

**FOOD DELIVERY MANAGEMENT SYSTEM**

**DOCUMENTAȚIE**

**Facultatea:** Automatică și Calculatoare

**Specializarea:** Calculatoare și Tehnologia Informației

**Materia:** Tehnici de programare fundamentale

**Nume student:** Petricele Mihaela **Profesor coordonator**

**Grupa:** 30228 Dr. Pop Cristina Bianca

**Cuprins**

[**I.** **Obiectivul temei** 3](#_Toc75989243)

[**II.** **Analiza problemei, modelare, scenarii, cazuri de utilizare** 3](#_Toc75989244)

[***A.*** ***Diagrama Use – case a aplicației*** 4](#_Toc75989245)

[***B.*** ***Descrierea cazurilor de utilizare*** 4](#_Toc75989246)

[**III.** **Proiectare** 6](#_Toc75989247)

[*A)* *Proiectarea OOP a aplicației* 6](#_Toc75989248)

[*B)* *Diagramele UML ale aplicației* 6](#_Toc75989249)

[*a.* *Diagrama de pachete* 6](#_Toc75989250)

[*b.* *Diagrama de clase* 7](#_Toc75989251)

[*C)* *Structurile de date folosite* 8](#_Toc75989252)

[*D)* *Interfețele definite* 8](#_Toc75989253)

[**IV.** **Implementare** 8](#_Toc75989254)

[**A)** **Pachetul Business** 8](#_Toc75989255)

[**B)** **Pachetul Data** 10](#_Toc75989256)

[**C)** **Pachetul Presentation** 11](#_Toc75989257)

[**V.** **Rezultate** 13](#_Toc75989258)

[**VI.** **Concluzii** 14](#_Toc75989259)

[**VII. Bibliografie** 15](#_Toc75989260)

# **Obiectivul temei**

*Obiectivul principal* al temei este proiectarea și implementarea unui sistem de management al livrării alimentelor pentru o companie de catering. Clientul poate comanda produse din meniul companiei, produse care sunt introduse de către administratorul companiei, iar angajatul poate se vadă produsele comandate de către client. Sistemul ar trebui să aibă trei tipuri de utilizatori care se conectează utilizând un nume de utilizator și o parolă: administrator, angajat și client.

*Obiectivele secundare ale temei sunt:*

* **Analiza problemei** (în capitolul 2) => se prezintă cerințele funcționale, cazurile de utilizare sub formă de

diagrame use-case

* **Proiectare** (în capitolul 3) => proiectarea OOP aplicației, diagramele UML de clase și de pachete,

structurile de date folosite, interfețele definite și algoritmii folosiți.

* **Implementare** (în capitolul 4) => descrierea claselor, metodelor folosite și descrierea implementării

interfeței cu utilizatorul

* **Rezultate** ( în capitolul 5) => prezentarea rapoartelor generate de către administrator pentru verificarea

corectitudinii aplicației

# **Analiza problemei, modelare, scenarii, cazuri de utilizare**

Din cerința temei, in funcție de rolul utilizatorului care se loghează în aplicație, rezultă cerințele funcționale

ale temei. Prin urmare aplicația trebuie sa permită utilizatorului:

* Să se conecteze utilizând un username și o parolă
* Să se înregistreze ca și client nous au angajat nou introducând un username și o parolă nouă
* Administorul poate să realizeze următoarele operații:
  + Importarea setului inițial de produse care va completa meniul dintr-un fișier .csv
  + Gestionarea produselor din meniu: adăugare / ștergere / modificare produse și crearea de produse noi compuse din mai multe produse
  + Crearea de rapoarte despre comenzile efectuate luând în considerare următoarele criterii:
    - intervalul de timp al comenzilor - ar trebui generat un raport cu comenzile efectuate între o anumită oră de început și o anumită oră de sfârșit, indiferent de dată
    - produsele comandate de mai mult de un număr specific de ori până în acel moment.
    - clienții care au comandat mai mult de un număr specificat de ori și valoarea comenzii a fost mai mare decât o sumă specificată
    - produsele comandate într-o zi specificată și de câte ori au fost comandate
* Clientul poate să realizeze următoarele operații:
* Vizualizarea listei cu produsele din meniu
* Căutarea produselor pe baza unuia sau mai multor criterii, cum ar fi titlul, numarul de calorii, prețul etc.
* Crearea unei comenzi formată din mai multe produse - pentru fiecare comandă, data și ora vor fi memorate și va fi generată o factură care va lista produsele comandate și prețul total al comenzii
* Angajatul este notificat de fiecare dată când o nouă comandă este efectuată de către un client, astfel încât acesta să poată pregăti livrarea produselor comandate.

