**FUNDAMENTAL PROGRAMMING TECHNIQUES**

**ASSIGNMENT 4**

**FOOD DELIVERY MANAGEMENT SYSTEM**

Teacher: prof. Ioan Salomie

Teacher Assistant: Cristina Bianca Pop

Student: Bucur Alexandra

Group: 30424

2020-2021

Contents

[1. Assignment objective 3](#_Toc72835784)

[2. Problem analysis, modeling, scenarios, use cases 5](#_Toc72835785)

[Analyzing the problem 5](#_Toc72835786)

[Modelling 5](#_Toc72835787)

[Scenarios and Use cases 5](#_Toc72835788)

[3. Design 8](#_Toc72835789)

[Design decisions 8](#_Toc72835790)

[Relationships, packages 8](#_Toc72835791)

[UML diagrams 9](#_Toc72835792)

[Data Structures 9](#_Toc72835793)

[Class Design 10](#_Toc72835794)

[User Interfaces 12](#_Toc72835795)

[4. Implementation 15](#_Toc72835796)

[5. Results 16](#_Toc72835797)

[6. Conclusions 18](#_Toc72835798)

[Further implementation 18](#_Toc72835799)

[7. Bibliography 19](#_Toc72835800)

# 1. Assignment objective

**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 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).

* 1. • 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.
     1. o the products ordered more than a specified number of times so far.
  2. o the clients that have ordered more than a specified number of times and the value of the order was higher than a specified amount.
  3. 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.

**Sub-objectives:**

• Analyze the problem and identify requirements

• Design the food delivery management system

• Implement the food delivery management system

• Test the food delivery management system

1. Analyze the problem and identify requirements

This part will analyze the requirements and decide the modelling, scenarios and use cases. As functional requirements the user, which have 3 types will perform certain operations, specific for their type.It will be presented in detail in part 2 of the documentation.

2. Design the food delivery management system.This part will analyze the design of food delivery management system. It will decide on a certain design pattern, decide on package configurations and the design of the classes. It will also present the OOP design of the application, the UML class and package diagrams, the data structures used, the defined interfaces and the algorithms used (if applicable) will be presented. This part will be presented in detail in part 3.

3. Implement the food delivery management system. This part will present in detail the implementation of the project. Each class will be described with important fields and methods. The implementation of the user interface will be described. This will be explained later, in part 4.

4. Test the food delivery management system. This part will present the result of the testing of the application. It will be presented later, in part 5.

# 2. Problem analysis, modeling, scenarios, use cases

## Analyzing the problem

General overview

This application should simulate the food delivery management system of any supermarket/restaurant, shop with clients and products. Clients should be able to create orders based on the products that are available in the stock, just like in real life. An administrator is creating new products or updating them and an employee will see all new orders and start preparing them.

The idea is to discover how working with a serialization works, by having some clients and orders in files and performing certain operations in an appropriate interface, designed especially for this scope.

This application should be able to fulfil all the requirements in order to display, modify, and keep track of orders, clients and products.

## Modelling

The user can be of 3 types: client, employee, administrator.

The user will be able to select between the available types of users and perform certain operations related to the products/orders, such as insert/delete/update or view. The client can also create orders, based on the existing products that are available.

Based on the input introduced by the client, the employee will be notified in the interface, in the designated place.

The administrator can add or modify or even delete the products that are available for the user.

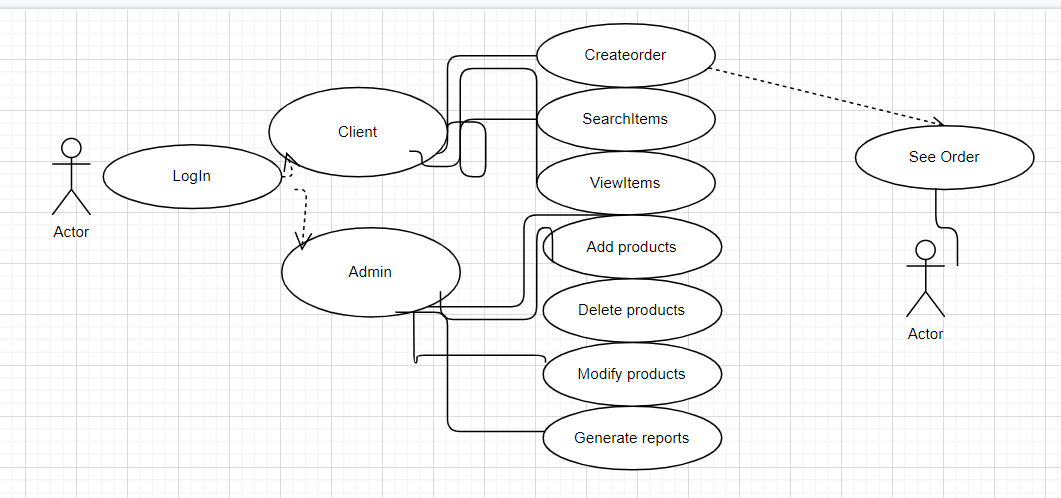
Input and Output

When talking about the input in the application, the user can choose to manage 3 types of users. All pages have specific operations available for the user.

For example, for the Client type the user can introduce search a product based on some key words, such as the name of the product, the price, rating and so on. All fields have to obey the standard rules such as price has to be and integer number, the name of the product can contain only letters and spaces.

When creating a new account the email address has to contain the “@” and the “.” Symbols.

## Scenarios and Use cases



Use cases: all the users have to be connected to their accounts

* Use case: View menu

Actors: administrator or client

Resume: The admin/ client can see the items that are available, after logging in into their account and pressing the designated button.

* Use case: Make Order

Primary Actor: client

Resume: The user selects the item he/she wants to order then he/she must press the button addItem. After the client added all the items desired in the cart, the user must press the finalize order button in order to be placed. A bill will be automatically created.

