DOCUMENTATIE

TEMA 4

NUME STUDENT : FURTOS RARES-MIHAI

GRUPA: 30228

CUPRINS

1.Obiectivul Temei

2.Analiza problemei

3.Proiectare

4.Implementare

5.Rezultate

6.Concluzii

7.Bibliografie

1.Obiectivul Temei

Principalul motiv al acestei teme este de a proiecta si implementa un program care sa simuleze gestiunea unei afaceri bazate pe livrari la domiciliu . Avem la baza trei tipuri de utilizatori , administrator , client si angajat . Toti au cateva functii de baza .

Administratorul poate sa editeze meniul , sa stearga elemente din meniu si sa genereze anumite rapoarte care il ajuta sa vada mai bine modul in care functioneaza afacerea si sa poata inova sistemul de livrari pentru a produce mai multe resurse .

Clientul poate sa vada meniul , poate sa adauge produse in cos , poate sa filtreze produsele in functie de preferinte si poate sa plaseze comanda.

Angajatul este informat cand o comanda ajunge la el , pentru a o putea pregati .

De asemenea pentru fiecare comanda se va afisa un box cu rol de confirmare si un bon fiscal care arata data si ora la care a fost plasata comanda si produsele .

Proiectul permite utilizatorilor sa se inregistreze sub forma de client , administrator sau angajat , iar mai apoi sa se logheze in platforma .

Obiective secundare :

Analiza problemei – stabilirea cerintelor la care programul trebuie sa raspunda.

Modelare – imaginarea modelului conceptual pentru programul ce urmeaza a fi dezvoltat.

Scenarii posibile – analiza eventualelor probleme si gestionarea lor/

Cazuri de utilizare – determinarea cazurilor pentru care utilizatorul doreste sa foloseasca programul create/

Proiectare – definirea structurilor de date necesare , pe care urmeaza sa le implementam , a tabelelor si a claselor.

Implementare – descrierea claselor si a metodelor definite

Testare – generarea fisierelor (rapoarte) pentru a putea vedea corectitudinea functionarii aplicatiei.

2.Analiza problemei

1)Cerinte functionale

a) Aplicatia trebuie sa permita utilizatorului sa introduca datele de logare

b) Aplicatia trebuie sa permita crearea de clienti noi

c) Aplicatia trebuie sa permita filtrarea produselor din meniu

d) Aplicatia trebuie sa permita angajatului sa vada comenzile plasate

e) Aplicatia trebuie sa importe meniul din fisierul .csv

f) Aplicatia trebuie sa verifice input-urile

2)Cerinte non-functionale

a) Aplicatia trebuie sa fie usor de folosit

b) Aplicatia trebuie sa aiba o interfata interactiva , cu un design placut

3.Proiectare

Pentru a respecta conceptele OOP , fiecare clasa implementata este cat mai specifica , iar proiectul este impartin in doua parti : front-end si back-end.

a)Pachetele folosite:

-business

-dao

-model

-presentation

b)Clasele folosite:

- in business

-DeliveryService

-Order

- in dao

-FileWriter

-Serializer

-UsersWrite

- model

-BaseProduct

-CompositeProduct

-MenuItem

-User

- presentation

-AdminView

-ClientView

-FilterView

-LoginView

-RegisterView

-ReportsView

-Controller

-Main

Fiecare clasa are utilitatea ei :

Clasa DeliveryService:

In aceasta clasa se gaseste tot procesul din spatele front-end ului .

O sa arat cateva metode folosite.

