " +
        o.getIdProdus() + ", " + o.getCantitate() + ")";
    connectionFactory.executeQuery(sql);
}
public void delete(int idPerson)
{
    String sql = "DELETE FROM orders WHERE idPerson=" + idPerson;
    connectionFactory.executeQuery(sql);
}
```

## 8. CLASA PRODUCTBLL

Clasa ProductBLL este responsabilă de gestionarea operațiilor de logică de afaceri legate de produse în cadrul aplicației. Ea oferă următoarele funcționalități:

- \* insert(Product product): Inserează un obiect Product în baza de date. Informațiile despre produs sunt extrase din obiectul product și sunt utilizate pentru a construi și executa instrucțiunea SQL corespunzătoare în baza de date.
- \* delete(int id): Șterge un produs din baza de date bazat pe ID-ul specificat. Se construiește și se execută instrucțiunea SQL pentru ștergerea produsului cu ID-ul respectiv.
- \* update(Product p): Actualizează informațiile unui produs existent în baza de date. Informațiile actualizate sunt extrase din obiectul p și sunt utilizate pentru a construi și executa instrucțiunea SQL corespunzătoare în baza de date.
- \* view(): Recuperează toate produsele din baza de date sub forma unui ResultSet. Se construiește și se execută instrucțiunea SQL pentru a selecta toate înregistrările din tabela produs.



- \* `readProductNames()`: Recuperează numele tuturor produselor din baza de date sub forma unui `ArrayList`. Se construiește și se execută instrucțiunea SQL pentru a selecta numele tuturor produselor din tabela produs.
- \* `getIdByName(String nume)`: Recuperează ID-ul unui produs bazat pe numele său. Se construiește și se execută instrucțiunea SQL pentru a selecta ID-ul produsului cu numele specificat.
- \* `getId(int id)`: Recuperează un produs din baza de date bazat pe ID-ul specificat. Se construiește și se execută instrucțiunea SQL pentru a selecta produsul cu ID-ul respectiv.
- \* `updateCantitate(int idProdus, int newValue)`: Actualizează cantitatea unui produs în baza de date. Se construiește și se execută instrucțiunea SQL pentru a actualiza cantitatea produsului cu ID-ul specificat la noua valoare.

## 9. CLASA ABSTRACTDAO

Una dintre cele mai importante clase ale proiectului în ceea ce privește **java reflection**, clasa `AbstractDAO<T>` este o clasă generică care furnizează operații comune de creare, citire, actualizare și ștergere (CRUD) pentru entități. Aceasta este folosită pentru a implementa obiecte de acces la date pentru diferite tipuri de entități din aplicație. Principala funcționalitate a clasei constă în construirea și executarea interogărilor SQL pentru a accesa și manipula datele din baza de date. Aceasta include:

- \* Metode pentru crearea de interogări `SELECT` pentru selectarea unui câmp specific sau a tuturor înregistrărilor unei entități.
- \* Metode pentru găsirea și recuperarea entităților din baza de date, folosind interogări `SELECT` și criterii specifice, cum ar fi ID-ul entității.
- \* Metoda pentru crearea de obiecte de tip `T` din rezultatele unei interogări `SELECT`, folosind reflexia și introspecția pentru a atribui valorile câmpurilor obiectului.
- \* Metode pentru inserarea și actualizarea entităților în baza de date, prin construirea și executarea interogărilor `INSERT` și `UPDATE`.
- \* Această clasă abstractă poate fi extinsă pentru a crea obiecte de acces la date specifice pentru fiecare entitate în parte. Aceasta oferă o abstracție convenabilă pentru a gestiona operațiile CRUD comune în aplicație, fără a fi necesară rescrierea codului pentru fiecare entitate în parte.