* Use case: Search item

Actors: client

Resume: The user presses the search button and then enters the filters he/she want to use for searching the item in the corresponding text fields (title, rating, calories, protein, fat, sodium and price). Then the user has to press the search button, the result of the search will be displayed in the table present in the interface.

* Use case: Modify item

Actors: client

Resume: The user presses the modify button and selects the object that is needed to be modified then enters the filters he/she want to use for searching the item in the corresponding text fields (title, rating, calories, protein, fat, sodium and price). If the data is correct the operation will be performed. If not an error will be displayed.

* Use case: Create item

Primary Actor: administrator

Resume: The user presses the create base product/ create composite product in function of what type of product will be created. A new window will open. The user will have to add all the data for the base products or the title and other items for the composite prod.

* Use case: Delete item

Primary Actor: administrator

Resume: The user selects the item to be deleted, and then the delete button should be pressed in order for the product to be deleted from the menu. The prod will no longer appear in the table.

* Use case: Generate reports

Primary Actor: administrator

Resume: The admin clicks the go to reports button, that will open a new window where you can enter the necessary data for the creation of the reports, and then it has to press the corresponding button in order to generate the report desired. The results will be displayed in the interface in a text area. For example, for the interval hours, the user will have to introduce the corresponding hours. If the data is correct the operation will be performed. If not an error will be displayed

.

* Use case: See order

Primary Actor: employee

Resume: Every time an order is created the employee will be notified, the order will appear in a designated place.

# 3. Design

## Design decisions

This project uses the layered architecture design.

A Layered Architecture, as I understand it, is the organization of the project structure into four main categories: presentation, application, domain, and infrastructure. Each of the layers contains objects related to the particular concern it represents.

**Presentation layer**: contains all of the classes responsible for presenting the UI to the end-user or sending the response back to the client (in case we’re operating deep in the back-end).

**Application layer:** contains all the logic that is required by the application to meet its functional requirements and, at the same time, is not a part of the domain rules.

In most systems the application layer consisted of services orchestrating the domain objects to fulfill a use case scenario.

**The domain layer**: represents the underlying domain, mostly consisting of domain entities and, in some cases, services. Business rules, like invariants and algorithms, should all stay in this layer.

**The infrastructure layer** (also known as the persistence layer): contains all the classes responsible for doing the technical stuff, like persisting the data in the database, like DAOs, repositories.

There are two important rules for a classical Layered Architecture to be correctly implemented:

1. All the dependencies go in one direction, from presentation to infrastructure. (Well, handling persistence and domain are a bit tricky because the infrastructure layer often saves domain objects directly, so it actually knows about the classes in the domain)

2. No logic related to one layer’s concern should be placed in another layer. For instance, no domain logic or database queries should be done in the UI.

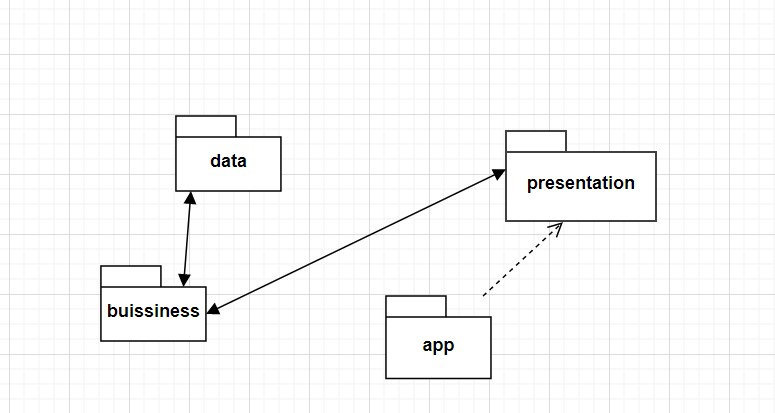
Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application (e.g., presentation logic or business logic). Although the layered architecture pattern does not specify the number and types of layers that must exist in the pattern, most layered architectures consist of four standard layers: presentation, business, persistence, and database (Figure 1-1). In some cases, the business layer and persistence layer are combined into a single business layer, particularly when the persistence logic (e.g., SQL or HSQL) is embedded within the business layer components. Thus, smaller applications may have only three layers, whereas larger and more complex business applications may contain five or more layers.

Each layer of the layered architecture pattern has a specific role and responsibility within the application. For example, a presentation layer would be responsible for handling all user interface and browser communication logic, whereas a business layer would be responsible for executing specific business rules associated with the request. Each layer in the architecture forms an abstraction around the work that needs to be done to satisfy a particular business request. For example, the presentation layer doesn’t need to know or worry about *how* to get customer data; it only needs to display that information on a screen in particular format. Similarly, the business layer doesn’t need to be concerned about how to format customer data for display on a screen or even where the customer data is coming from; it only needs to get the data from the persistence layer, perform business logic against the data (e.g., calculate values or aggregate data), and pass that information up to the presentation layer.

