UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Aplicație pentru gestionarea unui lanț de magazine de parfumuri

Arhitectura Model-View-Controller

Nume student: *Dumitru Isabela-Bianca*

Cuprins

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.

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

2. Introducere

Modelul Model-View-Controller (MVC) este un pattern arhitectural software care facilitează separarea responsabilităților în dezvoltarea aplicațiilor(Reenskaug, 1979), fiind adoptat în numeroase platforme și tehnologii, precum **Java, .NET, Ruby on Rails** și multe altele (Leff & Rayfield, 2001).

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

Un **exemplu notabil de utilizare a modelului MVC este dezvoltarea aplicației Twitter** realizată de compania Twitter, Inc. (Gascoigne, 2011).

Modelul MVC 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 MVC este format din trei componente principale: Model, View și Controller, care au următoarele responsabilități:

- **Componenta Model** reprezintă structura datelor și logica de business a aplicației. Acesta gestionează accesul la date, precum și operațiunile și validările specifice domeniului. Modelul nu are cunoștință despre modul în care datele sunt prezentate sau despre interacțiunile cu utilizatorul.
- **Componenta View** se ocupă de prezentarea datelor către utilizator. Aceasta reprezintă interfața grafică a aplicației și actualizează datele afișate în funcție de schimbările care apar în Model. View-ul nu interacționează direct cu Modelul, ci comunică cu acesta prin intermediul Controller-ului.
- **Componenta Controller** gestionează fluxul de date între Model și View și prelucrează interacțiunile utilizatorului. Controller-ul primește input-uri de la utilizator prin View, procesează aceste input-uri și, dacă este necesar, actualizează Modelul. De asemenea, Controller-ul este responsabil pentru actualizarea View-ului în funcție de modificările din Model.

Așadar, Modelul gestionează datele și logica de business, View-ul se ocupă de prezentarea datelor, iar Controller-ul gestionează interacțiunile între celelalte două componente și coordonează fluxul de date în aplicație.

Avantajele modelului MVC

Separarea responsabilităților (Reenskaug, 1979): Modelul MVC împarte aplicația în cele trei componente distincte discutate anterior - Model, View și Controller - care își asumă responsabilități separate, simplificând astfel menținerea și dezvoltarea aplicației.

Mentenabilitatea și modularitatea (Leff & Rayfield, 2001): Separarea responsabilităților în componente individuale îmbunătățește mentenabilitatea aplicației, facilitând actualizarea și îmbunătățirea componentelor în mod independent, fără a afecta celelalte componente.

Reutilizarea codului (Alur et al., 2003): Componentele Model, View și Controller pot fi reutilizate în diferite părți ale aplicației sau în alte aplicații, reducând astfel duplicarea codului și efortul de dezvoltare.

Testabilitatea (Martin, 2000): Datorită separării responsabilităților, componentele pot fi testate în mod independent, ceea ce facilitează testarea unitară și testarea integrării.

Flexibilitatea în dezvoltarea aplicațiilor (Gascoigne, 2011): Modelul MVC permite dezvoltatorilor să se concentreze asupra aspectelor specifice ale aplicației, în loc să se preocupe de detalii tehnice complexe, oferind flexibilitate în dezvoltarea aplicațiilor.

Studii recente continuă să demonstreze valoarea și relevanța modelului Model-View-Controller în dezvoltarea aplicațiilor software, în special în contextul tehnologiilor și platformelor moderne, cum ar fi aplicațiile mobile, web și IoT:

- *"A Model-View-Controller (MVC) Framework for Android Applications."* (Shaout, A., & Al-Mutlaq, I., 2016): Această lucrare prezintă un framework MVC pentru dezvoltarea aplicațiilor Android, evidențiind avantajele utilizării modelului MVC în contextul dezvoltării aplicațiilor mobile, cum ar fi modularitatea, reutilizarea codului și testabilitatea.

- *"Model-View-Controller (MVC) Based Design Pattern for Web Applications."* (Gilani, S. S., & Qureshi, M. R., 2017): Această lucrare analizează modul în care modelul MVC poate fi utilizat pentru a îmbunătăți dezvoltarea aplicațiilor web, punând accent pe separarea responsabilităților și facilitarea testării și mentenabilității aplicațiilor.

- *"Research and Design of MVC-Based Web Application Architecture."* (Wang, X., Qiao, Y., & Zhang, Z., 2018): În acest studiu, autorii investighează arhitectura aplicațiilor web bazate pe modelul MVC și prezintă avantajele acestui model în dezvoltarea aplicațiilor web, cum ar fi îmbunătățirea performanței și scalabilității aplicațiilor.

- *"Model-View-Controller-Based Design for Implementing Reusable IoT System Software Components."* Damaševičius, R., Maskeliūnas, R., & Kazanavičius, E. (2019): Această lucrare explorează utilizarea modelului MVC în contextul sistemelor software pentru Internet of Things (IoT), subliniind avantajele modelului în ceea ce privește reutilizarea componentelor software și facilitarea adaptării sistemelor IoT la diferite contexte de utilizare.

