DOCUMENTATIE

TEMA *4*

NUME STUDENT: Stefan Razvan Mihai
GRUPA: 30224

CUPRINS

1. Obiectivul temei

Obiectivul principal al acestei teme este dezvoltarea unei aplicatii cu ajutorul careia sa se gestioneze un restaurant. Acest restaurant are o aplicatie prin care se pot efectua livrari. Utilizatorii aplicatiei pot efectua urmatoarele: administratorul acesteia se poate loga in aplicatie si poate gestiona meniul restaurantului si poate obtine anumite rapoarte despre comenzi, clientii se pot inregistra si loga in aplicatie si pot efectua comenzi, dar si sa caute produse dupa mai multe criterii, angajatul este notificat atunci cand are de realizat o noua comanda. Aplicatia persista datele despre comenzi anterioare si meniu folosind serializarea.

Obiective secundare:

-analizarea problemei şi identificarea cerinţelor
-proiectarea unei aplicatii (diagrame de entitati, diagrame cu cazuri de utilizare etc.)
-implementarea aplicatiei
-testarea aplicatiei

2. Analiza problemei, modelare, scenarii, cazuri de utilizare

Serializarea reprezinta transformarea unui obiect intr-o secventa de octeti din care sa poata fi refacut ulterior obiectul original. Deserializarea reprezinta procesul invers, prin care se citesc obiectele serialzate si se refac obiectele originale.

Cerinte functionale:

-posibilitatea de a introduce, edita, sterge produse.
-stocarea datelor in mod constant (nu doar cand programul ruleaza) prin serializare(scrierea si citirea din anumite fisiere)
-plasarea de comenzi (clientii pot filtra produsele, introducand valorile pe care le doresc pentru proprietatile itemului din meniu)

Cazuri de utilizare:

Utilizatorul acestui program poate fi administrator, client sau angajat.

Administratorul:

- adauga un produs in meniu prin completarea unor campuri cu valorile acelui produs si prin apasarea butonului de adaugare
- sterge un produs din meniu prin selectarea acestuia si apasarea butonului de delete
- editeaza un produs, introducand in aceleasi campuri ca cele de la adaugare, valorile care vrea sa fie modificare (daca un camp ramane null, nu se va schimba acel camp din produsul original) si apasand butonul de editare
- vizualizeaza produsele apasand butonul corespunzator.
- creeaza un produs compus, format din mai multe produse simple, selectand produsele necesare si apasand butonul de creare.
- poate vizualiza 4 rapoarte : comenzile dintr-un anumit interval orar indiferent de data, produsele comandate de mai mult de un numar dat de ori, clientii care au comandat de mai multe ori decat un numar specificat (si comenzile fiind fiecare mai scumpa decat un pret specificat),

produsele comandate intr-o anumita zi, impreuna cu numarul de dati in care au fost comandate in acea zi.‘

