UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Aplicație pentru gestionarea unui lanț de magazine de parfumuri

Arhitectura Client-Server

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:

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

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.
- ❖ Salvare liste cu situația parfumurilor din lanțul de parfumerii în mai multe formate: csv, json, xml, txt;
- ❖ Vizualizarea unor statistici legate de produsele din lanțul de parfumerii utilizând grafice (structură radială, structură inelară, de tip coloană, etc.).

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

- ❖ Operații CRUD pentru informațiile legate de utilizatori;
- ❖ Vizualizarea listei tuturor utilizatorilor și filtrarea acesteia după tipul utilizatorilor

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

2. Introducere

Arhitectura este o structură de aplicație distribuită care împarte sarcinile între furnizorii de resurse sau servicii, numiți servere, și solicitanții de servicii, numiți clienți. Acest model poate fi utilizat în cadrul unei rețele locale sau chiar pe un singur calculator, unde clientul și serverul sunt aplicații separate.

- **Client:** Clientul este o aplicație web sau desktop care rulează pe un computer personal sau pe o stație de lucru și care se bazează pe un server pentru a efectua anumite operațiuni. Clientul trimite o cerere către un server pentru a utiliza o resursă. Poate solicita o cerere de date specifice, o cerere de adăugare a unei înregistrări într-o bază de date sau o cerere de resurse de calcul.
- **Server:** Serverul este un program de calculator sau un proces care gestionează accesul la o resursă sau un serviciu centralizat. Serverul așteaptă solicitările primite de la clienți și răspunde la acestea, îndeplinind cererea clientului.
- **Comunicare:** Clientul și serverul comunică, de obicei, folosind un protocol agreat atât de client, cât și de server. Această comunicare poate avea loc prin intermediul unei rețele sau prin comunicare între procese pe o singură mașină.

Modelul este utilizat pe scară largă în aplicațiile desktop sau web deoarece este versatil și eficient. El permite centralizarea resurselor, simplificând întreținerea și actualizările. De asemenea, permite clienților să fie mai simpli, deoarece aceștia pot descărca procesarea grea pe servere.

Cu toate acestea, un potențial dezavantaj al acestui model este că se bazează foarte mult pe server. În cazul în care serverul nu funcționează, clienții nu pot accesa resursele pe care acesta le oferă. De asemenea, dacă prea mulți clienți solicită resurse simultan, serverul poate fi supraîncărcat. Acest aspect poate fi atenuat prin diverse strategii, cum ar fi echilibrarea încărcăturii, redundanța și sistemele de redistribuire.

Avantajele modelului

În ceea ce privește beneficiile utilizării unui model pentru aplicațiile desktop, mai multe studii evidențiază următoarele avantaje:

Control centralizat: Unul dintre avantajele cheie ale modelului este controlul centralizat al datelor și resurselor, ceea ce duce la o mai bună integritate și coerență a datelor (Furht, B., & Escalante, A., 2010).

Scalabilitate: Modelul poate găzdui un număr din ce în ce mai mare de clienți fără o degradare semnificativă a performanței. Acest lucru îl face potrivit pentru aplicațiile care trebuie să susțină o bază de utilizatori în creștere (Pautasso, C., Zimmermann, O., & Leymann, F., 2008).

Flexibilitate și interoperabilitate: Modelul permite ca diferite tipuri de clienți să interacționeze cu serverul, ceea ce poate fi benefic în mediile informatice eterogene (Papazoglou, M. P., & Georgakopoulos, D., 2003).

Securitate îmbunătățită: Deoarece datele sunt gestionate la nivel central, implementarea protocoalelor de securitate și monitorizarea accesului la date se poate face mai eficient, ceea ce duce la o securitate sporită a datelor (Stallings, W., & Tahiliani, M. P., 2018).

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

- ***Eficiența în IoT (Internet of Things) și Edge Computing***: Studiile au arătat aplicarea modelului în IoT, punând accentul pe utilizarea edge computing. Dispozitivele edge (clienții) trimit date către serverele centralizate pentru procesare și analiză. Acest model este utilizat pentru **a reduce latența și a îmbunătăți securitatea în aplicațiile IoT.** (Li et al., 2019).

- ***Cloud Computing***: Modelul este un concept fundamental în cloud computing. Clienții (dispozitivele utilizatorilor) trimit cereri către servere (furnizorii de servicii cloud) care procesează cererile și trimit înapoi răspunsurile. Cercetarea în acest domeniu se concentrează pe **îmbunătățirea eficienței, securității și confidențialității în serviciile cloud**. Yui et al., 2016).

- ***Arhitectura microserviciilor:*** Arhitectura modernă de software a evoluat pentru a utiliza o versiune a modelului sub forma microserviciilor. Fiecare microserviciu poate fi văzut ca un server care oferă o anumită funcționalitate. Clienții, în acest caz, pot fi alte microservicii sau aplicații orientate către utilizator. Cercetările din acest domeniu se concentrează pe **gestionarea comunicării între servicii, pe coerența datelor și pe toleranța la erori**.(Dragoni et al., 2017)

- ***Big Data Analytics:*** Arhitectura este, de asemenea, crucială în big data analytics. Clienții trimit sarcini de procesare a datelor către serverele care au acces la big data. Serverele execută sarcinile și trimit înapoi rezultatele. Acest model permite manipularea unor volume mari de date care nu ar fi fezabile pe o singură mașină. (Chen et al., 2014)

- ***Aplicații mobile:*** Modelul este esențial pentru multe aplicații mobile. Clientul (aplicația mobilă) trimite cereri către server (serviciul back-end), care procesează cererea și trimite înapoi un răspuns. Cercetarea în acest domeniu include **îmbunătățirea eficienței transmiterii datelor, reducerea latenței și îmbunătățirea securității**.(Sharafeddine & Hajj, 2016).

3. Instrumente utilizate

Instrumentele utilizate în proiectarea și dezvoltarea aplicației au fost alese pe baza eficienței și a compatibilității lor cu modelul Client-Server 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. 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 Client-Server (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 Client-Server, 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 aplicația Server din modelul Client-Server.

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).

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 Client-Server, 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 Client-Server. 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 arhitectura Client- Server** este recomandată din mai multe motive:

- **Independența de platformă:** Java urmează principiul *"Scrie o dată, rulează oriunde"* **(**"Write Once, Run Anywhere."). Astfel, codul Java poate rula pe orice platformă care suportă o mașină virtuală Java (JVM), ceea ce face ca distribuția aplicației client/server să fie mult mai ușoară pe diferite platforme.

- **Securitate:** Java dispune de caracteristici de securitate integrate, cum ar fi autentificarea avansată și controlul accesului, ceea ce îl face o alegere bună pentru dezvoltarea de aplicații sigure.

- **Multi-threading**: Java dispune de suport încorporat pentru multi-threading, care este o caracteristică esențială pentru dezvoltarea de aplicații eficiente și de înaltă performanță.

- **Networking:** Java dispune de un pachet de rețea puternic (java.net), care oferă un suport excelent pentru dezvoltarea de aplicații în rețea, inclusiv programarea socket, care este fundamentală în dezvoltarea de aplicații .

- **Programarea orientată pe obiecte:** Java este un limbaj de programare orientat pe obiecte, care încurajează utilizarea de componente modulare și reutilizabile (Gupta & Sharma, 2021). Astfel, natura orientată pe obiecte ajută la menținerea sistemului modular, flexibil și extensibil necesar arhitecturii .

- **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 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 Client-Server, 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. Etapele de dezvoltare a aplicației informatice

Dezvoltarea aplicației pentru gestionarea unui lanț de parfumerii a fost dezvoltată în 3 etape:

1.În etapa de analiză, a fost creată o diagramă de cazuri de utilizare pentru a evidenția cerințele funcționale ale aplicației.

2. În etapa de proiectare, au fost elaborate două diagrame de clase care respectă principiile SOLID pentru aplicație Client, respecitv, pentru aplicația Server. În plus, a fost creată o diagramă entitate-relație pentru proiectarea bazei de date la care acces aplicația Server.

3. În etapa de implementare, a fost scris codul pentru îndeplinirea funcționalităților specificate în diagrama de cazuri de utilizare în limbajul de programare Java pornind de la diagramele de clase.

Etapa de analiză

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

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.

