DOCUMENTATION

Assignment Nr. 4

FOOD DELIVERY MANAGEMENT SYSTEM

NUME STUDENT: Bar Luca-Narcis
GRUPA:30421

CONTENTS

1. Assignment Objectives

Main objective:

• Design and implement an application for managing the food orders for a catering company

sub-objectives:

• Analyze the problem and identify requirements

• Design the orders management application

• Implement the orders management application

• Test the orders management application

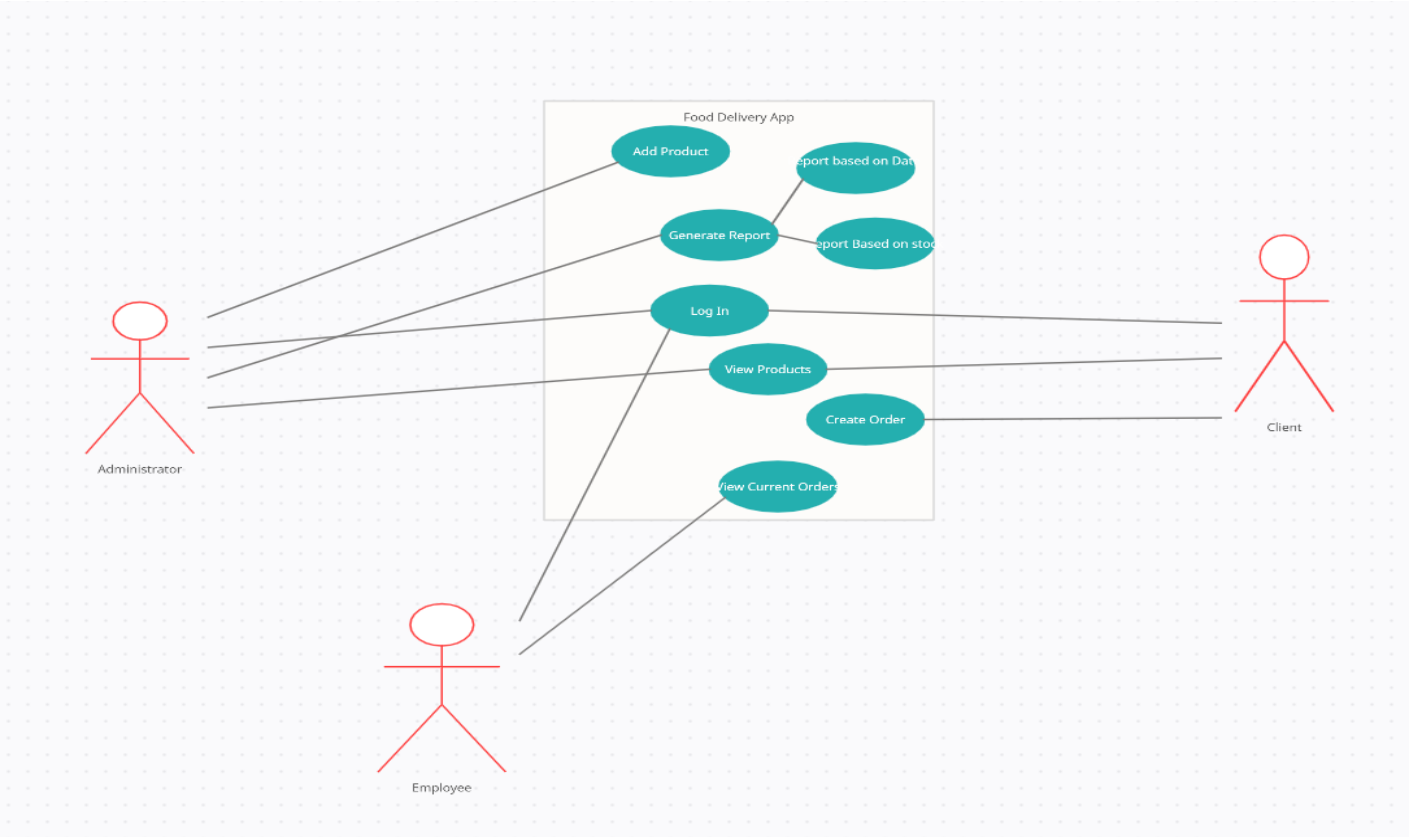
A JavaDOC is also included in the gitlab repository for this assignment