```
protected static final Logger LOGGER =
    Logger.getLogger(AbstractDAO.class.getName());
private final Class<T> type;
```

## 10. CLASA CLIENTDAO

Clasa `ClientDAO` este responsabilă de operațiile de acces la date legate de entitățile de tip client. Aceasta extinde clasa `AbstractDAO<Client>`, ceea ce înseamnă că moștenește funcționalitatea comună de bază pentru operațiile CRUD. Funcționalitatea principală a clasei `ClientDAO` include: metoda `findById(int clientId)`: Această metodă recuperează un client din baza de date pe baza unui ID specificat. Aceasta construiește și execută o interogare `SELECT` care selectează înregistrarea corespunzătoare ID-ului specificat și creează un obiect `Client` cu datele obținute din rezultatul interogării.

```
public class ClientDAO extends AbstractDAO<Client> {
    protected static final Logger LOGGER =
Logger.getLogger(ClientDAO.class.getName());
    private static final String insertStatementString = "INSERT INTO person
(nume,email,adresa)"
        + " VALUES (?, ?, ?)";
    private static final String findStatementString = "SELECT * FROM person
where id = ?";
```

## 11. CLASA REFLECTION

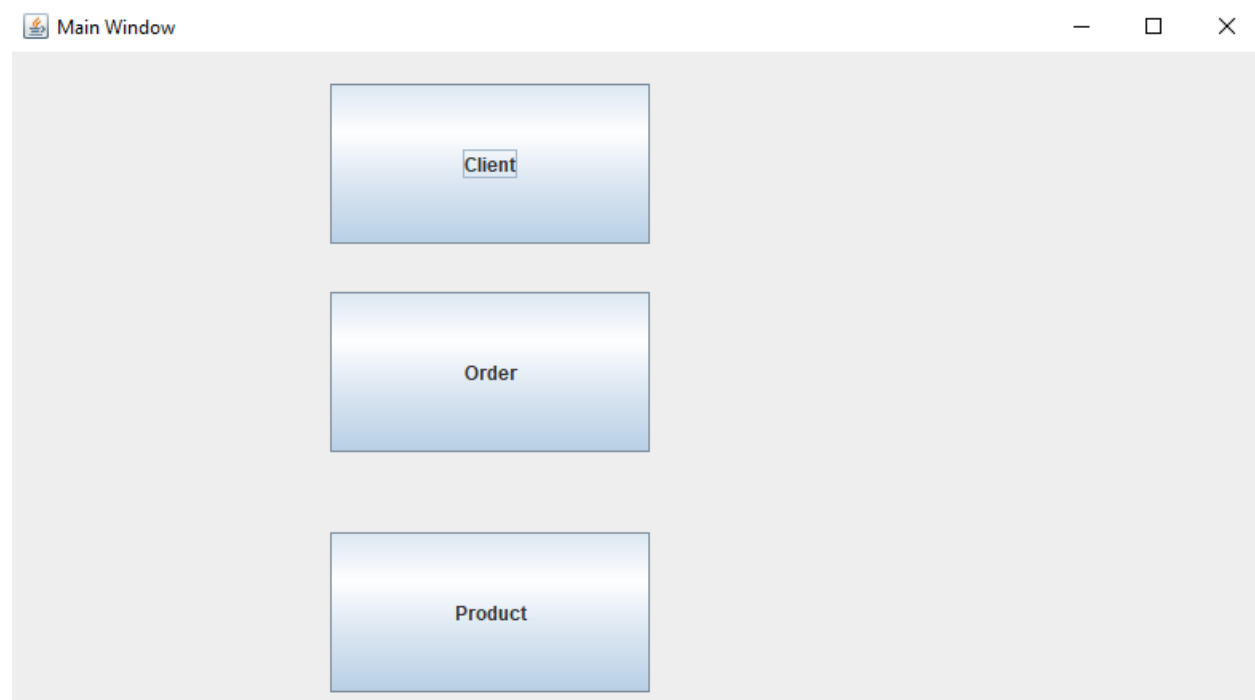
Clasa Reflection furnizează metode de utilitate pentru recuperarea proprietăților unui obiect folosind reflexia. Funcționalitatea principală a clasei Reflection include metoda `retrieveProperties(Object object)`. Această metodă primește un obiect ca parametru și utilizează reflexia pentru a recupera proprietățile obiectului. Utilizând `object.getClass().getDeclaredFields()`, se obține o listă a câmpurilor declarate ale obiectului. Apoi, pentru fiecare câmp, se permite accesul la acesta prin apelarea metodei `setAccessible(true)`. Apoi, se extrage valoarea câmpului utilizând metoda `get(object)` și se afișează numele câmpului.

