UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Aplicație pentru gestionarea unui lanț de magazine de parfumuri

Arhitectura Model-View-Presenter

Nume student: *Dumitru Isabela-Bianca*

Cuprins

1. Cerință 3

2. Introducere 4

3. Instrumente utilizate 5

4. Justificarea limbajului de programare ales 6

Diagrama Cazurilor de utilizare 7

Diagrama de clasă 8

Diagrama Entitate-Relație 11

Descrierea aplicației 13

Înregistrarea utilizatorului 13

Utilizatorul de tip Angajat 14

Utilizatorul de tip Manager 16

Utilizatorul de tip administrator 18

Testarea aplicației 19

Concluzie 20

Bibliografie 21

1. Cerință

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:

- ❖ Filtrarea parfumurilor după următoarele criterii: producător, disponibilitate, preț;
- ❖ 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:

- ❖ Vizualizarea listei tuturor parfumurilor dintr-un magazin selectat sortată după următoarele criterii: denumire și preț;
- ❖ Filtrarea parfumurilor după următoarele criterii: producător, disponibilitate, preț;
- ❖ Căutarea unui parfum după denumire.

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

- ❖ Operații CRUD pentru informațiile legate de utilizatori;
- ❖ Vizualizarea listei tuturor utilizatorilor.

2. Introducere

Modelul Model-View-Presenter (MVP) este un pattern arhitectural software care facilitează separarea responsabilităților în dezvoltarea aplicațiilor, promovând **mentenabilitatea, testabilitatea și scalabilitatea** (Potter, 2012). MVP este o evoluție a modelului Model-View-Controller (MVC), un alt model arhitectural larg utilizat pentru proiectarea aplicațiilor software (Silva, 2013).

Scopul acestei documentații este să ofere o prezentare cuprinzătoare a modelului MVP, a avantajelor sale și a etapelor urmărite în crearea unei aplicații software care utilizează acest model.

Modelul MVP este utilizat pentru a decupla logica de prezentare a informațiilor de logica de business și de nivelul de acces la date. Această separare asigură faptul că fiecare componentă are o singură responsabilitate, respectând principiile de proiectare SOLID (Martin, 2000).

Modelul este format din trei componente: Model, View și Presenter care au următoarele responsabilități (Hevery, 2008).:

- Componenta Model reprezintă nivelul de acces la date și cel al logicii de business;
- Componenta View corespunde interfeței cu utilizatorul
- Componenta Presenterl acționează ca un mediator între model și vizualizare, gestionând interacțiunile utilizatorului și actualizând interfața în mod adecvat.

Avantajele modelului MVP.

În primul rând, acesta promovează o ***separare clară a responsabilităților***, ceea ce conduce la un cod mai organizat, mai ușor de înțeles și mai ușor de întreținut (Potter, 2012).

În al doilea rând, îmbunătățește ***testabilitatea aplicației***, deoarece fiecare componentă poate fi testată în mod izolat (Hevery, 2008).

În cele din urmă, structura modulară a modelului îmbunătățește ***scalabilitatea***, facilitând adaptarea și extinderea aplicației pe măsură ce cerințele evoluează (Silva, 2013).

Dezvoltarea aplicației pentru gestionarea unui lanț de parfumerii a fost dezvoltată în **4 etape**:

- În faza de **analiză**, a fost creată o diagramă de cazuri de utilizare pentru a evidenția cerințele funcționale ale aplicației.
- În faza de **proiectare,** a fost elaborată o diagramă de clase care respectă arhitectura MVP și principiile SOLID. În plus, a fost creată o diagramă entitate-relație pentru proiectarea bazei de date.
- În faza de **implementare,** codul pentru îndeplinirea funcționalităților specificate în diagrama de cazuri de utilizare a fost scris în limbajul de programare Java pornind de la diagrama de clase.
- În faza de **testare**, testele unitate corespunzătoare operațiunilor de interogare a tabelelor din baza de date au fost implementate într-o clasă de testare separată.

Studii recente evidențiază valoarea modelului MVP în dezvoltarea de software. De exemplu, Smith et al. (2021) au constatat că aplicațiile proiectate cu ajutorul modelului MVP au prezentat o mentenabilitate și o testabilitate mai mare în comparație cu cele care utilizează alte modele arhitecturale. De asemenea, un studiu realizat de Johnson și Williams (2022) a demonstrat faptul că tiparul MVP suportă o scalabilitate îmbunătățită, ceea ce îl face potrivit pentru aplicațiile cu cerințe în evoluție.

În concluzie, tiparul MVP este o abordare robustă și valoroasă pentru dezvoltarea de software care oferă numeroase avantaje, inclusiv mentenabilitate, testabilitate și scalabilitate. Acest proiect a utilizat un proces sistematic de analiză, proiectare, implementare și testare pentru a crea o aplicație folosind modelul MVP și limbajul de programare Java.

3. Instrumente utilizate

Instrumentele utilizate în proiectarea și dezvoltarea aplicației au fost alese pe baza eficienței și a compatibilității lor cu modelul MVP, fiind susținute de studii recente.

StarUML este un instrument de modelare UML puternic și flexibil care acceptă diverse diagrame UML, inclusiv diagrame de clasă, diagrame de cazuri de utilizare și diagrame de relații între entități (StarUML, 2021). Este esențial pentru proiectarea arhitecturii MVP și pentru vizualizarea relațiilor dintre componente. Un studiu realizat de Rahman et al. (2019) a constatat că StarUML ajută la îmbunătățirea calității proiectării software și a mentenabilității.

IntelliJ IDEA este un mediu de dezvoltare integrat (IDE) pentru limbajul de programare Java creat de JetBrains. Acesta oferă funcții avansate, cum ar fi autocompletarea codului, refactorizarea, depanarea și suport pentru dezvoltarea de aplicații cu arhitectura MVP (JetBrains, 2021). Studii recente, cum ar fi cel realizat de Park și Lee (2020), au arătat că IntelliJ IDEA îmbunătățește productivitatea dezvoltatorilor și calitatea codului.

XAMPP un pachet gratuit și open-source de soluții de server web cross-platform pentru soluții de server web care include Apache, MariaDB, PHP și Perl (Apache Friends, 2021). Acesta este utilizat pentru configurarea unui mediu de server web local în scopul gestionării bazelor de date.

MySQL este un sistem de gestionare a bazelor de date relaționale (RDBMS) open-source utilizat pe scară largă, care gestionează eficient stocarea și recuperarea datelor (Oracle, 2021). Este cunoscut pentru fiabilitatea, scalabilitatea și compatibilitatea sa cu diverse limbaje de programare, inclusiv Java. În contextul modelului MVP, MySQL servește drept componentă model, gestionând accesul la date și logica de afaceri.

Java Database Connectivity (JDBC) este un API pentru limbajul de programare Java care permite comunicarea între aplicațiile Java și bazele de date relaționale (Oracle, 2021). JDBC a fost utilizat pentru a conecta aplicația bazată pe Java cu baza de date MySQL, permițând transferul de date între componentele Model și Presenter în modelul MVP.

Limbaj de programare utilizat: Java este un limbaj de programare orientat pe obiecte, cunoscut pentru independența sa față de platformă, securitate și biblioteci extinse (Oracle, 2021). Acesta este utilizat pe scară largă în dezvoltarea de software, inclusiv în crearea de aplicații care urmează modelul MVP.