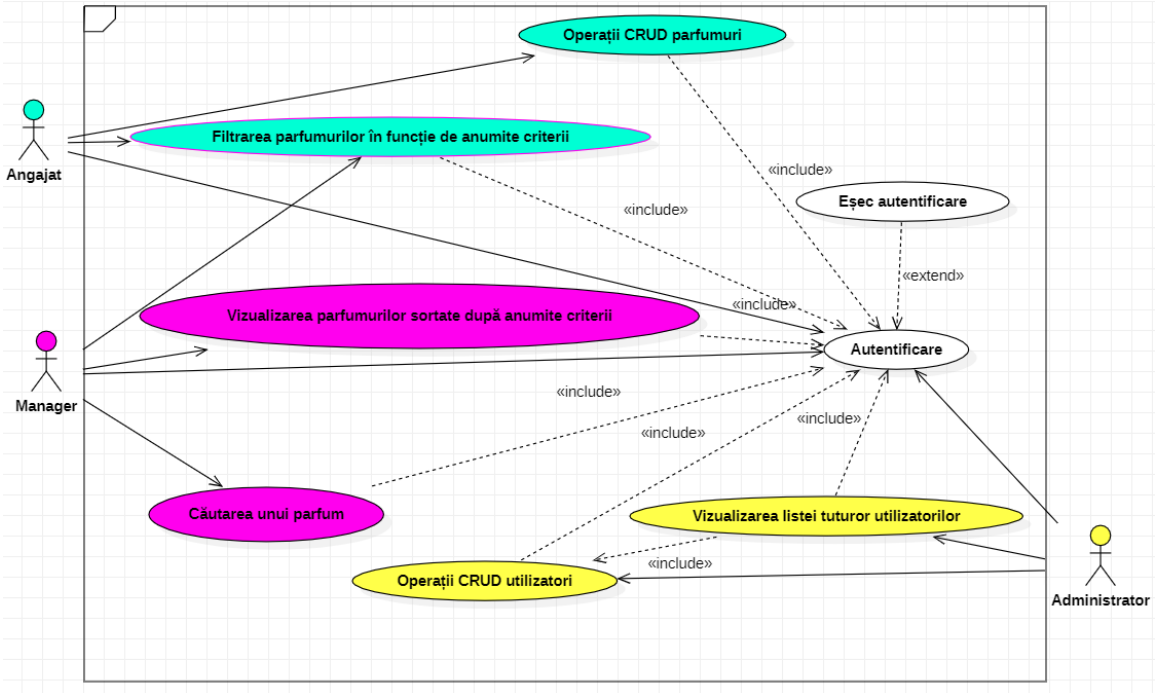


Fig. 1 - Diagrama cazurilor de utilizare

Etapa de proiectare

În etapa de proiectare a fost dezvoltată **diagrama de clasă pentru aplicația Client** (Fig. 2) și **diagrama de clasă pentru aplicația Server** (Fig. 3), precum și **diagrama entitate-relație** corespunzătoare bazei de date (Fig. 4).

Etapa de Proiectare pentru Aplicația Client

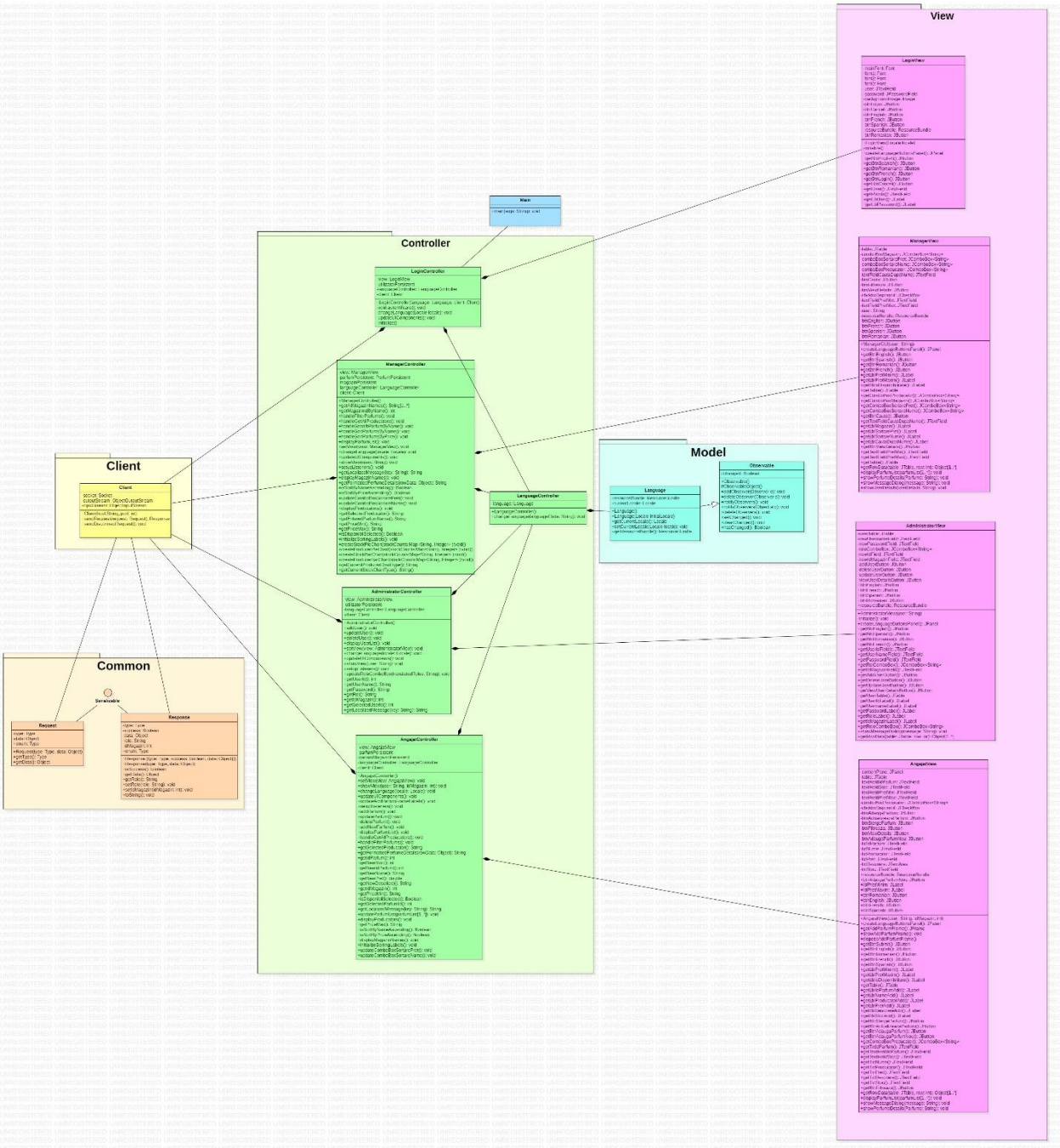


Fig. 2 Diagrama de clasă pentru aplicația Client

În diagrama de clasă a aplicației Client se pot observa pachetele View, Controller, Model, Client și Common.

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.
- **AngajatView:** Clasa AngajatView reprezintă componenta vizuală a funcționalităților angajatului. Acesta include elemente de interfață pentru afișarea și modificarea informațiilor despre parfumuri.
- **AdministratorView**: Clasa AdministratorView reprezintă componenta vizuală a funcționalităților administratorului. Acesta include elemente de interfață pentru gestionarea utilizatorilor.
- **ManagerView:** Clasa ManagerView reprezintă componenta vizuală a funcționalităților managerului. Acesta include elemente de interfață pentru vizualizarea informațiilor în legătură cu parfumuri din diferite maagzine.

Clasa Client din pachetul Client reprezintă clientul în modelul de comunicare . Obiectul Client se conectează la un server utilizând un anumit host și port, trimite cereri (requests) către server și primește răspunsuri (responses) de la acesta. Metoda sendRequest este utilizată pentru a trimite un obiect Request către server și a primi un obiect Response înapoi. Metoda sendDisconnectRequest este utilizată pentru a trimite o cerere specială de deconectare către server. Metoda close este utilizată pentru a închide conexiunea cu serverul.

Pachetul Common

- **Request: Clasa Request** reprezintă o cerere trimisă de la client la server. Aceasta conține un tip, care indică natura cererii (de exemplu, autentificare, deconectare, adăugare utilizator, actualizare utilizator etc.), și date, care reprezintă informațiile suplimentare necesare pentru a procesa cererea. Enumerația Type conține toate tipurile posibile de cereri care pot fi trimise.
- **Response: Clasa Response** reprezintă un răspuns trimis de la server la client. Acesta conține un tip, care indică natura răspunsului (de exemplu, rezultatul autentificării, rezultatul adăugării utilizatorului, etc.), un flag de succes care indică dacă operațiunea a fost realizată cu succes sau nu, și date, care reprezintă informațiile suplimentare legate de răspuns. Enumerația Type conține toate tipurile posibile de răspunsuri care pot fi trimise.

Clasele din pachetul Controller au rolul de a media interacțiunea dintre View și Server, facilitând comunicarea dintre acestea și gestionând logica de afișare și trimiterea request-urilor către Server, precum și trimiterea Response-urilor către view. Fiecare clasă din Controller își creează un Obiect de tip Client pentru a trimite request-uri către Server și pentru a primi response-uri de la acesta.

- **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. gestionează procesul de autentificare al utilizatorilor. Aceasta primește datele de autentificare de la utilizator prin intermediul view-ului, și apoi trimite un request către server pentru a verifica validitatea acestor date. În funcție de rezultatul obținut, utilizatorul este autentificat și se apelează AngajatController, AdministratorController sau ManagerController, în funcție de rolul returnat în Response-ul de la Server

- **AngajatController:** Clasa AngajatController gestionează funcționalitățile specifice angajaților în cadrul aplicației. Aceasta mediază interacțiunea dintre angajați și sistem, permițându-le să execute operațiuni specifice rolului lor, cum ar fi operații CRUD asupra parfumurilor din magazinul în care lucrează sau sortarea parfumurilor dintr-un magazin selectat în funcție de denumire sau preț
- **AdministratorController:** Clasa AdministratorController gestionează funcționalitățile specifice unui administrator al aplicației. Acesta gestionează acțiunile specifice administratorilor de sistem. Acesta le permite administratorilor să efectueze operațiuni CRUD în legătură cu utilizatorii aplicației sau filtrarea utilizatorilor în funcție de rol.
- **ManagerController:** Clasa ManagerController gestionează funcționalitățile specifice unui manager al magazinului. Aceasta mediază interacțiunea dintre manageri și sistem, permițându-le să execute operațiuni specifice rolului lor, cum ar fi sortarea parfumurilor dintr-un magazin selectat în funcție de denumire sau preț, vizualizarea unor statistici în legătură cu parfumurile din magazinul selectat sau descărcarea listei de parfumuri în diferite formate.

