Universitatea Națională de Știință și Tehnologie
POLITEHNICA București

PROIECT 3

Profesor coordonator: Ș. I. Dr. Ing. Pupezescu Valentin

STUDENT: Angan Lavinia Marilena

GRUPA: 443A

CUPRINS

1. *Tema proiectului*
2. *Microsoft SQL Server*
3. *Tehnologia Java Swing*
4. *Descrierea aplicației*
5. *Concluzii*
6. *Bibliografie*

Tema proiectului

Tema proiectului se bazează pe dezvoltarea unei aplicații care să conțină o bază de date creată în MS SQL Server și o interfață pentru aceasta. La crearea interfeței se va folosi tehnologia Java Swing.

Interfața va trebui să permită utilizatorului să execute operații CRUD pe toate tabelele, adică vizualizare, adăugare, modificare și ștergere de date. Vizualizarea tabelelor de legătură va presupune vizualizarea datelor referite din celelalte tabele. Asocierea pentru tabelele din baza de date este de M:N.

Microsoft SQL Server

Microsoft SQL Server este un sistem de gestionare a bazelor de date relaționale (RDBMS) dezvoltat de Microsoft, care facilitează stocarea și prelucrarea datelor pentru diverse aplicații de tranzacționare, inteligență de afaceri și analize de date în mediile IT corporative.

La baza sa, SQL Server utilizează limbajul standardizat Structured Query Language (SQL) pentru gestionarea bazelor de date și interogarea datelor. În plus, este integrat cu Transact-SQL (T-SQL), limbajul proprietar al Microsoft, care permite aplicațiilor și instrumentelor să comunice și să se conecteze la o instanță sau bază de date SQL Server.

Componenta principală a SQL Server este Database Engine, responsabilă de stocarea, accesul, procesarea și securitatea datelor. Aceasta include un motor relațional care procesează comenzi și interogări, precum și un motor de stocare care gestionează fișierele bazei de date, tabelele, paginile, indexurile, buffer-ele de date și tranzacțiile.

De-a lungul anilor, Microsoft a lansat multiple versiuni ale SQL Server, extinzându-i capacitățile pentru a concura cu alte platforme RDBMS de top, precum Oracle Database și IBM Db2. Fiecare versiune a adus funcționalități noi, inclusiv suport pentru tehnologii emergente, cum ar fi cloud computing și dispozitive mobile.

SQL Server oferă, de asemenea, diverse componente și tehnologii suplimentare, cum ar fi:

- **Machine Learning Services (MLS):** Integrează limbajele R și Python pentru a facilita învățarea automată în fluxurile de lucru enterprise.
- **Integration Services (SSIS):** O platformă pentru construirea soluțiilor de integrare a datelor, inclusiv procese ETL (extracție, transformare și încărcare) pentru depozitarea datelor.
- **Analysis Services (SSAS):** O platformă analitică pentru inteligența de afaceri, care suportă soluții OLAP tradiționale și modelare tabulară.
- **Reporting Services (SSRS):** Furnizează funcționalități de raportare web la nivel enterprise, permițând crearea și gestionarea centralizată a rapoartelor din diverse surse de date.
- **Replication:** Tehnologii pentru copierea și distribuirea datelor și obiectelor bazei de date între diferite locații, menținând consistența prin sincronizare.
- **Data Quality Services (DQS):** O soluție bazată pe cunoștințe pentru curățarea datelor, permițând corectarea și deduplicarea acestora prin metode asistate de computer și interactive.

Aceste componente extind funcționalitățile SQL Server, oferind soluții complete pentru gestionarea și analiza datelor în diverse scenarii de afaceri.

Deși SQL Server este un produs de sine stătător, care poate fi instalat pe calculatoare care rulează sistemele de operare Windows și Linux, poți integra instanțele SQL Server cu mai multe servicii Azure. Mașinile Virtuale Azure permit utilizarea versiunilor complete ale SQL Server în cloud, fără a fi necesar să gestionezi hardware-ul pe premise. Mașinile virtuale SQL Server simplifică și costurile de licențiere, oferind opțiunea de plată pe măsură ce se utilizează. Aceste mașini virtuale sunt disponibile în diverse regiuni geografice și cu diferite dimensiuni, iar galeria de imagini pentru mașini virtuale permite crearea unui VM SQL Server cu versiunea și sistemul de operare corespunzător, fiind o opțiune excelentă pentru multe tipuri de sarcini SQL Server.

Java Swing

Java Swing este o bibliotecă de interfață grafică pentru utilizator (GUI) inclusă în Java Foundation Classes (JFC). Este un set avansat de componente grafice dezvoltat pentru a extinde capabilitățile Abstract Window Toolkit (AWT). Spre deosebire de AWT, Swing oferă componente complet independente de platformă, ceea ce înseamnă că aplicațiile dezvoltate cu Swing pot avea un comportament și un aspect uniform pe toate sistemele de operare.

