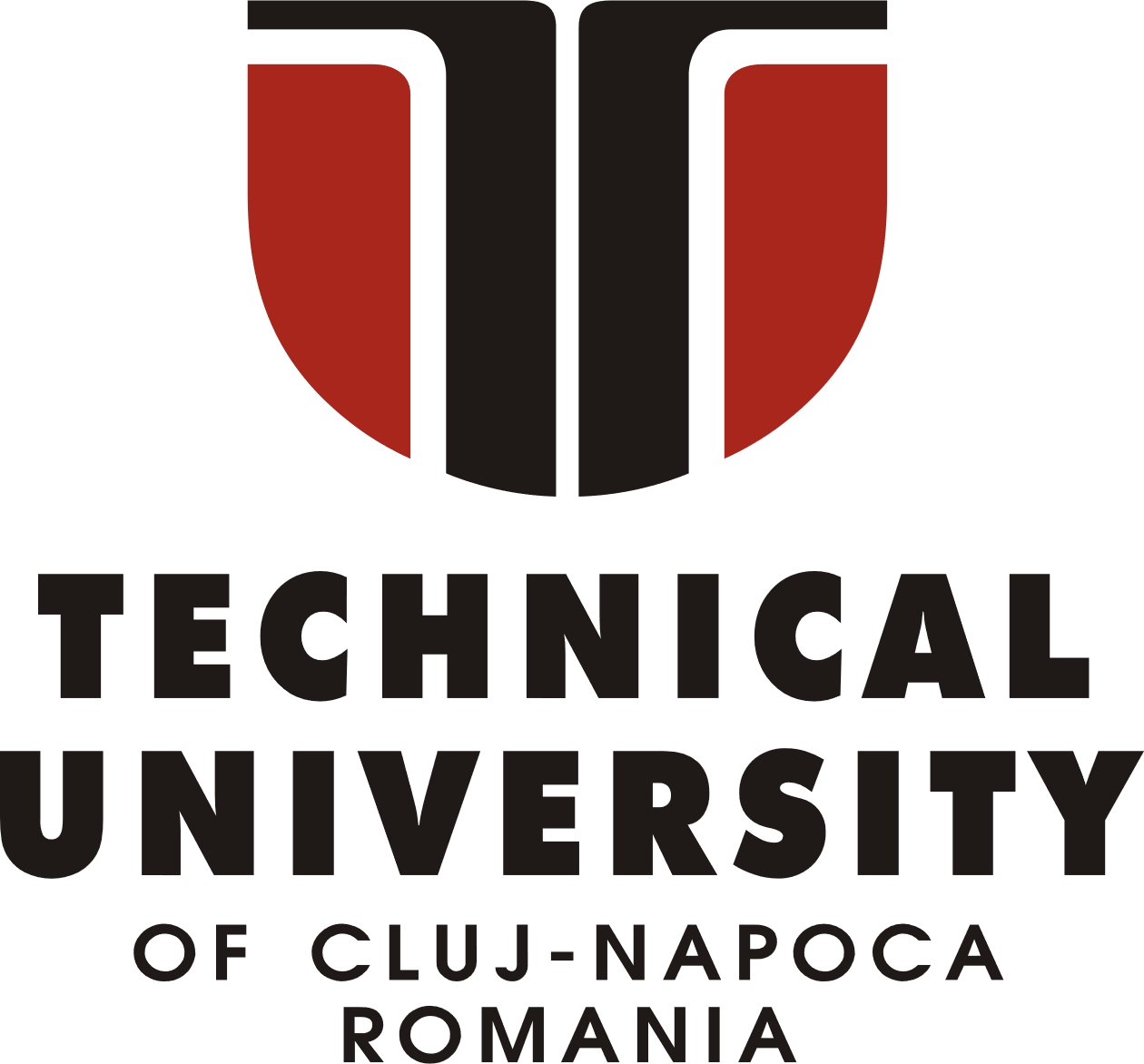
****

**\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_**

**DOCUMENTATION**

**Assignment 4. Food Delivery Management System**

Fundamental Programming Techniques

Student: **Radu-Augustin Vele**

Group: **30422**

Table of Contents

1. Project Objectives3

2. Problem Analysis. Modelling. Scenarios. Use cases3

3. Design7

4. Implementation8

5. Results11

6. Conclusions11

7. Bibliography11

**1.Project Objectives**

The main objective of the project is to create a food delivery management application, useful for three types of users: administrator, client, employee. It was designed to support most of the processes met in a real restaurant (especially related to orders and menu management).

