Delivery Service Application

Nadu Laura-Andreea
Group 30421

Table of Contents

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**.

Sub-Objectives

O1. Analyse the problem and identify the requirements

O2. Design the delivery service for the catering company

O3. Implement the ordering process

O4. Implement by using the Observer Pattern and Design by Contract

O5. Implement User Interfaces for the login, client, administrator and employee

O6. Test the delivery service

O7. Save the current state of the delivery service into a serialized file

Problem analysis, modelling, scenarios, use cases

Analysing the problem

Delivery systems are a growing industry in the world, especially considering the current living conditions. A delivery system should be able to the used by every type of user involved into the delivery process.

The types of users involved are:

- ✓ The administrators which are in charge of managing the products available for the clients and for generating reports about the order statistics
- ✓ The employees receive notifications about the new orders which should be prepared
- ✓ The clients can order the products they wish

Taking these into consideration, a delivery system is required in order to keep track of the order and product evolutions and keep every user involved updated with the new changes made.

Functional Requirements

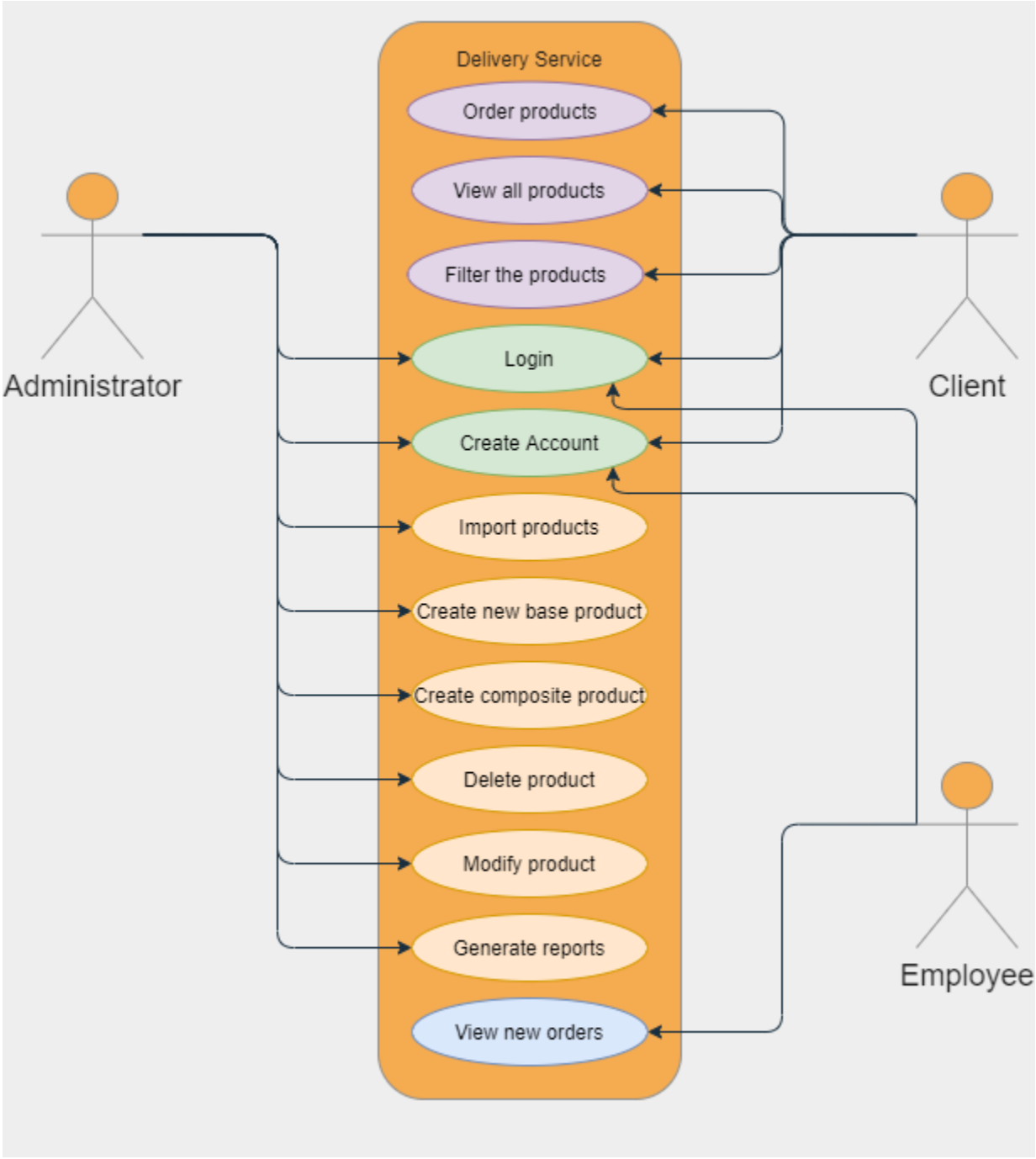
- ✓ All the users should be able to create an account and to login into the said account
- ✓ The administrator should be able to import the initial set of products which will populate the menu from a .csv file
- ✓ The administrator should be able to add/delete/modify the base products and create new composed products of several other products
- ✓ 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 should be able to see a list of all the products from the menu
- ✓ The client should be able to search for products based on one or multiple criteria such as keyword (e.g. "soup"), rating, number of calories/proteins/fats/sodium/price
- ✓ The client should be able to create an order consisting of several products