Aceasta metoda importa meniul din fisierul „products.csv”.

public Set*<*MenuItem*>* createMenuSet*() {* // menuItemSet=null;  
 if *(*menuItemSet == null*) {* FileWrite fileWriter = new FileWrite*()*;  
 menuItemSet = fileWriter.importMenu*(*"products.csv"*)*;  
 new Serializer*()*.serializeMenu*(*menuItemSet*)*;  
 *}* return menuItemSet;  
*}*

Aceasta metoda ajuta la crearea tabelului de pe ecran.

public String*[][]* getProductData*() {* menuItemSet = new Serializer*()*.deserializeMenu*()*;  
 String*[][]* result = new String*[*menuItemSet.size*()][*7*]*;  
 int idx = 0;  
 for *(*MenuItem currentItem : menuItemSet*) {* result*[*idx*][*0*]* = currentItem.getTitle*()*;  
 result*[*idx*][*1*]* = Double.*toString(*currentItem.getRating*())*;  
 result*[*idx*][*2*]* = Double.*toString(*currentItem.getCalories*())*;  
 result*[*idx*][*3*]* = Double.*toString(*currentItem.getProtein*())*;  
 result*[*idx*][*4*]* = Double.*toString(*currentItem.getFat*())*;  
 result*[*idx*][*5*]* = Double.*toString(*currentItem.getSodium*())*;  
 result*[*idx*][*6*]* = Double.*toString(*currentItem.getPrice*())*;  
 idx++;  
 *}* return result;  
*}*

Urmatoarea metoda ajuta la adaugarea , editarea si stergerea produselor din tabel , aceste lucruri fiind facute de administrator .

public void addNewBaseItem*(*String title, double rating, double calories, double protein, double fat, double sodium, double price*)* throws IOException *{* MenuItem menuItem = new MenuItem*(*title, rating, calories, protein, fat, sodium, price*)*;  
 boolean checkTitle = true;  
 for *(*MenuItem currentItem : menuItemSet*) {* if *(*currentItem.getTitle*()*.equals*(*title*)) {* checkTitle = false;  
 break;  
 *}  
 }* if *(*checkTitle == false*) {* JOptionPane.*showMessageDialog(*null, "Title is taken!"*)*;  
 *}* else *{* /\*String File = ("src\\products.csv");  
 FileWriter fw = new FileWriter(File, true);  
 BufferedWriter bw = new BufferedWriter(fw);  
 PrintWriter pw = new PrintWriter(bw);  
 pw.println(title + "," + rating + "," + calories + "," + protein + "," + fat + "," + sodium + "," + price);  
 pw.flush();  
 pw.close();\*/  
 JOptionPane.*showMessageDialog(*null, "Food added!"*)*;  
 menuItemSet.add*(*menuItem*)*;  
 new Serializer*()*.serializeMenu*(*menuItemSet*)*;  
 *}  
}*public void deleteItem*(*String title*) {* for *(*MenuItem currentItem : menuItemSet*) {* if *(*currentItem.getTitle*()*.equals*(*title*)) {* menuItemSet.remove*(*currentItem*)*;  
 break;  
 *}  
 }* new Serializer*()*.serializeMenu*(*menuItemSet*)*;  
*}*

public void editItem*(*String title, double rating, double calories, double protein, double fat, double sodium, double price*) {* for *(*MenuItem currentItem : menuItemSet*) {* if *(*currentItem.getTitle*()*.equals*(*title*)) {* currentItem.setRating*(*rating*)*;  
 currentItem.setCalories*(*calories*)*;  
 currentItem.setProtein*(*protein*)*;  
 currentItem.setFat*(*fat*)*;  
 currentItem.setSodium*(*sodium*)*;  
 currentItem.setPrice*(*price*)*;  
 break;  
 *}  
 }* for *(*MenuItem currentItem : menuItemSet*) {* System.*out*.println*(*"\n" + currentItem*)*;  
 *}* new Serializer*()*.serializeMenu*(*menuItemSet*)*;  
*}*

