

# **FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE**

---

# **Proiectarea unei aplicații de gestionare a unei reprezentanțe auto – AutoManager (OOP)**

Realizat de:

- Gavriș Alexandru-Cătălin

Coordonator de proiect:

- Prof. Timiș Iulia

---

Grupă:30222, CTI

Anul universitar: 2025 - 2026

## Cuprins:

<b>1. Introducere</b> .....	4
1.1 Contextul aplicației	
1.2 Scopul proiectului	
1.3 Domeniul aplicației (reprezentanță auto)	
<b>2. Funcționalități propuse</b> .....	4
<b>3. Funcționalități implementate</b> .....	5
<b>4. Cerințe funcționale</b> .....	5
4.1 Gestionarea clienților	
4.2 Gestionarea mașinilor	
4.3 Carte service și revizii	
4.4 Gestionarea angajaților	
4.5 Gestionarea vânzărilor	
4.6 Autentificare și roluri	
<b>5. Cerințe nefuncționale</b> .....	7
<b>6. Tehnologii utilizate</b> .....	7
<b>7. Arhitectura aplicației</b> .....	8
7.1 Model	
7.2 DAO (Data Access Object)	
7.3 View (Interfața grafică)	
7.4 Controller	
<b>8. Structura bazei de date</b> .....	10
8.1 Motivația folosirii unei baze relaționale	
8.2 Lista entităților	
8.3 Justificarea entităților	
8.4. Integritatea referențială	
<b>9. Autentificare și securitate</b> .....	12
9.1 Sistemul de autentificare	
9.2 Controlul accesului în funcție de rol	

## **Cuprins:**

<b>10. Tratarea erorilor.....</b>	<b>13</b>
10.1 Tratarea erorilor de bază de date	
10.2 Mesaje pentru utilizator	
<b>11. Interfața utilizator.....</b>	<b>14</b>
<b>12. Instrucțiuni de rulare a aplicației .....</b>	<b>16</b>
12.1 Cerințe software	
12.2 Configurarea bazei de date	
12.3 Configurarea conexiunii JDBC	
12.4 Rularea aplicației	
12.5 Pornirea aplicației	
<b>13. Procesul de implementare și raționament .....</b>	<b>21</b>
13.1 Păstrarea integrității datelor	
13.2 Alegerea arhitecturii MVC	
13.3 Utilizarea DAO pentru accesul la date	
13.4 Controlul accesului pe bază de rol	
<b>14. Testare.....</b>	<b>23</b>
<b>15. Limitări.....</b>	<b>23</b>
<b>16. Posibile îmbunătățiri.....</b>	<b>24</b>
<b>17. Concluzii finale.....</b>	<b>24</b>
<b>18. Bibliografie.....</b>	<b>24</b>

---

## **1. Introducere**

### **1.1 Contextul aplicației**

Aplicația AutoManager are ca scop simularea funcționării unei reprezentanțe auto moderne. Într-un context real, o reprezentanță auto trebuie să gestioneze simultan informații despre clienți, mașini, angajați, revizii, cărți service și vânzări, toate aceste date fiind strâns corelate.

### **1.2 Scopul proiectului**

Scopul proiectului este dezvoltarea unei aplicații desktop robuste, cu interfață grafică, care să permită administrarea completă a operațiunilor unei reprezentanțe auto, respectând principiile programării orientate pe obiecte și utilizând o bază de date relațională.(consistența datelor, integritatea relațiilor dintre entități, controlul accesului utilizatorilor, ușurința în utilizare)

### **1.3 Domeniul aplicației**

Aplicația este destinată utilizării interne într-o reprezentanță auto și este folosită de două tipuri de utilizatori:

- administratori
- angajați

---

## **2. Funcționalități propuse**

La începutul dezvoltării aplicației AutoManager, au fost propuse următoarele funcționalități principale:

- Gestionarea clientilor (adăugare, modificare, ștergere, afișare)
- Gestionarea mașinilor asociate clientilor

- Evidența cărții de service și a reviziilor
  - Validarea kilometrelor la revizii
  - Gestionarea angajaților
  - Înregistrarea vânzărilor între clienți, mașini și angajați
  - Autentificare utilizatori și control pe bază de rol
  - Asigurarea integrității datelor prin constrângeri la nivel de bază de date
- 

### **3. Funcționalități implementate**

Toate funcționalitățile propuse inițial au fost implementate cu succes.  
Aplicația oferă:

- operații CRUD complete pentru entitățile Client, Mașină, Angajat, Revizie și Vanzare
- relații funcționale între entități (ex: o vânzare leagă un client, o mașină și un angajat)
- validări la nivel de interfață și bază de date
- restricții de ștergere pentru datele implicate în vânzări
- autentificare și control de acces pe bază de rol

Funcționalitățile au fost testate manual și funcționează conform cerințelor.