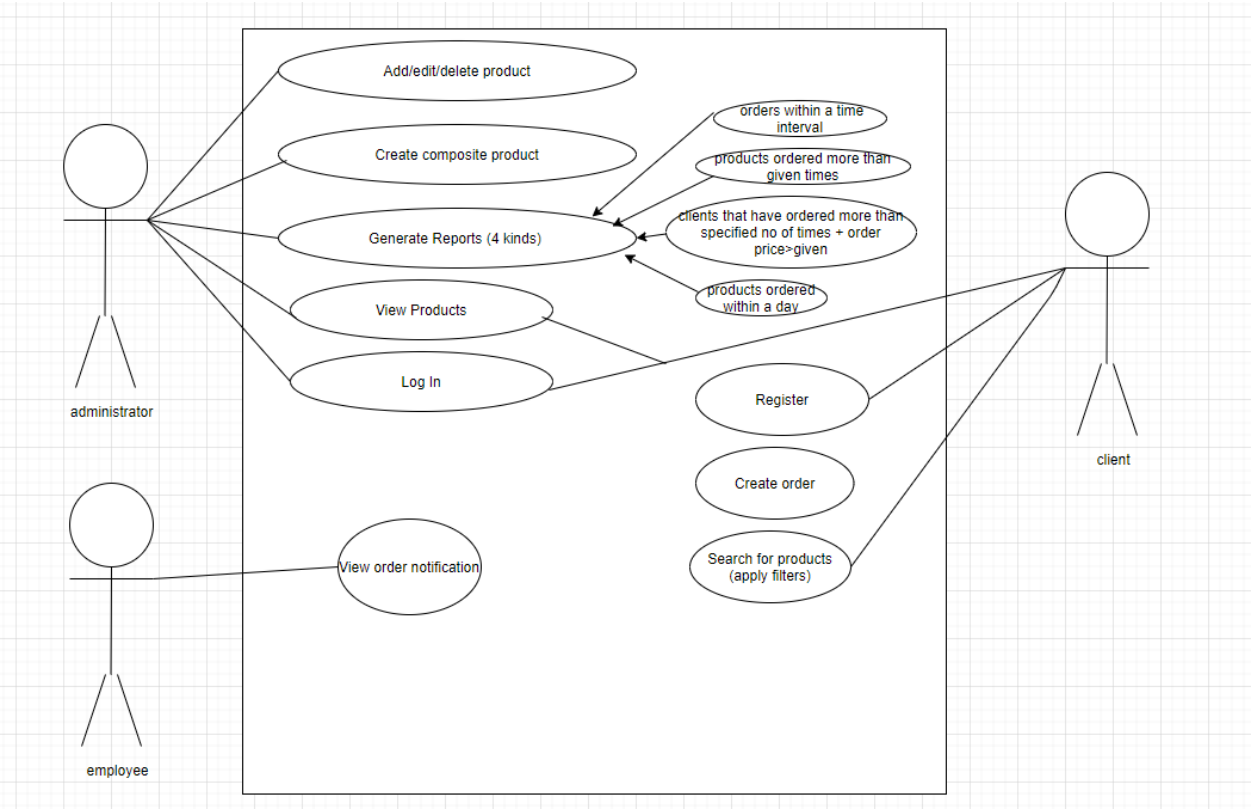
Clientii:

- se pot inregistra si loga in aplicatie
- pot vizualiza produsele din meniu
- pot cauta produse in functie de : cuvant cheie, pret, calorii, sodiu, rating, proteine, grasimi
- pot plasa comenzi formate din mai multe produse (inclusiv produse compuse)

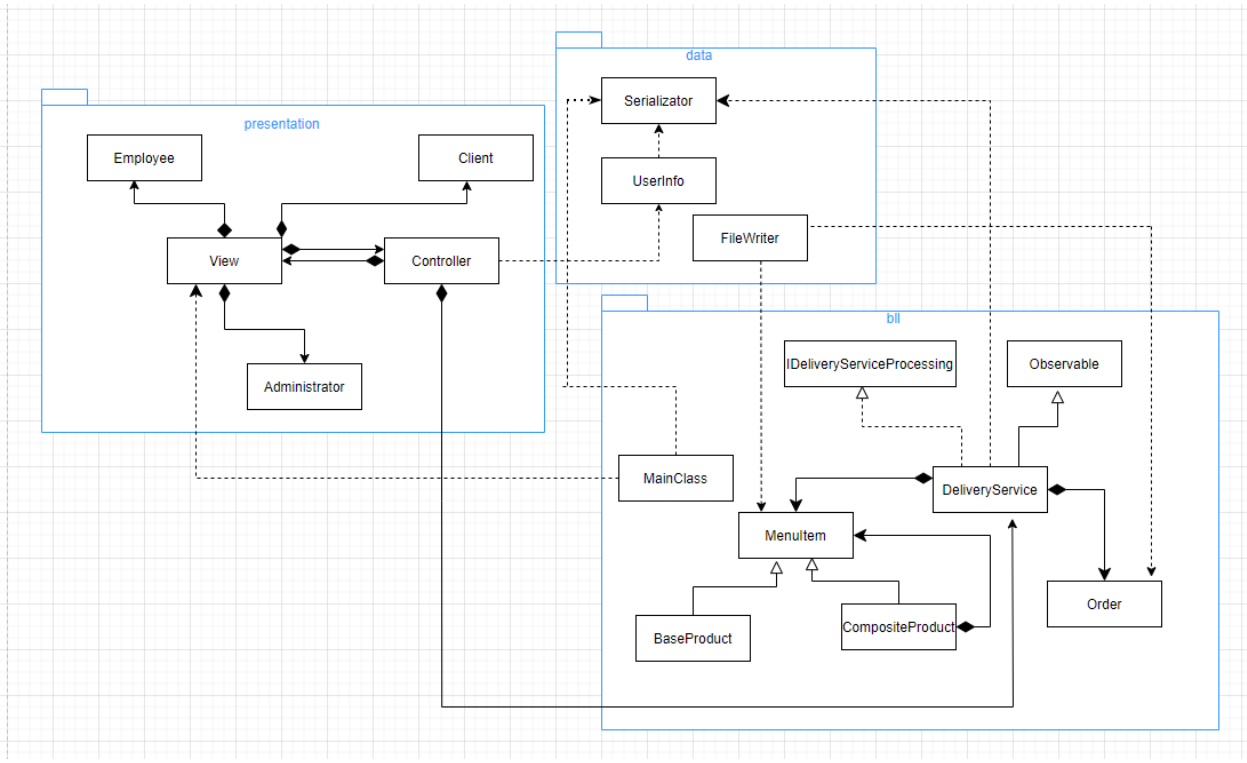
Angajatii:

-primesc notificari atunci cand se plaseaza comenzin noi

Diagrama de use-case:



3. Proiectare



4. Implementare

Clasa BaseProduct

Contine atributele: titlu, rating, numar de calorii, numar de proteine, numar de grasimi, sodiu, pret.
Aceasta clasa reprezinta un produs de baza, “atomic”, care nu este format din alte produse existente in meniu.
Clasa implementeaza interfata serializable si mosteneste clasa MenuItem (de unde are si atributele enumerate mai sus).

Clasa CompositeProduct

Atribute: titlu, items. Items reprezinta o lista de produse din care este format produsul compus.
Si aceasta clasa implementeaza interfata Serializable si extinde clasa MenuItem.

Clasa MenuItem

Contine atributele: titlu, rating, numar de calorii, numar de proteine, numar de grasimi, sodiu, pret.

Reprezinta clasa parinte pentru clasele BaseProduct si MenuItem, deoarece acest trio foloseste design pattern-ul Composite (design pattern cu ajutorul caruia se pot crea obiecte compuse cu care se poate lucra ca si cum ar fi obiecte individuale). Aceasta clasa suprascrie functiile hashCode si equals pentru a nu exista duplicate atunci cand se construieste un HashMap care contine MenuItem.

De asemenea, in aceasta clasa este metoda clientFilter, care este folosita pentru cautarea produselor de catre client. Metoda returneaza fals daca atributele obiectului din meniu nu corespund cu cele specificate de client.

Clasa mai contine si metoda toStringComma care returneaza atributele produsului separate cu virgula pentru o citire mai usoara din JList uri.

Clasa Order

Contine atributele: orderID, clientID si dateFormat. orderID reprezinta identificatorul unic al unei comenzi si este un intreg care se autoincrementeaza obtinand numarul de comenzi prin deserializare si adaugand 1 acelui numar. clientID reprezinta numele de utilizator al clientului care plaseaza comanda (clientul care este logat in aplicatie in acel moment). dateFormat reprezinta un string care contine data si ora exacta la care este plasata comanda. Acesta este generat automat in momentul crearii comenzii. Clasa suprascrie metodele equals si hashCode.

Clasa DeliveryService

Aceasta clasa implementeaza interfata IDeliveryServiceProcessing, in care se afla metodele cu care se efectueaza operatiile clientilor si ale administratorului.

Are atributele: meniu si orders. Meniu este un HashSet de MenuItem, in care este stocat meniul restaurantului. Am folosit HashSet pentru a ma asigura ca nu exista produse duplicate in lista. Orders este un HashMap cu cheia de tip Order si valoarea de tip List<MenuItem>. Aici sunt stocate toate comenzile efectuate pe parcursul restaurantului. In order se regasesc id-ul comenzii, id-ul clientului si data la care se efectueaza comanda.