## 12. CLASA GUI

Reprezintă interfața grafică principală a aplicației. Ea conține butoane pentru gestionarea clienților, produselor și comenzilor. Fiecare buton are asociat un ascultător de evenimente care deschide o fereastră GUI specifică (ClientGUI, OrderGUI, ProductGUI) când este apăsat.

```
public JButton butonClient, butonProdus, butonComanda;
ConnectionFactory connectionFactory = null;

public GUI()
{
    connectionFactory = new ConnectionFactory();
    connectionFactory.executeQuery("set SQL_SAFE_UPDATES = 0");
    init();
}
```



### 13. CLASA CLIENTGUI

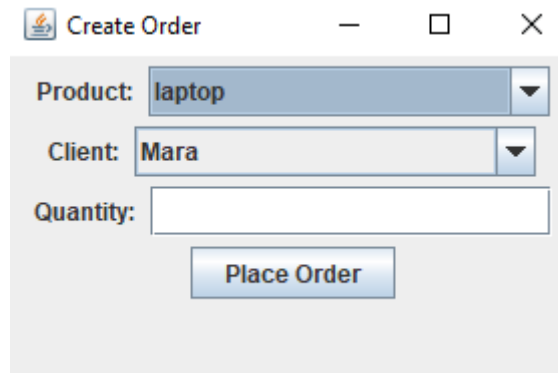
Reprezintă interfața grafică pentru gestionarea clienților în aplicație. Ea permite adăugarea, editarea și ștergerea clienților, precum și vizualizarea acestora într-un tabel. Utilizează clasele **ClientBLL** și **OrderBLL** pentru a interacționa cu baza de date prin intermediul clasei **ConnectionFactory**. Interacțiunea cu utilizatorul se realizează prin intermediul butoanelor și câmpurilor de introducere text. Metodele **fetchProductDataFromDatabase**, **populateTable**, **deleteClient**, **editNewClient** și **addNewClient** implementează logica necesară pentru preluarea datelor din baza de date, popularea tabelului, ștergerea și editarea clienților. Clasa utilizează obiecte **JFrame**, **JLabel**, **JBUTTON** și **JTextField** pentru construirea interfeței grafice.



### 14. CLASA ORDERGUI

Reprezintă interfața grafică pentru crearea de comenzi în aplicație. Aceasta permite utilizatorului să selecteze un produs dintr-un combobox, un client din alt combobox și să introducă cantitatea dorită într-un câmp de text. Apoi, prin apăsarea butonului "*Place Order*", comanda este plasată în baza de date. Clasa utilizează clasele **ProductBLL**, **ClientBLL** și **OrderBLL** pentru a interacționa cu baza de date prin intermediul clasei **ConnectionFactory**.