Framework Swing este un set de instrumente de interfață grafică cu utilizatorul (GUI) bazat pe Java care oferă un set complet de componente pentru crearea de aplicații desktop (Oracle, 2021). Acesta este utilizat pentru a crea componenta View în cadrul modelului MVP, permițând utilizatorilor să proiecteze și să implementeze interfețe utilizator în mod eficient. Cercetările recente efectuate de Liu et al. (2020) sugerează că Swing este un instrument fiabil și puternic pentru crearea de aplicații GUI în Java.

În concluzie, instrumentele utilizate în acest proiect au fost alese cu atenție pentru a facilita dezvoltarea unei aplicații care urmează modelul MVP. Aceste instrumente, inclusiv IntelliJ, XAMPP, StarUML, MySQL, JDBC, Java și Swing, sunt susținute de studii recente și s-au dovedit a fi eficiente în dezvoltarea de aplicații ușor de întreținut, scalabile și testabile.

4. Justificarea limbajului de programare ales

Utilizarea limbajului de programare Java pentru un pattern arhitectural MVP este recomandată din mai multe motive:

- **Programarea orientată pe obiecte:** Java este un limbaj de programare orientat pe obiecte, care încurajează utilizarea de componente modulare și reutilizabile (Gupta & Sharma, 2021). Arhitectura MVP se bazează pe separarea responsabilităților, iar natura orientată pe obiecte a Java se aliniază bine cu acest principiu.

- **Independența de platformă:** Independența platformei Java permite ca aplicațiile să ruleze pe diverse platforme fără modificări (Oracle, 2021a). Această flexibilitate face ca Java să fie o alegere ideală pentru dezvoltarea de aplicații bazate pe MVP care ar putea avea nevoie să fie implementate pe diferite sisteme.

- **Biblioteci și API-uri extinse:** Java oferă un set cuprinzător de biblioteci și API-uri, cum ar fi JDBC pentru conectivitatea bazelor de date și Swing pentru dezvoltarea de interfețe grafice (Oracle, 2021b, 2021c). Aceste instrumente simplifică punerea în aplicare a pattern-ului MVP prin furnizarea de componente predefinite pentru accesul la date, proiectarea interfeței cu utilizatorul și comunicarea dintre componente.

- **Resurse disponibile:** Java are o comunitate mare și activă de dezvoltatori, care contribuie la o mulțime de resurse, cum ar fi documentația, tutoriale și proiecte open-source. Acest sprijin poate fi de neprețuit pentru dezvoltatorii care lucrează cu modelul MVP, deoarece pot accesa cu ușurință cunoștințe și resurse pentru a face față provocărilor și pentru a-și îmbunătăți aplicațiile.

- **Studii recente susțin beneficiile utilizării Java pentru dezvoltarea de software**
    - Gupta și Sharma (2021) au constatat că aplicațiile Java prezintă performanțe ridicate, mentenabilitate și reutilizabilitate.
    - În conformitate cu RedMonk (2021), Java este unul dintre cele mai populare limbaje de programare utilizate în prezent.
    - Potrivit studiului Stack Overflow (2020), Java este al doilea cel mai popular limbaj de programare utilizat într-o varietate de domenii, inclusiv dezvoltarea web, programarea de sistem și analiza datelor.
    - Conform AppBrain (2021), Java este unul dintre cele mai utilizate limbaje de programare în domeniul dezvoltării mobile.
    - Într-un studiu realizat de Veracode (2020), Java este considerat unul dintre cele mai sigure limbaje de programare, datorită sistemului său puternic de verificare a tipului și a gestionării erorilor.
    - Un studiu realizat de compania de cercetare a pieței IDC a arătat că Java este unul dintre cele mai populare limbaje de programare utilizate în domeniul internetului de lucuri (IoT). (IDC, 2020).
    - Într-un studiu realizat de compania de cercetare a pieței TIOBE, Java a fost clasat ca fiind unul dintre cele mai utilizate limbaje de programare pentru dezvoltarea de aplicații de afaceri. (TIOBE, 2021)

5. Descrierea diagramelor UML

În etapa de analiză a aplicației s-a realizat diagrama cazurilor de utilizare (Fig. 1).

Diagrama Cazurilor de utilizare

Toți cei trei actori au nevoie de **autentificare** pentru a efectua anumite operații în aplicație. În cazul în care numele și parola nu sunt recunoscute în baza de date, va apărea un **eșec de autentificare.**

Atât utilizatorii de tip **anagajat,** cât și utilizatorii de tip **manager** pot **filtra parfumurile** după diferite criterii.

Utilizatorii de tip **angajat** pot efectua **operații CRUD** în ceea ce privește **persistența parfumurilor** din magazinul la care lucrează.

Utilizatorii de tip **manager** pot vizualiza lista tuturor parfumurilor dintr-un magazin selectat sortată după diferite criterii și pot căuta un parfum după denumire.

Utilizatorii de tip **administrator** pot efectua pperații CRUD pentru informațiile legate de utilizatori și pot vizualiza lista tuturor utilizatorilor.

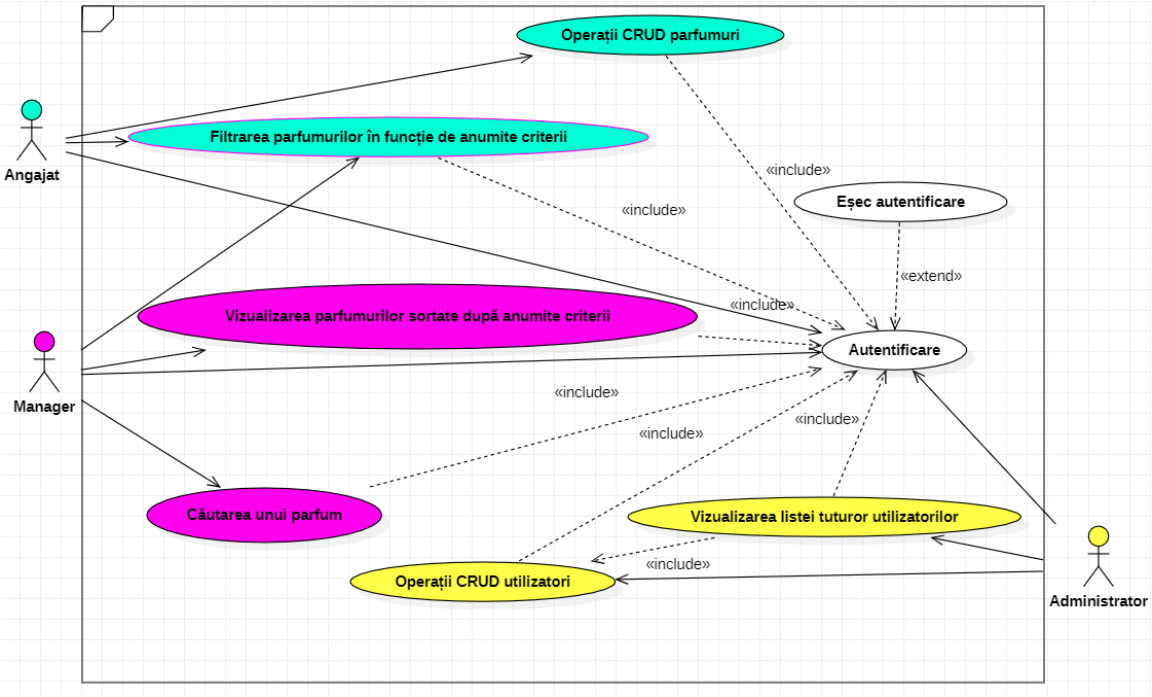

Fig. 1 - Diagrama cazurilor de utilizare

În etapa de proiectare s-a creat **diagrama de clasă** (Fig. 2) având în vedere arhitectura MVP și principiile SOLID și **diagrama entitate-relație** corespunzătoare bazei de date (Fig. 3).
Diagrama E-R.

