Documentatie

Tema 4

Blendea Adrian

1.Obiectivul temei

Proiectarea si implementarea unei aplicatii cu scopul folosirii acesteia ca un sistem de management pentru o companie de catering cu urmatoarele cerinte:

1. Requirements

Design and implement a food delivery management system for a catering company. The client can

order products from the company’s menu. The system should have three types of users that log in

using a username and a password: administrator, regular employee, and client.

The administrator can:

• Import the initial set of products which will populate the menu from a .csv file.

• Manage the products from the menu: add/delete/modify products and create new products

composed of several products from the menu (an example of composed product could be

named “daily menu 1” composed of a soup, a steak, a garnish, and a dessert).

• Generate reports about the performed orders considering the following criteria:

o time interval of the orders – a report should be generated with the orders performed

between a given start hour and a given end hour regardless the date.

o the products ordered more than a specified number of times so far.

o the clients that have ordered more than a specified number of times so far and the

value of the order was higher than a specified amount.

o the products ordered within a specified day with the number of times they have

been ordered.

The client can:

• Register and use the registered username and password to log in within the system.

• View the list of products from the menu.

• Search for products based on one or multiple criteria such as keyword (e.g., “soup”), rating,

number of calories/proteins/fats/sodium/price.

• Create an order consisting of several products – for each order the date and time will be

persisted and a bill will be generated that will list the ordered products and the total price

of the order.

The employee is notified each time a new order is performed by a client so that it can prepare the

delivery of the ordered dishes.

-recunoasterea claselor necesare implementarii si implementarea OOP a acestora;

-proiectarea unei interfete grafice user friendly si implementarea acesteia folosind java swing;

-preluarea datelor prin intermediul interfetei grafice create;

-crearea unei legaturi intre interfata si partea de calcul a programului folosind un patern MVC (model-view-controller)

- citirea si prelucrarea datelor dintr un fisier csv

- lucru cu serializarea , pastrarea datelor dupa terminarea programului , refolosirea acestora prin deserializarea din fisierele scrise inainte de inchiderea programului

- crearea unui sistem de login , care recunoaste tipul utilizatorului cautandul intr o baze de date si oferindu-i accesul necesar , conform rolului detectat in baza de date

- crearea unui sistem de sign in , care permite crearea unor conturi de utilizator ( aceste conturi pot avea doar gradul de client) conturile de admin si employee sunt create din interiorul programului pentru a ne asigura ca un utilizator strain nu poate avea permisiuni de alterare a datelor programului

2. Implementare

Pachete: avem 3 pachete Model View si Controller

Clasele din pachetul Model sunt standard, conform cerintelor iar cele din pachetul View(cele 5 interfete) sunt realizate folosind window builder pentru estetica, si creeand manual listener e pentru butoanele continute.

Clasele din model:

User -este folosit pentru a reprezenta userii pe caz general , acestia sunt particularizati prin mostenirea acestei clase.

Atribute:

private static int *counter*=0;  
private int id;  
private String username;  
private String password;  
private UserType type;

Admin – mosteneste clasa user , toate obiectele ale acestei clase sunt instantiate avand user type Admin,clasa contine doar un constructor

Client - mosteneste clasa user , toate obiectele ale acestei clase sunt instantiate avand user type Client,clasa contine doar un constructor

Employee - mosteneste clasa user , toate obiectele ale acestei clase sunt instantiate avand user type Employee,clasa contine doar un constructor

Login in aplicatie se va face in functie de tipul utilizatorului, fiecare tip de utilizator avand diferite permisiuni si operatii pe care le poate efectua .

MenuItem – este folosit pentru a representa pe caz general un produs oferit de firma de catering,acesta este particularizat prin mostenire.

BaseProduct - este folosit pentru a reprezenta un produs unic oferit de firma, aceste produse se citesc dintr un fisier csv folosind metoda readProducts din controlerul principal:

public void readProducts() {  
  
 String file = "src\\products.csv";  
 BufferedReader reader = null;  
 String line = "";  
  
 try {  
 reader = new BufferedReader(new FileReader(file));  
 boolean firstLine=true;  
 while((line = reader.readLine()) != null) {  
 if(!firstLine) {  
 String[] row = line.split(",");  
 BaseProduct b=new BaseProduct (row[0].toString(),Float.*parseFloat*(row[1].toString()),Integer.*parseInt*(row[2].toString()),Integer.*parseInt*(row[3].toString()),Integer.*parseInt*(row[4].toString()),Integer.*parseInt*(row[5].toString()),Integer.*parseInt*(row[6].toString()));  
 this.products.add(b);  
 }  
 firstLine=false;  
 }  
 }  
 catch(Exception e) {  
 e.printStackTrace();  
 }  
 finally {  
 try {  
 reader.close();  
 } catch (IOException e) {  
 // *TODO Auto-generated catch block* e.printStackTrace();  
 }  
  
  
  
  
 }}

Atribute:

private float rating;  
private int calories;  
private int proteins;  
private int fats;  
private int sodium;  
private int price;

CompositeProduct - este alcatuit din mai multe base producturi .

