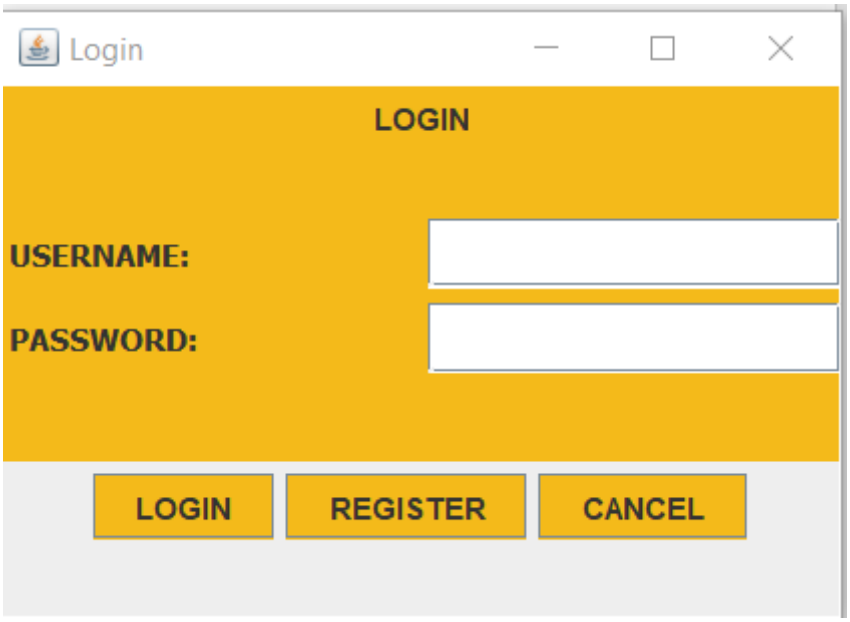
MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
SPECIALIZAREA CALCULATOARE SI TEHNOLOGIA INFORMATIEI

Sistem de gestiune al livrarilor unei firme de catering



-Documentație-

Ana-Maria Cusco

An academic: 2020 – 2021

Cuprins

Obiectivul temei

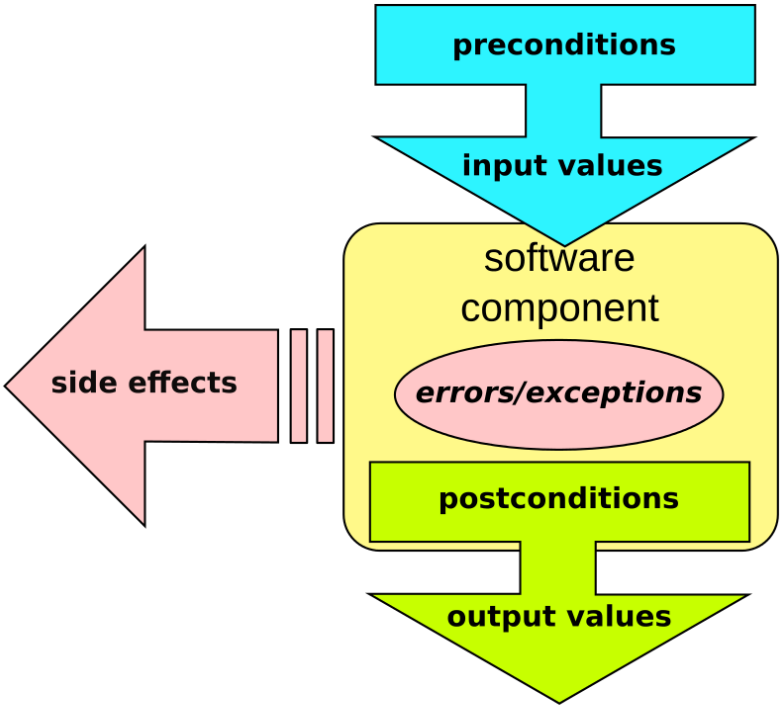
Scopul acestei teme este implementarea unui sistem de management al livrarilor de produse pentru o firma de catering. Sistemul are 3 tipuri de utilizatori care se pot loga utilizand o parola si un username: administrator, angajat si client.

„Food Delivery Management System” este o aplicatie destinata clientilor pentru a le usura munca. Cu totii avem momente cand nu avem chef sau timp sa gatim si recurgem la ideea de a comanda mancare online sau de a merge sa mancam la restaurant. Aplicatia mea isi propune sa vina in sprijinul clientilor, care vor putea sa comande produsele imediat cu doar cateva click-uri. Astfel clientul plaseaza comanda, angajatii firmei de catering vizualizeaza in timp real comenzile si livreaza produsele spre consumatori. Se preteaza pentru un singur restaurant sau mai multe - parteneri ai firmei de catering. Ofera un mod eficient de a comanda mancare care sa fie livrata in cel mai scurt timp. Clientul trebuie sa se autentifice utilizand un username si o parola intr-o interfata de logare pentru a se asigura securitatea informatiilor si poate sa isi selecteze produsele favorite si sa plaseze o comanda. Imediat dupa plasarea comenzii angajatii vor fi notificati despre aceasta, in acest fel realizandu-se procesarea eficienta a comenzilor.

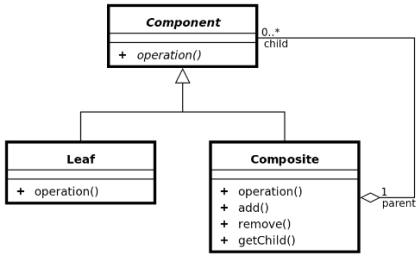
In ziua de azi fiecare domeniu tehnic incearca sa faca viata omului mai usoara. In ultimii ani a existat o crestere enorma ale restaurantelor care ofera servicii online pentru a comanda produse.

In termeni simpli, „Online Food Ordering System” (sistem se comenzi de mancare online) este o modalitate usoara si convenabila pentru un client de a comanda mancare/produse online fara a fi nevoie sa mearga la restaurant. Clientii nu trebuie sa aiba cunostinte tehnice avansate pentru a folosi aplicatia. Ofera toate datele intr-o singura pagina unde clientii pot cauta si sorta produsele dupa bunul plac, si sa le adauge in cos cu un singur click. Dupa adaugarea produselor in cos, se calculeaza pretul comenzii, iar clientul poate plasa comanda cu un singur buton, urmand ca produsele sa ajunga la el in cel mai scurt timp.

