Food delivery

**1.Obiectivul temei**

Obiectivul principal al temei este proiectarea si implementarea unei aplicatii de management a produselor care pot fi cumparate dintr-un restaurant. Aceasta aplicatie va permite crearea de conturi pentru a accesa diversele ei functionalitati. Astfel, vor exista trei tipuri de utilizatori: client, employee si administrator. Utilizatorul client va avea acces la produse, avand la dispozitie optiuni de cautare si filtrare a acestora. In plus, clientul va putea adauga produsele dorite intr-un cart pentru a le comanda. Utilizatorul administrator are posibilitatea de a adauga/sterge/moodifica produsele disponibile, de a crea meniuri care contin produse sau alte meniuri si de a genera diverse rapoarte despre produse, comenzi sau client.

Sub-obiective:

* Analiza problemei si identificarea necesitatilor
* Proiectarea aplicatiei
* Implementarea aplicatiei
* Testarea aplicatiei

**2. Analiza problemei, modelare, scenariu, cazuri de utilizare**

Aplicatia va folosi fisiere .txt pentru a salva conturile create: client.txt, employee.txt, administrator.txt si comenzile efectuate de clienti: orders.txt. Pe langa acestea, va mai exista un fisier .csv unde vor fi salvate toate produsele: products.csv.

Se va realiza o interfata grafica dedicata pentru fiecare utilizator in parte prin care acesta va folosi functionalitatile implementate de aplicatie intr-un mod user-friendly.

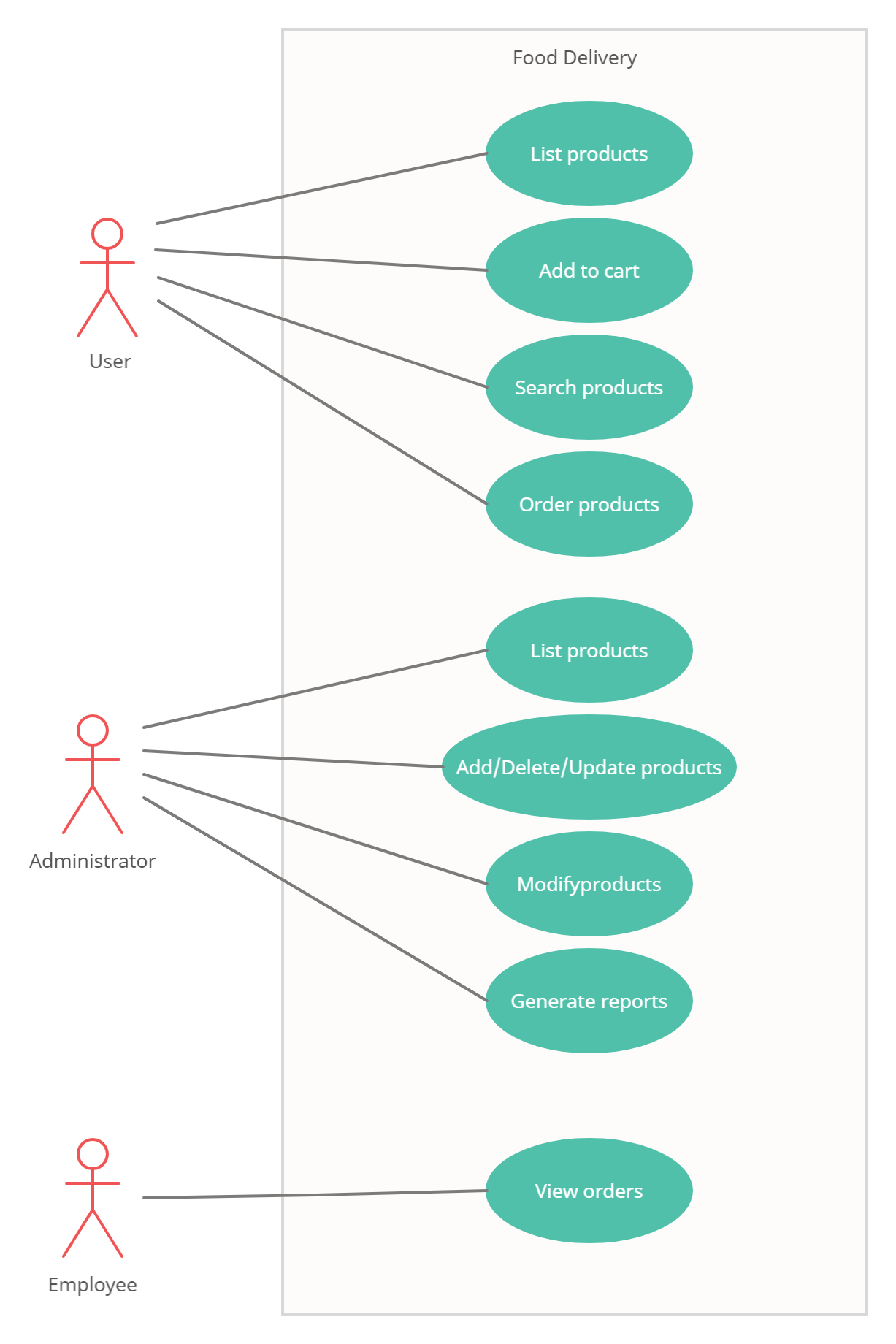
Astfel, clientul va avea acces la o lista cu produse pe care le va putea adauga intr-un cos de cumparaturi(cart) ca mai apoi sa plaseze o comanda. Lista de produse va putea fi filtrata in functie de nume, pret minim si rating minim. Cand produsul dorit este gasit se va face click pe acesta (pentru a-l selecta) si apasand un buton, acesta va fi adaugat in cart. Dupa ce toate produsele dorite se afla in cart, comanda se va putea efectua instant apasand alt buton.