## ***Diagrama Use – case a aplicației***

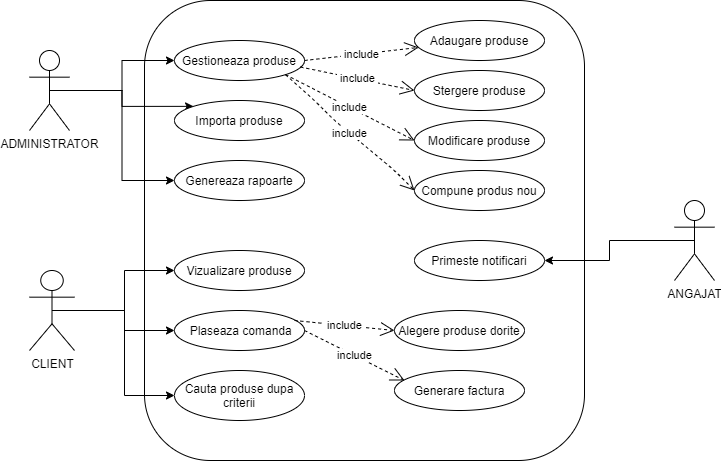


Figura 2.1

## ***Descrierea cazurilor de utilizare***

În funcție de rolul utilizatorului există mai multe cazuri de utlizare a aplicației:

***Use case:*** *Gestioneaza produse/ Importa produse/ Genereaza rapoarte*

***Actor:*** Administrator

***Scenariul de success:*** Administratorul poate să:

1. Gestioneze produsele

* Acesta reușește să adauge un produs nou, să șteargă un produs, să modifice un produs introducând corect datele necesare
* Acesta reușește să compună un produs nou din produsele existente, selectând corect produsele și alegând un nume pentru produsul compus

1. Importa produse

* A fost apăsat butonul corect *,,Importa’’* și importarea s-a realizat cu succes

1. Generare rapoarte

* S-au important cu success produsele și s-a realizat minim o comandă de către un client, iar datele pentru fiecare raport sunt introduse corect in *TextField,* caz în care rapoartele se realizează cu succes

***Scenariu alternativ:***

1. Actorul nu completează un anumit câmp la adăugare/ștergere/modificare, caz în care operațiile nu se realizează cu succes
2. La importarea produselor, fișierul csv. a suferit anumite schimbări, iar importul nu se mai poate realiza
3. În momentul generării rapoartelor, nu s-a realiza nici o comandă caz în care nu se pot genera rapoartele

***Use case:*** *Vizualizare produse/ Plasează comanda/ Caută produse după criterii*

***Actor:*** Client

***Scenariul de success:*** Clientul realizează operațiile:

1. Vizualizare produse

* Acesta reușește să vizualizeze lista de produse într-un tabel

1. Caută produse după criterii

* S-a introdus cel puțin un criteriu de căutare în casetele corespunzătoare (title,rating etc.) și la apăsarea butonului de *,,Search’’* se afișează produsele găsite în tabel

1. Plasează comanda

* Clientul selectează din tabelul cu produse cel puțin un produs pentru a fi comandat și apasă pe *,,Order’’*, caz în care primește un mesaj că comanda a fost realizată cu succes

***Scenariu alternativ:***

1. Actorul nu reușeste să vizualizeze lista de produse
2. Acesta nu introduce nici un criteriu de căutare, caz în care căutarea nu se poate realiza
3. Clientul nu selectează nici un produs pentru a-l comanda, caz în care primește un mesaj de atenționare