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

1. În faza de **analiză**, a fost creată o diagramă de cazuri de utilizare pentru a evidenția cerințele funcționale ale aplicației.
2. În faza de **proiectare,** a fost elaborată o diagramă de clase care respectă arhitectura MVC și principiile SOLID. În plus, a fost creată o diagramă entitate-relație pentru proiectarea bazei de date.
3. 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.
4. 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.

3. Instrumente utilizate

Instrumentele utilizate în proiectarea și dezvoltarea aplicației au fost alese pe baza eficienței și a compatibilității lor cu modelul MVC, 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 MVC 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 MVC (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 MVC, 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 Controller în modelul MVC.

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 MVC.

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 MVC, 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 MVC. 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 MVC 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 MVC 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 MVC 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 MVC 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 MVC, 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.

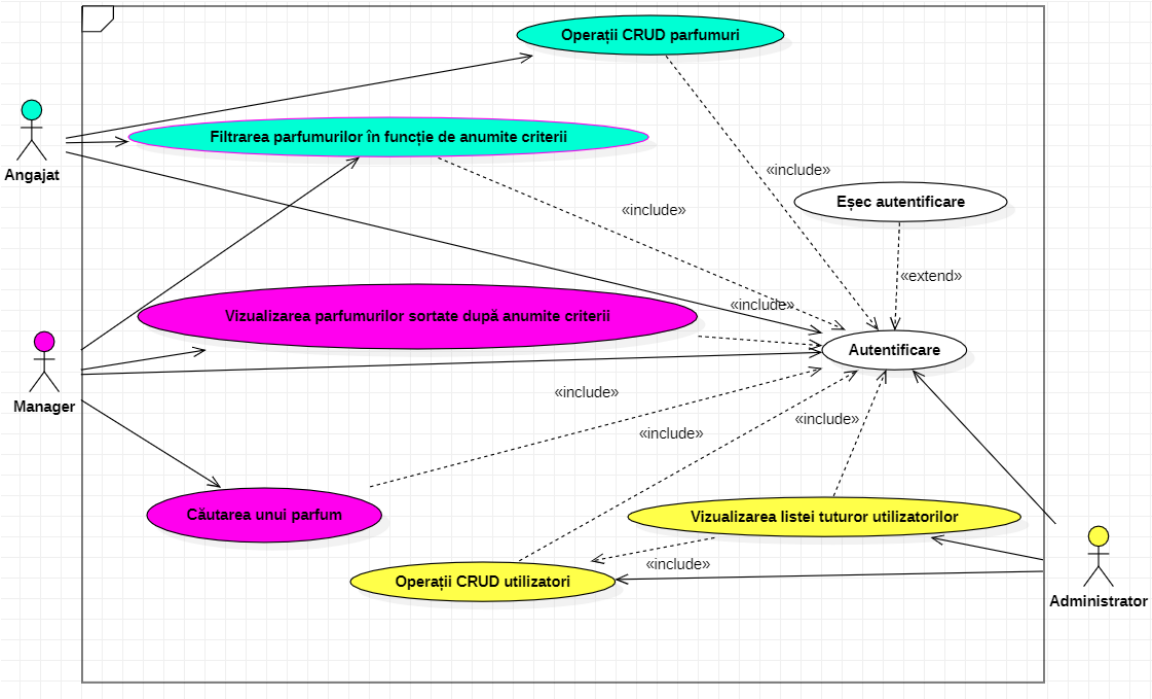

Fig. 1 - Diagrama cazurilor de utilizare

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

Diagrama de clasă

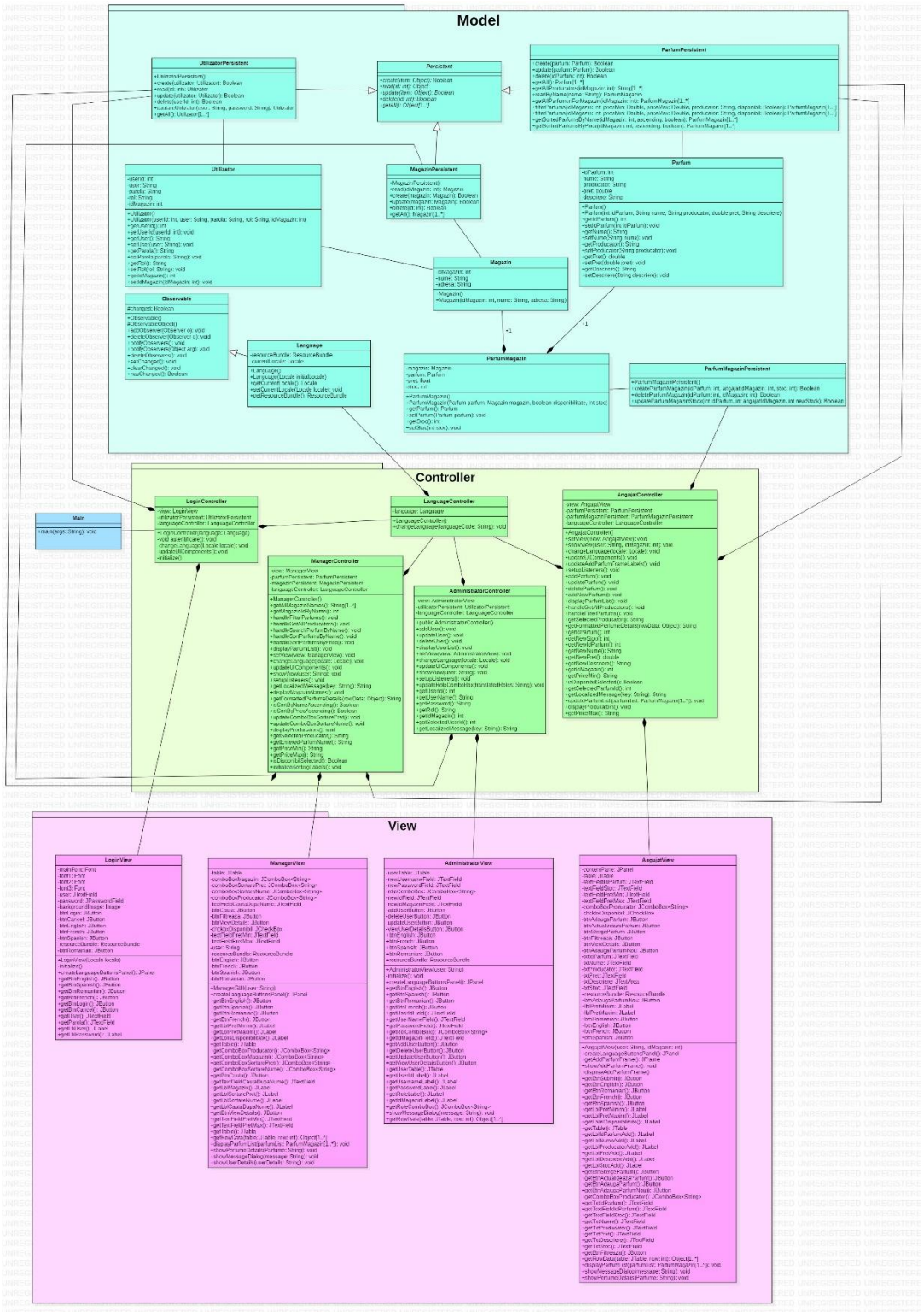

Fig. 2 Diagrama de clasă

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

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:

- **LoginView:** Clasa LoginView reprezintă componenta vizuală a procesului de autentificare. Acesta include elemente de interfață pentru introducerea numelui de utilizator și a parolei și comunică cu LoginController pentru a realiza autentificarea.
- **AngajatView:** Clasa AngajatView reprezintă componenta vizuală a funcționalităților angajatului. Acesta include elemente de interfață pentru afișarea și modificarea informațiilor despre parfumuri și comunică cu AngajatController pentru a realiza operațiile necesare.
- **AdministratorView**: Clasa AdministratorView reprezintă componenta vizuală a funcționalităților administratorului. Acesta include elemente de interfață pentru gestionarea utilizatorilor și a magazinelor și comunică cu AdministratorController pentru a realiza operațiile necesare.
- **ManagerView:** Clasa ManagerView reprezintă componenta vizuală a funcționalităților managerului. Acesta include elemente de interfață pentru gestionarea stocului de parfumuri și a informațiilor despre parfumuri și comunică cu ManagerController pentru a realiza operațiile necesare.