---

### **4. Cerințe funcționale**

Aplicația trebuie să permită:

#### **4.1 Gestionarea clienților**

- adăugarea unui client nou
- modificarea datelor unui client existent

- ștergerea unui client (doar dacă nu există entități dependente)
- afișarea listei complete de clienți

#### **4.2 Gestionarea mașinilor**

- asocierea unei mașini unui client
- actualizarea informațiilor despre mașină
- afișarea mașinilor unui client selectat
- restricționarea ștergerii mașinilor deja vândute

#### **4.3 Carte service și revizii**

- creare carte service pentru o mașină
- adăugare revizii succesive
- validarea kilometrajului
- calcul automat al costului total
- evidențierea ultimei revizii

#### **4.4 Gestionarea angajaților**

- adăugare, modificare, ștergere angajați
- afișare listă angajați

#### **4.5 Gestionarea vânzărilor**

- înregistrarea unei vânzări
- asocierea client – mașină – angajat
- păstrarea istoricului vânzărilor
- interzicerea ștergerii entităților implicate într-o vânzare

#### **4.6 Autentificare și roluri**

- autentificare cu user și parolă
- roluri diferite: ADMIN / ANGAJAT
- restricții funcționale în funcție de rol

---

## 5. Cerințe nefuncționale

- aplicația trebuie să fie stabilă
  - să trateze erorile de bază
  - să ofere feedback utilizatorului
  - să aibă o interfață intuitivă
  - să respecte principiile OOP
- 

## 6. Tehnologii utilizate:

Componentă	Tehnologie
Limbaj	Java 17
UI	JavaFX
BD	SQL Server
Acces date	JDBC
Build	Maven
Arhitectură	MVC

---

## 7. Arhitectura aplicației

Aplicația este construită pe baza arhitecturii **Model–View–Controller (MVC)**.

### 7.1 Model

Conține clasele care reprezintă entitățile din baza de date. Aceste clase sunt simple POJO-uri (Plain Old Java Objects), având:

- atribute private
- constructori
- metode getter/setter

### 7.2 DAO (Data Access Object)

Pentru fiecare entitate există o clasă DAO responsabilă de:

- executarea interogărilor SQL
- izolarea logicii de acces la date
- manipularea conexiunilor JDBC

Această abordare permite modificarea ușoară a bazei de date fără impact major asupra aplicației.

### 7.3 View (Interfața grafică)

Interfața este definită în fișiere .fxml, folosind JavaFX.

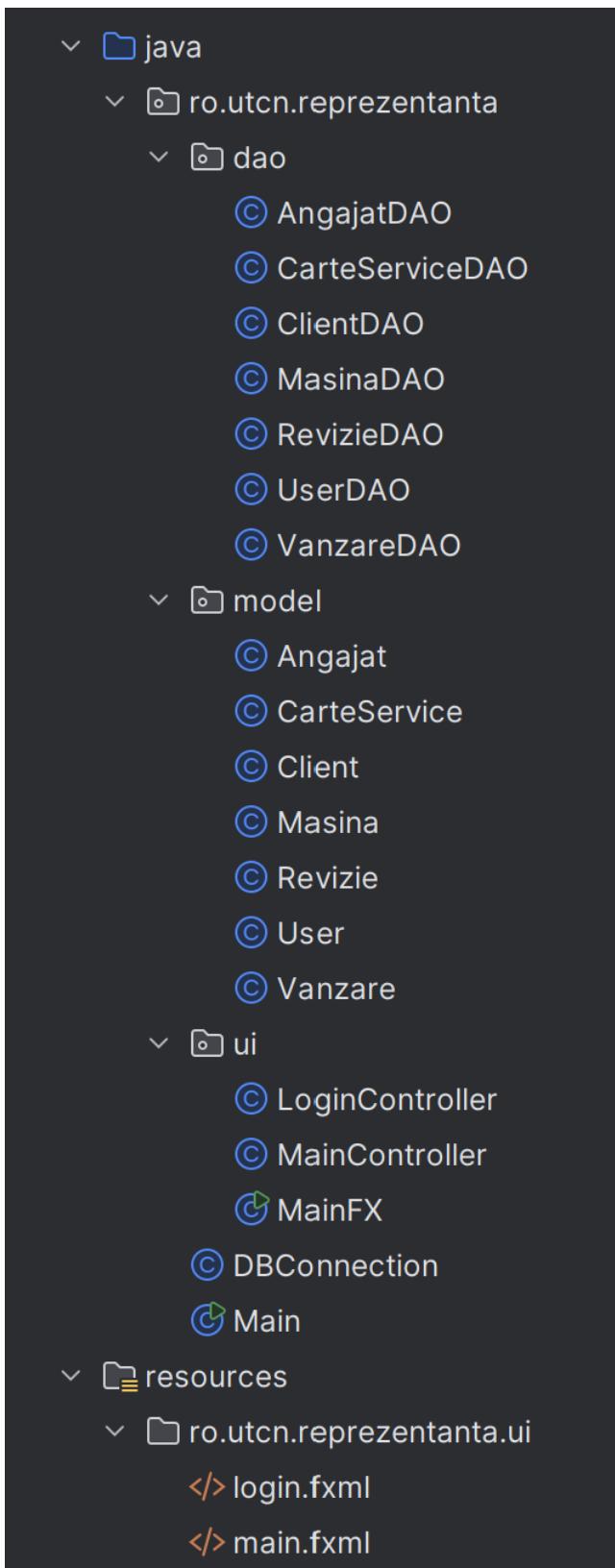
Aceasta este organizată în **Tab-uri**, fiecare corespunzând unui modul al aplicației.