Administratorul, ca si clientul, va avea acces la lista de produse, dar spre deosebire de acesta el o va putea modifica dupa bunul plac. Astfel, cand selectam un produs din lista, detaliile acestuia vor fi afisate in paralel de unde vor putea fi modificate. El va mai putea adauga sau sterge produse si sa creeze meniuri compuse din alte produse sau meniuri. Modul de creare a acestor produse este simplu: administratorul va furniza un nume si un rating pentru meniu iar apoi va selecta din lista produsele care il compun. Pentru finalizare se va apasa pe un buton special care crea propriu-zis meniul si il va adauga in lista de produse. In plus, administratorul va avea posibilitatea de a genera rapoarte despre client, produse sau comenzi.

Angajatul (employee), va avea acces la comenzile clientilor si va fi notificat cand o comanda este plasata.

Principalele proprietati ale aplicatiei sunt:

* Exista trei actori: client, administrator si employee.
* Acestia beneficiaza de functionalitatile aplicatiei prin interfata grafica dedicata
* Conturile create si comenzile sunt salvate in fisiere .txt
* Produsele sunt salvate intr-un fisier .csv
* Pentru fiecare user va exista o interfata grafica proprie tipului de utilizator (client, administrator, employee)

****

Caz de utilizare: Plasare unei comenzi de catre client

Actor principal: client

Scenariu:

1. Clientul introduce filtrele de cautare dorite si apasa pe butonul <List products>.
2. Clientul selecteaza produsul dorit si il introduce in cosul de cumparaturi apasand butonul <Add product>
3. Clientul adauga in acest fel toate produsele dorite in cosul de cumparaturi
4. Cand apasa butonul <Place order> pentru a plasa comanda

Caz de utilizare: Modificarea produselor din lista

Actor principal: administrator

Scenariu:

1. Administratorul apasa butonul <List products>
2. Acesta va selecta un produs din lista cu un click pe produsul respectiv
3. Detaliile produsului vor aparea in mai multe TextField-uri de unde se vor putea modifica(daca produsul nu este unul compus)
4. Pentru a salva modificarile se va apasa butonul <Add/Update product>

Caz de utilizare: Crearea de meniuri

Actor principal: administrator

Scenariu:

1. Administratorul apasa butonul <List products>
2. Acesta va selecta un produs din lista cu un click pe produsul respectiv
3. Se va apasa butonul <Add to composite> pentru a adauga produsul in meniu
4. Se va da un nume si un rating meniului prin TextField-urile aferente
5. Pentru a salva produsul se va apasa butonul <Add composite product>

Caz de utilizare: Generarea de rapoarte

Actor principal: administrator

Scenariu:

1. Administratorul apasa butonul <Reports>
2. Acesta va facilita datele necesare pentru generarea raportului dorit si va apasa butonul aferent

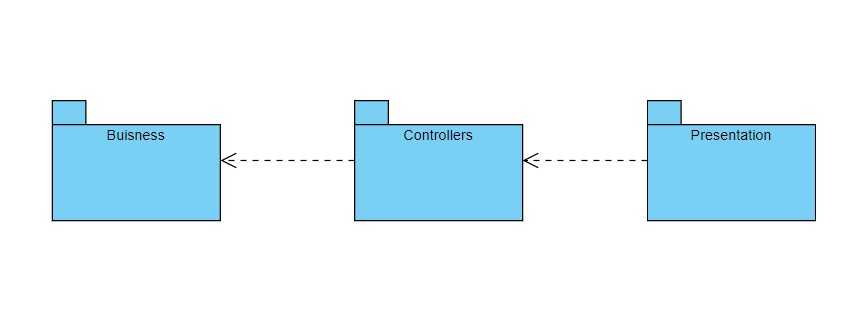
**3.Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, algoritmi, interfata utilizator)**

Proiectul va fi impartit in 3 pachete: Buisness, Controllers si Presentation.

In pachetul Buisness se vor implementa functionalitatile de baza ale aplicatiei. Acesta va contine atat clase care vor stoca datele necesare aplicatiei (conturile utilizatorilor, produsele si comenzile) cat si unele functionalitati (e.g. importul de produse din fisierul .csv)

In pachetul Controllers se afla toate controller-ele interfetelor grafice. In aceste clase controller se vor implementa functionalitatile propriu-zise ale interfetelor grafice: client, administrator, reports, login, sugn up.

In pachetul Presentation se afla fisierele .fxml aale interfetelor grafice.



In primul rand, pentru stocarea produselor vom folosi design pattern-ul composite. Astfel aplicatia va avea o mare flexibilitate in a crea produse noi compuse din alte produse. Astfel clasa abstracta MenuItem va fi extinsa de clasele: BaseProduct (folosita pentru produsele de baza) si CompositeProduct (folosita pentru produsele compuse din mai multe produse de baza sau/si alte produse compuse).