Metode:

-metoda importProducts este folosita pentru a extrage meniul initial dintr-un fisier csv. Metoda foloseste expresii lambda si stream-uri pentru parcurgerea liniilor fisierului csv si crearea si adaugarea de produse.

-metoda addProduct adauga un produs in meniul restaurantului

-metoda editProduct editeaza produsul selectat, modificand valorile cu valorile introduse de administrator in campurile corespunzatoare. Daca un camp este gol, acel atribut nu se va sterge, ci va ramane nemodificat.

-metoda deleteProduct sterge din meniu produsul selectat.

-metoda toProduct transforma un string dat in obiectul corespunzator (de tip base product) (cunoscandu -se ordinea atributelor din string).

-metoda generateReport primeste un intreg criteriu, care poate avea valori intre 1 si 4 si 2 string-uri care reprezinta textul din 2 campuri. Cand criteriul este 1, se creaza un fisier .txt cu comenzile efectuate intr-un interval orar dat de cele 2 campuri. Se folosesc lambda expressions si streams pentru a parcurge hashmap-ul orders si a filtra in mod corespunzator datele. Cand criteriul este2, se creaza un fisier .txt cu produsele care au fost comandate de un numar mai mare decat cel dat de ori. Pentru aceasta, am creat un hashmap de "frecventa", care are cheia un MenuItem si valoarea un Integer. In acest map se actualizeaza numarul de aparitii al produselor in timp ce se parcurg in hashmap-ul original. Cand criteriul este 3, se genereaza un fisier .txt cu numele clientilor care au comandat de minim un numar dat de ori (comenzile avand valoarea totala mai mare decat un alt pret dat de administrator). Pentru acest criteriu am folosit din nou un hashMap auxiliar cu cheia client si valoarea Integer. Algoritmul este asemanator cu cel de la criteriul 2, dar se adauga un filtru de pret in plus. Cand criteriul este 4, se genereaza un fisier .txt cu produsele comandate intr-o anumita zi, impreuna cu numarul de comenzi al acestora.

-metoda reportFilter1 este folosita pentru criteriul 1 din metoda anterioara. Aici, orele date ca interval, precum si ora comenzii la care a ajuns iteratia se transforma in minute (h* 24 + m) pentru o comparare h1<hcurent<h2 mai usoara. In cazul in care h2 este mai mic decat h1 (cazul special in care o ora se afla inainte si cealalta dupa miezul noptii), se modifica acea comparatie negand rezultatul comparatiei hcurent>h2 && hcurent < h1.
-metoda searchProductsClient() este folosita pentru realizarea cautarii de catre client, apeland metoda clientFilter pe fiecare intrare (folosind si functia .filter in lambda expressions)

-metoda addProductToOrder adauga un produs la comanda care este in curs de efectuare (se adauga produsul intr-o lista auxiliara).

-metoda createOrder creeaza un obiect de tip orders, folosind lista auxList si il adauga in orders.

-metoda createCompositeProduct se foloseste tot de auxList pentru a retine produsele initiale din care se doreste crearea unui produs compus.

-metoda viewProducts incarca in interfata grafica produsele din meniu, folosind functiile toString specifice fiecarui tip de produs.

-metoda toString orders creeaza un string care contine datele cheii, precum si produsele din valoarea hashmap-ului orders.

Clasa FileWriter

Aceasta clasa nu contine atribute, insa contine cateva metode folosite pentru crearea fisierelor generate de administrator in functie de criteriu, precum si pentru realizarea fisierelor de tip bill, cu comenzi.

Clasa Serializator

Contine metodele serialize si deserialize, care functioneaza pentru orice atribut din orice clasa care implementeaza Serializable. Serialize primeste calea spre fisierul unde se vor scrie octetii corespunzatori si este de tip void. Deserialize primeste calea spre fisierul din care se vor extrage valori si returneaza Object, pentru a putea fi folosita de orice clase.

Clasa UserInfo

Contine atributele statice finale si de tip transient USER_ADMIN si USER_PASS, care contin userul si parola administratorului aplicatiei. Acestea sunt declarate ca transient, pentru a nu fi serializate deoarece sunt constante.

