A picture containing logo

Description automatically generated**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**CATEDRA CALCULATOARE**

**LANT DE MAGAZINE DE PARFUMURI**

**Documentatie**

**Proiectul Final**

**Viman Andrei-Liviu**

**Grupa: 30236**

# Enuntul problemei

Scopul acestui proiect este de a dezvolta o aplicatie desktop care poate fi utilizata intr-un lant de magazine de parfumuri. Aplicatia are doua tipuri de utilizatori: angajat al lantului de magazine dar si administratorul acestui lant. Fiecare are un set de operatii pe care le poate face in functie de rolul pe care il are in magazine. In al doilea rand scopul acestui proiect este de a creea o aplicatie Client Server in java si de a folosi mai multe sabloane de proiectare. Cele folosite de mine sunt MVC, Observer, Singleton, Decorator si Builder.

# Instrumente utilizate si justificarea limbajului ales

Inainte sa ma apuc de implementarea acestui lant de magazine, a fost nevoie sa inteleg mai bine ce am de facu si cum trebuie sa functioneze aceasta aplicatie a trebuit sa folosesc StarUML pentru a construi diagrama cazurilor de utilizare iar mai apoi diagrama de clase care sa respecte arhitectura MVC. Numai dupa ce am facut aceste lucruri m-am apucat efectiv de implementarea proiectuli ales in java.

Pentru a reusi sa implementej in java am folosit mediul de dezvoltare IDE IntelliJ IDEEA 2021.3.2. Am ales sa folosesc acest ide doarece este cel mai prietenos si cel mai usor de folosit. Am ales sa folosesc limbajul java pentru implementarea proiectului deoarece este un limbaj foarte des intalnit intre programatori si este cel pe care il stiu cel mai bine. Pe de alta parte am ales java deoarece are framework-ul Java Swing cu care sunt destul de familiarizat, avand nevoie de el pentru a implementa interfata grafica. Recunos ca astfel am reusit sa invat foarte multe lucruri noi in acest limbaj.

# Descrierea diagramelor UML

Petru inceput o sa vorbesc despre diagrama cazurilor de utilizare, unde am doi actor, un angajat si un aministrator. La incept ei sunt nevoiti sa se logheze in aplicatie iar in caz de succes pentru administrator acsta trebuie sa poate prelua datele despre angajati din basa de date salvata in mySQL dedicata, trebuie sa poata sa adauge un utilizator nou al aplicatiei, sa poata sa stearga un angajat din aceasta baza de date dar si sa actualizeze un angajat in aceasta baza de date. Astfel fiind nevoit sa fac operatii CRUD pe o baza de date stocata in mySQL, astfel am descoperit ca este cu mult mai usor sa lucrezi cu o baza de date stocata in mySQL decat in XML. In cazul in care are probleme la logare apar mici mesaje de notificare pentru a-i spune ce greseste.

Diagram

Description automatically generated

In caz de succes pentru un angajat acesta poate sa faca mai multe operatii pe o baza de date salvata tot sub forma unui fisier XML, adica operatii de adaugare a unui nou parfum in aceasta baza de date, dar si stergera unui anumit parfum din aceasta baza de date da nu in ultimul rand si o actualizare a anumitor parfumuri. El mai poate sa vizualizeze toate parfumurile dintr-un magazin selectat, poate sa filtreze parfumurile dupa diverse criterii : producator, disponibilitate dar si pret, poate sa caute un anumit parfum dupa denumire dar poate si sa salveze liste de parfumuri in mai multe formate de exemplu csv sau json sau xml. Se pot vizualiza si diverse statistici despre pret, disponibilitate si magazine. Dupa cum se poate vedea in poza de mai jos.

Diagram

Description automatically generated

Mai sus se poate vizualiza diagrama cazurilor de utilizare pentru server care se ocupa doar de baza de date, primeste requesturi si se ocupa de ele.

Mai departe o sa descriu diagrama UML realizata realizata dupa conceptul arhitectural MVC un model care faciliteaza crearea aplicatiilor web de aceea a devenit asa de popular.

Astfel modelul defineste datele care trebuie utilizate in interfata cu utilizatorul. Prima clasa care apare aici este Parfume care contine numele parfumului si un obiect de tipul ParfumeInfo astfel reusind sa surprind toate proprietatile pe care le are un parfum. Clasa ParfumeXmlWriter este o interfata pe care o utilizez la salvarea rapoartelor in format .xml. ParfumeCsvWriter este o interfata pe care o utilizez la salvarea rapoartelor in format .csv. Clasa ParfumeInfo care contine urmatoarele informatii despre un parfum: numele fabricantului, numarul de copii ramase pe stoc, codul de bare, pretul, cantitatea de parfum dintr-o sticluta dar si numarul de copii vandute. ParfumeJsonWriter este interfata cu care salvez rapoartele in formatul .json. ParfumeService este clasa care se ocupa de persistenta unui parfum adica apeleaza metode de salvare creare sau updatare ale unui parfum. Clasa ParfumeStore contine ca si atribute numele unui magazin si o lista cu mai multe parfumuri care se afla in acel magazin. Aici mai exista si doua clase ParfumeInfoBuilder si ParfumeBuilder care ne ajuta sa respectam patternul creational builder, ele creeand obiecte in mod indirect.

Acum urmeaza clasela pentru utilizatori, acestea sunt: User care are numele de utilizator, parola lui si rolul acestuia in lantul de magazine. Clasa UserService realizeaza persistenta unui utilizator si apeleaza metode din clasa abstracta pe care o extinde. Aici mai exista si clasa UserBuilder care ne ajuta sa respectam patternul creational builder, ea creeand obiecte in mod indirect.