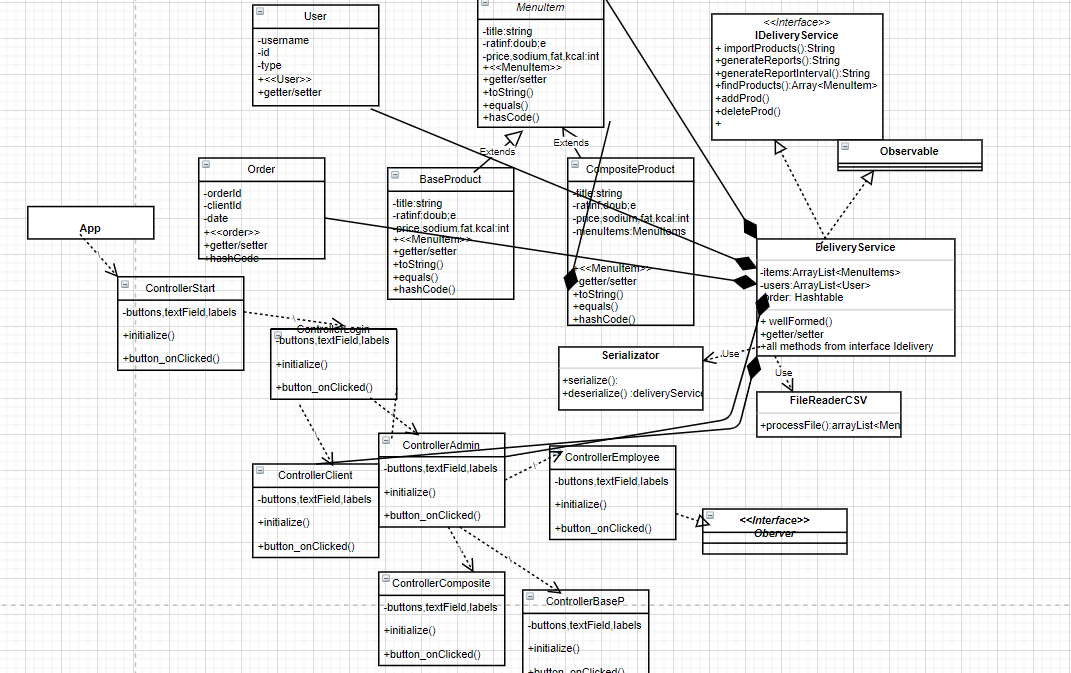
## Relationships, packages

Java packages help in organizing multiple modules and group together related classes and interfaces.

A Layered Architecture is the organization of the project structure into four main categories: presentation, application, domain, and infrastructure. Each of the layers contains objects related to the particular concern it represents.



UML diagrams



## Data Structures

The data structures that have been used in this application are either primitive data types, such as integers or doubles. I used lists and ArrayLists in order to make the code easy to be read and to facilitate the use of memory.

The rest of the data used are simple. I use strings in order to create the queries for the database and I mainly use reflection techniques in order to write generic methods for multiple types.

One more important data structure that I use are hashSets for the orders. Here, each order represents a key that will place the order items for that specific order in a designated “bucket”.

## Class Design

The whole idea of splitting your program into classes is based on a general rule named divide and conquer. The application divides a problem into smaller problems and then those small classes solve the simple and well-known problems.

1) The Model package – contains the logic of the application

**User Class**

Fields:

ID: int

Name:string

username:string

Type:ADMIN,CLIENT,EMPLOYEE

Constructors:

– public **User** (int ID,String name, Strinf username, Type type) : the constructor that initializes the user with the transmitted fields

– public **User** (String name, Strinf username, String type, UUID id) : the constructor that initializes the user with the transmitted fields

– public **User** () basic constructor

Methods:

-setters and getters

-public void toString() : displays one client on the screen

**Product Classes**

* There are 3 types of products classes. The most important one is the MenuItem one, then there is the BaseProduct and CompositeProduct
* The composite product contains a list of several simple products

Fields:

Name, rating, sodium, fat, protein, calories, price

Constructors:

– public Product () basic construcor

Methods:

-setters and getters

-public void toString() : displays one product on the screen

**Order Class**

Fields:

ID: int

idClient:int

date:LocalDate

Constructors:

– public Order (int ID,int idClient, date date) : the constructor that initializes the order with the transmitted fields

– public Order () basic construcor

Methods:

-setters and getters

-public void toString() : displays one order on the screen

2) The presentation package

**ControllerStart Class**

-deals with events for buttons and labels

-has a subclass **ProcessService** that helps displaying labels for a certain amount of time

-based on the clicked button it opens more pages

**ControllerRegister Class**

**-**deals with creating new user accounts

-deals with events for buttons and labels

-has a subclass **ProcessService** that helps displaying labels for a certain amount of time

-based on the clicked button it opens more pages

**ControllerLoginClass**

**-**deals with getting into an account of a user

-deals with events for buttons and labels

-has a subclass **ProcessService** that helps displaying labels for a certain amount of time

-based on the clicked button it opens more pages

**ControllerClient Class**

-deals with events for buttons and labels, but also validating data.

-creates new order

-displays the table with all the entries of items on the screen

-has a subclass **ProcessService** that helps displaying labels for a certain amount of time

-based on the clicked button it opens more pages

**ControllerAdmin Class**

-deals with events for buttons and labels, but also validating data.

-creates new products, modifies products, deletes products

-has a subclass **ProcessService** that helps displaying labels for a certain amount of time

-based on the clicked button it opens more pages

**ControllerEmployee Class**

-deals with events for buttons and labels, but also validating data.

-sees recently placed orders

-has a subclass **ProcessService** that helps displaying labels for a certain amount of time

-based on the clicked button it opens more pages

**ControllerOrder Class**

-displays a recently placed order

**ControllerBaseProduct Class**

-takes data to create a new object

**ControllerCompositeProcuct Class**

-takes data to create a new object

3) The Dao(Data access package)

