Fundamental Programming Techniques 2021

Assignment 4

Food delivery management system

Java application

Chereji Iulia-Adela

Group 30424-1

CONTENTS

1. ASSIGNMENT OBJECTIVE

1.1 Main objective

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 employee is notified each time a new order is performed by a client so that it can prepare the delivery of the ordered dishes.

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 (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: 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, the products ordered more than a specified number of times so far, the clients that have ordered more than a specified number of times and the value of the order was higher than a specified amount, 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

1.2 Secondary objectives

1.2.1 Analyze the problem and identify requirements

The modeling of the problem will be performed (i.e. identify the functional and non-functional requirements, identify the scenarios, detail and analyze the use cases). – Ch. 2

1.2.2 Design the application

The OOP design decisions will be presented (i.e. UML diagrams, data structures, class design, relationships, packages, interfaces, user interfaces). –Ch. 3

1.2.3 Implement the application

The implementation of the classes (i.e. fields and important methods) and of the graphical user interface will be presented. –Ch. 4

2. PROBLEM ANALYSIS

2.1 Functional requirements

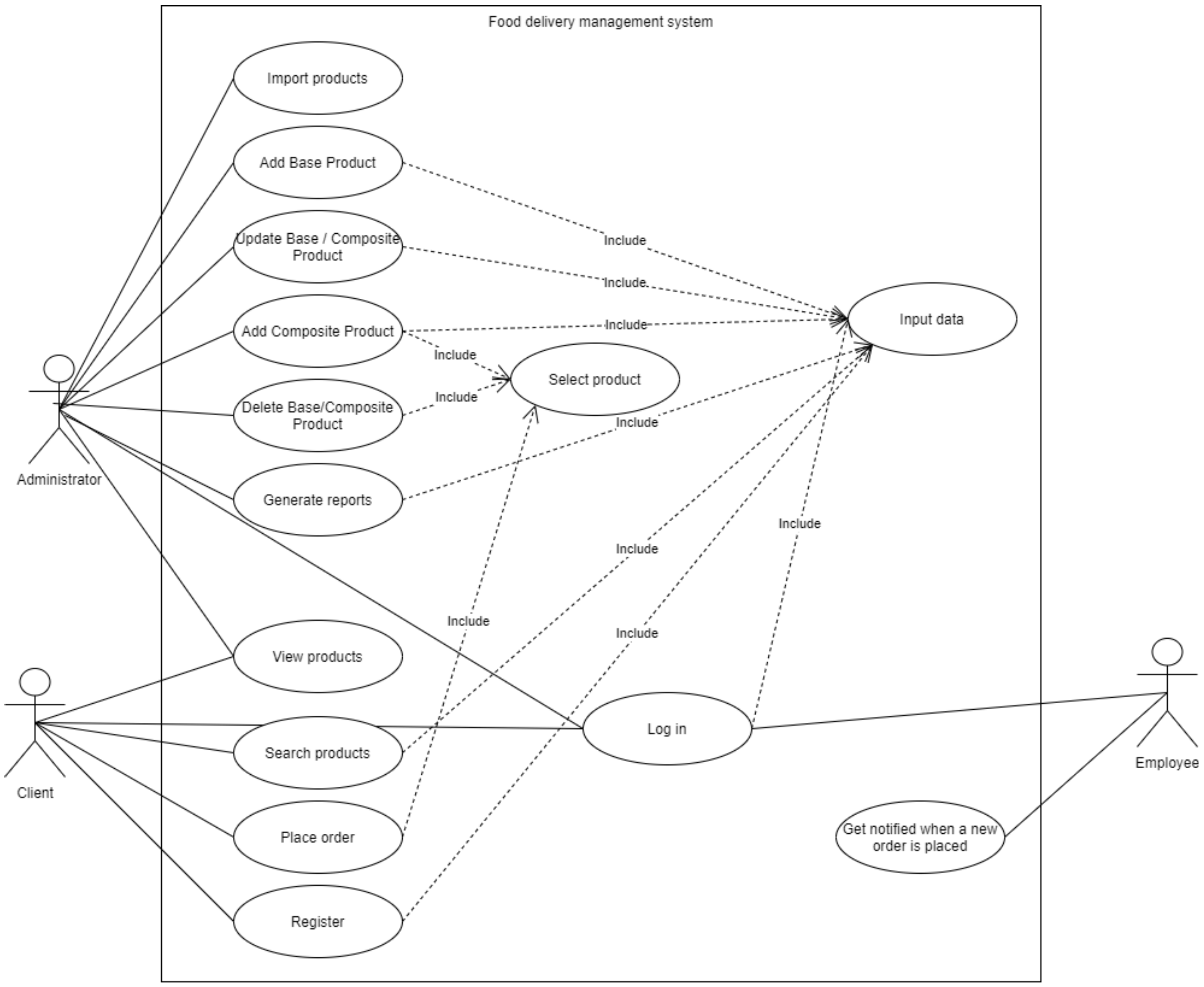
- The application should allow users to log in using a verified username and a password.
- The application should allow administrators to import base products from a .csv file using lambda expressions and stream processing, and eliminate duplicates.
- The application should allow administrators to create composite products containing more base products.
- The application should allow administrators to add/delete/modify base/composite products and check the provided input.
- The application should allow administrators to generate reports as .txt files based on some criteria using lambda expressions and stream processing.
- The application should allow clients to register by checking the provided input.
- The application should allow clients to see the products from the menu in a table.
- The application should allow clients to search for products based on some criteria using lambda expressions and stream processing.
- The application should allow clients to place an order composed of one or more products.
- The application should allow employees to be notified when a new order is placed, using the Observable-Observer design pattern.
- The application should use Serialization to maintain the information about users, products and orders.
- The application should be documented using JavaDoc files.
- The application should have five windows for log in, register, administrator operations, client operations and employee operations.
- The application should use the layered architectural pattern.
- The application should create a bill for each order as a .txt file.

2.2 Non-functional requirements

- The application should be intuitive and easy to use by the user.
- The application should display an error message if one of the inputs is incorrect or in the wrong format.

- The application should display an error message if the administrator wants to edit or delete a product without selecting it.
- The application should display an error message if the log in information is incorrect.

2.3 Use cases



2.3.1 Use Case: Import products

Primary Actor: administrator

Main Success Scenario:

1. The administrator clicks on the “Import products” button in the graphical user interface and sees the base products in a table.

2.3.2 Use Case: Add Base Products

Primary Actor: administrator

Main Success Scenario:

1. The administrator inputs data in the graphical user interface.
2. The administrator clicks on the “Add base product” button in the graphical user interface.
3. The product is added.

Alternative Sequence 1: Wrong input

1. The administrator inputs data in the graphical user interface.
2. The administrator clicks on the “Add base product” button in the graphical user interface.
3. The application displays an error message.
4. The scenario returns to step 1.

2.3.3 Use Case: Update Base/Composite Product

Primary Actor: administrator

Main Success Scenario:

1. The administrator clicks on a product from one of the two tables.
2. The administrator clicks on the “Update” button.
3. The administrator inputs data in the graphical user interface.
4. The administrator clicks on the “Save” button.
5. The product is updated

Alternative Sequence: Product not selected

1. The administrator clicks on the “Update” button in the graphical user interface.
2. The application displays an error message.
3. The scenario returns to step 1.

Alternative Sequence 2: Wrong input

1. The administrator clicks on a product from one of the two tables.
2. The administrator clicks on the “Update” button.
3. The administrator inputs data in the graphical user interface.
4. The administrator clicks on the “Save” button.
5. The application displays an error message.
6. The scenario returns to step 1.

2.3.4 Use Case: Add composite product

Primary Actor: administrator

Main Success Scenario:

1. The administrator clicks on a product from the base products table and then clicks on the “Add product to composite” button.
2. The step 1 repeats multiple times.
3. The administrator inserts a name into the field.
4. The administrator clicks on the “Add composite product” button.
5. The product is added.

Alternative Sequence 1: Products are not added to the composite or the name is not inserted

1. The administrator clicks on the “Add composite product” button.
2. The application displays an error message.
3. The scenario returns to step 1.

2.3.5 Use Case: Delete base/composite product

Primary Actor: administrator

Main Success Scenario:

1. The administrator clicks on a product from one of the two tables.
2. The administrator clicks on the “Delete” button.
3. The product is deleted.

Alternative Sequence: Product not selected

1. The administrator clicks on the “Delete” button in the graphical user interface.
2. The application displays an error message.
3. The scenario returns to step 1.

2.3.6 Use Case: Generate reports

Primary Actor: administrator

Main Success Scenario:

1. The administrator inputs data for reports.
2. The administrator clicks on the “Generate report” button.
3. The application generates the report in a .txt file.

