# 

**DOCUMENTATIE TEMA 4**

**Food delivery management system**

**Varvara Raluca Ana-Maria**

**Grupa 30222**

**Profesor Laborator: Oana Andreea Marin**

Contents

[1](file:///D:\FACULTATE\AN%202\SEM%202\TP\Assignments\PT2021_30222_Varvara_Raluca_Assignment_3\Documentatie_Tema_3.docx#_Toc69723038)

[**1.** **Obiectivul temei** 3](file:///D:\FACULTATE\AN%202\SEM%202\TP\Assignments\PT2021_30222_Varvara_Raluca_Assignment_3\Documentatie_Tema_3.docx#_Toc69723039)

[**2.** **Analiza** 3](file:///D:\FACULTATE\AN%202\SEM%202\TP\Assignments\PT2021_30222_Varvara_Raluca_Assignment_3\Documentatie_Tema_3.docx#_Toc69723040)

[**3.** **Proiectarea sistemului** 4](file:///D:\FACULTATE\AN%202\SEM%202\TP\Assignments\PT2021_30222_Varvara_Raluca_Assignment_3\Documentatie_Tema_3.docx#_Toc69723041)

[3.1 Decizii de proiectare 4](file:///D:\FACULTATE\AN%202\SEM%202\TP\Assignments\PT2021_30222_Varvara_Raluca_Assignment_3\Documentatie_Tema_3.docx#_Toc69723048)

[3.2 Divizarea in subsisteme/pachete 4](file:///D:\FACULTATE\AN%202\SEM%202\TP\Assignments\PT2021_30222_Varvara_Raluca_Assignment_3\Documentatie_Tema_3.docx#_Toc69723049)

[3.3 Împărțirea în clase 5](file:///D:\FACULTATE\AN%202\SEM%202\TP\Assignments\PT2021_30222_Varvara_Raluca_Assignment_3\Documentatie_Tema_3.docx#_Toc69723050)

[3.4 Diagrama de clase 8](file:///D:\FACULTATE\AN%202\SEM%202\TP\Assignments\PT2021_30222_Varvara_Raluca_Assignment_3\Documentatie_Tema_3.docx#_Toc69723051)

[3.5 Structuri de date 9](file:///D:\FACULTATE\AN%202\SEM%202\TP\Assignments\PT2021_30222_Varvara_Raluca_Assignment_3\Documentatie_Tema_3.docx#_Toc69723052)

[3.6 Interfețe folosite/definite 9](file:///D:\FACULTATE\AN%202\SEM%202\TP\Assignments\PT2021_30222_Varvara_Raluca_Assignment_3\Documentatie_Tema_3.docx#_Toc69723053)

[3.7 Interfața 10](file:///D:\FACULTATE\AN%202\SEM%202\TP\Assignments\PT2021_30222_Varvara_Raluca_Assignment_3\Documentatie_Tema_3.docx#_Toc69723054)

[4. Implementare 12](file:///D:\FACULTATE\AN%202\SEM%202\TP\Assignments\PT2021_30222_Varvara_Raluca_Assignment_3\Documentatie_Tema_3.docx#_Toc69723055)

[5. Rezultate 18](file:///D:\FACULTATE\AN%202\SEM%202\TP\Assignments\PT2021_30222_Varvara_Raluca_Assignment_3\Documentatie_Tema_3.docx#_Toc69723056)

[6. Concluzii 18](file:///D:\FACULTATE\AN%202\SEM%202\TP\Assignments\PT2021_30222_Varvara_Raluca_Assignment_3\Documentatie_Tema_3.docx#_Toc69723057)

[7. Bibliografie 18](file:///D:\FACULTATE\AN%202\SEM%202\TP\Assignments\PT2021_30222_Varvara_Raluca_Assignment_3\Documentatie_Tema_3.docx#_Toc69723058)

# **Obiectivul temei**

Obiectivul principal al proiectului este de a creea o aplicație care să pună la dispoziție utilizatorului un șistem de management al unui restaurant. Aceste sisteme se folosesc pretutindeni, fiecare site se foloseste de un asemenea sistem pentru vanzari online. Clientul se poate loga, poate vizualiza produsele si poate sa isi adauge in cosul de cumparaturi oricate produse. dupa ce s-a incheiat comanda, clientul primeste o facture pe care trebuie sa o plateasca. Toate aceste date sunt serializate, atat meniul cat si comenzile. Administratorul poate sa adauge produse, sa stearga sau modifice, sa creeze produse compuse, si sa creeze reporturi, sim ai exista un employee care este notificat atunci cand se creaza o comanda pentru a o prepara.

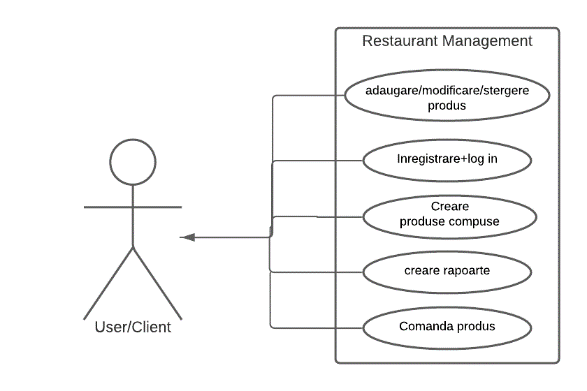
Sub-obiectivele principale ale temei reprezintă pașii care au fost urmați pentru atingerea obiectivului final:

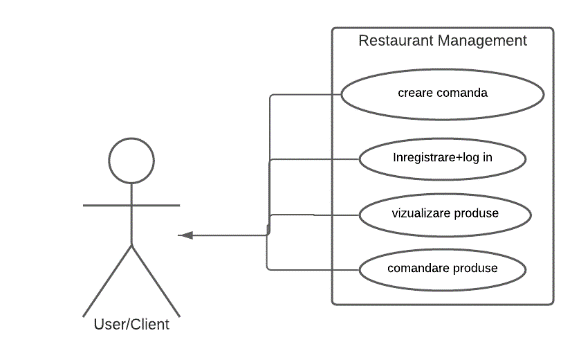
* Proiectarea unui user;
* Proiectarea unei menu item (normal sau compus);
* proiectarea unei comenzi;
* serializarea datelor;
* folosirea streamurilor si a lamda expressions pt a manipula datele;
* crearea unui patter observer-observable;
* Implementarea interfeței cu utilizatorul.

# **Analiza**

**Cerințele funcționale** vor include următoarele:

* introducerea, editarea si stergerea unui produs;
* crearea unui produs compus;
* importarea unor produse din fisierul csv;
* afisarea listei de produse;
* cautarea unui produs;
* introducerea de comenzi;
* facturarea comenzii.
* crearea rapoartelor;
* observarea creari comenzilor.