Swing este utilizat pentru a construi aplicații desktop interactive și intuitive, având suport pentru componente grafice avansate precum ferestre, butoane, meniuri, tabele, arbori și panouri cu file.

Cum funcționează tehnologia Swing?

Swing este bazat pe arhitectura Model-View-Controller (MVC), care separă logica aplicației (Model), gestionarea interacțiunilor utilizatorului (Controller) și reprezentarea grafică (View). Această separare face ca aplicațiile să fie mai ușor de întreținut și de extins.

Componentele Swing sunt "lightweight," ceea ce înseamnă că nu folosesc componente native ale sistemului de operare pentru a desena pe ecran. În schimb, toate componentele sunt desenate utilizând Java 2D API, asigurând consistența grafică între platforme.

Swing oferă de asemenea un sistem flexibil de "Look and Feel," ceea ce permite schimbarea stilului vizual al aplicației pentru a semăna cu cel al unui anumit sistem de operare (Windows, macOS, etc.) sau pentru a adopta un stil personalizat.

Avantajele Java Swing

1. **Portabilitate ridicată**: Aplicațiile dezvoltate cu Swing pot rula pe orice platformă care suportă Java, fără a necesita modificări ale codului.
2. **Set extins de componente**: Swing include componente grafice complexe precum:
    - JTable (pentru afișarea datelor tabelare).
    - JTree (pentru afișarea ierarhiilor de date).
    - JTabbedPane (pentru panouri cu file).
    - JList, JComboBox, și multe altele.
3. **Flexibilitate și personalizare**: Dezvoltatorii pot modifica aspectul și comportamentul componentelor pentru a se potrivi cu cerințele specifice ale aplicației. Posibilitatea de a crea componente personalizate de la zero.

4. **Arhitectură bazată pe evenimente**: Swing folosește un sistem de ascultători (listeners) pentru a răspunde la acțiunile utilizatorilor, cum ar fi apăsarea unui buton sau selectarea unui element dintr-o listă.
5. **Maturitate și stabilitate**: Fiind utilizat pe scară largă de ani de zile, Swing este bine documentat și susținut de comunitatea Java.

Dezavantajele Java Swing

1. **Performanță redusă**: Fiind bazat pe Java și neutilizând componente native, aplicațiile Swing pot consuma mai multe resurse și pot fi mai lente în comparație cu aplicațiile care folosesc framework-uri native.
2. **Aspect grafic mai puțin nativ**: Deși Swing poate emula aspectul diferitelor platforme, utilizatorii pot observa diferențe subtile între aplicațiile Swing și cele dezvoltate nativ pentru un anumit sistem de operare.
3. **Complexitate sporită**: Deși flexibilitatea este un avantaj, poate fi dificil pentru dezvoltatorii începători să înțeleagă și să implementeze interfețe complexe.

Java Swing este important deoarece permite crearea de aplicații robuste și portabile, este o soluție matură și stabilă pentru dezvoltarea aplicațiilor desktop și este ușor integrabil cu alte tehnologii Java, precum baze de date, rețele și sisteme de fișiere.

Swing rămâne o alegere populară pentru aplicațiile Java desktop, chiar dacă există tehnologii mai noi precum JavaFX.

De ce este util să afișăm datele într-un JTable sau altă componentă grafică?

Afișarea datelor într-un JTable sau alte componente grafice aduce multiple beneficii:

1. **Interactivitate**: Utilizatorii pot explora datele mai ușor prin navigare, sortare sau selectarea rândurilor direct din interfață.
2. **Accesibilitate vizuală**: Datele sunt afișate într-un format organizat, tabelar, ceea ce le face mai ușor de înțeles și utilizat.
3. **Funcționalitate extinsă**: Utilizatorii pot efectua operațiuni direct pe date, cum ar fi adăugarea, modificarea sau ștergerea acestora, prin interfața grafică.
4. **Profesionalism**: O interfață grafică bine realizată oferă un aspect profesional aplicației, ceea ce este important pentru utilizatorii finali.

Cum am implementat afișarea datelor în JTable?

În cadrul aplicației, am utilizat componente Swing pentru a crea panouri dedicate fiecărei tabele din baza de date (Clienti, Produse, etc.). Fiecare panou include:

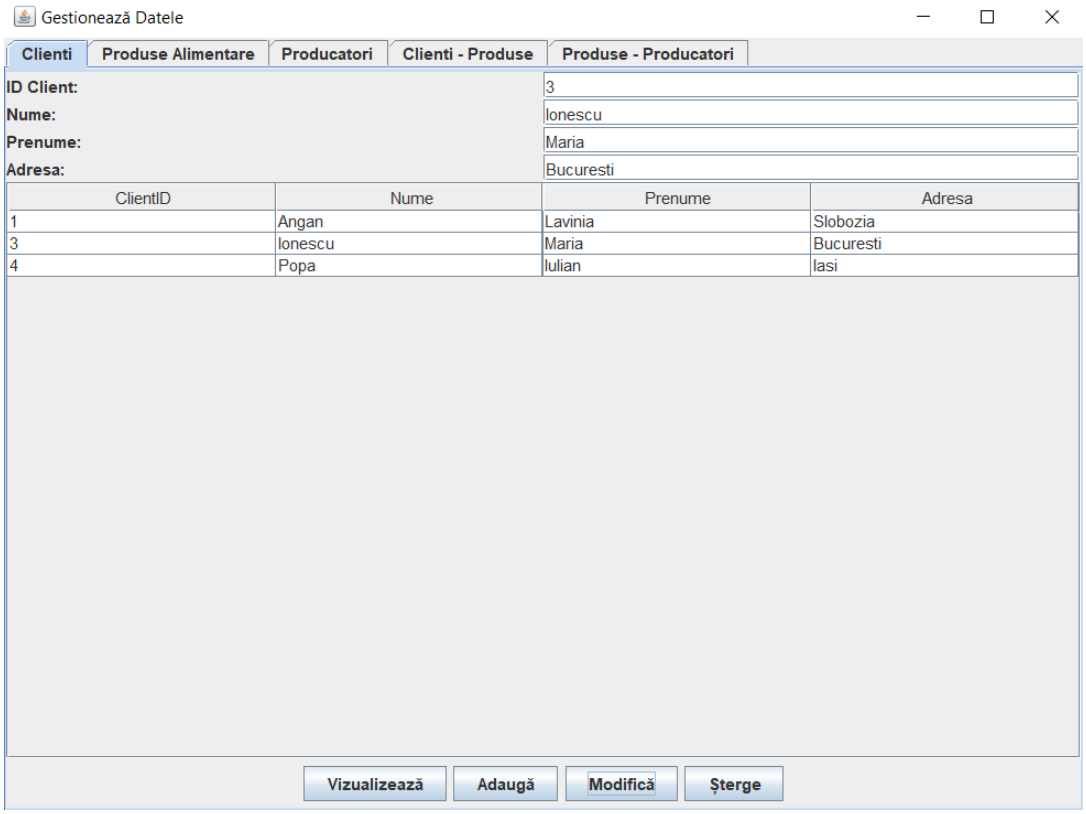
- Un JTable pentru afișarea datelor.
- Câmpuri text (JTextField) pentru introducerea de informații.
- Butoane (JButton) pentru efectuarea operațiunilor de bază: vizualizare, adăugare, modificare, ștergere.

Exemplu de flux pentru operațiuni:

1. Utilizatorul introduce datele într-un formular.
2. Apasă pe un buton precum "Adaugă."
3. Metoda corespunzătoare interacționează cu baza de date utilizând un PreparedStatement.
4. JTable este actualizat automat pentru a reflecta noile date.

Această abordare combină puterea SQL pentru manipularea datelor cu interactivitatea oferită de Swing.

Descrierea aplicației



Baza de date

Tema individuală se bazează pe crearea unei baze de date compusă din următoarele tabele:

1. Clienti(cu atributele: ClientID, Nume, Prenume, Adresa)
2. ProdusAlimentar(cu atributele: ProdusID, Denumire, DataProducere, DataExpirare)
3. Producatori(cu atributele: ProducatorID, Denumire, TaraOrigine, Adresa).

Asocierile între tabele sunt asocieri de tip M:N între tabela **Clienti** şi tabela **ProdusAlimentar** și între tabela **ProdusAlimentar** şi tabela **Producatori**.

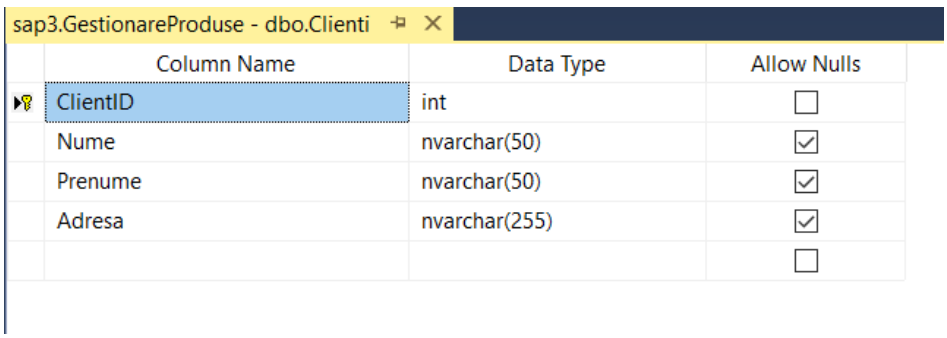


Figura 1: *tabela Clienti*

sap3.GestionarePr...bo.ProdusAlimentar | sap3.GestionarePro...e - dbo.Producatori

	Column Name	Data Type	Allow Nulls
🔑	ProducatorID	int	☐
	Denumire	nvarchar(100)	☑
	TaraOrigine	nvarchar(50)	☑
	Adresa	nvarchar(255)	☑
			☐

Figura 2: *tabela Producatori*

sap3.GestionarePr...bo.ProdusAlimentar | sap3.GestionarePro...e - dbo.Producatori

	Column Name	Data Type	Allow Nulls
🔑	ProdusID	int	☐
	Denumire	nvarchar(100)	☑
	DataProducere	date	☑
	DataExpirare	date	☑
			☐

Figura 3: *tabela ProdusAlimentar*

Ce este asocierea M:N?

