**Food Delivery Management System**



Teacher: Ioan Salomie

Teacher Assistant: Ciprian Stan

Student: Bodea Razvan-Marius

Group: English\_30422

Year: 2021

1. **Assignment 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).

• Generate reports about the performed

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

Why create a food delivery management system?

This project mimics a real life need of this kind of app. Having a food delivery management system is vital in this day and age in order to have a functional restaurant. An app like this simplifies a lot the process of ordering and delivering food for the restaurant staff and also for the clients. The client can simply see a list of all the products and create an order with what he needs. The restaurant staff can manage what is in the list, see reports about the restaurant and orders, have a list of all the users, see all the orders so they can be delivered as soon as possible.

1. **Problem analysis, modeling, scenarios, use cases**
   1. Analysis

Have a database for storing information about the 3 types of users (admin,client,employee).

Have a security system for the passwords (Hashing the passwords).

Create a suitable design of the classes.

Add the functionalities required (see the task above).

Use serialization.

Use stream processing and lambda expressions.

Create a bill for the orders.

* 1. Modeling

I created different interface pages for all types of users that they can reach if they enter a valid account.

The normal users can register to the restaurant. Staff members cannot register (if used in a real case, they would get their account and password that is already stored in the database)

I used hashing to secure the passwords so that if someone breaks in the database, they cannot see the passwords of all the users. (This can be further done to the personal information, but my database does not store sensitive information about the users).

The process of ordering food for the client should be as easy as possible and intuitive. They should get a preview of what they ordered and the price before confirming the order.

The admin can edit the menu and add composite products.

* 1. Scenarios & Use Cases

There are 3 types of users: admin, employee, client.

The client can log in by using a correct username and password. If they do not have an account, they can register and then use the registered account. They can see a list of all the products they can order, they can filter these items based on specific information and can enter the ids of the products they want to order. After placing an order, a preview of the order (bill) will be displayed and after confirming the order, a txt file with the bill is generated and the order is saved.

The employee can log in and see all the orders.

The admin can edit the list of products: add/modify/delete/find specific products or create a menu from already existing products.

The admin can see list of all the orders and filter them based on some specific information.

The buttons are labeled so that the user can figure out what they do and what they are supposed to do in order to use the app accordingly, but I also included some verifications and error messages based on the mistakes. For the cases that are not handled, the app will not break but the order is not saved and must be redone.

1. **Design (design decisions, UML Diagrams, data structures, class design, interfaces, relationship packages, algorithms, user interface)**

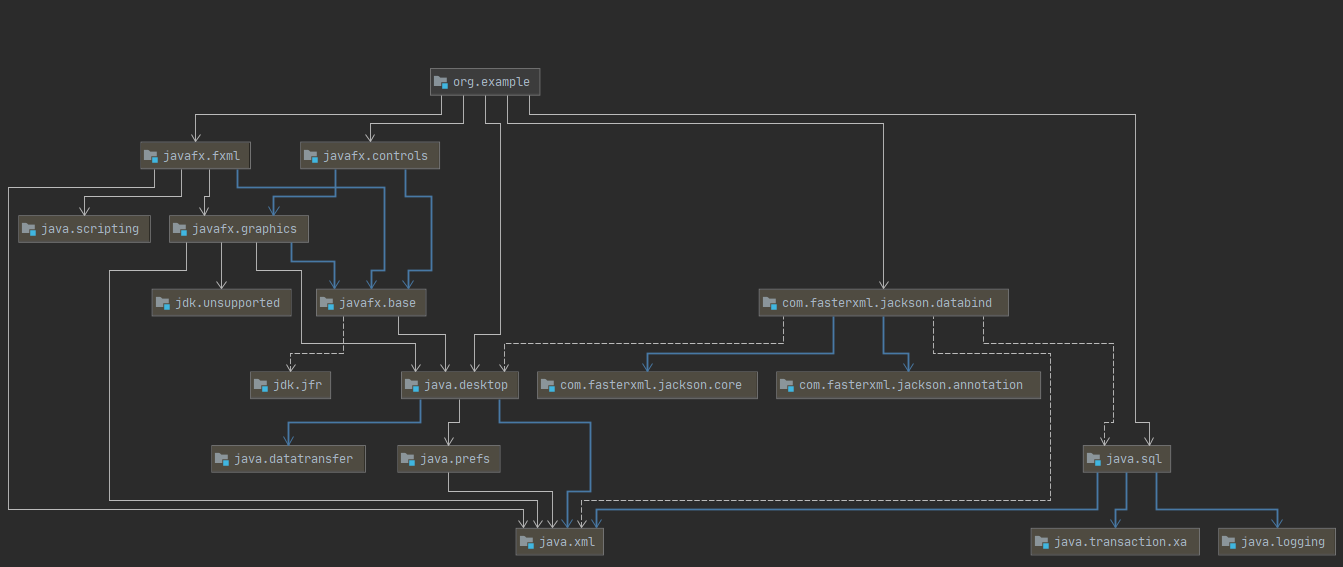
3.1 Design decisions

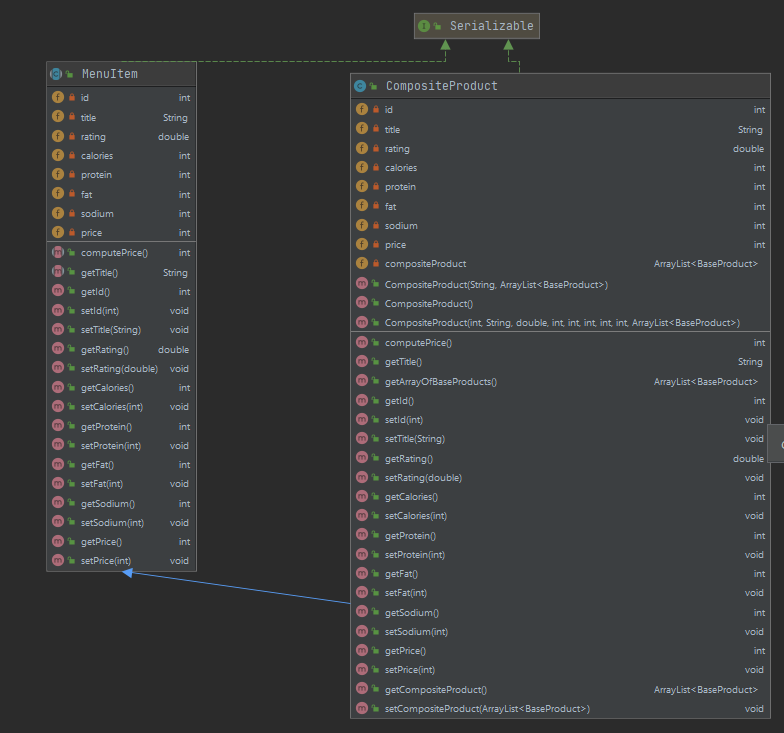
The base of the project is the architecture presented in the assignment presentation.