**Cerințele non-funcționale** vor fi următoarele:

* Aplicatia ar trebui să fie intuitiva și ușor de folosit de către user;
* Incapsularea datelor;

**Use case:** crearea unei comenzi

**Actorul principal:** user-ul

**Scenariul principal de success:**

1. User-ul se logheaza in sistem
2. daca nu exista se inregistreaza, apoi se logheaza
3. User-ul adauga la comanda cate produse vrea, putand consulta meniul
4. User-ul termina comanda si primeste bonul/factura;

**Secvența alternativă:** date introduce incorect

Userul introduce în unul dintre câmpurile de date ceva ce nu este un număr, sau lasă gol un câmp. Acesta va fi atenționat cu mesajul necesar.

**Secventa alternativa:**nu exista userul, prima oara se inregistreaza apoi se logheaza

**Secventa alternativa:** nu exista clientul sau produsul introduce: se va primi un mesaj de atentionare.

# **Proiectarea sistemului**



## Decizii de proiectare

Din punct de vedere al proiectării am luat următoarele decizii:

* Crearea unei clase abstracte AbstractTable<T> care creaza headerul si continutul unui table in functie de o lista de obiecte cu tehnica reflection;
* Crearea clasei User pentru a putea sa serializez datele pentru fiecare tip de user;
* Serializarea datelor pentru useri pentru a putea retine administratorii, clientii si angajatii.

## Divizarea in subsisteme/pachete

Am decis sa folosesc o arhitectura de tip Layered care imparte sistemul in 3 layers. Layerul pentru dataModels unde se retin clasele care definesc obiectele concrete cu care se lucreaza, structurile de date. Layerul de prezentare unde sunt clasele ce implementeaza interfata grafica. Business Logic layer ce continue clasele care se ocupa de logica aplicatiei.

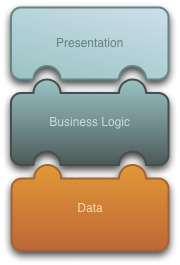
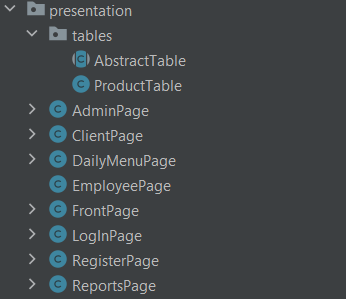
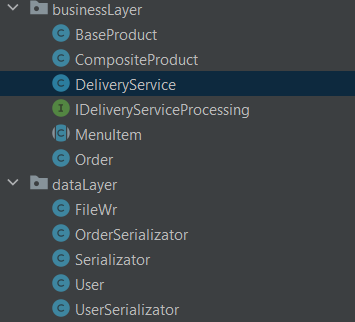
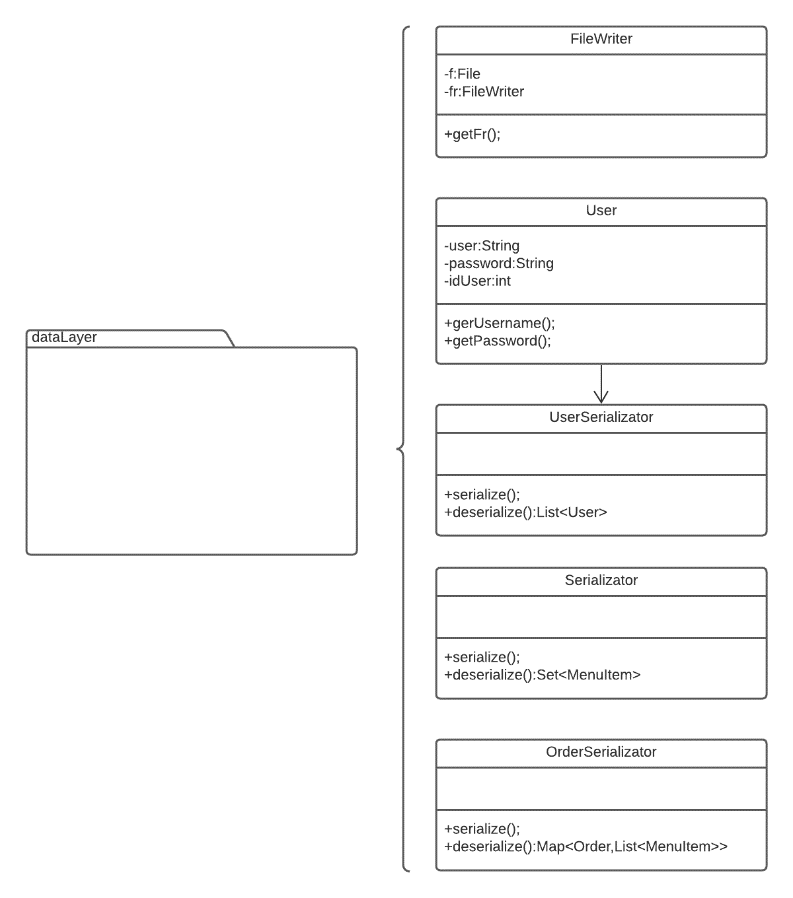