**EmailValidator Class**

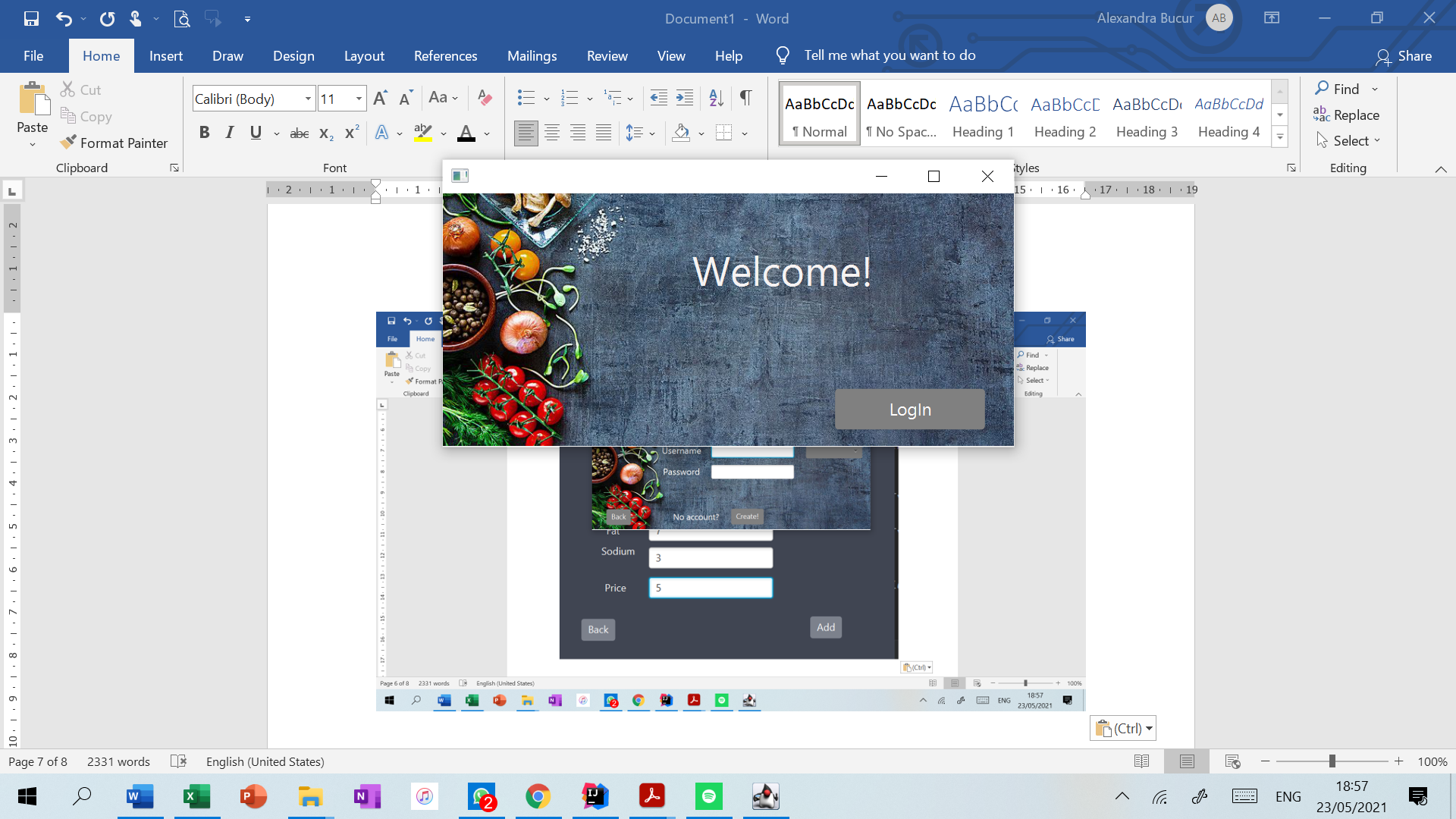
-implements Validator

-checks if a client have a valid email or not

**App class**

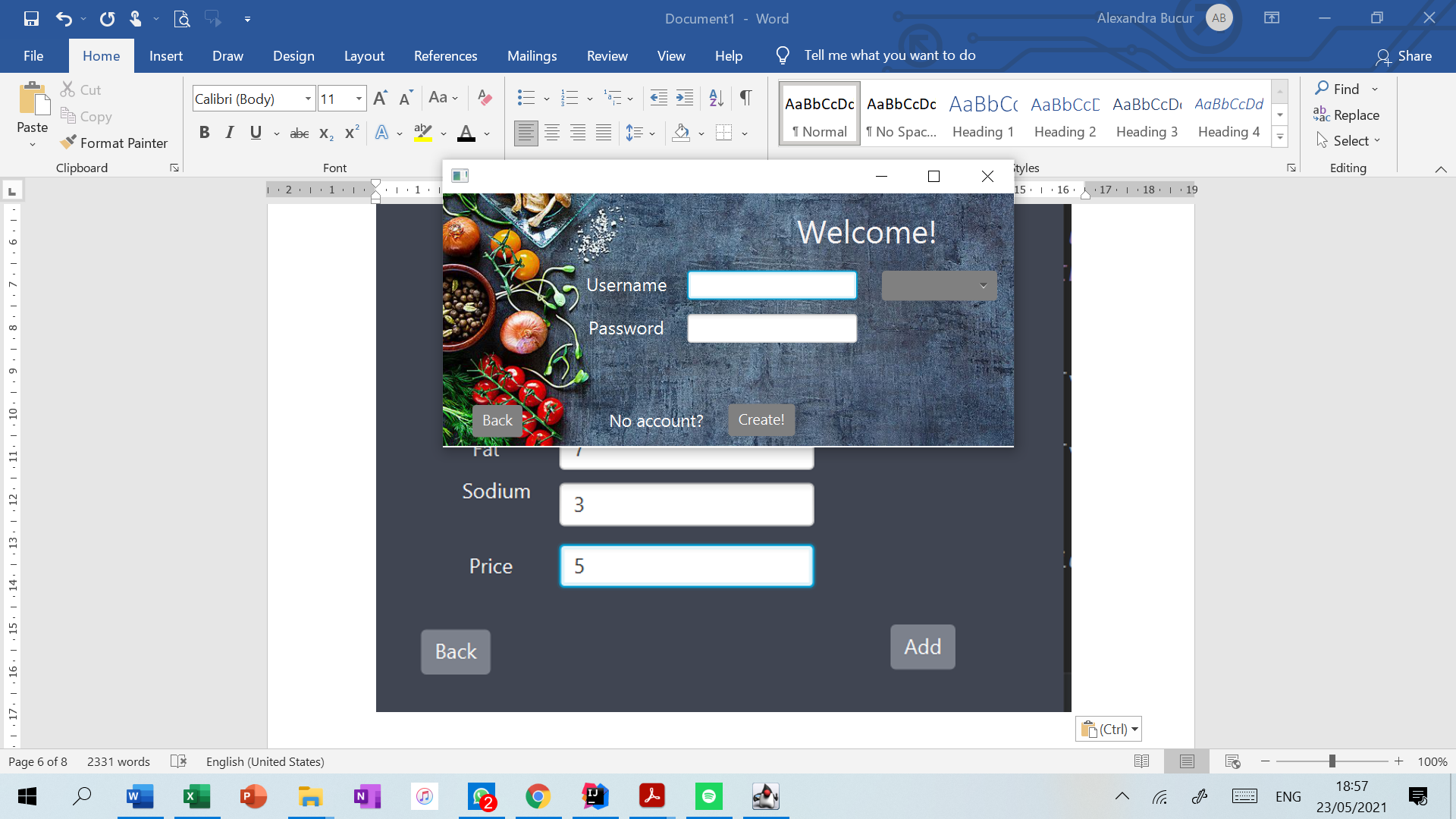
–App class which runs the start() method in order to “turn on” the application.