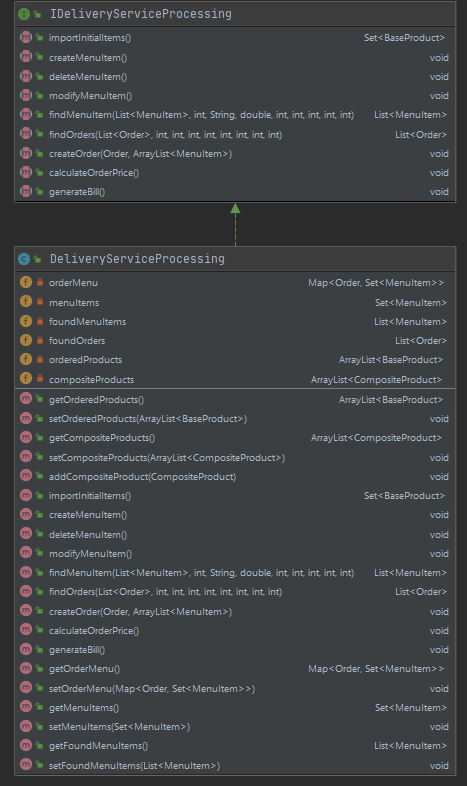
I also added some classes that are related to the database access.

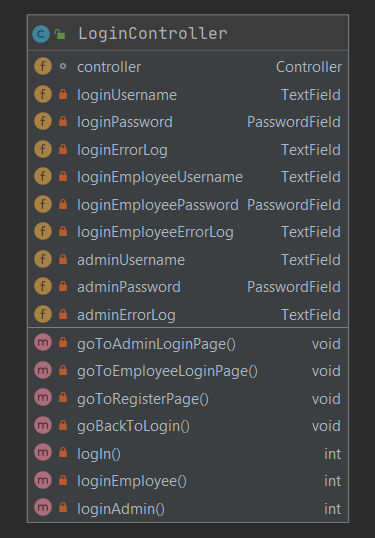
The user interface should be as easy to use as possible and have all the required functionalities.

A text field is used only for displaying error logs so that the user can see what was the mistake if the selected action did not occur.









3.2 UML Diagrams & Classes

The DAO classes are used for interacting with the database.

The Serialization class handles serialization and deserialization of data form the program

The Hasing calss handles hasing the passwords when a new account is created and also for comparing the password at login with the stored password for that account.

ConnectionFactory handles connecting to the database.

The ..BLL classes are used to access the methods for different models in the app (User, Admin, Employee).

The Controller class handles the interactions between the user and the app

The Resources file contains the elements of UI

3.3 Data Structures/data types & other elements

Map(Order, List<MenuItems>) maps the orders and the products specific to that order together.

ObservableLists<MenuItems> are used for interacting with the user interface.

Set<Item> does not allow duplicates. This is important for the initial populating of the tables.

LocalDateTime is used to generate information about the orders.

3.4 Class Design, Interfaces, Packages

The class design respects the design suggested in the presentation, but I also added some classes related to the database. The client, admin, employee are classes that have the same fields as their corresponding database entries so that we can perform operations on them and work with the data we stored in the database about them. The ...DAO classes are used for accessing the database and the …BLL classes are used to call the functions for the class we want to perform operations to.

The serialization class handles the serialization and deserialization of data.

3.5 Algorithms

The hasing algorithm I used is inspired by the cryptocurrency hashing algorithm but it is a simplified version.

The main idea of the app is to have buttons in the UI that , when pressed, will trigger a set of events. (ex : client places an order -> we create the order, we create a bill, we add the order to the data of the program, we serialize the information)

3.6 User Interface

The user interface should be as easy to use and as intuitive as possible. This app uses a few pages that are specific for each type of user and contains buttons for actions specific for that user.

The admin has the most complex page. There are fields for every attribute of a BaseProduct that the admin can complete and then use different commands on the data entered. The admin can add a new product, delete a product, search for a specific product, create a Menu that is composed from the selected baseProducts. The amin can also see a list of all the orders and filter them based on some criteria lie hour, minute, day of order.

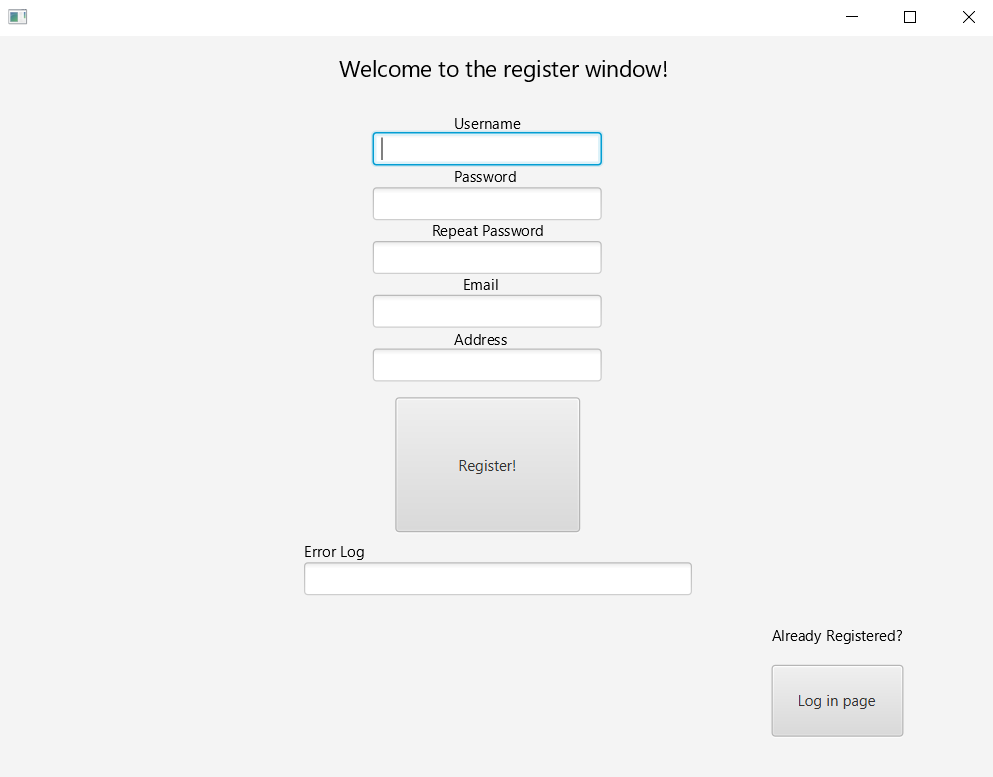
The client has the next page in complexity. He also has fields for all product characteristics for filtering them based on what he wants. A textArea where the user imputs the ids of the products he wants to order is also present. After the user presses the “Order” button, a preview of the bill is presented to him and then he can confirm the order. If he confirms, a bill is generated and the order is added in the memory of the program.