- ✓ The employee should be s notified each time a new order is performed by a client so that it can prepare the delivery of the ordered dishes

Non-functional Requirements

- ✓ The delivery system should be intuitive and easy to use by any user, regardless of their knowledge
- ✓ The delivery system should verify the correctness of the input data inserted by the user

Use Case Diagram



Use Case: Create Account
Actor: Administrator, Client, Employee
Success Scenario:
- The user selects the "Create New Account" operation
- The user introduces valid data for the account (username, password and role)
- Confirms creation and the new account is created

Alternative Scenario:
- The user introduces invalid data which could mean that the fields either contain illegal characters, they contain too few characters or that the username is already taken.
- An error message is displayed
- The user may try again

Use Case: Login
Actor: Administrator, Client, Employee
Success Scenario:
- The user selects the "Login" operation
- The user introduces valid data for the account (username and password)
- Confirms login and the corresponding window opens

Alternative Scenario:
- The user introduces invalid data which could mean that the user with the corresponding username could not be found.
- An error message is displayed
- The user may try again

Use Case: Order Products
Actor: Client
Success Scenario:
- The user selects the products he/she wishes to order than he presses the order button
- The order is created and signaled to the employees

Alternative Scenario:
- The user tries to create an order with no items

Use Case: Filter Products
Actor: Client
Success Scenario:
- The user introduces data about the filtration criterion and then proceeds to filter the product list
- The reduced list is shown to the user

Alternative Scenario 1:
- The user introduces invalid data into the filter
- An error message is displayed
- He/She may try again to introduce the data

Alternative Scenario 2:
- The user imposes too strict criteria and the list would be empty

Use Case: Import Products
Actor: Administrator
Success Scenario:
- The user asks for the product list to be imported
- The user can now see all the products

Use Case: Create new Base Product
Actor: Administrator
Success Scenario:
- The user introduces valid data about the new base product

- The user confirms the creation of the new product
- The product is added to the products list

Alternative Scenario:
- The user inserts invalid data about the product
- An error message is displayed
- He/She may try again

Alternative Scenario:
- The user tries to create a product which already exists
- An error message is displayed

Use Case: Create new Composite Product
Actor: Administrator
Success Scenario:
- The user selects the products for the new compound product
- The user confirms the creation of the new product
- The product is added to the products list

Alternative Scenario:
- The user selects less than 2 products which means that the compound product does not make sense
- An error message is displayed
- He/She may try again

Alternative Scenario:
- The user tries to create a product which already exists
- An error message is displayed
- He/She may try again

Use Case: Delete Product
Actor: Administrator
Success Scenario:
- The user selects the products to be deleted
- The user confirms the operation
- The product is deleted from the products list

Alternative Scenario:
- The user selects no product which means that the operation does not make sense
- An error message is displayed
- He/She may try again

Use Case: Modify Product
Actor: Administrator
Success Scenario:
- The user introduces the new data of the product
- The user confirms the operation
- The product is modified in the products list

Alternative Scenario:
- The user inserts invalid data about the product
- An error message is displayed
- He/She may try again

Use Case: Generate Reports
Actor: Administrator
Success Scenario:
- The user introduces the filtration criteria
- The user confirms the operation
- The report is being generated

Alternative Scenario:

- The user inserts invalid data about for the filtration
- An error message is displayed
- He/She may try again