Atribute :

private ArrayList<BaseProduct> products;  
private float averageRating;  
private int totalCalories;  
private int totalProteins;  
private int totalFats;  
private int totalSodium;  
private int totalPrice;

Atributele unui produs compus este alcatuit din suma atributelor base producturilor componente, exceptand ratingul care pentru care se efectueaza o media aritmetica a produselor componente.

Order – este reprezentarea programului a unei comenzi

Atribute :

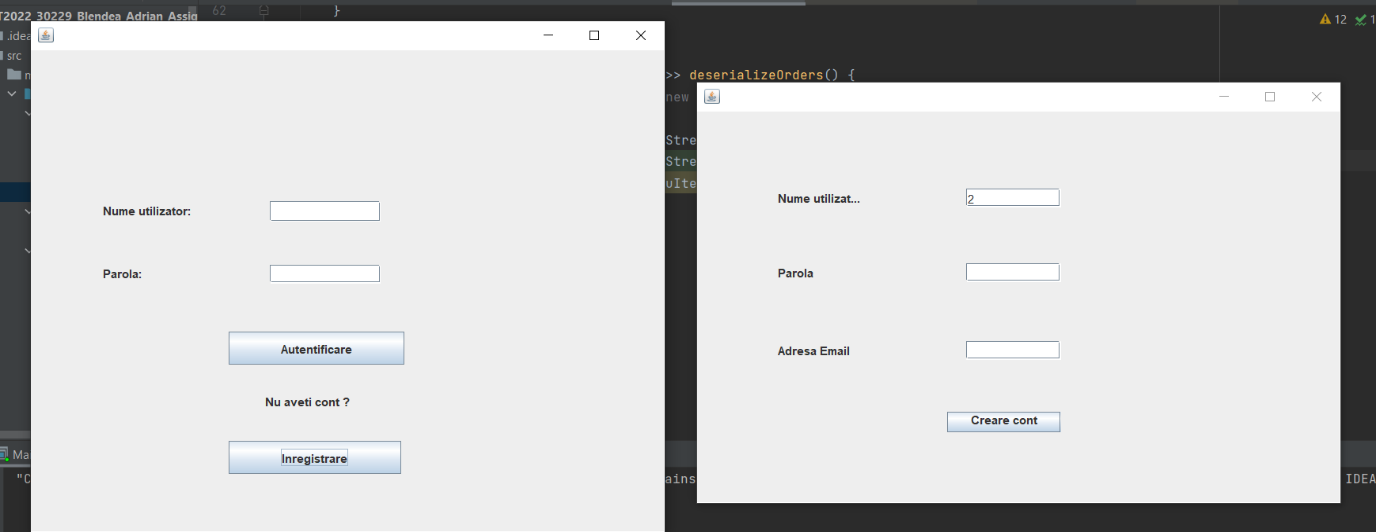
private static int *counter*=0;  
private String orderId;  
private int clientId;  
private LocalDateTime orderDate;  
private int orderPrice;

Pentru a ne asigura ca fiecare order are un orderId diferit acesta este compusa prin concatenarea clientul id ului al orderPriceului dar si al counterului care este un camp incrementat pentru fiecare comanda noua asadar nicio comanda nu poate sa aiba acelasi order Id

public Order (int clientId,LocalDateTime orderDate,int orderPrice) {  
 this.clientId = clientId;  
 this.orderDate = orderDate;  
 this.orderPrice = orderPrice;  
 this.orderId = Integer.*toString*(clientId) + Integer.*toString*(*counter*) +Integer.*toString*(orderPrice);  
 *counter*++;  
}

Clasele relevante despre care vom vorbi mai mult in aceasta documentatie sunt cele din Controller:

Controller (controlerul pentru interfata de start):



Aceasta clasa controleaza autentificarea si inregistrarea utilizatorului , cele 2 viewuri:

First View cu atributele:

private JPanel contentPane;  
private JLabel userLabel;  
private JTextField userText;  
private JTextField passwordText;  
private JLabel passwordLabel;  
private JButton loginButton;  
private JLabel contLabel;  
private JButton inregistrareButton;

si Register View cu atributele:

private JPanel contentPane;  
private JTextField username;  
private JTextField password;  
private JTextField email;  
private JLabel usernameLabel;  
private JLabel passwordLabel;  
private JLabel emailLabel;  
private JButton createButton;

Serializator: aceasta clasa de control se ocupa cu pastrarea unor date importante pentru program la inchiderea acestuia. Este folosit pentru stoca prin serializare baza de date a utilizatoriilor, istoricul comenziilor ale firmei de catering dar si lista produselor din stoc si a gestiona toate modificarile asupra acestor date.Metodele acestei sunt apelate de alte controlere pentru a obtine rezultatul dorit.

public static ArrayList<User> deserializeClients() {  
  
 try {  
 FileInputStream fileIn = new FileInputStream("database.ser");  
 ObjectInputStream in = new ObjectInputStream(fileIn);  
 ArrayList<User> userList;  
 userList = (ArrayList<User>) in.readObject();  
 in.close();  
 fileIn.close();  
 return userList;  
  
 } catch (IOException i) {  
 i.printStackTrace();  
 return null;  
 } catch (ClassNotFoundException c) {  
 // System.out.println("Employee class not found");  
 c.printStackTrace();  
 return null;  
 }  
}  
  
  
public static void serializeClients(ArrayList<User> userList) {  
  
 try {  
 FileOutputStream fileOut =  
 new FileOutputStream("database.ser");  
 ObjectOutputStream out = new ObjectOutputStream(fileOut);  
 out.writeObject(userList);  
 out.close();  
 fileOut.close();  
 // System.out.printf("Serialized data is saved in /tmp/employee.ser");  
 } catch (IOException i) {  
 i.printStackTrace();  
 }  
}  
public static void serializeOrders(HashMap<Order,ArrayList<MenuItem>> orders) {  
  
 try {  
 FileOutputStream fileOut =  
 new FileOutputStream("orderHistory.ser");  
 ObjectOutputStream out = new ObjectOutputStream(fileOut);  
 out.writeObject(orders);  
 out.close();  
 fileOut.close();  
  
 } catch (IOException i) {  
 i.printStackTrace();  
 }  
}  
  
public static HashMap<Order,ArrayList<MenuItem>> deserializeOrders() {  
 HashMap<Order,ArrayList<MenuItem>> orders=new HashMap<Order,ArrayList<MenuItem>>();  
 try {  
 FileInputStream fileIn = new FileInputStream("orderHistory.ser");  
 ObjectInputStream in = new ObjectInputStream(fileIn);  
 orders = (HashMap<Order, ArrayList<MenuItem>>) in.readObject();  
 if(orders!=null)  
 return orders;  
 in.close();  
 fileIn.close();  
 } catch (IOException i) {  
 i.printStackTrace();  
 return null;  
 } catch (ClassNotFoundException c) {  
  
 c.printStackTrace();  
 return null;  
 }  
 return null;  
}  
  
public static void serializeProducts(HashSet<MenuItem> products) {  
  
 try {  
 FileOutputStream fileOut =  
 new FileOutputStream("productsDatabase.ser");  
 ObjectOutputStream out = new ObjectOutputStream(fileOut);  
 out.writeObject(products);  
 out.close();  
 fileOut.close();  
  
 } catch (IOException i) {  
 i.printStackTrace();  
 }  
}  
  
public static HashSet<MenuItem> deserializeProducts() {  
  
 try {  
 FileInputStream fileIn = new FileInputStream("productsDatabase.ser");  
 ObjectInputStream in = new ObjectInputStream(fileIn);  
 *productsDatabase* = (HashSet<MenuItem>) in.readObject();  
 in.close();  
 fileIn.close();  
 return *productsDatabase*;  
 } catch (IOException i) {  
 i.printStackTrace();  
 return null;  
 } catch (ClassNotFoundException c) {  
  
 c.printStackTrace();  
 return null;  
 }  
}

AdminController

Acesta controleaza operatiile ce pot fii efectuate de un admin:

The administrator can:

• Import the initial set of products which will populate the menu from a .csv file.

• Manage the products from the menu: add/delete/modify products and create new products

composed of several products from the menu (an example of composed product could be

named “daily menu 1” composed of a soup, a steak, a garnish, and a dessert).