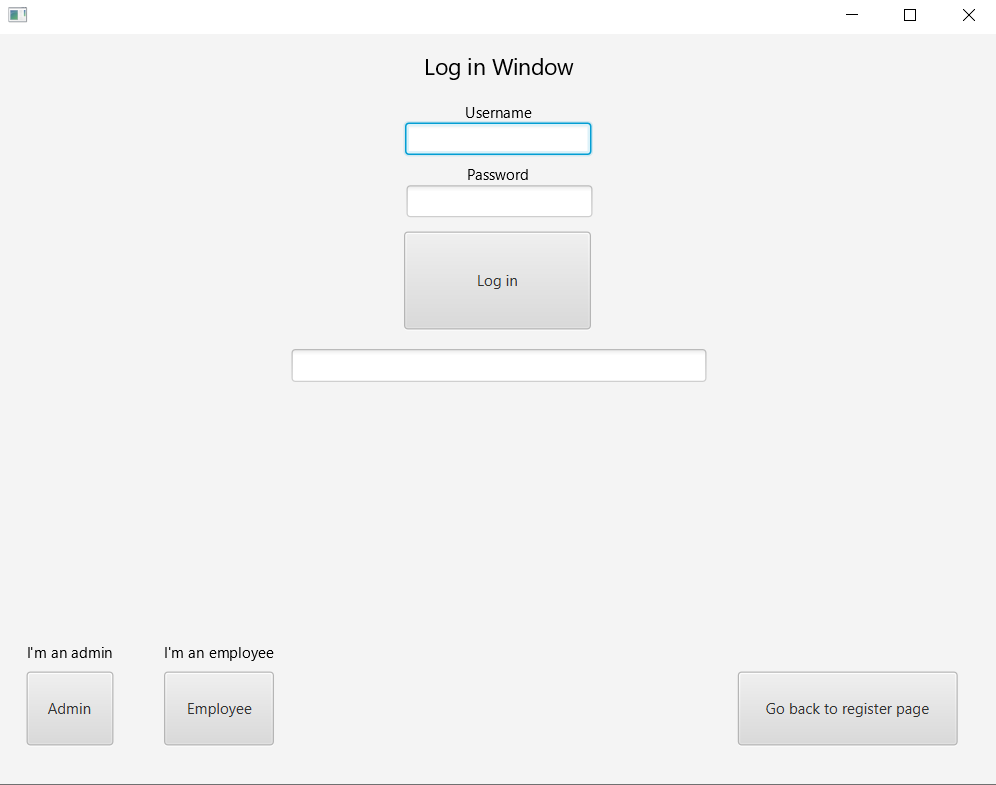
The employee can see a list of all the orders and filter them.

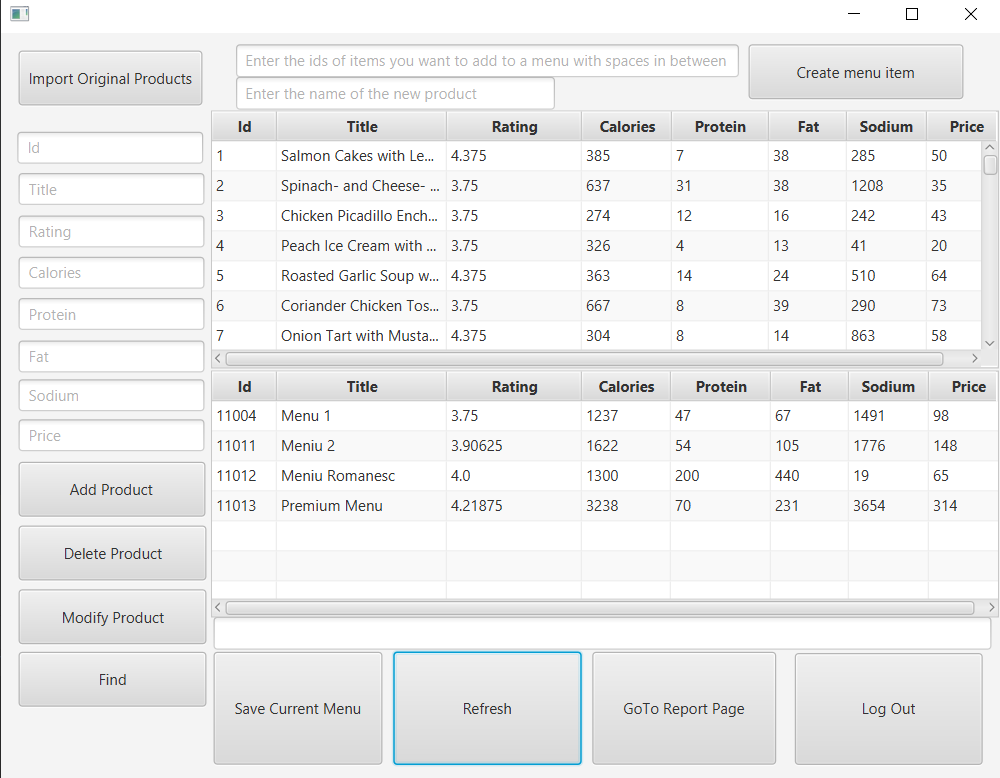
There are also a page for login and register. The register page requires some extra information and also has 2 fields for the password for confirmation. After the register button is pressed, if all the information is valid and no account with the same username exists, the account and information are stored in a database for further use (login). The password is stored as an encoded version, for extra security. The idea of the hashcode is to be easy to compute but hard to break if you know the code.