2. Problem analysis, modelling, scenarios, use-cases

Functional requirements:
- The application should allow the administrator to add a new product to the menu
- The application should allow the client to order products
-The application should allow the administrator to view all orders from each day
-The employee should be able to see current orders
-The application should allow the administrator to generate reports based on time or date of orders,stock of items and loyalty of customers(e.g based on how much they've spent or how many times they've ordered)
-The administrator should be allowed to see all of the menu items
-The application should allow users to register as their specific types(administrator,employee,client)

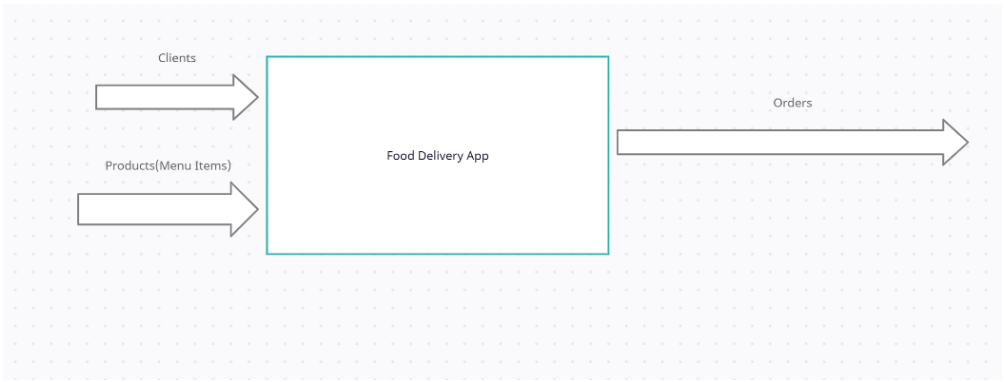
Non-Functional requirements:
- The application should be intuitive and easy to use by the user
- The application should use authentication to restrict the access of clients and regular employees to the administrator's functionalities
-The application should feature a login button
-The application should print the bills in a .txt format
An example of a use-case for the application would be as follows:

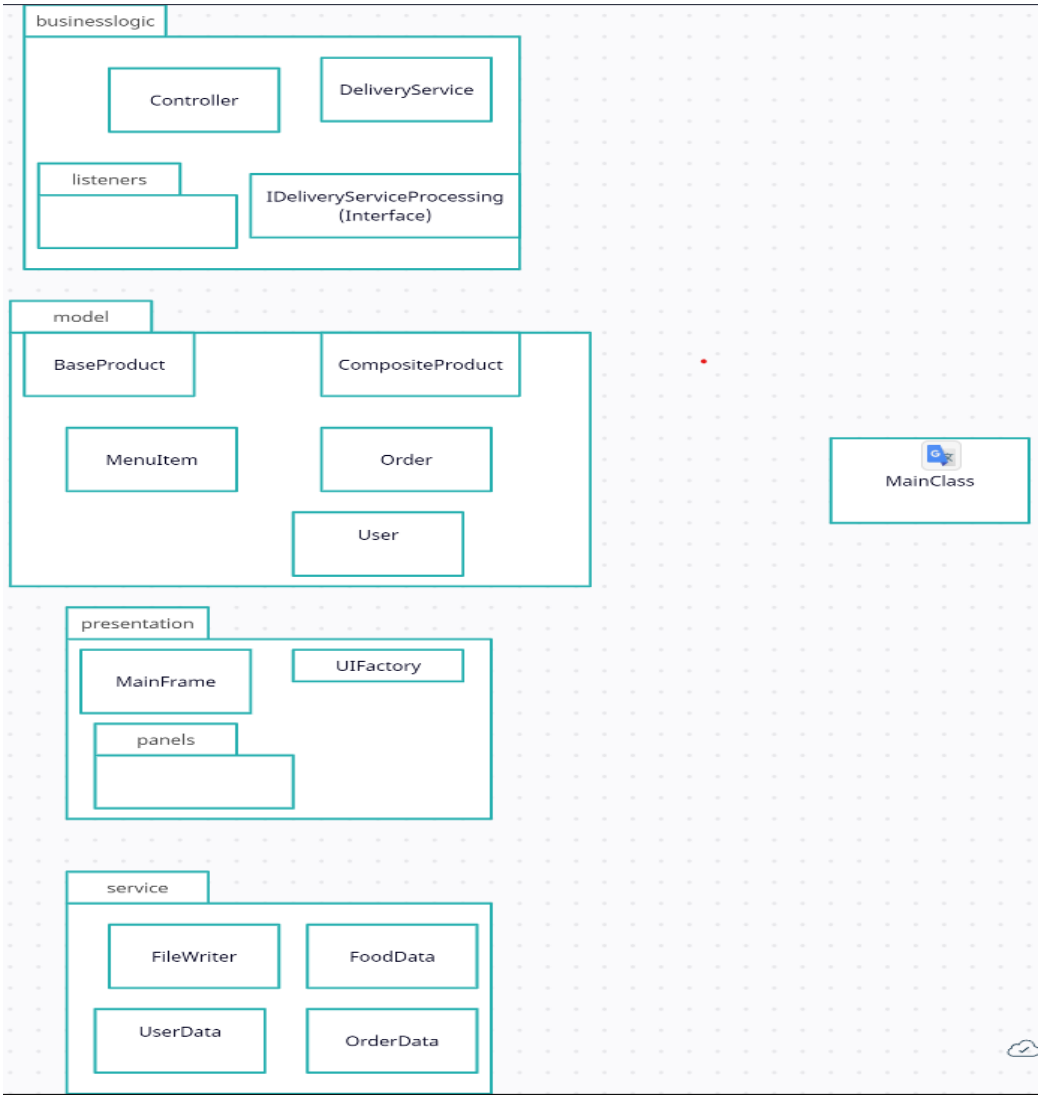
Use Case: add product to the menu
Primary Actor: administrator
Main Success Scenario:
1. The administrator selects the option to add a new product
2. The application will display a form in which the product details should be inserted
3. The administrator inserts the name of the product, the number of calories, proteins, fats, sodium and its price
4. The administrator clicks on the “Add” button
5. The application saves the product’s data and displays an acknowledge message
Alternative Sequence: Invalid values for the product’s data
- The administrator inserts a negative value for the number of calories/proteins/fats/sodium/price of the product
- The application displays an error message and requests the employee to insert a valid value for the number of calories/proteins/fats/sodium/price
- The scenario returns to step 3



3. *Design*

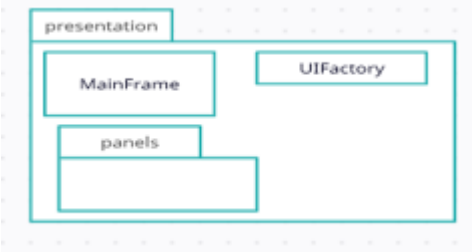
I have designed the app using a Conceptual(Layered) architecture using business logic, model, a service and presentation packages for a coherent and clean implementation. The “black box” below shows a pretty blunt explanation of how the app should work: basically, the customers uses the app to select a product/multiple products to be ordered and an order is created. Below the black box, I have included a minimalistic package diagram of the application and next up , I will (briefly) explain each of them.





I. The Presentation package:

The presentation package is mainly focused on the graphical user interface(GUI) of the application. Mainly, the UIFactory is a “general purpose” class which is used by the panel classes to implement their buttons , text fields, labels, and so on.

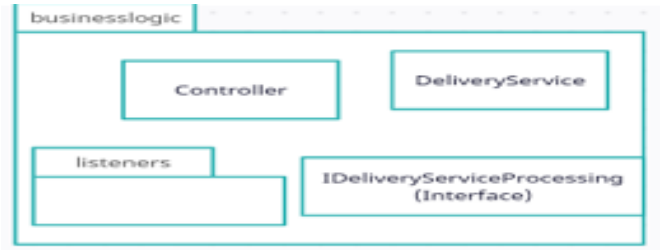


II.The Model package

As the name suggests, the model package contains the different object types that the application is going to be working with. BaseProduct is the”skeleton” of products, while CompositeProduct simply uses it to create items based on multiple base products. MenuItem is the one that finalizes the addition of an item in the menu, by establishing its final attributes and price.

