Problema 3

Dezvoltați o aplicație care poate fi utilizată într-un **lanț de magazine de parfumuri**. Aplicația va avea 3 tipuri de utilizatori: angajat al unui magazin de parfumuri, manager al lanțului de magazine de parfumuri și administrator.

Utilizatorii de tip **angajat** al unui magazin de parfumuri pot efectua următoarele operații după autentificare:

1. Vizualizarea listei tuturor parfumurilor dintr-un magazin selectat sortată după următoarele criterii: denumire și preț;
2. Filtrarea parfumurilor după următoarele criterii: producător, disponibilitate, preț;
3. Căutarea unui parfum după denumire;
4. Operații CRUD în ceea ce privește persistența parfumurilor din magazinul la care lucrează acel angajat.

Utilizatorii de tip **manager** al lanțului de magazine de parfumuri pot efectua următoarele operații după autentificare:

1. Vizualizarea listei tuturor parfumurilor dintr-un magazin selectat sortată după următoarele criterii: denumire și preț;
2. Filtrarea parfumurilor după următoarele criterii: producător, disponibilitate, preț;
3. Căutarea unui parfum după denumire;
4. Salvare liste cu situația parfumurilor din lanțul de parfumerii în mai multe formate: csv, json, xml, txt;
5. Vizualizarea unor statistici legate de produsele din lanțul de parfumerii utilizând grafice (structură radială, structură inelară, de tip coloană, etc.).

Utilizatorii de tip **administrator** pot efectua următoarele operații după autentificare:

1. Operații CRUD pentru informațiile legate de utilizatori;
2. Vizualizarea listei tuturor utilizatorilor și filtrarea acesteia după tipul utilizatorilor.

**Interfața grafică a aplicației va fi disponibilă în cel puțin 3 limbi de circulație internațională** (implicit limba română).

Pentru dezvoltarea aplicației s-a ales **Java** atât pentru partea de backend (Spring Boot), cât și pentru frontend (JavaFX), din următoarele motive:

* Java este un limbaj robust, portabil și orientat pe obiecte, potrivit pentru aplicații scalabile.
* Spring Boot oferă suport extins pentru dezvoltarea rapidă a serviciilor REST, fiind ideal pentru separarea logicii de business.
* JavaFX permite dezvoltarea unei interfețe grafice moderne, cu suport pentru arhitecturi clare (MVP).
* Există suport foarte bun pentru integrarea între cele două componente prin REST, menținând un grad înalt de modularitate.

**Etapa de analiză – descriere diagrame UML**

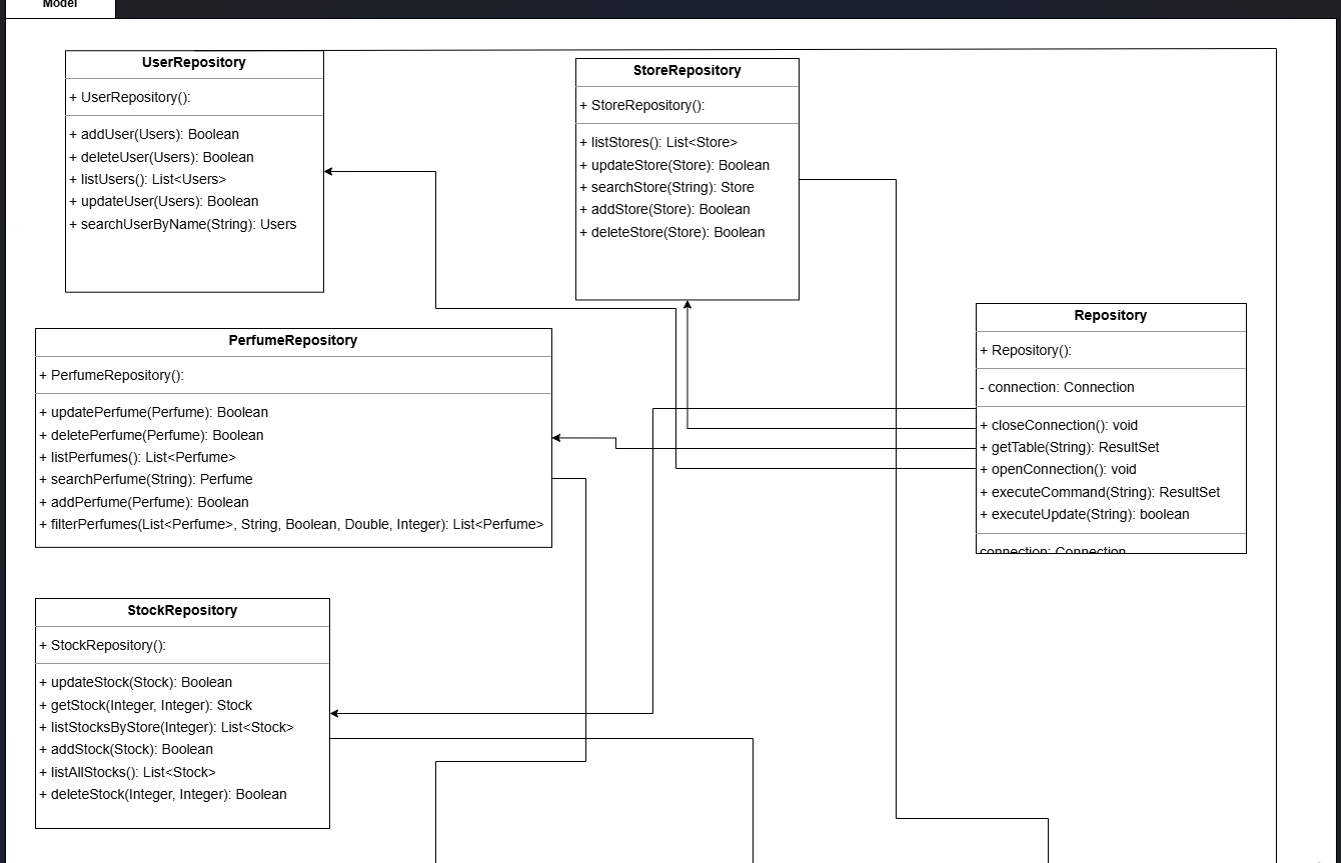
În această etapă s-au identificat entitățile principale și relațiile dintre acestea. Diagrama UML de clase aferentă backend-ului evidențiază următoarele:



* **Entități principale**:
  + Perfume – reprezintă un parfum, cu atribute precum brand, denumire, preț și disponibilitate.
  + User – reprezintă un utilizator al sistemului, având un nume, un rol și o legătură cu un anumit magazin.
  + Store – reprezintă un magazin fizic, care deține stocuri de parfumuri.
  + Stock – reprezintă relația dintre un Store și un Perfume, indicând cantitatea disponibilă.
* **Relații**:
  + Un Store poate avea mai multe Stock (relație one-to-many).
  + Un Stock este legat de un Perfume și un Store.
  + Un User este asociat cu un Store și are un Role.

Pentru partea de frontend (restserviceclient.drawio), analiza a identificat nevoia utilizării arhitecturii **Model-View-Presenter (MVP)**. Pentru fiecare entitate (ex. Perfume, User), sunt definite:

* un model (clasa de date),
* un view (interfața grafică, realizată cu FXML),
* un presenter (logica de legătură între view și model).



O imagine care conține text, captură de ecran, diagramă, Paralel

Conținutul generat de inteligența artificială poate fi incorect.

O imagine care conține text, captură de ecran, Paralel, document

Conținutul generat de inteligența artificială poate fi incorect.

**1. Perfume**

Clasa Perfume este o entitate JPA care reprezintă un parfum. Are atribute precum:

* id – identificator unic;
* name, brand – denumirea și brandul parfumului;
* price – prețul;
* availability – dacă este disponibil sau nu.

Este mapată în baza de date prin adnotări JPA (@Entity, @Id, etc.).

**2. User**

Reprezintă un utilizator al aplicației. Atributele includ:

* id, name, username, password;
* role – rolul utilizatorului (EMPLOYEE, MANAGER, ADMIN);
* store – legătura cu magazinul în care lucrează (dacă e cazul).

**3. Store**

Clasa Store definește un magazin din cadrul lanțului. Conține:

* id, name, location – detalii de identificare;
* listă de stocks – legătura cu parfumul prin intermediul clasei Stock.

**4. Stock**

Reprezintă relația dintre un parfum și un magazin, incluzând:

* store – magazinul;
* perfume – parfumul;
* quantity – cantitatea disponibilă.

Este o entitate compusă care gestionează stocurile individuale ale fiecărui magazin.

**5. PerfumeRepository**

Interfață care extinde JpaRepository<Perfume, Long>, oferind metode CRUD și posibilitatea de a adăuga metode personalizate (ex: căutare după brand sau disponibilitate).

**6. UserRepository**

Interfață JpaRepository<User, Long> ce permite gestionarea utilizatorilor. Poate include metode precum: findByRole, findByUsername.

**7. StockRepository**

Gestionează accesul la stocuri în baza de date. Extinde JpaRepository<Stock, Long>.

**8. StoreRepository**

Permite gestionarea magazinelor. Include metode standard și personalizate.

**9. PerfumeService**

Clasa care conține logica de business pentru parfumuri:

* filtrare după brand, disponibilitate, preț;
* adăugare, ștergere, modificare parfumuri.

Este apelată de controller și interacționează cu repository-ul.

**10. UserService**

Gestionează autentificarea, înregistrarea și filtrarea utilizatorilor.

**11. StockService**

Permite obținerea stocurilor, filtrare pe magazin sau parfum, precum și actualizarea cantității.

**12. StoreService**

Oferă acces la informațiile despre magazine, utilizată mai ales în contextul managerilor sau administratorilor.

**13. PerfumeController**

Expune endpoint-uri REST pentru operații legate de parfumuri (ex: /perfumes, /perfumes/filter). Apelează PerfumeService.

**14. UserController**

Responsabil de gestionarea operațiilor asupra utilizatorilor (ex: autentificare, CRUD, filtrare). Apelează UserService.