Fig. 2

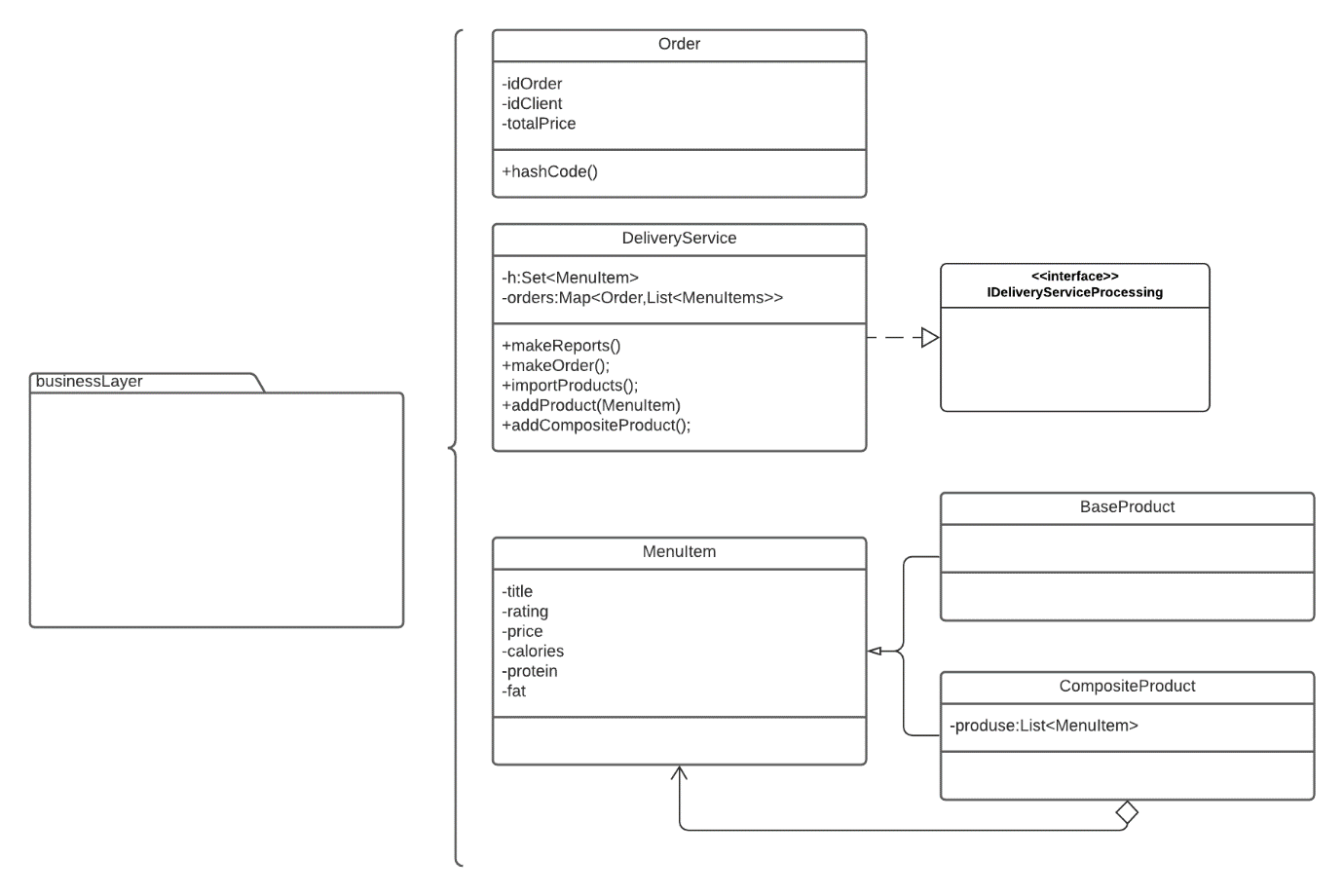
Este un mod ușor și intuitiv de implementare și împărtire în pachete, și nu există confuzii în legătură cu interacționarea dintre pachete.

## Împărțirea în clase

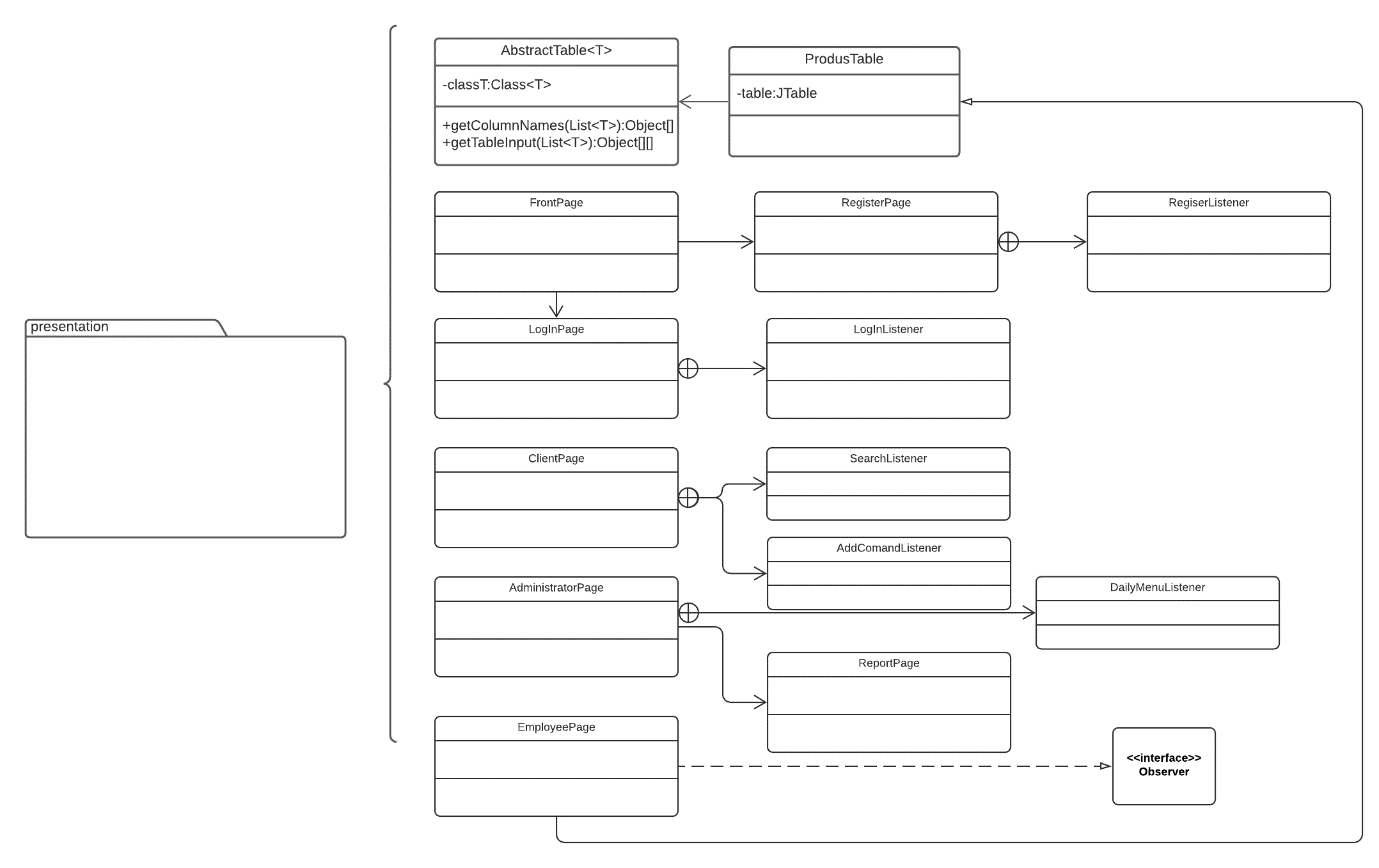




**Pachetul dataModel** va reține clasele care definesc modelele de date și care le vor putea stoca persistent astfel avem urmatoarele structure: FileWriter, User, Serializator, UserSerializator, OrderSerializator. Useri vor retine datele importante despre ei pentru procesare.