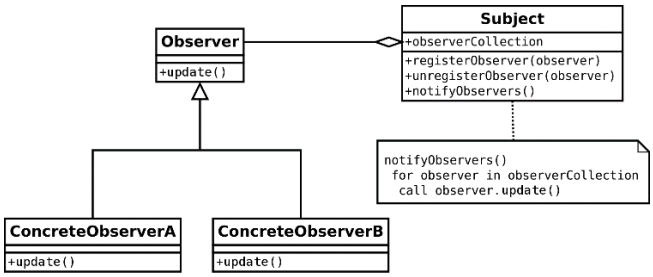
In realizarea aplicatiei am avut in vedere utilizarea tehnicii de programare Design by Contract, folosirea de Design Patterns precum Observer si Composite si implementarea serializarii pentru a stoca datele sistemului. Mai mult corespondenta comanda-meniu este realizata prin intermediul unei structuri de tip HashMap. Obiectivul principal consta in familiarizarea cu aceste concepte si utilizarea lor in construirea logicii aplicatiei.



Design By Contract Programming Technique



Composite Design Pattern



Observer Design Pattern

Analiza problemei, modelare scenarii, cazuri de utilizare

In analiza problemei pornim de la cele 3 tipuri de utilizatori: client, administrator si angajat

si de la atributiile fiecaruia in parte.

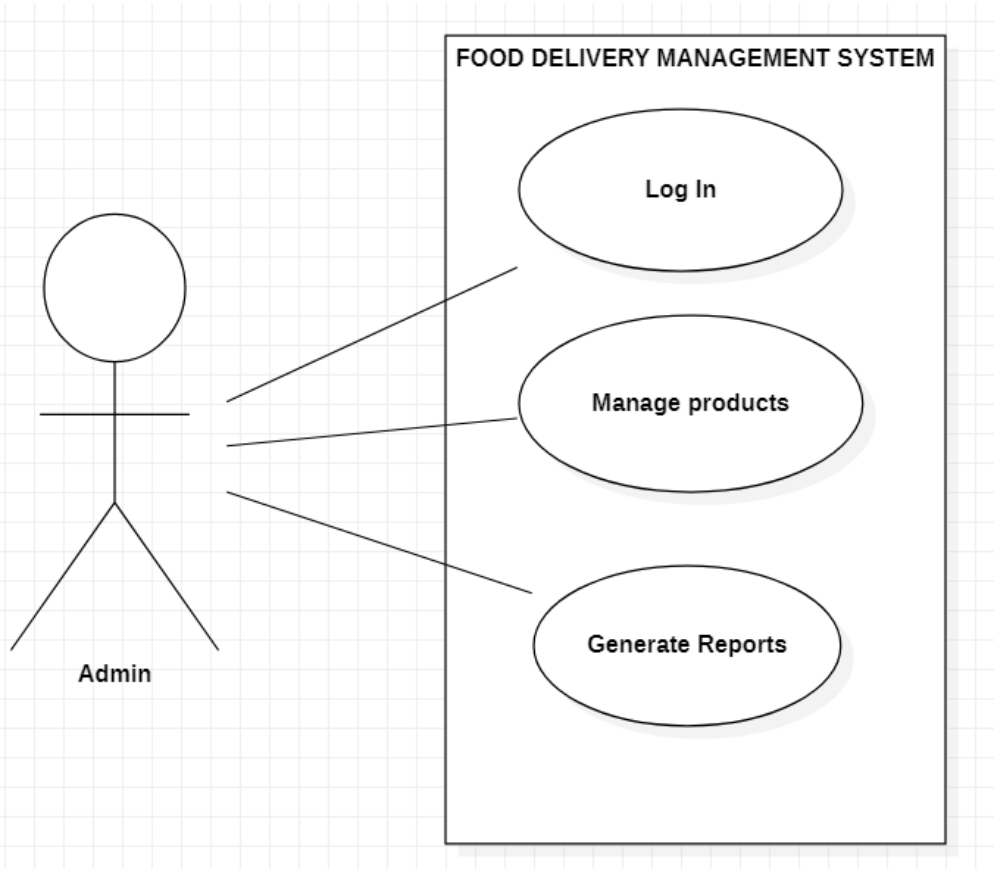
Administatorul poate efectua urmatoarele operatii:

-sa importe setul initial de produse cu care va fi populat meniul dintr-un fisier .csv

- sa gestioneze produsele din meniu prin adaugarea/stergerea/modificarea acestora si crearea unui meniu nou compus din produsele existente

-sa genereze rapoarte despre comenzi cu urmatoarele criterii:

- Intervalul orar al comenzilor – se vor selecta toate comenzile plasate intr-un anumit interval orar indiferent de zi
- Produsele comandate de mai multe ori de cat un numar specificat
- Clientii care au comandate de un numar de ori mai mare decat un numar specificat si valoarea comenzii depaseste o suma specificata
- Produsele comandate intr-o anumita zi si de cate ori au fost comandate