Secondary objectives:

* analyze the problem while identifying requirements (see Problem Analysis)
* design the project using the object-oriented approach (see Design section)
* organize and implement the application using the Layered Architecture (see Implementation section)
* use serialization and various design pattern to reach the desired results.
* test the application carefully. (see Results section)
* use stream processing and lambda expressions for searching and generating reports
* provide a javadoc file for the project

**2. Problem Analysis. Modelling. Scenarios. Use cases.**

The problem imposes both functional and non-functional requirements.

**Functional requirements:**

*a. Administrator-related:*

* the administrator should be able to import an initial set of products from a .csv file
* they should add/edit/delete menu items
* they should be able to combine products to create composite menus
* they should be able to generate four types or reports (most popular time interval, popular products, regular customers, daily report)

*b. Client-related:*

* the clients should be able to search for the desired item based on no/one/multiple filters.
* the clients should be able to create an order of several products and receive a bill

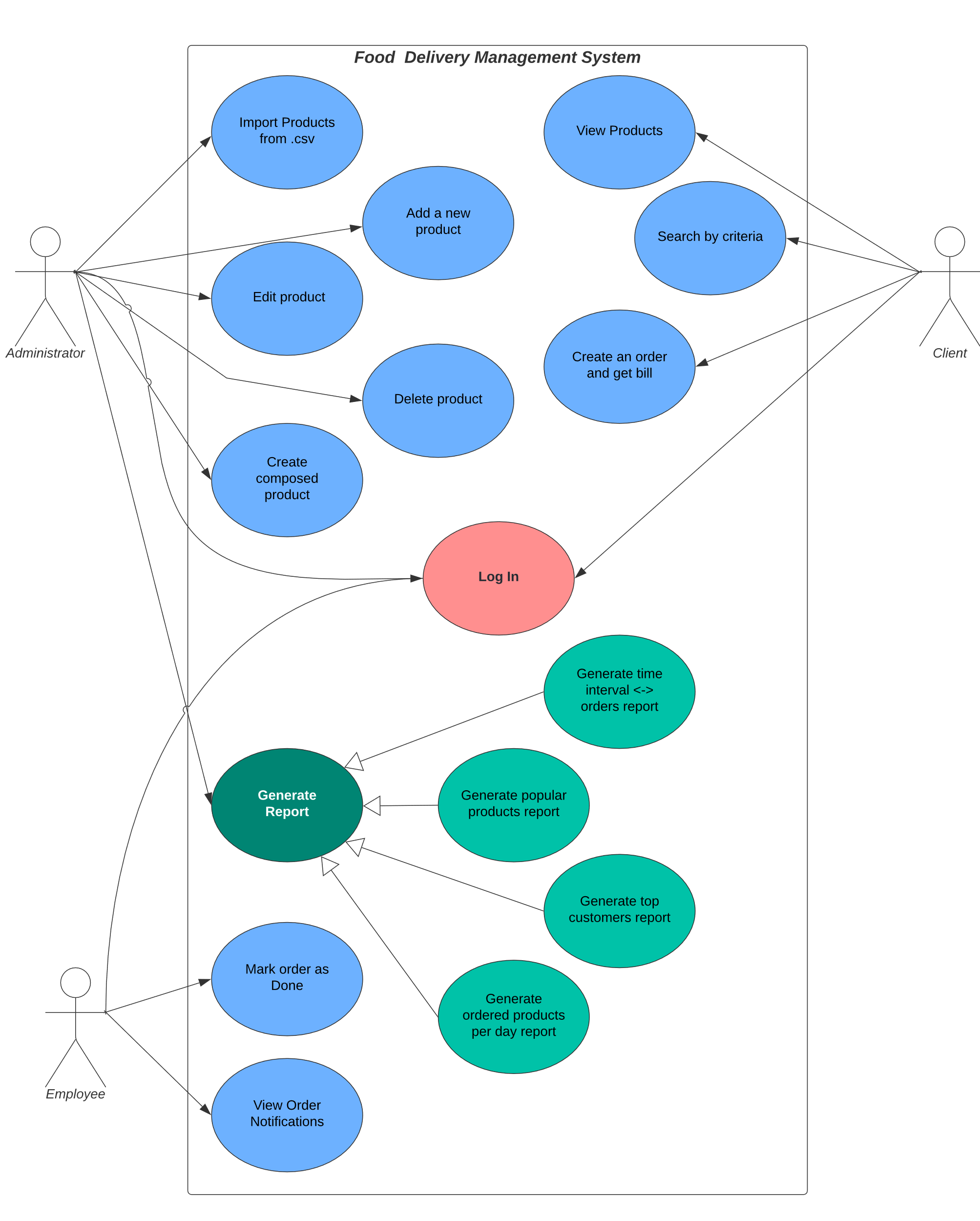
*c. Employee-related:*

* + the employees should be notified each time a new order appears, and be able to mark it as done.

**Non-functional requirements:**

* + the application should be intuitive and easy to use.
  + the application should use an authentication procedure to create different scopes for the different types of users.
  + the interface should not contain many different windows.
  + the data regarding the menu items, orders, clients should be persisted through .txt files.

**Use cases:**

The use cases of the food delivery application are presented in the figure. While there are many different use cases, the trivial ones will not be presented in detail, but some of the more complex ones are described below.

Use case: Administrator imports the initial set of products

Actor: Administrator

a. Success scenario:

* + The administrator logs in and the Admin Window appears.
  + A pop up window appears, informing the admin that some of the information regarding the menu items or orders was not loaded (as the initial set of products was not imported).
  + The administrator closes the pop up, stays on the initial tab, and presses the Import Initial Set of Products Button.
  + A success label appears, and the Administrator could check the initial set of products by clicking the Refresh Products Table button.

b. Alternative scenarios:

* + the administrator doesn't input the right password, and a pop-up appears, informing they that the username exists, but the password is incorrect.
  + the administrator tries to input the initial set of products while some menu items have been loaded at the start of the program. A label that informs the administrator about this appears and the set of menu items is not overwritten.

Use case: Administrator creates a composite product

Actor: Administrator

a. Success scenario:

* + the administrator logs in and selects the Menu Wizard tab in the Admin Window
  + the administrator clicks on "Refresh Table" button and chooses more items to add to the selected items table.
  + they can remove some or all of them
  + if there are at least two selected, the administrator can choose a new title and press: Create New Menu Item.
  + a success label appears and the new menu item can be visible in the product list (its attributes are computed based on the base products it is composed of).

b. Alternative scenarios:

* + the administrator selects only one item to add to the newly created product, and a pop up window informs they that there need to be at least two products selected.
  + the administrator doesn't input a title for the new item, and a pop up window informs they that they need to complete that text field

Use case: Administrator generates a report of the products ordered within a time interval

Actor: Administrator

a. Success scenario:

* + the administrator logs in and selects the Reports generator tab
  + the administrator chooses from the right part of the window the desired type of report (Time interval, in our case).
  + the administrator inputs the start and end hour obeying the indicated format.
  + a txt file with the orders performed within that interval, regardless of the day is generated.
  + a success label becomes visible.

b. Alternative scenarios:

* + the admin doesn't input the right hour format in the text fields, and a pop up window appears informing they that the format should be HH:MM.

Use case: Client searches products and creates an order

Actor: Client

a. Success scenario:

* + the client logs in and the Client window is displayed.
  + the client checks the boxes with the attributes they want to filter the menu items list by and presses the Search Button
  + the table of products matching the filters appears.
  + the client selects one or more items and presses Add selected
  + to view the cart contents, the client presses on "go to cart" and another panel containing the selected items and the total price is displayed.
  + the client can delete the selected products, delete all products, or add more items, before pressing on Finish order.
  + a success window is displayed and a .pdf bill is generated containing client information and the ordered products.

b. Alternative scenarios:

* + the client presses on Finish order while no products are selected, and a pop up window appears informing they that at least an item should be selected.
  + the client checks a box and doesn't input any text for filtering and presses on the search button. A pop up window appears informing they that when a box is selected, there should be input in the corresponding field.

Use case: Employee is notified, and marks order as done

Actor: Employee

a. Success scenario:

* + the employee logs in and the Employee window is displayed.
  + the employees sees the list of orders that have not been served.
  + they select the orders that have been finished and press on "Mark selected order" as done.
  + the order disappears from the list and the number of orders is decremented.