### 7.4 Controller

Controller-ele gestionează:

- evenimentele UI
- validările
- apelurile către DAO

- actualizarea componentelor grafice



## **8. Structura bazei de date:**

### **8.1 Motivația folosirii unei baze relaționale**

O bază de date relațională permite:

- definirea clară a relațiilor
- aplicarea de constrângeri
- prevenirea inconsistentelor

### **8.2 Lista entităților**

- **Clienți:** persoane fizice care achiziționează mașini, cu date de contact și preferințe.
- **Angajați:** personalul reprezentanței (vânzători, mecanici, recepționeri, manageri), cu funcție, salariu și date de contact.
- **Mașini:** vehicule disponibile sau deja vândute, cu detalii despre marcă, model, an de fabricație și preț.
- **Vânzări:** tranzacțiile efectuate între clienți și reprezentanță, cu detalii despre mașină, angajat și preț final.
- **Revizii:** înregistrarea tuturor intervențiilor asupra mașinilor, cu date, cost și descriere a lucrărilor.
- **Carte Service:** evidența istoricului service al fiecărei mașini, inclusiv kilometraj inițial, ultima revizie și data expirării garanției.
- **Opțiuni:** echipamente suplimentare ale mașinilor (scaune încălzite, navigație, pachet sport etc.).
- **Mașina\_Opțiune:** tabel intermedian pentru relația N-N între mașini și opțiuni.

### **8.3 Justificarea entităților**

Fiecare entitate corespunde unui obiect real sau unei activități specifică unei reprezentanțe auto. Aceasta permite:

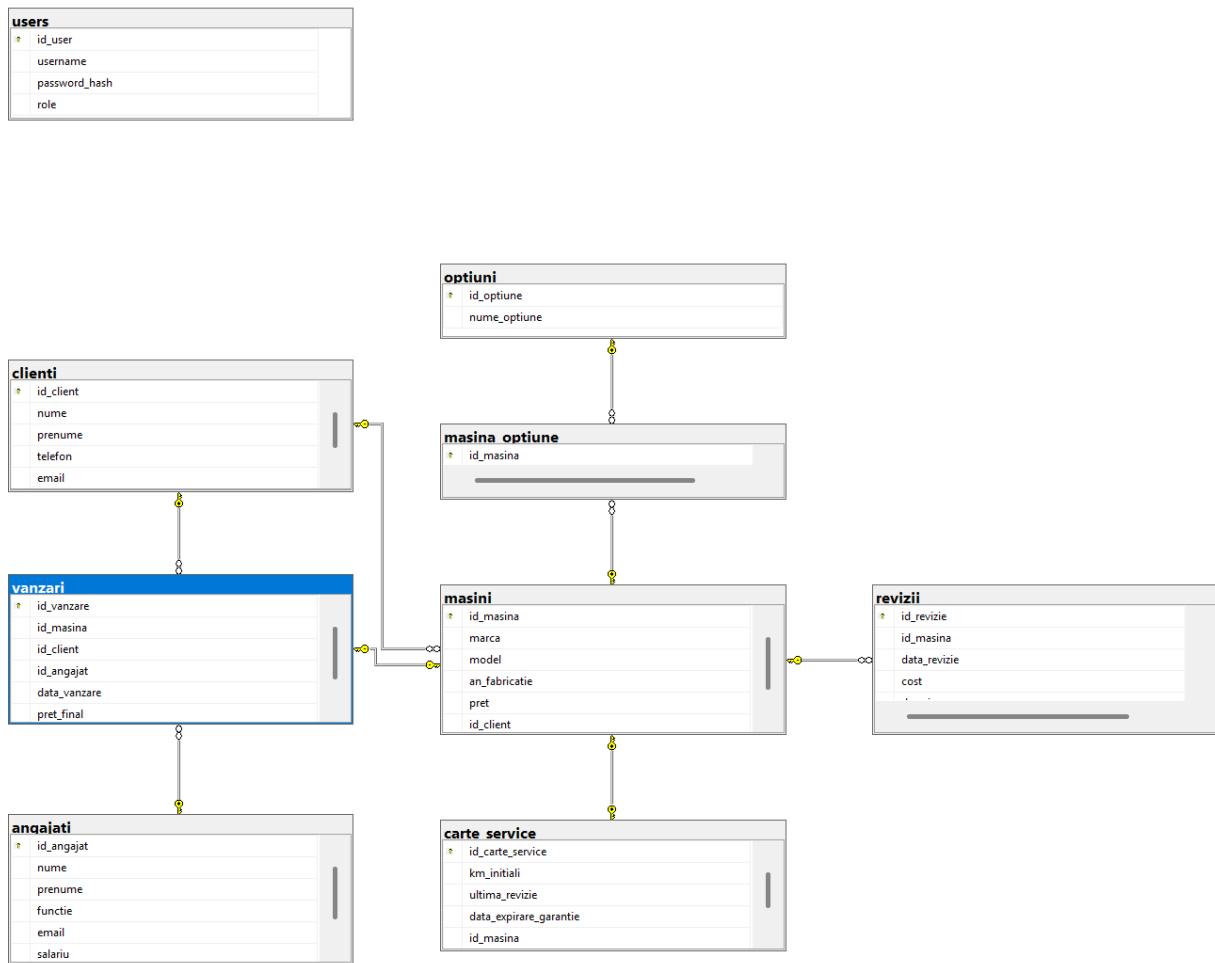
- Urmărirea corectă a relației între clienți și mașinile pe care le dețin.

- Corelarea angajaților cu vânzările și reviziile efectuate.
- Flexibilitate în alocarea opțiunilor către diverse mașini fără duplicarea datelor.
- Organizarea informațiilor într-un mod care reduce redundanța și facilitează raportarea și analiza.

## 8.4 Integritatea referențială

Aplicația se bazează pe **foreign keys**, ceea ce înseamnă:

- nu se poate șterge o mașină deja vândută
- nu se poate șterge un client cu mașini active
- nu se poate șterge o vânzare



## **9. Autentificare, securitate și controlul accesului pe bază de rol**

Aplicația AutoManager implementează un mecanism de autentificare menit să asigure accesul controlat la funcționalitățile aplicației și să prevină utilizarea neautorizată a datelor.

