**DOCUMENTATIE**

# **PROIECTARE SOFTWARE**

# **TEMA 2**

## **Lant de magazine de parfumuri**

### Tivadar Maria Simona

### Grupa 30233

### Cuprins

▪ Enunțul problemei

▪ Instrumente utilizate

▪ Justificarea limbajului de programare ales

▪ Descrierea diagramelor UML

▪ Descrierea aplicației

1. Enuntul problemei

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.

❖ Vizualizarea listei tuturor parfumurilor dintr-un magazin selectat sortată după următoarele criterii: denumire și preț;

❖ Cautarea unui parfum dupa denumire

❖Salvare liste cu situatia parfumurilor din parfumeria la care este angajat in mai multe formate: csv, json, xml, txt

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, parfumerie;

❖Salvare liste cu situatia parfumurilor din parfumerii in mai multe formate: csv, json, xml, txt

❖ 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

❖Filtarea listei utilizatorilor dupa tipul utilizatorilor

1. Instrumente Utilizate

La nivelul instrumentelor utilizate am ales sa ma folosesc de IDE-ul (Integrated Development Eviorment) IntelliJ IDEA pentru scrierea si editarea codului scris in limbajul JAVA. Acest IDE a fost ales datorita usurintei sale de folosire dar si faptului ca marea majoritate a proiectelor scrise pana acum de mine au fost realizate in IntelliJ IDEA. Am ales ca proiectul sa se incadreze in tipologia de tip *Maven*. Versiunea IDE-ului este **2021.2.**

La nivelul bazei de date am folosit MySQL Workbench deoarece acesta a fost recomandat in cadrul orelor de laborator la disciplina Proiectare Software si a experientei dobandite in lucrul cu aceasta baza de date de a lungul anilor trecuti de facultate. Ca baza de date am ales SQL – aceasta este o basa de date relationala care se bazeaza pe idea ca fiecare tupla apartine unui anumit table iar tuplele sunt unice prin cheia lor primara. Legaturile dintre tabele de realizeaza prin legaturi PK (Primary Key) 🡪 FK (Foreign Key) si pot fi 1 to 1 sau 1 to many. Eventualele legaturi many to many au fost prelucrate pentru a respecta standardele impuse de buna practica in lucrul cu baze de date relationale – acest lucru va fi detaliat ulterior intr-un capitol urmator.

Pentru a realiza conexiunea intre proiectul in Java si baza mea de date am folosit un **Connector** de tip Jar, ale carui dependentele au fost adaugate proiectului prin intermediul dependentelor de tip Maven.

Connectorul se numeste *mysql-connector-java-8.0.30.jar*

La nivelul interfetei grafice am ales ca folosesc Java FX. Am ales aceasta pentru ca a fost recomandata deopotriva la laborator si pentur ca creaza mediul propice dezvoltarii unui proiect care respecta arhitectura MVVM (*Model- View- View-Model*).

In ceea ce priveste testarea proiectului am folosit deopotriva un tool deja existent in Java si anume *JUnit.* Acesta a facilitate scrierea de teste manual care s-au ocupat de verificarea conexiunii cu baza de date, crearea tabelele, operatii crud si alte teste care tin de persistenta intregului proiect.

Pentru efectuarea cu success a salvarii datelor de tip *parfum* in diferite formate de fisiere (**xml,txt,json,csv**) am folosit cateva dependente necesare, pe care le-am preluat de pe maven si apoi le-am inclus in fisierul de *pom.xml* pentru a corespunde cu cerintele de care avea nevoie proiectul meu. Printre noile dependente adaugate se numara

<dependency>  
 <groupId>org.openjfx</groupId>  
 <artifactId>javafx-fxml</artifactId>  
 <version>11.0.2</version>  
</dependency>  
<dependency>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-databind</artifactId>  
 <version>2.13.3</version>  
</dependency>

Pentru realizarea operatiei de scriere in fisiere JSON

<dependency>  
 <groupId>org.glassfish.jaxb</groupId>  
 <artifactId>jaxb-runtime</artifactId>  
 <version>2.3.1</version>  
</dependency>

Pentru realizarea operatiei de scriere in fisiere XML

1. Justificarea limbajului de programare ales
   1. Java

Dupa cum am mentionat si anterior, limbajul in care am ales sa scriu proiectul meu software este Java. Motivele sunt multiple, dar voi dezvolta 3 pe care le consider cele mai relevante situatiei si cerintei proiectului meu.

a. **Portabilitatea și interoperabilitatea:** Java este un limbaj de programare multiplatforma, ceea ce inseamna ca aplicatia poate fi executata pe orice sistem de operare care are instalat un runtime Java. De asemenea, exista multe librarii si framework-uri disponibile in Java, care faciliteaza interoperabilitatea cu alte aplicatii si servicii.