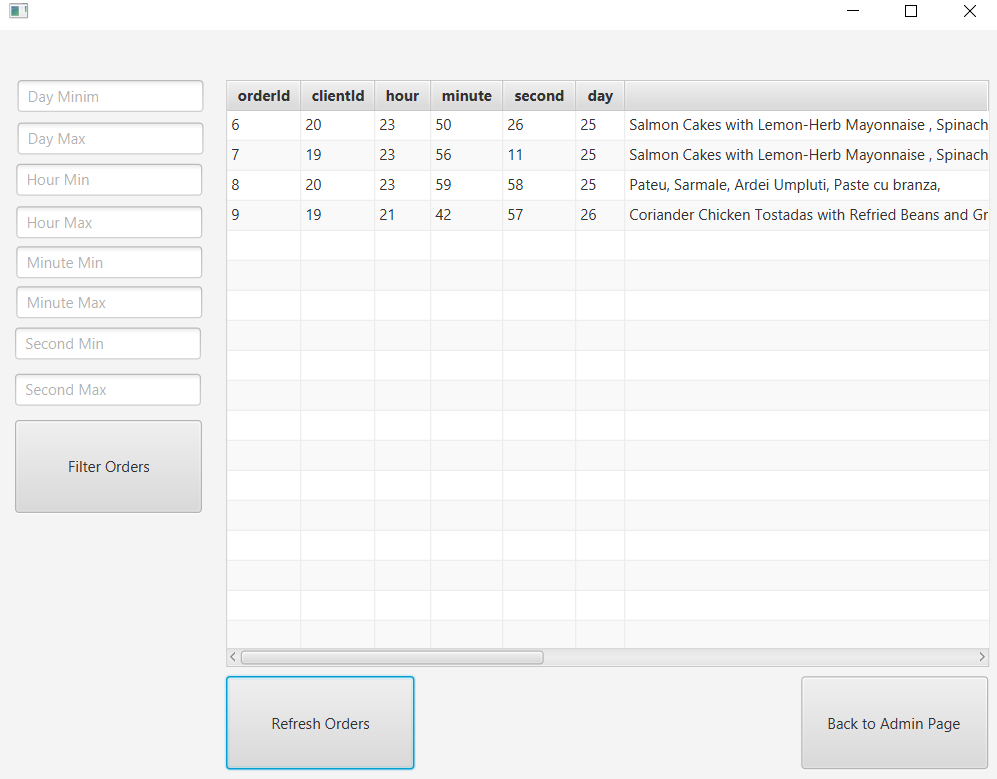
The login page presents the user with just a username and a password field. When the login button is pressed, the information entered is compared with the information stored in the database. If it is valid, the user is presented with his specific page.

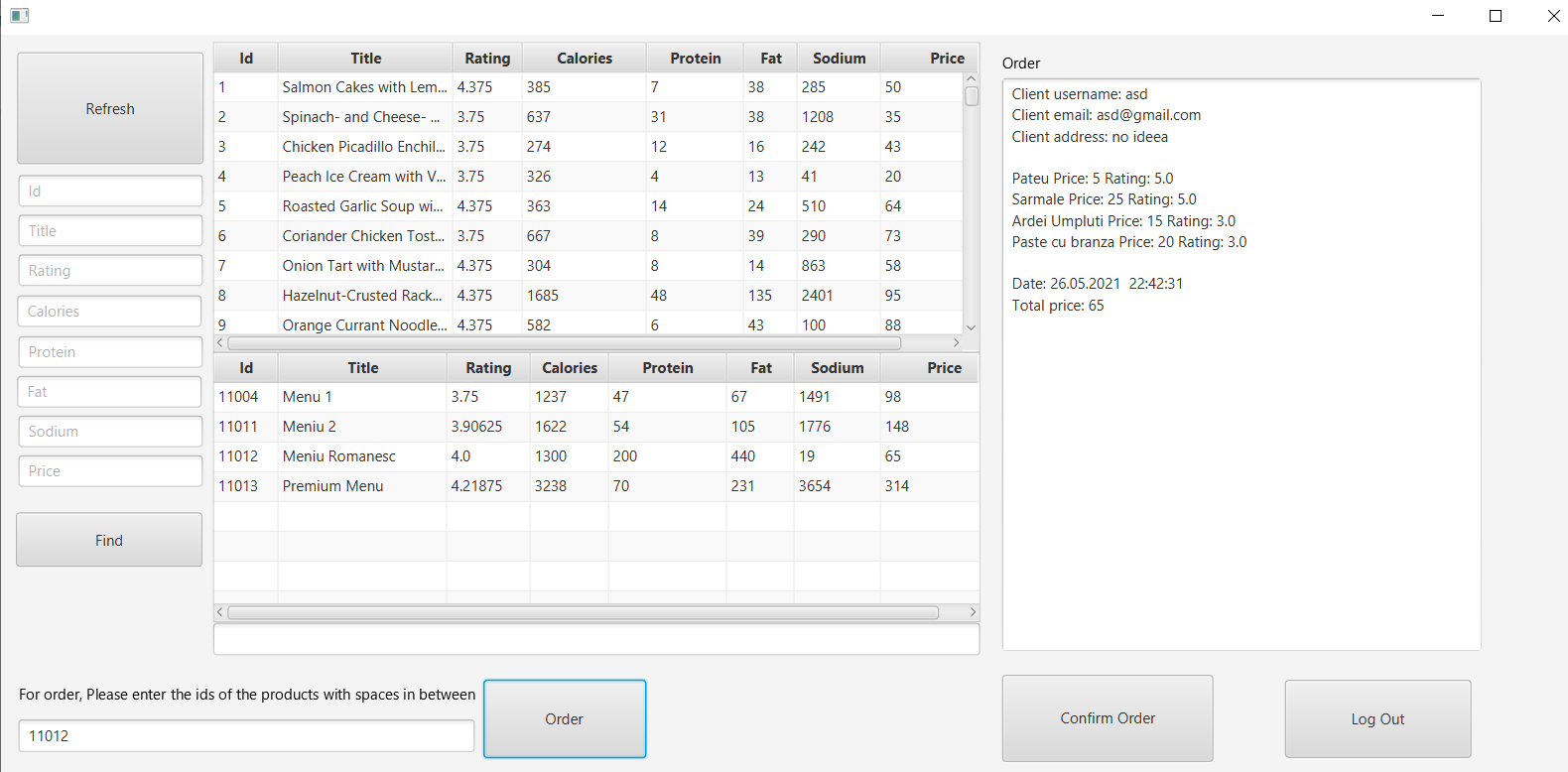
A logout and goBack button are also available through all the pages to assure a good flow.