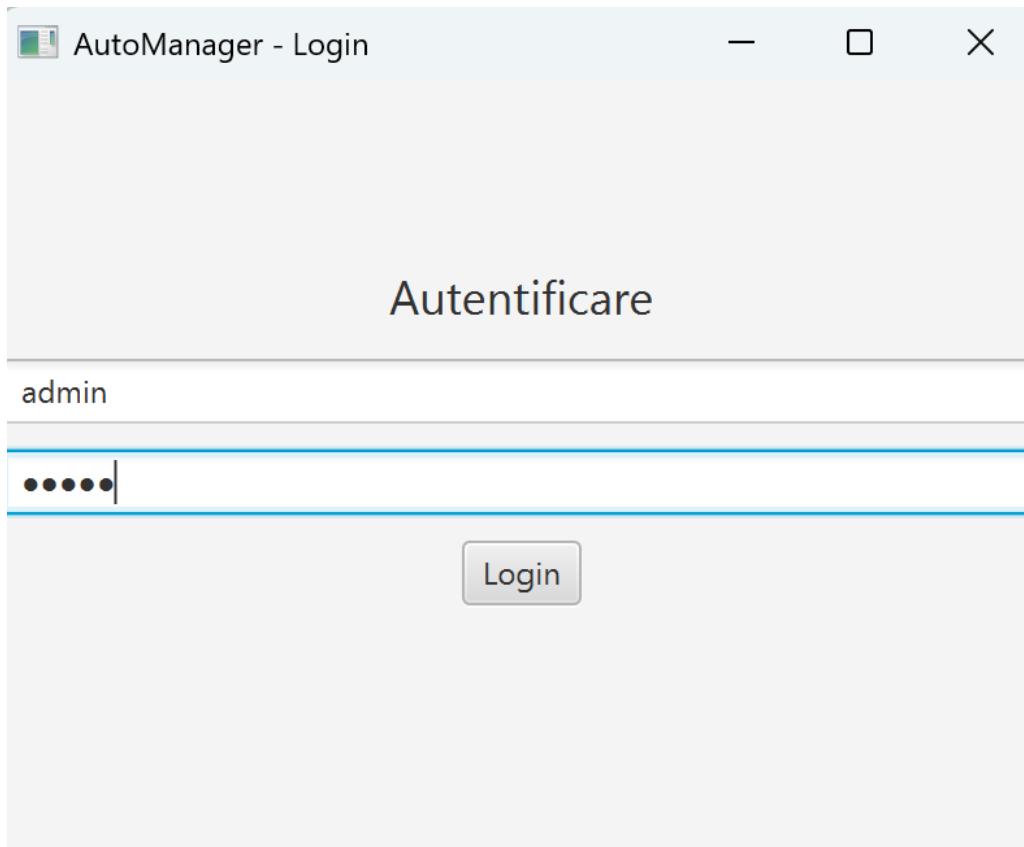
### **9.1 Sistemul de autentificare**

Autentificarea se realizează prin intermediul unei ferestre dedicate de login, în care utilizatorul trebuie să introducă:

- numele de utilizator (username)
- parola asociată

Datele utilizatorilor sunt stocate într-o tabelă dedicată din baza de date, denumită users.

La apăsarea butonului de autentificare, aplicația verifică existența utilizatorului și corectitudinea datelor introduse. În cazul în care autentificarea eșuează, utilizatorul primește un mesaj explicit de eroare.



## 9.2 Controlul accesului în funcție de rol

Controlul accesului este implementat la nivelul interfeței grafice și al logicii aplicației. Pentru anumite acțiuni critice, aplicația verifică rolul utilizatorului și permite sau restricționează operația respectivă.

Acțiune	ADMIN	ANGAJAT
Ștergere client	✓	x
Ștergere mașină	✓	x
Ștergere angajat	✓	x
Adăugare vânzare	✓	✓

Pentru acțiunile nepermise, aplicația nu execută operația și afișează un mesaj explicit de tipul „*Nu aveți permisiunea pentru această acțiune*”.

---

## 10. Tratarea erorilor

Aplicația AutoManager include mecanisme de bază pentru gestionarea situațiilor neprevăzute, atât la nivel de bază de date, cât și la nivelul interfeței grafice.

### 10.1 Tratarea erorilor de bază de date

Toate operațiile de acces la baza de date sunt încapsulate în blocuri try-catch. Acest lucru permite:

- capturarea exceptiilor SQL
- prevenirea blocării aplicației
- afișarea unor mesaje clare pentru utilizator

Exemple de situații tratate:

- încercarea de a șterge o entitate implicată într-o vânzare
- introducerea unor date incompatibile cu tipurile definite în baza de date

- Încălcarea constrângerilor de integritate referențială

## 10.2 Mesaje pentru utilizator

În cazul apariției unei erori, aplicația afișează mesaje explice, astfel încât utilizatorul să înțeleagă cauza problemei. Nu sunt afișate mesaje tehnice sau stack trace-uri în interfața finală, ci doar explicații relevante pentru utilizatorul final.

---

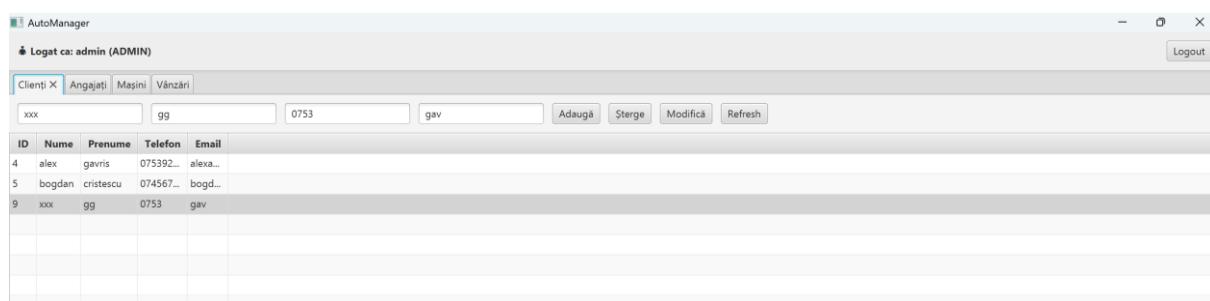
## 11. Interfața utilizator

Interfața grafică a aplicației a fost realizată folosind JavaFX și este structurată într-un mod modular, intuitiv și ușor de utilizat.