Alternative Sequence 1: Wrong input or no input

1. The administrator clicks on the “Generate report” button.
2. The application displays an error message.
3. The scenario returns to step 1.

2.3.7 Use Case: Log in

Primary Actor: user

Main Success Scenario:

1. The user inserts a username and a password.
2. The user clicks on the “LOG IN” button.
3. The user is logged in the application.

Alternative Sequence: Wrong input or no input

1. The user clicks on the “LOG IN” button.
2. The application displays an error message.
3. The scenario returns to step 1.

2.3.8 Use Case: Search products

Primary Actor: client

Main Success Scenario:

1. The client inserts data into the fields.
2. The client clicks on the “SEARCH” button.
3. The products matching the search appear.

Alternative Sequence: Wrong data or no data

1. The client clicks on the “SEARCH” button.
2. The application displays an error message.
3. The scenario returns to step 1.

2.3.9 Use Case: Place order

Primary Actor: client

Main Success Scenario:

1. The client clicks on a product from one of the two tables and then clicks on the "ADD TO CART" button.
2. The step 1 repeats multiple times.
3. The client clicks on the "PLACE ORDER" button.
4. The order is placed and the employees are notified.
5. The user clicks on the "ADD ITEM" button.
6. The user clicks on the "PLACE ORDER" button.

2.3.10 Use Case: Register

Primary Actor: client

Main Success Scenario:

1. The client clicks on the "REGISTER" button.
2. The client inserts data into the fields.
3. The client clicks on the "SAVE" button.
4. The client is added to the application.

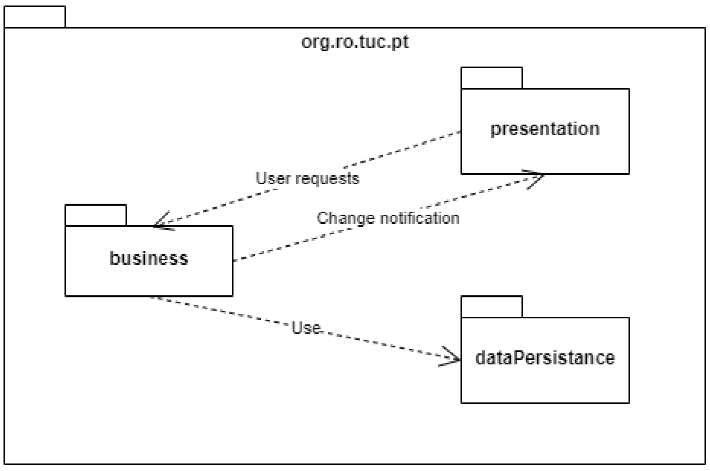
Alternative Sequence: Wrong data or no data

1. The client clicks on the "REGISTER" button.
2. The client inserts data into the fields.
3. The client clicks on the "SAVE" button
4. The application displays an error message.
5. The scenario returns to step 1.

3.DESIGN DECISIONS

3.1 Division into packages

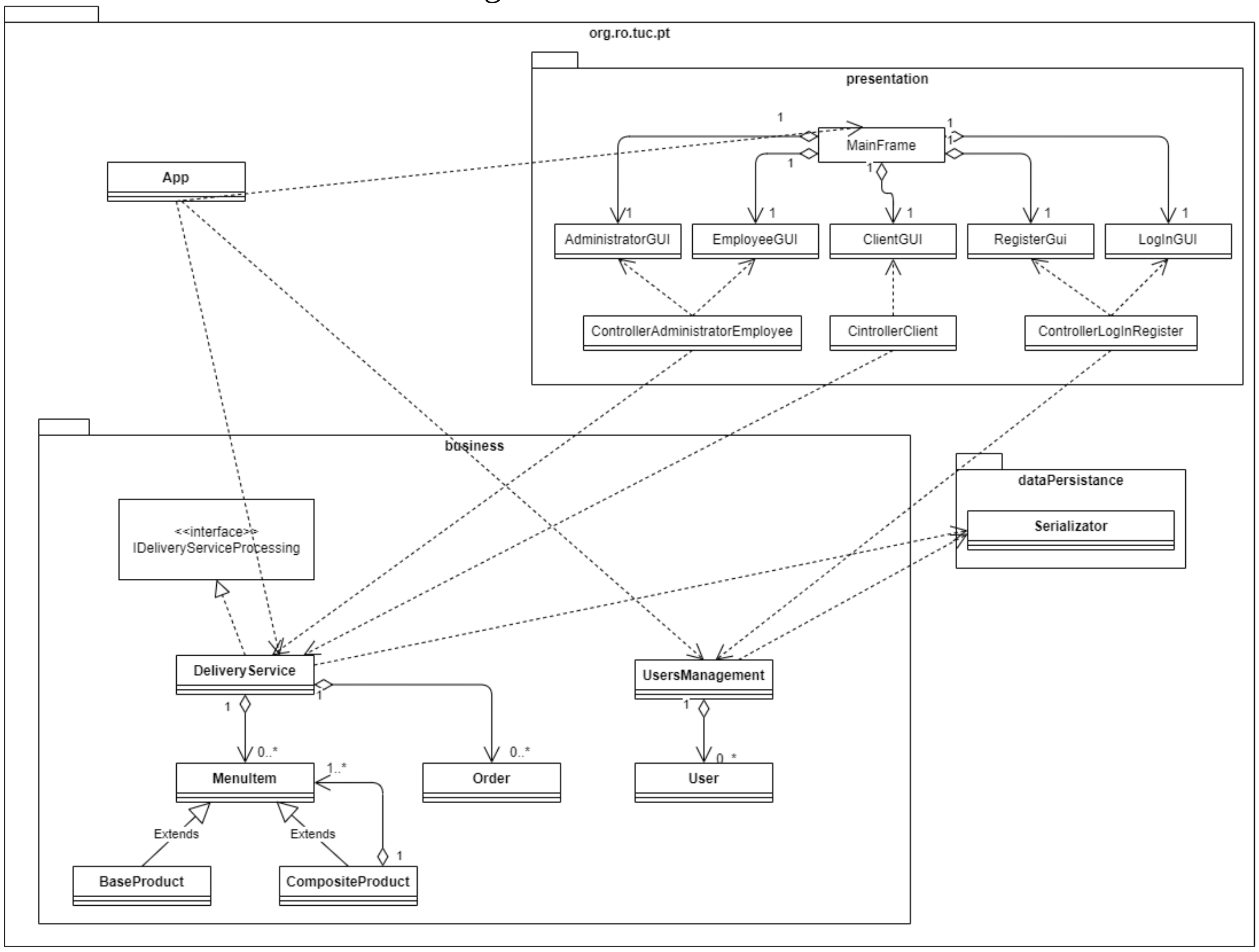
The Layered Architectural Pattern was used.



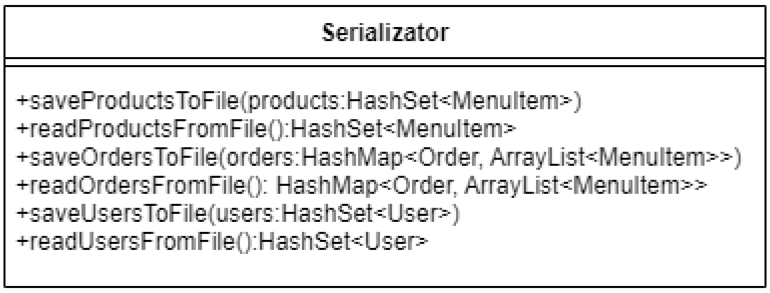
- presentation package contains the classes defining the user interface.
- business package contains the classes that encapsulate the application logic.
- dataPersistance package contains the class that takes care of persisting the application data (users, products, orders) using serialization.