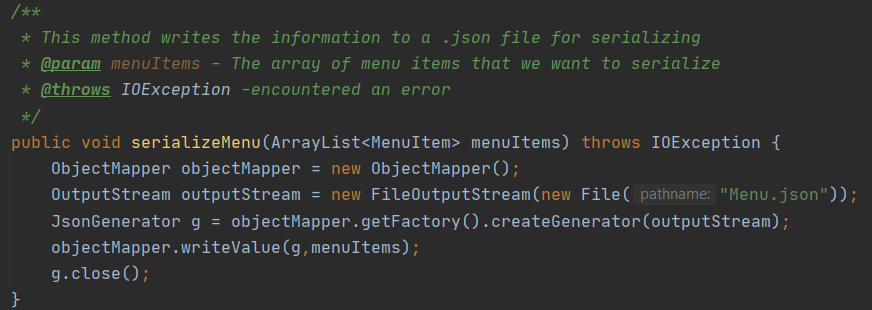


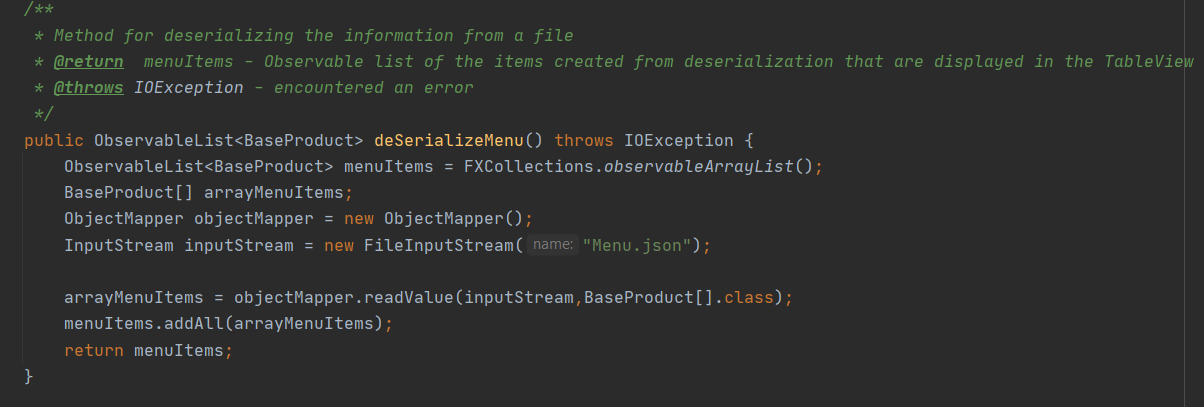


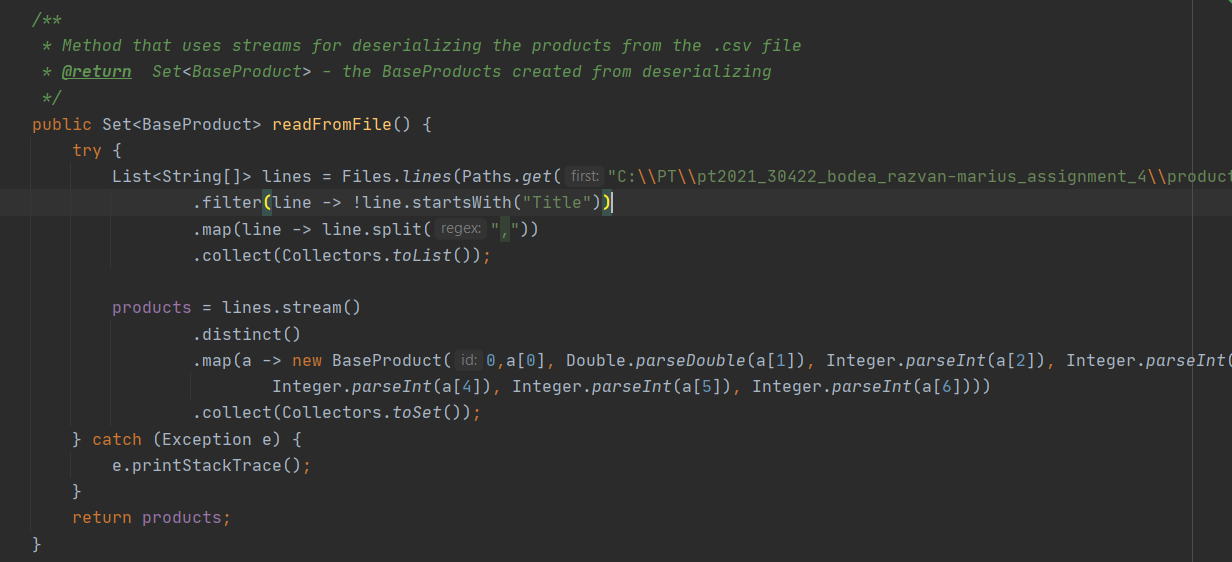


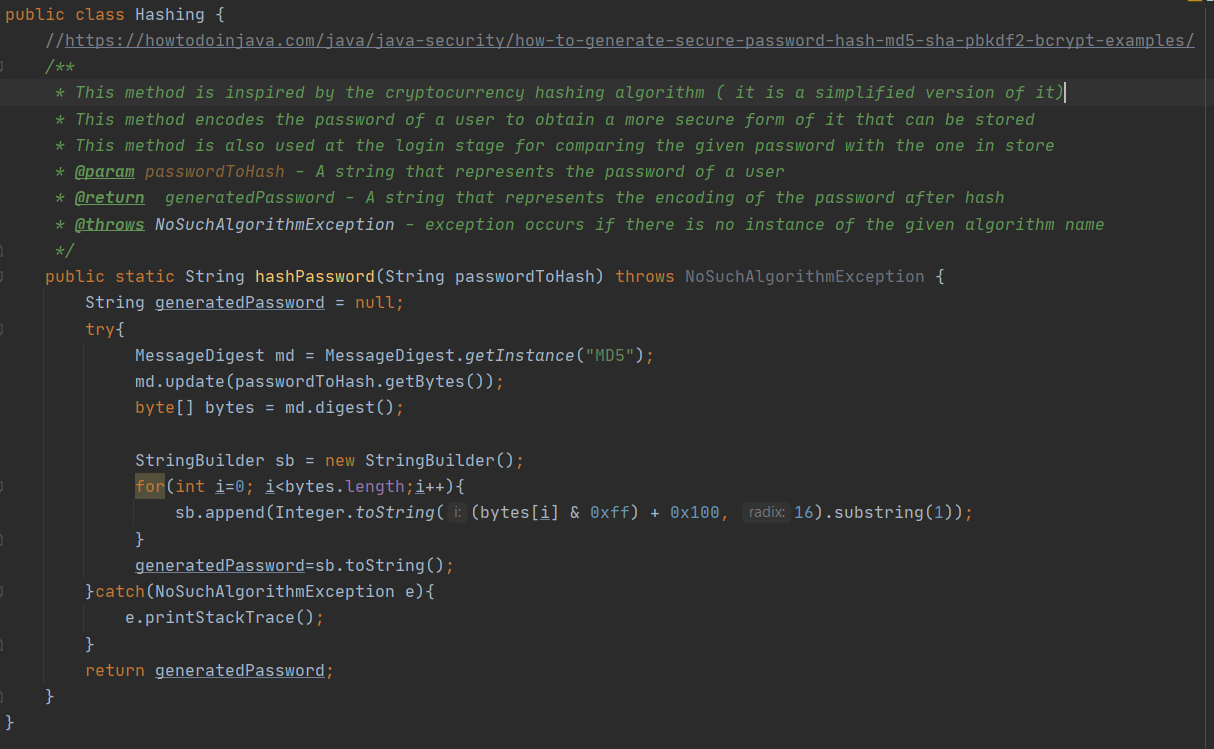