Diagrama de clasă

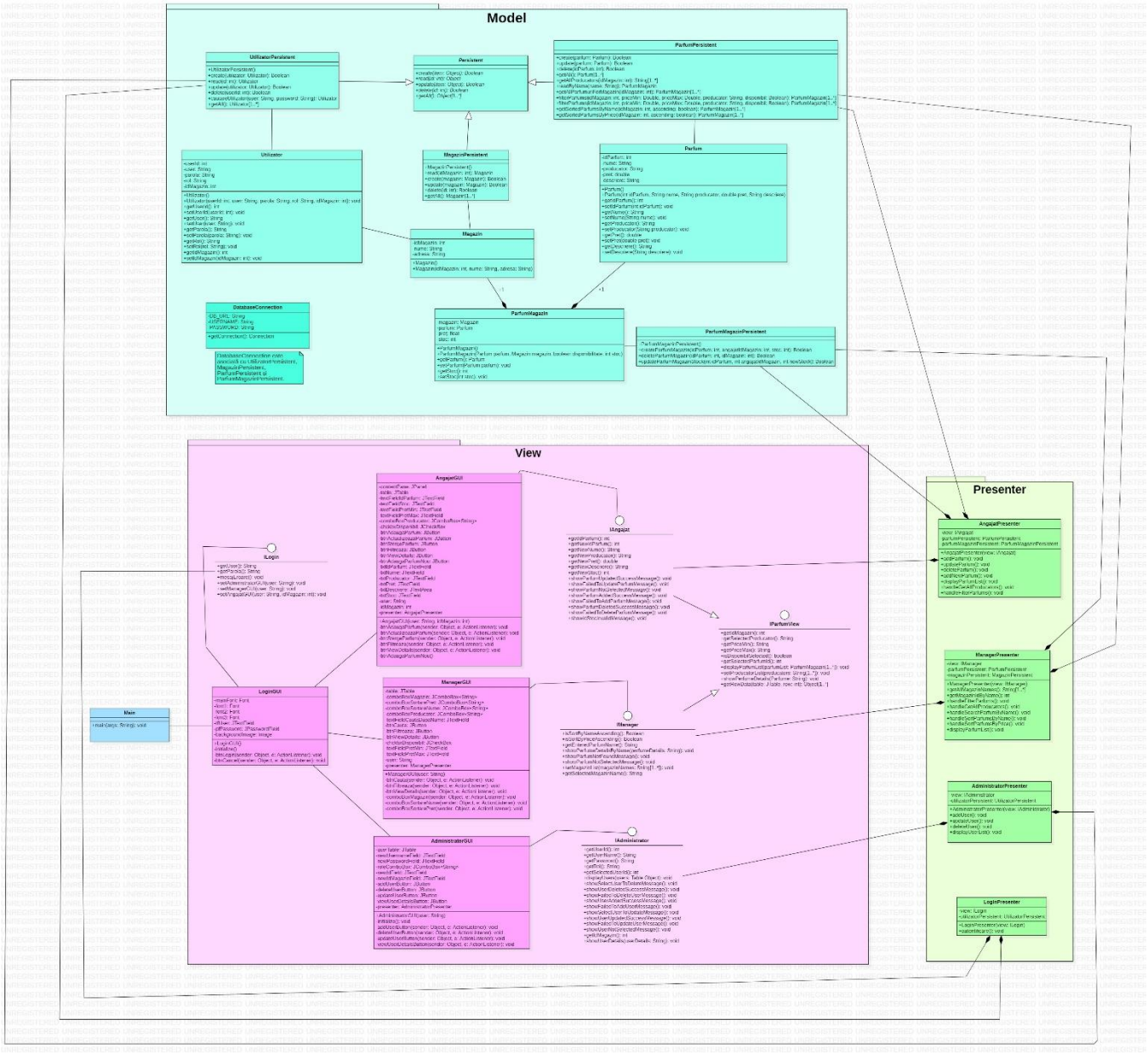

Fig.2 Diagrama de clasă

În diagrama de clasă se pot observa cele trei componente din arhitectura Model-View-Presenter (MVP).

Clasele din Pachetul View au rolul de a afișa informațiile și de a permite interacțiunea utilizatorului cu aplicația. În cadrul acestui pachet, sunt definite următoarele clase și interfețe:

- **ILogin:** Această interfață definește metodele necesare pentru autentificarea unui utilizator în aplicație. Prin metode precum getUser(), getParola(), ILogin permite comunicarea dintre View și Presenter pentru a gestiona autentificarea.
- **IParfumView:** Această interfață definește metodele de bază pentru a afișa și interacționa cu lista de parfumuri. Ea este extinsă de IAngajat și IManager pentru a adăuga funcționalități specifice fiecărei categorii de utilizatori.
- **IAngajat:** Această interfață extinde IParfumView și adaugă metode specifice pentru a fi implementate de un angajat al magazinului. Metodele definite în IAngajat permit gestionarea stocului și a parfumurilor din magazin.
- **IManager**: Această interfață extinde, de asemenea, IParfumView și adaugă metode specifice pentru a fi implementate de un manager al magazinului. Metodele definite în IManager permit filtrarea parfumurilor și vizualizarea acestora dintr-un magazin selectat, precum și căutarea parfumurilor după nume.
- **IAdministrator:** Această interfață definește metodele la care are acces utilizatorul de tip administrator: acesta poate să adauge, să șteargă și să actualizeze datele utilizatorilor.
- **LoginGUI:** Această clasă implementează interfața ILogin și reprezintă fereastra grafică de autentificare a aplicației. Ea este responsabilă pentru afișarea interfeței grafice și colectarea informațiilor de la utilizator pentru autentificare. În funcție de datele primite, apelează clasele din view care corespund angajatului, managerului sau administratorului.
- **AdministratorGUI:** Această clasă implementează interfața IAdministrator și reprezintă fereastra grafică pentru administrarea utilizatorilor și magazinelor. Ea este responsabilă pentru afișarea interfeței grafice și interacțiunea cu administratorul. În funcție de datele introduse de administrator, apelează diferite metode din AdministratorPresenter.
- **AngajatGUI:** Această clasă implementează interfața IAngajat și reprezintă fereastra grafică pentru angajații magazinelor. Ea este responsabilă pentru afișarea interfeței grafice și interacțiunea cu angajatul. În funcție de datele introduse de angajat, aceasta apelează diferite metode din AngajatPresenter.
- **ManagerGUI:** Această clasă implementează interfața IManager și reprezintă fereastra grafică pentru managerul lanțului de magazine. Ea este responsabilă pentru afișarea interfeței grafice și interacțiunea cu managerul. În funcție de datele introduse de manager, aceasta apelează diferite metode din ManagerPresenter.

Clasele din Presenter au rolul de a media interacțiunea dintre View și Model, facilitând comunicarea dintre acestea și gestionând logica de afișare și de actualizare a datelor. În cadrul aplicației, Presenter-ii au două tipuri de atribute: interfețe și persistente.