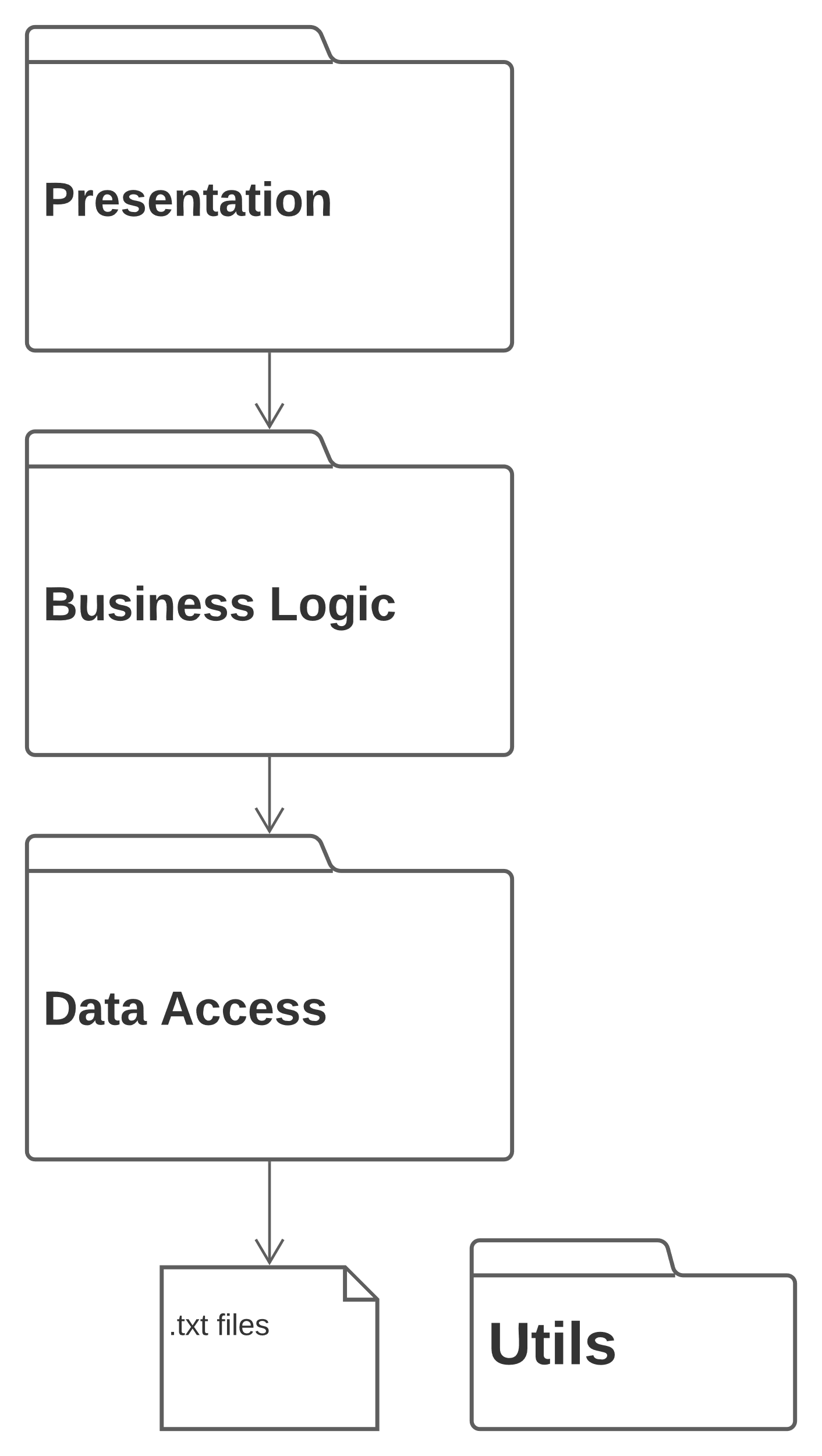
b. Alternative scenario:

* + the employee presses on the button while no order is selected, and a pop up window appears and informs they that an order should be selected.

**3. Design**

The solution was designed according to the Object-Oriented Programming paradigms, using the layered architecture design, and various design patterns that will be described below.