III.The BusinessLogic package

It contains the IDeliveryServiceProcessing interface which basically defines the parameters of each function that the application will perform(everything that the client and administrator have access to, like creating orders, adding new products, generating reports, etc) as well as the DeliveryService class which defines their implementation. It also contains the Controller class and all of the listeners for the various operations and creations of new panels for those specific operations



IV. The service package

The service package takes care of the Serialization part. Its main focus is providing a direct line of communication between the app and the .ser files containing all of the users , products and orders, saving them when the app closes , and reloading them once we open it up again.

4. Implementation

Here you can see an ensemble view of the whole project and each of its classes. In this section I will explain the functionality of each of my classes. For an easier understanding and better corelation, I'll go package by package.

For dealing with storing data(e.g. users, products, orders) , I have used HashSet and my hashCode for those specific objects is based on their names, in order to avoid having duplicates.

I.PRESENTATION

MainFrame takes care of actually displaying the whole GUI, setting up the first panel that shows when opening the application .

UIFactory is used as a "general purpose" class, since every method used in the GUI is defined here, and every panel just extends it and uses its specific methods. Those methods include: creating buttons and labels, password and username fields and the displaying of user-specific panels.

In the panels subpackage we have the credentials panel, which is what shows up alongside the MainFrame when we start the application. Besides it, there are the panels for each of the 3 user types, which display all available actions for each of them. And also the UserPanel, which the AdminPanel and ClientPanel extend, where the operations on the menu(and menu items, respectively) are implemented.

```java
public static void setUpSpinners(ArrayList<Object> spinnerz) {
    spinnerz.add(UIFactory.createSpinner());
    spinnerz.add(UIFactory.createSpinner());
    spinnerz.add(UIFactory.createComboBox(new String[]{"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"}));
}

public static void setUpAdminButtons(ArrayList<JButton> buttonz) {
    buttonz.add(UIFactory.createButton(""));
    buttonz.add(UIFactory.createButton("Import products"));
    buttonz.add(UIFactory.createButton("Manage products"));
    buttonz.add(UIFactory.createButton("Generate reports"));
}

public static void setUpClientButtons(ArrayList<JButton> buttonz) {
    buttonz.add(UIFactory.createButton(""));
    buttonz.add(UIFactory.createButton("View products"));
    buttonz.add(UIFactory.createButton("Search product"));
    buttonz.add(UIFactory.createButton("Place order"));
}
```

II.MODEL

BaseProduct contains all the attributes that a product in the menu has(protein,price,fat,sodium,etc) as well as their hash code, used to identify them in the HashSet in which they are stored.

CompositeProduct is used for creating objects which comprise of multiple base products.

MenuItem is used to compute all the attributes(values) and price of an item, based on the BaseProducts that it is comprised of.

Order and User contain all of the attributes of the objects with their respective name.

In fact,all of them contain the methods equals and hashCode, because their code is generated based on their name, to be able to distinguish between two objects of the same type.

```java
public class MenuItem implements Serializable {
   @Serial
   private static final long serialVersionUID = 2590153167416271592L;

   public MenuItem() { }

   @Override
   public int hashCode() { return this.hashCode(); }

   @Override
   public boolean equals(Object obj) { return this.equals(obj); }

   public String getTitle() { return this.getTitle(); }

   public Object[] getValues() {
      if (this instanceof BaseProduct) {
         return ((BaseProduct) this).getValues();
      } else if (this instanceof CompositeProduct){
         return ((((CompositeProduct) this).getBaseProduct().getValues()));
      }
      return null;
   }

   public int computePrice() { return this.computePrice(); }
}
```

III.BUSINESS LOGIC

The IDeliveryServiceProcessing and DeliveryService interface and class contain the definition and implementation of the methods which the Client and Administrator use when dealing with products and orders.(e.g viewing, adding, deleting, modifying , creating orders , generating reports and so on), and also the method for printing the bill in a .txt file.