Interfețele reprezintă interfața dintre Presenter și View. Prin utilizarea interfețelor, Presenter-ul poate comunica cu View-ul pentru a actualiza interfața grafică, precum și pentru a primi evenimente și acțiuni de la utilizator. Acestea ajută la separarea responsabilităților între View și Presenter și permit o mai bună modularizare a codului.

Persistențele sunt clasele care se ocupă de gestionarea datelor și a interacțiunii cu baza de date. Ele permit realizarea operațiunilor CRUD (Create, Read, Update, Delete) pe datele din Model. Prin intermediul acestor persistențe, Presenter-ul poate să citească, să actualizeze sau să șteargă datele din baza de date și să le transmită View-ului pentru afișare sau modificare.

Așadar, Presenter este un strat intermediar între Model și View. Presenter-ul preia datele din Model, le procesează și le trimite spre View. De asemenea, gestionează evenimentele de interacțiune cu utilizatorul, care sunt inițiate de View, și actualizează Modelul și View-ul în consecință.

- **LoginPresenter:** Această clasă se ocupă de autentificarea utilizatorului. Ea are o metodă numită autentificare() care primește un nume de utilizator și o parolă din interfața ILogin, și apoi verifică dacă utilizatorul există în baza de date prin intermediul metodelor din UtilizatorPersistent. Dacă un utilizator valid este găsit, se setează interfața grafică corespunzătoare pentru rolul utilizatorului (administrator, manager sau angajat).
- **ManagerPresenter:** Această clasă gestioneaază interacțiunea dintre interfața IManager și clasele din Model care accesează baza de date(ParfumPersistent și MagazinPersistent). Ea conține metode pentru a gestiona acțiunile inițiate de manager prin apelul metodelor definite în clasele din model pentru filtrarea, sortarea și vizualizarea listei de parfumuri în funcție de magazinul selectat, precum și căutarea parfumurilor după nume.
- **AngajatPresenter:** Această clasă gestionează interacțiunea dintre interfața IAngajat și clasele din Model care accesează baza de date(ParfumPersistent și MagazinPersistent). Ea conține metode pentru a gestiona operațiile inițializate de angajat prin intermediul interfeței IAngajat și apoi apelează metodele definite în Model pentru a actualiza informațiile pe interfața grafică.
- **AdministratorPresenter:** Această clasă gestionează interacțiunea dintre IAdministrator și UtilizatorPersistent. acțiunile la care are acces administratorul (adăugarea, actualizarea și ștergerea utilizatorilor). Ea apelează metodele din UtilizatorPersistent pentru accesul la baza de date și actualizează lista de utilizatori pe interfața grafică corespunzătoare.

Clasele din pachetul Model sunt responsabile pentru interacțiunea cu baza de date și gestionarea obiectelor din domeniul aplicației.

- **Magazin:**Această clasă reprezintă entitatea Magazin din aplicație. Un magazin conține un id unic, nume și adresa. Clasa Magazin are metode pentru a seta și a obține valorile acestor atribute. Id-ul magazinului este folosit pentru a identifica în mod unic fiecare magazin în cadrul aplicației.
- **Parfum:** Clasa Parfum reprezintă entitatea Parfum în aplicație. Fiecare parfum are un id unic, nume, producător, preț și descriere. Clasa Parfum are metode pentru a seta și a obține valorile acestor atribute. Id-ul parfumului este folosit pentru a identifica în mod unic fiecare parfum în cadrul aplicației.
- **Utilizator:** Clasa Utilizator reprezintă entitatea Utilizator în aplicație. Fiecare utilizator are un id unic, nume de utilizator, parolă, rol și id-ul magazinului la care este asociat. Clasa Utilizator are metode pentru a seta și a obține valorile acestor atribute. Id-ul utilizatorului este folosit pentru a identifica în mod unic fiecare utilizator în cadrul aplicației.
- **ParfumMagazin:**Clasa ParfumMagazin reprezintă legătura dintre un parfum și un magazin. Aceasta conține instanțe ale obiectelor Parfum și Magazin, precum și informații despre stocul parfumului în magazinul respectiv. Clasa ParfumMagazin are metode pentru a seta și a obține valorile acestor atribute.
- **Persistent:**Clasa Persistent este o clasă generică abstractă ce definește o serie de metode pentru a manipula entitățile din baza de date. Aceasta include metode pentru a crea, citi, actualiza și șterge entități (CRUD), precum și pentru a obține toate înregistrările dintr-un tabel. Clasa Persistent este apoi moștenită de către clasele concrete pentru fiecare entitate din Model, precum ParfumPersistent, UtilizatorPersistent și MagazinPersistent.
- **ParfumPersistent**: este o subclasă a clasei generice Persistent<Parfum> și se ocupă de gestionarea obiectelor de tip Parfum. Ea conține metode pentru a efectua operațiuni CRUD (Create, Read, Update, Delete) pe entitatea Parfum în baza de date.

- **ParfumMagazinPersistent**: Această clasă se ocupă de gestionarea înregistrărilor din tabelul "ParfumMagazin" din baza de date. În această clasă sunt implementate metode pentru a crea, șterge și actualiza înregistrările legate de relația dintre parfumuri și magazine.
- **UtilizatorPersistent** extinde clasa Persistent<Utilizator>. Clasa UtilizatorPersistent oferă implementarea pentru metodele de realizare a operațiilor CRUD și alte metode suplimentare pentru obiectele Utilizator, folosind baza de date relațională.
- **MagazinPersistent:** Această clasă este o extensie a clasei Persistent și oferă implementarea efectivă a metodelor CRUD pentru entitatea Magazin.

Diagrama Entitate-Relație

Diagrama E-R dată este formată din patru tabele: „utilizator”, „parfum”, „magazin” și „parfumMagazin”.

Tabelul „Utilizator” stochează informații despre utilizator și are următoarele coloane:

- userId (PK): O cheie primară de tip int care identifică în mod unic fiecare utilizator.
- user: O coloană de tip char care stochează numele de utilizator.
- parola: O coloană de tip char care stochează parola utilizatorului.
- rol: O coloană de tip char care indică rolul utilizatorului (angajat, manager, administrator).
- idMagazin (FK): O cheie străină de tip int care se referențiază coloana idMagazin din tabelul „magazin”. Aceasta asociază fiecare utilizator de tip „angajat" cu un anumit magazin.

Tabelul „Parfum” stochează informații despre parfumuri și are următoarele coloane:

- idParfum (PK): O cheie primară de tip int care identifică în mod unic fiecare parfum.
- nume: O coloană de tip char care stochează numele parfumului.
- producător: O coloană de tip char care stochează numele producătorului parfumului.
- descriere: O coloană de tip char care conține o descriere a parfumului.
- pret: O coloană de tip float care stochează prețul parfumului.

Tabelul „magazin” stochează informații despre magazin și are următoarele coloane:

- idMagazin (PK): O cheie primară întreagă care identifică în mod unic fiecare magazin.
- nume: O coloană de tip char care stochează numele magazinului.
- adresa: O coloană de tip char care stochează adresa magazinului.

Tabelul "parfumMagazin" este un tabel asociativ creat pentru a evita relația de tip „many-to-many” între „Parfum” și „Magazin”. Acesta stochează informații despre stocul de parfumuri din fiecare magazin și are următoarele coloane:

- idParfum: O cheie străină de tip int care referențiază coloana idParfum din tabelul „Parfum”
- idMagazin: O cheie străină de tip int care referențiază coloana idMagazin din tabelul „Magazin”.
- stoc: O coloană de tip care stochează stocul unui anumit parfum într-un anumit magazin.
- **Cheia primară este compusă din coloanele idParfum și idMagazin**. Astfel, nu există în mai multe intrări care au același idMagazin și același idPArfum.