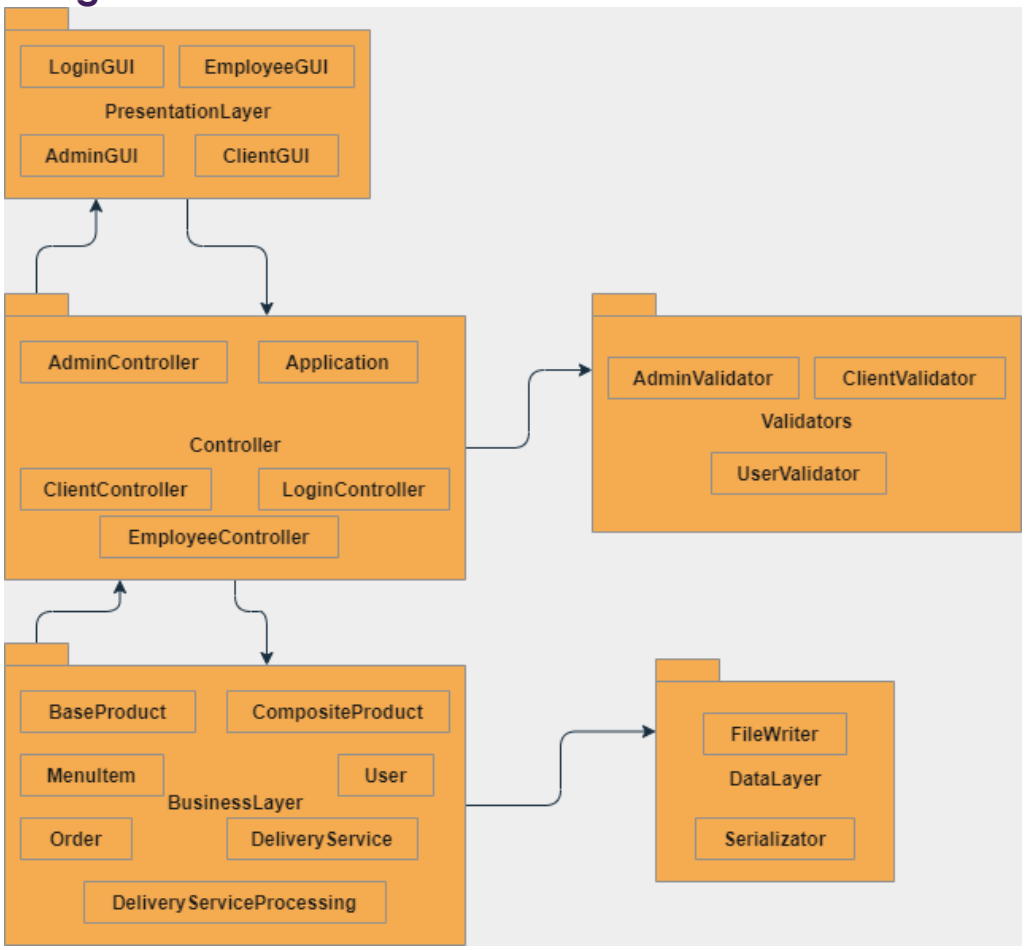
Use Case: View New Orders
Actor: Employee
Success Scenario:

- The user views in real time the newly made orders

Design Decisions

Overall System Design

Package Diagram



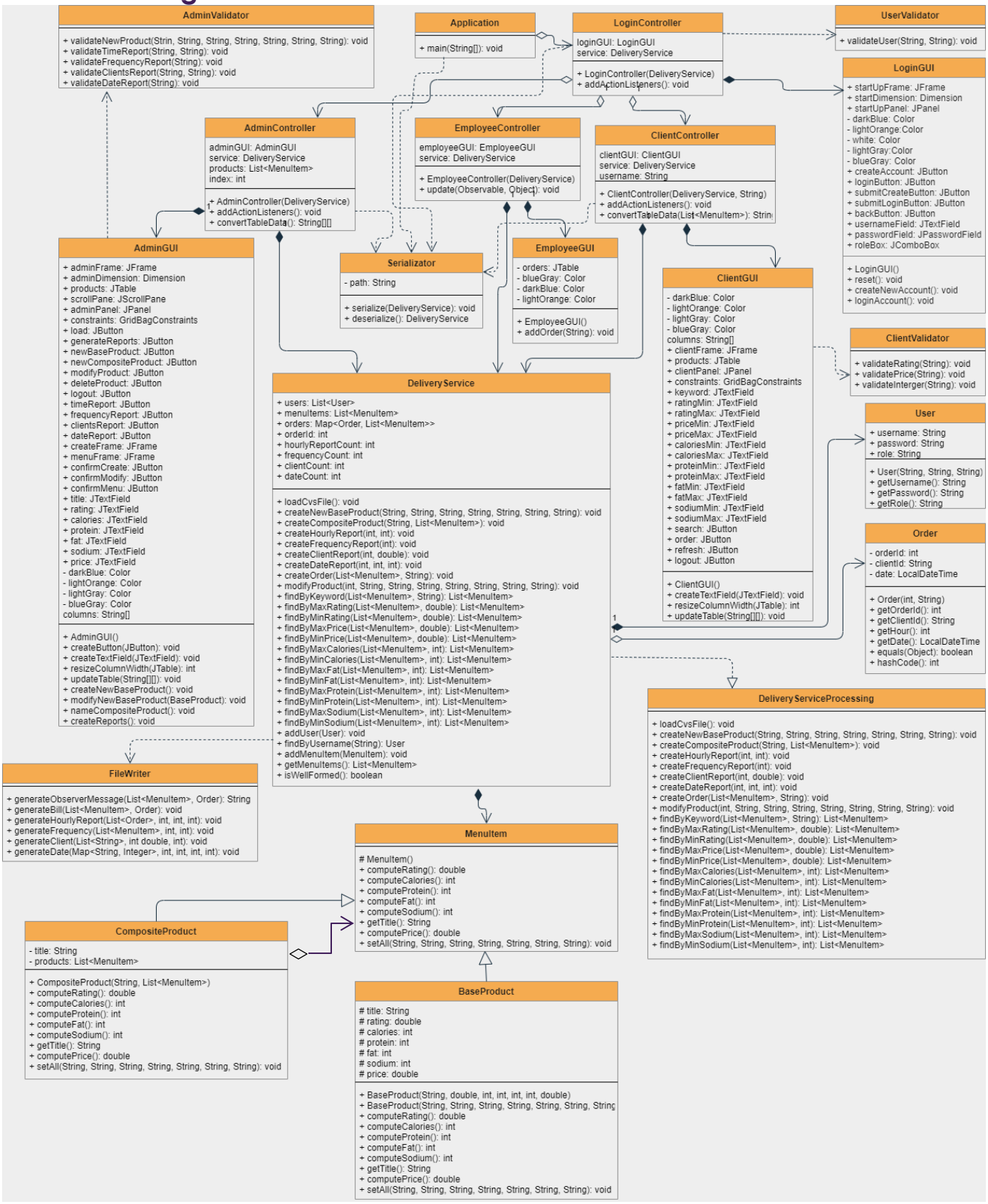
Layered Architecture

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.

In the case of the Delivery Service Application cases, the business layer and persistence layer are combined into a single business layer, particularly when the persistence logic is embedded within the business layer components. Thus, smaller applications have only three layers, whereas larger and more complex business applications may contain five or more layers.

Another modification brought to the classical "Layered Architectural" pattern is the replacement of the Database Layer with a DataLayer. That is because this application does not use a database, because the information is stored into serialized files.

UML Class Diagram



Data Structures

One of the data structures used is the ArrayList. ArrayList, a member of the Java Collection Framework, was used mostly to store the list of users and the avaukable items on the menu. It is a dynamic data structure whose size changes based on the number of elements stored. By using these data structures, access was granted to methods that allow access to any element in the list, which can be just returned or removed, or to methods that return the size of the array so we won't have to store it in a special variable.

Beside this structure, another collection used is the Map<Key, Value> collection, also a member of the Java Collection Framework. For this structure, the keys were the orders, while the values were the items belonging to the said order. This collection is used to access the memory by the value of the order given as key, just as for an ArrayList we would use integers.

Class Design

Non-user Interface Classes

MenuItem is an abstract class defining the general methods of a product. The class is abstract, because each method inside would be overridden anyway by the child classes of the MenuItem class. These children are BaseProduct and CompositeProduct. The BaseProduct defines the most simple instance of a product as an item which can no longer be split into sub-items. The CompositeProduct is a product made from other products, either simple or compound ones.

While the compute methods of the base products are just simple getters for the specific fields of a simple product, the overridden methods of a compound product compute the statistics based on the items it is made of.

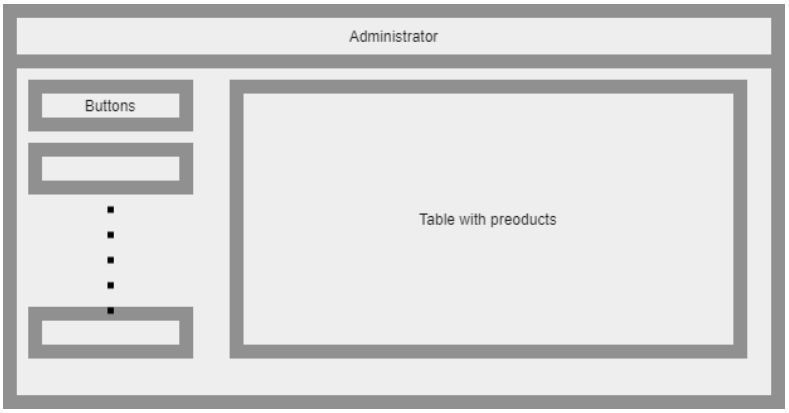
DeliveryService is responsible for implementing the requirements for the administrator, client and employee operations. Because this is the class that both stores and manipulates the data, this is the class responsible for notifying the employees when a new order is created.