Asocierea M:N este o relație între două tabele într-o bază de date care permite unui element din prima tabelă să aibă mai multe legături cu mai multe elemente din a doua tabelă și invers. Acest lucru se realizează prin intermediul unei tabele intermediere care conține informații despre legăturile dintre elementele din cele două tabele.

Ce reprezintă tabela de legatură?

Pentru a crea o relație mai-mulți-la-mai-mulți, trebuie să se creeze o a treia tabelă denumită deseori tabelă de joncțiune, care împarte relația mai-mulți-la-mai-mulți în două relații unu-la-mai-mulți. În cazul nostru, am ales ca și tabele de legatură tabela **Clienti_ProdusAlimentar** și tabela **ProdusAlimentar_Producatori**. În aceste noi tabele, atributele ce au fost selectate ca și chei primare pentru tabelele anterioare vor deveni chei străine (FK) pentru tabela de legatură.

sap3.GestionarePro...ti_ProdusAlimentar

	Column Name	Data Type	Allow Nulls
🔑	ClientID	int	☐
🔑	ProdusID	int	☐
			☐

Figura 4: *tabela Clienti_ProdusAlimentar*

sap3.GestionarePro...mentar_Producatori

	Column Name	Data Type	Allow Nulls
🔑	ProdusID	int	☐
🔑	ProducatorID	int	☐
			☐

Figura 5: *tabela ProdusAlimentar_Producatori*

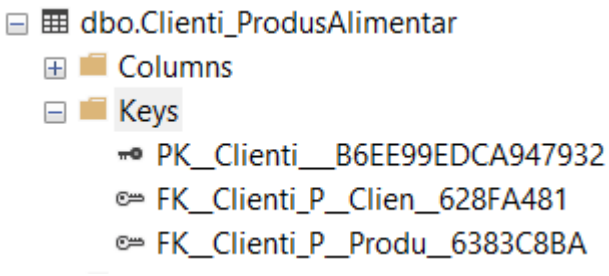


Figura 6.a: *tabela Clienti_ProdusAlimentar foreign keys*

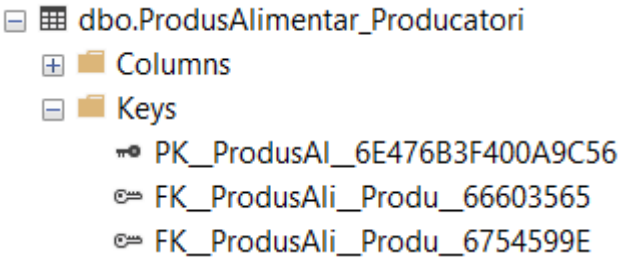
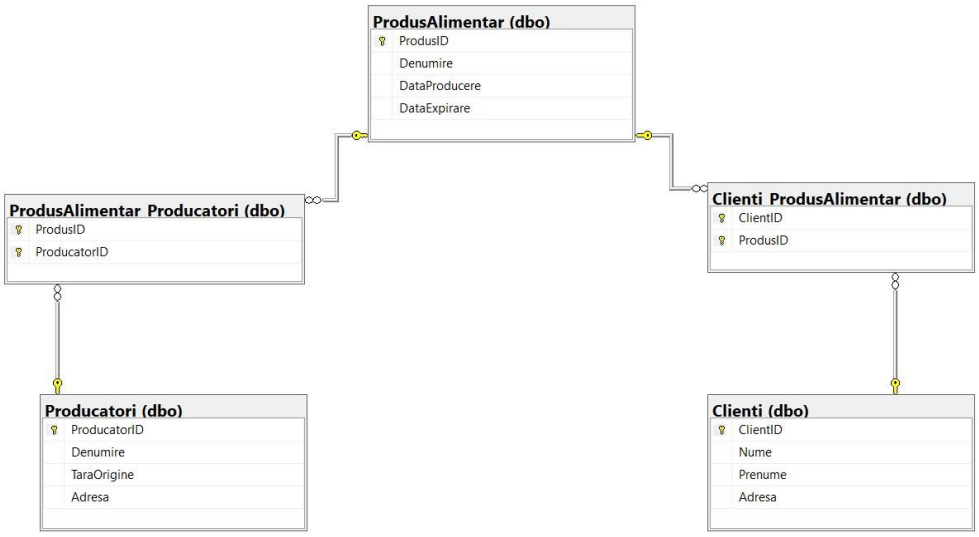