• Generate reports about the performed orders considering the following criteria:

o time interval of the orders – a report should be generated with the orders performed

between a given start hour and a given end hour regardless the date.

o the products ordered more than a specified number of times so far.

o the clients that have ordered more than a specified number of times so far and the

value of the order was higher than a specified amount.

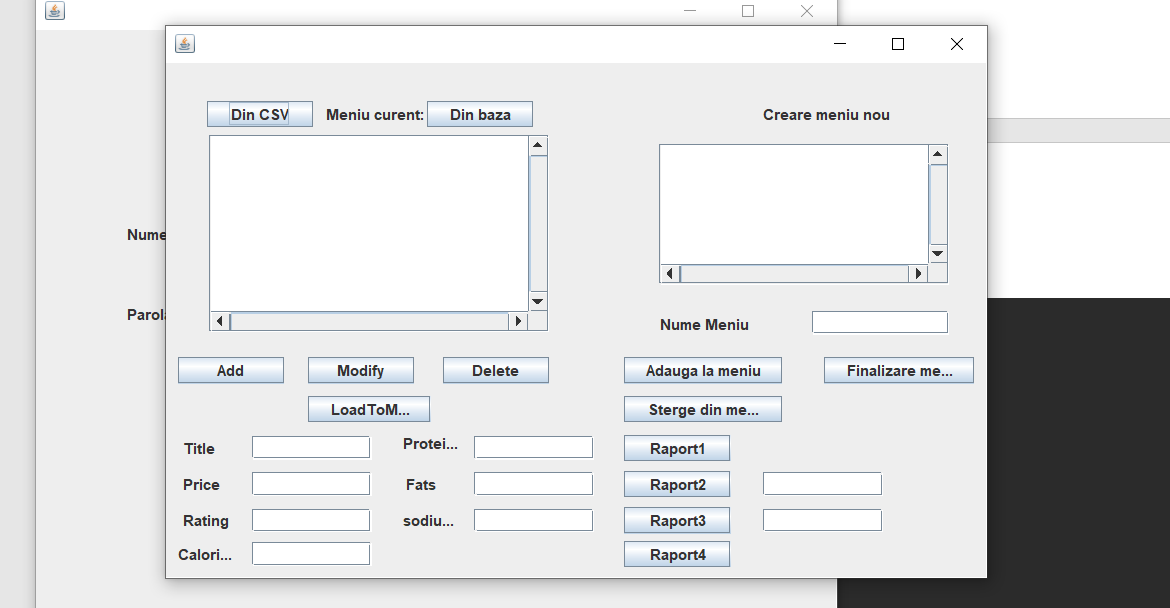
o the products ordered within a specified day with the number of times they have

been ordered.

Aceste operatii sunt implementate din interfata AdmininstratorView ce este controlata de acest control si care are urmatoarele atribute:

private JPanel contentPane;  
private JList menuList;  
private JList newMenuList;  
private JScrollPane menuListPane;  
private JScrollPane newMenuListPane;  
private JLabel newMeniuLabel;  
private JTextField numeTextField;  
private JLabel nameLabel;  
private JButton adaugaButton;  
private JButton finishButton;  
private JButton bazaButton;  
private JButton importCsvButton;  
private DefaultListModel<MenuItem> menu;  
private DefaultListModel<MenuItem> compositeProduct;  
private JButton stergeButton;  
private JButton addBaseButton;  
private JButton modifyBaseButton;  
private JButton deleteBaseButton;  
private JButton loadToModify;  
private JTextField titleTextField;  
private JTextField priceTextField;  
private JTextField ratingTextField;  
private JTextField caloriesTextField;  
private JTextField proteinsTextField;  
private JTextField fatsTextField;  
private JTextField sodiumTextField;  
private JLabel titluLAbel;  
private JLabel priceLabel;  
private JLabel ratingLabel;  
private JLabel caloriesLabel;  
private JLabel proteinsLabel;  
private JLabel fatsLabel;  
private JLabel Sodium;  
private JButton raport1Button;  
private JButton raport2Button;  
private JButton raport3Button;  
private JButton raport4Button;  
private JTextField infoRaportTextField;  
private JTextField infoRaportTextField2;

Numite sugestiv pentru a reprezenta elemente din View



ClientController

Acesta controleaza operatiile ce pot fii efectuate de un client:

The client can:

• Register and use the registered username and password to log in within the system.

• View the list of products from the menu.

• Search for products based on one or multiple criteria such as keyword (e.g., “soup”), rating,

number of calories/proteins/fats/sodium/price.