Dupa parerea mea , cea mai complexa metoda din tot proiectul este metoda de filtrare a produselor , deoarece trebuie verificat fiecare camp daca este completat , iar mai apoi verificat fiecare element din tabel cu fiecare camp :

filterSet.clear*()*;  
int countMust = 0;  
if *(*!title.equals*(*""*)) {* countMust++;  
*}*if *(*!minRating.equals*(*""*)) {* countMust++;  
*}*if *(*!maxRating.equals*(*""*)) {* countMust++;  
*}*if *(*!minCalories.equals*(*""*)) {* countMust++;  
*}*if *(*!maxCalories.equals*(*""*)) {* countMust++;  
*}*if *(*!minProtein.equals*(*""*)) {* countMust++;  
*}*if *(*!maxProtein.equals*(*""*)) {* countMust++;  
*}*if *(*!minFat.equals*(*""*)) {* countMust++;  
*}*if *(*!maxFat.equals*(*""*)) {* countMust++;  
*}*if *(*!minSodium.equals*(*""*)) {* countMust++;  
*}*if *(*!maxSodium.equals*(*""*)) {* countMust++;  
*}*if *(*!minPrice.equals*(*""*)) {* countMust++;  
*}*if *(*!maxPrice.equals*(*""*)) {* countMust++;  
*}*

Aceste if- uri verifica cate casute din tabel sunt completate , pentru a putea vedea de cate comparari avem nevoie , dupa parcurgem toate elementele din tabel si cate elemente din lista au exact numarul de comparari facute egale cu numarul de casute completate , iar mai apoi adaugam aceste elemente intr o lista .

int count=0;  
if *(*!title.equals*(*""*)) {* if *(*currentItem.getTitle*()*.equals*(*title*)) {* count++;  
 *}  
}*if *(*!minRating.equals*(*""*)) {* minRatingD = Double.*parseDouble(*minRating*)*;  
 if *(*currentItem.getRating*()* >= minRatingD*) {* count++;  
 *}  
}*if *(*!maxRating.equals*(*""*)) {* maxRatingD = Double.*parseDouble(*maxRating*)*;  
 if *(*currentItem.getRating*()* <= maxRatingD*) {* count++;  
 *}  
}*if *(*!minCalories.equals*(*""*)) {* minCaloriesD = Double.*parseDouble(*minCalories*)*;  
 if *(*currentItem.getCalories*()* >= minCaloriesD*) {* count++;  
 *}  
}*if *(*!maxCalories.equals*(*""*)) {* maxCaloriesD = Double.*parseDouble(*maxCalories*)*;  
 if *(*currentItem.getCalories*()* <= maxCaloriesD*) {* count++;  
 *}  
}*if *(*!minProtein.equals*(*""*)) {* minProteinD = Double.*parseDouble(*minProtein*)*;  
 if *(*currentItem.getProtein*()* >= minProteinD*) {* count++;  
 *}  
}*if *(*!maxProtein.equals*(*""*)) {* maxProteinD = Double.*parseDouble(*maxProtein*)*;  
 if *(*currentItem.getProtein*()* <= maxProteinD*) {* count++;  
 *}  
}*if *(*!minFat.equals*(*""*)) {* minFatD = Double.*parseDouble(*minFat*)*;  
 if *(*currentItem.getFat*()* >= minFatD*) {* count++;  
 *}  
}*if *(*!maxFat.equals*(*""*)) {* maxFatD = Double.*parseDouble(*maxFat*)*;  
 if *(*currentItem.getFat*()* <= maxFatD*) {* count++;  
 *}  
}*if *(*!minSodium.equals*(*""*)) {* minSodiumD = Double.*parseDouble(*minSodium*)*;  
 if *(*currentItem.getSodium*()* >= minSodiumD*) {* count++;  
 *}  
}*if *(*!maxSodium.equals*(*""*)) {* maxSodiumD = Double.*parseDouble(*maxSodium*)*;  
 if *(*currentItem.getSodium*()* <= maxSodiumD*) {* count++;  
 *}  
}*if *(*!minPrice.equals*(*""*)) {* minPriceD = Double.*parseDouble(*minPrice*)*;  
 if *(*currentItem.getPrice*()* >= minPriceD*) {* count++;  
 *}  
}*if *(*!maxPrice.equals*(*""*)) {* maxPriceD = Double.*parseDouble(*maxPrice*)*;  
 if *(*currentItem.getPrice*()* <= maxPriceD*) {* count++;  
 *}  
}*if *(*count == countMust*) {* filterSet.add*(*currentItem*)*;  
*}*

La final if-ul este cel mai important , deoarece acolo se produce adaugarea in lista.