3.2 Division into classes

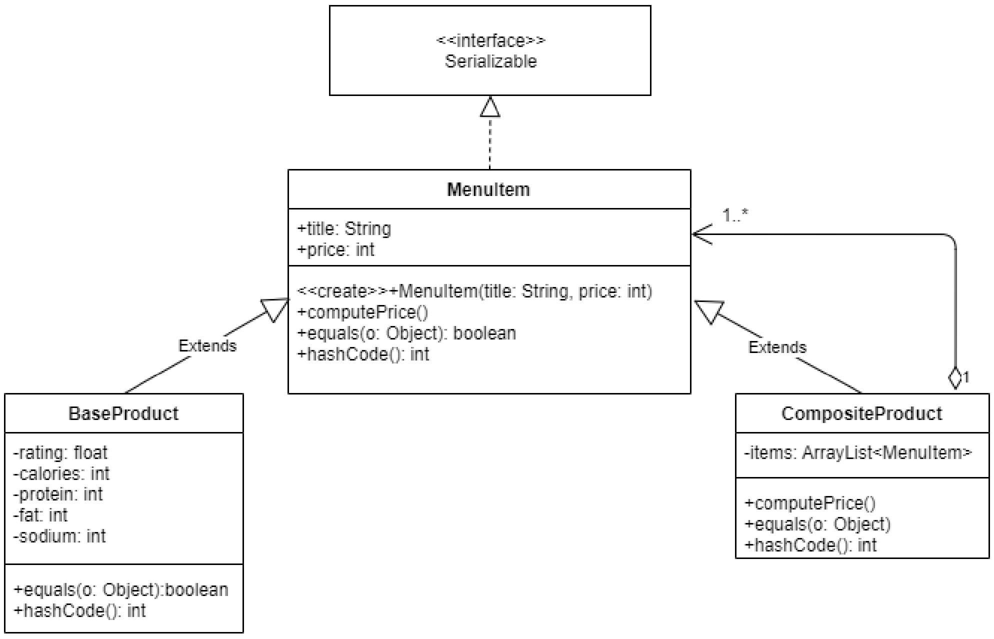
3.2.1 General class diagram



3.2.2 Class Serializator

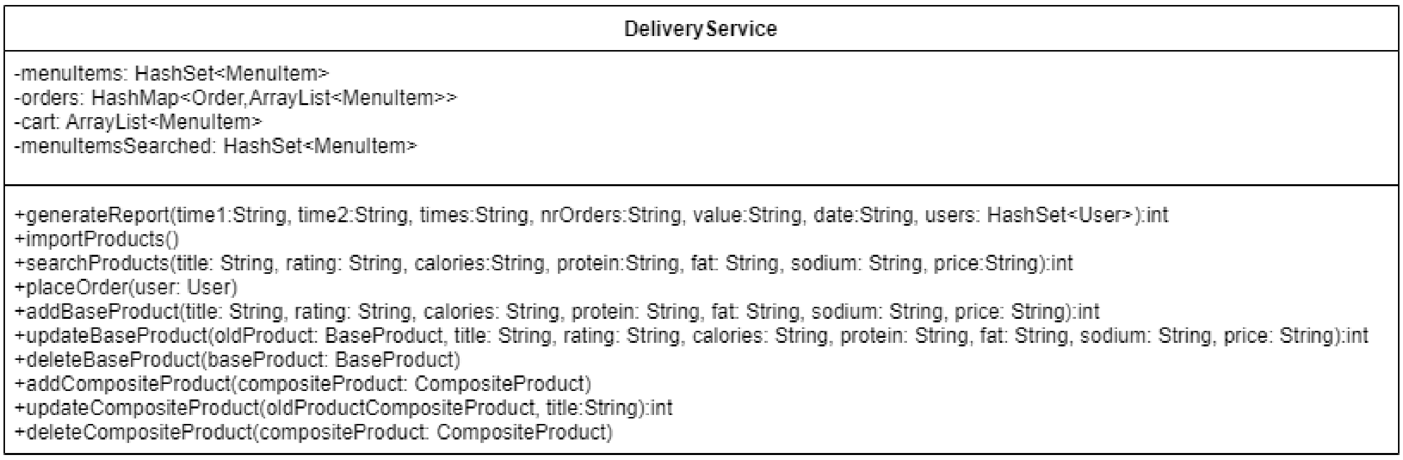

Serializator
+saveProductsToFile(products:HashSet<MenuItem>)
+readProductsFromFile():HashSet<MenuItem>
+saveOrdersToFile(orders:HashMap<Order, ArrayList<MenuItem>>)
+readOrdersFromFile(): HashMap<Order, ArrayList<MenuItem>>
+saveUsersToFile(users:HashSet<User>)
+readUsersFromFile():HashSet<User>

3.2.3 Classes MenuItem, BaseProducts, CompositeProducts

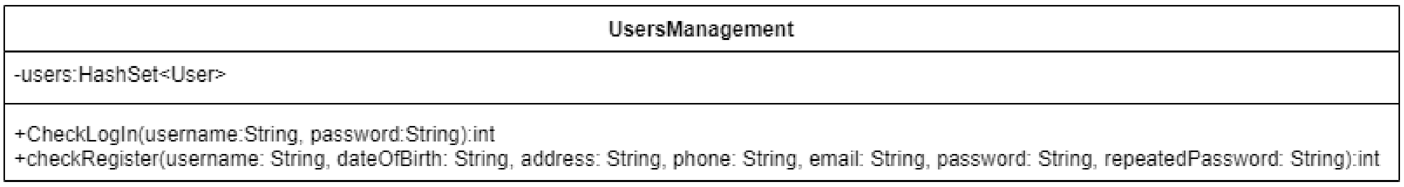


The composite design pattern was used.

3.2.4 Class DeliveryService


DeliveryService
-menuItems: HashSet<MenuItem>
-orders: HashMap<Order,ArrayList<MenuItem>>
-cart: ArrayList<MenuItem>
-menuItemsSearched: HashSet<MenuItem>
+generateReport(time1:String, time2:String, times:String, nrOrders:String, value:String, date:String, users: HashSet<User>):int
+importProducts()
+searchProducts(title: String, rating: String, calories:String, protein:String, fat: String, sodium: String, price:String):int
+placeOrder(user: User)
+addBaseProduct(title: String, rating: String, calories: String, protein: String, fat: String, sodium: String, price: String):int
+updateBaseProduct(oldProduct: BaseProduct, title: String, rating: String, calories: String, protein: String, fat: String, sodium: String, price: String):int
+deleteBaseProduct(baseProduct: BaseProduct)
+addCompositeProduct(compositeProduct: CompositeProduct)
+updateCompositeProduct(oldProductCompositeProduct, title:String):int
+deleteCompositeProduct(compositeProduct: CompositeProduct)

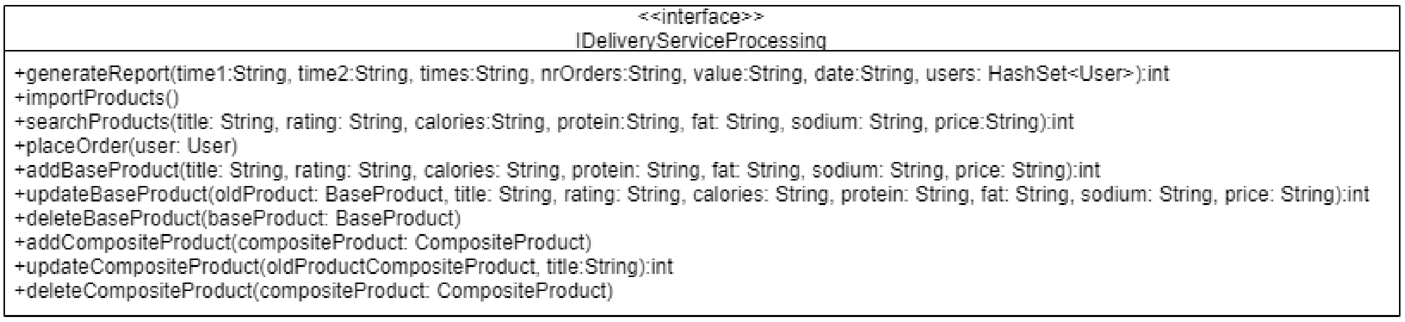
3.2.5 Class UsersManagement



```
UsersManagement
-users:HashSet<User>
+CheckLogIn(username:String, password:String):int
+checkRegister(username: String, dateOfBirth: String, address: String, phone: String, email: String, password: String, repeatedPassword: String):int
```

3.3 Interfaces

3.3.1 Interface IDeliveryServiceProcessing



```
<<interface>>
IDeliveryServiceProcessing
+generateReport(time1:String, time2:String, times:String, nrOrders:String, value:String, date:String, users: HashSet<User>):int
+importProducts()
+searchProducts(title: String, rating: String, calories:String, protein:String, fat: String, sodium: String, price:String):int
+placeOrder(user: User)
+addBaseProduct(title: String, rating: String, calories: String, protein: String, fat: String, sodium: String, price: String):int
+updateBaseProduct(oldProduct: BaseProduct, title: String, rating: String, calories: String, protein: String, fat: String, sodium: String, price: String):int
+deleteBaseProduct(baseProduct: BaseProduct)
+addCompositeProduct(compositeProduct: CompositeProduct)
+updateCompositeProduct(oldProductCompositeProduct, title:String):int
+deleteCompositeProduct(compositeProduct: CompositeProduct)
```

3.4 Data structures

The HashSet data structure was used for storing the menu items and the users:

```
HashSet<MenuItem> menuItems;

HashSet<User> users;
```

The HashMap data structure was used for storing the orders, maping them to the products they are composed of:

```
HashMap<Order,ArrayList<MenuItem>> orders
```