De asemenea, clasa contine atributul clientInfo de tip HashMap<String, String>, unde sunt stocate datele clientilor inregistrati.

-metoda registerClient adauga in clientInfo datele introduse de client, daca nu coincide user-ul introdus cu un user deja creat

-metoda loginClient ii da acces clientului la aplicatie daca datele introduse coincid cu o pereche de date din hashmap-ul clientInfo. In caz ca nu coincid, se va afisa in consola ce camp nu corespunde.

Aceste 2 metode serializeaza si deserializeaza campul clientInfo.

Clasa View

Contine:

-obiectele din java Swing necesare afisarii ferestrei initiale, unde se selecteaza tipul de utilizator.

-atribute de tip Administrator si Client, unde se afla interfetele pentru logare administrator, respectiv logare si inregistrare client, precum si urmatoarele 2 interfete pentru interactiunea propriu-zisa cu aplicatia.

Pentru majoritatea lucrurilor afisate in interfata s-au folosit obiecte de tip JList, care se folosesc de functia toString a diferitelor obiecte pentru afisarea acestora. De asemenea, la selectarea unui obiect se obtine referinta directa catre acesta.

Clasele administrator si client contin toate componentele corespondente interfetelor acestora, insa adaugarea de comenzi si de actionListenere asupra butoanelor se realizeaza in clasa View.

Clasa Controller

In aceasta clasa se creeaza un obiect de tip DeliveryService, obiect in care vor fi retinute toate datele care persista.

De asemenea, se realizeaza actiunile corespunzatoare tuturor butoanelor din toate interfetele.

In plus, clasa contine metode de tipul redrawScrollPane, unde se redeseneaza in interfata tabelele cu meniul, in urma oricaror operatii care le modifica pe acestea (add product, delete product, edit product, create composite product).

5. Concluzii

Aceasta tema mi s-a parut cea mai voluminoasa dintre cele 4, dar si cea mai complexa din punct de vedere al tehnicilor de programare folosite. Am reusit sa inteleg serializarea, care mi se pare un concept esential din dezvoltarea aplicatiilor care au o utilizare in lumea reala. Am invatat despre expresiile lambda si stream-uri, cu ajutorul carora se pot parcurge si filtra datele intr-un mod mai usor pentru programator (numar de linii foarte redus si o generalitate ridicata). Am reusit sa inteleg design pattern-ul Composite, care mi se pare un mod elegant de a putea trata in acelasi mod obiecte normale precum si obiecte formate din mai multe obiecte de baza dintre cele normale.

De asemenea, am invatat despre folosirea observatorilor pentru declansarea anumitor actiuni odata cu modificarea anumitor campuri.

Ca dezvoltari ulterioare, acestui proiect i-ar putea fi adaugate urmatoarele functionalitati:

- o interfata mai usor de utilizat si mai intuitiva cu mai multe ferestre de tip pop- up;
- posibilitatea de recuperare a parolei unui client in cazul in care acesta o uita (folosind un email sau un numar de telefon al acestui pentru confirmarea identitatii)
- posibilitatea coexistentei a mai multor administratori (cu eventuale permisiuni variabile (de ex: administratorul a poate doar sa creeze produse compuse, nu sa si modifice produsele de baza))
- adaugarea mai multor tipuri de rapoarte care se pot genera (de ex: produsele comandate cel mai putin, produsele cu rating cel mai mic, frecventa comenzilor din partea anumitor clienti)
- imposibilitatea comandarii produselor care nu se afla in stoc momentan
- abilitatea clientului de a oferi feedback in legatura cu comanda si de a da note felurilor de mancare pentru a se putea actualiza rating-ul

6. Bibliografie

https://dsrl.eu/courses/pt/materials/A4_Support_Presentation.pdf

https://refactoring.guru/design-patterns/composite

https://profs.info.uaic.ro/~acf/java/slides/extra/serializare_slide.pdf

https://www.w3schools.com/java/java_lambda.asp#:~:text=A%20lambda%20expression%20is%20a,the%20body%20of%20a%20method.