Relații:

- **Relația de asociere între tabelele "Utilizator" și "Magazin".** Fiecare utilizator de tipul „angajat” are un idMagazin asociat. Pentru utilizatorii „manager” și „administrator”, idMagazin este nul.

- **Relație de tip “one-to-many” între tabelul “Parfum” și tabelul “ParfumMagazin".** Astfel, pentru o instanță de tipul ”Parfum”, există zero, una sau mai multe instanțe de tip ”ParfumMagazin”, dar pentru o instanță de tip ”ParfumMagazin” există o singură instanță de tip ”Parfum”.

- **Relație de tip “one-to-many” între tabelul “Magazin” și tabelul “ParfumMagazin".** Astfel, pentru o instanță de tipul ”Magazin”, există zero, una sau mai multe instanțe de tip ”ParfumMagazin”, dar pentru o instanță de tip ”ParfumMagazin” există o singură instanță de tip "Magazin".

- În această configurație, tabelul “ParfumMagazin” acționează ca un intermediar (composite entity/ link table) între tabelele “Parfum” și "Magazin", evitându-se astfel o relație directă de tip "many-to-many" între acestea.

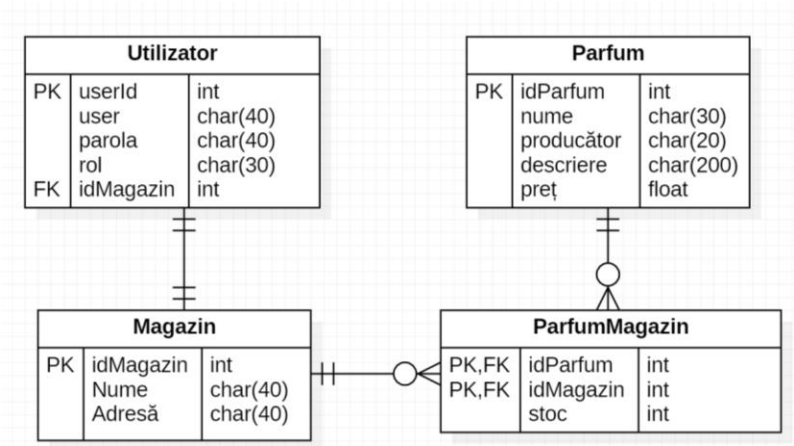

Fig. 3 - Diagrama Entitate-Relație

Descrierea aplicației

Aplicația reprezintă un sistem de gestiune a unui lanț de magazine de parfumuri care poate fi utilizată de trei tipuri de utilizatori după autentificare: angajat, manager și administrator. Utilizatorii au roluri diferite (de exemplu, angajat sau administrator), iar un utilizator cu rolul de angajat are și un id de magazin asociat.

Înregistrarea utilizatorului

Pachetul **View** conține interfața **ILogin** și clasa **LoginGUI.**
Interfața ILogin definește următoarele metode:

- **String getUser()** - returnează numele de utilizator introdus de utilizator în câmpul corespunzător.
- **String getParola()** - returnează parola introdusă de utilizator în câmpul corespunzător.
- **void mesajEroare()** - afișează un mesaj de eroare atunci când autentificarea eșuează.
- **void setAdministratorGUI(String user)** - inițializează și afișează interfața grafică pentru un utilizator cu rol de administrator.
- **void setManagerGUI(String user)** - inițializează și afișează interfața grafică pentru un utilizator cu rol de manager.
- **void setAngajatGUI(String user, int idMagazin)** - inițializează și afișează interfața grafică pentru un utilizator cu rol de angajat.

Clasa LoginGUI implementează interfața ILogin și este responsabilă pentru crearea și gestionarea interfeței grafice de autentificare. Aceasta include câmpuri pentru introducerea numelui de utilizator și a parolei, precum și butoane pentru autentificare și anulare.

Pachetul Presenter conține clasa LoginPresenter, care se ocupă de logica de autentificare a utilizatorilor prin intermediul acestei metode:

- **public void autentificare()** - Această metodă preia numele de utilizator și parola introduse de utilizator prin intermediul interfeței ILogin. Apoi, folosește clasa UtilizatorPersistent pentru a căuta un utilizator cu aceste credențiale în baza de date. Dacă utilizatorul există, metoda stabilește ce interfață grafică trebuie să fie afișată în funcție de rolul utilizatorului (administrator, manager sau angajat). Dacă utilizatorul nu există, metoda afișează un mesaj de eroare prin intermediul metodei mesajEroare() din interfața ILogin.

În timpul procesului de autentificare, utilizatorul introduce numele de utilizator și parola în interfața grafică. Aceste informații sunt preluate de clasa LoginPresenter, care le folosește pentru a interoga baza de date prin intermediul metodei **căutareUtilizator** a clasei UtilizatorPersistent. Dacă există un utilizator cu credențialele furnizate, aplicația redirecționează utilizatorul către interfața grafică corespunzătoare rolului său. În caz contrar, un mesaj de eroare este afișat pentru a informa utilizatorul că numele de utilizator sau parola sunt invalide.

Pachetul Model conține clasa UtilizatorPersistent în care este definită următoarea metodă:

- **public Utilizator cautareUtilizator(String user, String password):** Această metodă caută în baza de date un Utilizator cu un anumit nume de utilizator și o anumită parolă primite de la Presenter. Prin intermediul conexiunii la baza de date, se pregătește o interogare SQL care verifică dacă există în baza de date un rând care să corespundă acestor informații. Dacă se găsește un astfel de rând, informațiile sunt extrase și utilizate pentru a crea un obiect Utilizator, care este returnat ca rezultat al metodei. Dacă nu se găsește niciun rând care să corespundă criteriilor de căutare, atunci valoarea returnată va fi null.

Utilizatorul de tip Angajat

Un utilizator de tip Angajat are acces la funcționalități precum adăugarea unui parfum nou în baza de date, adăugarea unui parfum existent din baza de date a lanțului de magazine în magazinul său, actualizarea stocului pentru parfumurile din magazin, ștergerea unui parfum din magazin, precum și filtrarea parfumurilor în funcție de disponibilitate, preț și producător sau vizualizarea detaliilor unui parfum.

Pachetul **View** conține interfața **IAngajat** și **AngajatGUI** care implementează interfața IAngajat.