1. **Implementation**

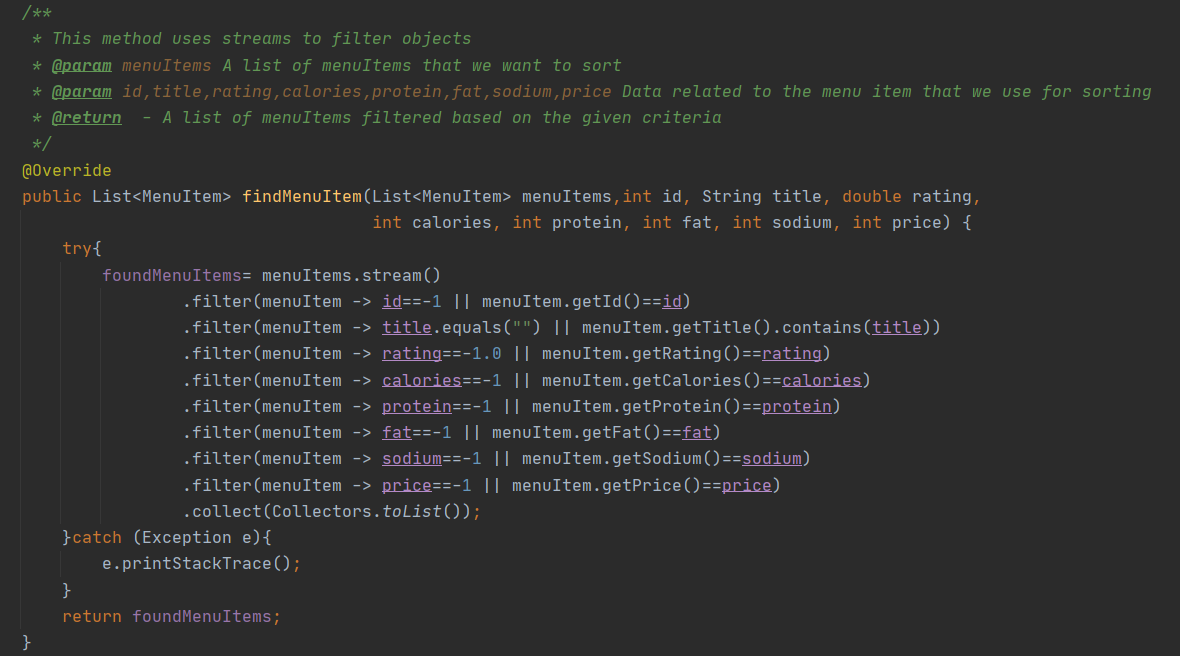
Here I will put some screenshots of the code with the comments that explain it pretty well.