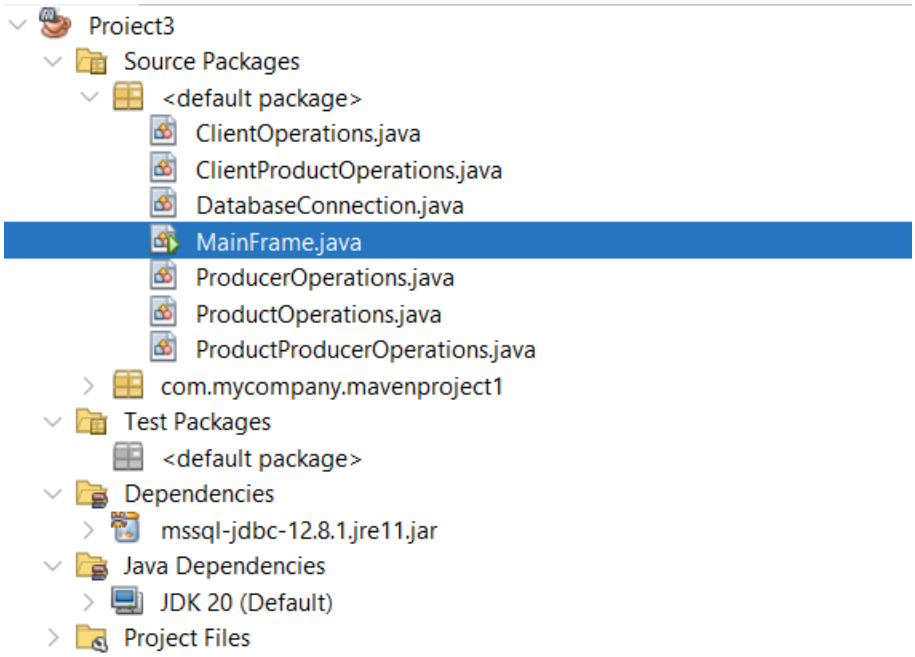
Figura 6.b: *tabela ProdusAlimentar_Producatori foreign keys*

Diagrama logică a bazei de date (Diagrama ERD)



Funcționalitatea aplicației

1. *Arhitectura proiectului*



Proiectul este organizat într-un singur pachet *<default package>*, ceea ce sugerează că toate clasele sunt în directorul rădăcină.

- **MainFrame.java**: este responsabilă pentru interfața grafică (GUI) a aplicației, utilizează Swing pentru a crea un tabbed pane cu diferite secțiuni pentru gestionarea datelor.
- **ClientOperations.java, ProducerOperations.java, ProductOperations.java, ProductProducerOperations.java, ClientProductOperations.java**: aceste clase implementează logica de interacțiune cu baza de date pentru fiecare entitate sau tabel. Fiecare dintre ele conține metode pentru operațiuni CRUD (Creare, Citire, Actualizare, Ștergere), organizate logic după tipul de date gestionate.
- **DatabaseConnection.java**: asigură conexiunea la baza de date utilizând biblioteca JDBC. Este punctul central pentru configurarea conexiunii către serverul MSSQL (bazat pe faptul că este *biblioteca mssql-jdbc-12.8.1.jre11.jar* în dependențe).
- **mssql-jdbc-12.8.1.jre11.jar**: este driver-ul JDBC necesar pentru conectarea aplicației la o bază de date Microsoft SQL Server.
- **JDK 20**: aplicația este construită folosind Java 20. Este important să te asiguri că versiunea driverului JDBC este compatibilă cu JDK 20.
- **Swing pentru GUI**: am folosit biblioteca Swing pentru a crea o interfață grafică cu mai multe tab-uri pentru gestionarea entităților și relațiilor.
- **JDBC pentru acces la date**: aplicația utilizează JDBC pentru conectarea și interacțiunea cu baza de date. Fiecare metodă din clasele Operations execută interogări SQL directe.

2. *Implementarea functiilor*

Fișierul pom.xml este utilizat pentru configurarea proiectului și managementul dependențelor în cadrul Maven. Acesta definește structura proiectului, dependențele externe, proprietățile proiectului și setările de compilare.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocatior
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.mycompany</groupId>
    <artifactId>mavenproject1</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>com.microsoft.sqlserver</groupId>
            <artifactId>mssql-jdbc</artifactId>
            <version>12.8.1.jre11</version>
        </dependency>

    </dependencies>
```

Blocul <*dependencies*> definește biblioteca externă utilizată de proiect:, **mssql-jdbc**, driverul JDBC pentru conectarea la o bază de date Microsoft SQL Server. Versiunea specificată este 12.8.1.jre11, care este compatibilă cu Java 11 și versiuni ulterioare. Maven descarcă automat această bibliotecă și o adaugă în classpath-ul proiectului.