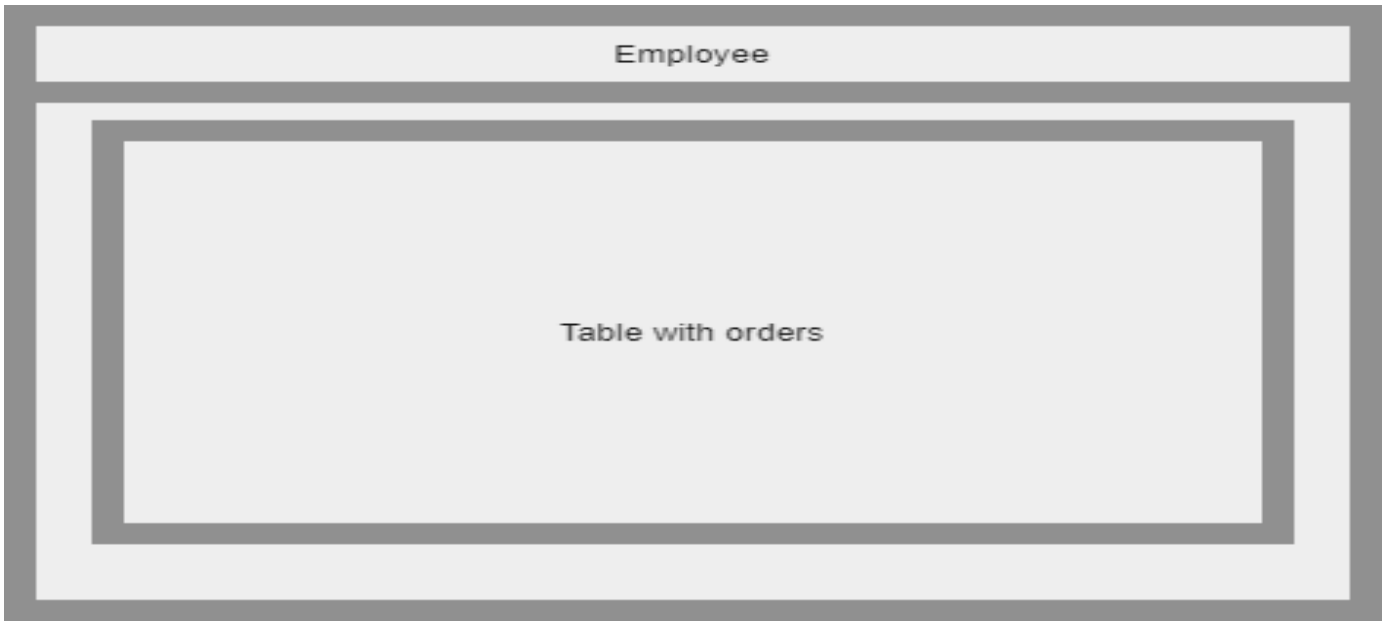
For this, the Observer Pattern will be implemented. The DeliveryService is the Observable object. Or the subject to be observed for new modifications. These updates will be sent to the observers which in this case are the employees.

FileWriter and Serializator are classes that help saving the information about the reports and the current state of the delivery service into files.

User Interface Classes

There are 4 graphical user interface classes, one for each type of users and one for the login page.





Implementation

The implementation was mostly done using stream processing for the collection data structures used, especially by the DeliveryService class.

Streams are wrappers around a data source, allowing us to operate with that data source and making bulk processing convenient and fast. Streams allow us to operate on collection data structures more easily, without having to manually iterate through each item of the structure. The stream operations can be terminal or intermediate.

Terminal operations are those operations which return a value of a specific type. The ones used in the application are .findFirst(), .findAny() and .collect(). The first two operations return one single value specific for the data structure, while collect returns more objects of the data structure into any type of Collection of the Java Collection Framework.

The below example shows the method to select all the products which contain a certain keyword:

```
List<MenuItem> result = data.stream()
        .filter(item -> item.getTitle().contains(keyword))
        .collect(Collectors.toList());
```

Intermediate operations return a stream of the same type, thus allowing us to chain multiple operations.

The below example shows us how the filter method is applied twice by chaining, to create the hourly report:

```
List<Order> validOrders = orders.keySet().stream()
        .filter(order -> order.getHour() >= startHour)
        .filter(order -> order.getHour() <= endHour)
        .collect(Collectors.toList());
```

The intermediate operations are followed by terminal operation, because otherwise they would return a stream on which we could not access the data.

Another example of the usage of stream processing and lambda expressions is in generating reports about the delivery service status:

```
@Override
public void createDateReport(int day, int month, int year) {
    assert day > 0 && day <= 31;
```

```
    assert month > 0 && month <= 12;
    assert year > 0 && year <= LocalDateTime.now().getYear();

    Map<String, Integer> validProducts = new HashMap<>();
    orders.entrySet().stream()
            .filter(entry ->
                    entry.getKey().getDate().getDayOfMonth() == day &&
                    entry.getKey().getDate().getMonthValue() == month &&
                    entry.getKey().getDate().getYear() == year)
            .forEach(items -> items.getValue()
                    .forEach(item -> {
                        if (validProducts.containsKey(item.getTitle()))
                            validProducts.computeIfPresent(item.getTitle(), (c, i)
-> i + 1);
                        else validProducts.put(item.getTitle(), 1);
            }));


    dateCount++;
    FileWriter.generateDate(validProducts, day, month, year, dateCount);
}
```

In the above example, the report based on the date is generated by computing using streams a new hash map inside the stream of the original hash map of the orders.

Beside streams and lambda expressions, the application uses serialization for saving the current state of the delivery service. To serialize an object means to convert its state to a byte stream so that the byte stream can be reverted back into a copy of the object.

The intention of using serialization is to save the progress of the current state of the delivery service object shared by the users. Thus, by deserializing the object, we can retrieve the old state of the service object.