4. IMPLEMENTATION

4.1 Class descriptions

4.1.1 Class Serializator

Class used to serialize and deserialize the data of the application from files (Menu items, Orders, Users).

```
public static void saveProductsToFile(HashSet<MenuItem> products)
{
    try{
        File file1=new File("products.txt");
        FileOutputStream fileOutputStream1=new FileOutputStream(file1);
        ObjectOutputStream objectOutputStream1=new
ObjectOutputStream(fileOutputStream1);
        Iterator<MenuItem> iterator1= products.iterator();
        while (iterator1.hasNext()) {
            objectOutputStream1.writeObject((MenuItem)iterator1.next());
        }
        fileOutputStream1.close();
        objectOutputStream1.close();
    }
    catch (IOException e){
        e.printStackTrace();
    }
}
public static HashSet<MenuItem> readProductsFromFile()
{
    HashSet<MenuItem> toRet= new HashSet<>();
    try{
        File productsFile1= new File("products.txt");
        FileInputStream fileInputStream1= new FileInputStream(productsFile1);
        ObjectInputStream objectInputStream1 = new
ObjectInputStream(fileInputStream1);
        MenuItem menuItem;
        while((menuItem=(MenuItem) objectInputStream1.readObject())!=null)
            toRet.add(menuItem);
        fileInputStream1.close();
        objectInputStream1.close();
    }
    catch(EOFException ex1){
        //all products were read
        return toRet;
    }
    catch (Exception e){e.printStackTrace(); return null;}
    return toRet;
}
```

4.1.2 Class DeliveryService

Class is used to manage the application functionality for managing products and orders.
It extends Observable (to notify the employees when new orders are placed), and implements IDeliveryServiceProcessing.

Generating a report:

```
@Override
public int generateReport(String time1, String time2, String times, String
nrOrders, String value, String date, HashSet<User> users){
    if(time1!=null && !time1.isEmpty() && time2!=null && !time2.isEmpty()) {
        try {
            java.sql.Time time01 = java.sql.Time.valueOf(time1+ ":00");
java.sql.Time time02 = java.sql.Time.valueOf(time2+ ":00");
            int i=Integer.parseInt(time1.substring(0,2)); int
j=Integer.parseInt(time2.substring(0,2)); int
x=Integer.parseInt(time1.substring(3,5)); int
y=Integer.parseInt(time2.substring(3,5));
            if(i<0 || i>23 || j<0 || j>23 || i>j || (i==j && x<y) || x<0 ||
x>59 || y<0 || y>59) return -1;
            generateTimeIntervalReport(time1, time2);
            return 0;
        }
        catch (Exception e){ return -1; }
    }
    if(times!=null && !times.isEmpty()){
        try{
            int nr=Integer.parseInt(times);
            if(nr<0) return -1;
            generateReportProductsOrdered(nr);
            return 0;
        }catch (Exception e){return -1;}
    }
    if(nrOrders!=null && !nrOrders.isEmpty() && value!=null &&
!value.isEmpty()){
        try{
            int nr=Integer.parseInt(nrOrders); int
nr2=Integer.parseInt(value);
            if(nr<0 || nr2<0) return -1;
            generateReportClients(nr,nr2,users);
            return 0;
        }catch (Exception e){return -1;}
    }
    if(date!=null && !date.isEmpty()){
        if(date.length()!=10) return -1; char[] chars2 = date.toCharArray();
if (chars2[4]!='-' || chars2[7]!='-') return -1;
        try {
            int year=Integer.parseInt(date.substring(0,4)); int
month=Integer.parseInt(date.substring(5,7)); int
day=Integer.parseInt(date.substring(8,10));
            if(year>2021 || year<1920 || month>12 || month<1 || day>31 ||
day<1) return -1;
            generateReportProductsOnDate(date);
            return 0;
        } catch (NumberFormatException e) { return -1; }
    }
    return -1; //no input
}
```

```java
private void generateTimeIntervalReport(String time1, String time2) {
    java.sql.Time time01= java.sql.Time.valueOf(time1+":00"); java.sql.Time
time02= java.sql.Time.valueOf(time2+":00");
    Set<Order> ordersAfterFilter= orders.keySet();
    ordersAfterFilter= ordersAfterFilter
            .stream()
            .filter(x-
>(java.sql.Time.valueOf(x.getDateAndTime().substring(11))).after(time01) &&
java.sql.Time.valueOf(x.getDateAndTime().substring(11)).before(time02))
            .collect(Collectors.toSet());
    String str="Report "+nextReport +"\nOrders performed between "+time1+"
and "+time2+":\n";
    Iterator<Order> iterator=ordersAfterFilter.iterator();
    while (iterator.hasNext())
        str=str+iterator.next().toString()+"\n";
    createReport(str);
    nextReport++;
}

private void generateReportProductsOrdered(int times) {
    ArrayList<ArrayList<MenuItem>> itemsOrdered =new
ArrayList<>(orders.values());
    ArrayList<MenuItem> items=new ArrayList<>();
    Iterator<ArrayList<MenuItem>> iterator1= itemsOrdered.iterator();
    while (iterator1.hasNext()){
        Iterator<MenuItem> iterator2=iterator1.next().iterator();
        while (iterator2.hasNext()){ items.add(iterator2.next()); }
    }
    Map<MenuItem, Long> counts= items
            .stream()
            .collect(Collectors.groupingBy(e-> e,Collectors.counting()));

    ArrayList<MenuItem> products=new ArrayList<>();
    for(Map.Entry<MenuItem, Long> entry: counts.entrySet()){
        if(entry.getValue()>times)
            products.add(entry.getKey());
    }
    String str="Report "+nextReport +"\nProducts ordered more than "+times+"
times:\n";
    Iterator<MenuItem> iterator=products.iterator();
    while (iterator.hasNext()) str=str+iterator.next().toString()+"\n";
    createReport(str);
    nextReport++;
}

private void generateReportClients(int nrTimes, int minValue, HashSet<User>
users) {
    Set<Order> ordersAfterFilter= orders.keySet();
    Map<String, Long> counts=ordersAfterFilter
            .stream()
            .filter(x->x.getTotalPrice()>minValue)
            .collect(Collectors.groupingBy(e-
>e.getClientID(),Collectors.counting()));

    ArrayList<String> ids= new ArrayList<>();
    for(Map.Entry<String, Long> entry: counts.entrySet()) {
        if (entry.getValue() > nrTimes)
            ids.add(entry.getKey());
    }

    String str="Report "+nextReport +"\nClients who made more than
"+nrTimes+" orders with a value greater than "+minValue+":\n";
    Iterator<User> iterator=users.iterator();
    while (iterator.hasNext()){
        User usr= iterator.next();
        if(ids.contains(usr.getId())) str=str+usr.toString()+"\n";
    }
    createReport(str);
    nextReport++;
}

private void generateReportProductsOnDate(String date) {
    Map<Order,ArrayList<MenuItem>> ordersAfterFilter= orders.entrySet()
            .stream()
            .filter(e-
>e.getKey().getDateAndTime().substring(0,10).equals(date))
            .collect(Collectors.toMap(entry->entry.getKey(), entry-
>entry.getValue()));

    ArrayList<ArrayList<MenuItem>> itemsOrdered =new
ArrayList<>(ordersAfterFilter.values());
    ArrayList<MenuItem> items=new ArrayList<>();
    Iterator<ArrayList<MenuItem>> iterator1= itemsOrdered.iterator();

    while (iterator1.hasNext()){
        Iterator<MenuItem> iterator2=iterator1.next().iterator();
        while (iterator2.hasNext()){ items.add(iterator2.next()); }
    }

    Map<MenuItem, Long> counts= items
            .stream()
            .collect(Collectors.groupingBy(e-> e,Collectors.counting()));

    String str="Report "+nextReport +"\nProducts order on "+date+":\n";
    for(Map.Entry<MenuItem, Long> entry: counts.entrySet()) {
str=str+entry.getKey().toString()+",\nordered "+entry.getValue()+" times\n";
}
    createReport(str);
    nextReport++;
}
```