The Controller makes sure that each element in the graphical interface(which can be interacted with, of course) has its usage. To do this, each of the buttons in the app have their own respective listener ,which are stored in the listeners package. I've chosen this method to provide a more easy to read implementation of the app, since it is easier to read multiple classes of 40-50 lines of code rather than a big one with hundreds of lines of code.

IV. SERVICE

FileWriter basically takes care of serializing and deserializing the objects, while OrderData, ProductData and UserData call the 2 methods mentioned in order to add, search for or delete an object from those in the serialized files.

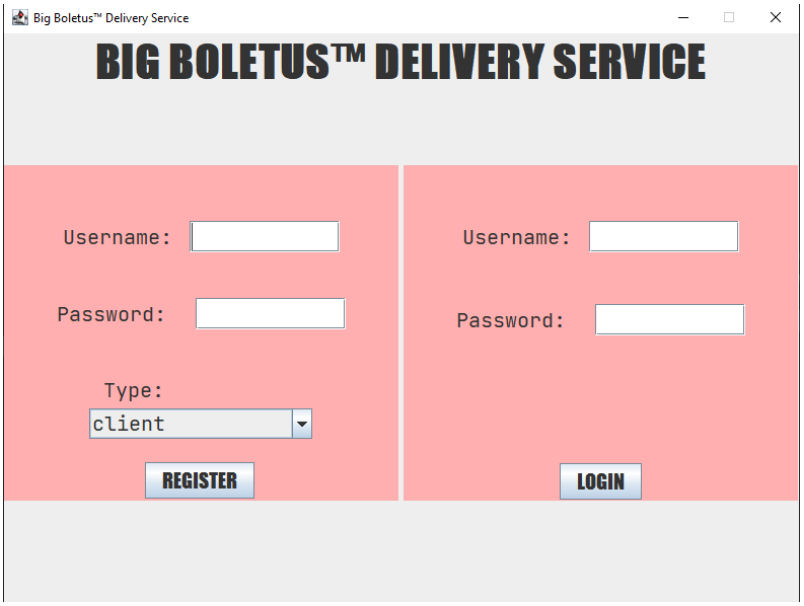
```java
public static boolean writeObject(Object data, String filePath) {
    try {
        FileOutputStream fout = new FileOutputStream(filePath);
        ObjectOutputStream outputStream = new ObjectOutputStream(fout);
        outputStream.writeObject(data);
        outputStream.close();
        fout.close();
    } catch (IOException e) {
        e.printStackTrace();
        return false;
    }

    return true;
}
```

```java
public static Object readObject(String filePath) {
    Object object = null;
    try {
        File file = new File(filePath);
        FileInputStream fin;
        if (file.exists()) {
            fin = new FileInputStream(file);
            ObjectInputStream inputStream = new ObjectInputStream(fin);
            object = inputStream.readObject();
            inputStream.close();
            fin.close();
        }
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
    }
    return object;
}
```

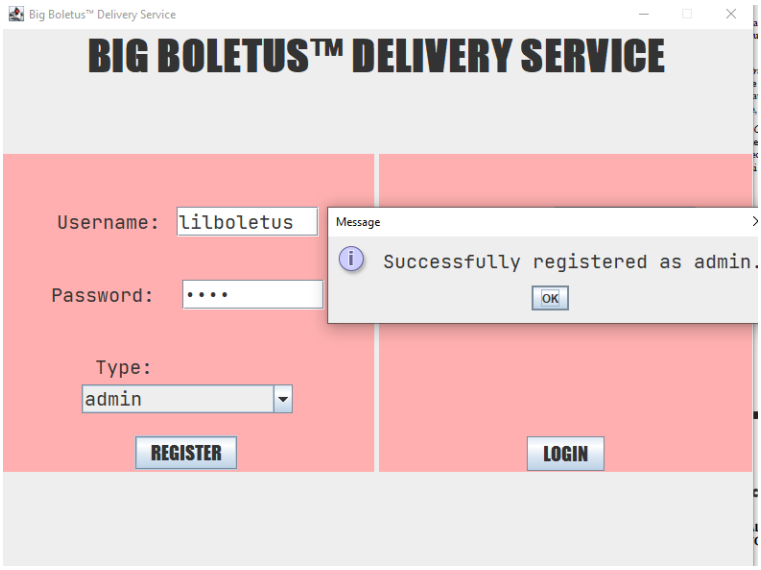
5. Results

In the following section, I will add some screenshots with the way the application works.