Clasele din Controller au rolul de a media interacțiunea dintre View și Model, facilitând comunicarea dintre acestea și gestionând logica de afișare și de actualizare a datelor.Controller-ul preia datele din Model, le procesează și le trimite spre View. De asemenea, gestionează evenimentele de interacțiune cu utilizatorul, și actualizează Modelul și View-ul în consecință.

- **LanguageController:** Clasa LanguageController gestionează interacțiunea dintre modelul Language și componentele vizuale. Acesta controlează schimbarea limbii în aplicație și se asigură că textele sunt traduse corespunzător.
- **LoginController**: Clasa LoginController gestionează procesul de autentificare al utilizatorilor. Acesta interacționează cu modelul Utilizator pentru a verifica dacă datele de autentificare sunt corecte și, în funcție de rolul utilizatorului, deschide fereastra corespunzătoare.
- **AngajatController:** Clasa AngajatController gestionează funcționalitățile specifice unui angajat al magazinului. Acesta interacționează cu modelele Magazin, Parfum și ParfumMagazin pentru a realiza operațiile de vizualizare și modificare a informațiilor despre parfumuri.
- **AdministratorController:** Clasa AdministratorController gestionează funcționalitățile specifice unui administrator al aplicației. Acesta interacționează cu modelele Magazin, Parfum, ParfumMagazin și Utilizator pentru a realiza operațiile de gestionare a utilizatorilor.
- **ManagerController:** Clasa ManagerController gestionează funcționalitățile specifice unui manager al magazinului. Acesta interacționează cu modelele Magazin, Parfum și ParfumMagazin și pentru a realiza operațiile de gestionare a stocului de parfumuri și a informațiilor despre parfumuri**.**

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.
- **Language** reprezintă entitatea Limba în aplicație și extinde clasa **Observable**. Clasa Observable face parte din șablonul Observer al limbajului Java, care permite obiectelor să notifice alte obiecte cu privire la schimbările din starea lor. În cazul acestei aplicații, obiectul Language poate fi observat de alte obiecte care s-au înregistrat ca observatori folosind metoda addObserver(). Când metoda setCurrentLocale() este apelată, metoda setChanged() este apelată pentru a indica faptul că starea obiectului a fost modificată, iar metoda notifyObservers() este apelată pentru a notifica toți observatorii înregistrați cu privire la schimbare. Obiectul Locale care a fost transmis ca argument metodei setCurrentLocale() este trimis observatorilor ca notificare. Prin urmare, obiectul observabil este obiectul Language, iar observatorii sunt obiecte care s-au înregistrat folosind metoda addObserver().