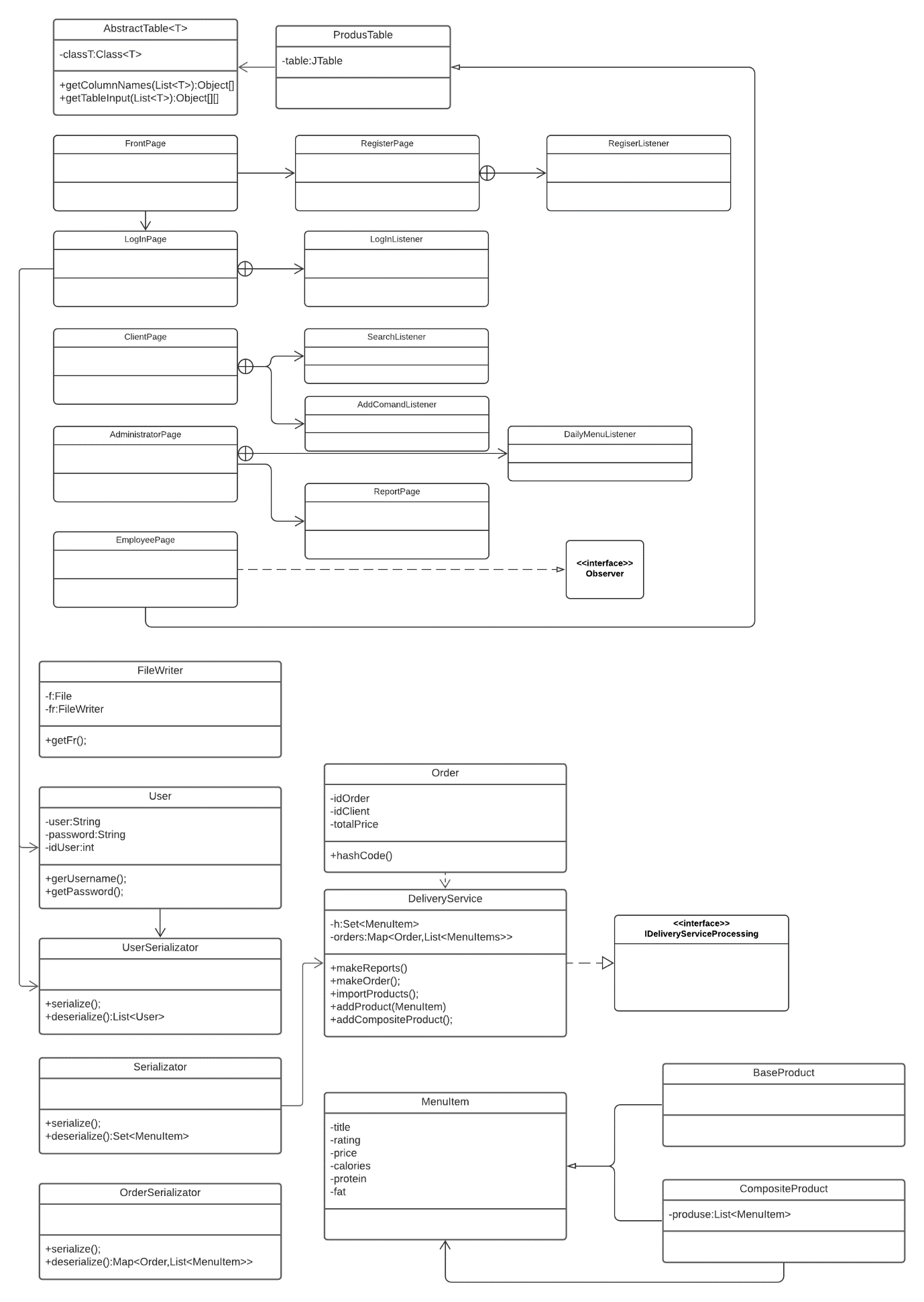
**Pachetul businessLogic** retine toata logica din spatele operatiei, creaza toata operatiile cu ajutorul claselor de access la date(serialzatoarelor) si valideaza imputurile. Acest pachet este responsabil de toate operatiile pe date.



**Pachetul presentation** detine 8 clase pentru fiecare fereastra si inca 2 pentru JTables, una dintre ele fiind abstracta pentru crearea tabelelor in mod abstract cu reflection. Este pachetul ce se ocupa cu interactionarea cu utilizatorul, abstractizeaza datele si lasa utilizatorul sa utilizeze metodele in mod corect.

## Diagrama de clase

Unified Modeling Language (prescurtat UML) este un limbaj standard pentru descrierea de modele și specificații software. Diagrama de clase UML este folosită pentru reprezentarea vizuală a claselor și a interdependențelor, taxonomiei și a relațiilor de multiplicitate dintre ele. Diagramele de clasă sunt folosite și pentru reprezentarea concretă a unor instanțe de clasă, așadar obiecte și a legăturilor concrete dintre acestea.



## Structuri de date

Structurile de date predefinite pe care le-am folosit sunt: ArrayList, List, HashMap, HashSet, Map, Set. Toate acestea sunt folosite pentru retinerea datelor. Set este folosit pentru retinerea meniului deoarece nu este corect sa existe 2 itemi exact la fel, iar set ne ofera aceasta posibilitate de a retine obiecte unice, daca se intalneste inca un ob la fel, nu se va introduce.

HashMap e implementarea bazată pe tabelul Hash a interfeței Map. Această implementare oferă toate operațiunile de hartă opționale și permite valori nule și cheia nulă. (Clasa HashMap este aproximativ echivalentă cu Hashtable, cu excepția faptului că este nesincronizată și permite valori nule.) Această clasă nu oferă garanții cu privire la ordinea hărții; în special, nu garantează că comanda va rămâne constantă în timp.

De asemenea, clasele din Model pot fi văzte ca niste structure de date ce definesc atributele importante pentru fiecare obiect in parte, deoarece clasele din model definesc clasele concrete, palpabile, care exista si in realitate.