**Package Diagram**



The layered architecture imposes a vertical communication based on requests from the upper level and returned data from the lower level. The package diagram below shows three main levels: Presentation, Business Logic, and Data Access. The presentation layer contains all the GUI elements (described in implementation), the business logic contains, as the name suggests, the program logic that takes applies the requests received from the presentation layer to the data that was fetched from the Data Access Layer. The data access layer supports the loading and saving (persistence) of data (Menu Items, Orders, Accounts, Orders to be Delivered). The utils package contains some custom exceptions used to generate specific behaviors when they are thrown.

**OOP Design**

The overall organization of the classes in the project, and the relationships between them, is presented in the UML class diagram that follows.

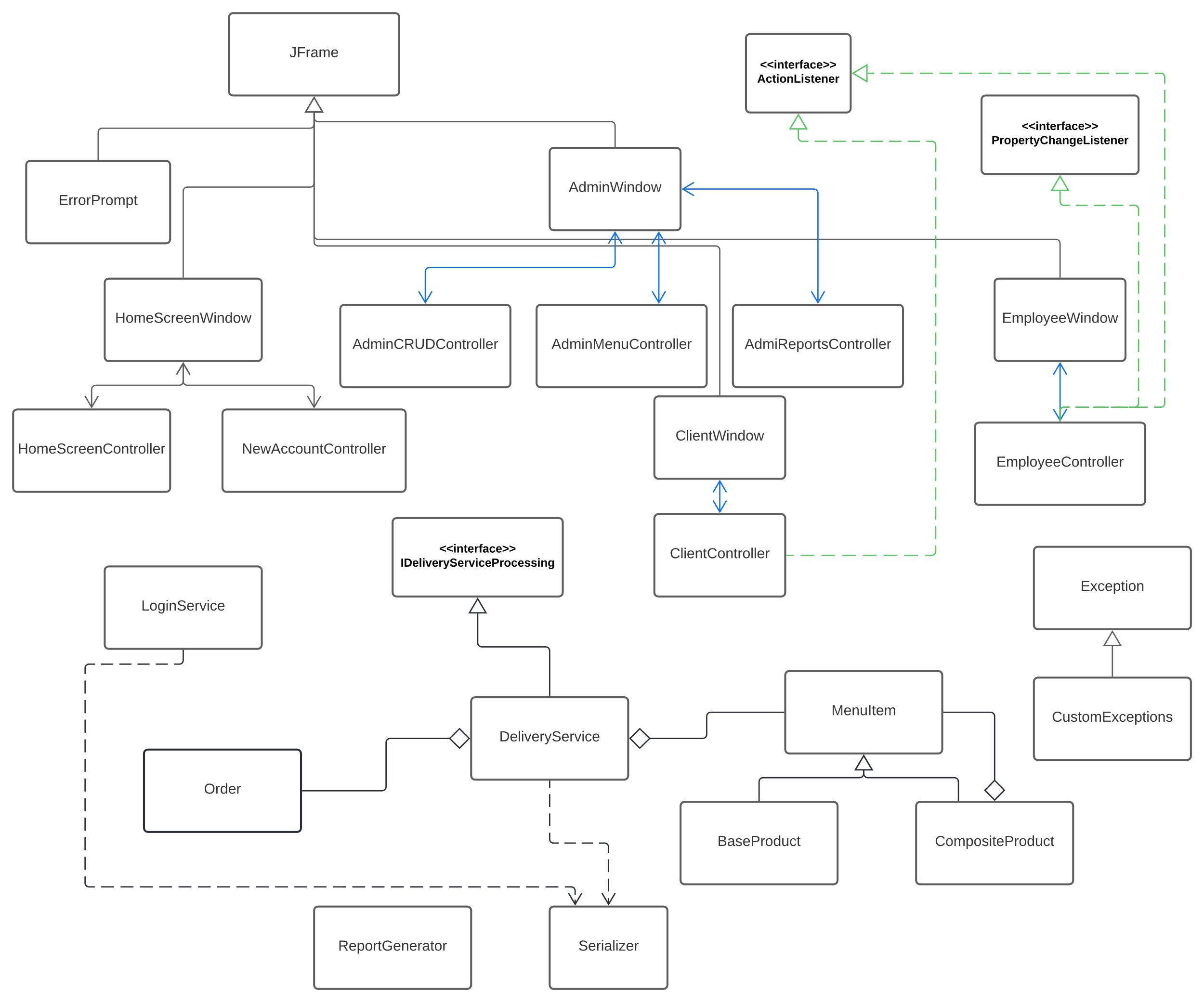
In the presentation part, there is a strong dependency (double association) between the GUI (window) and controller.