b. **Securitatea:** Java este cunoscut pentru a fi un limbaj de programare securizat si robust. Din acest motiv, este adesea utilizat pentru dezvoltarea aplicatiilor critice pentru securitate, precum aplicatiile bancare sau cele de comert electronic. Aplicatia pentru lanțul de magazine de parfumuri contine informatii sensibile despre clienti si produse, deci este important sa fie protejata impotriva vulnerabilitatilor si a atacurilor cibernetice.

c. **Scalabilitatea**: Java este scalabil si poate gestiona volume mari de date si trafic. Aplicatia pentru lanțul de magazine de parfumuri trebuie sa poata gestiona cantitati mari de date despre produse, stocuri si utilizatori, iar Java ofera suport pentru aceste sarcini. De asemenea, aplicatia poate fi scalata prin adaugarea de noi functii si caracteristici, in functie de nevoile si cerintele viitoare.

* 1. MySQL

Dintre bazele de date relationale existente pe piata am ales sa folosesc MySQL – deopotriva, voi mentiona cateva caracteristici care m-au facut sa aleg aceasta baza de date.

**Fiabilitate**: MySQL este una dintre cele mai fiabile baze de date relationale disponibile pe piata. Aceasta este conceputa pentru a oferi performante constante si a minimiza pierderile de date, prin tehnici de backup si replicare. MySQL are, de asemenea, o comunitate puternica de dezvoltatori, care se asigura ca aceasta este actualizata si dezvoltata in mod constant.

**Scalabilitate:** MySQL este proiectat pentru a gestiona volum mare de date si trafic, facandu-l potrivit pentru aplicatiile web cu trafic ridicat. Aceasta poate fi extinsa prin clustere si replicare, ceea ce permite sa adaugati mai multe servere la baza de date, pentru a mentine performanta in timpul cresterii traficului.

**Flexibilitate:** MySQL poate fi utilizat pentru o varietate de aplicatii, de la aplicatii mici, cu un volum redus de date, pana la aplicatii complexe cu volume mari de date. MySQL este compatibil cu o varietate larga de sisteme de operare, iar acest lucru il face o alegere excelenta pentru dezvoltarea aplicatiilor multi-platforma. In plus, MySQL are o interfață SQL standard, ceea ce face mai ușoară migrarea aplicației la altă bază de date, în cazul în care este necesar în viitor.

1. Descrierea diagramelor UML
   1. O imagine care conține diagramă

      Descriere generată automatUseCase Diagram

Diagrama de mai sus surprinde toate usecase-urile posibile aplicatiei mele. Dupa cum se mentioneaza si in cerinta, orice actiune poate fi facuta doar dup ace persoana respective de autentifica in sistem. Astfel, persoana – initial lipsita de vreun rol anume – este supusa operatiei de logare. Daca acesta se efectueaza cu success, UseCase ul meu se extinde pe 3 ramuri posibile, datorate celor 3 roluri posibile ale utilizatorului: Admin, Manager, Employee.

Adminul are putere doar asupra operatiilor care tin de restul utilizatorilor din sistem. El poate sa-I vizualizeze pe acestia si in plus, poate efectua operatiile de tip CRUD pe orice tupla de tip Person din baza mea de date.

Managerul si Employee (angajatul) au amandoi putere asupra datelor referitoare la parfumuri. Cel dintai poate sa le vizualizeze pe toate si sa caute dupa nume sau sa filtreze produsele dupa Brand, Disponibilitate, Stoc.

Cel din urma are putere si asupra operatiilor de CRUD asupra tuplelor din tabelele legate de Parfum, ParumeShop….

In plus, s-au adaugat noi cerinte fata de proiectul anterior. Asadar, Adminul este capabil sa vizualizeze toti utilizatorii dupa rol. Mai mult, Managerul si Angajatul pot genera fisiere de diferite formate pentru a stoca informatii despre parfumuri, totul la un click distanta.

* 1. Class Diagram

O imagine care conține text, chitanță, captură de ecran

Descriere generată automat

Diagrama de clase este structurata pe pachete menite sa respecte arhitectura de tip MVVM – Model View View Model.

In pachetul Model sunt incluse clasele Perfume si PerfumeShop, care reprezinta modelele de date ale aplicatiei. Clasa Perfume contine informatii despre un anumit parfum, cum ar fi denumirea, producatorul si pretul. Clasa PerfumeShop este responsabila pentru stocarea si administrarea inventarului de parfumuri, precum si pentru efectuarea operatiunilor CRUD.