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.

Tabelul "Parfum_Translation" stochează traducerile pentru descrierea parfumurilor în patru limbi și are următoarele coloane

- idParfum: O cheie străină de tip int care referențiază coloana idParfum din tabelul „Parfum”
- language: O coloană de tip String care stochează limba pentru descriere. română – ro, engleză – en, spaniolă – es, franceză-fr).
- descriere_translation: O coloană de tip String care stochează descrierea parfumului într-o anumită limbă în funcție de coloana language
- **Cheia primară este compusă din coloanele idParfum și language**. Astfel, nu există în tabel mai multe intrări care au același aceeași limbă și același id pentru un parfum.

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.

- **Relație de tip “one-to-many” între tabelul “Parfum” și tabelul “Parfum_Translation".** Astfel, pentru o instanță de tipul ”Parfum”, există patru instanțe de tip ”Parfum_Translation”(aferente fiecărei limbi), dar pentru o instanță de tip ”Parfum_Translation” există o singură instanță de tip "Parfum".

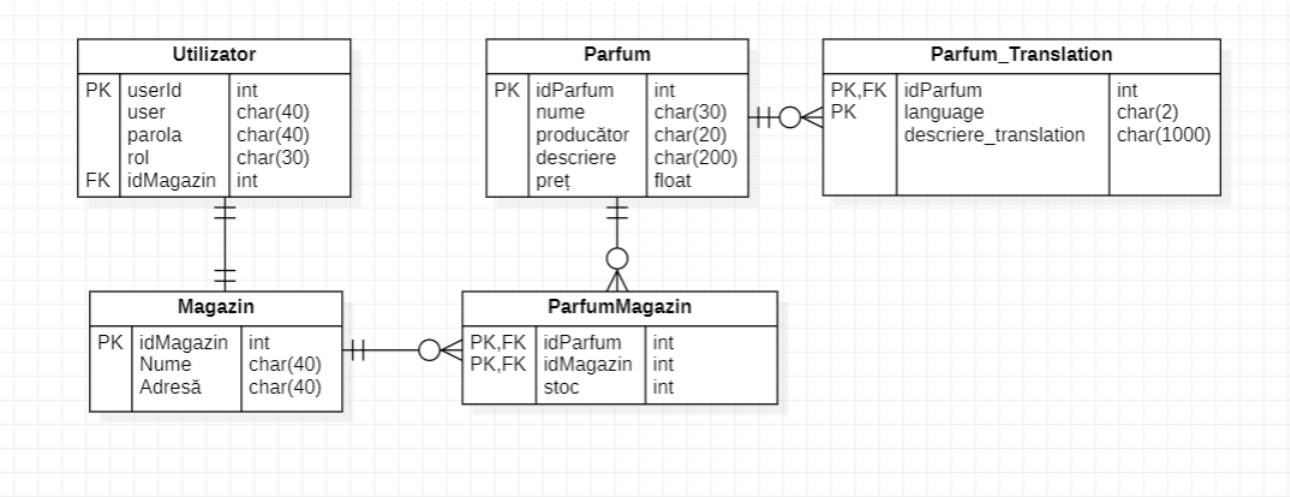

Fig. 3 - Diagrama Entitate-Relație

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 (angajat, manager sau administrator), iar un utilizator cu rolul de angajat are și un id de magazin asociat.

Gestionarea limbii în diferite interfețe

Limba este gestionată în aplicație prin intermediul claselor **Language (în Model)** și **LanguageController (în Controller).** Clasa Language se ocupă de gestionarea limbilor și resurselor asociate acestora, în timp ce LanguageController manipulează clasa Language pentru a schimba limba interfeței grafice.

Clasa Language extinde clasa **Observable din Java**, ceea ce înseamnă că orice modificări ale stării sale vor fi notificate observatorilor care se abonează la aceasta. Clasa Language gestionează o instanță de Locale și o instanță de ResourceBundle. Locale-ul curent reprezintă limba și regiunea selectată, iar ResourceBundle conține toate textele traduse pentru interfața grafică.

LanguageController se ocupă de gestionarea limbii în aplicație. Acesta conține o instanță de Language, pe care o instantiază și setează limba inițială pe română. Pentru a schimba limba, metoda changeLanguage este apelată cu un cod de limbă. Acesta setează noul Locale în clasa Language și notifică observatorii cu privire la schimbarea limbii.

În main, este creată o instanță de Language și apoi este pasată unui LoginController. Acest lucru este valabil și pentru celelalte controllere (AngajatController, ManagerController și AdministratorController), care primesc de asemenea o instanță de LanguageController. Astfel, fiecare controller are posibilitatea de a schimba limba interfeței grafice.