În clasa **AngajatGUI** sunt implementate următoarele metode:

- **getIdParfum():** returnează Id-ul parfumului introdus de angajat (pentru parfumurile deja existente în baza de date a lanțului de magazine)
- **getStoc():** returnează stocul parfumului introdus de angajat(pentru parfumurile deja existente în baza de date a lanțului de magazine)
- **getNewIdParfum():** returnează Id-ul introdus de angajat pentru un nou parfum în baza de date
- **getNewNume():**returnează numele introdus de angajat pentru un nou parfum în baza de date
- **getNewProducator():**returnează producătorul introdus de angajat pentru un nou parfum în baza de date **getNewPret():**returnează pretul introdus de angajat pentru un nou parfum în baza de date
- **getNewDescriere():** returnează descriere introdusă de angajat pentru un nou parfum în baza de date
- **getNewStoc():**returnează stocul introdus de angajat pentru un nou parfum în baza de date showParfumUpdatedSuccessMessage(): afișează un mesaj de succes când parfumul este actualizat cu succes.
- **showFailedToUpdateParfumMessage():** afișează un mesaj de eroare când actualizarea parfumului eșuează.
- **showParfumNotSelectedMessage():** afișează un mesaj de eroare când nu este selectat niciun parfum.
- **showParfumAddedSuccessMessage():** afișează un mesaj de succes când parfumul este adăugat cu succes.
- **showFailedToAddParfumMessage():** afișează un mesaj de eroare când adăugarea parfumului eșuează.
- **showParfumDeletedSuccessMessage():** afișează un mesaj de succes când parfumul este șters cu succes.
- **showFailedToDeleteParfumMessage():** afișează un mesaj de eroare când ștergerea parfumului eșuează.
- **showIdStocInvalidMessage():** afișează un mesaj de eroare când ID-ul parfumului sau stocul este invalid.
- De asemenea, AngajatGUI conține butoane pentru a declanșa diferite acțiuni, cum ar fi adăugarea, actualizarea, ștergerea, filtrarea și vizualizarea detaliilor parfumurilor. Aceste butoane vor apela metodele corespunzătoare din AngajatPresenter pentru a efectua operațiunile necesare.

În clasa **AngajatPresenter** din pachetul **Presenter** sunt implementate următoarele metode:

- **addParfum():** gestionează adăugarea unui parfum nou în magazin, utilizând valorile furnizate de utilizator prin interfața AngajatGUI. În Presenter este apelată funcția **createParfumMagazin(int idParfum, int idMagazin, int stoc)** din Model și încearcă să creeze un nou obiect ParfumMagazin. Dacă operațiunea reușește, parfumul este adăugat cu succes și se afișează un mesaj corespunzător. În caz contrar, se afișează un mesaj de eroare.
- **updateParfum():** actualizează stocul unui parfum existent în magazin. Funcția verifică dacă ID-ul parfumului și stocul sunt valori valide și apoi încearcă să actualizeze stocul în baza de date, apelând metoda **updateParfumMagazinStock(int idParfum, int idMagazin, int stoc)** din Model. Dacă actualizarea reușește, parfumul este actualizat cu succes și se afișează un mesaj corespunzător. În caz contrar, se afișează un mesaj de eroare.
- **deleteParfum():** șterge un parfum selectat din magazin. Metoda verifică dacă un parfum a fost selectat și apoi încearcă să șteargă parfumul din baza de date apelând metoda **deleteParfumMagazin(int idParfum, int idMagazin)** din Model. Dacă ștergerea reușește, parfumul este șters cu succes și se afișează un mesaj corespunzător. În caz contrar, se afișează un mesaj de eroare.
- **addNewParfum():** Această funcție adaugă un parfum complet nou în baza de date. Funcția verifică dacă toate câmpurile introduse de utilizator sunt valabile și apoi încearcă să creeze un nou obiect Parfum și să-l adauge în baza de date, apelând atât metoda **create(Parfum parfum),** cât și metoda **createParfumMagazin(int idParfum, int idMagazin, int stoc)** din Model . Dacă adăugarea reușește, parfumul este adăugat cu succes și se afișează un mesaj corespunzător. În caz contrar, se afișează un mesaj de eroare.
- **displayParfumList():** afișează lista de parfumuri disponibile în magazin. Funcția creează preia toate parfumurile din baza de date prin intermediul metodei **getAllParfumuriForMagazin(int idMagazin)** din Model și le trimite către interfața AngajatGUI pentru afișare.
- **handleGetAllProducators():** preia lista de producători de parfumuri din baza de date apelând metoda getAllProducators(int idMagazin): și le trimite către interfața AngajatGUI pentru afișare în comboBox-ul pentru producători astfel încât angajatul să poate selecta direct producătorii.
- **handleFilterParfums():** filtrarea parfumurilor în funcție de criteriile furnizate de utilizator prin interfața AngajatGUI. Funcția extrage valori pentru prețul minim, prețul maxim, producător și disponibilitate și, în funcție de valorile furnizate, va filtra parfumurile din baza de date apelând metoda filterParfums(int idMagazin, Double priceMin, Double priceMax, String producator, Boolean disponibil). Rezultatul filtrării este apoi trimis către interfața AngajatGUI pentru afișare.

În pachetul **Model** sunt implementate următoarele metode:
În clasa **ParfumPersistent:**

- **create(Parfum parfum):** Această metodă este folosită pentru a crea un nou obiect Parfum în baza de date. primește ca argument un obiect Parfum și întoarce un boolean care indică dacă operațiunea a avut succes sau nu.
- **getAllParfumuriForMagazin(int idMagazin):** Extrage toate parfumurile disponibile pentru un anumit magazin. Ea primește ca argument ID-ul magazinului și returnează o listă de obiecte ParfumMagazin care conțin toate parfumurile și stocul lor pentru acel magazin.
- **getAllProducators(int idMagazin):** Extrage toți producătorii de parfumuri pentru un anumit magazin. Ea primește ca argument ID-ul magazinului și returnează o listă de String-uri care conține numele tuturor producătorilor.
- **filterParfums(int idMagazin, Double priceMin, Double priceMax, String producator, Boolean disponibil):** Filtrează parfumurile din baza de date în funcție de criteriile furnizate. Ea primește ca argumente Id-ul magazinului, prețul minim, prețul maxim, numele producătorului și disponibilitatea parfumului și returnează o listă de obiecte ParfumMagazin care corespund criteriilor de filtrare.

În clasa ParfumMagazinPersistent:

- **createParfumMagazin(int idParfum, int idMagazin, int stoc):** Această metodă creează un nou obiect ParfumMagazin în baza de date. Ea primește ca argumente ID-ul parfumului, ID-ul magazinului și stocul și întoarce un boolean care indică dacă operațiunea a avut succes sau nu.
- **updateParfumMagazinStock(int idParfum, int idMagazin, int stoc):** Această metodă actualizează stocul unui ParfumMagazin în baza de date. Ea primește ca argumente ID-ul parfumului, ID-ul magazinului și noul stoc și întoarce un boolean care indică dacă operațiunea a avut succes sau nu.
- **deleteParfumMagazin(int idParfum, int idMagazin):** Această metodă șterge un obiect ParfumMagazin din baza de date. Ea primește ca argumente ID-ul parfumului și ID-ul magazinului și întoarce un boolean care indică dacă operațiunea a avut succes sau nu.

Utilizatorul de tip Manager

Pachetul **View** conține interfața **IManager** și **ManagerGUI** care implementează interfața **Imanager**
În clasa ManagerGUI sunt implementate următoarele metode:
initUI(): Inițializează interfața grafică, adăugând componente precum tabele, casete combo, etichete și butoane.