Clasa Order:

package business;  
  
import java.io.Serializable;  
import java.time.LocalDateTime;  
import java.time.format.DateTimeFormatter;  
import java.time.format.DateTimeFormatterBuilder;  
import java.util.Objects;  
  
public class Order implements Serializable *{* private int orderId;  
 private int clientId;  
 private LocalDateTime orderTime;  
  
  
 public Order*(*int orderId, int clientId*){* this.orderId=orderId;  
 this.clientId=clientId;  
 this.orderTime=LocalDateTime.*now()*;  
 *}* public int getOrderId*() {* return orderId;  
 *}* public void setOrderId*(*int orderId*) {* this.orderId = orderId;  
 *}* public int getClientId*() {* return clientId;  
 *}* public void setClientId*(*int clientId*) {* this.clientId = clientId;  
 *}* public LocalDateTime getOrderTime*() {* return orderTime;  
 *}* public void setOrderTime*(*LocalDateTime orderTime*) {* this.orderTime = orderTime;  
 *}* @Override  
 public int hashCode*(){* return Objects.*hash(*orderId*)*;  
 *}* @Override  
 public String toString*() {* return "\nOrder{" +  
 "orderId= " + orderId +  
 ", clientId= " + clientId +  
 ", orderTime= " + orderTime.format*(*DateTimeFormatter.*ofPattern(*"dd/MM/yyyy HH:mm"*))* +  
 '}';  
 *}* @Override  
 public boolean equals*(*Object obj*){* if*(*this==obj*)*return true;  
 if*(*obj==null || getClass*()*!=obj.getClass*())* return false;  
 Order order=*(*Order*)* obj;  
 return orderId==order.getOrderId*()*;  
 *}  
}*

Avem nevoie de un orderID, clientId si un orderTime , pentru a putea vedea cand si de cine a fost realizata comanda .

Clasa User :

package model;  
  
import java.io.Serializable;  
  
public class User implements Serializable *{* private int id;  
 private String username;  
 private String password;  
  
 //0 for admin , 1 for employee , 2 for client  
 private int type;  
  
 public User*(*int id, String username, String password, int type*) {* this.id = id;  
 this.username = username;  
 this.password = password;  
 this.type = type;  
 *}* public int getId*() {* return id;  
 *}* public void setId*(*int id*) {* this.id = id;  
 *}* public String getUsername*() {* return username;  
 *}* public void setUsername*(*String username*) {* this.username = username;  
 *}* public String getPassword*() {* return password;  
 *}* public void setPassword*(*String password*) {* this.password = password;  
 *}* public int getType*() {* return type;  
 *}* public void setType*(*int type*) {* this.type = type;  
 *}  
}*

In aceasta clasa vedem exact din ce este alcatuit un user , id specific fiecarui user , username-ul si parola , alese de utilizator si un type ( admin,client,angajat)