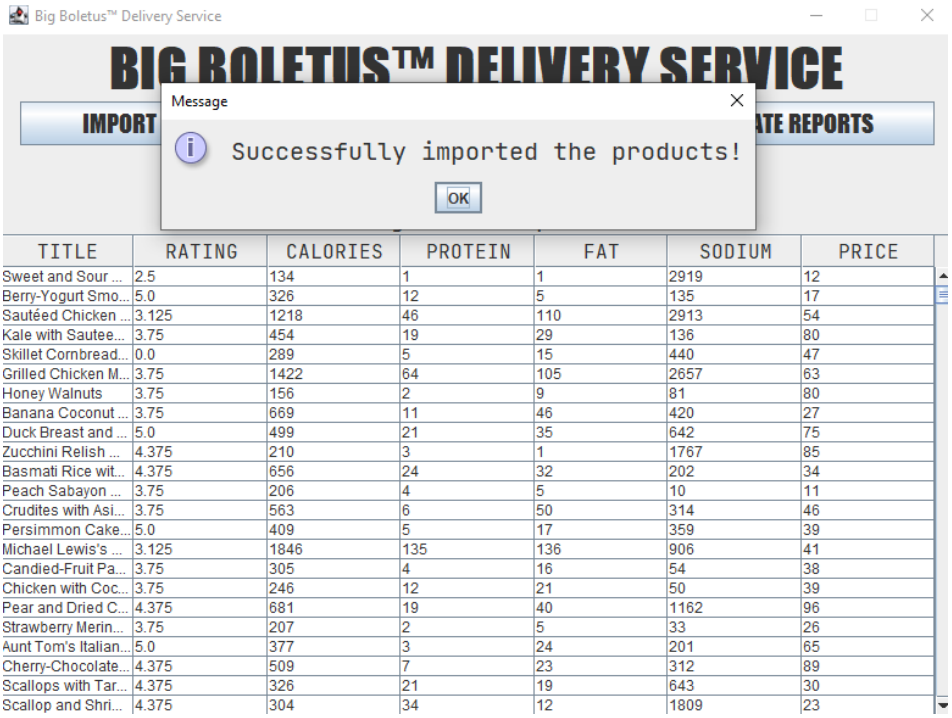
The Log in window:



Registering as admin

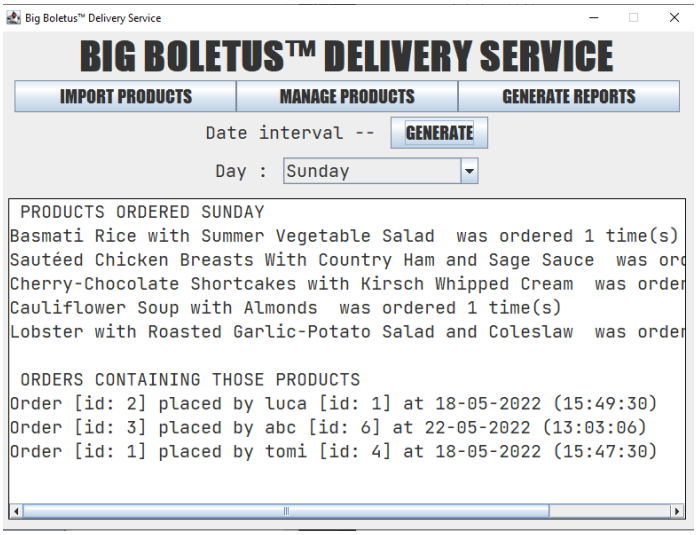


Admin operations:
Importing items from .csv

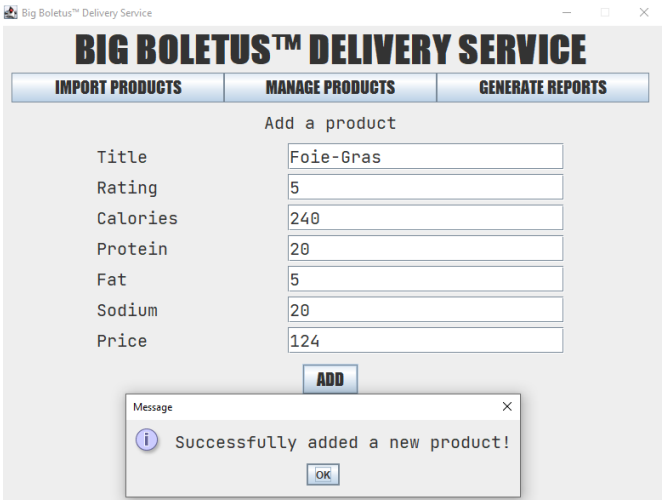


TITLE	RATING	CALORIES	PROTEIN	FAT	SODIUM	PRICE
Sweet and Sour ...	2.5	134	1	1	2919	12
Berry-Yogurt Smo...	5.0	326	12	5	135	17
Sautéed Chicken ...	3.125	1218	46	110	2913	54
Kale with Sautee...	3.75	454	19	29	136	80
Skillet Cornbread...	0.0	289	5	15	440	47
Grilled Chicken M...	3.75	1422	64	105	2657	63
Honey Walnuts	3.75	156	2	9	81	80
Banana Coconut ...	3.75	669	11	46	420	27
Duck Breast and ...	5.0	499	21	35	642	75
Zucchini Relish ...	4.375	210	3	1	1767	85
Basmati Rice wit...	4.375	656	24	32	202	34
Peach Sabayon ...	3.75	206	4	5	10	11
Crudites with Asi...	3.75	563	6	50	314	46
Persimmon Cake...	5.0	409	5	17	359	39
Michael Lewis's ...	3.125	1846	135	136	906	41
Candied-Fruit Pa...	3.75	305	4	16	54	38
Chicken with Coc...	3.75	246	12	21	50	39
Pear and Dried C...	4.375	681	19	40	1162	96
Strawberry Merin...	3.75	207	2	5	33	26
Aunt Tom's Italian...	5.0	377	3	24	201	65
Cherry-Chocolate...	4.375	509	7	23	312	89
Scallops with Tar...	4.375	326	21	19	643	30
Scallop and Shri...	4.375	304	34	12	1809	23

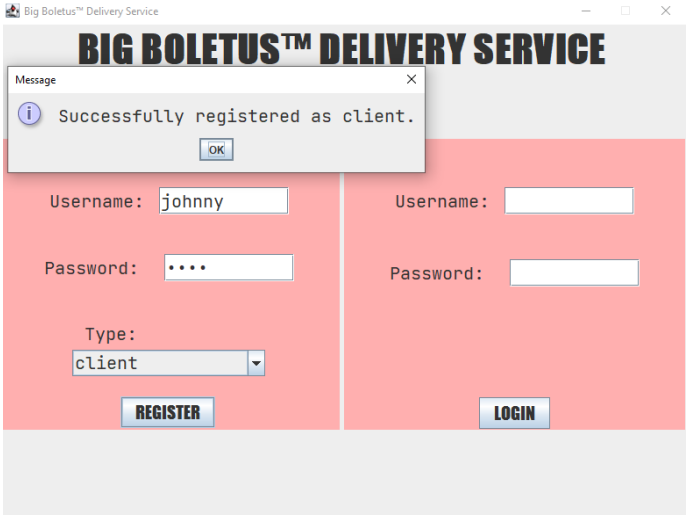
Generating a Report(Date)