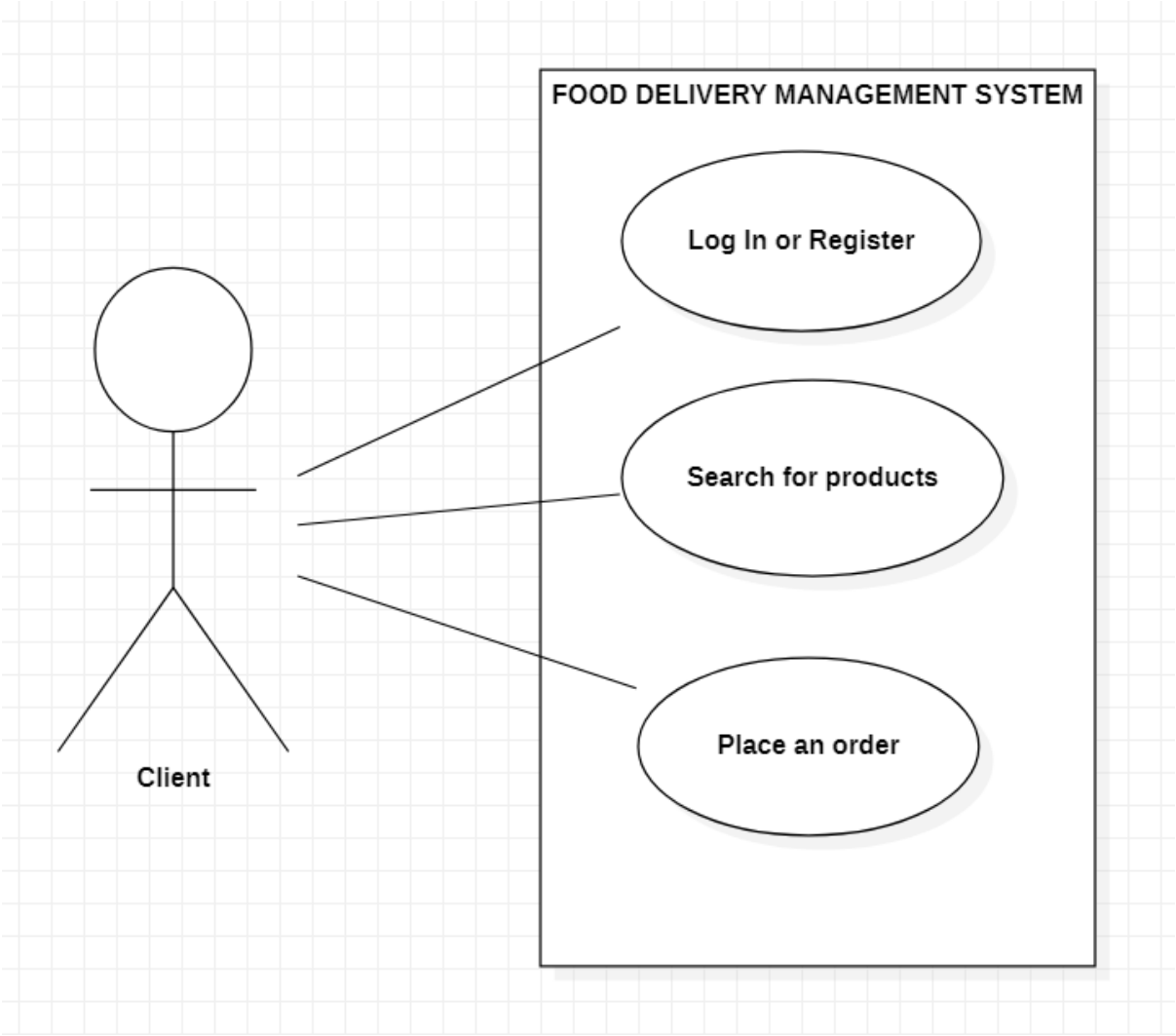
Clientul poare sa:

-se inregistreze in aplicatie sau sa se autentifice folosind un username si o parola