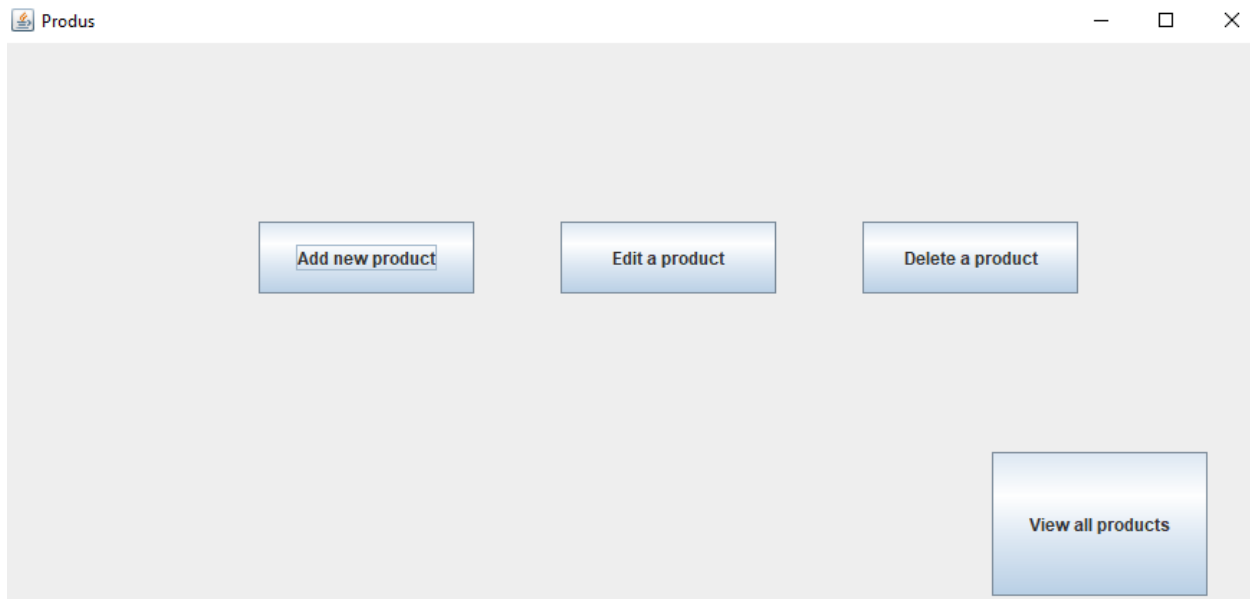
Metoda **initializeComponents** inițializează componentele interfeței grafice, precum **combobox**-urile pentru produse și clienți, câmpul de introducere a cantității și butonul pentru plasarea comenzii. Metoda **populateComboBoxes** populează combobox-urile cu datele corespunzătoare din baza de date, prin intermediul metodelor **readProductNames** și **readClientNames** ale claselor **ProductBLL** și **ClientBLL**. Metoda **placeOrder** este apelată când utilizatorul dorește să plaseze o comandă și extrage informațiile despre produsul, clientul și cantitatea selectate. Apoi, verifică dacă cantitatea disponibilă a produsului este suficientă pentru a face comanda și actualizează cantitatea disponibilă în baza de date. În caz contrar, se afișează un mesaj de eroare.



## 15. CLASA PRODUCTGUI

Reprezintă interfața grafică pentru gestionarea produselor într-o aplicație. Aceasta oferă utilizatorului funcționalități pentru adăugarea, editarea, ștergerea și vizualizarea produselor într-un tabel. Componentele GUI includ butoane pentru adăugarea, editarea și ștergerea unui produs, precum și un buton pentru vizualizarea tuturor produselor. Când utilizatorul dă clic pe butonul "Vizualizare toate produsele", se afișează o fereastră separată cu un tabel ce conține datele tuturor produselor din baza de date.

Clasa utilizează obiecte din straturile **BusinessLogic** și **DataAccess** pentru a interacționa cu baza de date și a efectua operațiile **CRUD** (Create, Read, Update, Delete) asupra produselor. Aceasta se bazează pe obiectul "productBLL" de tip "ProductBLL" pentru a realiza operațiile necesare. În metoda "**viewAll()**", se obține lista de produse din baza de date utilizând obiectul "productBLL" și se populează tabelul cu aceste date utilizând un obiect de tip "**DefaultTableModel**". Metodele "**addNewProduct()**", "**editProduct()**" și "**deleteProduct()**" oferă interacțiunea cu utilizatorul pentru adăugarea, editarea și ștergerea produselor din baza de date.