In pachetul View sunt incluse interfetele si clasele pentru interfata utilizator, care permit utilizatorilor sa interactioneze cu aplicatia. In pachetul ViewModel sunt incluse clasele ViewModel, care controleaza View-ul si comunica cu Model-ul pentru a gestiona logica de afaceri si de prezentare a datelor. ViewModel nu are voie sa cunoasca nimic despre View, despre interfete grafice sau orice tine de instante grafice, in timp de View nu are voie sa cunoasca nimic ce tine de Model, de entitatile mele si obiectele instantiate. El stie doar sa afiseze sau sa updateze cee ace ViewModel i trimite.

* 1. Entity Relationship Diagram

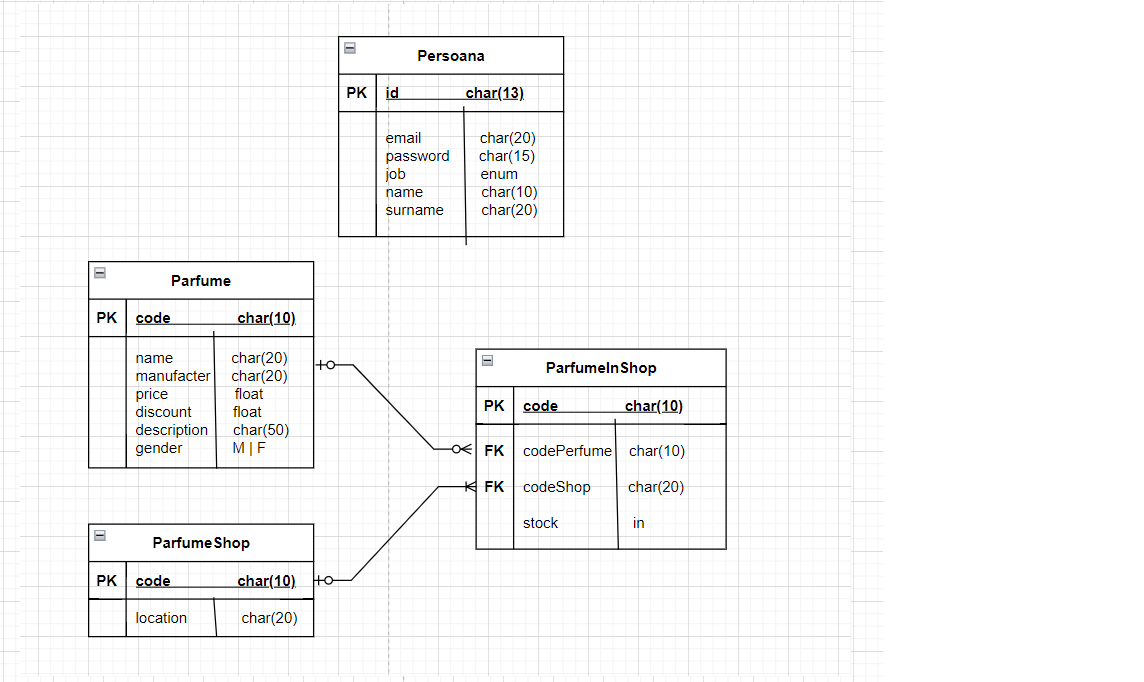


Diagrama entitate-relatie este o reprezentare grafica a relatiilor dintre tabelele dintr-o baza de date. In aceasta diagrama exista patru tabele: Parfum, ParfumShop, ParfumInShop si Person.

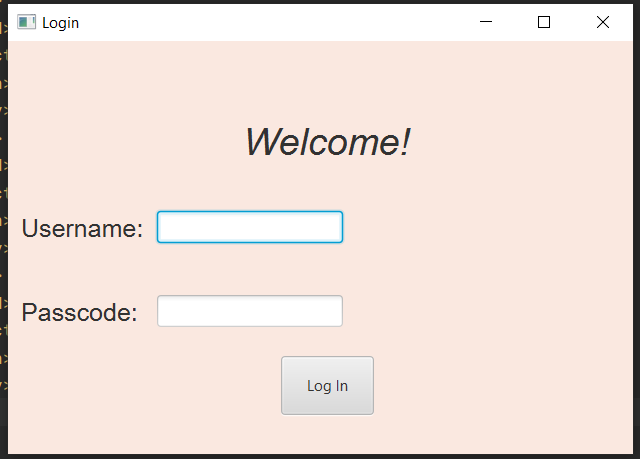
Tabela Parfum contine informatii despre un parfum, cum ar fi numele, producatorul, pretul si compozitia. Tabela ParfumShop contine informatii despre magazinele de parfumuri, cum ar fi numele, adresa si numarul de telefon. Tabela ParfumInShop este o tabela de legatura intre cele doua tabele, care contine informatii despre stocul de parfumuri disponibil in fiecare magazin. Tabela Person contine informatii despre utilizatorii aplicatiei, cum ar fi numele, adresa de email si parola. Aceasta diagrama este utila pentru a intelege structura bazei de date si relatiile dintre tabele, astfel incat sa se poata dezvolta aplicatii care sa acceseze si sa manipuleze datele stocate in aceasta baza de date.