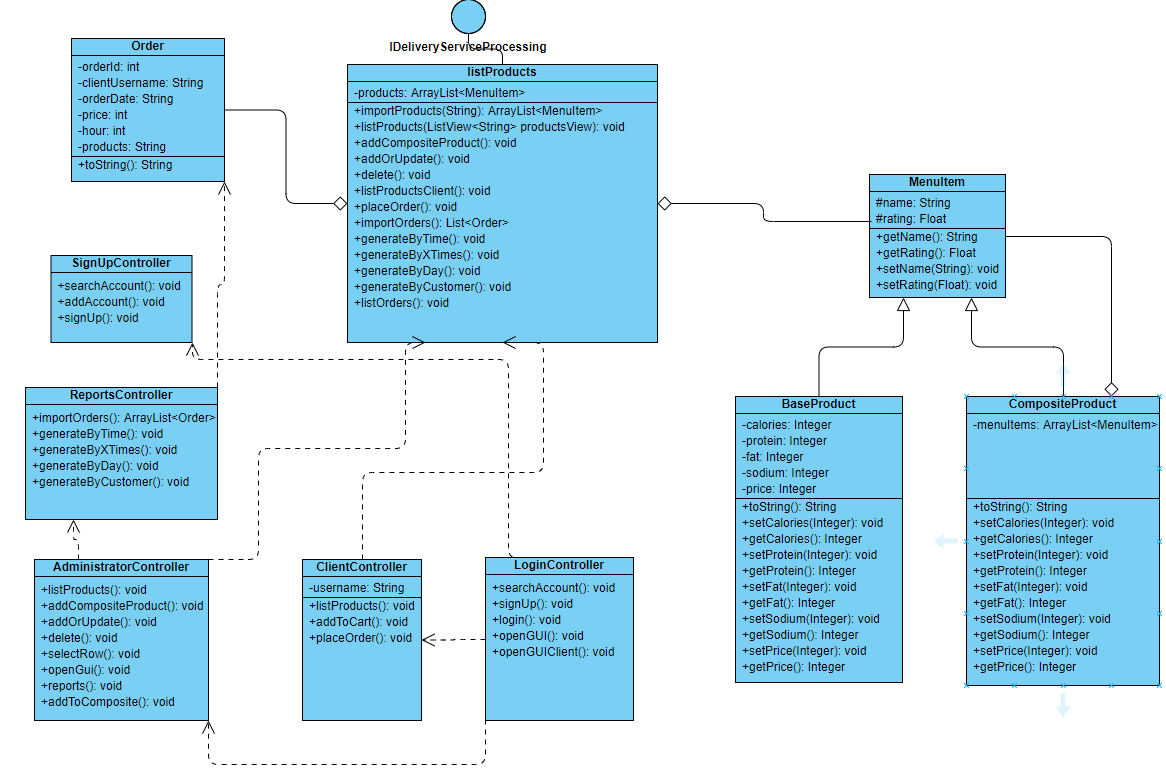
In al doilea rand, clasele controller vor implementa functionalitatile necesare interfetei grafice. Clasa LoginController va trebui sa acceseze fisierele .txt in care sunt salvate conturile deja create, avand un scop de gateway pentru celelalte interfete. In clasa SignUpController se vor implementa functionalitatile de creare a unui utilizator nou. Clasa ReportsController va fi folosita de administrator in cazul in care acesta doreste sa genereze anumite rapoarte legate de produse, comenzi sau client.

Pe langa aceste clase vom avea si clasa Order destinata modelarii comenzilor effectuate de client.

Ca si clasa centrala vom avea DeliveryService, care va contine meniul restauratului (toate produsele).

Pentru filtrarea produselor dupa nume, raiting sau pret se vor folosi stream-uri si expresii lambda. Acestea vor mai fi folosite recurent prin codul aplicatiei pentru simplitatea de utilizare.

Diagrama UML pentru clase:



**4. Implementare**

***Clasa MenuItem:***

Are ca atribute elementele definitorii ale unui produs: nume(String name) si rating(String rating)

Este o clasa abstracta folosita in design pattern-ul composite. Aceasta va avea rolul de a reuni subclasele BaseProduct si CompositeProduct prin crearea unor functionalitati commune. Singurele metode implementate sunt getter-e si setter-e. Pe langa acestea se declara metode abstracte de get si toString care vor fi implementate de subclasele sale.

public abstract class MenuItem *{* protected String name;  
 protected Float rating;  
  
 public MenuItem*(*String name, Float rating*) {* this.name = name;  
 this.rating = rating;  
 *}* public String getName*() {* return name; *}* public Float getRating*() {* return rating; *}* public void setName*(*String name*) {* this.name = name; *}* public void setRating*(*Float rating*) {* this.rating = rating; *}* public abstract String toString*()*;  
  
 public abstract Integer getPrice*()*;  
 public abstract Integer getCalories*()*;  
 public abstract Integer getProtein*()*;  
 public abstract Integer getFat*()*;  
 public abstract Integer getSodium*()*;

}

***Clasa BaseProduct:***

Are ca atribute detaliile unui produs de baza: calorii, proteine, grasimi, sodiu, pret. Aceasta clasa mosteneste MenuItem si ii implementeaza metodele abstracte. Acestea sunt doar metode de get si o metoda toString. Pe lanaga acestea s-au implementat si metode de set care vor fi folosite pentru a face update unui produs.

public class BaseProduct extends MenuItem*{* private Integer calories;  
 private Integer protein;  
 private Integer fat;  
 private Integer sodium;  
 private Integer price;  
  
 public BaseProduct*(*String name, Float rating, Integer calories, Integer protein, Integer fat, Integer sodium, Integer price*) {* super*(*name, rating*)*;  
 this.calories = calories;  
 this.protein = protein;  
 this.fat = fat;  
 this.sodium = sodium;  
 this.price = price;  
 *}* public void setCalories*(*Integer calories*) {* this.calories = calories;  
 *}* public void setProtein*(*Integer protein*) {* this.protein = protein;  
 *}* public void setFat*(*Integer fat*) {* this.fat = fat;  
 *}* public void setSodium*(*Integer sodium*) {* this.sodium = sodium;  
 *}* public void setPrice*(*Integer price*) {* this.price = price;  
 *}* @Override  
 public Integer getCalories*() {* return calories; *}* @Override  
 public Integer getProtein*() {* return protein; *}* @Override  
 public Integer getFat*() {* return fat; *}* @Override  
 public Integer getSodium*() {* return sodium; *}* @Override  
 public Integer getPrice*() {* return price; *}* @Override  
 public String toString*() {* return super.getName*()*; *}*

*}*

***Clasa CompositeProduct:***

In virtutea design pattern-ului composite aceasta clasa are ca atribut un o lista de MenuItems. Astfel se creaza o anumita “recursivitate” care ne va permite sa cream meniuri compuse din produse de baza dar si din alte meniuri. Aceasta clasa are ca metode implementarea getter-lor declarate abstract in MenuItem si metoda toString.