Searching products:

```
@Override
public int searchProducts(String title, String rating, String calories,
String protein, String fat, String sodium, String price) {
    if((title==null || title.isEmpty()) && (rating==null || rating.isEmpty())
&& (calories==null || calories.isEmpty()) && (protein==null ||
protein.isEmpty()) && (fat==null || fat.isEmpty()) && (sodium==null ||
sodium.isEmpty()) && (price==null || price.isEmpty()))
        return 1;
    int cal=-1, prot=-1, fa=-1, sod=-1, pri=-1; float rat=-1;
    try{
        if(rating!=null && !rating.isEmpty()) rat=Float.parseFloat(rating);
        if(calories!=null && !calories.isEmpty())
cal=Integer.parseInt(calories);
        if(protein!=null && !protein.isEmpty())
prot=Integer.parseInt(protein);
        if(fat!=null && !fat.isEmpty()) fa=Integer.parseInt(fat);
        if(sodium!=null && !sodium.isEmpty()) sod=Integer.parseInt(sodium);
        if(price!=null && !price.isEmpty()) pri=Integer.parseInt(price);
    }
    catch (NumberFormatException e){return 0; }
    List<MenuItem> menuItemsAfterFilter=new
ArrayList<MenuItem>(getMenuItemsBase());
    List<MenuItem> menuItemsAfterFilterComposite=new
ArrayList<MenuItem>(getMenuItemsComposite());

    float finalRat = rat; int finalCal = cal, finalProt = prot, finalFa = fa,
finalSod = sod, finalPri = pri;
```

```
    Stream<BaseProduct> stream = menuItemsAfterFilter.stream().map(x ->
(BaseProduct)x);
    Stream<CompositeProduct> stream2 =
menuItemsAfterFilterComposite.stream().map(x -> (CompositeProduct) x);

    if(title!=null && !title.isEmpty()) {
        stream = stream.filter(x ->
x.getTitle().toLowerCase(Locale.ROOT).contains(title.toLowerCase(Locale.ROOT)
));
        stream2=stream2.filter(x-
>x.getTitle().toLowerCase(Locale.ROOT).contains(title.toLowerCase(Locale.ROOT
)));
    }
    if(rat!=-1) {
        stream=stream.filter(x->x.getRating()== finalRat);
        stream2=stream2.filter(x->x.getTotalRating()== finalRat);
    }
    if(cal!=-1) {
        stream=stream.filter(x->x.getCalories()== finalCal);
        stream2=stream2.filter(x->x.getTotalCalories()== finalCal);
    }
    if(prot!=-1){
        stream=stream.filter(x->x.getProtein()== finalProt);
        stream2=stream2.filter(x->x.getTotalProtein()== finalProt);
    }
    if(fa!=-1) {
        stream=stream.filter(x->x.getFat()== finalFa);
        stream2=stream2.filter(x->x.getTotalFat()== finalFa);
    }
    if(sod!=-1) {
        stream=stream.filter(x->x.getSodium()== finalSod);
        stream2=stream2.filter(x->x.getTotalSodium()== finalSod);
    }
    if(pri!=-1) {
        stream=stream.filter(x->x.getPrice()== finalPri);
        stream2=stream2.filter(x->x.getPrice()== finalPri);
    }

    menuItemsSearched=new HashSet<>();
    menuItemsAfterFilter=stream.collect(Collectors.toList());
    menuItemsSearched.addAll(menuItemsAfterFilter);
    menuItemsAfterFilterComposite=stream2.collect(Collectors.toList());
    menuItemsSearched.addAll(menuItemsAfterFilterComposite);
    return -1;
}
```

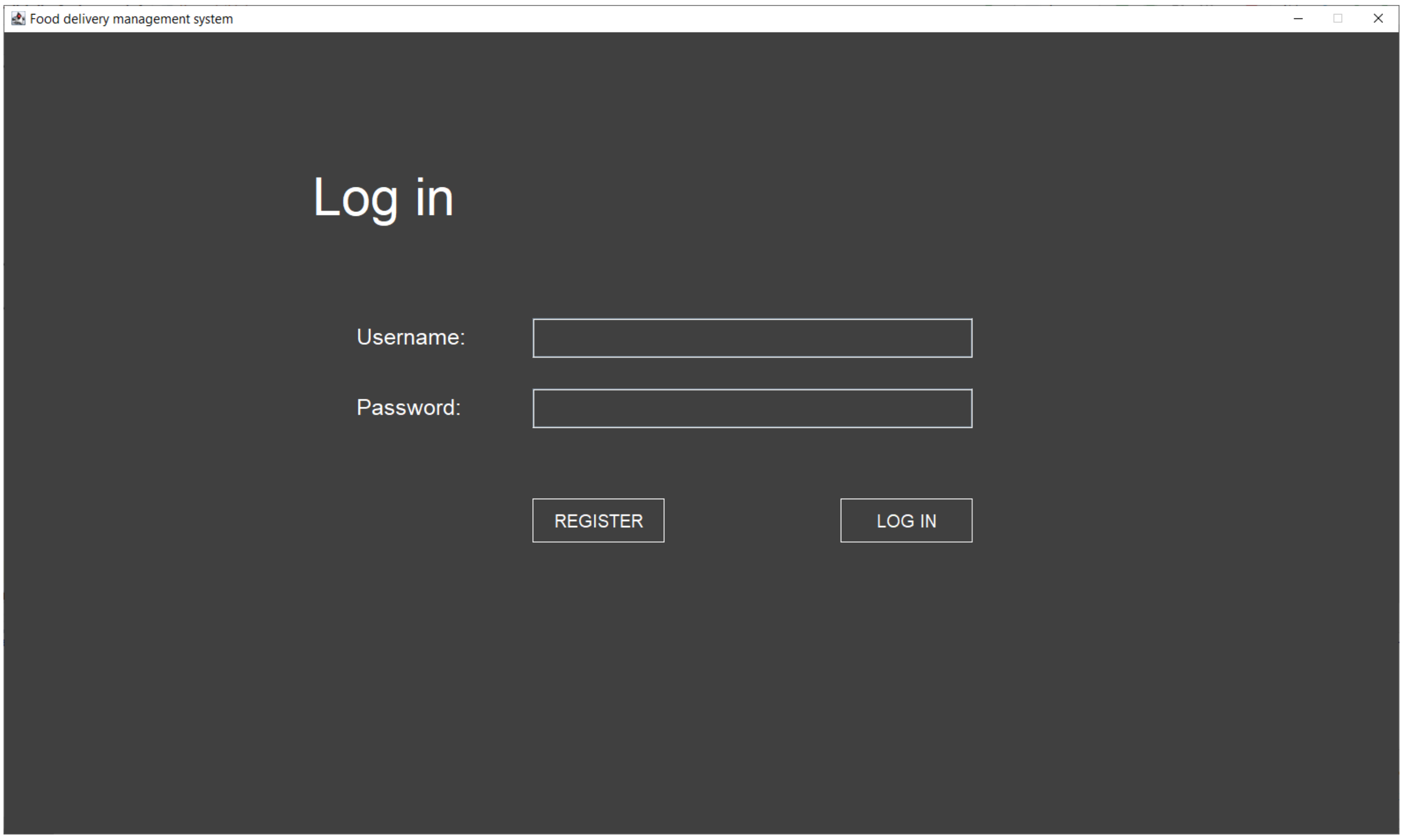
4.1.3 Class UsersManagement

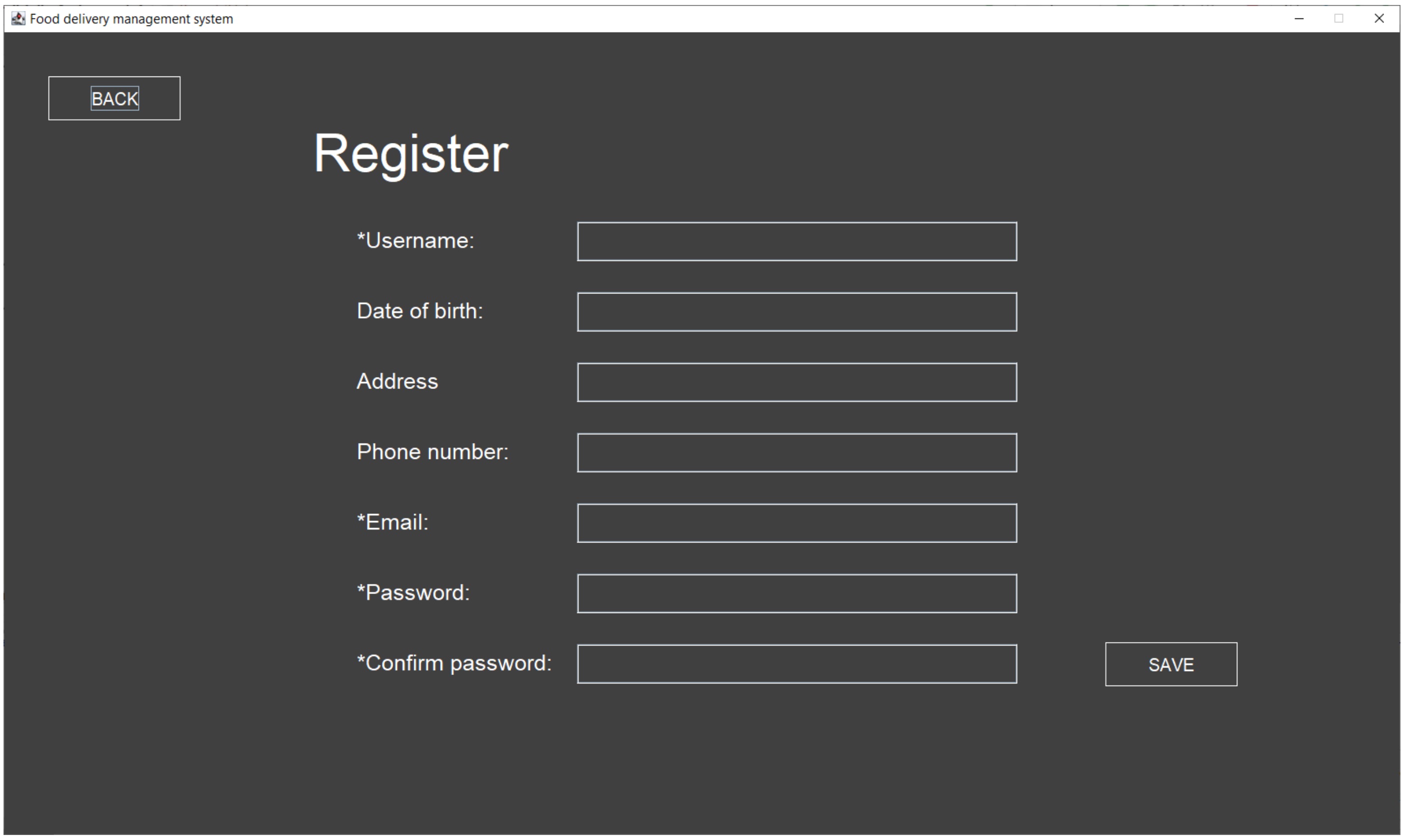
Class used for holding the users of the application, and for managing the register and log in operations.