***Use case:*** *Primește notificări*

***Actor:*** Angajat

***Scenariul de success:*** Angajatul primește notificare cum că un client a realizat o comandă și pregătește produsele pentru livrare

***Scenariu alternativ:*** Angajatul nu primește nici o notificare cu produsele comandate, caz în care nu poate pregăti produsele pentru livrare

# **Proiectare**

## *Proiectarea OOP a aplicației*

Aplicația este organizată în pachete și în fiecare pachet se află clasele care implementează toată logica

aplicației. Aceasta conține pachetul *Business* cu clasele aferente pentru implementarea logicii de business a aplicației. În pachetul *Data* se află clasele care implementează serializarea produselor și clasa pentru generarea rapoartelor în format .txt. In final în pachetul *Presentation* se implementează interfața cu utilizatorul a aplicației.

## *Diagramele UML ale aplicației*

### *Diagrama de pachete*

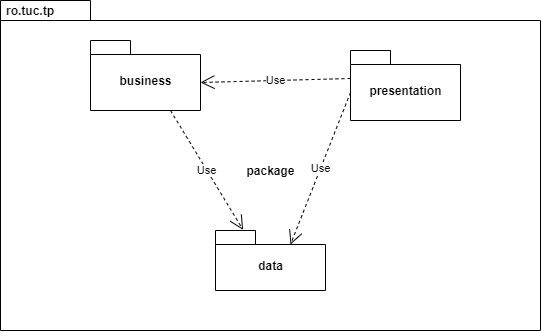


Figura 3.1

### *Diagrama de clase*

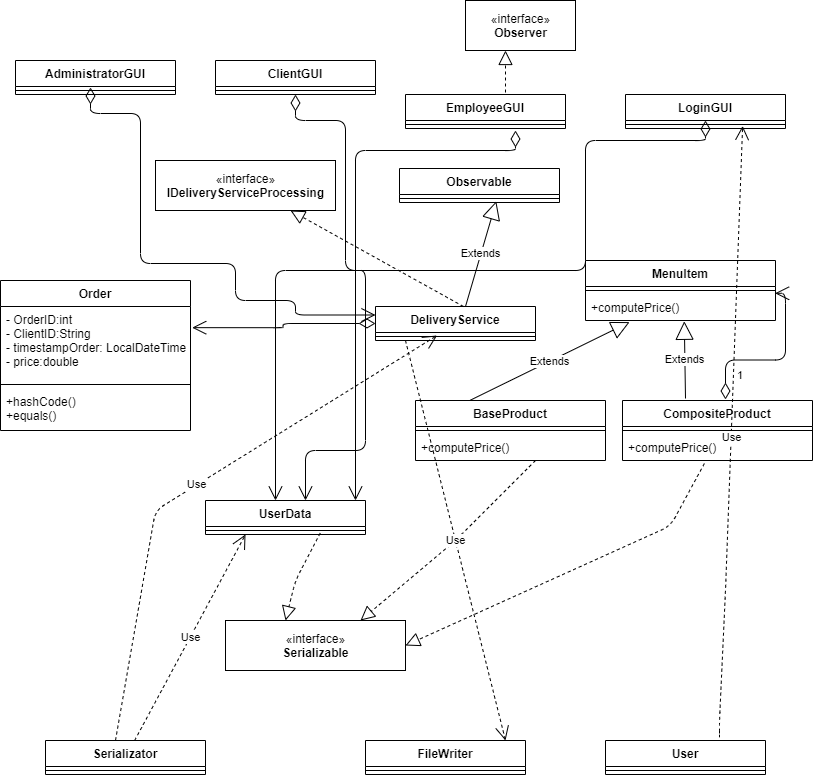


Figura 3.2

## *Structurile de date folosite*

Ca structură de date am folosit tipul Map<Order,List<MenuItem>> pentru a stoca datele despre comenzi și

produse din clasa DeliveryService. Cheia colecției este reprezentată de un obiect de tip *Order* care reprezintă comanda realizată de către un client, iar valorile colecției sunt o lista cu produsele comandate. In clasele *Order* și *MenuItem* sunt suprascrise metodele hashCode() și equals pentru funcția colecției în funcție de datele din clase.