Tot in model mai avem clasele care citesc date din doua fisiere xml pentru a putea creea interfata in mai multe limbi de circulatie internationala , aceste clase sunt LanguageUser si LanguageParfume. Pe langa aceste clase mai avem o intervata Observer cu o singura metoda neimplementata si o clasa abstracta AbstractObserver pentru a putea aduga observatori claselor care implementeaza interfata Observer. Pentru ca totul sa imi fie mai usor am mai creat inca doua clase ModelUser si Model Parfume ca sa pot accesa mult mai usor persistenta limba si operatia curenta care se executa din controller.

Nu in ultimul rand tot in model avem o clasa de client care este destl de importanta, ea facand legatura intre client si server conectandu-se la un socket cu o anumita adresa ip, la un anumit port si in functie de butonul apasat pe interfata ea va transmita un anumit request serverului care se va ocupa mai apoi cu interogarea bazei de date.

View-ul este interfata grafica a acestei aplicatii care afiseaza datele existente in model prin intermediul data bindingului legat intre el si viewmodel. Acest pachet contine 6 clase dintre care 3 clase ajuta la afisarea statisticilor. Celelalte 3 clase implementeaza libraria JfreeChart pentru statisticie de care aveam nevoie.In clasa UserView se afiseaza interfața pt pagina de login prin numeroasele atribute prezente, și se gasesc doar gettere si settere pentru ca restul lucrilor se implementeaza in controller. AdministratorView este clasa unde se afiseaza interfata pentru administrator unde se gasesc elementele de care are el nevoie si in acelasi timp se si gettere si settere . Nu in ultimul rand clasa EmployeeView reprezinta interfața pt angajat prin numeroasele atribute prezente, și numeroase gettere si setter fara de care controllerul nu ar putea exista. In acest pachet se realizeaza efectiv legarea datelor intre view si controller.

Controller-ul preia datele din model si le proceseza pentru a fi disponibile interfetei grafice. Acesta contine trei clase prima pe care o prezint este UserController care realizeaza conexiunea dintre interfata grafica pentru angajat dar si interfata pentru administrator. Clasa Employee Controller realizeaza interacțiunea dintre modelul corespunzator pentru lantul de magazine de parfumuri și dintre interfața pentru angajat, iar Administrator Controller realizeaza interacțiunea dintre modelul corespunzator pentru utilizator și dintre interfața pentru administrator. In aceste clase se implementeaza efectiv ascultatorii pentru butoane si tot aici se notifica viewl daca operatia care se executa este sa n schimbata si daca operatia este schimbata viewl va efectua diverse moduri de a se ocupa de aceste operatii.

In aplicatia pentru server avem doua clase care se ocupa de conectarea clientilor si de procesarea de requesturi pentru accesarea bazei de date. De aceasta data clasele de baza de date sunt intr-o aplicatie diferita asa cum este si normal in viata de zi cu zi, intr-o parte doar interfata iar in cealalta parte baza de date. Mai avem aici 4 clase care respecta sablonul decorator care se ocupa de transmiterea notificarilor in cazul in care are loc cea mai mica schimbare a credentialelor.

Acum vreau sa descriu sabloanele de proiectare utilizate la acest proiect. **Observer** l am utilizat pentru a notifica atat controllerele cat si view-urile in momentul in care acestea trebuie sa execute o anumita operatie. Scopul lui este de a face un MVC activ. **Singleton** pentru a realiza conexiunea cu baza de date deoarece pentru acest caz avem nevoie de o singura instanta si numai una. **Buildrer** l am folosit pentru a construi obiecte incomplete pentru cazul in care nu am nevoie sa retin toate datele despre un anumit obiect si ca sa fiu sigur ca se construieste corect un obiect. **Decorator** l am folosit in aplicatia de server pentru a grupa notificarile de email si sms deoarece a fost simplu de implementat si a fost un design pattern care s-a mulat perfect pentru cazul meu de utilizare.

# Descrierea aplicatiei

Graphical user interface

Description automatically generated with medium confidence

Aceasta ete fereastra de login unde utilizatorul se poate loga fie ca si administrator fie ca si angajat. Contine un camp unde se poate introduce un nume de utilizator cat si un camp pentru parola dar si butonul de login. Asfel utilizatorul dupa ce introduce aceste date la apasarea butonului de login se cauta dupa credentialele lui in aceasta baza de date.

Codul care stă în spatele acestei aplicații este unul destul de simplu și basic,am încercat să scriu cât mai clar și simplu toate operațiile menționate mai sus,am împărțit aplicația în clase specifice, am respectat modelul arhitectural MVC , respectiv modelul SOLID,dar si sablonul BUILDER si sablonul OBSERVER, pentru a fi mai simplu și eficient , a modifica sau a adăuga alte functionalități ,fără să se producă modificări majore liniilor de cod existente , ci doar să adăugăm alte linii.

Urmeaza mai jos sa pun celelate doau interfete grafice care fac referire la utilizatorul de tip administrator dar si utilizatorul de tip angajat.

Graphical user interface, application

Description automatically generated

Pot spune ca a fost a patra oara in viata mea cand am lucrat cu expresii landa in java pentru a realiza filtrarea dupa diverse criterii si chiar a fost destul de interesant. Am inteles cum trebuie sa se modeleze viata reala datele din mai multe domenii . Principalul obiectiv al acestei teme este de a ne invata sa lucram cu expresii landa si cu MVC dar si respectarea metodologiei CRUD deoarece acest lucru ne poate usura foarte mult viata deoarece nu mai suntem nevoiti sa scriem asa de mult cod; sa recunoastem ca un programator bun va sti intodeauna sa isi refoloseasca codul pentru a creea o mai buna stabilitate peste timp a programului.

Graphical user interface, application

Description automatically generated

# Tabelele din baza de date

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface, text

Description automatically generated