5.Descrierea aplicatiei

Aplicatia are o interfata grafica folosind JavaFX si se bazeaza pe arhitectura Model-View-View-Model (MVVM).

Pentru a realiza interfata utilizator, am folosit JavaFX. Aceasta consta in pregatirea dependentelor potrivite pentru a fi mentionate in pom.xml. Mai mult decat atat, codul pentru a creea o astfel de interfata grafica este scris in *fxml.* Pe langa aceasta, am clasa de java care are ca attribute de tip private elemente din interfata propriuzisa, care sunt adnotate cu **@FXML.**

In ceea ce priveste arhitectura de programare, am optat pentru MVVM, care separa logica aplicatiei in trei parti: Model, View si View-Model. Modelul contine datele aplicatiei, View afiseaza aceste date utilizatorului si ViewModel continue toata logica de calcul, asteapta sa fie notificat de catre view si vicevers, in ceaa ce priveste update-urile la nivel de aplicatie. Pentru a surprinde cat mai bine acest lucru am folosit liste de tipul *ObservableList*.



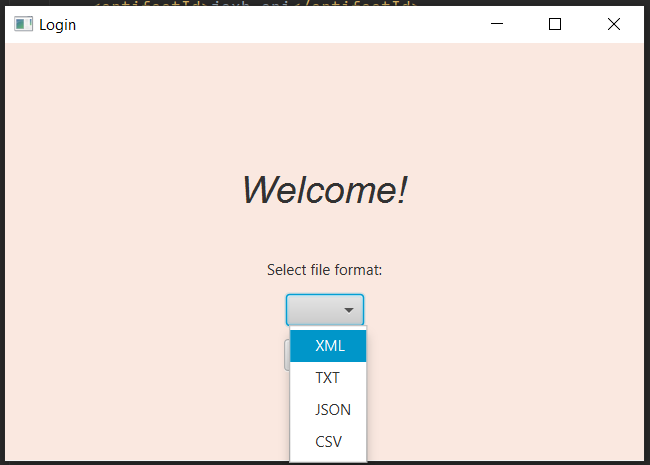
Cand un utilizator se conecteaza, aplicatia verifica daca este admin, manager sau angajat si il directioneaza catre interfata corespunzatoare.

Pentru a realiza autentificarea, am creat o baza de date cu informatii despre utilizatori si nivelurile de acces corespunzatoare fiecarui utilizator. Dupa logare, aplicatia verifica nivelul de acces al utilizatorului si il redirecteaza catre interfata corespunzatoare.

Admin-ul are acces la tabelele de utilizatori si poate realiza operatii CRUD pentru a gestiona conturile de utilizator.

Interfata de administrare permite admin-ului sa vizualizeze, sa creeze, sa modifice si sa stearga utilizatori din baza de date. Aceasta operatie se realizeaza prin intermediul Viewmodelului care se ocupa de logica de business, iar View-ul afiseaza datele utilizatorului si permite acestuia sa introduca informatii.

Managerul are acces la tabelele de magazine de parfumuri si poate gestiona produsele disponibile in fiecare magazin.De asemena, poate cere exportarea infromatiilor referitoare la toate parufmurile in 4 formate diferita.



Interfata de management a magazinelor permite managerului sa vizualizeze stocurile de parfumuri din fiecare magazin, sa adauge sau sa stearga produse, sa actualizeze stocurile si preturile, sa filtreze informatiile in functie de criterii precum producator, disponibilitate sau pret.

Angajatul poate efectua operatii de tip CRUD pe parfumuri si le poate vizualiza dupa pret sau nume, le poate filtra in 3 moduri diferite.

O imagine care conține masă

Descriere generată automat

Interfata de angajat permite acestuia sa vizualizeze stocurile de parfumuri din magazinul in care lucreaza si sa adauge, sa stearga sau sa actualizeze informatii despre produsele disponibile.

Pentru a testa functionalitatea bazei de date si operatiile CRUD, am creat un pachet separat pentru testare, folosind JUnit.

JUnit este o biblioteca Java pentru unit testing, care permite testarea automata a aplicatiilor. Prin crearea unui pachet separat pentru testare, am putut verifica daca baza de date functioneaza corespunzator si daca operatiile CRUD sunt realizate cu succes.

O imagine care conține text

Descriere generată automat

O imagine care conține text

Descriere generată automat

Scenariul de utilizare poate fi: un utilizator se autentifica, se redirecteaza catre interfata corespunzatoare, acceseaza tabelele sau filtreaza informatiile, face actualizari si/sau adaugari in baza de date.