```
    <packaging>jar</packaging>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>20</maven.compiler.source>
        <maven.compiler.target>20</maven.compiler.target>
        <exec.mainClass>com.mycompany.mavenproject1.Mavenproject1</exec.mainClass>
    </properties>
    <name>Proiect3</name>
</project>
```

În continuare **project.build.sourceEncoding**, specifică codificarea surselor (în acest caz, UTF-8), **maven.compiler.source și maven.compiler.target** indică versiunea limbajului Java utilizată pentru compilare (Java 20) și **exec.mainClass**: definiește clasa principale care conține metoda main, necesară pentru rularea aplicației.

Fișierului **pom.xml** este important în *managementul dependențelor*, deoarece Maven gestionează automat descărcarea și includerea bibliotecilor externe, cum este driverul mssql-jdbc, în *automatizarea proceselor* pentru că include configurări pentru compilare, rulare și generarea fișierelor livrabile (JAR) și pentru *portabilitate*, permițând altor dezvoltatori să cloneze proiectul și să îl configureze rapid doar cu fișierul pom.xml.

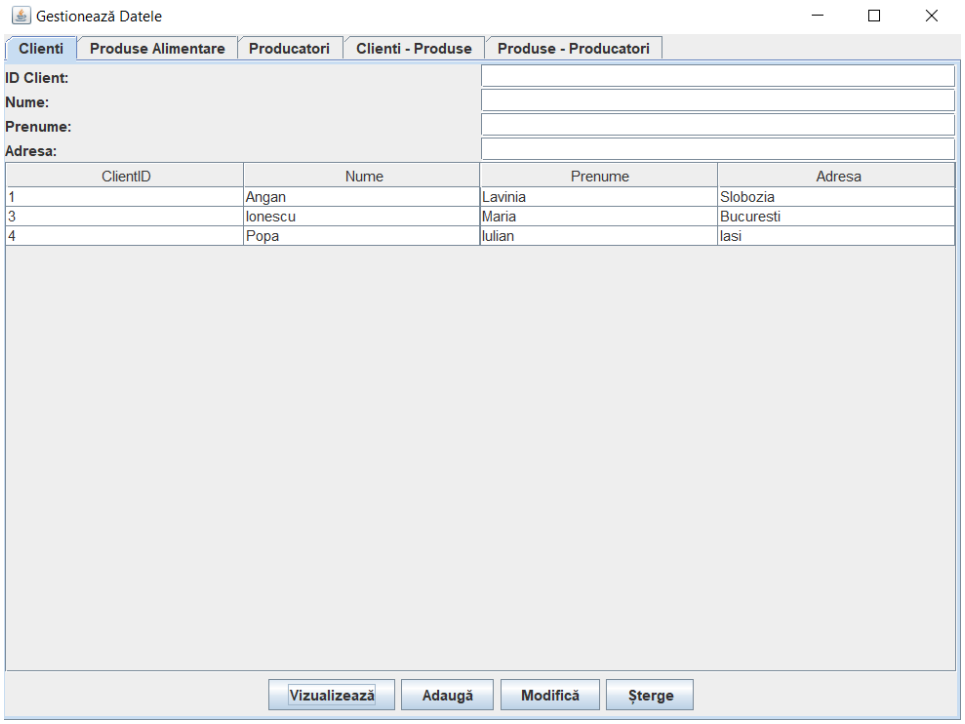
Fișierul DatabaseConnection asigură o metodă centralizată pentru conectarea aplicației Java la baza de date Microsoft SQL Server, iar conexiunea este gestionată prin intermediul clasei DriverManager din biblioteca standard Java.

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseConnection {
    private static final String URL = "jdbc:sqlserver://sap3.database.windows.net:1433;databaseName=GestionareProduse;
    private static final String USER = "lavinia";
    private static final String PASSWORD = "Marilena20";

    public static Connection getConnection() {
        try {
            // Load driver (optional for newer Java versions)
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");


            // Return the connection
            return DriverManager.getConnection(URL, USER, PASSWORD);
        } catch (SQLException | ClassNotFoundException e) {
            throw new RuntimeException("Nu s-a putut stabili conexiunea cu baza de date!", e);
        }
    }
}
```

Metoda getConnection încarcă driverul JDBC specific SQL Server (necesar doar pentru versiuni mai vechi de Java, dar păstrat aici pentru compatibilitate), creează o conexiune folosind datele furnizate în variabilele URL, USER și PASSWORD și genstionează erorile, adică dacă apare o problemă (e.g., date incorecte sau server indisponibil), metoda va arunca o excepție (RuntimeException), oferind un mesaj clar pentru depanare.

3. *Afișarea datelor*

Alegem drept exemplu tabela Clienti Pentru care vom analiza cum sunt efectuate operațiile CRUD.