public class CompositeProduct extends MenuItem*{* private ArrayList*<*MenuItem*>* menuItems;  
  
 public CompositeProduct*(*String name, Float rating*) {* super*(*name, rating*)*;  
 this.menuItems = new ArrayList*<>()*;  
 *}* public void addMenuItem*(*MenuItem menuItem*) {* this.menuItems.add*(*menuItem*)*;  
 *}* @Override  
 public String toString*() {* String s = new String*()*;  
 s = super.getName*()* + "(";  
 for *(*MenuItem menuItem : this.menuItems*) {* s += menuItem.toString*()* + "; ";  
 *}* s = s.substring*(*0, s.length*()* - 2*)*;  
 s += ")";  
  
 return s;  
 *}* @Override  
 public Integer getPrice*() {* Integer price = 0;  
 for *(*MenuItem menuItem : this.menuItems*) {* price += menuItem.getPrice*()*;  
 *}* return price;  
 *}* @Override  
 public Integer getCalories*() {* Integer calories = 0;  
 for *(*MenuItem menuItem : this.menuItems*) {* calories += menuItem.getCalories*()*;  
 *}* return calories;  
 *}* @Override  
 public Integer getProtein*() {* Integer protein = 0;  
 for *(*MenuItem menuItem : this.menuItems*) {* protein += menuItem.getProtein*()*;  
 *}* return protein;  
 *}* @Override  
 public Integer getFat*() {* Integer fat = 0;  
 for *(*MenuItem menuItem : this.menuItems*) {* fat += menuItem.getFat*()*;  
 *}* return fat;  
 *}* @Override  
 public Integer getSodium*() {* Integer sodium = 0;  
 for *(*MenuItem menuItem : this.menuItems*) {* sodium += menuItem.getSodium*()*;  
 *}* return sodium;  
 *}*

*}*

***Clasa DeliveryService:***

In aceasta clasa se vor stoca produsele extrase din fisierul .csv intr-o lista de MenuItems.

Metoda importProducts() foloseste stream-uri si expresii lambda pentru a citi linie cu linie produsele stocate in fisierul .csv. Apoi se va face split asupra liniei rezultate pentru a delimita caracteristicile produsului care vor avea rol de argumente in constructorul clasei BaseProduct. Astfel, toate produsele stocate in fisierul .csv vor fi instantiate intr-o lista de MenuItems care va fi returnata de metoda.

@Override  
public ArrayList*<*MenuItem*>* importProducts*(*String path*) {* if*(*products!=null*)* products.clear*()*;  
 Pattern pattern = Pattern.*compile(*","*)*;  
 try *(*Stream*<*String*>* lines = Files.*lines(*Path.*of(*"products.csv"*))) {* List*<*BaseProduct*>* products = lines.skip*(*1*)*.map*(*line -> *{* String*[]* arr = pattern.split*(*line*)*;  
 return new BaseProduct*(* arr*[*0*]*,  
 Float.*parseFloat(*arr*[*1*])*,  
 Integer.*parseInt(*arr*[*2*])*,  
 Integer.*parseInt(*arr*[*3*])*,  
 Integer.*parseInt(*arr*[*4*])*,  
 Integer.*parseInt(*arr*[*5*])*,  
 Integer.*parseInt(*arr*[*6*]))*;  
 *})*.filter*(filtering(*BaseProduct::getName*))* .collect*(*Collectors.*toList())*;  
 ArrayList*<*MenuItem*>* productsList= new ArrayList*<*MenuItem*>()*;  
 for *(*BaseProduct baseProduct : products*) {* productsList.add*(*baseProduct*)*;  
 *}* return productsList;  
 *}*catch *(*Exception e*){* e.printStackTrace*()*;  
 *}* return null;

}

***Clasa LoginController:***

Aceasta clasa implementeaza functionalitatile de logare ca unul dintre cei trei utilizatori posibili: client, administrator, employee. Pentru a alege ca ce user urmeaza sa te loghezi se foloseste un ChoiceBox. Conturile existente sunt salvate in trei fisiere diferite in conformitate cu tipurile de utilizatori posibili: client.txt, administrator.txt, employee.txt.

Metoda searchAccount() este folosita in momentul autentificarii pentru a valida daca username-ul si parola exista in baza de date.

private boolean searchAccount*(*String username, String password, String fileName*)* throws FileNotFoundException *{* String s;  
 File file = new File*(*fileName + ".txt"*)*;  
 Scanner sc = new Scanner*(*file*)*;  
  
 while *(*sc.hasNextLine*())  
 {* s = sc.nextLine*()*;  
 String*[]* split = s.split*(*" "*)*;  
 if*(*split*[*0*]*.equals*(*username*)* && split*[*1*]*.equals*(*password*)) {* sc.close*()*;  
 return true;  
 *}  
 }* sc.close*()*;  
  
 return false;

}

Metodele login() este folosita pentru a lansa interfata grafica specifica tipului de utilizator pentru care username-ul si parola au fost gasite in baza de date. Metoda searchAccout() este folosita pentru validarea acestora.

public void login*()* throws FileNotFoundException *{* if*(*chooseUser.getSelectionModel*()*.getSelectedItem*()* == choiseList.get*(*0*)) {* if*(*searchAccount*(*this.username.getText*()*,this.password.getText*()*, "client"*)) {* openGUIClient*(*"../presentation/Client.fxml",1154, 520*)*;  
 *}  
 }* if*(*chooseUser.getSelectionModel*()*.getSelectedItem*()* == choiseList.get*(*1*)) {* if*(*searchAccount*(*this.username.getText*()*,this.password.getText*()*, "employee"*)) {* openGUI*(*"../presentation/Employee.fxml",1154, 520*)*;  
 *}  
  
 }* if*(*chooseUser.getSelectionModel*()*.getSelectedItem*()* == choiseList.get*(*2*)) {* if*(*searchAccount*(*this.username.getText*()*,this.password.getText*()*, "administrator"*)) {* openGUI*(*"../presentation/Administrator.fxml",1154, 520*)*;  
 *}  
 }*

*}*

Metoda openGUI() este folosita pentru lansarea propriu-zisa a interfetelor grafice.

private void openGUI*(*String name, int x, int y*)  
{* try*{* FXMLLoader loader = new FXMLLoader*(*getClass*()*.getResource*(*name*))*;  
 Parent root = loader.load*()*;  
 Stage stage = new Stage*()*;  
 stage.setTitle*(*"Food Delivery"*)*;  
 stage.setScene*(*new Scene*(*root, x, y*))*;  
 stage.show*()*;  
 *}*catch*(*Exception e*){* e.getCause*()*;  
 e.printStackTrace*()*;  
 *}*

*}*

***Clasa ClientController:***

Aceasta clasa implementeaza functionalitatile interfetei grafice ale clientului.