**15. StockController**

Interfață REST pentru accesul la date despre stocuri. Comunicarea se face cu StockService.

**16. StoreController**

Expune date despre magazine, necesare mai ales pentru funcționalitățile managerului.

**Frontend – JavaFX cu MVP**

**1. Perfume**

Clasa model utilizată pentru a reprezenta un parfum pe partea de client. Conține aceleași câmpuri ca pe backend.

**2. User**

Modelul utilizatorului logat, cu atribute pentru nume, rol, magazin etc.

**3. Store**

Modelul unui magazin din aplicație.

**4. Stock**

Reprezintă o intrare de stoc locală în aplicația client. Conține referințe către parfum și magazin, precum și cantitatea disponibilă.

**5. PerfumeRepository**

Clasa care comunică cu API-ul backend pentru operații asupra parfumurilor (GET, POST, PUT, DELETE, filtrări).

**6. UserRepository**

Comunică cu UserController de pe backend pentru autentificare, obținere listă utilizatori, modificări.

**7. StoreRepository**

Se ocupă de extragerea magazinelor din backend pentru afișare și selecție.

**8. StockRepository**

Folosit pentru obținerea stocurilor disponibile și afișarea acestora în interfața managerului.

**9. LoginPresenter**

Gestionează autentificarea. Primește inputul de la View, apelează UserRepository și redirecționează utilizatorul către interfața aferentă rolului.

**10. EmployeePresenter**

Logică pentru interfața angajatului. Se ocupă de:

* adăugarea parfumurilor;
* modificarea și ștergerea parfumurilor;
* aplicarea filtrelor.

**11. ManagerPresenter**

Gestionează:

* listarea parfumurilor din toate magazinele;
* filtrarea pe stocuri;
* exportul datelor în CSV, TXT sau JSON.

**12. AdminPresenter**

Se ocupă de:

* crearea, modificarea și ștergerea utilizatorilor;
* filtrarea utilizatorilor după rol;
* actualizarea UI-ului în funcție de operații.

**13. EmployeeViewController**

Preia evenimentele de UI și le transmite către EmployeePresenter. Nu conține logică, doar manipulare UI.

**14. ManagerViewController**

Controller-ul grafic pentru manager. Primește date de la ManagerPresenter și le afișează.

**15. AdminViewController**

View-ul pentru administrator. Apelează AdminPresenter pentru toate operațiile legate de utilizatori.

**16. LoginViewController**

Controlează interfața de login, trimițând datele către LoginPresenter.

O imagine care conține text, diagramă, Paralel, captură de ecran

Conținutul generat de inteligența artificială poate fi incorect.

O imagine care conține text, diagramă, Paralel, Plan

Conținutul generat de inteligența artificială poate fi incorect.

O imagine care conține text, captură de ecran, afișaj, diagramă

Conținutul generat de inteligența artificială poate fi incorect.

**1. Perfume**

Clasa model utilizată pentru a reprezenta un parfum pe partea de client. Conține aceleași câmpuri ca pe backend.

**2. User**

Modelul utilizatorului logat, cu atribute pentru nume, rol, magazin etc.

**3. Store**

Modelul unui magazin din aplicație.

**4. Stock**

Reprezintă o intrare de stoc locală în aplicația client. Conține referințe către parfum și magazin, precum și cantitatea disponibilă.

**5. PerfumeRepository**

Clasa care comunică cu API-ul backend pentru operații asupra parfumurilor (GET, POST, PUT, DELETE, filtrări).

**6. UserRepository**

Comunică cu UserController de pe backend pentru autentificare, obținere listă utilizatori, modificări.

**7. StoreRepository**

Se ocupă de extragerea magazinelor din backend pentru afișare și selecție.

**8. StockRepository**

Folosit pentru obținerea stocurilor disponibile și afișarea acestora în interfața managerului.

**9. LoginPresenter**

Gestionează autentificarea. Primește inputul de la View, apelează UserRepository și redirecționează utilizatorul către interfața aferentă rolului.

**10. EmployeePresenter**

Logică pentru interfața angajatului. Se ocupă de:

* adăugarea parfumurilor;
* modificarea și ștergerea parfumurilor;
* aplicarea filtrelor.

**11. ManagerPresenter**

Gestionează:

* listarea parfumurilor din toate magazinele;
* filtrarea pe stocuri;
* exportul datelor în CSV, TXT sau JSON.

**12. AdminPresenter**

Se ocupă de:

* crearea, modificarea și ștergerea utilizatorilor;
* filtrarea utilizatorilor după rol;
* actualizarea UI-ului în funcție de operații.

**13. EmployeeViewController**

Preia evenimentele de UI și le transmite către EmployeePresenter. Nu conține logică, doar manipulare UI.

**14. ManagerViewController**

Controller-ul grafic pentru manager. Primește date de la ManagerPresenter și le afișează.

**15. AdminViewController**

View-ul pentru administrator. Apelează AdminPresenter pentru toate operațiile legate de utilizatori.

**16. LoginViewController**

Controlează interfața de login, trimițând datele către LoginPresenter.