Aplicația este organizată în tab-uri, fiecare tab corespunzând unui modul distinct:

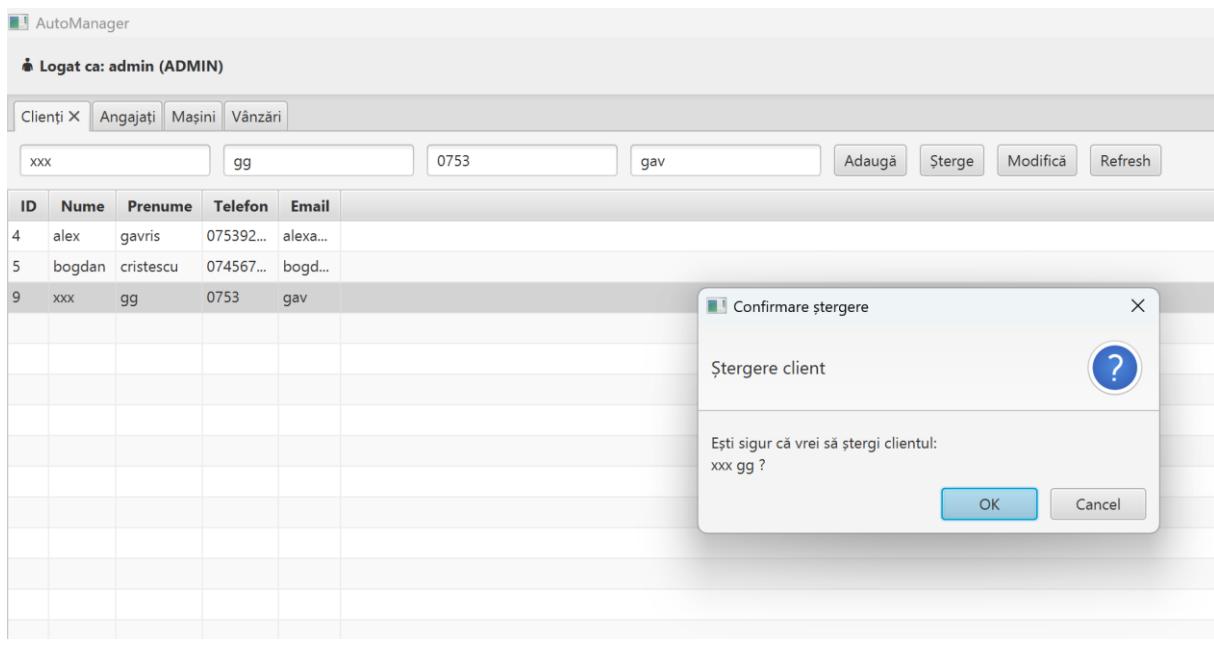
- clienți
- angajați
- mașini
- carte service și revizii
- vânzări

Layout-ul este conceput astfel încât utilizatorul să poată naviga rapid între funcționalități, fără a fi necesare cunoștințe tehnice avansate.



ID	Data	Client	Mașină	Angajat	Pret	
1	2025-12-16	alex gavris	bmw seria 5 (2013)	pop antonio	90000.0	
2	2025-12-16	alex gavris	bmw x5 (2015)	pop antonio	500.0	
3	2025-12-16	alex gavris	bmw e93 (2009)	pop antonio	60000.0	
4	2025-12-16	bogdan cristescu	bmw seria 3 (2010)	pop antonio	90000.0	
5	2025-12-16	bogdan cristescu	bmw m3 (2000)	pop antonio	900000.0	

A screenshot of the AutoManager application interface. At the top, there's a menu bar with 'AutoManager' and a user status 'Logat ca: angajat (ANGAJAT)'. On the right, there's a 'Logout' button. Below the menu is a navigation bar with tabs: 'Clienți', 'Angajații', 'Mașini', and 'Vânzări X'. The 'Vânzări' tab is currently selected. A modal dialog box titled 'Warning' is centered over the main content area. The dialog contains a yellow triangle icon with an exclamation mark, the word 'Warning' in bold, and the message 'Nu ai permisiuni pentru vânzări!'. In the bottom right corner of the dialog is an 'OK' button. The main table area shows a list of vehicle sales with columns for ID, Date, Client, and other details. The last row in the table is partially visible.



## 12. Instrucțiuni de rulare a aplicației

Această secțiune descrie pașii necesari pentru instalarea, configurarea și rularea aplicației **AutoManager** într-un mediu de dezvoltare standard.