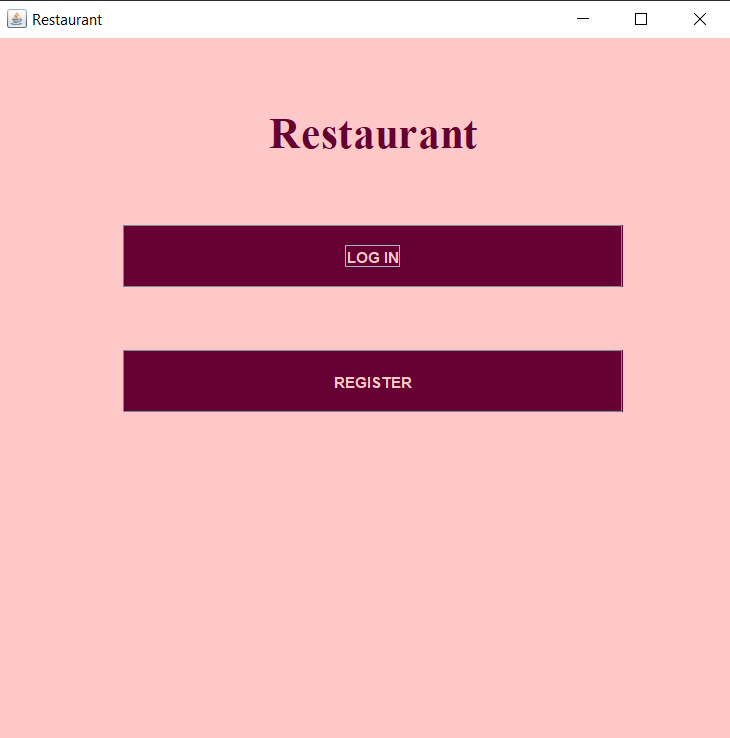
## Interfețe folosite/definite

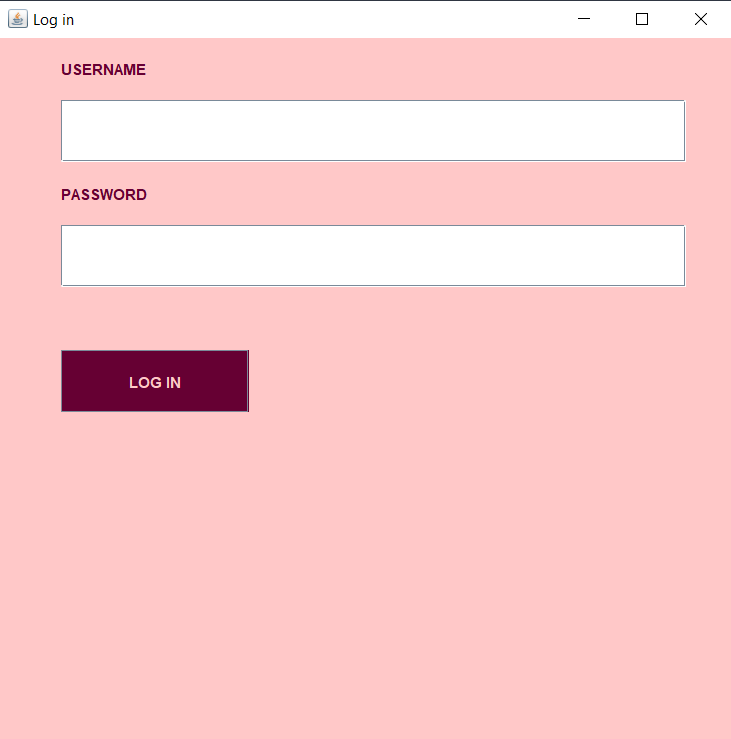
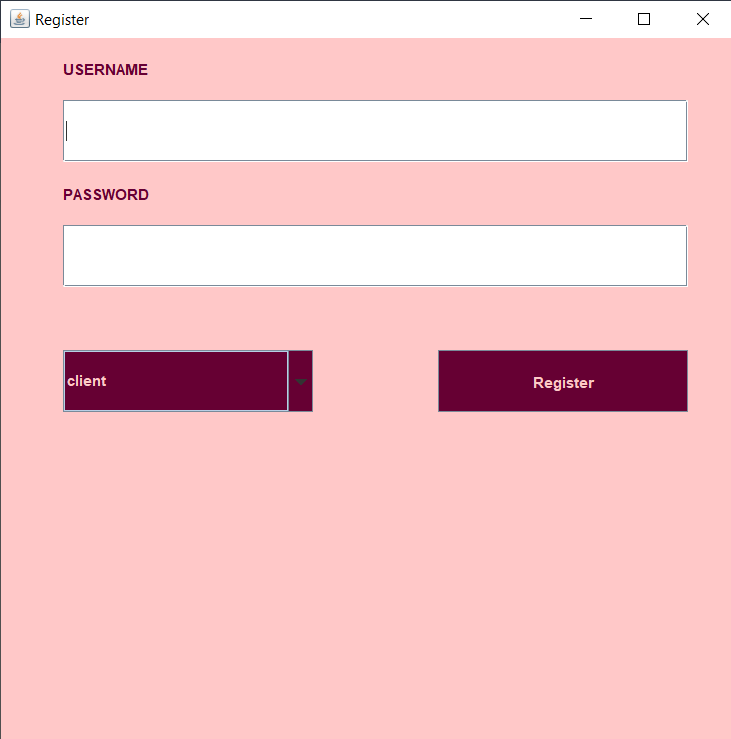
Observatorul este un model comportamental de proiectare. Specifică comunicarea între obiecte: observabil și observator. Un observabil este un obiect care notifică observatorii despre schimbările în starea sa.

Mi-am definit de asemenea propria interfata, IDeliveryService care este un contract pentru functionalitatile sistemului, si este implementata de clasa DeliveryService.