-sa vada lista de produse din meniu

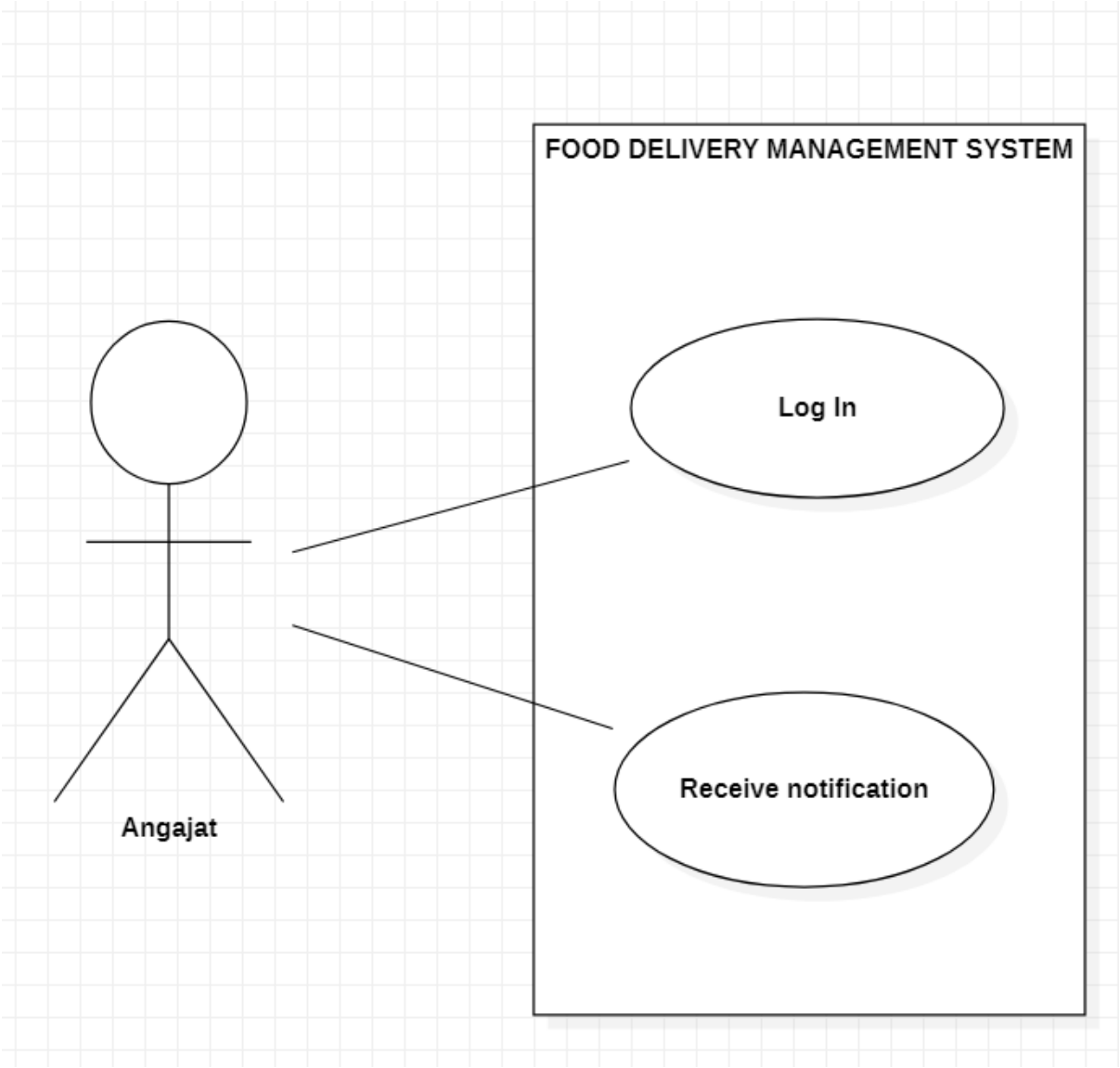
-sa caute produse dupa diverse criterii

-sa plaseze comenzi adaugand diverse produse: pentru fiecare comanda se va specifica data si ora, pretul total al comenzii si se va genera o factura continand detaliile comenzii



Angajatul

-este notidicat de fiecare data cand se plaseaza o noua comanda de catre un client pentru a putea sa livreze produsele



Alte cerinte specifice:

- Definirea clasei IDeliveryServiceProcessing care sa contina operatiile principale care pot fi executate de client si administrator dupa cum urmeaza:
  -Administrator: importa produse, gestioneaza produsele din meniu, genereaza rapoarte
  -Client: creaza o comanda noua care presupune calcularea pretului per comanda si generatea unei facturi in format .txt, cautarea produselor dupa mai mutle criterii
- Definirea si implementarea claselor din diagrama UML de mai sus
  -Folosurea design pattern-ului Composite pentru definirea claselor MenuItem, BaseProduct si CompositeProduct
  -Folosirea design pattern-ului observer pentru a notifica un angajat de fiecate data cand se plaseaza o comanda noua
- Implementarea clasei DeliveryService folosind o structura predefinita in JCF(Java Collection Framework) care foloseste ca si structura de date un hashtable. Cheia tabelei va fi generata folosind clasa Order, iar fiecare comanda va avea asociata o colectie de MenuItems
  -Definirea unei structuri Map<Order, Collection <MenuItem>> pentru a stoca informatiile despre o comanda in clasa DeliveryService. Cheia Map-ului va fi formata dintr-un obiect de tip Order, pentru care se va suprascrie metoda hashCode() pentru a calcula valoarea has-ului in cadrul Map-ului din atributele clasei Order (OrderID, date, etc.)
  -Definirea unei structuri de tip Collection<MenuItem> care va stoca meniul (toate produsele) oferite de firma de catering.
  -Definirea unei metode de tip „well formed” pentru clasa DeliveryService.
  -Implementarea clasei folosind metoda Design By Contract (contine pre,post conditii, invarianti si asertiuni)
- Produsele de baza folosite initial pentru a popula meniul (obiectul DeliveryService) vor fi citie de din fisierul **products.csv** folosind expresii lamda si procesare pe stream-uri
- Produsele din meniu, comenzile si informatiile despre utilizatori vor reprezenta informatie persistenta, si vor si salvate folosind serializarea pentru a fi disponibile pentru accesari ulterioare ale sistemului folosind deserializarea

Proiectare

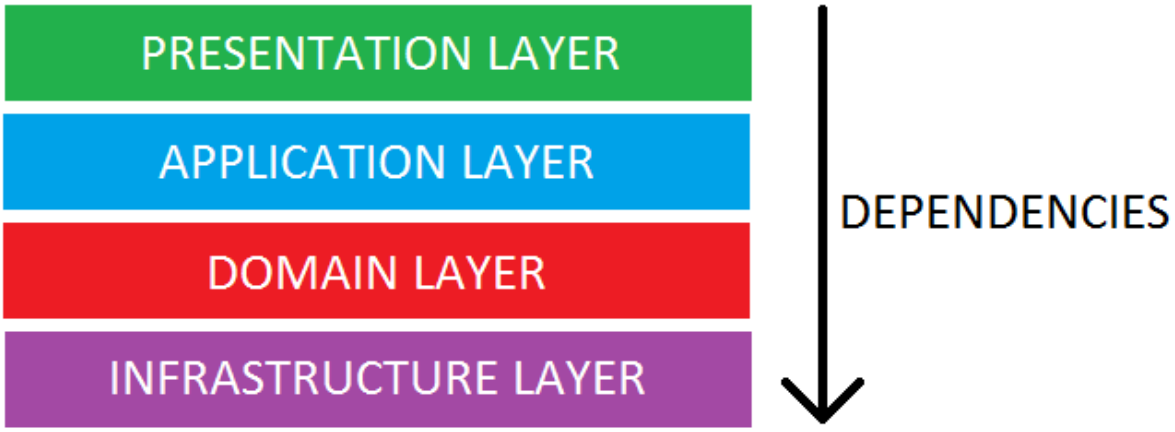
Decizii de proiectare

In construirea acestei aplicatii, si dupa cum specificau cerintele am ales sa folosesc o arhitectura pe mai multe niveluri (eng. multitier architecture).

Arhitectura de aplicații N -tier oferă un model prin care dezvoltatorii pot crea aplicații flexibile și reutilizabile. Prin separarea unei aplicații în niveluri, dezvoltatorii dobândesc opțiunea de a modifica sau adăuga un anumit strat, în loc să refacă întreaga aplicație. O arhitectură logică

multistratificată pentru un sistem informațional cu un design orientat obiect este compusă de obicei din mai multe straturi dupa cum urmeaza:

- Stratul de prezentare (cunoscut și ca stratul UI, stratul de vizualizare, nivelul de prezentare în arhitectura pe mai multe niveluri)
- Stratul de aplicație (cunoscut și ca strat de serviciu sau stratul de control GRASP)
- Stratul de afaceri (alias strat de logică de afaceri (BLL), strat de domeniu)
- Stratul de acces la date (cunoscut și ca strat de persistență , jurnalizare, rețea și alte servicii care sunt necesare pentru a sprijini un anumit strat de afaceri)



Structuri de date

Ca si structuri de date am folosind JCF, mai precis **List<?>** pentru stocarea produselor din meniu si a utilizatorilor si **HashMap<?, ?>** pentru stocarea comenzilor.