### 12.1 Cerințe software

Pentru rularea aplicației sunt necesare următoarele componente software:

- **Java Development Kit (JDK) 17** sau o versiune mai recentă  
Aplicația este dezvoltată folosind Java 17, iar rularea cu versiuni mai vechi nu este garantată.
- **Apache Maven**  
Maven este utilizat pentru gestionarea dependențelor și pentru rularea aplicației JavaFX.
- **Microsoft SQL Server**  
Baza de date relațională utilizată pentru stocarea persistentă a datelor aplicației.
- **IntelliJ IDEA** (recomandat, dar optional)  
IDE-ul facilitează rularea, depanarea și analiza structurii proiectului.

## **12.2 Configurarea bazei de date**

Pentru ca aplicația să funcționeze corect, este necesară crearea bazei de date înainte de rularea aplicației.

Pașii sunt următorii:

1. Se pornește Microsoft SQL Server.
2. Se creează o bază de date nouă (ex: gestionare\_reprezentanta\_auto).
3. Se rulează scriptul SQL furnizat împreună cu proiectul.

```
CREATE TABLE clienti (
    id_client INT IDENTITY(1,1) PRIMARY KEY,
    nume VARCHAR(50) NOT NULL,
    prenume VARCHAR(50) NOT NULL,
    telefon VARCHAR(30),
    email VARCHAR(50)
);
```

```
CREATE TABLE angajati (
    id_angajat INT IDENTITY(1,1) PRIMARY KEY,
    nume VARCHAR(50),
    prenume VARCHAR(50),
    functie VARCHAR(50),
    email VARCHAR(50),
    salariu DECIMAL(10,2)
);
```

```
CREATE TABLE masini (
```

```
    id_masina INT IDENTITY(1,1) PRIMARY KEY,  
    marca VARCHAR(30),  
    model VARCHAR(30),  
    an_fabricatie INT,  
    pret DECIMAL(10,2),  
    id_client INT,  
    CONSTRAINT FK_masini_client  
        FOREIGN KEY (id_client)  
        REFERENCES clienti(id_client)  
);
```

```
CREATE TABLE carte_service (  
    id_carte_service INT IDENTITY(1,1) PRIMARY KEY,  
    km_initiali INT,  
    ultima_revizie DATE,  
    data_expirare_garantie DATE,  
    id_masina INT NOT NULL UNIQUE,  
    CONSTRAINT FK_carte_masina  
        FOREIGN KEY (id_masina)  
        REFERENCES masini(id_masina)  
);
```

```
CREATE TABLE revizii (  
    id_revizie INT IDENTITY(1,1) PRIMARY KEY,  
    id_masina INT NOT NULL,  
    data_revizie DATE,
```

```
km INT NOT NULL,  
descriere VARCHAR(200),  
cost DECIMAL(10,2),  
CONSTRAINT FK_revizii_masina  
    FOREIGN KEY (id_masina)  
    REFERENCES masini(id_masina)  
);
```

```
CREATE TABLE optiuni (  
    id_optiune INT IDENTITY(1,1) PRIMARY KEY,  
    nume_optiune VARCHAR(100)  
);
```

```
CREATE TABLE masina_optiune (  
    id_masina INT NOT NULL,  
    id_optiune INT NOT NULL,  
    PRIMARY KEY (id_masina, id_optiune),  
    CONSTRAINT FK_masina_optiune_masina  
        FOREIGN KEY (id_masina)  
        REFERENCES masini(id_masina),  
    CONSTRAINT FK_masina_optiune_optiune  
        FOREIGN KEY (id_optiune)  
        REFERENCES optiuni(id_optiune)  
);
```

```
CREATE TABLE users (
```

```
    id_user INT IDENTITY(1,1) PRIMARY KEY,  
    username VARCHAR(50) NOT NULL UNIQUE,  
    password_hash VARCHAR(255) NOT NULL,  
    role VARCHAR(20) NOT NULL  
);
```

```
CREATE TABLE vanzari (  
    id_vanzare INT IDENTITY(1,1) PRIMARY KEY,  
    id_masina INT NOT NULL UNIQUE,  
    id_client INT NOT NULL,  
    id_angajat INT NOT NULL,  
    data_vanzare DATE,  
    pret_final DECIMAL(10,2),  
    CONSTRAINT FK_vanzari_masina  
        FOREIGN KEY (id_masina)  
        REFERENCES masini(id_masina),  
    CONSTRAINT FK_vanzari_client  
        FOREIGN KEY (id_client)  
        REFERENCES clienti(id_client),  
    CONSTRAINT FK_vanzari_angajat  
        FOREIGN KEY (id_angajat)  
        REFERENCES angajati(id_angajat)  
);
```

### 12.3 Configurarea conexiunii JDBC

Conexiunea la baza de date este realizată prin intermediul clasei DBConnection.