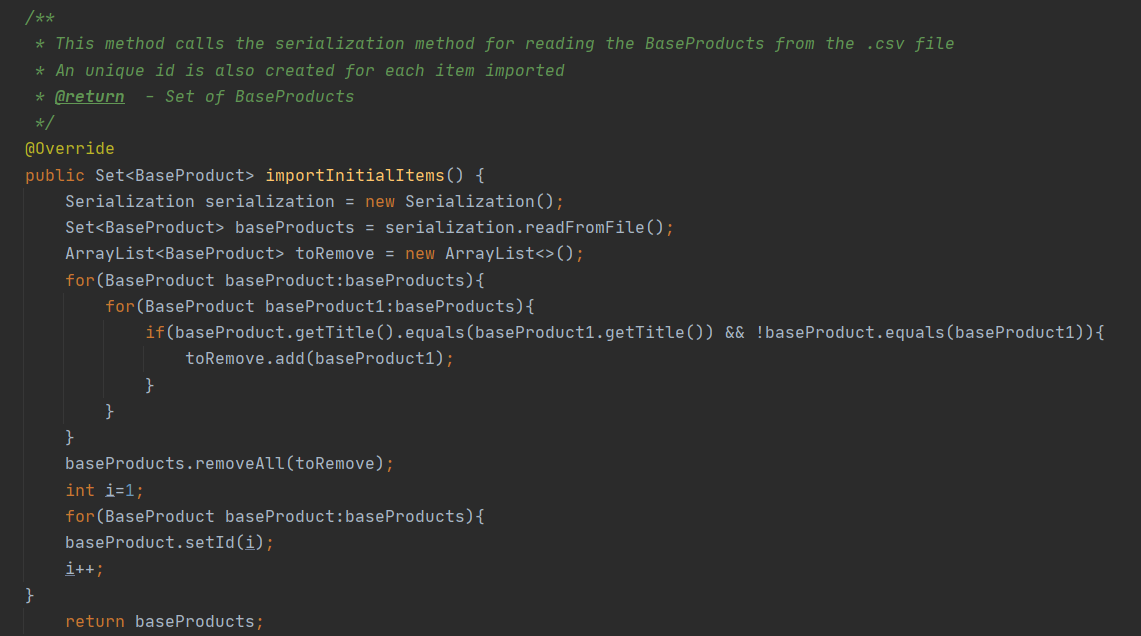


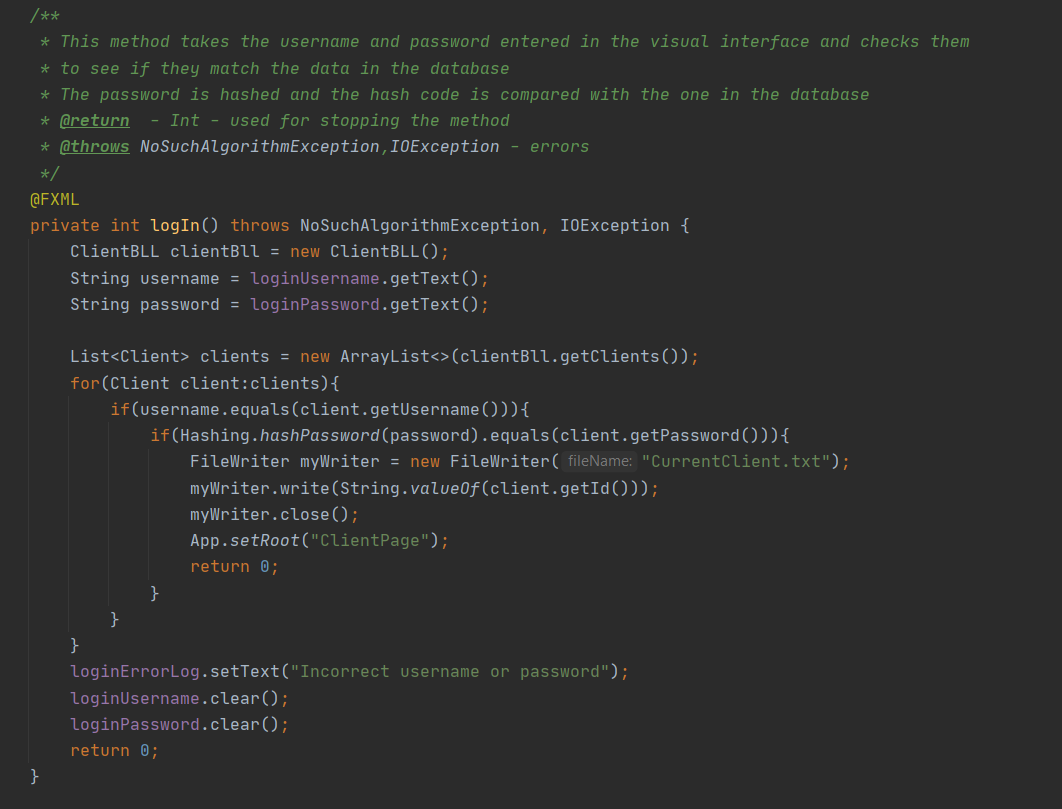


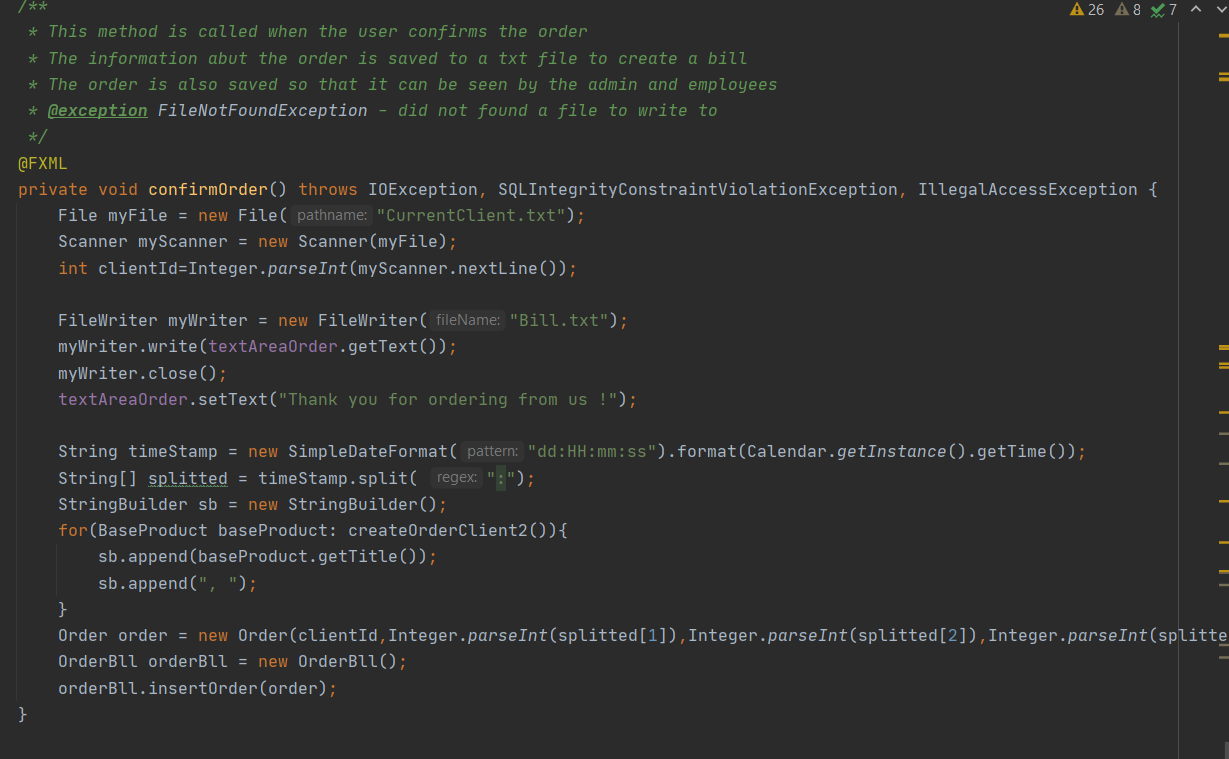


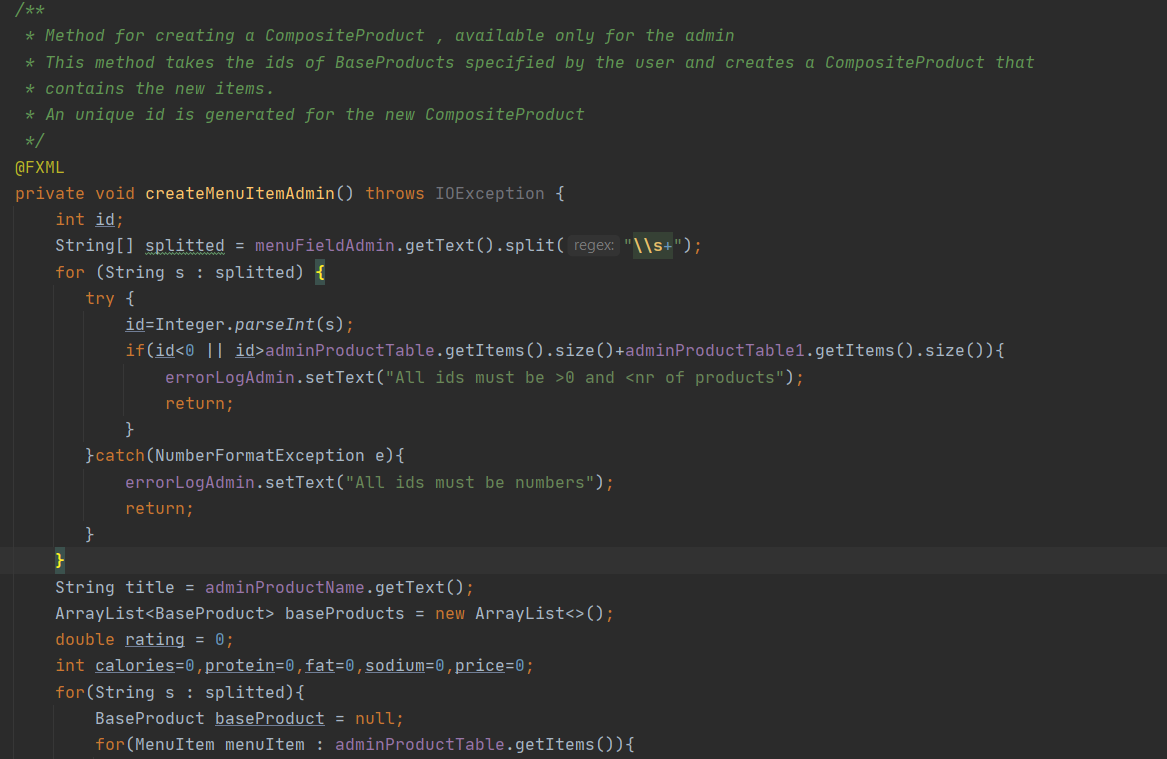














1. **Results**

At the end of this project, I managed to realize the desired functionalities of the program.

The admin can add new products to the menu, can search/delete/modify existing products and can also create a menu from products that are already in the menu. The admin also has access to a reports page where he can see all the orders and filter them based on some criteria (like day of order, time of order, minute, hour). The employee can see all the orders and filter them. The client can log in, filter the products and order products from the list. A report and a bill for the order will be created after he confirms the order. Before ordering, the client gets a preview of the bill.

I tested this program in some cases and I tried to handle all the exceptions and problems that could occur. Like the client entering words where he should enter numbers etc.

1. **Conclusions**

This project was a great learning experience because I used a lot of what I learned so far during the lecture and lab of this subject. On top of what I used in the previous projects, I also used serialization, stream processing and lambda expressions. Serialization was used to store in a text file the data of the program so it can be deserialized and used at the start of the program or when it is needed. Stream processing is an interesting way to process arrays in order to obtain the desired data (like using .filter() to get only a specific part of the array list or .distinct() to obtain a set of the array). I also used a database for the accounts and the register system and learned about hashing(I used a hashing algorithm that is a simplified version of the cryptocurrency hashing algorithm).