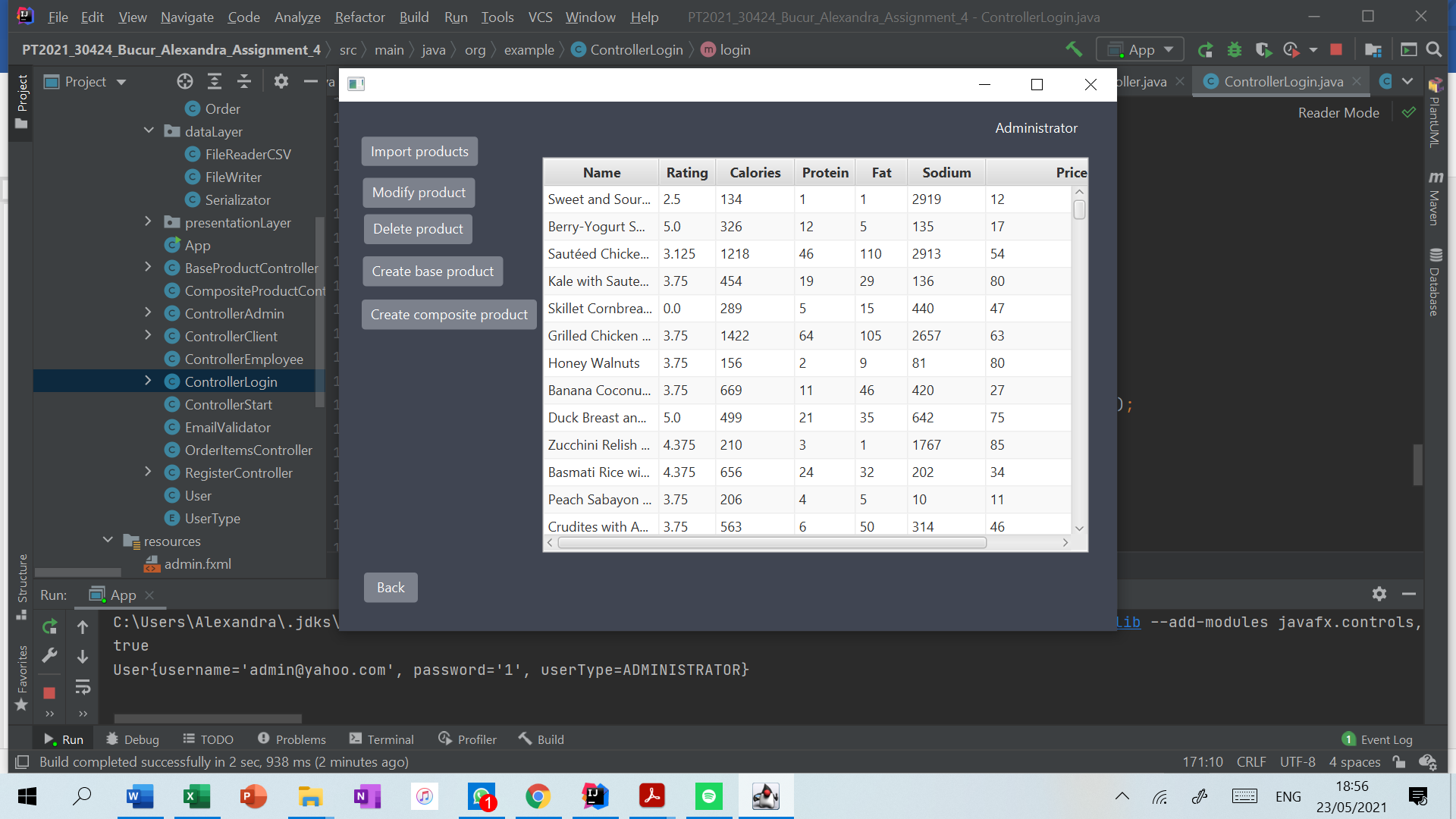
## User Interfaces

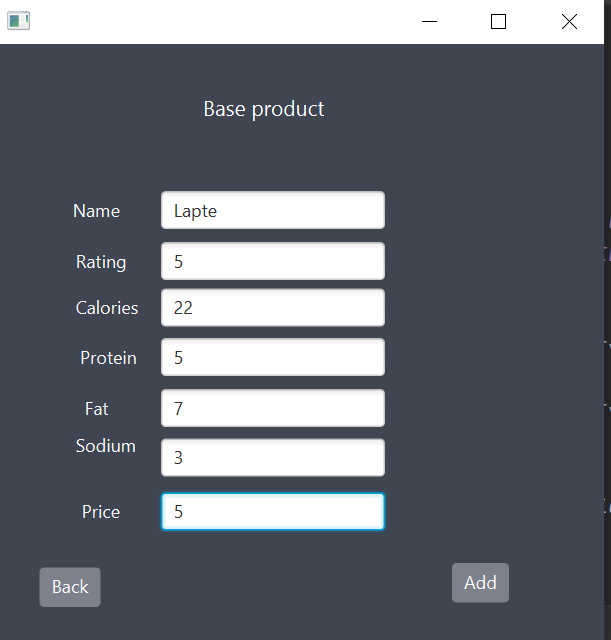


This is the first page of the application.