Afișarea datelor este realizată prin metoda **viewClientsInTable**. Aceasta returnează un obiect JTable care conține datele din baza de date.

```
8   public class ClientOperations {
9
10      // Vizualizare date în JTable
11      public static JTable viewClientsInTable() {
12          DefaultTableModel model = new DefaultTableModel();
13          model.addColumn("ClientID");
14          model.addColumn("Nume");
15          model.addColumn("Prenume");
16          model.addColumn("Adresa");
17
18          try (Connection conn = DatabaseConnection.getConnection();
19               Statement stmt = conn.createStatement()) {
20
21              String query = "SELECT * FROM Clienti";
22              ResultSet rs = stmt.executeQuery(query);
23
24              while (rs.next()) {
25                  model.addRow(new Object[]{
26                          rs.getInt("ClientID"),
27                          rs.getString("Nume"),
28                          rs.getString("Prenume"),
29                          rs.getString("Adresa")
30                  });
31              }
32          } catch (Exception e) {
34              e.printStackTrace();
35          }
36
37          return new JTable(model);
38      }
```

Începând cu linia 12 un model de tabel este inițializat și se adaugă coloanele care corespund datelor din tabela Clienti. La linia 18 se obține o conexiune la baza de date folosind metoda DatabaseConnection.getConnection() și se creează un obiect Statement pentru a executa interogări SQL.

La linia 21 se definește interogarea SQL pentru a selecta toate înregistrările din tabela Clienti, iar rezultatele sunt stocate într-un obiect ResultSet. Apoi datele din ResultSet sunt citite rând cu rând și adăugate în modelul tabelului (liniile 24-30) . Linia 36 realizează returnarea unui **JTable** cu modelul populat**.**

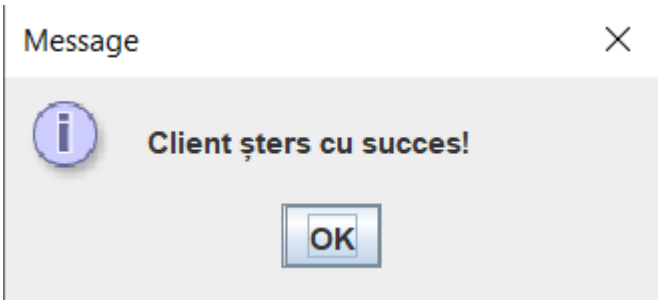
Pe scurt, **afișarea datelor** implică o interogare SELECT și popularea unui model de tabel cu rezultatele.

4. *Ștergerea datelor*

Ștergerea unui client este realizată prin **metoda deleteClient**.

```
74      // Ștergere client
75      public static void deleteClient(int clientID) {
76          try (Connection conn = DatabaseConnection.getConnection();
77               PreparedStatement pstmt = conn.prepareStatement(
78                       "DELETE FROM Clienti WHERE ClientID = ?")) {
79
80              pstmt.setInt(1, clientID);
81              pstmt.executeUpdate();
82
83              JOptionPane.showMessageDialog(null, "Client șters cu succes!");
84          } catch (Exception e) {
86              e.printStackTrace();
87          }
    }
```

La linia 77 se definește o interogare SQL care șterge o înregistrare din tabela Clienti pe baza valorii cheii primare ClientID. Se înlocuiește simbolul **?** din interogare cu valoarea specifică a ClientID. Interogarea este executată la linia 81, iar rândul specificat este șters din baza de date. La final se oferă feedback către utilizator prin mesajul **"Client șters cu succes!"**.



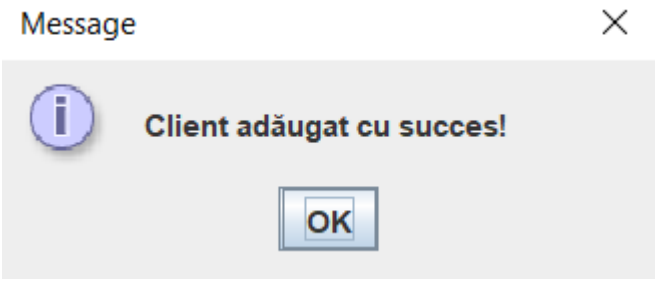
În concluzie, **ștergerea datelor** utilizează o interogare DELETE parametrizată pentru a elimina un client pe baza cheii primare (ClientID).

5. *Adaugarea datelor*

Adăugarea unui nou client este realizată prin metoda **addClient.**

```
38
39      // Adăugare client
40      public static void addClient(String nume, String prenume, String adresa) {
41          try (Connection conn = DatabaseConnection.getConnection();
42               PreparedStatement pstmt = conn.prepareStatement(
43                       "INSERT INTO Clienti (Nume, Prenume, Adresa) VALUES (?, ?, ?)")) {
44
45              pstmt.setString(1, nume);
46              pstmt.setString(2, prenume);
47              pstmt.setString(3, adresa);
48              pstmt.executeUpdate();
49
50              JOptionPane.showMessageDialog(null, "Client adăugat cu succes!");
51          } catch (Exception e) {
                e.printStackTrace();
53          }
```

La linia 42 se definește o interogare SQL care adaugă un nou rând în tabela Clienti. Coloanele *Nume, Prenume și Adresa* primesc valorile specificate. Apoi valorile parametrilor sunt furnizate prin argumentele metodei. Se execută interogarea și se afișează mesajul de feedback **"Client adăugat cu succes!"** .