Metoda listProducts() afiseaza intr-un ListView toate produsele care pot fi comandate de catre client. In plus, in interfata grafica a clientului exista trei TextField-uri: name, maximum price si minimum rating. Cu informatile oferite de client in aceste TextField-uri, produsele vor fi filtrate dupa caracteristicile mentionate si vor fi afisate doar cele relevante. Aceasta filtrare se realizeaza cu ajutorul stream-urilor si a expresilor lambda.

public void listProducts*() {* if*(*name.getText*()*.equals*(*""*)* && price.getText*()*.equals*(*""*)* && minRating.getText*()*.equals*(*""*)) {* deliveryService.products.forEach*(*a -> productsView.getItems*()*.add*(*a + " Price: " + a.getPrice*()* + " Rating: " + a.getRating*()* + " Calories: " + a.getCalories*()* +" Protein: " + a.getProtein*()* + " Fat: " + a.getFat*()* + " Sodium: " + a.getSodium*()))*;  
 *}* else if*(*name.getText*()*.equals*(*""*)* && price.getText*()*.equals*(*""*)) {* productsView.getItems*()*.clear*()*;  
  
 assert deliveryService.products != null;  
 List*<*MenuItem*>* list = deliveryService.products  
 .stream*()* .filter*(*c->c.getRating*()* >= Float.*parseFloat(*minRating.getText*()))* .collect*(*Collectors.*toList())*;  
 list.forEach*(*a->productsView.getItems*()*.add*(*a + " Price: " + a.getPrice*()* + " Rating: " + a.getRating*()* + " Calories: " + a.getCalories*()* +" Protein: " + a.getProtein*()* + " Fat: " + a.getFat*()* + " Sodium: " + a.getSodium*()))*;  
 *}* else if*(*name.getText*()*.equals*(*""*)* && minRating.getText*()*.equals*(*""*)) {* productsView.getItems*()*.clear*()*;  
  
 assert deliveryService.products != null;  
 List*<*MenuItem*>* list = deliveryService.products  
 .stream*()* .filter*(*c->c.getPrice*()* <= Integer.*parseInt(*price.getText*()))* .collect*(*Collectors.*toList())*;  
 list.forEach*(*a->productsView.getItems*()*.add*(*a + " Price: " + a.getPrice*()* + " Rating: " + a.getRating*()* + " Calories: " + a.getCalories*()* +" Protein: " + a.getProtein*()* + " Fat: " + a.getFat*()* + " Sodium: " + a.getSodium*()))*;  
 *}* else if*(*price.getText*()*.equals*(*""*)* && minRating.getText*()*.equals*(*""*)) {* productsView.getItems*()*.clear*()*;  
  
 assert deliveryService.products != null;  
 List*<*MenuItem*>* list = deliveryService.products  
 .stream*()* .filter*(*c->c.getName*()*.toLowerCase*()*.contains*(*name.getText*()*.toLowerCase*()))* .collect*(*Collectors.*toList())*;  
 list.forEach*(*a->productsView.getItems*()*.add*(*a + " Price: " + a.getPrice*()* + " Rating: " + a.getRating*()* + " Calories: " + a.getCalories*()* +" Protein: " + a.getProtein*()* + " Fat: " + a.getFat*()* + " Sodium: " + a.getSodium*()))*;  
 *}* else if*(*name.getText*()*.equals*(*""*)) {* productsView.getItems*()*.clear*()*;  
  
 assert deliveryService.products != null;  
 List*<*MenuItem*>* list = deliveryService.products  
 .stream*()* .filter*(*c->c.getRating*()* >= Float.*parseFloat(*minRating.getText*()))* .filter*(*c->c.getPrice*()* <= Integer.*parseInt(*price.getText*()))* .collect*(*Collectors.*toList())*;  
 list.forEach*(*a->productsView.getItems*()*.add*(*a + " Price: " + a.getPrice*()* + " Rating: " + a.getRating*()* + " Calories: " + a.getCalories*()* +" Protein: " + a.getProtein*()* + " Fat: " + a.getFat*()* + " Sodium: " + a.getSodium*()))*;  
 *}* else if*(*price.getText*()*.equals*(*""*)) {* productsView.getItems*()*.clear*()*;  
  
 assert deliveryService.products != null;  
 List*<*MenuItem*>* list = deliveryService.products  
 .stream*()* .filter*(*c->c.getName*()*.toLowerCase*()*.contains*(*name.getText*()*.toLowerCase*()))* .filter*(*c->c.getRating*()* >= Float.*parseFloat(*minRating.getText*()))* .collect*(*Collectors.*toList())*;  
 list.forEach*(*a->productsView.getItems*()*.add*(*a + " Price: " + a.getPrice*()* + " Rating: " + a.getRating*()* + " Calories: " + a.getCalories*()* +" Protein: " + a.getProtein*()* + " Fat: " + a.getFat*()* + " Sodium: " + a.getSodium*()))*;  
 *}* else if*(*minRating.getText*()*.equals*(*""*)) {* productsView.getItems*()*.clear*()*;  
  
 assert deliveryService.products != null;  
 List*<*MenuItem*>* list = deliveryService.products  
 .stream*()* .filter*(*c->c.getName*()*.toLowerCase*()*.contains*(*name.getText*()*.toLowerCase*()))* .filter*(*c->c.getPrice*()* <= Integer.*parseInt(*price.getText*()))* .collect*(*Collectors.*toList())*;  
 list.forEach*(*a->productsView.getItems*()*.add*(*a + " Price: " + a.getPrice*()* + " Rating: " + a.getRating*()* + " Calories: " + a.getCalories*()* +" Protein: " + a.getProtein*()* + " Fat: " + a.getFat*()* + " Sodium: " + a.getSodium*()))*;  
 *}* else *{* productsView.getItems*()*.clear*()*;  
  
 assert deliveryService.products != null;  
 List*<*MenuItem*>* list = deliveryService.products  
 .stream*()* .filter*(*c->c.getName*()*.toLowerCase*()*.contains*(*name.getText*()*.toLowerCase*()))* .filter*(*c->c.getRating*()* >= Float.*parseFloat(*minRating.getText*()))* .filter*(*c->c.getPrice*()* <= Integer.*parseInt(*price.getText*()))* .collect*(*Collectors.*toList())*;  
 list.forEach*(*a->productsView.getItems*()*.add*(*a + " Price: " + a.getPrice*()* + " Rating: " + a.getRating*()* + " Calories: " + a.getCalories*()* +" Protein: " + a.getProtein*()* + " Fat: " + a.getFat*()* + " Sodium: " + a.getSodium*()) )*;  
 *}*