## 3.7 Interfața

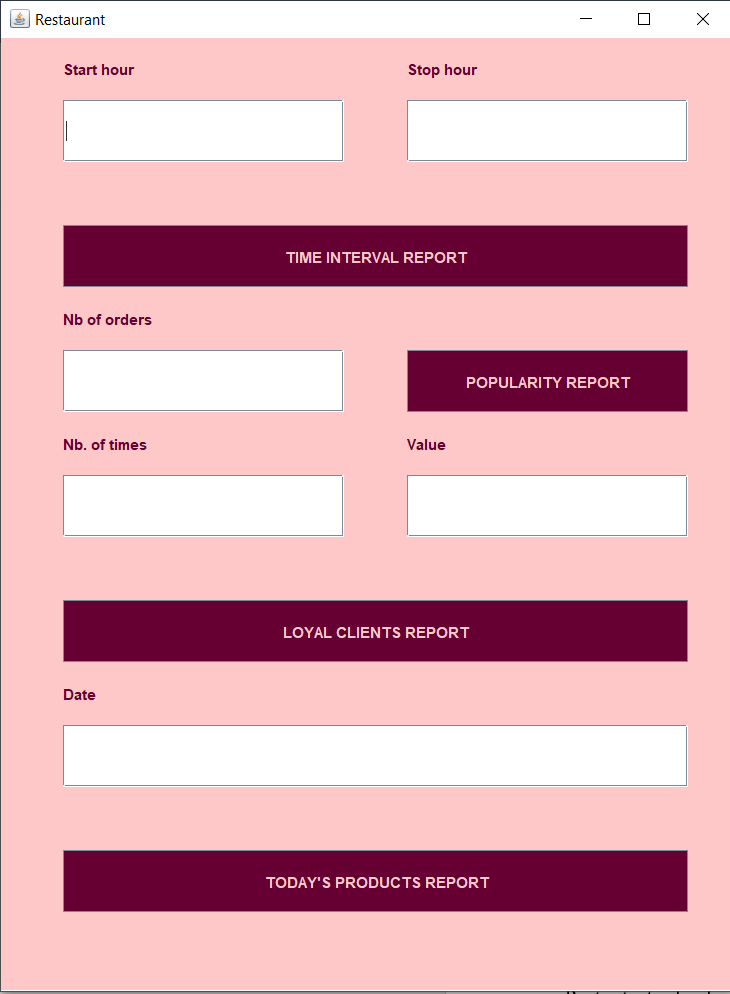
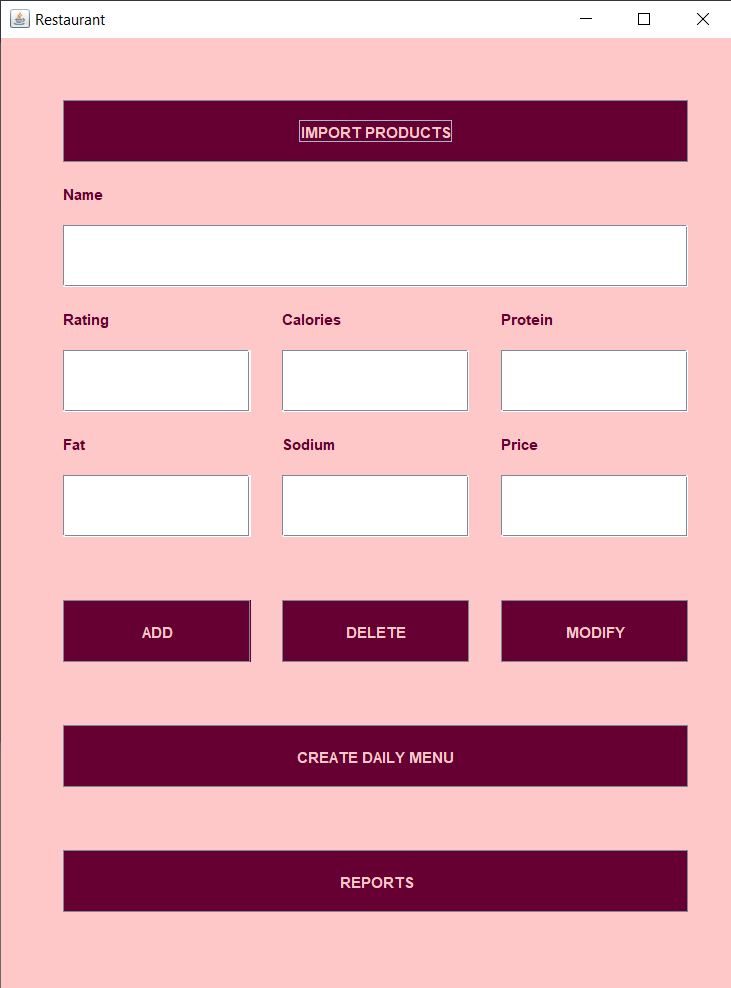
Am ales să implementez o interfata simpla și intuitivă, ușor de folosit de către user. Prima pagina are butoane ce te redirectioneaza catre cele 2 pagini principale, pagina care se ocupa cu logare, pagina ce se ocupa cu inregistraree, iar pagina de logare te redirectioneaza catre una dintre paginile celor 3 useri, in functie de username si parola.



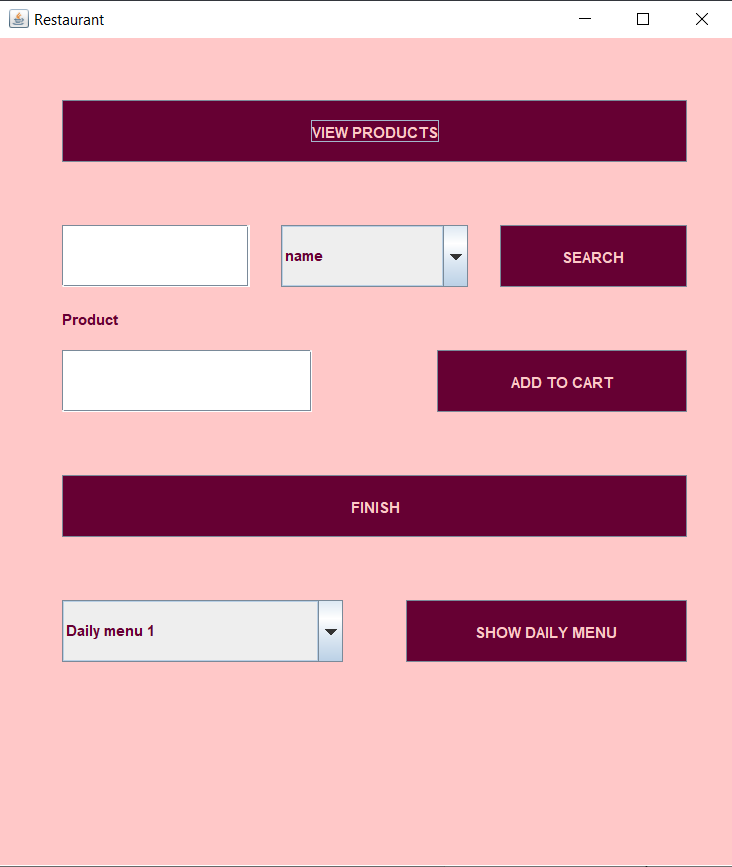


Register te inregistrea si iti serializeaza datele in fisierul ales de combobox, exista un fisier pentru fiecare tip de utilizator.

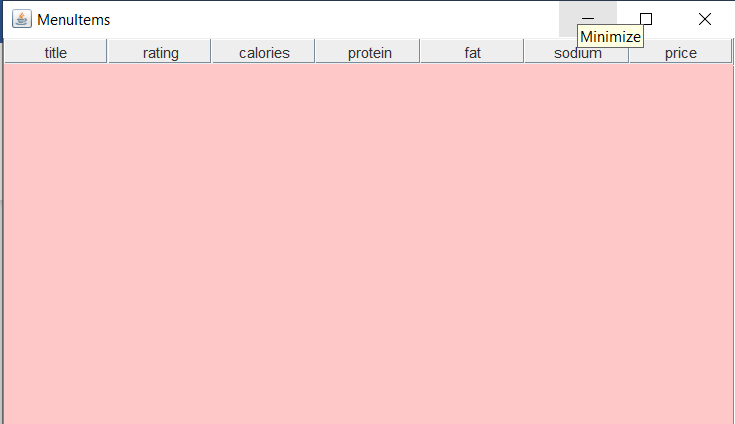
Pagina de log in te trimite catre una dintre paginile Client, Admin sau Employee, in functie de in care fisier se regaseste persoana.