Metoda de logare si inregistrare este urmatoarea :

public void readUsers*()* throws IOException *{* BufferedReader reader = null;  
 String File = *(*"src\\members.csv"*)*;  
 String line = "";  
 try *{* reader = new BufferedReader*(*new FileReader*(*File*))*;  
 String headerLine = reader.readLine*()*;  
 while *((*line = reader.readLine*())* != null*) {* String*[]* tkn = line.split*(*","*)*;  
 userSet.add*(*new User*(*Integer.*parseInt(*tkn*[*0*])*, tkn*[*1*]*, tkn*[*2*]*, Integer.*parseInt(*tkn*[*3*])))*;  
 *}  
 }* catch *(*FileNotFoundException e*) {* e.printStackTrace*()*;  
 *}* catch *(*IOException e*) {* e.printStackTrace*()*;  
 *}* finally *{* reader.close*()*;  
 *}  
}*public void writeUsers*(*String username, String password, int type*)* throws IOException *{* readUsers*()*;  
 boolean checkUsername = true;  
 for *(*User currentUser : userSet*) {* if *(*currentUser.getUsername*()*.equals*(*username*)) {* checkUsername = false;  
 *}  
 }* if *(*checkUsername == false*) {* JOptionPane.*showMessageDialog(*null, "Username is taken!"*)*;  
 *}* else *{* int id = userSet.size*()* + 1;  
 String File = *(*"src\\members.csv"*)*;  
 FileWriter fw = new FileWriter*(*File, true*)*;  
 BufferedWriter bw = new BufferedWriter*(*fw*)*;  
 PrintWriter pw = new PrintWriter*(*bw*)*;  
 pw.println*(*id + "," + username + "," + password + "," + type*)*;  
 pw.flush*()*;  
 pw.close*()*;  
 JOptionPane.*showMessageDialog(*null, "User created!"*)*;  
 *}  
}*public boolean find*(*String username, String password*) {* if *(*null == userSet*) {* throw new IllegalStateException*(*"user list is empty"*)*;  
 *}* return userSet.stream*()* .filter*(*u -> u.getUsername*()*.equals*(*username*))* .filter*(*u -> u.getPassword*()*.equals*(*password*))* .findFirst*()* .isPresent*()*;  
*}*

Cand un user apasa pe butonul register , username-ul se verifica cu toata lista , deoarece nu pot exista doi useri cu acelasi username , dupa care este creat contul , fiecare avand un type selectat anterior.

Logarea se face pe baza de username si parola , daca username-ul este gasit in fisier , iar parola corespunde cu cea din fisier , atunci metoda returneaza TRUE , iar user ul este lasat sa intre in aplicatie , altfel accesul la aplicatie este respins .

Conectivitatea aplicatiei (front si back) se face in clasa Controller , clasa in care sunt conectate butoanele interfetelor cu logica aplicatiei din DeliveryService.

public class Controller *{* private LoginView loginView;  
 private RegisterView registerView = new RegisterView*()*;  
 private ClientView clientView = new ClientView*()*;  
 private AdminView adminView = new AdminView*()*;  
 private FilterView filterView = new FilterView*()*;  
 private ReportsView reportsView=new ReportsView*()*;  
 private User user;  
 private Set*<*User*>* userSet;  
 private Set*<*model.products.MenuItem*>* menuItemSet;  
 private DeliveryService deliveryService;

Regasim toate view – urile conectate la backend.

Un exemplu de metoda din Controller:

class BtnAdminAddNewBaseProduct implements ActionListener *{* @Override  
 public void actionPerformed*(*ActionEvent e*) {* if *(*adminCheckBoxes*()* == false*) {* JOptionPane.*showMessageDialog(*null, "Some boxes are empty!"*)*;  
 *}* else *{* String title = adminView.getTextFieldTitle*()*;  
 double rating = Double.*parseDouble(*adminView.getTextFieldRating*())*;  
 double calories = Double.*parseDouble(*adminView.getTextFieldCalories*())*;  
 double proteins = Double.*parseDouble(*adminView.getTextFieldProtein*())*;  
 double fats = Double.*parseDouble(*adminView.getTextFieldFats*())*;  
 double sodium = Double.*parseDouble(*adminView.getTextFieldSodium*())*;  
 double price = Double.*parseDouble(*adminView.getTextPrice*())*;  
 try *{* deliveryService.addNewBaseItem*(*title, rating, calories, proteins, fats, sodium, price*)*;  
 //adminView.refreshTable();  
 *}* catch *(*IOException ex*) {* ex.printStackTrace*()*;  
 *}* adminView.refreshTable*()*;  
 *}  
 }  
}*

Aceasta actiune prezentata mai sus transmite parametrii preluati din frontend catre backend , unde se realizeaza adaugarea elementelor .