## 5. Rezultate

Am obtinut rezultate corecte la fiecare testare, atat din interfata cat si cand am rulat(diferite teste sunt puse in comentariu). M-am folosit de javadoc, o utilitate în limbajul de programare Java care permite generarea automată a documentației pentru codul sursă. Această documentație poate fi ulterior accesată de către dezvoltatori pentru a înțelege mai bine funcționalitatea și utilizarea codului. Mai jos, pentru o mai buna vizibilitate a tabelelor pe care le-am folosit, am atasat o imagine chiar din baza de date personala:

```
12 • SELECT * FROM person;
13 • DELETE FROM orders;
14 • set global FOREIGN_KEY_CHECKS = 0;
```

	id	nume	email	adresa
▶	8	Mara	mara@yahoo.com	brazilor 15
	9	David Bogdan	davidM@gmail.	Daicoviciu18
	10	Alexandru	alex@personal.ro	Iugoslaviei 72
*	NULL	NULL	NULL	NULL

```
10 • SELECT * FROM orders;
11 • SELECT * FROM produs;
```

	id	nume	categorie	pret	cantitate
	9	laptop	hp	2000	NULL
	11	calculator	nou	3000	1
	12	new	product	15	6
	13	samponul	aici	30	5
	15	Cartea	Literatura	33.5	5
*	NULL	NULL	NULL	NULL	NULL

```
10 • SELECT * FROM orders;
```

	idPerson	idProdus	cantitate
▶	7	6	1
	11	6	1
	13	9	3
*	NULL	NULL	NULL

OVERVIEW	PACKAGE	CLASS	TREE	INDEX	HELP
----------	---------	-------	------	-------	------

Packages	
Package	Description
BusinessLogic	
BusinessLogic.Validators	
Connection	
DataAccess	
Model	
org.example	
Presentation	
Start	

## 6. Concluzii

Proiectarea aplicatiei de gestionare a comenzilor implică înțelegerea nevoilor angajaților și clienților, identificarea funcționalităților necesare și definirea cerințelor pentru a dezvolta o soluție adecvată. Am inteles importanța facilitării unui flux eficient al comenzilor, gestionării stocurilor, și asigurării securității datelor. De asemenea, am inteles că interfața utilizator intuitivă și ușurința în utilizare sunt esențiale pentru a spori eficiența și productivitatea angajaților. Totodata, datorita acestei teme, am invatat sa folosesc reflection techniques, topic despre care nu stiam inainte, si mai ales, sa reusesc sa vad functionalitate in timp real, in ceea ce priveste editarea unor tabele, lucrul cu sql, inserari, stergeri si multe altele.

### Dezvoltări ulterioare posibile:

- Implementarea unei funcționalități de gestionare a plăților și a facturilor, spre exemplu.
- Adăugarea de funcționalități de comunicare și notificare între angajați și clienți, cum ar fi alerte pentru actualizări de comenzi sau întârzieri în livrare.
- Integrarea cu alte sisteme și platforme utilizate în cadrul companiei, cum ar fi sistemul de contabilitate sau platformele de comerț electronic.
- Dezvoltarea unei aplicații mobile, care să permită angajaților accesul la informații și gestionarea comenzilor de pe dispozitivele lor mobile.
- Îmbunătățirea performanței și optimizarea aplicației pentru a reduce timpul de răspuns și a permite gestionarea eficientă a volumelor mari de date.

## **7. Bibliografie**

1. *Bruce Eckel, Thinking in Java (4th Edition), Publisher: Prentice Hall PTR Upper Saddle River, NJ United States, ISBN:978-0-13-187248-6 Published:01 December 2005.*
2. [www.tutorialspoint.com](http://www.tutorialspoint.com)
3. [sql - java.sql.SQLException: Column not found - Stack Overflow](https://stackoverflow.com/questions/44209904/sql-jdbc-sql-exception-column-not-found)
4. [src/main/java/start/ReflectionExample.java · master · Utcn Dsrl / pt-reflection-example · GitLab](https://github.com/utcn-dsrl/pt-reflection-example)
5. [Guide to Java Reflection | Baeldung](http://www.baeldung.com/java-jdbc)
6. <https://www.baeldung.com/java-jdbc>
7. <http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/>
8. <https://dzone.com/articles/layers-standard-enterprise>
9. <https://www.baeldung.com/java-pdf-creation>
10. <https://www.baeldung.com/javadoc>