Aici sunt toate functionalitatile pe care le fpoate face administratorul, butonul reports te va redirectiona catre pagina care face reportsurile.



Aceasta este pagina cu toate functionalitatile pentru Client. Poate sa caute un obiect in functie de atributele lui, nume, protein calorii, pret, rating si asa mai departe.



Aceasta este interfata pentru Angajat care implementeaza Obervator si este notificat de fiacre data cand se creeaza o comanda si I se afiseaza produsele comandate.

Am ales să implementez interfața cu Java Swing. Swing este un set de instrumente widget GUI pentru Java. Face parte din Oracle Java Foundation Classes (JFC) - un API pentru furnizarea unei interfețe grafice de utilizator (GUI) pentru programele Java. De asemenea pentru a nu încarca threadul principal cu printarea în text fielduri am folosit SwingWorker care permite crearea unui thread worker pentru interfața care să o updateze la intervalul necesar.

# Implementare

**Pachetul dataModel**

**Pachetul dataModel** va reține clasele care definesc modelele de date și care le vor putea stoca persistent astfel avem urmatoarele structure: FileWriter, User, Serializator, UserSerializator, OrderSerializator. Useri vor retine datele importante despre ei pentru procesare.

**Clasa User**

Clasa User este folosita pentru a retine datele importante username, password si id , pentru a putea fi serializati si bagati in fisiere in functie de cip de useri sunt, client, admini sau muncitori.

**Atribute:**

* **private int id; -** folosit pentru a identifica mai bine clienții si pentru a folosi in comenzi
* **private String username;**
* **private String password;**

Toate atributele sunt **private**, folosind conceptul specific progrămarii orientate pe obiect de incapsulare a datelor.

**Metode:**

* **public int getId()**
* **public int getName()** – getter pentru nume si id
* **public int setName(String s) --**settere pentru nume si id
* **public void setId(int id)**

**Clasa FileWriter**

Este clasa care se ocupa de scrierea in fisiere si este folosita pentru bonuri, si pentru toate rapoartele

**Atribute:**

* **private File f;**
* **private FileWriter fr;**

**Metode:**

* **public int getF();**
* **public getFr();**
* **FileWriter(String filename);**

**Clasa UserSerializator**

Este clasa care executa serializarea userilor si in functie de ce fel de useri sunt ii serializeaza intr-un anumit fisier: client, admin si employee.

**Atribute:**

* **private final String FILENAME;**
* **private final static long serialVersionUID = 911136765023976231l;**

**Metode:**

* **public void serialize(List<User>)**
* **public List<User> deserialize();**
* **public UserSerializator(String filename)**

**Clasa Serializator**

Este clasa care executa serializarea pentru meniu, serializeaza hashsetul de MenuItems din deliveryprocessing si deserializeaza de fiecare data cand se intra in aplicatie sau cand se actualizeaza ceva se serializeaza pentru a nu pierde ici o data.

**Atribute:**

* **private final String FILENAME;**
* **private final static long serialVersionUID = 911136765023976231l;**

**Metode:**

* **public void serialize(HashSet<MenuItem>)**
* **public HashSet<MenuItem>deserialize();**
* **public UserSerializator(String filename)**

**Clasa OrderSerializator**

Este clasa care executa serializarea pentru comenzi, serializeaza mapul de Order si List de MenuItems din deliveryprocessing si deserializeaza de fiecare data cand se intra in aplicatie sau cand se actualizeaza ceva se serializeaza pentru a nu pierde ici o data.

**Atribute:**

* **private final String FILENAME;**
* **private final static long serialVersionUID = 911136765023976231l;**

**Metode:**

* **public void serialize(HashMap<Order,List<MenuItem>>)**
* **public HashMap< Order,List<MenuItem >> deserialize();**
* **public UserSerializator(String filename)**

**Pachetul businessLayer**

**Pachetul businessLogic** retine toata logica din spatele operatiei, creaza toata operatiile cu ajutorul claselor de access la date(serialzatoarelor) si valideaza imputurile. Acest pachet este responsabil de toate operatiile pe date.

**Clasa MenuItem**

Este o clasa abstracta care implementeaza serializable. se va folosi composite product pentru a se implementa BaseProduct si CompositeProduct.

Se va folosi **modelul compozit**, care este un model de proiectare de partiționare. Modelul compozit descrie un grup de obiecte care sunt tratate la fel ca o singură instanță a aceluiași tip de obiect. Intenția unui compozit este de a „compune” obiecte în structuri de copac pentru a reprezenta ierarhii parțiale. Implementarea modelului compozit permite clienților să trateze obiecte individuale și compoziții în mod uniform.

**Atribute:**

* **private String title;**
* **private double rating;**
* **private double calories;**
* **private double protein;**
* **private double fat;**
* **private double sodium;**
* **private double price;**

**Metode:**

* **getTitle();**
* **getRatings();**
* **getCalories()**
* **getProtein();**
* **getFat();**

**Clasa BaseProduct**

Este clasa care extinde MenuItem si va fi un produs de baza, acesta va putea folosi pentru a crea produse compuse.

**Clasa CompositeProduct**

Este clasa care extinde MenuItem dar continue si o liste de MenuItems din care se compune.

**Atribute:**

* **private List<MenuItem> products = new ArrayList<MenuItem>();**

**Metode:**

* **computePrice();**

**Clasa Order**

Retine atributele specific unui Order

**Atribute:**

* **private int clientId;**
* **private int orderId;**
* **private LocalDateTime orderDate;**
* **private double totalPrice;**

**Interfata IDeliveryServiceProcessing**

Este un contract pentru functionalitatile sistemului, si este implementata de clasa DeliveryService.

**Metode:**

* **void importProducts();**
* **void modifyProduct(MenuItem newM);**
* **void addProduct(MenuItem newM);**
* **void deleteProduct(String title);**
* **void timeIntervalReport(int hStart, int hStop);**
* **void popularProductsReport(int nbOfTimes);**
* **void loyalClientsReport(int nbOfTimes, int value);**
* **void todaysProducts(LocalDate date);**
* **List<MenuItem> searchByName(String name);**
* **List<MenuItem> searchByRating(double rating);**
* **List<MenuItem> searchByCalories(double calories);**
* **List<MenuItem> searchByProtein(double protein);**
* **List<MenuItem> searchByFat(double fat);**
* **List<MenuItem> searchBySodium(double sodium);**
* **List<MenuItem> searchByPrice(double price);**

**Clasa DeliveryService**

**Atribute:**

* **private HashSet<MenuItem> h = new HashSet<MenuItem>();**
* **private Map<Order, List<MenuItem>> orders = new HashMap<Order, List<MenuItem>>();**
* **private Serializator serializator = new Serializator();**