- **getSelectedMagazinName():** Returnează numele magazinului selectat în comboBoxMagazin.
- **getIdMagazin():** Returnează ID-ul magazinului selectat.
- **setProducatorList(List<String> producators):** Setează lista de producători în comboBoxProducator.
- **getRowData(JTable table, int row):** Returnează datele rândului selectat în tabel.
- **getPriceMin():** Returnează prețul minim introdus în câmpul textFieldPretMin.
- **getPriceMax():** Returnează prețul maxim introdus în câmpul textFieldPretMax.
- **getSelectedProducator():** Returnează producătorul selectat în comboBoxProducator.
- **isDisponibilSelected():** returnează true dacă Managerul bifează căsuța chckbxDisponibil și false în caz contrar.
- **getSelectedParfumId():** Returnează ID-ul parfumului selectat în tabel.
- **setMagazinList(List<String> magazinNames):** Setează lista de nume ale magazinelor în comboBoxMagazin.
- **isSortByPriceAscending():** Verifică dacă parfumurile trebuie să fie sortate în ordine crescătoare sau descrescătoare în funcție de preț.
- **isSortByNameAscending():** Verifică dacă parfumurile trebuie să fie sortate în ordine crescătoare sau descrescătoare în funcție de nume.
- **showParfumNotFoundMessage():** Afișează un mesaj de eroare când parfumul introdus nu este găsit.
- **showParfumNotSelectedMessage():** Afișează un mesaj de eroare când nu este selectat niciun parfum.
- **getEnteredParfumName():** Returnează numele parfumului introdus în câmpul textFieldCautaDupaNume.
- **showPerfumeDetailsByName(String perfumeDetails):** Afișează detalii despre parfumul găsit după nume.
- **showPerfumeDetails(String perfumeDetails):** Afișează detalii despre parfumul selectat din tabel.
- **displayParfumList(Object[][] parfumList):** Afișează lista de parfumuri în formă tabelară

În clasa **ManagerPresenter** din pachetul **Presenter** sunt implementate următoarele metode care apelează metodele din Model pentru a extrage sau a actualiza date solicitate de manager:

- **getAllMagazinNames():**extrage toate numele magazinelor și le returnează într-o listă de tip String
- **getMagazinIdByName():**returnează ID-ul unui magazin în funcție de numele său introdus de manager în interfața vizuală.
- **handleGetAllProducators():**gestionează extragerea tuturor producătorilor de parfumuri dintr-un magazin și actualizează lista de producători în interfața vizuală.
- **handleFilterParfums():** gestionează filtrarea parfumurilor în funcție de criteriile introduse de utilizator (preț minim, preț maxim, producător, disponibilitate) și actualizează lista de parfumuri în interfața vizuală.
- **handleSortParfumsByName():**gestionează sortarea parfumurilor în funcție de nume, în ordine crescătoare sau descrescătoare, și actualizează lista de parfumuri în interfața vizuală.
- **handleSortParfumsByPrice():**gestionează sortarea parfumurilor în funcție de preț, în ordine crescătoare sau descrescătoare, și actualizează lista de parfumuri în interfața vizuală.
- **handleSearchParfumByName():**gestionează căutarea unui parfum după nume și afișează detaliile parfumului găsit în interfața vizuală sau un mesaj de eroare dacă parfumul nu a fost găsit.
- **displayParfumList():** gestionează afișarea listei de parfumuri într-un tabel în interfața vizuală.

Pachetul Model conține **metodele de filtrare a parfumurilor și de vizualizare a listei de parfumuri descrise anterior (secțiunea Angajat)** și aceste metode apelate doar în ManagerPresenter:

- **getSortedParfumsByName(int idMagazin, boolean ascending):** returnează o listă de obiecte ParfumMagazin sortate în funcție de numele parfumurilor, în ordine crescătoare sau descrescătoare, pentru un magazin specificat prin ID.
- **getSortedParfumsByPrice(int idMagazin, boolean ascending):** returnează o listă de obiecte ParfumMagazin sortate în funcție de prețul parfumurilor, în ordine crescătoare sau descrescătoare, pentru un magazin specificat prin ID.
- **readByName(String parfumName):** caută un parfum după nume și returnează obiectul Parfum corespunzător sau null dacă parfumul nu a fost găsit.

Utilizatorul de tip administrator

În pachetul View avem interfața IAdministrator și clasaAdministratorGUI.
AdministratorGUI este o clasă care extinde JFrame și implementează interfața IAdministrator. Această clasă creează interfața grafică pentru un administrator, oferindu-i acestuia posibilitatea de a adăuga, șterge, actualiza și vizualiza detaliile utilizatorilor:

- **initialize():** inițializează interfața grafică a AdministratorGUI. Aici sunt adăugate toate elementele necesare, cum ar fi: tabelul cu utilizatori, panoul de control cu câmpurile de text, etichete și butoane.
- **ActionListener pentru addUserButton:** Atunci când se apasă butonul "Adaugă utilizator", acest listener apelează metoda addUser() din clasa AdministratorPresenter.
- **ActionListener pentru deleteUserButton:** Atunci când se apasă butonul "Șterge utilizator", acest listener apelează metoda deleteUser() din clasa AdministratorPresenter.
- **ActionListener pentru updateUserButton:** Atunci când se apasă butonul "Actualizează utilizator", acest listener apelează metoda updateUser() din clasa AdministratorPresenter.
- **ActionListener pentru viewUserDetailsButton:** Atunci când se apasă butonul "Vizualizează detaliile unui utilizator", acest listener preia informațiile despre utilizatorul selectat și le afișează într-o fereastră nouă, dacă un utilizator a fost selectat.
- **showSelectUserToDeleteMessage(), showUserDeletedSuccessMessage() etc.:** utilizate pentru a afișa diferite mesaje și notificări pentru utilizator în funcție de acțiunile realizate.
- **getUserId(), getUserName(), getPassword(), getRol(), getIdMagazin():**returnează informațiile despre utilizator din câmpurile de text corespunzătoare.
- **getSelectedUserId():**returnează ID-ul utilizatorului selectat în tabel.
- **getRowData(int row):** returnează datele unei anumite rânduri din tabelul cu utilizatori.
- **displayUsers(Object[][] users):** actualizează modelul tabelului cu utilizatori, afișând informațiile primite în parametru.
- **showUserDetails(String userDetails):** afișează o fereastră nouă cu detaliile utilizatorului selectat.

În pachetul Presenter clasa AdministratorPresenter face legătura între interfața grafică (IAdministrator) și persistența datelor (UtilizatorPersistent). Această clasă are următoarele metode care apelează metodele din presenter:
addUser(): crează un nou utilizator cu informațiile furnizate prin interfața grafică și îl adaugă în sistem prin intermediul metodei din Model. Dacă adăugarea utilizatorului este cu succes, afișează un mesaj de succes și actualizează lista de utilizatori. În caz contrar, afișează un mesaj de eroare.
updateUser():actualizează informațiile unui utilizator apelând metoda aferentă din Model, în funcție de datele introduse în interfața grafică. Dacă actualizarea este reușită, afișează un mesaj de succes și actualizează lista de utilizatori. În caz contrar, afișează un mesaj de eroare.
deleteUser(): șterge un utilizator selectat din sistem, apelând metoda responsabilă din Model. Dacă ștergerea este reușită, afișează un mesaj de succes și actualizează lista de utilizatori. În caz contrar, afișează un mesaj de eroare.
displayUserList():afișează lista de utilizatori în interfața grafică, apelând metoda din Model care extrage lista de utilizatori din UtilizatorPersistent și o trimite către interfața grafică pentru a fi afișată.

În pachetul model clasa UtilizatorPersistent are următoarele metode
create(Utilizator utilizator): adaugă un nou utilizator în sistem și returnează true dacă operațiunea este reușită sau false dacă nu.
update(Utilizator utilizator): actualizează datele unui utilizator existent în sistem și returnează true dacă operațiunea este reușită sau false dacă nu.
delete(int userId): șterge un utilizator cu ID-ul specificat din sistem și returnează true dacă operațiunea este reușită sau false dacă nu.
getAll(): returnează o listă cu toți utilizatorii din sistem.

read(int userId): returnează un utilizator cu ID-ul specificat sau null dacă acesta nu există în sistem.