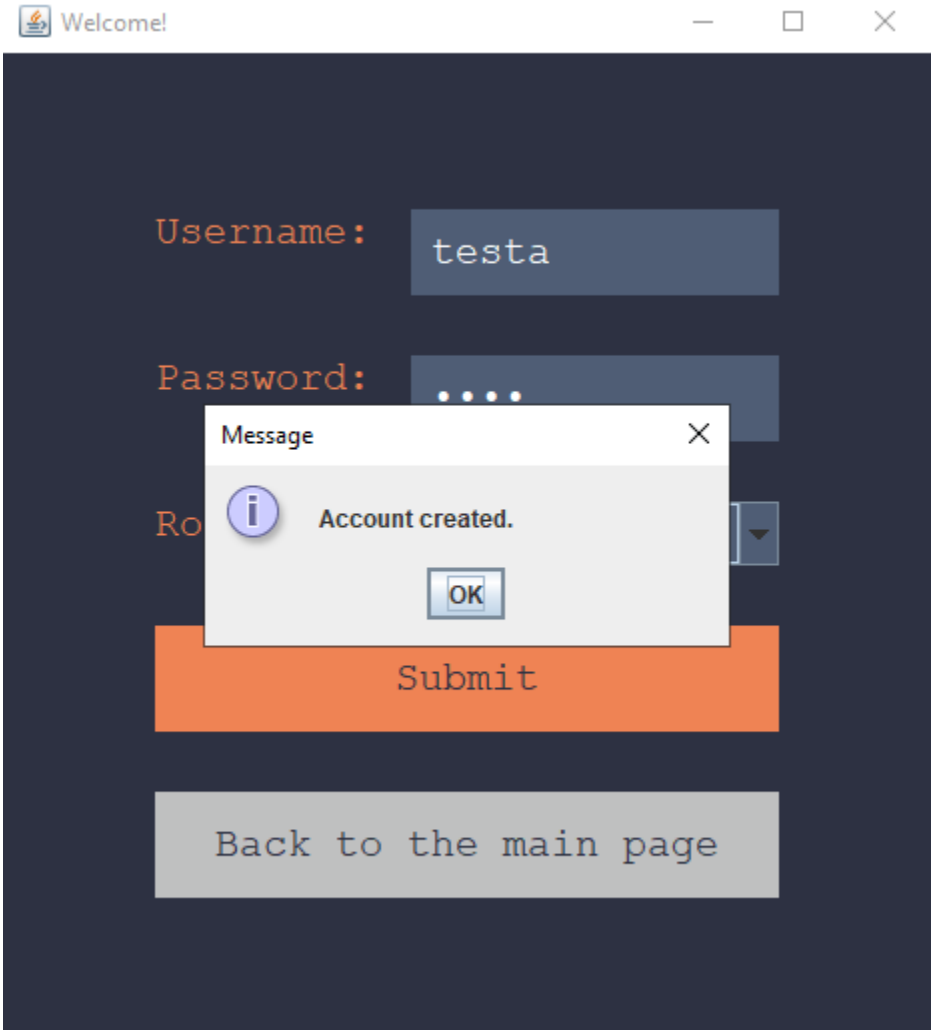
For generating the reports and bills, the FileWriter contains methods which insert step by step information into the files. Below is an example for generating the bills:

```
public static void generateBill(List<MenuItem> items, Order order) {
    File bill = new File("src/main/resources/Order Bills/Order_" +
order.getOrderId() + ".txt");

    try {
        bill.createNewFile();

        java.io.FileWriter writer = new java.io.FileWriter(bill.getPath(), true);
        writer.write("---------------------- ORDER NUMBER " + order.getOrderId() + "
----------------------\n");
        writer.write("Date: " + order.getDate().toString() + "\n");
        writer.write("Client: " + order.getClientId() + "\n");

        double total = items.stream()
                .mapToDouble(item -> item.computePrice())
                .sum();
        writer.write("Total: " + total + "\n\n");
        writer.write("---------------------- ORDERED PRODUCTS ----------------------
\n");
```