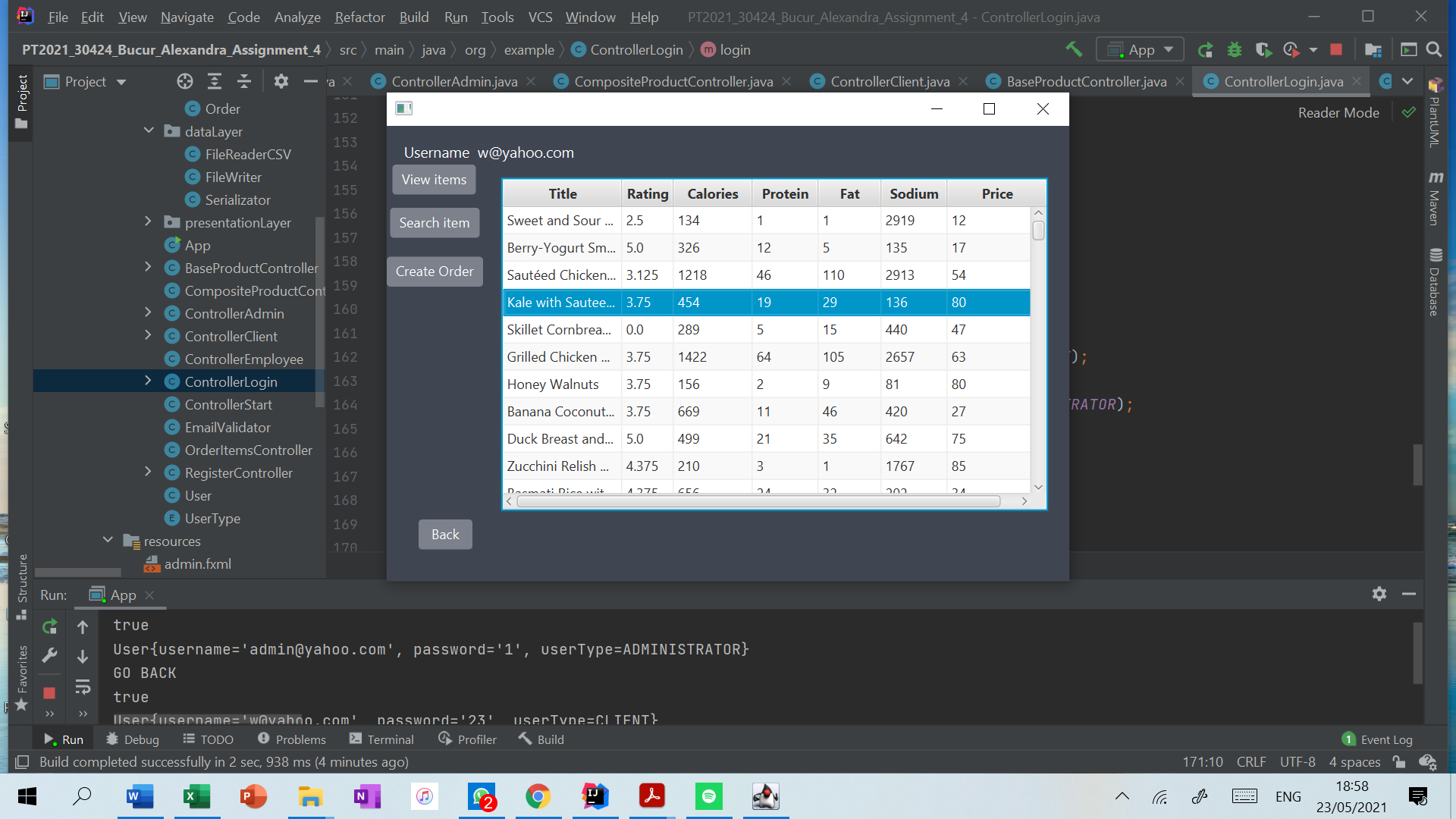
The nex one will be the one where the user cand log in into the account.

 Here a user can log on into the account, or, if a client has no account, can create a new one by clicking the designated button.

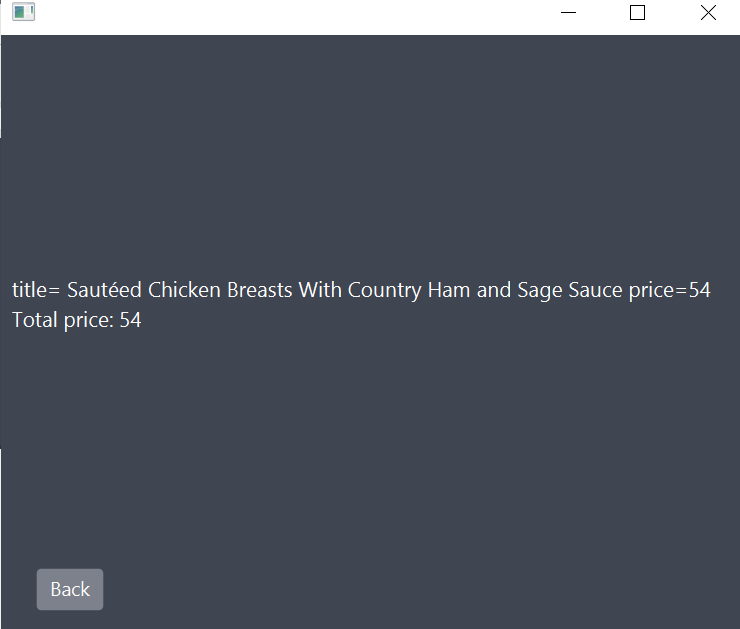
When opening the administrator pagem there are the available buttons and operations.