Pentru afisarea produselor in interfata grafica, am folosit ca un JTabel care are foloseste in spate un model ce contine o lista de produse.

Pentru mentinerea persistentei datelor am folosit procesul de serializare, care salveaza toata informatia sistemului in fisiere, iar la o noua accesare starea sistemului este restaurata din punctul in care a ramas.

Proiectare clase, diagrame UML

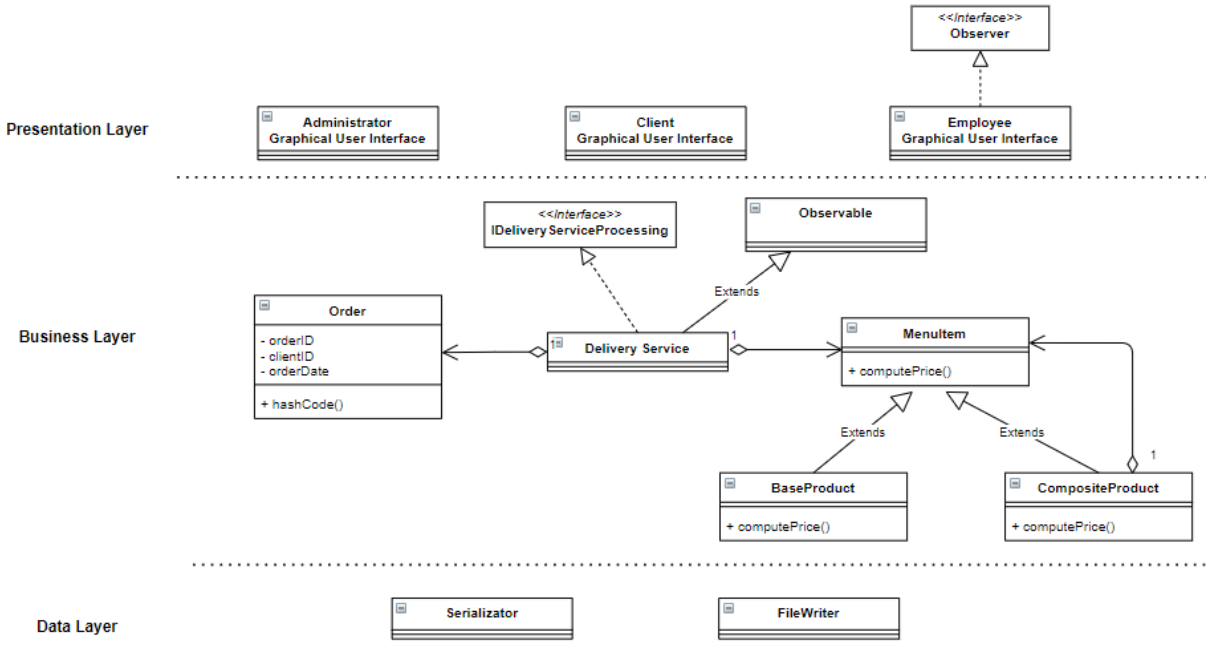


Diagrama de clase de la care s-a pornit in realizarea aplicatiei

Pe langa acestea s-au mai adaugat cateva clase necesare in realizarea aplicatiei.

- La nivelul pachetului **Data Layer** regasim clasele **Serializator** si **FileWriter**.

Clasa **Serializator** se ocupa cu serializarea datelor aplicatiei, in acest fel mentinandu-se persistenta acestora. Clasele care se serializeaza implementeaza interfata de tip marker Serializable pentru a notifica acest lucru. Informatiile aplicatiei se vor salva in 3 fisiere: unul care contine detaliile despre un user: id, username, parola si rol (client, admin sau angajat), alt fisier care contine meniul (List<MenuItem>), si ultimul care contine comenzile facute pana la momentul actual. Serializarea se realizeaza cu ajutorul obiectelor de tipul FileOutputStream/ FileInputStream si ObjectOutputStream/ObjectInputStream.

Despre Serializare ...

Ce este serializarea ?

Serializare = transformarea unui obiect intr-o secventa de octeti, din care sa poata fi refacut ulterior obiectul original.

Procesul invers, de citire a unui obiect serializat pentru a-i reface starea originala, se numeste **deserializare**.

Referintele care construiesc starea unuiobiect formeaza o intreaga retea de obiecte.

• DataOutputStream, DataInputStream

• ObjectOutputStream, ObjectInputStream

Utilitatea serializarii

• Mecanism simplu de utilizat pentrusalvarea si restaurarea datelor.

• Persistenta obiectelor

• Compensarea diferentelor intre sisteme de operare

• Transmiterea datelor in retea

• RMI (Remote Method Invocation)

• Java Beans - asigurarea persistentei componentelor.

Clasa **FileWriter** se ocupa cu printarea intr-un fisier .txt a facturii generate in urma plasarii unei noi comenzi, si de asemenea scrie in alt fisier rapoartele generate de catre administator.

Tot la nivelul acesta am adaugat clasa **User**, care stocheaza informatii referitoare la un utilizator al sistemului: ID, username, password si rol : client, admin sau angajat. Atat username-ul cat si parola sunt folosite pentru a identifica user-ul la logare.

- Pachetul **BusinessLayer** contine urmatoarele clase: BaseProduct, CompositeProduct, MenuItem, DeliveryService, IDeliveryServiceProcessing si Order.

Clasele **BaseProduct, MenuItem** si **CompositeProduct** sunt implementate folosind design pattern-ul Composite. Astfel atat clasa CompositeProduct cat si clasa BaseProduct extind MenuItem, iar clasa CompositeProduct contine, in plus, o lista de obiecte de tip MenuItem.