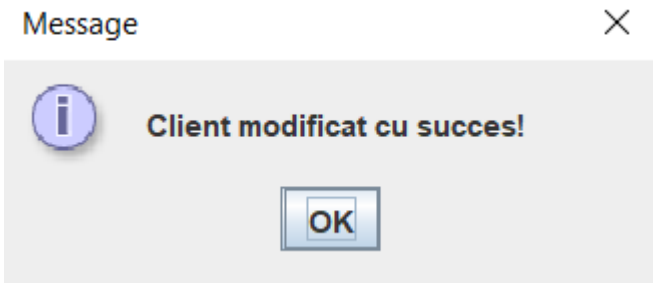
Așadar **adăugarea datelor** folosește o interogare INSERT parametrizată pentru a introduce un nou rând în tabel.

6. *Modificarea datelor*

Modificarea unui client existent este realizată prin metoda **updateClient.**

```
56      // Modificare client
57      public static void updateClient(int clientID, String nume, String prenume, String adresa) {
58          try (Connection conn = DatabaseConnection.getConnection();
59               PreparedStatement pstmt = conn.prepareStatement(
60                       "UPDATE Clienti SET Nume = ?, Prenume = ?, Adresa = ? WHERE ClientID = ?")) {
61
62              pstmt.setString(1, nume);
63              pstmt.setString(2, prenume);
64              pstmt.setString(3, adresa);
65              pstmt.setInt(4, clientID);
66              pstmt.executeUpdate();
67
68              JOptionPane.showMessageDialog(null, "Client modificat cu succes!");
69          } catch (Exception e) {
              e.printStackTrace();
71          }
72      }
```

Se definește o interogare SQL pentru a actualiza datele unui client pe baza valorilor furnizate. Rândul este identificat prin ClientID, apoi fiecare parametru din interogare este înlocuit cu valoarea corespunzătoare din argumentele metodei. Interogarea este executată, iar rândul specificat este actualizat în baza de date și se afișează mesajul de feedback "Client adăugat cu succes!" .



Pe scurt, **modificarea datelor** utilizează o interogare UPDATE parametrizată pentru a actualiza valorile existente, identificând rândul prin ClientID.

Concluzii

Proiectul realizat demonstrează o bună înțelegere a conceptelor fundamentale de dezvoltare software, în special utilizarea limbajului Java pentru implementarea operațiilor CRUD (Create, Read, Update, Delete) într-o aplicație desktop. Integrarea unei baze de date relaționale cu Microsoft SQL Server și gestionarea acesteia prin intermediul JDBC arată o aplicare practică a principiilor de manipulare a datelor, utilizând conexiuni centralizate și interogări parametrizate pentru a asigura securitatea și eficiența. De asemenea, separarea funcționalităților în clase distincte și utilizarea unui framework de build precum Maven contribuie la lizibilitatea și modularitatea proiectului, facilitând mentenanța și extinderea ulterioară.

Prin implementarea unei interfețe grafice care afișează datele într-un *JTable* conectat direct la baza de date, proiectul adaugă un nivel de interacțiune prietenos pentru utilizator, combinând gestionarea datelor cu vizualizarea acestora într-un mod intuitiv. Soluția respectă bunele practici de codare, utilizând blocuri *try-with-resources* pentru a evita scurgerile de resurse și implementând conexiunea cu baza de date într-un mod reutilizabil. În plus, structura modulară a proiectului și claritatea codului indică o gândire orientată spre extindere, permițând adăugarea ușoară de noi funcționalități, cum ar fi rapoarte detaliate sau căutări avansate.

În concluzie, proiectul reprezintă o aplicație practică și bine fundamentată care evidențiază competențele dezvoltatorului în utilizarea tehnologiilor moderne pentru gestionarea datelor. Această realizare oferă o bază solidă pentru explorarea unor tehnologii mai avansate, cum ar fi framework-urile pentru aplicații web (Spring, Hibernate) sau integrarea cu servicii RESTful, demonstrând totodată o pregătire pentru aplicații reale și complexe.

Bibliografie

1. https://en.wikipedia.org/wiki/Microsoft_SQL_Server
2. https://www.techtarget.com/searchdatamanagement/definition/SQL-Server
3. https://learn.microsoft.com/en-us/sql/sql-server/what-is-sql-server?view=sql-server-ver16
4. https://www.javatpoint.com/java-swing
5. https://www.geeksforgeeks.org/introduction-to-java-swing/
6. https://en.wikipedia.org/wiki/Swing_(Java)
7. https://netbeans.apache.org/tutorial/main/kb/docs/java/quickstart-gui/
8. https://openai.com/blog/chatgpt/
9. https://www.youtube.com/