Each window has one or more controllers that implement behaviors for the buttons pressed inside itself (Action Listeners). In the case of the business logic layer, there is the Composite Design Pattern-specific relationship (described in the implementation chapter), alongside composition between the DeliveryService, and Order and MenuItem. That is because the DeliveryService class (that implements the interface IDeliveryServiceProcessing) contains the loaded data regarding the application (apart from the LoginService that hosts the accounts).

**Data Structures:**

The requirements of the project impose the need to use suitable data structures. An important operation in the application is searching, therefore, data structures that use a Hash Table (searching is done in constant time).

For storing the menu items the data structure used in a Hash Set that identifies. The .equals() and .hashCode() methods in the MenuItem class are modified such that the set won't allow products having the same title to repeat. For storing the orders, the chosen data structure is a LinkedHashMap<Order, Collection<MenuItem>>, as it also keeps a predictable iteration order.

****

**4. Implementation**

Some of the more complex classes of the application will be presented in the section below.

**MenuItem**

This is an abstract class useful for implementing the Composite design pattern. It contains multiple fields describing a product (title, calories, protein, fats, sodium, rating, price), mutators, and override hashcode(), clone(), equals() in order to facilitate its use within a HashSet collection. It is also serializable to support the persistence of data.

**Order**

The class contains three fields: orderID, clientID, orderDate alongise mutators for managing their contents. It is designed to be used in a Map<Order, Collection<MenuItem>>. Its date is generated based on the moment the an instance is created.

**LoginService**

This class supports the login service, receiving through serialization a set of all the accounts in the application, when it is lauched. It also contains a method for generating the order ID (based on the number of orders that have already been made).

**DeliveryService**

This is the most important class of the system as it contains a set of all menu items, a map with all the orders and ordered products, .and a map with all the orders that need to be served, all of them being loaded and saved at the beginning and end of the application. It is created using the singleton design pattern as only one instance of it should be allowed at one time. It contains methods that support the management of these sets and maps based on the requests coming from the controllers. It also implements the IDeliveryServiceProcessing interface that contains the core methods needed in the application: importProducts, manageProduct, generateReport, createNewOrder

**ReportGenerator**

This class is used for generating the reports requested in the Admin Window. It supports 4 types of reports. All of them are generated using stream processing and lambda expressions, and the obtained results are printed in .txt files.