Clasa **MenuItem** este o clasa abstracta si are urmatoarele variabile de instanta*: title, rating, fat, calories, sodium, price*, iar fiecare dintre cele doua clase descendente are propria implementare pentru calculul acestor caracteristici ale unui produs. Cu ajutorul acestei tehnici putem sa tratam un grup de obiecte intr-un mod similar cu un singur obiect. Prin folosirea acestei metode declaram o referinta la un MenuItem si instantiem obiectul fie ca un BaseProduct fie ca un CompositeProduct. Prin apelarea unei metode, la runtime se va apela metoda din clasa a carei instanta am declarat-o.

Clasa **Order** stocheaza informatii despre o comanda precum: orderID, clientID, orderDate. Totodata reprezinta cheia pentru HashMap<Order, List<MenuItem>> din clasa DeliveryService unde sunt stocate comenzile firmei de catering. Pentru a ne asigura ca intrarile in tabela sunt unice, in clasa Order a fost suprascrisa metoda **hashCode()** cu atributele acestei clase.

Clasa **DeliveryService** implementeaza functionalitatea aplicatiei si defineste toate operatiile executate de administrator si de catre client. Aceasta clasa foloseste tehnica Design By Contract. Aceasta clasa isi defineste comportamentul prin implementareainterfetei **IDeliveryServiceProcessing** care defineste comportamentele fiecarui tip de utilizator si ce operatii poate sa realizeze.

Despre „Design By Contract" ...

- reprezinta un "contract" care specifica restrictiile la care trebuie sa se supuna datele de intrare ale unei metode, valorile posibile de iesire si starile in care se poate afla programul

- aceste restrictii sunt date sub forma unor:

a) preconditii: reprezinta obligatiile pe care datele de intrare ale unei metode trebuie sa le respecte pentru ca metoda sa functioneze corect

b) postconditii: reprezinta garantiile pe care datele de iesire ale unei metode le ofera

c) invarianti: reprezinta conditii impuse starilor in care programul se poate afla la un moment dat

Aceasta clasa implementeaza urmatoarele functionalitati:

- Operatii realizate de ADMIN
  - void importProducts(); //import products from .csv file
  - void deleteMenuItem (MenuItem menuItem); //delete menu item
  - void createMenuItem (MenuItem menuItem); //add menu item to menu
  - void addProduct(MenuItem menuItem); //add product to menu item
  - void deleteProduct(MenuItem menuItem); //delete product from a composite
  - void modifyMenuItem (MenuItem menuItem);
    // Report: time interval of the orders
  - Map<Order,List<MenuItem>> generateReport0(int startHour, int endHour);
    // Report: the products ordered more than a specified number of times so far.
  - List<MenuItem> generateReport1(int inputValue);

// Report: The clients that have ordered more than a specified number of times and the value of the order was higher than a specified amount.

- List<User > generateReport2(int inputValue, int amount);

// Report: the products ordered within a specified day with the number of times they have been ordered.

- Map<MenuItem,Long> generateReport3(Date selectedDate);

➢ Operatii realizate de CLIENT

void createOrder(Order order, List<MenuItem> menu, String userName); //place an order

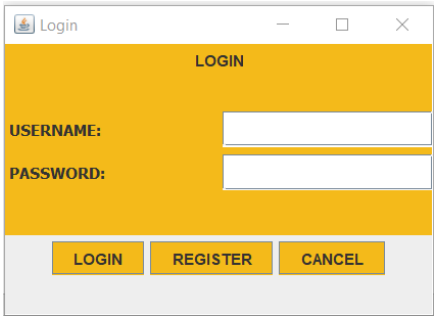
List<MenuItem> searchProduct(String criteria, String value); //search for products

❖ Pachetul **PresentationLayer** inglobeaza urmatoarele clase:AdminView, ClientView, EmployeeView, LoginView, ProductsTable, ReportsFrame precum si pachetul Controller care gestioneaza clasele View si defineste ascultatori pentru butoate, facand legatura intre stratul de Bussiness si interfata grafica.

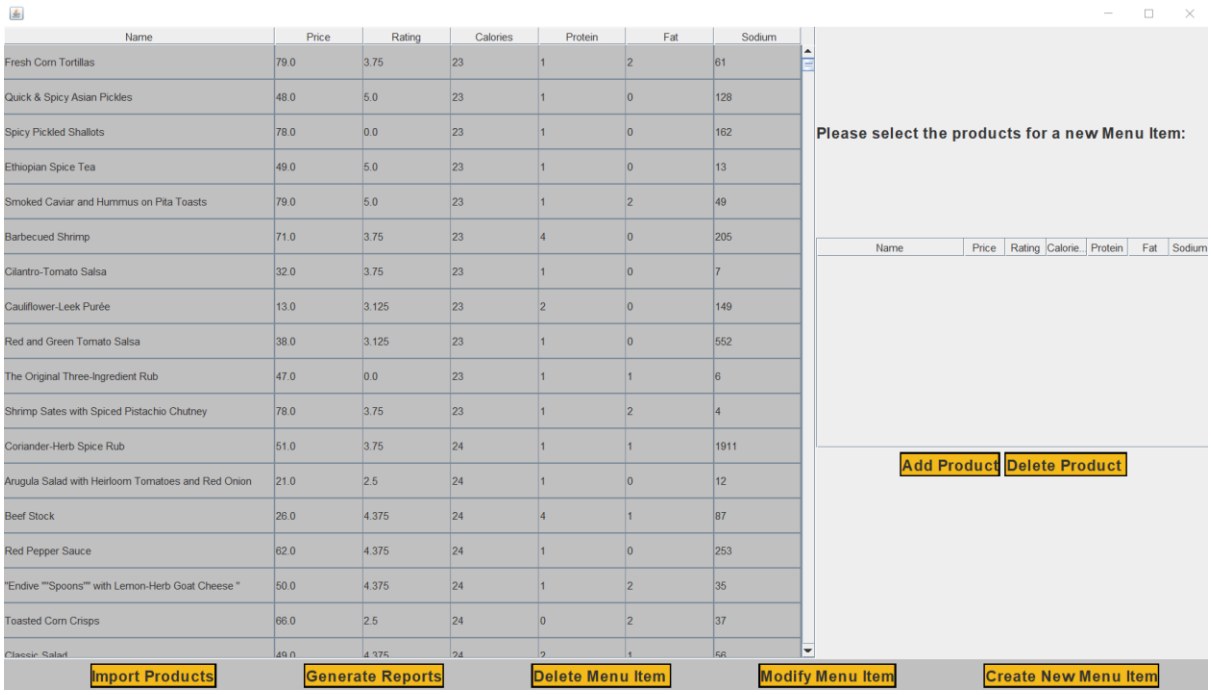
Clasa **EmployeeView** implementeaza **interfata Observer**, deoarece un angajat trebuie sa fie notificat in momentul in care se plaseaza o noua comanda de catre un client. Obiectul observat in cazul acesta este clasa DeliveryService care implementeaza **interfata Observable**, iar observatorul este clasa Employee View. In metoda createOrder() se semnaleaza un eveniment, se trimite o notificare catre observator, si de aici se trimite argumentul metodei update() pe care o suprascrie clasa EmplyeeView, afisandu-se notificarea.