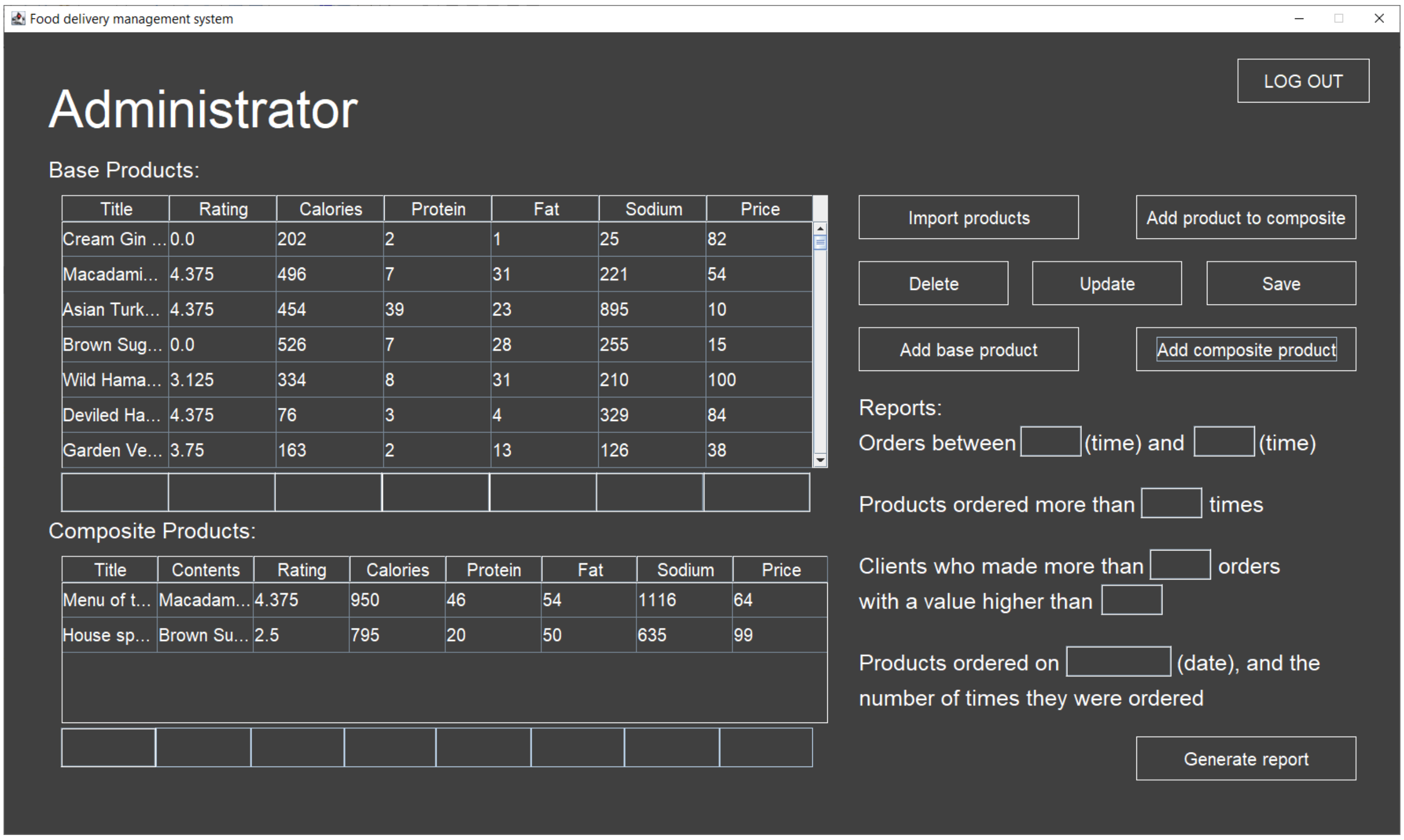
```
public int CheckLogIn(String username, String password)
{
    Iterator<User> iterator=users.iterator();
    while (iterator.hasNext()) {
        User user= iterator.next();
        if(user.getUsername().equals(username) &&
user.getPassword().equals(password)) {
            currentUser=user;
            char c=user.getId().charAt(0);
            if(c=='A') return 1;
            else if(c=='C') return 2;
            else return 3;
        }
    }
    return 0;
}
public int checkRegister(String username, String dateOfBirth, String address,
String phone, String email, String password, String repeatedPassword)
{
    if(username==null || username.isEmpty()) return 0;
    Iterator<User> iterator1= users.iterator();
    while (iterator1.hasNext())
if(iterator1.next().getUsername().equals(username)) return 1;

    if(dateOfBirth!= null && !dateOfBirth.isEmpty()) {
        if(dateOfBirth.length()!=10) return 2;
        char[] chars2 = dateOfBirth.toCharArray();
        if (chars2[4]!='-' || chars2[7]!='-') return 2;
        try {
            int year=Integer.parseInt(dateOfBirth.substring(0,4)); int
month=Integer.parseInt(dateOfBirth.substring(5,7)); int
day=Integer.parseInt(dateOfBirth.substring(8,10));
            if(year>2020 || year<1920 || month>12 || month<1 || day>31 ||
day<1) return 2;
        }
        catch (NumberFormatException e) { return 2; }
    }
    if(phone != null && !phone.isEmpty()) {
        if(phone.length()!=10) return 3;
        for(int i=0;i<9;i++) if(!Character.isDigit(phone.charAt(i))) return
3;
    }
    if(email==null || email.isEmpty() || !this.valEmail(email)) return 4;
    Iterator<User> iterator= users.iterator();
    while (iterator.hasNext()) if(iterator.next().getEmail().equals(email))
return 5;
    if(password==null || password.isEmpty()) return 6;
    if(repeatedPassword==null || repeatedPassword.isEmpty() ||
password.compareTo(repeatedPassword)!=0) return 7;

    User newUser= new
User(nextClientID,username,dateOfBirth,address,phone,email,password);
    users.add(newUser);
    currentUser=newUser;
    int nr=Integer.parseInt(nextClientID.substring(1));
    nextClientID="C"+(nr+1);
    return -1;
}
```

4.2 GUI description

The application has 5 windows that allow for log in, register, administrator operations, client operations and employee operations.