Here we create a new base product that will be added into the list of all products. The composite product page looks familiar, but with a table to select products that will be a part of the composite product.



There is the client page, where a client can create a new order.



Here is the page with an order created by a client.

# 4. Implementation

One of the most important methods, from the delivery service class. The find items method. This method will find all items that satisfy the conditions regarding the name, price etc.

The method uses lambda expressions and streams.

public ArrayList<MenuItem> findItems(String name, String rating, String cal, String protein, String fat, String sodium, String price) {  
 ArrayList<MenuItem> aux = items;  
 if (!name.equals("Name") && !name.equals("")) {  
 aux = (ArrayList<MenuItem>) items.stream().filter(itm -> itm.getTitle().toLowerCase().contains(name.toLowerCase())).collect(Collectors.*toList*());  
 }  
 if (!rating.equals("Rating") && !rating.equals("")) {  
 aux = (ArrayList<MenuItem>) aux.stream().filter(itm -> String.*valueOf*(itm.getRating()).equals(rating)).collect(Collectors.*toList*());  
 }  
 if (!cal.equals("Calories") && !cal.equals("")) {  
 aux = (ArrayList<MenuItem>) aux.stream().filter(itm -> String.*valueOf*(itm.getCalories()).equals(cal)).collect(Collectors.*toList*());  
 }  
 if (!protein.equals("Protein") && !protein.equals("")) {  
 aux = (ArrayList<MenuItem>) aux.stream().filter(itm -> String.*valueOf*(itm.getProtein()).equals(protein)).collect(Collectors.*toList*());  
 }  
 if (!fat.equals("Fat") && !fat.equals("")) {  
 aux = (ArrayList<MenuItem>) aux.stream().filter(itm -> String.*valueOf*(itm.getFat()).equals(fat)).collect(Collectors.*toList*());  
 }  
 if (!sodium.equals("Sodium") && !sodium.equals("")) {  
 aux = (ArrayList<MenuItem>) aux.stream().filter(itm -> String.*valueOf*(itm.getSodium()).equals(sodium)).collect(Collectors.*toList*());  
 }  
 if (!price.equals("Price") && !price.equals("")) {  
 aux = (ArrayList<MenuItem>) aux.stream().filter(itm -> String.*valueOf*(itm.getPrice()).equals(price)).collect(Collectors.*toList*());  
 }  
 return aux;

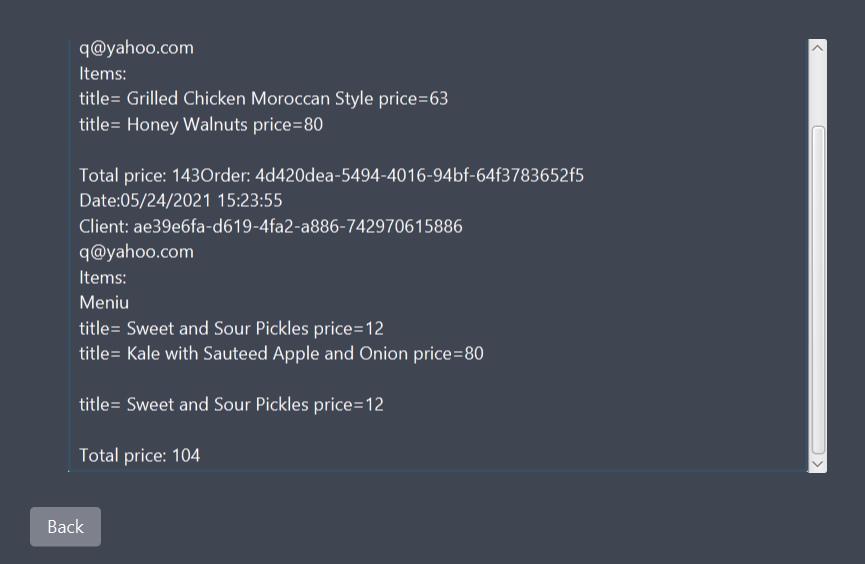
Another important method is the one that serializes/deserializes the data from the file ser.txt, but also the ones regarding reading from the .csv file

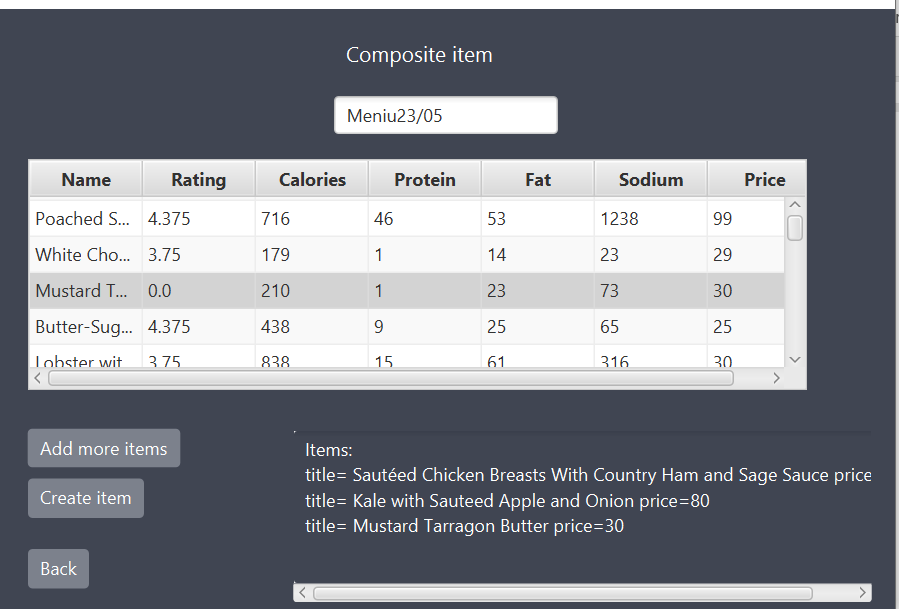
public HashSet<MenuItem> processInputFile() {  
  