Alt pachet prezent in proiect este **pachetul Utils** care inglobeaza cateva clase utile precum **DisplayableObjectTable, ObjectTableModel, ToStringHelper, JTableUtil**.

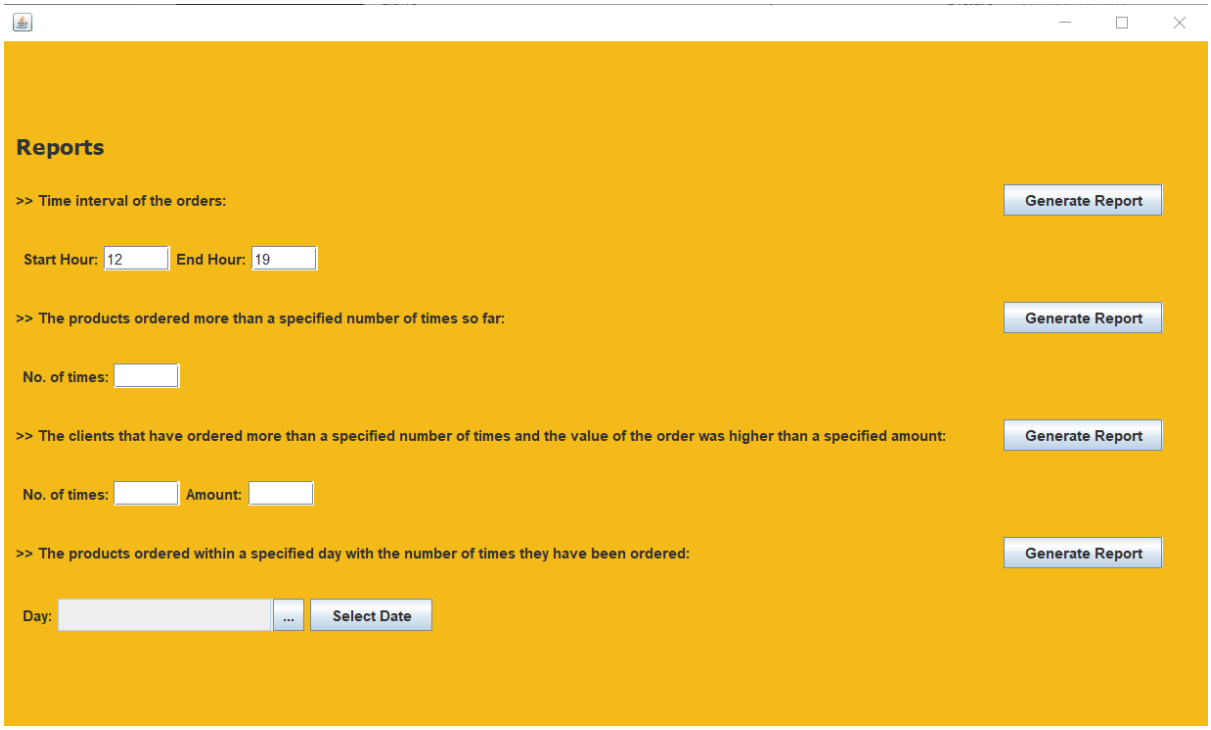
Interfata Utilizator



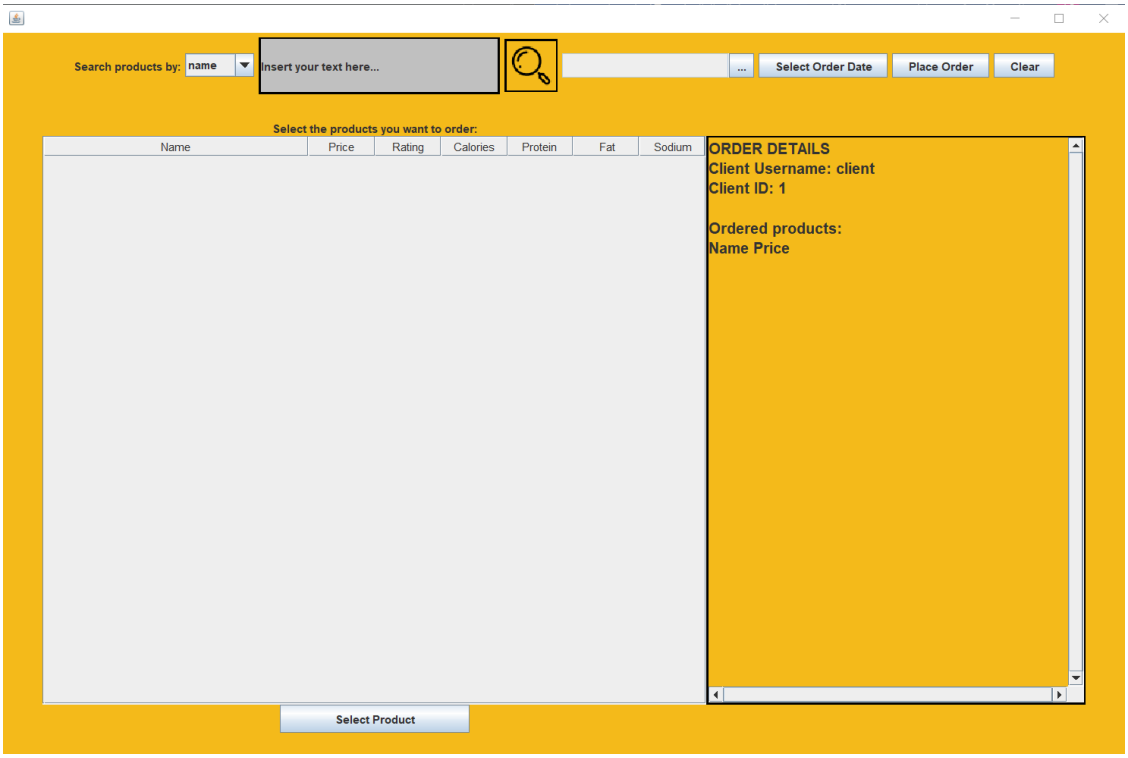
➢ Pagina de Login



Admin View



Generate Reports Frame



Client View

Implementare

Proiectul „Food Delivery Management System” este o aplicatie desktop realizata in limbajul de programare Java folosind mediul de dezvoltare IntelliJ IDEA.

- Implementarea claseiDeliveryService, metodele **createOrder()**

```
@Override
public void createOrder(Order order, List<MenuItem> menu, String userName) {
    assert order != null: " The new order can't be null!";
    assert menuItems.size() >= 1 : " You need to add at least one item!";
    orders.put(order, menu);

    StringBuilder message =new StringBuilder();
    message.append("A new order has been placed!\n");
    message.append("ORDER DETAILS\n");
    message.append(order);
    message.append("\n");

    setChanged();
    notifyObservers(message);
    double totalPrice= computeOrderPrice(menu);
    FileWriterClass.printBill(order, userName, menu, totalPrice);
    Serializator.serializeOrder(orders);
}
```

- Implementarea clasei Serializator, mai precis **serializeUser()** si **deserializeUser()**

```
public static void serializeUser(List<User> users){
    try {
        FileOutputStream fileOutputStream = new FileOutputStream( name: "users.txt");
        ObjectOutputStream objectOutputStream = new ObjectOutputStream(fileOutputStream);
        objectOutputStream.writeObject(users);
    } catch (IOException e) { e.printStackTrace(); }
}

public static List<User> deserializeUser(){
    ArrayList<User> users = null;
    try {
        FileInputStream fileInputStream = new FileInputStream( name: "users.txt");
        ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);
        users = (ArrayList<User>) objectInputStream.readObject();
    } catch (IOException | ClassNotFoundException e){ e.printStackTrace(); }

    return users;
}
```

- Implementarea interfetei **IDeliveryServiceProcessing** folosind tehnica Design By Contract

```
public interface IDeliveryServiceProcessing {
    //ADMIN operations
    /**
     *   @post menuItems.size() >0
     */
     void importProducts();

    /**
     *
     * @inv emptyList()
     * @pre index >= 0 && index < listSize()
     * @pre menuItems.size() >0
     * @post listSize() == listSize()@pre - 1
     */
     void deleteMenuItem (MenuItem menuItem);
```

Rezultate

Rezultatele se pot observa direct in interfata grafica, iar efectul serializarii/deserializarii este direct vizibil intrucat starea anterioara a sistemului este salvata la o noua logare.