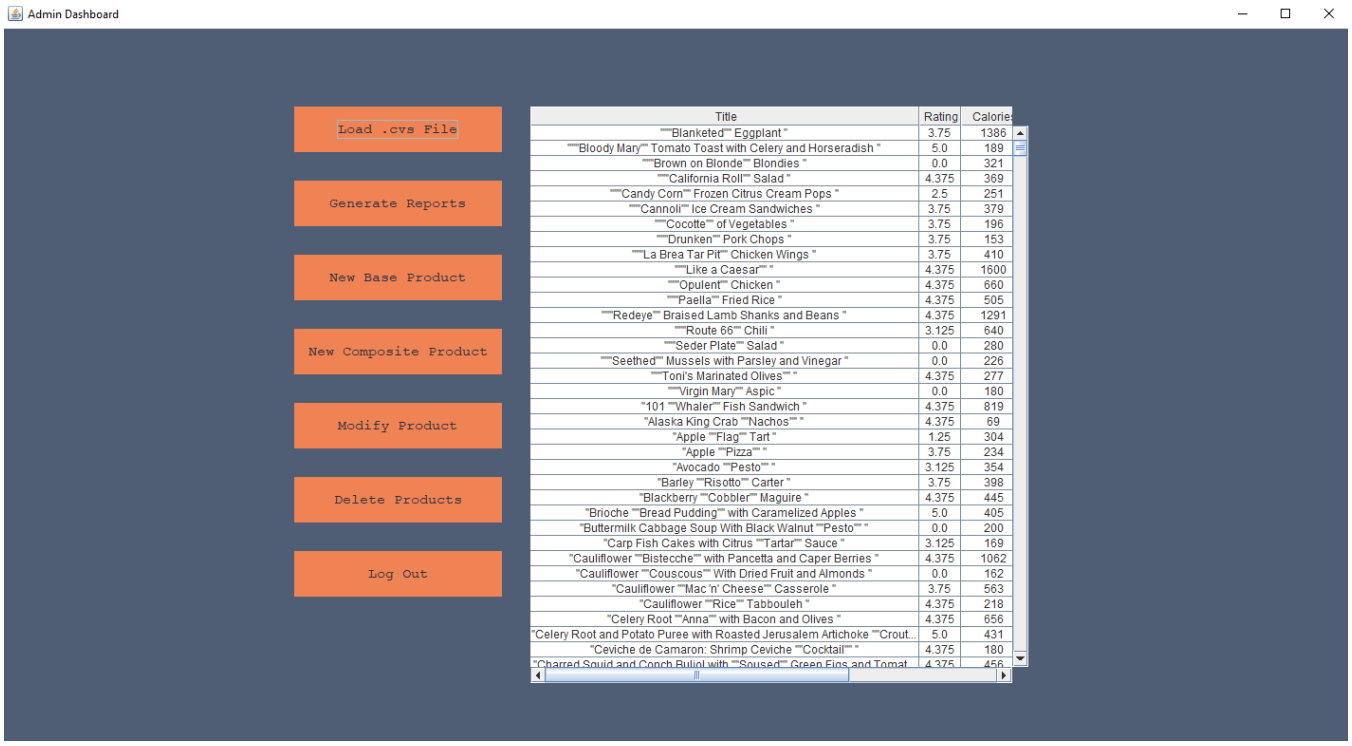
```
        for (MenuItem item: items)
            writer.write("Product Name: " + item.getTitle() + "\n");

        writer.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Testing

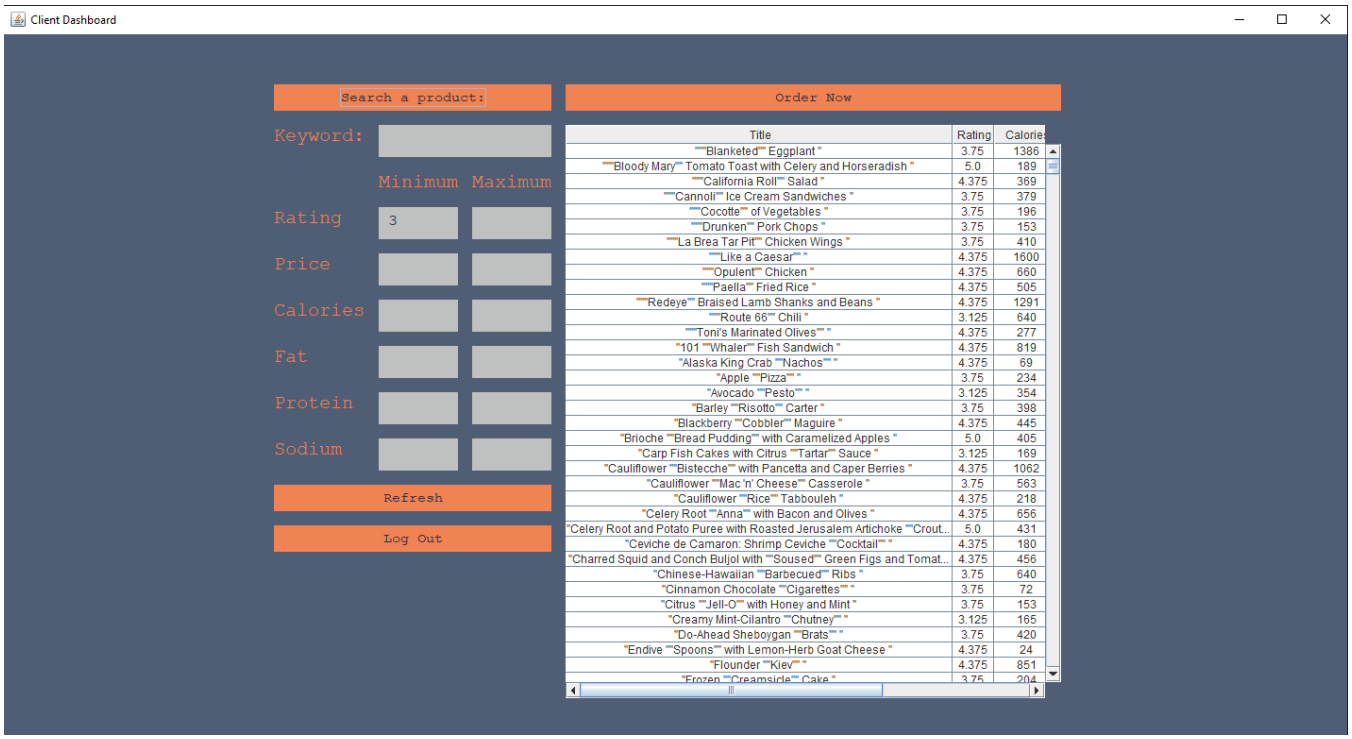
Login



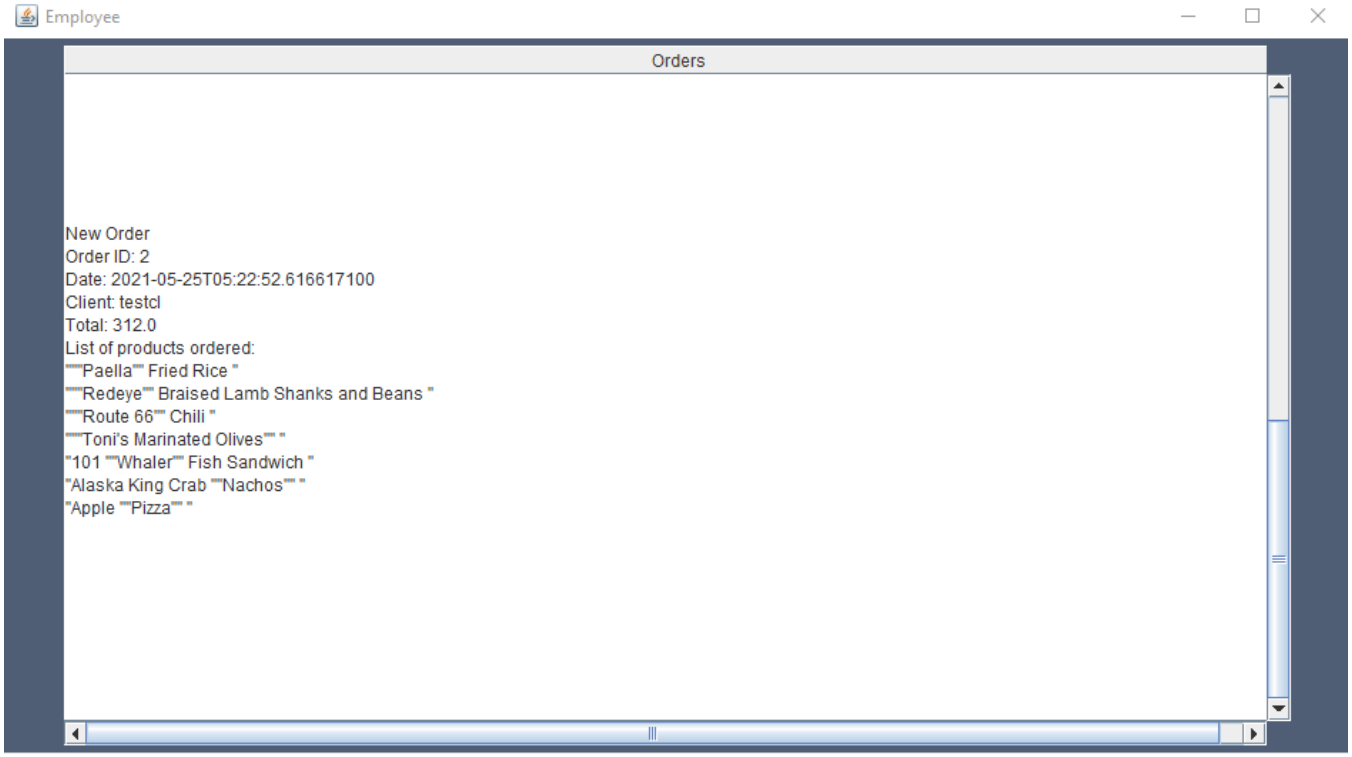
Administrator



Client



Employee



Conclusions

This assignment was a bit ambiguous at first. The problem was vaguely specified, but the biggest issue was that the assignment required us to implement a lot of new techniques which I have never used before. Thus, I ran into a lot of exceptions like a serialized object which was empty or a stream that filtered more than it should filter.

It was a nice experience to learn all of these, however I felt a little bit overwhelmed. I must admit that I should have assigned more working days for this project.

Unfortunately, I didn't get to use the JavaFX framework for the graphical user interfaces. However, I think I finally mastered my Java Swing abilities.

Bibliography

- Serializable Objects, Oracle
- Observer Design Pattern, Derek Banas
- Java read CSV File, Bro Code
- Programming with Assertions, Oracle
- Enable assertions, StackOverflow
- Serialization, GeeksForGeeks