Clasa Language din pachetul Model 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().

Etapa de Proiectare pentru Aplicația Server

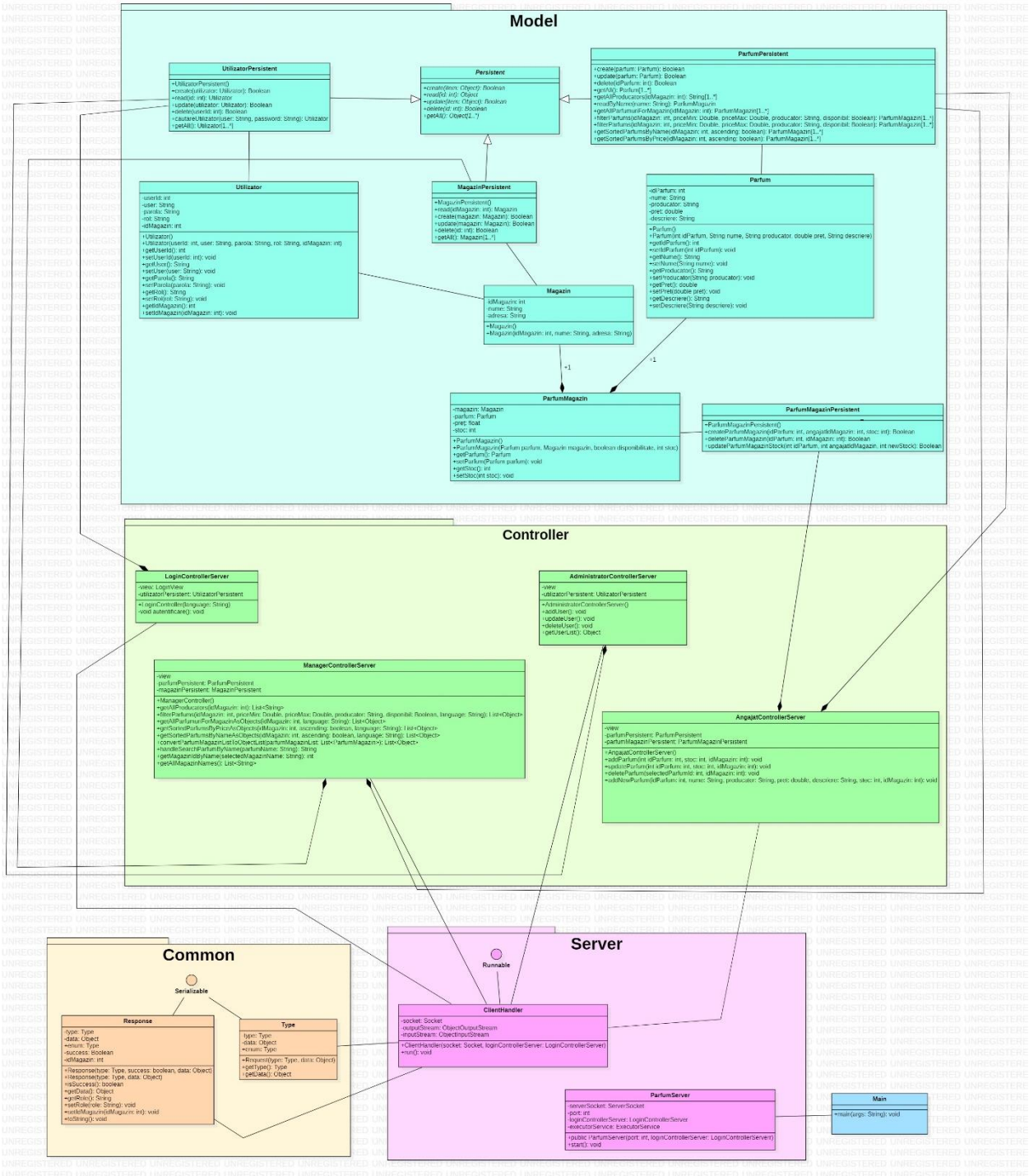


Fig. 3 Diagrama de clasă pentru aplicația Server

În diagrama de clasă a aplicației Server se pot observa pachetele Server, Controller, Model și Common.

Pachetul Server

- **ParfumServer:** Această clasă este serverul principal care așteaptă conexiunile de la clienți. Ea utilizează un obiect ServerSocket pentru a accepta conexiuni de la clienți. Când un client se conectează, serverul creează un nou obiect ClientHandler pentru a gestiona comunicarea cu clientul respectiv. Metoda start() este responsabilă pentru a porni serverul. În primul rând, serverul încearcă să obțină adresa IP locală, apoi să inițializeze ServerSocket cu portul specificat. După aceea, serverul intră într-o buclă infinită în care așteaptă clienții să se conecteze. Când un client se conectează, serverul creează un nou obiect ClientHandler, îl trimite la ExecutorService și revine pentru a aștepta următorul client. În cazul unei erori, excepția este afișată și ServerSocket este închis.
- **ClientHandler:** Această clasă este responsabilă pentru gestionarea comunicării cu un client specific. Este implementată ca un **Runnable,** ceea ce înseamnă că poate fi rulată ca un fir de execuție.Constructorul ClientHandler primește un socket și un obiect LoginControllerServer și inițializează fluxurile de intrare și de ieșire pentru a comunica cu clientul.Metoda run() implementează logica principală de comunicare cu clientul. În funcție de tipul cererii, ClientHandler poate efectua diverse operații, cum ar fi autentificarea unui utilizator, adăugarea, actualizarea sau ștergerea unui utilizator sau a unui parfum, sortarea sau filtrarea parfumurilor etc. prin apelarea metodelor din clasele din pachetul Controller. Dacă se produce o eroare, aceasta este afișată și resursele sunt închise. Metoda closeResources() închide fluxurile de intrare și de ieșire și socket-ul clientului.

Clasele din pachetul Common au fost descrise anterior.

Pachetul Controller

- **Clasa AdministratorControllerServer** implementează operațiile pe care le poate face un administrator. Aceasta include crearea, actualizarea și ștergerea unui utilizator, precum și obținerea unei liste cu toți utilizatorii. Toate aceste operații sunt realizate prin intermediul unei instanțe a clasei UtilizatorPersistent.
- **Clasa AngajatControllerServer** oferă funcționalitatea necesară pentru a un angajat să manipuleze parfumurile din sistem. Aceasta include adăugarea unui parfum nou, adăugarea unui parfum existent în magazinul în care lucrează, actualizarea stocului unui parfum și ștergerea unui parfum din magazinul la care lucrează.
- **Clasa LoginControllerServer** gestionează autentificarea unui utilizator. În constructorul său, se creează o nouă instanță a clasei UtilizatorPersistent. Există o metodă de autentificare care primește un nume de utilizator și o parolă și returnează un obiect Utilizator în cazul în care credențialele sunt corecte.
- **Clasa ManagerControllerServer** oferă funcționalități pentru un manager. Aceasta include obținerea tuturor producătorilor, filtrarea parfumurilor pe baza unor criterii, sortarea parfumurilor după preț sau nume, obținerea tuturor parfumurilor pentru un anumit magazin, căutarea unui parfum după nume și obținerea ID-ului unui magazin după nume. Toate aceste operații sunt realizate prin intermediul unei instanțe a claselor ParfumPersistent și MagazinPersistent.

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.

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".

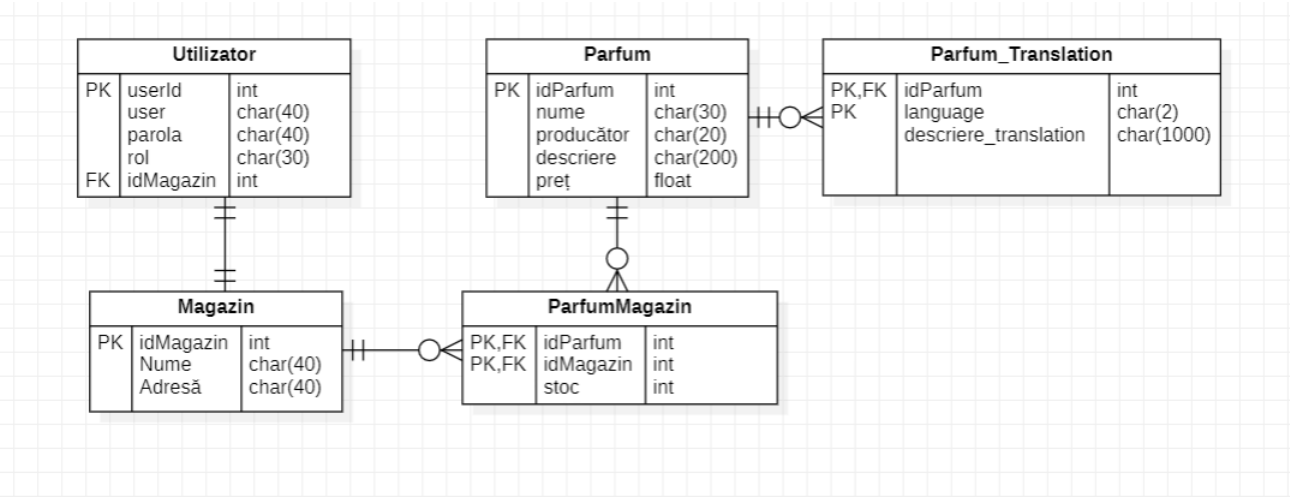


Fig. 4 - Diagrama Entitate-Relație

Etapa de implementare

În această etapă, au fost dezvoltate cele două aplicații (Aplicația Client și Aplicația Server) pentru gestiunea 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 corespunzător magazinului în care lucrează.