Aplicația folosește **fișiere de tip properties** pentru gestionarea textelor interfeței în cele patru limbi (română, engleză, franceză și spaniolă). Aceste fișiere properties conțin chei și valori corespunzătoare textelor traduse pentru fiecare limbă. ResourceBundle este folosit pentru a încărca aceste fișiere properties în funcție de limba selectată.

De exemplu, în clasa LoginController se adaugă listener-ele pentru butoanele de schimbare a limbii. Atunci când unul dintre aceste butoane este apăsat, metoda changeLanguage este apelată cu noul Locale corespunzător limbii selectate. Această metodă apelează changeLanguage din LanguageController pentru a seta noul Locale și a notifica observatorii despre schimbare. După ce limba este schimbată, metoda updateUIComponents din LoginController este apelată pentru a actualiza componentele UI (etichete, butoane etc.) cu noile texte traduse, utilizând ResourceBundle. Acest proces este similar în celelalte controllere (AngajatController, ManagerController și AdministratorController), asigurând astfel o gestionare coerentă a limbilor în întreaga aplicație.

Prin utilizarea fișierelor properties și a clasei ResourceBundle, aplicația poate gestiona eficient textele traduse și schimbarea limbii interfeței grafice într-un mod organizat și ușor de întreținut.

Prin intermediul LanguageController, fiecare controller poate gestiona și schimba limba interfeței grafice, asigurând o funcționalitate coerentă în întreaga aplicație. Atunci când limba este schimbată, toți observatorii sunt notificați, iar interfața grafică este actualizată în mod corespunzător.

Înregistrarea utilizatorului

Pachetul View conține clasa LoginView.
În clasa **LoginView** care extinde clasa JFrame, avem următoarele metode:

- **public LoginView(Locale locale):** constructorul clasei, primește un obiect Locale și inițializează interfața grafică a ferestrei de autentificare.
- **public ResourceBundle getResourceBundle():** returnează obiectul ResourceBundle curent.
- **public void updateResourceBundle(Locale locale):** actualizează obiectul ResourceBundle cu noul Locale primit ca parametru.
- **private void initialize():** inițializează componentele interfeței grafice, setează dimensiunile, culorile și adaugă panourile în fereastra principală.
- **public JButton getBtnEnglish():** returnează butonul pentru schimbarea limbii în engleză.
- **public JButton getBtnRomanian():** returnează butonul pentru schimbarea limbii în română.
- **public JButton getBtnFrench():** returnează butonul pentru schimbarea limbii în franceză.
- **public JButton getBtnSpanish():** returnează butonul pentru schimbarea limbii în spaniolă.
- **public JButton getBtnLogin():** returnează butonul de autentificare.
- **public JButton getBtnCancel():** returnează butonul de anulare.
- **public JTextField getUser():** returnează câmpul de text în care utilizatorul introduce numele de utilizator.
- **public JPasswordField getPassword():** returnează câmpul de text în care utilizatorul introduce parola.
- **public JLabel getLbUser():** returnează eticheta pentru numele de utilizator.
- **public JLabel getLbPassword():** returnează eticheta pentru parola.
- **public void mesajEroare():** afișează un mesaj de eroare atunci când autentificarea eșuează.

Aceste metode permit crearea și gestionarea interfeței grafice pentru fereastra de autentificare, oferind posibilitatea de schimbare a limbii și afișarea mesajelor de eroare în cazul unui eșec în procesul de autentificare.

Clasa LoginController din Pachetul Controller include următoarele metode:

- **public LoginController(Language language):** constructorul clasei, primește un obiect Language, creează obiecte pentru LoginView, UtilizatorPersistent și LanguageController, și adaugă ascultători pentru evenimentele butoanelor (schimbarea limbii, autentificare, anulare).
- **public void initialize():** setează fereastra de autentificare ca vizibilă.
- **public void autentificare():** verifică dacă numele de utilizator și parola introduse sunt corecte și, în funcție de rolul utilizatorului (administrator, manager, angajat), afișează apelează controller-ul corespunzător responsabil de crearea interfeței grafice. Dacă autentificarea eșuează, afișează un mesaj de eroare.
- **private void changeLanguage(Locale locale):** schimbă limba curentă, actualizează obiectul ResourceBundle și actualizează componentele interfeței grafice cu noile șiruri de caractere pentru limba aleasă.
- **private void updateUIComponents():** folosește obiectul ResourceBundle pentru a actualiza etichetele, butoanele și alte componente ale interfeței grafice cu șirurile de caractere corespunzătoare limbii selectate.

Aceste metode permit gestionarea evenimentelor generate de interacțiunea utilizatorului cu fereastra de autentificare, precum și actualizarea interfeței grafice în funcție de limba aleasă. În timpul procesului de autentificare, utilizatorul introduce numele de utilizator și parola în interfața grafică. Aceste informații sunt preluate de clasa LoginController, 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 Controller. 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.