De asemenea prin utilizarea tehnicii Design By Contract pentru clasa DeliveryService este verificata corectitudinea datelor, si nu se permit propagarea unor erori care sa ajunga sa fie serializate. Folosind assert-uri este foarte usor de depistat o problema, si de facut debug.

Concluzii

Din aceasta tema am invatat tehnica Design By Contract, cum sa serializez datele pentru a le mentine persistente, desgin pattern-urile Observer si Composite si cum sa lucrez cu stream-uri si expresii lambda. Consider ca a fost un proiect util, in urma caruia am reusit sa descopar lucruri noi.

Bibliografie

[1] https://en.wikipedia.org/wiki/Multitier_architecture

[2] FUNDAMENTAL PROGRAMMING TECHNIQUES (ASSIGNMENT 4 SUPPORT PRESENTATION)

Lambda expressions and stream processing

o https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html

o https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html

o https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8-streams.html

o https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/

o https://howtodoinjava.com/java8/java-stream-distinct-examples/

Java serialization

o http://www.tutorialspoint.com/java/java_serialization.htm

o https://www.baeldung.com/java-serialization

o https://www.geeksforgeeks.org/serialization-in-java/

o https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html

Java HashMap

o http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html

Java assert

o http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html

o http://javarevisited.blogspot.ro/2012/01/what-is-assertion-in-java-java.html

o http://stackoverflow.com/questions/11415160/how-to-enable-the-java-keyword-assert-in-eclipse-program-wise

o https://intellij-support.jetbrains.com/hc/en-us/community/posts/207014815-How-to-enable-assert

Adding custom tags to javadoc

o https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#tag

ANEXA

Facturarea comenzii

```
ORDER DETAILS:

Client Name:client
ClientID: 1
Order ID: 1
Order Date: Wed May 12 18:08:04 EEST 2021

Products:
Quick & Spicy Asian Pickles 48.0
Arugula Salad with Heirloom Tomatoes and Red Onion 21.0
South American-Style Jícama and Orange Salad 25.0

Total Price:94.0
```

BillNo1.txt

Raport

```
Report:The products ordered within a specified day with the number of times
they have been ordered.

Prosciutto-Wrapped Asparagus Spears 25.0
->2
 South American-Style Jícama and Orange Salad 25.0
->2
 Arugula Salad with Heirloom Tomatoes and Red Onion 21.0
->1
 Ancho-Guajillo Chile Sauce 35.0
->1
 Sugar Snap Pea Tempura 31.0
->1
 Quick & Spicy Asian Pickles 48.0
->1
 Asian Cabbage Salad 37.0
->2
 Olive and Anchovy Stuffed Eggs 76.0
->1
 Arugula with Lemon and Olive Oil 99.0
->1
```