## *Interfețele definite*

În cadrul acestei teme am lucrat cu următoarele interfețe pentru a mă ajuta sa implementez mai eficient

aplicația:

* Interfața *ActionListener* pe care am utlizat-o la clasele de GUI.
* Interfața *IDeliveryServiceProcessing* care implementează metodele care realizează operațiile pentru administrator, client și angajat.
* Interfața *Serializable* pe care am utilizat-o pentru operația de serializare a produselor și a datelor clienților. Această interfață este implementată de fiecare clasă care face parte din procesul de serializare.

# **Implementare**

În acest capitol se prezintă clasele și metode corespunzătoare care au ajutat la realizarea aplicației

Food Delivery Service.

## **Pachetul Business**

1. **Clasa MenuItem**

Această clasă este abastractă și conține următoarele câmpuri: title, rating,calories,protein, fat,

sodium și price care reprezintă datele cunoscute despre fiecare produs. Totodată această clasa descrie metodele de compute() care vor fi implementate pentru compunerea datelor produselor compuse.

1. **Clasa BaseProduct**

Această clasă extinde clasa MenuItem ca și clasa derivată din aceasta și prin intermediul

constructorului se apelează super de constructorul clasei MenuItem. In metodele de compute se returnează doar metodele de get() pentru accesara datelor despre fiecare produs.

1. **Clasa CompositeProduct**

Această clasă extinde MenuItem și implementează logica pentru realizarea unui produs

compus. Clasa conține o listă cu produsele de bază pentru compunerea produsului compus și totodată implementează metodele de compute() pentru obținerea field-urilor pentru produsul compus. Exemplu de metode compute() pentru compunerea prețului și rating-ului:

*public double computePrice() {*

*double totalPrice = 0;*

*for (MenuItem menuItem : productList) {*

*totalPrice += menuItem.getPrice();*

*}*

*return totalPrice;*

*}*

*public double computeRating() {*

*int dim = productList.size();*

*double sumRating = 0;*

*for (MenuItem menuItem : productList) {*

*sumRating += menuItem.getRating();*

*}*

*return sumRating / dim;*

*}*

Totodată această clasa implementează și metoda Serializable pentru operația de serializare a produselor.

1. **Clasa DeliveryService**

În această clasă se implementează toate operațiile pentru client, administrator și angajat.

Implementarea metodelor pentru generarea rapoartelor și pentru căutarea după un anumit criteriu se realizează cu stream-uri și expresii lambda. Pentru administrator se implementează operațiile de adăugare, ștergere și modificare produse. Totodată se implementează și metodele pentru generarea celor 4 rapoarte. De exemplu metoda pentru importul produselor din fișierul .csv este următoarea:

*public void importItem()*

*{*

*try {*

*assert isWellFormed();*

*Path path = Paths.get("products.csv");*

*itemCollection = new ArrayList<>();*

*List<BaseProduct> productList;*

*if (Files.exists(path)) {*

*Stream<String> readAllLines = Files.lines(path);*

*productList = readAllLines*

*. skip (1)*

*. map (line -> line.split(","))*

*. map (line -> new BaseProduct(line [0], Double.parseDouble(line[1]), Integer.parseInt(line[2]), Integer.parseInt(line[3]), Integer.parseInt(line[4]), Integer.parseInt(line[5]), Integer.parseInt(line[6])))*

*. distinct ()*

*. sorted (Comparator.comparing(BaseProduct::getTitle))*

*. collect (toList());*

*itemCollection.addAll(productList);*

*}*

*} catch (IOException e) {*

*System.out.println("Eroare la import!");*

*}*

*assert !itemCollection.isEmpty();*

*assert isWellFormed();*

*}*

Metodele de căutare a unui produs după un anumit criteriu sunt implementate tot în această clasă, iar un exemplu de metodă pentru căutarea unui produs după numărul de calorii este următoarea:

*public List<MenuItem> searchByCalories(int criteria) {*

*assert criteria > 0;*

*assert isWellFormed();*

*List<MenuItem> itemList;*

*itemList = itemCollection*

*.stream()*

*.filter(c -> c.getCalories() == criteria)*

*.collect(toList());*

*assert isWellFormed();*

*return itemList;*

*}*

Totodată această clasă conține și metoda pentru crearea unei comenzi pe baza produselor selectate

de către client. În această metodă se apelează și metodele *setChanged()* și *notifyObserver()* cu care se realizează notificarea angajatului cu referință că s-a plasat o comandă

## **Pachetul Data**

1. **Clasa FileWriter**

În această clasă se implementează metodele pentru scrierea în fișier în format .txt a datelor pentru

fiecare raport generat de către administrator. Clasa suprascrie clasa deja implementată cu acealași nume FileWriter. În același timp această clasă conține și metoda statică pentru generarea facturii în format .txt în momentul realizării unei comenzi.

1. **Clasa Serialization**

Această clasă implementează logica pentru serializarea aplicației. Se serializează produsele și

datele clienților în momentul închiderii aplicației. Pentru serializare și deserializare am folosit JSon.

1. **Clasa User**

Această clasa reține datele de login a clientului care s-a înregistrat s-au conectat în momentul

respectiv. Aceasta este folosită în momentul în care se realizează comanda, deoarece avem nevoie să știm numele clientului care a realizat comanda.

## **Pachetul Presentation**

În acest pachet se implementează interfața cu utilizatorul a aplicației. Aceasta este formată din 4 ferestre:

1. Fereastra de Login

Această fereastră este formată din:

* Un JTextField unde se introduce username-ul utilizatorului
* Un JTextField unde se introduce parola utilizatorului
* Trei butoane pentru *Login, Reset* și *Register*
* Un FaceCombo pentru selectarea rolului în momentul conectării (client sau employee, administatorul având date statice care nu se pot schimba)
* Dacă conectarea s-a realizat cu succes se vor afișa anumite JLabel care vor arăta de exemplu: “Login successful as Admin!” sau dacă conectarea nu s-a realizat cu succes se va afișa un JLabel cu mesaj de eroare, de exemplu: “Login data don't match!”



Figura 4.1

1. Fereastra pentru Administrator

Această fereastră este formată din 3 zone:

* Zona în care se introduc produse noi prin precizarea datelor conținute de fiecare produs. În această zonă mai exista șase butoane pentru *Import, Add, Modify, Delete, Compute* și *List* pentru listarea produselor după modificare listei de produse
* Zona pentru vizualizarea produselor existente în acel moment
* Zona pentru generarea rapoartelor corespunzătoare. Fiecare raport are zone de JTextField corespunzătoare pentru datele despre care trebuie să fie generat raportul și un buton asociat. Butonul *RaportA* corespunde pentru primul raport, butonul *RaportB* pentru al doilea raport etc.

O imagine care conține masă

Descriere generată automat

Figura 4.2

1. Fereastra pentru Client

Această fereastra conține două zone:

* Zona în care se introduc criteriile pentru căutarea produselor și butonul de *Search* care în momentulapăsării va afișa produsele cu criteriile introduse în tabelul din zona de afișare
* Zona de afișare a produselor de unde clientul poate să își aleagă ce produse să comande prin selectarea acestora din tabel și apăsarea butonului de *Order*