Internaționalizarea aplicației în patru limbi diferite

Limba este gestionată în aplicație prin intermediul claselor **Language (în pachetul Model din aplicația Client)** și **LanguageController (în pachetul Controller din aplicația Client).** 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.

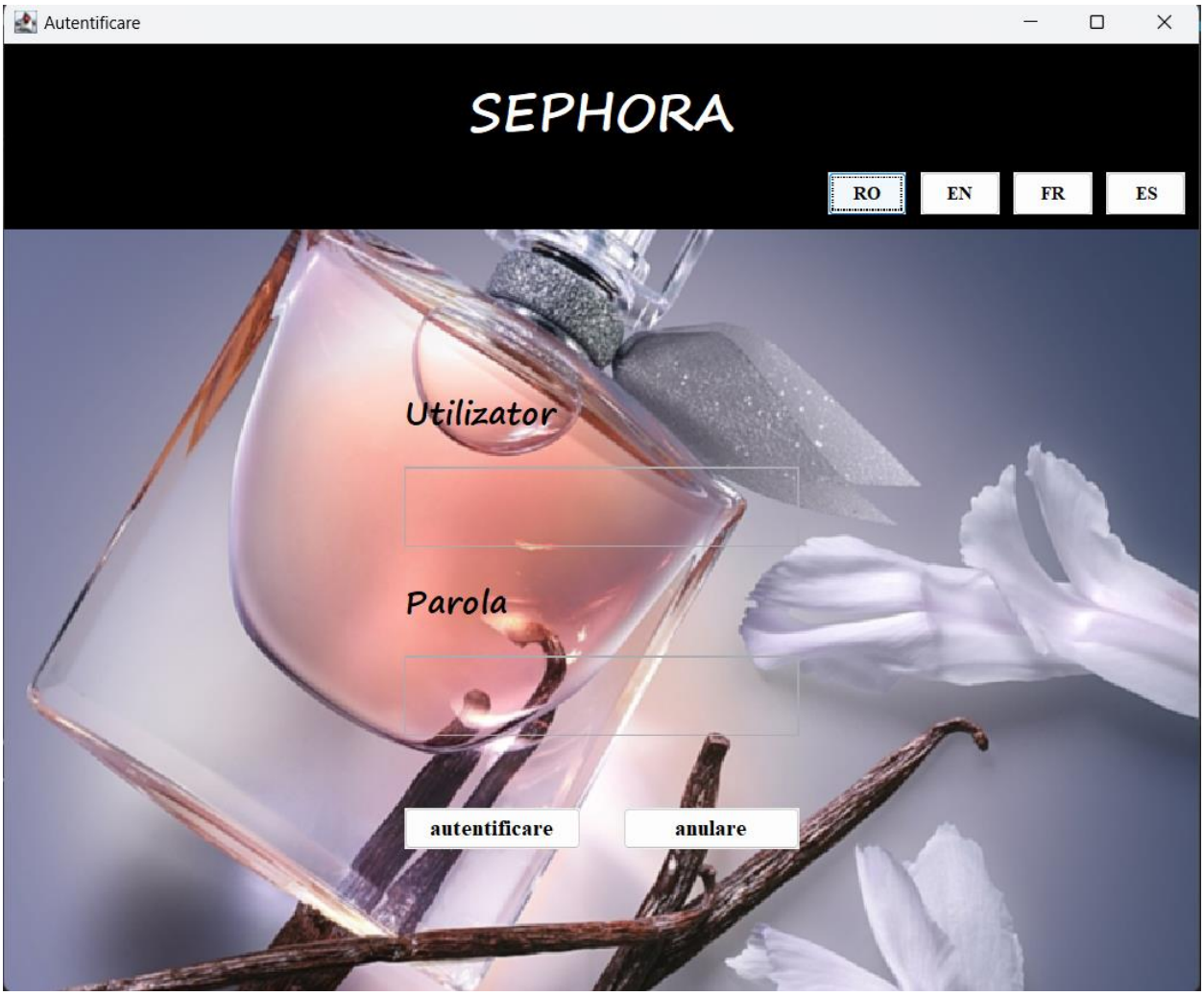
Aplicația folosește **fișiere de tip properties** pentru gestionarea textelor interfeței în cele patru limbi (română, engleză, franceză și spaniolă). Aceste fișiere properties conțin chei și valori corespunzătoare textelor traduse pentru fiecare limbă. ResourceBundle este folosit pentru a încărca aceste 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 din aplicația Client conține clasa LoginView care extinde clasa Jframe
Clasa LoginView:

- **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.
- **ublic 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 a aplicației Client:

- **public LoginController(Language language, Client client):** Acesta este constructorul pentru clasa LoginController. Acesta inițializează un nou LoginView, setează un nou LanguageController și asociază acțiuni cu butoanele din vizualizare.
- **public initialize():** Această metodă setează fereastra de autentificare ca vizibilă.
- **public autentificare():** Această metodă preia numele de utilizator și parola introduse, creează o **cerere de tip LOGIN** cu aceste date **și o trimite la server prin intermediul clientului**. În funcție de răspunsul primit, deschide o nouă fereastră corespunzătoare rolului utilizatorului (Administrator, Manager sau Angajat) sau afișează un mesaj de eroare în cazul în care autentificarea nu reușește.
- **changeLanguage(Locale locale):** Această metodă este responsabilă pentru schimbarea limbii în interfața grafică. Aceasta schimbă limba în LanguageController și actualizează ResourceBundle în LoginView.
- **updateUIComponents():** Această metodă actualizează componentele interfeței grafice (butoane, etichete etc.) cu șirurile de caractere corespunzătoare din noul limbaj selectat. 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 trimite request-uri către Server. 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.

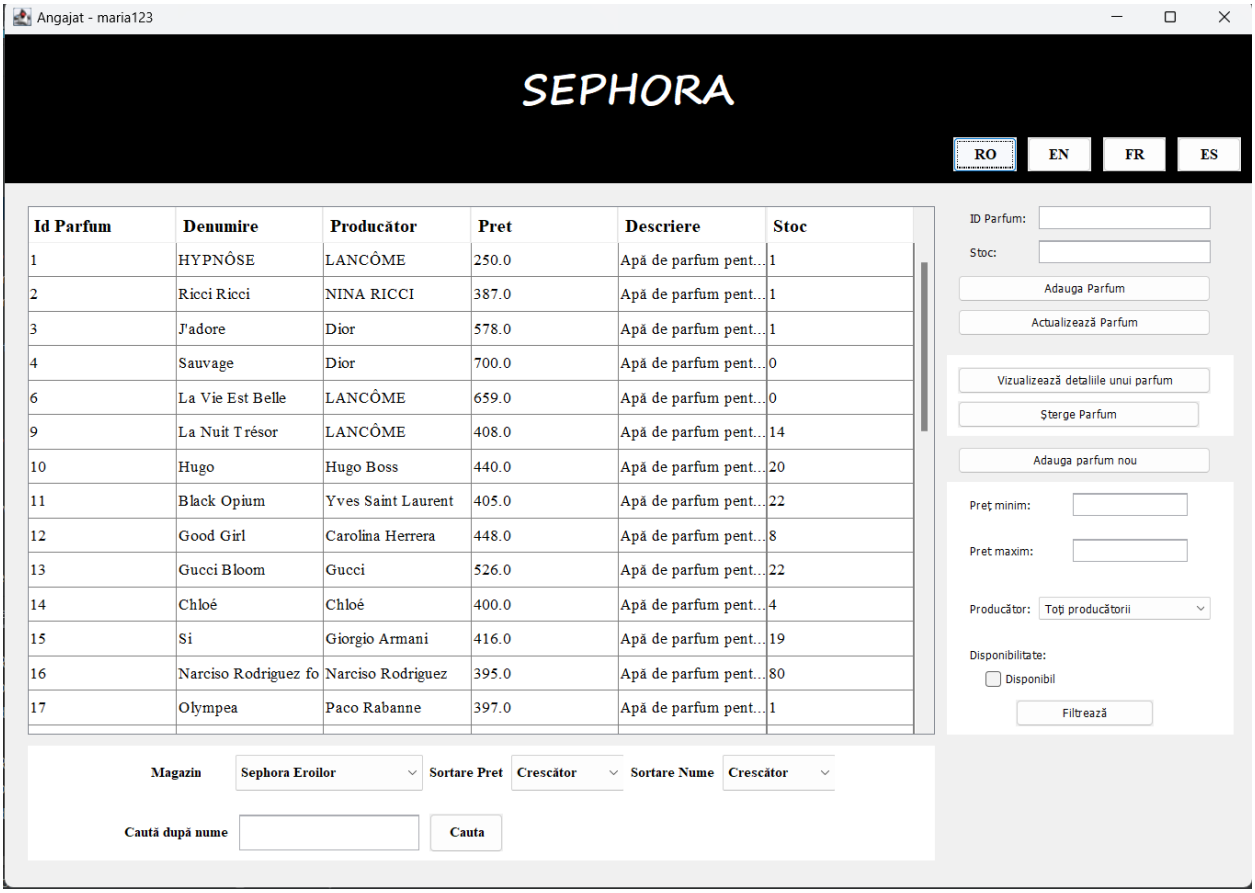
Cererea generată de Clientul din LoginController este gestionată de metoda **run()** din clasa **ClientHandler (CASE_LOGIN)** a pachetului **Server** din aplicația Server care apelează metoda cautareUtilizator(String user, String password) din Pachetul Model al aplicației Server.