*}*

Metoda placeOrder() adauga in baza de date order.txt comanda efectuata de client.

public void placeOrder*()* throws IOException *{* String order = username + "/";  
  
 //get order id  
 String s;  
 File file = new File*(*"orders.txt"*)*;  
 Scanner sc = new Scanner*(*file*)*;  
 s = sc.nextLine*()*;  
 int orderID = Integer.*parseInt(*s*)* + 1;  
 sc.close*()*;  
  
 order += orderID + "/";  
  
 //get items  
 for*(*int i=0; i<cartView.getItems*()*.size*()*; i++*) {* String line = *(*String*)*cartView.getItems*()*.get*(*i*)*;  
 String name = line.substring*(*0, line.indexOf*(*" Price:"*))*;  
 order += name + ",";  
 *}* LocalDateTime date = LocalDateTime.*now()*;  
 DateTimeFormatter format = DateTimeFormatter.*ofPattern(*"dd-MM-yyyy HH:mm:ss"*)*;  
 String formatDateTime = date.format*(*format*)*;  
  
 order = order.substring*(*0,order.length*()*-1*)*;  
 order += "/" + labelPrice.getText*()*;  
 order += "/" + formatDateTime + "\n";  
  
 System.*out*.println*(*order*)*;  
  
 sc = new Scanner*(*file*)*;  
 StringBuffer buffer = new StringBuffer*()*;  
 while *(*sc.hasNextLine*()) {* buffer.append*(*sc.nextLine*()*+System.*lineSeparator())*;  
 *}* String fileContents = buffer.toString*()*;  
 int noOfDigits = 0;  
 int aux = orderID;  
 while*(*aux!=0*)  
 {* noOfDigits ++;  
 aux/=10;  
 *}* fileContents = fileContents.substring*(*noOfDigits*)*;  
 fileContents = orderID + fileContents + order;  
  
 FileWriter fileWriter = new FileWriter*(*"orders.txt"*)*;  
 fileWriter.write*(*fileContents*)*;  
 fileWriter.close*()*;

}

***Clasa AdministratorController:***

Aceasta clasa implementeaza functionalitatile interfetei grafice ale administratorului.

Metoda addCompositeProduct() adauga un meniu in baza de date. Dupa ce numele, rating-ul si produsele care compun meniul au fost specificate, aceasta metoda le va cauta pe toate in lista de MenuItems si va crea o instanta de MenuItem compusa produsele specificate iar apoi se va adauga meniul nou creat si in baza de date aaplicatiei.

public void addCompositeProduct*()* throws IOException *{* CompositeProduct comp = new CompositeProduct*(*compositeProductName.getText*()*, Float.*valueOf(*compositeProductRating.getText*()))*;  
  
 for*(*int i=0; i<compositeProductView.getItems*()*.size*()*; i++*) {* String line = compositeProductView.getItems*()*.get*(*i*)*;  
 String name = line.substring*(*0, line.indexOf*(*" Price:"*))*;  
  
 List*<*MenuItem*>* list = deliveryService.products  
 .stream*()* .filter*(*c->c.getName*()*.equals*(*name*))* .collect*(*Collectors.*toList())*;  
 list.forEach*(*a->a.setName*(*a.getName*()*.substring*(*0,a.getName*()*.length*()*-1*)))*;  
 list.forEach*(*a->comp.addMenuItem*(*a*))*;  
 *}* deliveryService.products.add*(*comp*)*;  
 listProducts*()*;  
 compositeProductName.clear*()*;  
 compositeProductRating.clear*()*;  
 compositeProductView.getItems*()*.clear*()*;  
  
 //Title,Rating,Calories,Protein,Fat,Sodium,Price  
 String s = comp.toString*()* + " ," + comp.getRating*()* + "," + comp.getCalories*()* + "," + comp.getProtein*()* + "," + comp.getFat*()* + "," + comp.getSodium*()* + "," + comp.getPrice*()* + "\n";  
 FileWriter fileWriter = new FileWriter*(*"products.csv", true*)*;  
 fileWriter.write*(*s*)*;  
 fileWriter.close*()*;

}

Metoda addOrUpdate() va face append in fisierul .csv cu o linie noua in cazul de add sau va inlocui o linie deja existenta in cazul de update. La finalul metodei se apeleaza listProducts() pentru a face refresh la produsele afisate in ListView-ul administratorului.

public void addOrUpdate*()* throws IOException *{* List*<*MenuItem*>* list = deliveryService.products  
 .stream*()* .filter*(*c->c.getName*()*.equals*(*name.getText*()))* .collect*(*Collectors.*toList())*;  
  
 String item = name.getText*()* + "," + rating.getText*()* + "," + calories.getText*()* + "," + protein.getText*()* + "," + fat.getText*()* + "," + sodium.getText*()* + "," + price.getText*()*;  
 System.*out*.println*(*item*)*;  
 System.*out*.println*(*oldLine*)*;  
 if*(*list.isEmpty*()) {* Files.*write(*Paths.*get(*"products.csv"*)*,  
 item.getBytes*()*,  
 StandardOpenOption.*APPEND)*;  
 *}* else *{* try *(*Stream*<*String*>* lines = Files.*lines(*Path.*of(*"products.csv"*))) {* List*<*String*>* stringList = lines  
 .map*(*x-> x.replaceAll*(*oldLine, item*))* .collect*(*Collectors.*toList())*;  
 Files.*write(*Path.*of(*"products.csv"*)*, stringList*)*;  
 *}  
 }* oldLine = item;  
 listProducts*()*;

}

Metoda delete() va sterge produsul selectat din baza de date.