- **public AngajatView(String user, int idMagazin):** constructorul clasei, primește numele de utilizator și ID-ul magazinului, și setează titlul ferestrei.
- **public JFrame getAddParfumFrame():** returnează o instanță a ferestrei de adăugare a parfumului, sau o creează dacă nu există.
- **public void showAddParfumFrame():** creează și afișează fereastra de adăugare a unui nou parfum, cu toate componentele necesare.
- **public void disposeAddParfumFrame():** închide fereastra de adăugare a parfumului.
- **public Object[] getRowData(JTable table, int row):** returnează datele dintr-un rând al unui tabel într-un obiect de tip Object[].
- **public void showMessageDialog(String message):** afișează un mesaj informativ într-o fereastră de dialog.
- **public void displayParfumList(Object[][] parfumList):** actualizează tabela cu parfumuri cu noile date.
- **public void showPerfumeDetails(String title, String perfumeDetails):** afișează detalii despre un parfum într-o fereastră de dialog, cu un titlu specificat și un șir de caractere ce conține informațiile despre parfum.
- De asemenea, sunt definite **17 metode de tip get pentru obținerea diferitelor componente ale interfeței grafice, precum butoane, etichete, câmpuri de text și altele.** Aceste metode vor fi apelate în controller.

În pachetul Controller este definită clasa AngajatController cu următoarele metode:

- **AngajatController():** Constructorul inițializează parfumPersistent, parfumMagazinPersistent și languageController.
- **setView(AngajatView view):** Atribuie obiectul view și apelează metoda setupListeners().
- **changeLanguage(Locale locale):** Actualizează limba aplicației în funcție de obiectul Locale și actualizează componentele vizuale.
- **showView(String user, int idMagazin):** Afișează fereastra pentru angajați, inițializează AngajatView și setează listeneri pentru butoanele de schimbare a limbii. De asemenea, afișează lista de parfumuri și lista de producători.
- **updateUIComponents():** Actualizează etichetele și textele pentru toate componentele vizuale în funcție de ResourceBundle.
- **updateAddParfumFrameLabels():** Actualizează etichetele și textele pentru componentele din fereastra "Adaugă parfum".
- **addSubmitButtonListener(ActionListener listener):** Adaugă un ActionListener pentru butonul de trimitere a formularului de adăugare a parfumului.
- **setupListeners():** Configurează listeneri pentru butoanele de acțiuni, cum ar fi adăugarea, actualizarea, ștergerea parfumurilor și vizualizarea detaliilor parfumurilor.
- getSelectedProducator(): Returnează producătorul selectat în JComboBox, dacă este selectat unul.
- **addParfum():** Adaugă un parfum în baza de date.
- **updateParfum():** Actualizează un parfum existent în baza de date.
- **deleteParfum():** Șterge un parfum din baza de date.
- **showPerfumeDetails():** Afișează detaliile unui parfum selectat.
- **addNewParfum():** Adaugă un nou parfum în baza de date și închide fereastra de adăugare a parfumului.
- **getFormattedPerfumeDetails(Object[] rowData) -** formatează detaliile unui parfum pentru a fi afișate într-un mesaj, folosind resursele pentru localizare.
- **getIdParfum(), getStoc(), getNewIdParfum(), getNewNume(), getNewProducator(), getNewPret(), getNewDescriere() și getNewStoc()** - aceste metode obțin informațiile introduse de utilizator în câmpurile de text pentru id, stoc și detaliile unui nou parfum.
- **getPriceMin() și getPriceMax() -** obțin limitele de preț introduse de utilizator pentru filtrarea parfumurilor.
- **getSelectedParfumId() -** returnează ID-ul parfumului selectat în tabel.
- **isDisponibilSelected() -** verifică dacă opțiunea pentru disponibilitate este bifată.
- **handleFilterParfums()** - gestionează filtrarea parfumurilor în funcție de criteriile introduse de utilizator.
- **getLocalizedMessage(String key)** - returnează un mesaj localizat folosind resursele de localizare.
- **displayParfumList()** - afișează lista de parfumuri în funcție de setările de localizare ale aplicației.
- **displayProducators() -** afișează lista de producători de parfumuri.
- **updateParfumList(List<ParfumMagazin> parfumMagazinList)** - actualizează lista de parfumuri afișată în tabel cu noile valori.

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

ManagerView este o clasă care se ocupă cu aspectul vizual și interacțiunea cu utilizatorul în cadrul aplicației. Metodele din această clasă:

- **ManagerView(String user):** Constructorul clasei, inițializează interfața grafică și setează numele utilizatorului.
- **initUI():** Inițializează interfața grafică și setează aspectul vizual al componentelor.
- **showMessageDialog(String message):** Afișează un mesaj primit ca argument într-o fereastră de dialog.
- **showPerfumeDetailsByName(String perfumeDetails):** Afișează detalii despre un parfum pe baza numelui său.
- **showPerfumeDetails(String title, String perfumeDetails):** Afișează detalii despre un parfum, primind ca argumente titlul și detaliile acestuia.
- **displayParfumList(Object[][] parfumList):** Afișează o listă de parfumuri în tabelul din interfața grafică.
- De asemenea, sunt definite **metode de tip getter** pentru obținerea diferitelor componente ale interfeței grafice, precum butoane, etichete, câmpuri de text și altele. Aceste metode vor fi apelate în controller: getComboBoxMagazin(), getComboBoxSortarePret(), getComboBoxSortareNume(), getTextFieldCautaDupaNume(), getBtnCauta(), getLblMagazin(), getLblSortarePret(), getLblSortareNume(), getLblCautaDupaNume(), getBtnViewDetails(), getTextFieldPretMin(), getTextFieldPretMax(), getComboBoxProducator(), getChckbxDisponibil(), getBtnFiltreaza(), getTable(), getLblPretMinim(), getLblPretMaxim(), getLblProducator(), getLblDisponibilitate(), getBtnEnglish(), getBtnRomanian(), getBtnFrench(), getBtnSpanish().