Pachetul Model din Aplicația Server 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.



Pachetul View din aplicația Client

- **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 a aplicației Client este definită clasa AngajatController cu următoarele metode:

- **public AngajatController(Client client):** Inițializează un obiect AngajatController cu un client dat ca parametru și creează un controller de limbă (LanguageController).
- **public setView(AngajatView view):**Setează o instanță de AngajatView pentru acest controller și apelează metoda setupListeners().
- **public void changeLanguage(Locale locale):**Schimbă limba aplicației și actualizează componentele vizuale corespunzătoare. Această metodă apelează metoda changeLanguage(locale.getLanguage()) din LanguageController, actualizează ResourceBundle-ul pentru view și apelează metodele updateUIComponents(), displayProducators() și displayParfumList().
- **public void showView(String user, int idMagazin):**Afișează interfața utilizatorului angajat, inițializează view-ul cu o nouă instanță de AngajatView cu utilizatorul și id-ul magazinului specificate.
- **public void updateUIComponents():**Actualizează componentele vizuale ale aplicației în funcție de ResourceBundle și obține ResourceBundle-ul curent din view.Astfel, actualizează titlurile, etichetele și textele butoanelor din view cu valorile corespunzătoare din ResourceBundle.
- **public void updateAddParfumFrameLabels():**Actualizează etichetele din fereastra AddParfumFrame a view-ului în funcție de ResourceBundle, obține ResourceBundle-ul curent din view, actualizează titlul fereastrei AddParfumFrame cu valoarea corespunzătoare din ResourceBundle.
- **public void setupListeners():**Configurează ascultătorii pentru evenimentele generate de diferite componente vizuale (ex. adaugă ascultători pentru butoanele de adăugare, actualizare și ștergere a parfumurilor, adaugă ascultător pentru butonul de vizualizare a detaliilor unui parfum selectat din tabel, etc)
- **public void addSubmitButtonListener():** Adaugă un ascultător pentru butonul de submit din fereastra de adăugare a unui parfum nou (AddParfumFrame) și apelează metoda addNewParfum() pentru a adăuga noul parfum.
- **public String getSelectedProducator(): R**eturnează producătorul selectat din combobox-ul ComboBoxProducator.
- **public void updateComboBoxSortarePret():** Actualizează combobox-ul ComboBoxSortarePret cu opțiunile de sortare a prețurilor în funcție de limba curentă (ResourceBundle).
- **public void updateComboBoxSortareNume():**Actualizează combobox-ul ComboBoxSortareNume cu opțiunile de sortare după nume în funcție de limba curentă (ResourceBundle).
- **public String[] getFormattedPerfumeDetails(Object[] rowData):r**eturnează detaliile formatate ale unui parfum într-un șir de două elemente.
- **public boolean isDisponibilSelected():**Verifică dacă opțiunea "Disponibil" este selectată în caseta de selectare ChckbxDisponibil.

- **public void handleFilterParfums():** Gestionează filtrarea parfumurilor în funcție de criteriile selectate.Obține valorile prețului minim și maxim, producătorul selectat și opțiunea "Disponibil". Creează un obiect requestData cu informațiile necesare pentru filtrare.Trimite o cerere de filtrare parfumuri către server și primește un răspuns.Dacă răspunsul este de succes, actualizează lista de parfumuri în funcție de filtru.În caz contrar, afișează un mesaj de eroare.
- **public String getLocalizedMessage(String key):**Obține mesajul localizat pentru o cheie dată din ResourceBundle-ul curent al view-ului.Returnează mesajul localizat.
- **public void displayMagazinNames():** Afișează numele magazinelor disponibile în combobox-ul ComboBoxMagazin din view.Trimite o cerere pentru obținerea numelor magazinelor către server și primește un răspuns.Dacă răspunsul este de succes, actualizează combobox-ul cu numele magazinelor.Apoi, afișează lista de parfumuri.
- **public boolean addNewParfum():** Adaugă un parfum nou pe baza informațiilor introduse în câmpurile corespunzătoare din view. Trimite o cerere de adăugare a parfumului nou către server și primește un răspuns.Dacă răspunsul este de succes, afișează un mesaj de succes și actualizează lista de parfumuri.În caz contrar, afișează un mesaj de eroare.
- **public boolean addParfum():**Adaugă un parfum existent în baza de date pe baza informațiilor introduse în câmpurile corespunzătoare din view.Trimite o cerere de adăugare a parfumului către server și primește un răspuns.Dacă răspunsul este de succes, afișează un mesaj de succes și actualizează lista de parfumuri.În caz contrar, afișează un mesaj de eroare.
- **public boolean updateParfum():**Actualizează stocul unui parfum existent în baza de date pe baza informațiilor introduse în câmpurile corespunzătoare din view.Obține ID-ul parfumului și stocul nou din metodele getIdParfum() și getStoc().Verifică dacă ID-ul și stocul sunt valide.Trimite o cerere de actualizare a parfumului către server și primește un răspuns.Dacă răspunsul este de succes, afișează un mesaj de succes și actualizează lista de parfumuri în view. În caz contrar, afișează un mesaj de eroare.
- **public boolean deleteParfum():**Șterge un parfum existent din baza de date.Obține ID-ul parfumului selectat din metoda getSelectedParfumId().Verifică dacă a fost selectat un parfum. Trimite o cerere de actualizare a parfumului către server și primește un răspuns.Dacă răspunsul este de succes, afișează un mesaj de succes și actualizează lista de parfumuri în view. În caz contrar, afișează un mesaj de eroare.
- **public void displayParfumList():**Afișează lista de parfumuri în view.Obține limba curentă și ID-ul magazinului selectat din view. Trimite o cerere de actualizare a parfumului către server și primește un răspuns.Dacă răspunsul este de succes, afișează un mesaj de succes și actualizează lista de parfumuri în view. În caz contrar, afișează un mesaj de eroare.
- **public boolean isSortByPriceAscending():**Verifică dacă sortarea parfumurilor se face în ordine crescătoare după preț.Obține selecția curentă din combobox-ul comboBoxSortarePret din view. Returnează true dacă selecția este "Crescător", altfel returnează false.
- **public boolean isSortByNameAscending():**Verifică dacă sortarea parfumurilor se face în ordine crescătoare după nume. Obține selecția curentă din combobox-ul comboBoxSortareNume din view.Returnează true dacă selecția este "Crescător", altfel returnează false.
- **public void handleSortParfumsByName()** : Gestionează sortarea parfumurilor după nume.Obține informațiile necesare pentru sortare: ID-ul magazinului selectat, direcția de sortare (ascendentă sau descendentă) și limba curentă.Trimite o cerere de sortare a parfumurilor după nume către server, inclusiv informațiile obținute anterior, și primește un răspuns.Dacă răspunsul este de succes, extrage parfumurile sortate din răspuns și le actualizează în view.În caz contrar, afișează un mesaj de eroare.
- **public void handleSortParfumsByPrice()** : Gestionează sortarea parfumurilor după preț.Obține informațiile necesare pentru sortare: ID-ul magazinului selectat, direcția de sortare (ascendentă sau descendentă) și limba curentă.Trimite o cerere de sortare a parfumurilor după nume către server, inclusiv informațiile obținute anterior, și primește un răspuns.Dacă răspunsul este de

succes, extrage parfumurile sortate din răspuns și le actualizează în view.În caz contrar, afișează un mesaj de eroare.