public void delete*()* throws IOException *{* File file = new File*(*"products.csv"*)*;  
 System.*out*.println*(*name.getText*())*;  
 String sRating;  
 if*(*rating.getText*()*.contains*(*".0"*)* && rating.getText*()*.length*()* == 3*)  
 {* sRating = rating.getText*()*.substring*(*0,1*)*;  
 *}* else  
 sRating = rating.getText*()*;  
 String item = name.getText*()* + "," + sRating + "," + calories.getText*()* + "," + protein.getText*()* + "," + fat.getText*()* + "," + sodium.getText*()* + "," + price.getText*()*;  
 List*<*String*>* stringList = Files.*lines(*file.toPath*())* .filter*(*x -> !x.equals*(*item*))* .collect*(*Collectors.*toList())*;  
 System.*out*.println*(*item+"\n"+stringList.get*(*1*))*;  
 Files.*write(*file.toPath*()*, stringList, StandardOpenOption.*WRITE*, StandardOpenOption.*TRUNCATE\_EXISTING)*;  
 listProducts*()*;

}

Metoda selectRow() are rolul ca atunci cand se face click pe un produs din ListView sa afiseze detaliile acestuia in TextField-urile alaturate din interfata grafica.

public void selectRow*() {* String s = productsView.getSelectionModel*()*.getSelectedItem*()*;  
 if*(*s==null*)* return;  
 name.setText*(*s.substring*(*0,s.indexOf*(*" Price: "*)))*;  
 price.setText*(*s.substring*(*s.indexOf*(*"Price: "*)*+7,s.indexOf*(*" Rating:"*)))*;  
 rating.setText*(*s.substring*(*s.indexOf*(*"Rating: "*)*+8,s.indexOf*(*" Calories:"*)))*;  
 calories.setText*(*s.substring*(*s.indexOf*(*"Calories: "*)*+10,s.indexOf*(*" Protein:"*)))*;  
 protein.setText*(*s.substring*(*s.indexOf*(*"Protein: "*)*+9,s.indexOf*(*" Fat:"*)))*;  
 fat.setText*(*s.substring*(*s.indexOf*(*"Fat: "*)*+5,s.indexOf*(*" Sodium:"*)))*;  
 sodium.setText*(*s.substring*(*s.indexOf*(*"Sodium: "*)*+8*))*;  
 line = s;  
 oldLine = name.getText*()* + "," + rating.getText*()* + "," + calories.getText*()* + "," + protein.getText*()* + "," + fat.getText*()* + "," + sodium.getText*()* + "," + price.getText*()*;

}

***Clasa ReportsController:***

Aceasta clasa contine functionalitatile de generare a raporturilor de catre administrator. Conditile de generare a unui raport trebuie introduse in TextFild-urile specific acestuia.

Metoda generateByTime() afiseaza comenzile care au fost effectuate intr-un anumit interval de timp dintr-o zi.

public void generateByTime*() {* System.*out*.println*(*"Report by hour"*)*;  
  
 orders = importOrders*()*;  
 String startingTime = this.startingTime.getText*()*;  
 String endingTime = this.endingTime.getText*()*;  
  
 if*(*startingTime == null || endingTime == null*)* return;  
  
 int startTime = Integer.*parseInt(*this.startingTime.getText*())*;  
 int endTime = Integer.*parseInt(*this.endingTime.getText*())*;  
 List*<*Order*>* list = orders  
 .stream*()* .filter*(*c -> c.getHour*()* >= startTime && c.getHour*()* <= endTime*)* .collect*(*Collectors.*toList())*;  
 orders = list;  
 orders.forEach*(*System.*out*::println*)*;

}

Metoda GenerateByDay() afiseaza comenzile care au fost effectuate intr-o anumita zi.

public void generateByDay*(){* System.*out*.println*(*"Report by day"*)*;  
  
 orders = importOrders*()*;  
 List*<*Order*>* prod = orders  
 .stream*()* .filter*(*c ->c.getDay*()*.equals*(*date.getText*()))* .collect*(*Collectors.*toList())*;  
  
 for *(*Order x: prod*) {* System.*out*.println*(*x*)*;  
 *}*

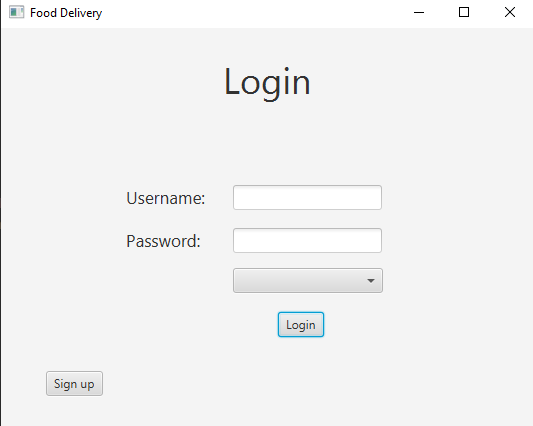
*}*

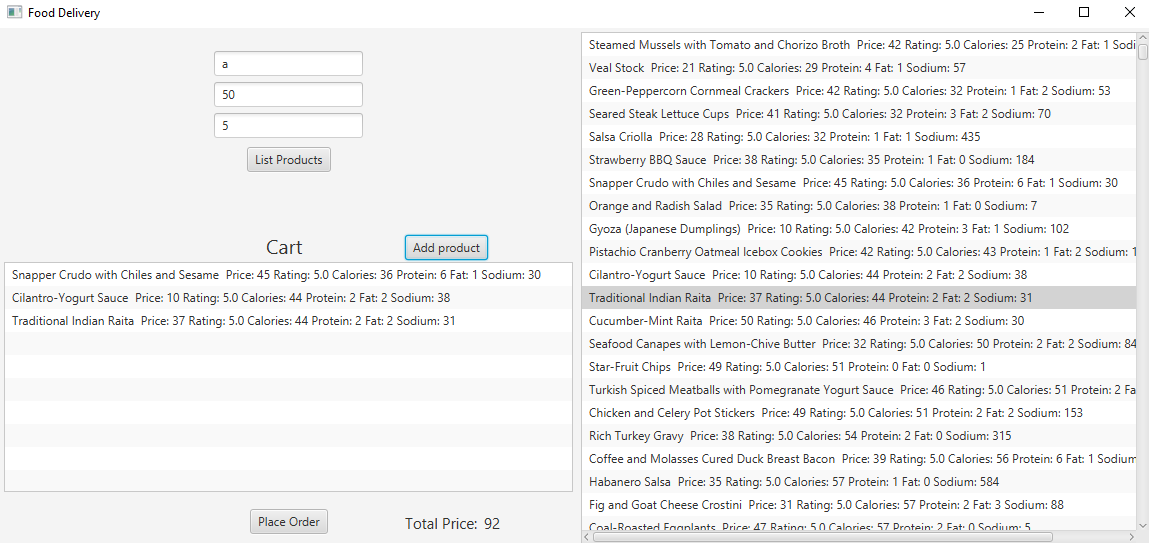
Metoda generateByCustomer() afiseaza toti clientii care au comandat de un numar minim de ori.