În această clasă trebuie configurate următoarele:

- URL-ul bazei de date
- utilizatorul SQL
- parola

Exemplu general (fără date sensibile):

```
jdbc:sqlserver://localhost:1433;databaseName=gestionare_reprezentanta  
_auto;
```

Această separare a logicii de conexiune permite modificarea rapidă a setărilor fără impact asupra restului aplicației.

## 12.4 Rularea aplicației

Rulare din linia de comandă: din directorul proiectului se execută comanda:

```
mvn javafx:run
```

## 12.5 Pornirea aplicației

La pornire, aplicația afișează **ecranul de autentificare**.

Utilizatorul trebuie să se autentifice folosind un cont existent din baza de date.

---

# 13. Procesul de implementare și raționament

Această secțiune prezintă principalele decizii de proiectare luate în dezvoltarea aplicației AutoManager, precum și motivațiile din spatele acestora.

## 13.1 Păstrarea integrității datelor

Una dintre cele mai importante decizii a fost **prioritizarea integrității datelor**.

Din acest motiv:

- nu este permisă ștergerea unei mașini implicate într-o vânzare
- nu este permisă ștergerea unui client care are mașini asociate
- nu este permisă ștergerea unei vânzări

Această abordare reflectă un scenariu real din aplicațiile comerciale, unde istoricul tranzacțiilor trebuie păstrat pentru:

- raportare
- responsabilitate juridică

### **13.2 Alegerea arhitecturii MVC**

Arhitectura **Model–View–Controller (MVC)** a fost aleasă pentru a asigura:

- separarea clară a responsabilităților
- mentenanță ușoară
- extensibilitate
- testabilitate crescută

#### **Beneficii concrete:**

- modificarea interfeței grafice fără a afecta logica
- schimbarea bazei de date cu impact minim
- cod mai clar și mai organizat

### **13.3 Utilizarea DAO pentru accesul la date**

Folosirea pattern-ului **DAO (Data Access Object)** permite:

- izolarea codului SQL
- reutilizarea logicii de acces la date
- evitarea dependențelor directe între UI și baza de date

Această decizie este esențială pentru aplicații scalabile și bine structurate.

## **13.4 Controlul accesului pe bază de rol**

Implementarea rolurilor ADMIN și ANGAJAT a fost realizată pentru a simula un sistem real de lucru într-o reprezentanță auto.

Decizia de a **dezactiva funcționalități**, nu de a le elimina complet, permite:

- reutilizarea aceleiași interfețe
  - claritate pentru utilizator
  - prevenirea erorilor accidentale
- 

## **14. Testare**

Testarea aplicației a fost realizată manual, prin scenarii controlate care simulează utilizarea reală a aplicației.

Au fost verificate următoarele aspecte:

- adăugarea și modificarea entităților
- restricțiile de ștergere impuse de baza de date
- validarea kilometrilor la revizii
- diferențele de comportament între rolurile ADMIN și ANGAJAT
- afișarea mesajelor de eroare

Testarea manuală a confirmat funcționarea corectă a aplicației în condițiile prevăzute.

---

## **15. Limitări**

Aplicația prezintă anumite limitări:

- nu există posibilitatea de anulare (rollback) a unei vânzări
- parolele sunt stocate necriptat
- aplicația este disponibilă exclusiv ca aplicație desktop
- nu există jurnal de activitate (audit log)

---

## **16. Posibile îmbunătățiri**

Pentru o versiune viitoare a aplicației, pot fi avute în vedere următoarele îmbunătățiri:

- criptarea parolelor folosind algoritmi standard
  - generarea de rapoarte PDF
  - implementarea unui audit log
  - arhivarea datelor vechi
  - suport pentru aplicație web sau mobilă
  - anumite filtre pentru sortare
- 

## **17. Concluzii finale**

Aplicația AutoManager reprezintă o soluție completă pentru gestionarea unei reprezentanțe auto, integrând într-un mod coerent concepte de programare orientată pe obiecte, arhitectura MVC și baze de date relaționale.

Proiectul demonstrează capacitatea de a analiza cerințe reale, de a proiecta o structură solidă de date și de a implementa funcționalități complexe într-o aplicație software coerentă și extensibilă.

---

## **18. Bibliografie / Resurse**

1. MySQL Documentation – <https://dev.mysql.com/doc/>
2. OpenJFX, JavaFX Documentation, - <https://openjfx.io/>
3. Oracle, Java SE 17 Documentation, - <https://docs.oracle.com/en/java/javase/17/>
4. ChatGPT (OpenAI) - <https://chat.openai.com/>