• Create an order consisting of several products – for each order the date and time will be

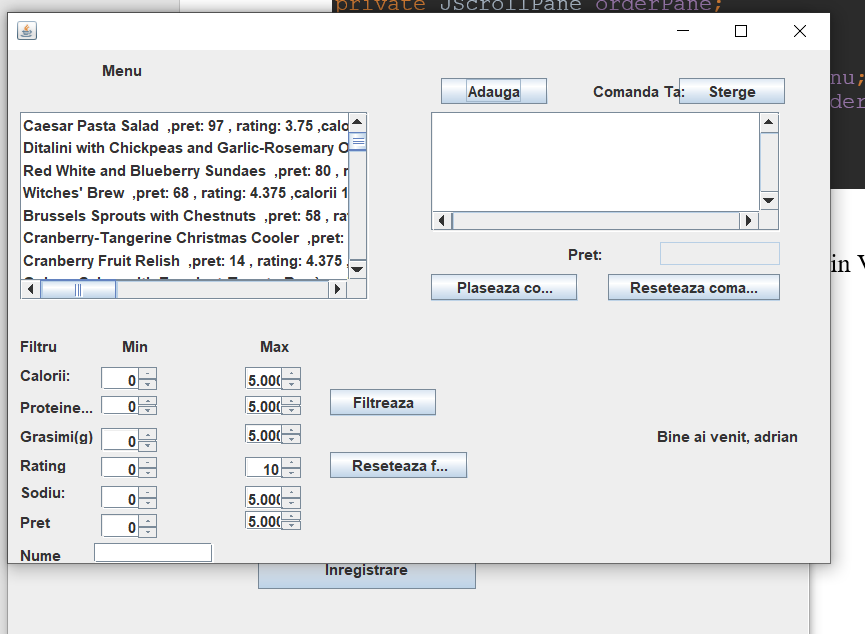
persisted and a bill will be generated that will list the ordered products and the total price

of the order.

Aceste operatii sunt implementate din interfata ClientView ce este controlata de acest control si care are urmatoarele atribute:

private JPanel comandaList;  
private JTextField priceText;  
private JLabel menuLabel;  
private JLabel orderLabel;  
private JLabel priceLabel;  
private JLabel filtersLabel;  
private JLabel minLabel;  
private JLabel maxLabel;  
private JLabel caloriiLabel;  
private JLabel grasimiLabel;  
private JLabel proteineLabel;  
private JLabel sodiuLabel;  
private JLabel pretLabel;  
private JTextField numeTextField;  
private JSpinner minCalorii;  
private JSpinner maxCalorii;  
private JSpinner maxProteins;  
private JSpinner minFats;  
private JSpinner maxFats;  
private JLabel ratingLabel;  
private JLabel numeLabel;  
private JSpinner maxSodium;  
private JSpinner minPrice;  
private JSpinner maxPrice;  
private JButton plaseazaButton;  
private JButton filterButton;  
private JButton resetFiltersButton;  
private JButton resetButton;  
private JSpinner minProteins;  
private JSpinner minRating;  
private JSpinner minSodium;  
private JSpinner maxRating;  
private JList<MenuItem> orderList;  
private JScrollPane orderPane;  
private JList<MenuItem>menuList;  
private JScrollPane menuPane;  
private DefaultListModel<MenuItem> menu;  
private DefaultListModel<MenuItem> order;  
private JButton addButton;  
private JButton stergeButton;  
private JLabel welcomeLabel;

Numite sugestiv pentru a reprezenta elemente din View



3.Concluzii

Aceasta tema, respectiv tema cu numarul 4, a fost foarte utila in recapitularea cunostintelor de java dobandite in semestrul anterior dar si asimilarea unor noi cunostinte cum ar fi folosirea lambda expresions, lucrul cu fisiere csv , dar si in dezvoltarea studentului in scrierea unui cod organizat , deoarece tema este una destul de complexa ce necesita un nivel ridicat de atentie si organizare.

Viitoarele dezvoltari ale acestei teme ar trebui sa includa operatiile ce pot fi efectuate de un employee, sa introduca niste conditii aditionale la colectarea datelor dar si la logare dar si sa rezolve unele erori ce pot aparea la serializarea si deserializarea multipla a datelor.

De asemenea testarea amanuntita a tuturor operatiilor ce se efectueaza deoarece , la nivelul unui cod asa de lung , pot aparea anumite scapari, testarea prin exemplu a unui cod nu poate demonstra functionalitatea completa al acestuia, ci doar disfunctionalitatile lui , asadar o testare amanuntita ar putea duce la gasirea si rezolvarea eventualelor probleme ce se regasesc in cod.