- **public void handleSearchParfumByName():**Gestionează căutarea unui parfum după nume.Obține numele parfumului introdus în câmpul corespunzător din view. Trimite o cerere de sortare a parfumurilor după nume către server, inclusiv informațiile obținute anterior, și primește un răspuns.Dacă răspunsul este de succes, extrage parfumurile sortate din răspuns și le actualizează în view.În caz contrar, afișează un mesaj de eroare.
- **public void displayProducators():**Afișează lista de producători în combobox-ul comboBoxProducator din view.Obține ID-ul magazinului selectat din view.Trimite o cerere de obținere a tuturor producătorilor către server pentru magazinul specificat și primește un răspuns.Dacă răspunsul este de succes, extrage lista de producători din răspuns și actualizează combobox-ul în view.Afișează lista de parfumuri în view.
- **public int getMagazinIdByName(String selectedMagazinName):** Obține ID-ul magazinului pe baza numelui magazinului dat ca parametru.Trimite o cerere către server pentru a obține ID-ul magazinului pe baza numelui și primește un răspuns.Dacă răspunsul este de succes, returnează id-ul magazinului.

Cererile generate de Clientul din AngajatController sunt gestionate de metoda run() din clasa **ClientHandler** a pachetului Server din aplicația Server care apelează metode din clasa **AngajatControllerServer.**

Pachetul Controller din Aplicația Server conține clasa AngajatControllerServer cu următoarele metode:

- **public boolean addParfumNew(int idParfum, String nume, String producator, double pret, String descriere, int stoc, int idMagazin):** Adaugă un nou parfum în baza de date cu informațiile primite ca parametri. Creează un obiect Parfum cu detaliile parfumului. Utilizează ParfumPersistent pentru a crea parfumul în baza de date. Returnează rezultatul adăugării parfumului.
- **public boolean addParfum(int idParfum, int stoc, int idMagazin):**Adaugă un parfum existent în stocul unui magazin în baza de date.Utilizează ParfumMagazinPersistent pentru a crea o înregistrare asociată parfumului în stocul magazinului specificat.Returnează rezultatul adăugării parfumului în stoc.
- **public boolean updateParfum(int idParfum, int stoc, int idMagazin):** Actualizează stocul unui parfum într-un anumit magazin din baza de date.Utilizează ParfumMagazinPersistent pentru a actualiza cantitatea disponibilă a parfumului în stocul magazinului specificat.Returnează rezultatul actualizării stocului parfumului.
- **public boolean deleteParfum(int selectedParfumId, int idMagazin):**Șterge un parfum din stocul unui magazin din baza de date.Utilizează ParfumMagazinPersistent pentru a șterge înregistrarea parfumului din stocul magazinului specificat.Returnează rezultatul ștergerii parfumului din stoc.

Pachetul Model din Aplicația Server
În clasa ParfumPersistent:

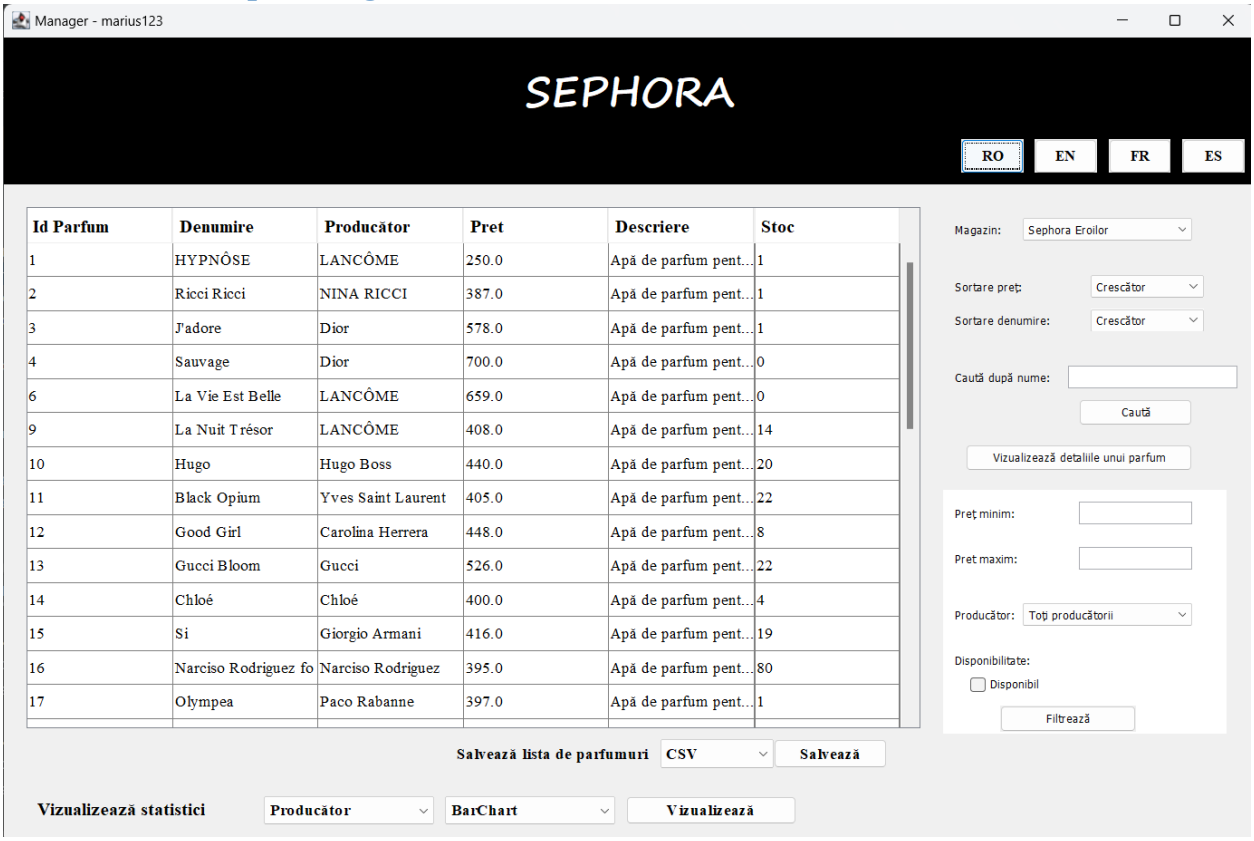
- **create(Parfum parfum):** Această metodă este folosită pentru a crea un nou obiect Parfum în baza de date. primește ca argument un obiect Parfum și întoarce un boolean care indică dacă operațiunea a avut succes sau nu.
- **getAllParfumuriForMagazin(int idMagazin):** Extrage toate parfumurile disponibile 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.
- **readByName(String parfumName):** caută un parfum după nume și returnează obiectul Parfum corespunzător sau null dacă parfumul nu a fost găsit.

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.
- **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.

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:

- **public ManagerView(String user):** Constructorul clasei, inițializează interfața grafică și setează numele utilizatorului.
- **public void initUI():** Inițializează interfața grafică și setează aspectul vizual al componentelor.
- **public void showMessageDialog(String message):** Afișează un mesaj primit ca argument într-o fereastră de dialog.
- **public void showPerfumeDetailsByName(String perfumeDetails):** Afișează detalii despre un parfum pe baza numelui său.
- **public void showPerfumeDetails(String title, String perfumeDetails):** Afișează detalii despre un parfum, primind ca argumente titlul și detaliile acestuia.
- **public void 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

- **public ManagerController(Client client):** Inițializează un obiect ManagerController cu un client dat ca parametru și creează un controller de limbă (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.
- **public void displayMagazinNames():** Obține numele tuturor magazinelor din baza de date prin intermediul unei cereri către server. Dacă răspunsul este de succes, actualizează combobox-ul cu numele magazinelor și afișează lista de parfumuri asociată magazinului selectat.
- **private void handleVisualizeStatistics():** Manipulează evenimentul de vizualizare a statisticilor. Preia tipul de statistică și tipul de diagramă selectate din view. Pe baza acestora, generează diagrama corespunzătoare (diagramă pentru producători sau stocuri) utilizând tipul de diagramă specificat. Apoi, afișează fereastra de statistici.
- **private void showStatisticsWindow():** Afișează fereastra de statistici. Preia resursele lingvistice pentru titlul ferestrei și creează un nou JFrame cu conținutul panelului de statistici din view. Setează comportamentul de închidere și afișează fereastra.
- **private void generateProducerChart(String chartType):** Generează diagrama pentru producători. Preia tipul de diagramă specificat și resursele lingvistice. Calculează numărul de parfumuri pentru fiecare producător pe baza datelor din tabelul de parfumuri din view. Apoi, în funcție de tipul de diagramă, creează diagrama de bare sau diagrama tip tort pentru producători în panelul de statistici din view.
- **private void generateStockChart(String chartType):** Generează diagrama pentru stocuri. Preia tipul de diagramă specificat și resursele lingvistice. Calculează numărul de parfumuri disponibile și indisponibile pe baza datelor din tabelul de parfumuri din view. Apoi, în funcție de tipul de diagramă, creează diagrama de bare sau diagrama tip tort pentru stocuri în panelul de statistici din view.
- **private String getCurrentProducerChartType():** Returnează tipul curent de diagramă pentru producători.
- **private String getCurrentStockChartType():** Returnează tipul curent de diagramă pentru stocuri.
- **private void createProducerPieChart(Map<String, Integer> producerCounts):** Creează diagrama tip tort pentru producători. Preia resursele lingvistice și setează dataset-ul pentru diagramă. Creează diagrama tip tort utilizând dataset-ul și setează opțiunile de afișare a etichetelor și fundalului. Adaugă diagrama în panelul de statistici din view.
- **private void createStockPieChart(Map<String, Integer> stockCounts):** Creează diagrama tip tort pentru stocuri. Preia resursele lingvistice și setează dataset-ul pentru diagramă. Creează diagrama tip tort utilizând dataset-ul și setează opțiunile de afișare a etichetelor și fundalului. Adaugă diagrama în panelul de statistici din view.
- **private void createProducerBarChart(Map<String, Integer> producerCounts):** Creează diagrama de bare pentru producători. Preia resursele lingvistice și setează dataset-ul pentru

diagramă. Calculează suma totală a contoarelor producătorilor și setează opțiunile de afișare a etichetelor și fundalului. Adaugă diagrama în panelul de statistici din view.