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

- **Constructorul ManagerController() -** creează obiecte pentru ParfumPersistent, MagazinPersistent și LanguageController.
- **setView(ManagerView view)** - stabilește ManagerView pentru acest controller și apelează metoda setupListeners().
- **changeLanguage(Locale locale)** - schimbă limba aplicației și actualizează componentele interfeței în funcție de limba selectată.
- **updateUIComponents() -** actualizează componentele vizuale ale aplicației în funcție de ResourceBundle curent.
- **showView(String user)** - afișează fereastra ManagerView, adaugă ActionListeners pentru butoanele de schimbare a limbii și inițializează componentele ferestrei.
- **setupListeners() -** stabilește ActionListeners pentru componentele ManagerView (ComboBox-uri, butoane etc.) și definește logica acestora.
- **getLocalizedMessage(String key)** - returnează un mesaj localizat în funcție de cheia specificată.
- **displayMagazinNames()** - afișează numele magazinelor în comboBoxMagazin și actualizează lista de parfumuri afișată.
- **getFormattedPerfumeDetails(Object[] rowData)** - returnează un vector cu titlul și detalii despre parfumul selectat, formatate în funcție de ResourceBundle.
- **isSortByPriceAscending():** Verifică dacă sortarea produselor după preț este în ordine crescătoare, returnând true dacă este, și false altfel.
- **isSortByNameAscending():** Verifică dacă sortarea produselor după nume este în ordine crescătoare, returnând true dacă este, și false altfel.

- **updateComboBoxSortarePret():** Actualizează conținutul JComboBox-ului pentru sortarea produselor după preț, adăugând opțiunile pentru ordine crescătoare și descrescătoare.
- **updateComboBoxSortareNume():** Actualizează conținutul JComboBox-ului pentru sortarea produselor după nume, adăugând opțiunile pentru ordine crescătoare și descrescătoare.
- **displayProducators():** Afișează lista de producători în JComboBox-ul pentru filtrarea produselor după producător, și apoi afișează lista de produse.
- **getSelectedProducator():** Returnează numele producătorului selectat în JComboBox-ul pentru filtrarea produselor după producător, sau un șir gol dacă nu este selectat niciun producător.
- **getPriceMin(), getPriceMax():** Returnează prețul minim, respectiv prețul maxim, introdus în câmpurile corespunzătoare.
- **getEnteredParfumName():** Returnează numele parfumului introdus în câmpul de căutare după nume.
- **isDisponibilSelected():** Verifică dacă opțiunea pentru a afișa doar produsele disponibile în stoc este selectată.
- **handleFilterParfums():** Găsește și afișează lista de produse care îndeplinesc criteriile de filtrare ale utilizatorului.
- **handleSortParfumsByName(), handleSortParfumsByPrice():** Sortează lista de produse după nume, respectiv după preț, în funcție de opțiunile selectate de utilizator și afișează lista actualizată.
- **initializeSortingLabels():** Inițializează etichetele pentru sortarea produselor după nume.
- **handleSearchParfumByName():** Caută un parfum după nume și afișează detaliile acestuia într-un mesaj informativ, sau afișează un mesaj că parfumul nu a fost găsit.
- **getMagazinIdByName():** Returnează ID-ul magazinului în funcție de numele magazinului.
- **displayParfumList():** Afișează lista de parfumuri din magazinul selectat.
- **updateParfumList():** Actualizează lista de parfumuri afișată în tabel pe baza listei de obiecte ParfumMagazin primite ca parametru.

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 ManagerController:

- **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 clasa AdministratorView permite administratorului să gestioneze utilizatorii și să schimbe limba interfeței. Metodele principale ale clasei AdministratorView**:**

- **createLanguageButtonsPanel:** Această metodă creează un panel cu butoane pentru schimbarea limbii interfeței. Butoanele disponibile sunt pentru engleză (EN), română (RO), franceză (FR) și spaniolă (ES).
- **initialize:** Această metodă inițializează componentele interfeței și setează layout-ul și stilul acestora. Aici sunt create și plasate etichete, câmpuri de text, combobox-uri și butoane într-un layout de tip BorderLayout.
- **displayUsers:** Această metodă afișează informațiile despre utilizatori într-un tabel (JTable) cu coloanele "ID user", "Username", "Parolă", "Rol" și "ID magazin".
- **showUserDetails:** Această metodă deschide o nouă fereastră pentru afișarea detaliilor unui utilizator. Detaliile sunt afișate într-un JTextArea în format text.
- **updateResourceBundle:** Această metodă actualizează pachetul de resurse pentru localizare în funcție de limba selectată.
- **setResourceBundle și getResourceBundle:** Aceste metode permit setarea și obținerea pachetului de resurse pentru localizare.
- **showMessageDialog:** Această metodă afișează un mesaj într-o fereastră modală de dialog.
- **getRowData:** Această metodă returnează datele unui rând din tabelul de utilizatori, în funcție de indicele rândului furnizat.
- **Metode de tipe getter pentru diverse componente ale interfeței: Aceste metode returnează referințe la componentele interfeței, cum ar fi butoanele de limbă, câmpurile de text pentru ID-ul utilizatorului, numele de utilizator, parola, rolul și ID-ul magazinului, precum și butoanele pentru adăugarea, ștergerea, actualizarea și vizualizarea detaliilor utilizatorilor.**