Conexiunea la baza de date
Conexiunea la baza de date se realizează prin intermediul unui obiect de tip Connection care este gestionat de o clasă Singleton numită DatabaseConnection. Această clasă gestionează conexiunea și asigură că există o singură instanță a acesteia în întreaga aplicație. Toate clasele de tip Persistent, precum ParfumPersistent, ParfumMagazinPersistent, UtilizatorPersistent și MagazinPersistent, apelează metoda getInstance() din DatabaseConnection pentru a obține conexiunea la baza de date.
Fiecare clasă Persistent conține metode pentru a efectua operații CRUD (Create, Read, Update, Delete) asupra entităților specifice din baza de date. Aceste metode utilizează obiectul Connection pentru a executa interogări SQL și pentru a procesa rezultatele.

Când o metodă dintr-o clasă Persistent necesită interacțiunea cu baza de date, aceasta apelează metoda getInstance() din DatabaseConnection pentru a obține conexiunea și apoi execută interogările și operațiunile necesare. Prin utilizarea acestei abordări, conexiunea la baza de date este gestionată într-un mod centralizat, asigurând o mai bună eficiență și consistență în întreaga aplicație.

Testarea aplicației

Testele unitate sunt o parte esențială în dezvoltarea și asigurarea calității unei aplicații, întrucât ele permit verificarea funcționalității și a corectitudinii codului. În cazul de față, s-au realizat teste unitate pentru fiecare interacțiune cu baza de date din clasele ParfumPersistent, ParfumMagazinPersistent, UtilizatorPersistent și MagazinPersistent. De exemplu, în clasa ParfumPersistentTest, înainte de fiecare test, se creează o conexiune la baza de date prin apelarea metodei DatabaseConnection.getConnection(), iar după fiecare test, conexiunea este închisă. Testele verifică corectitudinea funcționării metodelor CRUD (Create, Read, Update, Delete) ale clasei ParfumPersistent.

De asemenea, în clasa ParfumMagazinPersistentTest, se urmează o abordare similară pentru a testa metodele createParfumMagazin(), updateParfumMagazinStock() și deleteParfumMagazin(). În acest caz, se creează o conexiune la baza de date în metoda setUp() și se închide în metoda tearDown().

În ambele exemple, testele unitate au menirea de a verifica dacă metodele de interacțiune cu baza de date funcționează corect, prin compararea rezultatelor obținute cu valorile așteptate. Astfel, se asigură că operațiunile de adăugare, actualizare, ștergere și citire a datelor din baza de date se efectuează în mod corect și că aplicația se comportă conform cerințelor.

Această abordare este aplicată și în cazul celorlalte clase care interacționează cu baza de date, respectiv UtilizatorPersistent și MagazinPersistent. Testele unitate ajută la identificarea și remedierea eventualelor probleme sau erori în mod eficient, contribuind astfel la asigurarea calității și robusteții întregii aplicații.

Concluzie

Aplicația prezentată urmează modelul MVP (Model-View-Presenter), care este un model arhitectural folosit pentru a separa logica de business de interacțiunea cu utilizatorul și de gestionarea datelor. Modelul MVP facilitează modularitatea și testarea aplicației, deoarece componentele sunt izolate și pot fi modificate independent. În acest model, View reprezintă interfața cu utilizatorul, Model conține datele și logica de business, iar Presenter face legătura între View și Model, coordonând acțiunile utilizarorului și actualizările bazei de date

Aplicația gestionează informațiile referitoare la parfumuri și utilizatori, oferind funcționalități precum adăugarea, actualizarea, ștergerea și afișarea acestora. Clasele de tip Persistent se ocupă de interacțiunile cu baza de date, asigurând un mod eficient și corect de a opera cu datele. Interfețele și clasele View oferă un mod intuitiv și ușor de utilizat pentru a interacționa cu aplicația, iar clasele Presenter se ocupă de comunicarea dintre View și Model.

În concluzie, aplicația reprezintă o implementare eficientă și bine organizată a modelului MVP, care facilitează dezvoltarea, întreținerea și extinderea funcționalităților. Utilizarea acestui model arhitectural îmbunătățește modularitatea și testabilitatea aplicației, oferind o bază solidă pentru dezvoltarea ulterioară.

Bibliografie

Apache Friends. (n.d.). XAMPP. https://www.apachefriends.org/index.html
AppBrain. (2021). Top 10 most downloaded Android apps of all time. https://www.appbrain.com/stats/top-apps/all-time
Chen, P. P. (1976). The entity-relationship model: Toward a unified view of data. ACM Transactions on Database Systems (TODS), 1(1), 9-36.
Forbes. (2020). The Best Programming Languages for AI Development. https://www.forbes.com/sites/louiscolumbus/2020/02/16/the-best-programming-languages-for-ai-development/?sh=6c3835246f9f
Game Developer Magazine. (2020). 2020 Game Developer Survey. https://gamedevsurvey.com/survey-results-2020/
Gosling, J., Joy, B., Steele, G., & Bracha, G. (2005). The Java language specification (3rd ed.). Addison-Wesley.
Gupta, A., & Sharma, R. (2021). Comparative Analysis of Java and Python for Software Development. International Journal of Engineering Research and Applications, 11(3), 1-9.
IDC. (2020). Worldwide IoT Spending Forecast to Reach $1.1 Trillion in 2023 Led by Industry Discrete Manufacturing Followed by Process Manufacturing, Transportation and Utilities, According to a New IDC Spending Guide. https://www.idc.com/getdoc.jsp?containerId=prUS45316820
IntelliJ IDEA. (2021). JetBrains. Retrieved from https://www.jetbrains.com/idea/
Java Database Connectivity (JDBC) API. (2021). Oracle. https://docs.oracle.com/en/java/javase/16/docs/api/java.sql/module-summary.html
Java Swing. (2021). Oracle. Retrieved from https://www.oracle.com/java/technologies/javase/java-swing.html
Liu, X., Wang, Y., & Zhang, X. (2020). Design and implementation of computer-aided experiment system based on Java Swing. Journal of Physics: Conference Series, 1589(2), 022044.
Martin, R. C. (2000). Design principles and design patterns. Retrieved from https://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf
MySQL. (n.d.). Oracle. https://www.mysql.com/
Oracle. (2021a). Java Technologies. Retrieved from https://www.oracle.com/java/technologies/
Oracle. (2021b). JDBC - Java Database Connectivity. Retrieved from https://www.oracle.com/database/technologies/jdbc.html
Park, S., & Lee, S. (2020). Improving Developer Productivity and Software Quality with IntelliJ IDEA. In Proceedings of the 2020 International Conference on Software Engineering and Information Management (pp. 29-33).
Potter, N. (2012). The Model-View-Presenter (MVP) design pattern for PHP. Journal of Object Technology, 11(1), 1-19.
Preez, N. D., Kourie, D. G., & Boake, A. (2006). Java in academia: A survey. ACM SIGCSE Bulletin, 38(4), 131-136.
Rahman, A., Anwar, M. A., & Rahman, A. (2019). The Importance of UML in Developing Maintainable Software. International Journal of Computer Applications, 178(34), 1-4.
RedMonk. (2021). The RedMonk Programming Language Rankings: January 2021