- **private void createStockBarChart(Map<String, Integer> stockCounts):** Creează diagrama de bare pentru stocuri. Preia resursele lingvistice și setează dataset-ul pentru diagramă. Calculează suma totală a contoarelor stocurilor și setează opțiunile de afișare a etichetelor și fundalului. Adaugă diagrama în panelul de statistici din view.
- **private void saveTableAsCSV(JTable table):** Salvează tabelul în format CSV. Permite utilizatorului să selecteze locația și numele fișierului de salvare prin intermediul unui dialog de fișiere. Apoi, iterează prin tabel și scrie valorile celulelor în fișierul CSV. În caz de succes, afișează un mesaj de confirmare.
- **private void saveTableAsJSON(JTable table):** Salvează tabelul în format JSON. Permite utilizatorului să selecteze locația și numele fișierului de salvare prin intermediul unui dialog de fișiere. Transformă tabelul într-un obiect JSONArray, unde fiecare rând reprezintă un obiect JSONObject care conține valorile celulelor. Scrie obiectul JSONArray în fișierul JSON. În caz de succes, afișează un mesaj de confirmare.
- **private void saveTableAsXML(JTable table):** Salvează tabelul în format XML. Permite utilizatorului să selecteze locația și numele fișierului de salvare prin intermediul unui dialog de fișiere. Iterează prin tabel și scrie valorile celulelor în format XML, respectând structura specifică a fișierelor XML pentru acest tabel. În caz de succes, afișează un mesaj de confirmare.
- **private void saveTableAsTXT(JTable table):** Salvează tabelul în format text (TXT). Permite utilizatorului să selecteze locația și numele fișierului de salvare prin intermediul unui dialog de fișiere. Scrie valorile celulelor tabelului în fișierul TXT, separându-le prin tabulări și trecând la linia următoare pentru fiecare rând. În caz de succes, afișează un mesaj de confirmare.
- **public void updateComboBoxStatistics():** Actualizează combobox-ul de statistică.
- Aceste metode au fost descrise anterior la secțiunea Angajat: isSortByPriceAscending(), isSortByNameAscending(), updateComboBoxSortarePret(), updateComboBoxSortareNume(), displayProducators(), getSelectedProducator(),getPriceMin(), getPriceMax():getEnteredParfumName() isDisponibilSelected(), handleFilterParfums(), handleSortParfumsByName(), handleSortParfumsByPrice(), handleSearchParfumByName(), getMagazinIdByName() displayParfumList() updateParfumList():

Cererile generate de Clientul din ManagerController sunt gestionate de metoda run() din clasa **ClientHandler** a pachetului Server din aplicația Server care apelează metode din clasa **ManagerControllerServer.**

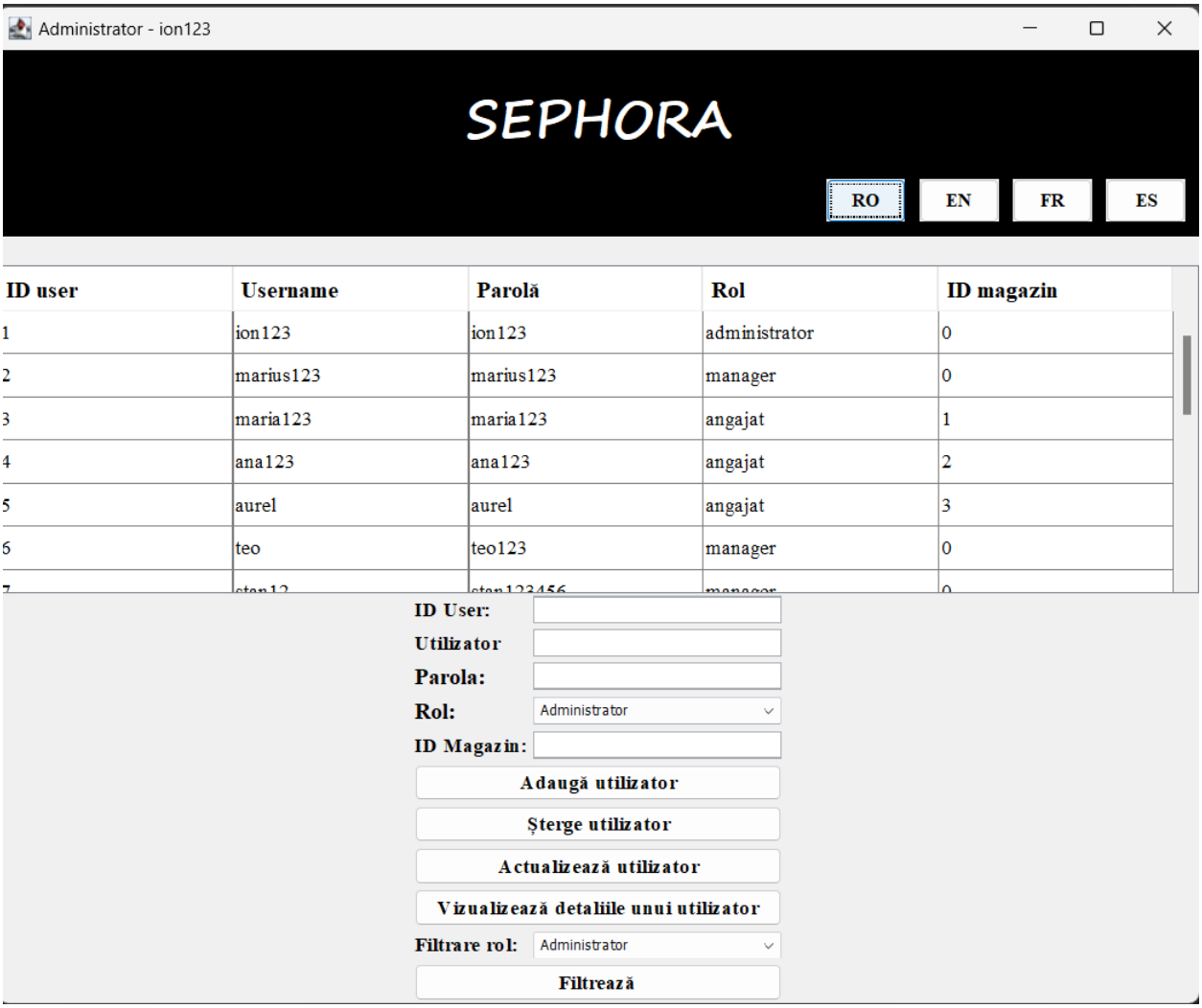
Pachetul Controller din Aplicația Server conține clasa ManagerControllerServer cu următoarele metode:

- **public List<String> getAllProducators(int idMagazin):** Returnează o listă cu toți producătorii disponibili pentru un anumit magazin identificat prin id-ul său.
- **public List<Object[]> filterParfums(int idMagazin, Double priceMin, Double priceMax, String producator, Boolean disponibil, String language):** Filtrază parfumurile în funcție de criteriile specificate, cum ar fi prețul minim și maxim, producătorul, disponibilitatea și limba. Returnează o listă de obiecte (rânduri) care conțin detalii despre parfumurile filtrate.
- **public List<Object[]> getAllParfumuriForMagazinAsObjects(int idMagazin, String language):** Returnează toate parfumurile disponibile într-un anumit magazin identificat prin id-ul său, sub formă de obiecte (rânduri) care conțin detalii despre parfumuri.
- **public List<Object[]> getSortedParfumsByPriceAsObjects(int idMagazin, boolean ascending, String language):** Returnează parfumurile sortate în funcție de preț, în ordine crescătoare sau descrescătoare, într-un anumit magazin identificat prin id-ul său. Rezultatul este returnat sub formă de obiecte (rânduri) care conțin detalii despre parfumuri.

- **public List<Object[]> getSortedParfumsByNameAsObjects(int idMagazin, boolean ascending, String language):** Returnează parfumurile sortate în funcție de nume, în ordine alfabetică crescătoare sau descrescătoare, într-un anumit magazin identificat prin id-ul său. Rezultatul este returnat sub formă de obiecte (rânduri) care conțin detalii despre parfumuri.
- **private List<Object[]> convertParfumMagazinListToObjectList(List<ParfumMagazin> parfumMagazinList):** Converteste o listă de obiecte ParfumMagazin într-o listă de obiecte (rânduri) care conțin detalii despre parfumuri.
- **public String handleSearchParfumByName(String parfumName):** Caută un parfum după nume și returnează detalii despre acesta sub forma unui șir de caractere formatat.
- **public int getMagazinIdByName(String selectedMagazinName):** Returnează id-ul unui magazin după numele său.
- **public List<String> getAllMagazinNames():** Returnează o listă cu toate numele de magazine disponibile.