**Metode:**

* **void importProducts();** - se importa toate produsele din csv prin citire cu streamuri si se filtreaza sa nu fie acelasi nume cu ajutorul seturilor.
* **void modifyProduct(MenuItem newM);** - se sterge produsul si se adauga ala nou
* **void addProduct(MenuItem newM); -** se deserializeaza lista de menuitems si se adauga la lista apoi se serializeaza iar
* **void deleteProduct(String title);**
* **void timeIntervalReport(int hStart, int hStop); -** de asemenea toate functiile de creare de rapoarte se fac tot cu streamuri
* **void popularProductsReport(int nbOfTimes);**
* **void loyalClientsReport(int nbOfTimes, int value);**
* **void todaysProducts(LocalDate date);**
* **List<MenuItem> searchByName(String name);** - toate metodele de cautare se face cu ajutorul streamurilor si functiilor de filtrare si map.
* **List<MenuItem> searchByRating(double rating);**
* **List<MenuItem> searchByCalories(double calories);**
* **List<MenuItem> searchByProtein(double protein);**
* **List<MenuItem> searchByFat(double fat);**
* **List<MenuItem> searchBySodium(double sodium);**
* **List<MenuItem> searchByPrice(double price);**

**Pachetul presentation**

**Pachetul presentation** detine 8 clase pentru fiecare fereastra si inca 2 pentru JTables, una dintre ele fiind abstracta pentru crearea tabelelor in mod abstract cu reflection. Este pachetul ce se ocupa cu interactionarea cu utilizatorul, abstractizeaza datele si lasa utilizatorul sa utilizeze metodele in mod corect.

**Clasa AbstractTable<T>**

Este clasa care creeaza cu ajutorul tehnicii reflection headerul si interiorul unui table in functie de o Lista de obiecte.

**Metode:**

* **public Object[] getColumnsNames(List<T> l)** – este metoda ce creaza numele fieldurilor din table
* **public Object[][] getTableInput(List<T> l)** – este metoda ce creaza interiorul tabelului.

**Clasa ProductTable**

**Atribute:**

* private final JFrame frame = new JFrame("MenuItems");
* private final JPanel content = new JPanel();
* private final JTable table;

**Clasa AdminPage**

**Atribute:**

* **private final JFrame frame = new JFrame("Restaurant");**
* **private final JPanel content = new JPanel();**
* **private final JButton importProd = new JButton("IMPORT PRODUCTS");**
* **private final JButton add = new JButton("ADD");**
* **private final JButton delete = new JButton("DELETE");**
* **private final JButton modify = new JButton("MODIFY");**
* **private final JButton dailyMenu = new JButton("CREATE DAILY MENU");**
* **private final JButton reports = new JButton("REPORTS");**
* **private final JLabel lName = new JLabel("Name");**
* **private final JLabel lRating = new JLabel("Rating");**
* **private final JLabel lCalories = new JLabel("Calories");**
* **private final JLabel lProtein = new JLabel("Protein");**
* **private final JLabel lFat = new JLabel("Fat");**
* **private final JLabel lSodium = new JLabel("Sodium");**
* **private final JLabel lPrice = new JLabel("Price");**
* **private final JTextField tName = new JTextField();**
* **private final JTextField tRating = new JTextField();**
* **private final JTextField tCalories = new JTextField();**
* **private final JTextField tProtein = new JTextField();**
* **private final JTextField tFat = new JTextField();**
* **private final JTextField tSodium = new JTextField();**
* **private final JTextField tPrice = new JTextField();**

# Rezultate

Asa arata un bon:

Order{clientId=0, orderId=1, orderDate=2021-05-16T13:07:11.691349400}

[MenuItem title='Daily menu 1', rating=3.75, calories=633.0, protein=22.0, fat=36.0, sodium=0.0, price=50.0

]

total50.0

Un raport pt produsele comandate in ziua 16 05 2021

products ordered on 2021-05-16 are: [MenuItem title='nume', rating=1.0, calories=1.0, protein=1.0, fat=1.0, sodium=1.0, price=1.0

, MenuItem title='Daily menu 1', rating=3.75, calories=633.0, protein=22.0, fat=36.0, sodium=0.0, price=50.0

]

un raport pentru time interval 9 15 la urmatoarele comenzi existente in sistem{Order{clientId=1, orderId=2, orderDate=2021-05-16T13:04:55.128096900}

=[MenuItem title='nume', rating=1.0, calories=1.0, protein=1.0, fat=1.0, sodium=1.0, price=1.0

], Order{clientId=0, orderId=2, orderDate=2021-05-20T13:08:18.914332200}

=[MenuItem title='Berry-Yogurt Smoothie ', rating=5.0, calories=326.0, protein=12.0, fat=5.0, sodium=135.0, price=17.0

, MenuItem title='Zucchini Relish Wiley ', rating=4.375, calories=210.0, protein=3.0, fat=1.0, sodium=1767.0, price=85.0

], Order{clientId=0, orderId=3, orderDate=2021-05-20T13:09:26.715881100}

=[MenuItem title='Cherry-Chocolate Shortcakes with Kirsch Whipped Cream ', rating=4.375, calories=509.0, protein=7.0, fat=23.0, sodium=312.0, price=89.0

], Order{clientId=0, orderId=1, orderDate=2021-05-16T13:07:11.691349400}

=[MenuItem title='Daily menu 1', rating=3.75, calories=633.0, protein=22.0, fat=36.0, sodium=0.0, price=50.0

]}

RAPORT:

[Order{clientId=1, orderId=2, orderDate=2021-05-16T13:04:55.128096900}

, Order{clientId=0, orderId=2, orderDate=2021-05-20T13:08:18.914332200}

, Order{clientId=0, orderId=3, orderDate=2021-05-20T13:09:26.715881100}

, Order{clientId=0, orderId=1, orderDate=2021-05-16T13:07:11.691349400}

]

# Concluzii

În concluzie realizarea acestui proiect a fost foarte utilă , amintind de o mare parte din principalele concepte ale programării orientate pe obiecte învățate semestrul trecut.

Am învățat să lucrez cu steamuri, sa folosesc Java Serializable, Lamda expressions si interfetele Observer si Observable.

O dezvoltare ulterioară ar fi adaugarea unui email clientului plus un validator pentru email, sa se poata updata produsele prin incrementarea cantitatii, si sa se creeze un pop-up cu factura. De asemenea se mai poate adauga un cont bancar pentru clienti si sa poata face comanda numai daca au destui bani in cont.

# Bibliografie

* <https://lucid.app/documents>
* <https://app.diagrams.net/>
* <https://docs.oracle.com/javase/tutorial/uiswing/components/table.html>
* https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html
* ASSIGNMENT\_4\_SUPPORT\_PRESENTATION