 HashSet<MenuItem> inputList = new LinkedHashSet<>();  
  
 try {  
 File inputF = new File("products.csv");  
 InputStream inputFS = new FileInputStream(inputF);  
 BufferedReader br = new BufferedReader(new InputStreamReader(inputFS));  
 // skip the header of the csv  
 inputList = (HashSet<MenuItem>) br.lines().skip(1).map(mapToItem).collect(Collectors.*toSet*());  
 br.close();  
 } catch (IOException e) {  
 System.*out*.println("File products.csv not found");  
 }  
 return inputList;  
}  
  
*/\*\*  
 \* Function that creates a new product from the line that was read from the file  
 \*/*private Function<String, MenuItem> mapToItem = (line) -> {  
  
 String[] p = line.split(",");// a CSV has comma separated lines  
 BaseProduct item = new BaseProduct();  
  
 item.setTitle(p[0]);//<-- this is the first column in the csv file  
 item.setRating(Double.*parseDouble*(p[1]));  
 item.setCalories(Integer.*parseInt*(p[2]));  
 item.setProtein(Integer.*parseInt*(p[3]));  
 item.setFat(Integer.*parseInt*(p[4]));  
 item.setSodium(Integer.*parseInt*(p[5]));  
 item.setPrice(Integer.*parseInt*(p[6]));  
 return item;  
};

This will map all the items from the file to an actual item into a designated list by reading each line from the file and parsing it.

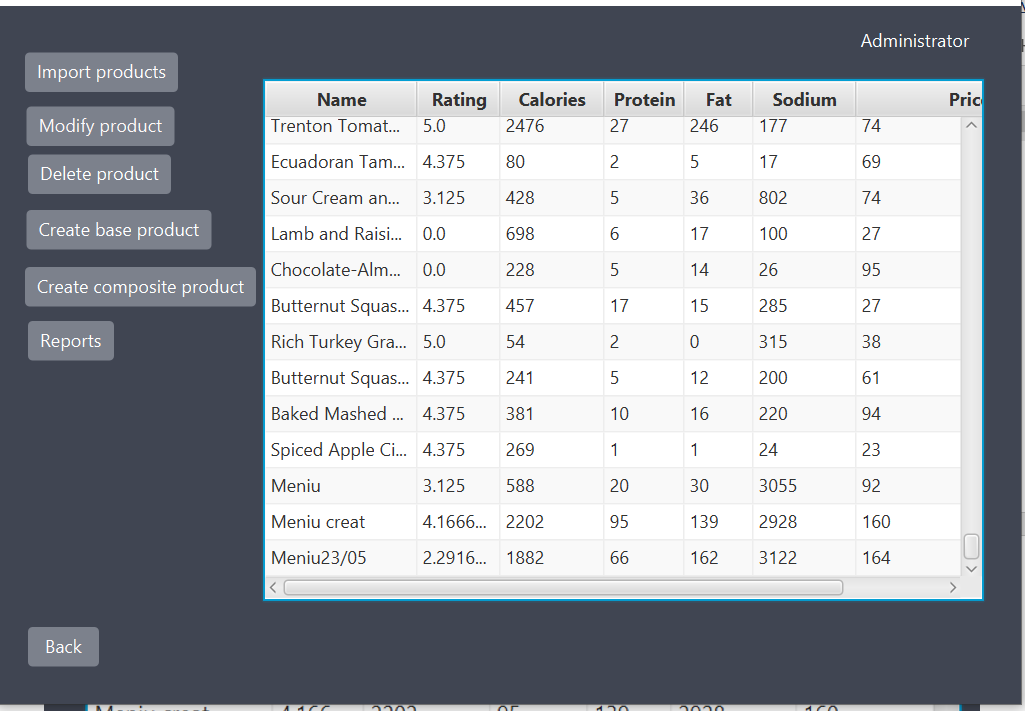
# 5. Results

Here is the employee interface after some orders have been placed, in order to demonstrate that the functionality is working.





Here is the functionality of administrator. He can add new products and then they will appear in the menu, as seen below.



# 6. Conclusions

This project was a good exercise in remembering the OOP concepts learned in the first semester, but also learning some new concepts regarding design patters, and the most important one, working with serialization. Even though I have not properly worked with this concept before, now, due to this project I now understand how they work and how to be used.

I also came to the conclusion that working by yourself and trying to solve the problems(bugs) by yourself is really helpful because the information will remain learned for further applications.

## Further implementation

• Make the interface more appealing

• Develop the application into an online shopping site

• Create a database

# 7. Bibliography

* 1. Lambda expressions and stream processing https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html

https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html

https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8-streams.html https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/ https://howtodoinjava.com/java8/java-stream-distinct-examples/

* 1. Java serialization o http://www.tutorialspoint.com/java/java\_serialization.htm https://www.baeldung.com/java-serialization

https://www.geeksforgeeks.org/serialization-in-java/

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

* 1. Java HashMap

http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html

* 1. Java assert

http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html http://javarevisited.blogspot.ro/2012/01/what-is-assertion-in-java-java.html <http://stackoverflow.com/questions/11415160/how-to-enable-the-java-keyword-assert-in-eclipse-program-wise>

https://intellij-support.jetbrains.com/hc/en-us/community/posts/207014815-How-to-enable-assert https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#tag

* 1. JAVADOC https://www.baeldung.com/javadoc
  2. JavaFX

https://docs.oracle.com/javafx/2/ui\_controls/choice-box.htm https://docs.oracle.com/javafx/2/ui\_controls/table-view.htm

www.stackoverflow.com