This project could use some extra work in order to be used as a real food ordering management system (Maybe I will work on it in the vacation).

1. **Bibliography**

--The documentation for the project, and the course

Connect to MySql from a Java application

[Introduction to JDBC | Baeldung](https://www.baeldung.com/java-jdbc)

[Connect to MySQL with JDBC driver - Mkyong.com](https://mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/)

Connect to MySql from a Java application

[Layers of a Standard Enterprise Application - DZone Java](https://dzone.com/articles/layers-standard-enterprise)

JAVADOC

[Introduction to JavaDoc | Baeldung](https://www.baeldung.com/javadoc)

SQL dump file generation

[MySQL :: MySQL Workbench Manual :: 6.5.2 SQL Data Export and Import Wizard](https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html)

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://www.oracle.com/technical-resources/articles/java/ma14-java-se-8-%20streams.html)

<https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/>

<https://howtodoinjava.com/java8/java-stream-distinct-examples/>

Java serialization

[Java - Serialization - Tutorialspoint](https://www.tutorialspoint.com/java/java_serialization.htm)

[Introduction to Java Serialization | Baeldung](https://www.baeldung.com/java-serialization)

[Serialization and Deserialization in Java with Example - GeeksforGeeks](https://www.geeksforgeeks.org/serialization-in-java/)

[Serializable (Java Platform SE 8 ) (oracle.com)](https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html)

Java HashMap

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

Java assert

<http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html>

Adding custom tags to Javadoc

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