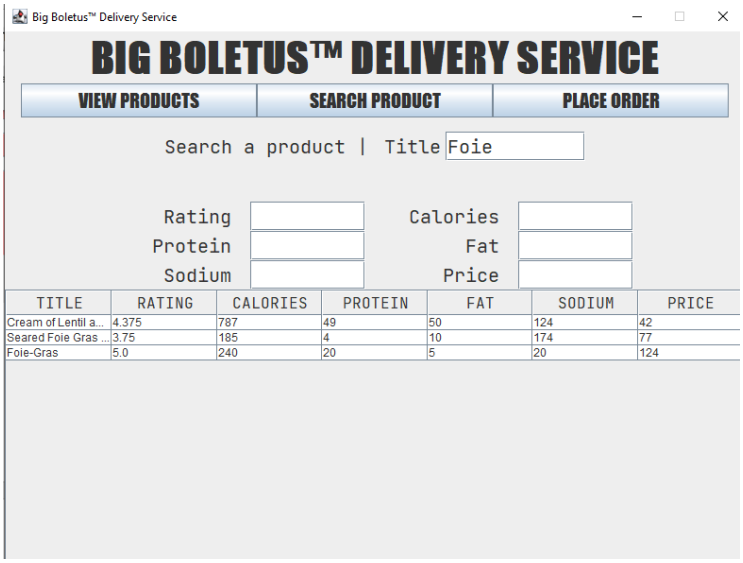
Adding a product:



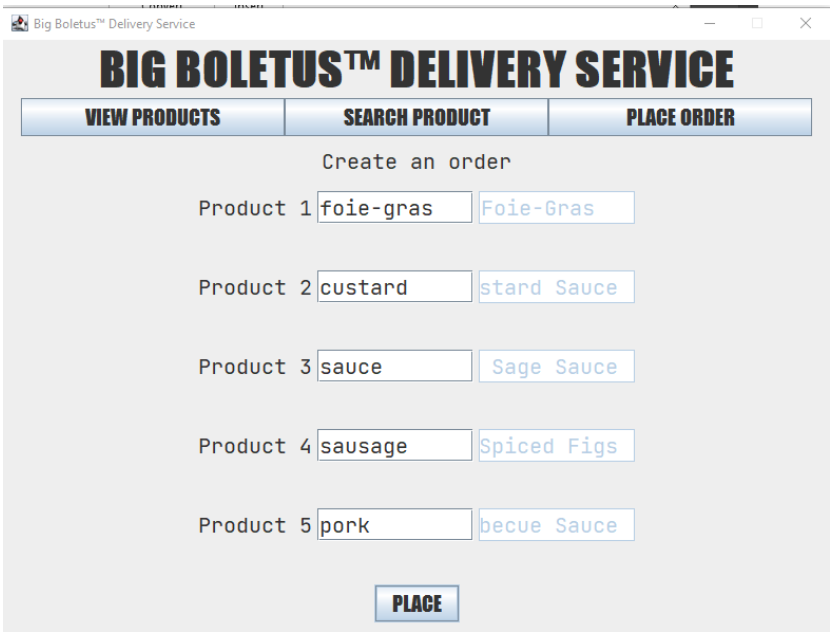
Registering as Client:



Searching for a product(the one added before as administrator)



Placing an order:



6.Conclusions

After doing this assignment, I consider that I've learned a few things about storing and reusing data in an application, without the use of a database. Though I still believe that using databases is a cleaner and more "elegant" way of dealing with large quantities of data. I've also gained more knowledge and experience with hashing and the usage of a Layered architecture in my projects.

As improvements, some that come off the top of my head would be:

- a logout button, to avoid the unpleasantry of having to reopen the app every time

-a display of current orders for the employees, since I didn't manage to get it to work in my case

- some images with the foods on the menu, and maybe some details regarding their prep time

- a system for delaying an order or setting it to arrive at a desired time

-a nicer, cleaner interface too

7.Bibliography

*HashSet (Java Platform SE 7) (oracle.com)

*Java 8 - Filter a Map examples - Mkyong.com

- FUNDAMENTAL PROGRAMMING TECHNIQUES (dsrl.eu)
- How to read and write Java object to a file - Mkyong.com