Administrator

Base Products:

Title	Rating	Calories	Protein	Fat	Sodium	Price
Cream Gin ...	0.0	202	2	1	25	82
Macadami...	4.375	496	7	31	221	54
Asian Turk...	4.375	454	39	23	895	10
Brown Sug...	0.0	526	7	28	255	15
Wild Hama...	3.125	334	8	31	210	100
Deviled Ha...	4.375	76	3	4	329	84
Garden Ve...	3.75	163	2	13	126	38

Composite Products:

Title	Contents	Rating	Calories	Protein	Fat	Sodium	Price
Menu of t...	Macadam...	4.375	950	46	54	1116	64
House sp...	Brown Su...	2.5	795	20	50	635	99

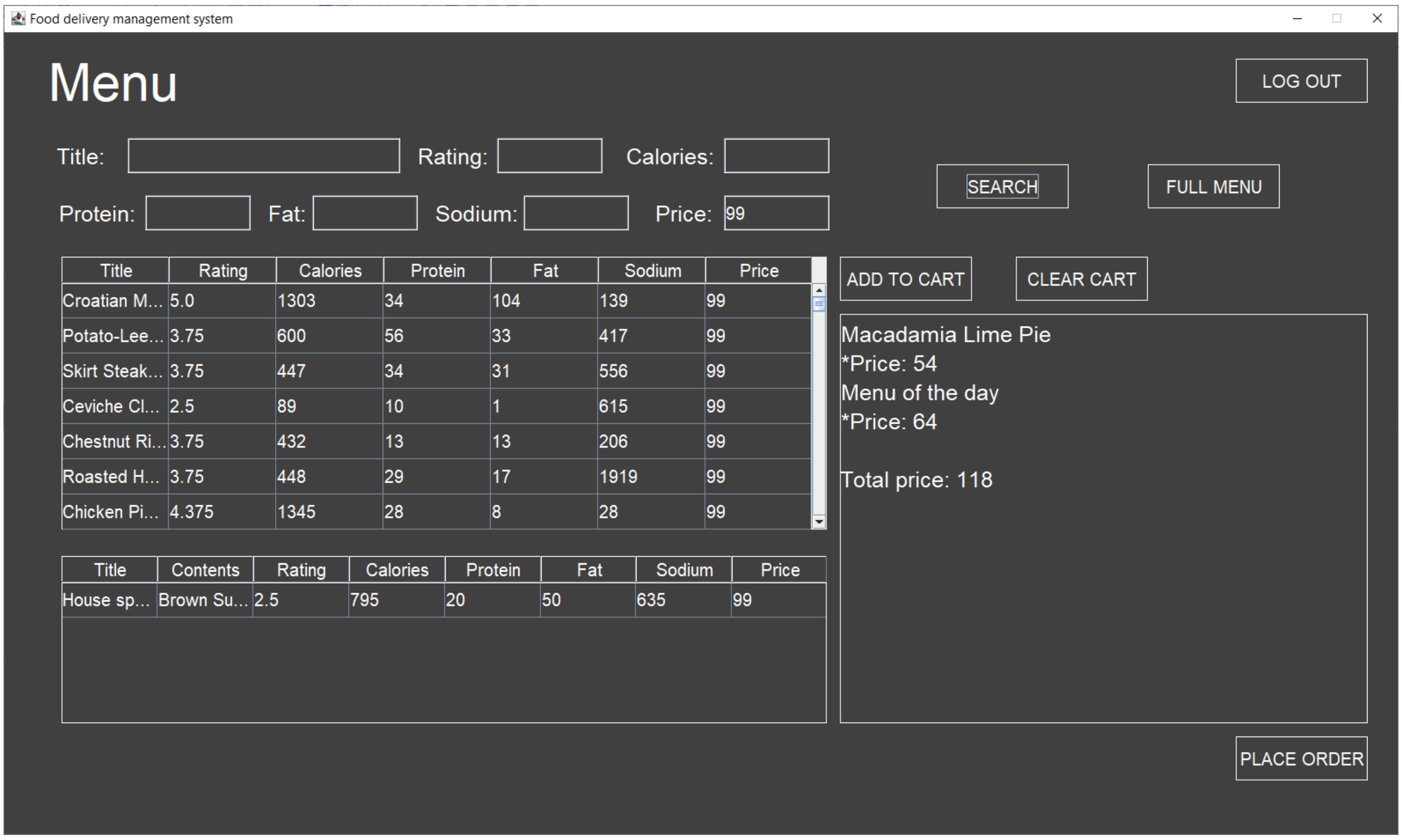
Reports:

Orders between ____ (time) and ____ (time)

Products ordered more than ____ times

Clients who made more than ____ orders with a value higher than ____

Products ordered on ____ (date), and the number of times they were ordered



Menu

Title: ____ Rating: ____ Calories: ____

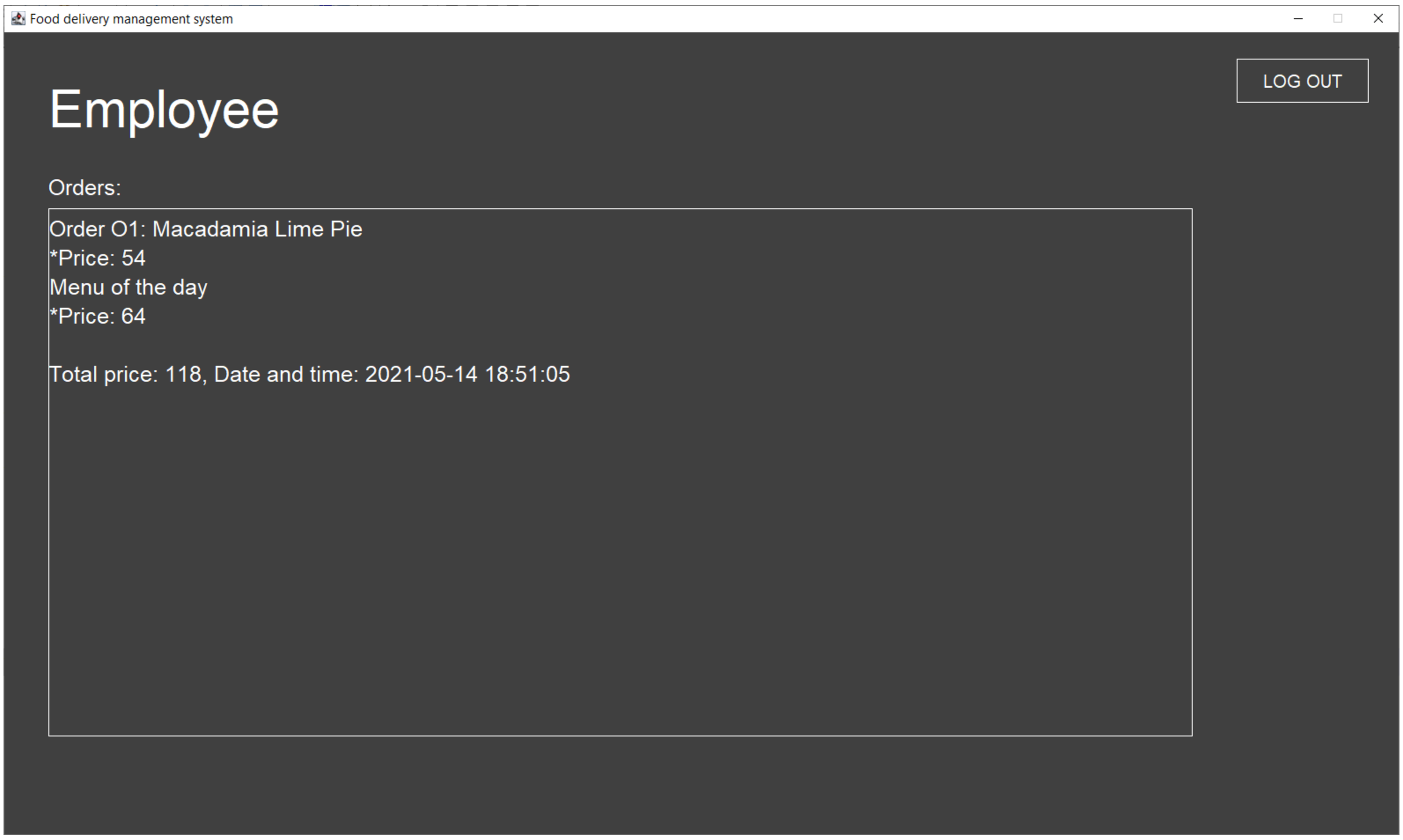
Protein: ____ Fat: ____ Sodium: ____ Price: 99

Title	Rating	Calories	Protein	Fat	Sodium	Price
Croatian M...	5.0	1303	34	104	139	99
Potato-Lee...	3.75	600	56	33	417	99
Skirt Steak...	3.75	447	34	31	556	99
Ceviche Cl...	2.5	89	10	1	615	99
Chestnut Ri...	3.75	432	13	13	206	99
Roasted H...	3.75	448	29	17	1919	99
Chicken Pi...	4.375	1345	28	8	28	99

Title	Contents	Rating	Calories	Protein	Fat	Sodium	Price
House sp...	Brown Su...	2.5	795	20	50	635	99