Pachetul Model conține **metodele de filtrare a parfumurilor și de vizualizare a listei de parfumuri descrise anterior (secțiunea Angajat)**

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.

- **public Angajat Controller(Client client):** Inițializează un obiect ManagerController cu un client dat ca parametru și creează un controller de limbă (LanguageController).
- **public void setView(AdministratorView view):** Metoda care setează instanța curentă a view și apelează setupListeners().
- **public void showView(String user):** Metoda pentru afișarea ferestrei AdministratorView și înregistrarea listenerilor pentru schimbarea limbii și afișarea listei de utilizatori.
- **public void changeLanguage(Locale locale):** Metoda care schimbă limba și actualizează componentele UI.
- **public void updateUIComponents():** Actualizează etichetele și traducerile componentelor UI în funcție de limba selectată.
- **public void updateRoleComboBox(String[] translatedRoles):** Actualizează combobox-ul cu rolurile traduse.
- Metodele **getUserId(), getUserName(), getPassword(), getRol() și getIdMagazin()** sunt utilizate 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():** Configurează listenerii pentru butoanele de adăugare, ștergere, actualizare și vizualizare a detaliilor utilizatorilor.
- **public void addUser():** Adaugă un utilizator în sistem. Obține informațiile necesare (id utilizator, nume utilizator, rol, parolă, id magazin) și trimite o cerere de adăugare a utilizatorului către server. Dacă adăugarea este realizată cu succes, afișează un mesaj de succes și actualizează lista utilizatorilor afișată în interfața grafică. În caz contrar, afișează un mesaj de eroare.
- **public void updateUser():** Actualizează informațiile unui utilizator existent în sistem. Obține informațiile necesare (id utilizator selectat, id utilizator nou, nume utilizator nou, rol nou, parolă nouă, id magazin nou) și trimite o cerere de actualizare a utilizatorului către server. Dacă actualizarea este realizată cu succes, afișează un mesaj de succes și actualizează lista utilizatorilor afișată în interfața grafică. În caz contrar, afișează un mesaj de eroare.
- **public void deleteUser():** Șterge un utilizator existent din sistem. Obține id-ul utilizatorului selectat și trimite o cerere de ștergere a utilizatorului către server. Dacă ștergerea este realizată cu succes, afișează un mesaj de succes și actualizează lista utilizatorilor afișată în interfața grafică. În caz contrar, afișează un mesaj de eroare.
- **public void displayUserList():** Afiseaza lista utilizatorilor din sistem. Trimite o cerere de afișare a listei de utilizatori către server. Dacă afișarea este realizată cu succes, primește datele utilizatorilor sub formă de matrice de obiecte și le afișează în interfața grafică. În caz contrar, afișează un mesaj de eroare.

Cererile generate de Clientul din AdministratorController sunt gestionate de metoda run() din clasa **ClientHandler** a pachetului Server din aplicația Server care apelează metode din clasa **AdministratorControllerServer.**

Pachetul Controller din Aplicația Server conține clasa AdministratorControllerServer cu următoarele metode:

- **public boolean addUser(int userId, String username, String password, String rol, int idMagazin)**: Adaugă un utilizator nou în sistem. Creează un obiect **Utilizator** cu informațiile utilizatorului și apelează metoda **create()** a obiectului **utilizatorPersistent** pentru a adăuga utilizatorul în baza de date. Returnează **true** în caz de succes sau **false** în caz contrar.
- **public boolean updateUser(int userId, String username, String password, String rol, int idMagazin)**: Actualizează informațiile unui utilizator existent în sistem. Creează un obiect **Utilizator** cu informațiile actualizate și apelează metoda **update()** a obiectului **utilizatorPersistent** pentru a actualiza utilizatorul în baza de date. Returnează **true** în caz de succes sau **false** în caz contrar.
- **public boolean deleteUser(int userId)**: Șterge un utilizator existent din sistem. Apelează metoda **delete()** a obiectului **utilizatorPersistent** pentru a șterge utilizatorul din baza de date. Returnează **true** în caz de succes sau **false** în caz contrar.
- **public Object[][] getUserList()**: Obține lista utilizatorilor din sistem. Apelează metoda **getAll()** a obiectului **utilizatorPersistent** pentru a obține o listă de utilizatori din baza de date persistentă. Construiește o matrice bidimensională de obiecte pentru a stoca informațiile utilizatorilor și o returnează. Fiecare rând din matrice reprezintă un utilizator și conține informații precum id utilizator, nume utilizator, parolă, rol și id magazin.

În pachetul model clasa UtilizatorPersistent are următoarele metode

- **public boolean create(Utilizator utilizator):** adaugă un nou utilizator în sistem și returnează true dacă operațiunea este reușită sau false dacă nu.
- **public boolean update(Utilizator utilizator):** actualizează datele unui utilizator existent în sistem și returnează true dacă operațiunea este reușită sau false dacă nu.
- **public void delete(int userId):** șterge un utilizator cu ID-ul specificat din sistem și returnează true dacă operațiunea este reușită sau false dacă nu.
- **public void getAll():** returnează o listă cu toți utilizatorii din sistem.
- **public Utilizator 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.

Concluzie

Arhitectura este un tip de design pentru sisteme de calcul distribuite care împarte responsabilitățile între două tipuri de componente: serverele și clienții. În acest model, serverele sunt entități care furnizează resurse sau servicii, în timp ce clienții sunt entități care solicită și folosesc aceste resurse sau servicii. Acest design poate fi implementat în cadrul unei rețele de calculatoare sau chiar pe un singur dispozitiv, în cazul în care aplicațiile client și server sunt separate și interacționează între ele.

Proiectul dezvoltat ilustrează modelul arhitectural și include suport pentru internaționalizare, permițând traducerea interfeței în mai multe limbi.

Toți actorii, indiferent de rol, necesită autentificare pentru a putea realiza diverse operațiuni în cadrul aplicației. Dacă datele de autentificare nu sunt valide, autentificarea va eșua. Clasa Client, care face parte din pachetul Client, reprezintă partea client în arhitectura . Aceasta stabilește o conexiune cu un server, trimite cereri către acesta și primește răspunsuri. Pachetul Common include clasele Request și Response. Clasa Request reprezintă o cerere trimisă de client către server, având un anumit tip și date suplimentare necesare procesării acesteia. Clasa Response reprezintă răspunsul primit de la server, având un anumit tip, un indicator de succes și date suplimentare. Clasele din pachetul Controller (aplicația Client) au rolul de a facilita interacțiunea dintre View și Server, gestionând logica de afișare, trimiterea cererilor către Server și primirea răspunsurilor. Fiecare clasă din Controller creează un obiect de tip Client pentru a comunica cu Serverul.

Pachetul Server conține două clase principale: ParfumServer și ClientHandler. ParfumServer reprezintă serverul central care gestionează conexiunile clienților. Folosește un obiect de tipul ServerSocket pentru a accepta conexiunile de la clienți. Când un client se conectează, serverul generează un nou obiect ClientHandler pentru a administra interacțiunea cu respectivul client. ClientHandler are rolul de a gestiona comunicarea cu un anumit client. În funcție de tipul cererii, ClientHandler poate efectua diverse operații, cum ar fi autentificarea unui utilizator, adăugarea, actualizarea sau ștergerea unui utilizator sau a unui parfum, sortarea sau filtrarea parfumurilor etc., apelând metodele corespunzătoare din clasele din pachetul Controller ale aplicației Server. În cazul în care apare o eroare, aceasta este afișată și resursele sunt închise.

În concluzie, cele două aplicații (Client și Server) reprezintă o implementare solidă a modelului arhitectural , care facilitează dezvoltarea ulterioară și adaptabilitatea.

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, M., Mao, S., & Liu, Y. (2014). Big data: A survey. Mobile networks and applications, 19(2), 171-209.
- Chen, P. P. (1976). The entity-relationship model: Toward a unified view of data. ACM Transactions on Database Systems (TODS), 1(1), 9-36.
- Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: yesterday, today, and tomorrow. In Present and Ulterior Software Engineering (pp. 195-216). Springer, Cham.
- 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
- Furht, B., & Escalante, A. (2010). Handbook of Cloud Computing. Springer.
- 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
- 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.
- Li, B., Yu, J., Wang, Z., Xu, Y., & Deng, Q. (2019). Edge computing in IoT based on the 5G network edge cloud system architecture. Wireless Communications and Mobile Computing, 2019.
- 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
- Papazoglou, M. P., & Georgakopoulos, D. (2003). Service-oriented computing. Communications of the ACM, 46(10), 25-28.
- 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).
- Pautasso, C., Zimmermann, O., & Leymann, F. (2008). Restful web services vs. "big" web services: making the right architectural decision. Proceedings of the 17th international conference on World Wide Web, 805-814.
- RedMonk. (2021). The RedMonk Programming Language Rankings: January 2021.
- Stallings, W., & Tahiliani, M. P. (2018). Cryptography and Network Security: Principles and Practice. Pearson.
- 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
- Yu, J., Subramanian, N., Ning, H., & Xin, X. (2016). A survey on the edge computing for the Internet of Things. IEEE Access, 6, 6900-6919.
- 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.