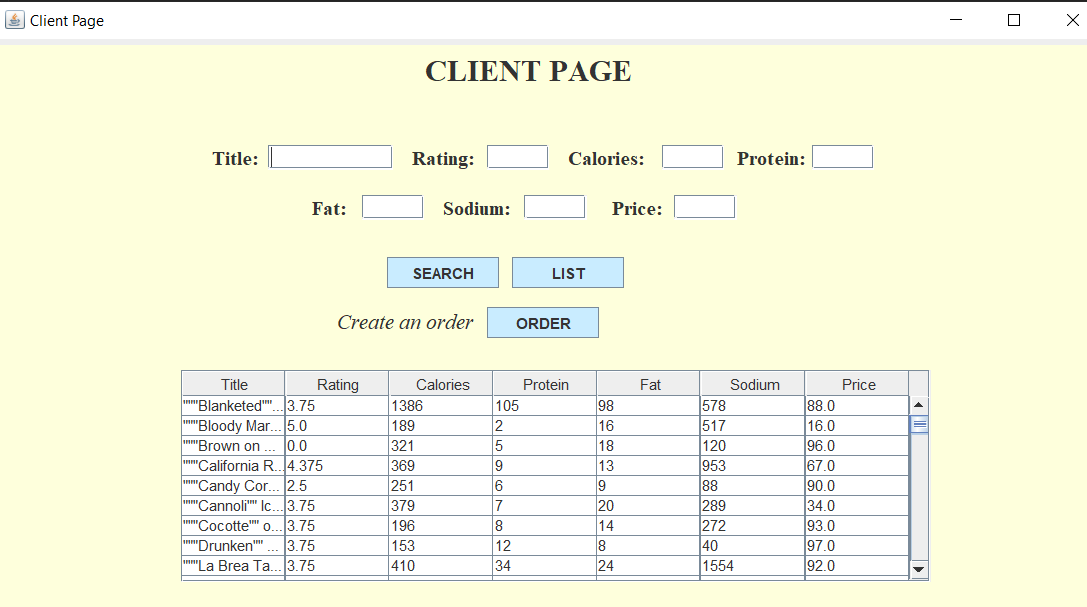


Figura 4.3

1. Fereastra pentru Angajat

În această fereastra se afișeaza produsele comandate de către un anumit client în momentul efectuării

comenzii.

# **Rezultate**

Pentru testarea aplicației am introdus date de test. Pentru pagina administratorului am introdus un produs

nou: *,,test, 20, 4, 5, 6, 10, 15.6”.* Am modificat un produs, am șters un produs și am realizat un produs compus cu numele de *,,testMenu”.* Am generat toate rapoartele introducând anumite date de test, iar acestea s-au generat corect. Pentru pagina clientului am introdus anumite date și am realizat căutarea dupa ratingul 3.75 și pretul 88.0. Am create o comandă selectând anumite produse din tabel. Comanda s-a transmis în timp real angajatului și s-a generat și factura corespunzătoare cu produsele comandate.

# **Concluzii**

În concluzie acest proiect mi-a aprofundat cunoștințele despre limbajul Java. Am învățat să lucrez cu

serializarea fișierelor, fiind prima dată când fac acest lucru. Serializarea cu JSON mi s-a părut ușor de înțeles și practică. Totodată am învățat să lucrez cu tag-urile *@pre, @post* și *@invariant* și cu assert-urile corespunzătoare. Am mai învățat să realizez logarea și înregistrarea unui user. În același timp am învățat să lucrez interfața Obervable și clasa Observer pentru notificarea angajatului că s-a realizat o comandă.

Ca dezvoltări ulterioare aș propune realizarea unei metode mai bune pentru update-ul unui produs. Totodată realizarea unei interfețe mai ,,prietenoase’’ și mai ușor de înțeles.

# **VII. Bibliografie**

1. <https://www.youtube.com/watch?v=uJUXyhya3YM>
2. <https://www.youtube.com/watch?v=Hiv3gwJC5kw>
3. <https://www.baeldung.com/java-stream-filter-count>
4. <https://www.baeldung.com/java-stream-filter-lambda>
5. <https://www.youtube.com/watch?v=vcV0DK45P-Q>
6. <https://www.youtube.com/watch?v=Zt4g6HiFNxo>
7. <https://stackoverflow.com/questions/20526917/load-arraylist-data-into-jtable>
8. <http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html>
9. <https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>
10. <https://github.com/stleary/JSON-java>
11. <https://stackoverflow.com/questions/2591098/how-to-parse-json-in-java>
12. Assignment\_4\_support\_presentation