Macadamia Lime Pie
*Price: 54
Menu of the day
*Price: 64

Total price: 118

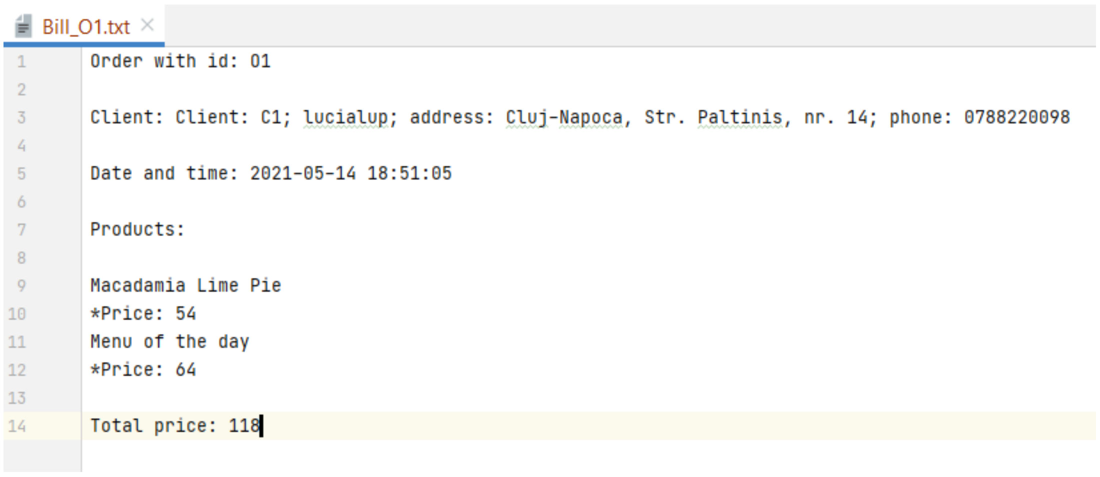


Employee

Orders:

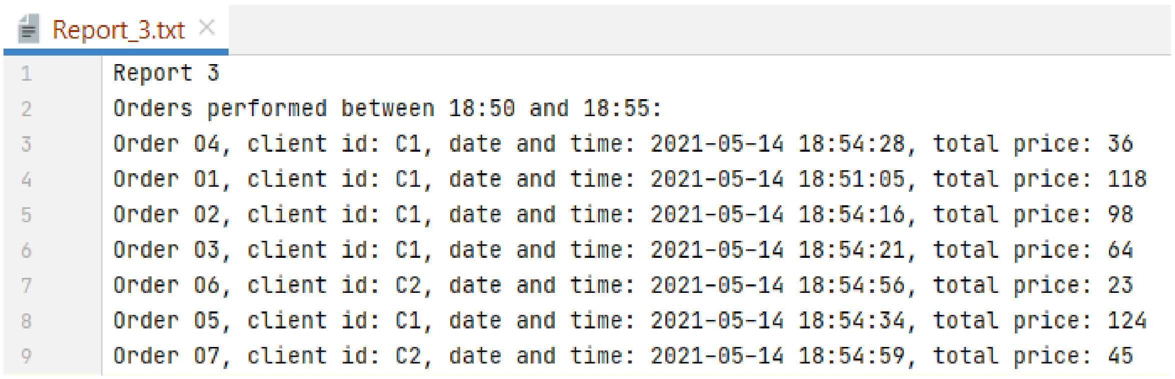
```
Order O1: Macadamia Lime Pie
*Price: 54
Menu of the day
*Price: 64

Total price: 118, Date and time: 2021-05-14 18:51:05
```

When an order is placed, a bill text file is generated.



```
Order with id: 01

Client: Client: C1; lucialup; address: Cluj-Napoca, Str. Paltinis, nr. 14; phone: 0788220098

Date and time: 2021-05-14 18:51:05

Products:

Macadamia Lime Pie
*Price: 54
Menu of the day
*Price: 64

Total price: 118
```

When a report is generated a .txt file is created



```
Report 3
Orders performed between 18:50 and 18:55:
Order 04, client id: C1, date and time: 2021-05-14 18:54:28, total price: 36
Order 01, client id: C1, date and time: 2021-05-14 18:51:05, total price: 118
Order 02, client id: C1, date and time: 2021-05-14 18:54:16, total price: 98
Order 03, client id: C1, date and time: 2021-05-14 18:54:21, total price: 64
Order 06, client id: C2, date and time: 2021-05-14 18:54:56, total price: 23
Order 05, client id: C1, date and time: 2021-05-14 18:54:34, total price: 124
Order 07, client id: C2, date and time: 2021-05-14 18:54:59, total price: 45
```

5. CONCLUSIONS

5.1 What I have learned

- How to use the composite design pattern.
- How to use serialization.
- How to read from a .csv file.
- How to use lambda expressions and stream processing.
- How to design by contract

5.2 Future development

- The application could have a more friendly user interface with images or icons.
- The option to delete items from cart could be implemented.
- The employees could mark orders as ‘in progress’ or ‘done’.

6. BIBLIOGRAPHY

- https://stackoverflow.com/questions/10378855/java-io-invalidclassexception-local-class-incompatible
- https://www.google.com/search?q=local+class+incompatible%3A+stream+classdesc+serialVersionUID+%2C+local+class+serialVersionUID+&rlz=1C1GCEA_enRO906RO906&sxsrf=ALeKk01KAgdFsjSdpU8II0IWrdRUB3etSg%3A1620983654041&ei=Zj-eYOqCAsXc-gTYvLLABg&oq=local+class+incompatible%3A+stream+classdesc+serialVersionUID+%2C+local+class+serialVersionUID+&gs_lcp=Cgdnd3Mtd2l6EAMyBggAEBYQHjIGCAAQFhAeMgYIABAWEB4yBggAEBYQHjIGCAAQFhAeMgYIABAWEB4yBggAEBYQHlCfOljoV2CMY2gAcAB4AIAB_QGIAf0BkgEDMi0xmAEEoAEBqgEHZ3dzLXdpesABAQ&sclient=gws-wiz&ved=0ahUKEwjqzYrX6sjwAhVFrp4KHVieDGgQ4dUDCA4&uact=5
- https://mkyong.com/java-best-practices/understand-the-serialversionuid/
- https://l.facebook.com/l.php?u=https%3A%2F%2Fen.m.wikipedia.org%2Fwiki%2FDesign_by_contract%3Ffbclid%3DIwAR2cQP97CNXKzqnUhROE56o5MaeHhUMR5q2padd7cE-9XOCna8NzpVDrK00%23Languages_with_third-party_support&h=AT1jFQaFaFOfyBBTOLazFT8WJ3RXTwZYq7lkNDI6fUhAI-NbB17nyb6zlzw-5QDlf5EPS0IwbBgnaYjdq-UUi6CTIfxwSb-3GHLobl1ymDq4BLIzPLVucKr0aUhrggn2_NUUfg
- https://l.facebook.com/l.php?u=https%3A%2F%2Fstackoverflow.com%2Fquestions%2F13447567%2Freplacing-a-hashset-java-member%3Ffbclid%3DIwAR1T9POuzsMXFBZ0zljgwwld6qJh7tN6Ix-2y0k42pqYNrTNU3VQGAm1jqA&h=AT1jFQaFaFOfyBBTOLazFT8WJ3RXTwZYq7lkNDI6fUhAI-NbB17nyb6zlzw-5QDlf5EPS0IwbBgnaYjdq-UUi6CTIfxwSb-3GHLobl1ymDq4BLIzPLVucKr0aUhrggn2_NUUfg
- https://l.facebook.com/l.php?u=https%3A%2F%2Fwww.tutorialspoint.com%2Fjava%2Futil%2Fhashset_remove.htm%3Ffbclid%3DIwAR2SE0ugQStKNPuy79k6w2RDbzL1tpgy70OA1WwSTE5I56NF327D_1jGRlA&h=AT1jFQaFaFOfyBBTOLazFT8WJ3RXTwZYq7lkNDI6fUhAI-NbB17nyb6zlzw-5QDlf5EPS0IwbBgnaYjdq-UUi6CTIfxwSb-3GHLobl1ymDq4BLIzPLVucKr0aUhrggn2_NUUfg
- https://l.facebook.com/l.php?u=https%3A%2F%2Fstackoverflow.com%2Fquestions%2F49660669%2Fparsing-csv-file-using-java-8-stream%3Ffbclid%3DIwAR0U0DVfdUY29VH3rprOT9H5-fWZ1yjxydGSYV1tOcMeYjwdWoGA89oPuU8&h=AT1jFQaFaFOfyBBTOLazFT8WJ3RXTwZYq7lkNDI6fUhAI-NbB17nyb6zlzw-5QDlf5EPS0IwbBgnaYjdq-UUi6CTIfxwSb-3GHLobl1ymDq4BLIzPLVucKr0aUhrggn2_NUUfg