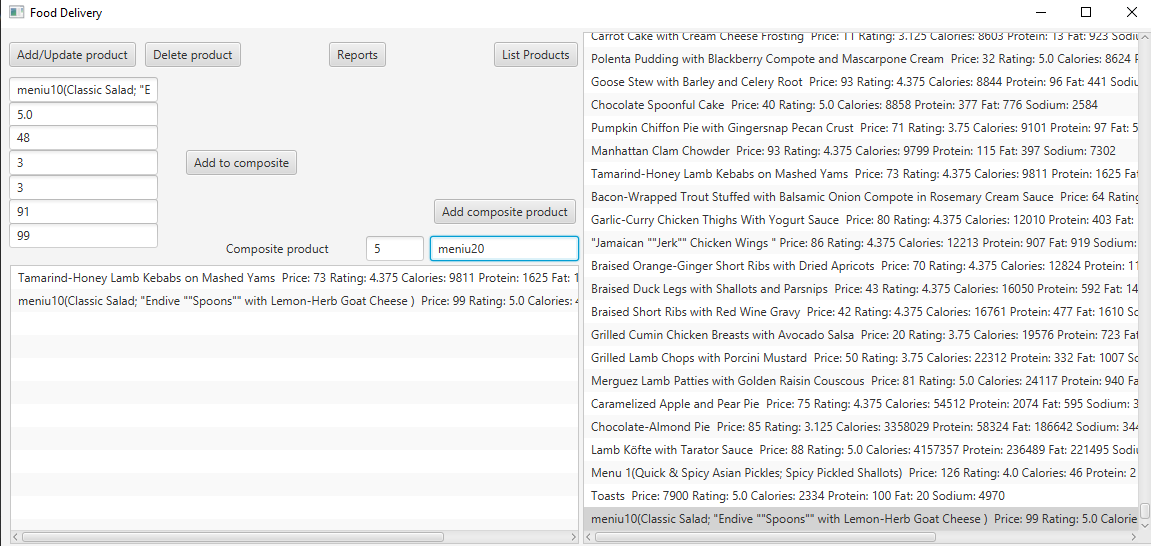
public void generateByCustomer*(){* System.*out*.println*(*"Report by customer"*)*;  
 orders = importOrders*()*;  
 List*<*Order*>* prod = orders  
 .stream*()* .filter*(*p->p.getPrice*()* >= Integer.*parseInt(*minPrice.getText*()))* .collect*(*Collectors.*toList())*;  
 ArrayList*<*String*>* usernames = new ArrayList*<>()*;  
 ArrayList*<*Integer*>* noOfOrders = new ArrayList*<>()*;  
 for *(*Order x: prod*) {* if*(*!usernames.contains*(*x.getClientUsername*()))  
 {* usernames.add*(*x.getClientUsername*())*;  
 noOfOrders.add*(*1*)*;  
 *}* else *{* noOfOrders.set*(*usernames.indexOf*(*x.getClientUsername*())*, noOfOrders.get*(*usernames.indexOf*(*x.getClientUsername*()))* + 1*)*;  
 *}  
 }* System.*out*.println*(*noOfOrders*)*;  
 int i=0;  
 for *(*Integer x : noOfOrders*) {* if*(*x>Integer.*parseInt(*clientMinOrders.getText*()))* System.*out*.println*(*usernames.get*(*i*))*;  
 i++;  
 *}*

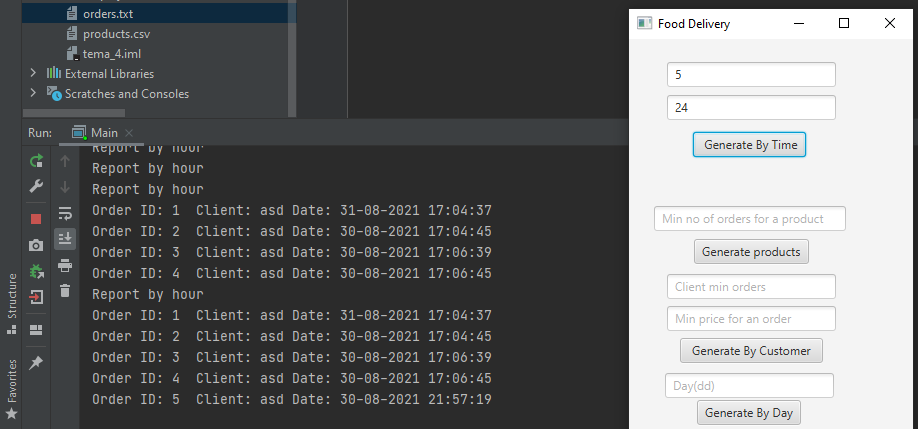
*}*

1. **Rezultate**









**6. Concluzie**

In timpul implementarii acestui proiect a trebuit sa caut informatii legate de metoda reflectarii si modul de conexiune cu o baza de date. Cum am spus sim ai sus, mai sunt unele erori pe care nu le-am rezolvat dar per total aplicatie functioneaza conform asteptarilor.

**7. Bibliografie**

* <https://www.oracle.com/technical-resources/articles/java/javareflection.html>
* <https://www3.ntu.edu.sg/home/ehchua/programming/java/JDBC_Basic.html>