În pachetul Controller clasa AdministratorController conține metodele și logica pentru gestionarea interacțiunilor cu AdministratorView.

- **AdministratorController():** Constructorul clasei care inițializează obiectele utilizatorPersistent și languageController.
- **setView(AdministratorView view):** Metoda care setează instanța curentă a view și apelează setupListeners().
- **showView(String user):** Metoda pentru afișarea ferestrei AdministratorView și înregistrarea listenerilor pentru schimbarea limbii și afișarea listei de utilizatori.
- **changeLanguage(Locale locale):** Metoda care schimbă limba și actualizează componentele UI.
- **updateUIComponents():** Actualizează etichetele și traducerile componentelor UI în funcție de limba selectată.
- **updateRoleComboBox(String[] translatedRoles):** Actualizează combobox-ul cu rolurile traduse.
- **getUserId(), getUserName(), getPassword(), getRol() și getIdMagazin():** Metode pentru a obține informații despre utilizator din câmpurile ferestrei.
- **getSelectedUserId():** Returnează ID-ul utilizatorului selectat în tabel sau -1 dacă nu există niciunul selectat.
- **getLocalizedMessage(String key):** Returnează mesajul localizat pentru o cheie specifică.
- **setupListeners():** Configurați listenerii pentru butoanele de adăugare, ștergere, actualizare și vizualizare a detaliilor utilizatorilor.

- **addUser():** Metoda pentru adăugarea unui nou utilizator în baza de date și actualizarea listei de utilizatori.
- **updateUser():** Metoda pentru actualizarea informațiilor unui utilizator existent și actualizarea listei de utilizatori.
- **deleteUser():** Metoda pentru ștergerea unui utilizator și actualizarea listei de utilizatori.
- **displayUserList():** Metoda pentru a afișa lista de utilizatori în tabel.

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 dezvoltată ilustrează eficiența modelului MVC (Model-View-Controller) care separă logica aplicației în trei componente: clasele din Model se ocupă de interacțiunile cu baza de date, asigurând un mod eficient și corect de a opera cu datele, clasele din View oferă o experiență intuitivă și ușor de utilizat, iar clasele din Controller gestionează comunicarea între View și Model.

Aplicația include suport pentru localizare și internaționalizare, permițând traducerea interfeței în mai multe limbi. Aceasta caracteristică evidențiază atenția acordată detaliilor și preocuparea pentru accesibilitatea aplicației în contextul unui public internațional divers.

În concluzie, aplicația reprezintă o implementare solidă a modelului MVC, care facilitează dezvoltarea ulterioară și adaptabilitatea. Prin separarea logicii de business de interacțiunea cu utilizatorul și gestionarea datelor în componente bine definite, aplicația se bucură de o organizare clară și de o flexibilitate sporită, demonstrând puterea modelului MVC în dezvoltarea de aplicații robuste și ușor de întreținut.

Bibliografie

Alur, D., Crupi, J., & Malks, D. (2003). Core J2EE Patterns: Best Practices and Design Strategies. Prentice Hall PTR.

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/

Gascoigne, B. (2011). Twitter: A Case Study on the Application of the Model-View-Controller Design Pattern. Retrieved from https://bernardodamele.blogspot.com/2011/09/twitter-case-study-on-application-of.html

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

Leff, A., & Rayfield, J. T. (2001). Web-application development using the Model/View/Controller design pattern. Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference. IEEE.

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. Object Mentor.

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.

Reenskaug, T. (1979). Thing-Model-View-Editor, an Example from a Planning System. Xerox PARC Technical Note.

RedMonk. (2021). The RedMonk Programming Language Rankings: January 2021.

Shaout, A., & Al-Mutlaq, I. (2016). A Model-View-Controller (MVC) Framework for Android Applications. Journal of Emerging Trends in Computing and Information Sciences, 7(7), 344-353.

The Java Tutorials. (2021). Oracle. Retrieved from https://docs.oracle.com/javase/tutorial/

The Java Virtual Machine Specification. (2021). Oracle. Retrieved from https://docs.oracle.com/javase/specs/jvms/se16/html/

Venners, B. (2000). Inside the Java Virtual Machine. McGraw-Hill Education.

W3Techs. (2021). Usage of programming languages for websites. https://w3techs.com/technologies/overview/programming_language/all

Wikipedia. (2021). Model–view–controller. Retrieved from https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

Zhang, J., Zhang, M., & Zhang, X. (2020). The development and design of the electronic commerce platform based on Java. Journal of Physics: Conference Series, 1565(4), 042039.