Urmatoarea actiune transmite parametrii din front si genereaza rapoartele specifice.

class BtnGOClients implements ActionListener *{* @Override  
 public void actionPerformed*(*ActionEvent e*) {* if*(*reportsView.getTextFieldClientsAndOrdersTimes*()*.isEmpty*()* || reportsView.getTextFieldClientsAndOrdersValue*()*.isEmpty*()){* JOptionPane.*showMessageDialog(*null, "Some boxes are empty!"*)*;  
 *}*else*{* int times=Integer.*parseInt(*reportsView.getTextFieldClientsAndOrdersTimes*())*;  
 int value=Integer.*parseInt(*reportsView.getTextFieldClientsAndOrdersValue*())*;  
 deliveryService.clientsAndOrders*(*times, value*)*;  
 JOptionPane.*showMessageDialog(*adminView, "Generated!"*)*;  
 *}  
 }  
}*

5.Rezultate

Rapoartele au forma aceasta:

The orders performed between 0:00 and 24:00 :  
Order{orderId= 1, clientId= 6, orderTime= 18/05/2022 21:10}  
Order{orderId= 2, clientId= 6, orderTime= 18/05/2022 21:15}  
Order{orderId= 3, clientId= 6, orderTime= 18/05/2022 21:15}  
Order{orderId= 4, clientId= 6, orderTime= 18/05/2022 22:57}

Menu items that were ordered more than 2:  
TITLE: Quick & Spicy Asian Pickles RATING: 5.0 CALORIES: 23.0 PROTEIN: 1.0 FAT: 0.0 SODIUM: 128.0 PRICE: 48.0 was ordered 2 times.  
TITLE: Fresh Corn Tortillas RATING: 3.75 CALORIES: 23.0 PROTEIN: 1.0 FAT: 2.0 SODIUM: 61.0 PRICE: 79.0 was ordered 2 times.

Users that have atleast 2 orders with value over 10.0:  
c has ordered 4 times.

Products that have been ordered in 2022-05-18:  
TITLE: ggg RATING: 1.0 CALORIES: 1.0 PROTEIN: 1.0 FAT: 1.0 SODIUM: 1.0 PRICE: 1.0 was ordered 1 times.  
TITLE: T RATING: 2.0 CALORIES: 22.0 PROTEIN: 2.0 FAT: 2.0 SODIUM: 2.0 PRICE: 2.0 was ordered 1 times.  
TITLE: Quick & Spicy Asian Pickles RATING: 5.0 CALORIES: 23.0 PROTEIN: 1.0 FAT: 0.0 SODIUM: 128.0 PRICE: 48.0 was ordered 2 times.  
TITLE: Fresh Corn Tortillas RATING: 3.75 CALORIES: 23.0 PROTEIN: 1.0 FAT: 2.0 SODIUM: 61.0 PRICE: 79.0 was ordered 2 times.  
TITLE: Ethiopian Spice Tea RATING: 5.0 CALORIES: 23.0 PROTEIN: 1.0 FAT: 0.0 SODIUM: 13.0 PRICE: 49.0 was ordered 1 times.

6.Concluzii

Tema de fata a prezentat elemente noi , in primul rand folosirea stream-urilor si definirea conditiilor . Precum si structuri de date noi (HashSet si HashMap) . Tema a fost una dificila , am invatat multe lucruri noi si pot spune ca mi-a placut sa lucrez la ea .

7.Bibliografie

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

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

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

<https://www.baeldung.com/java-serialization>