**Design Patterns**

1. Observer Design Pattern

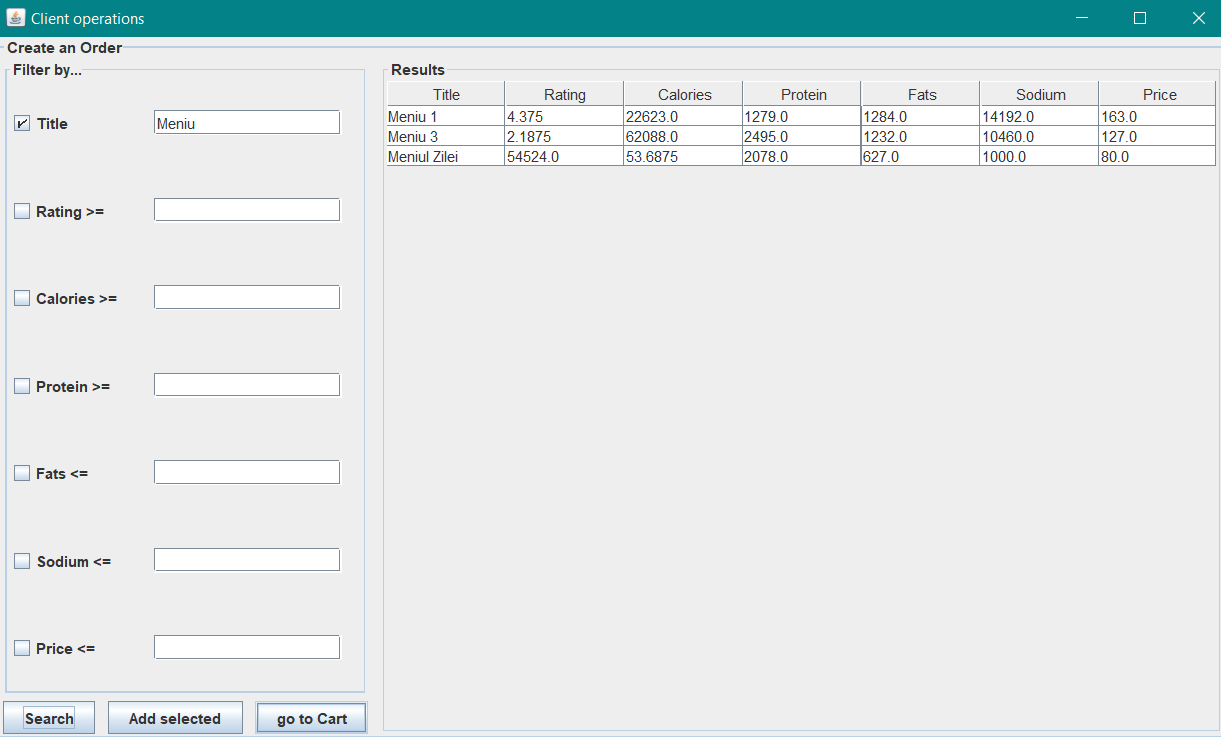
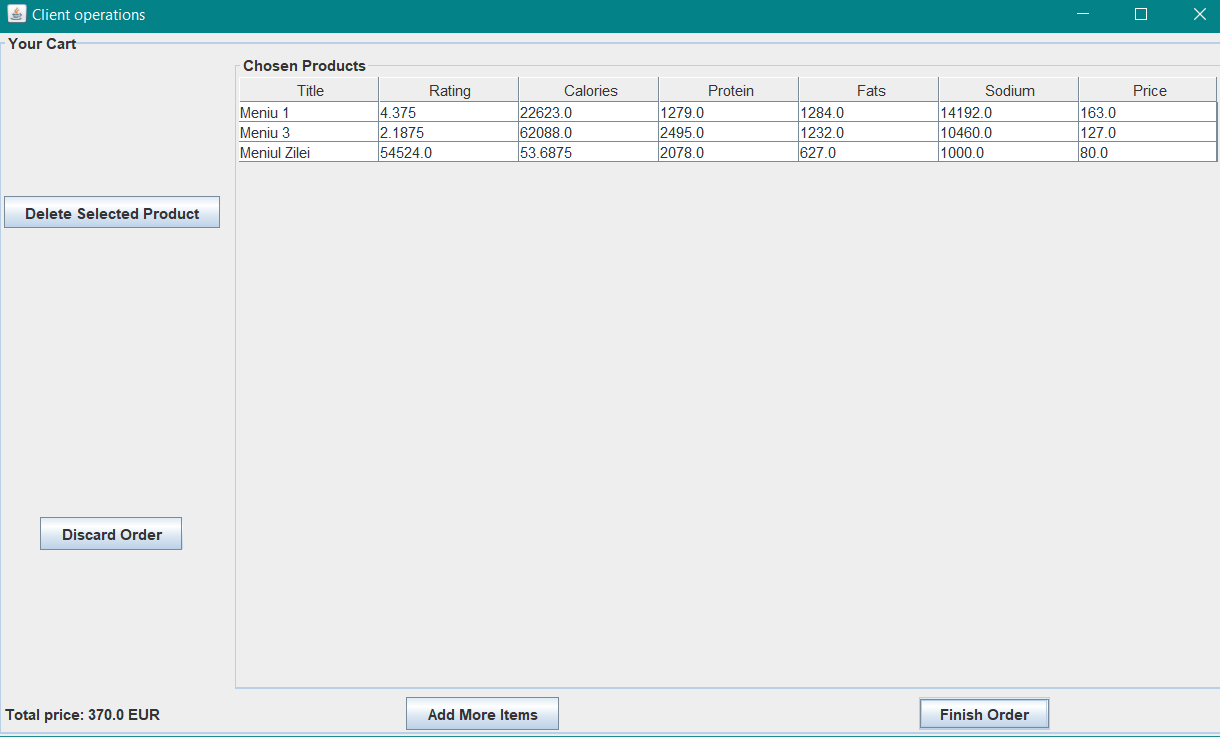
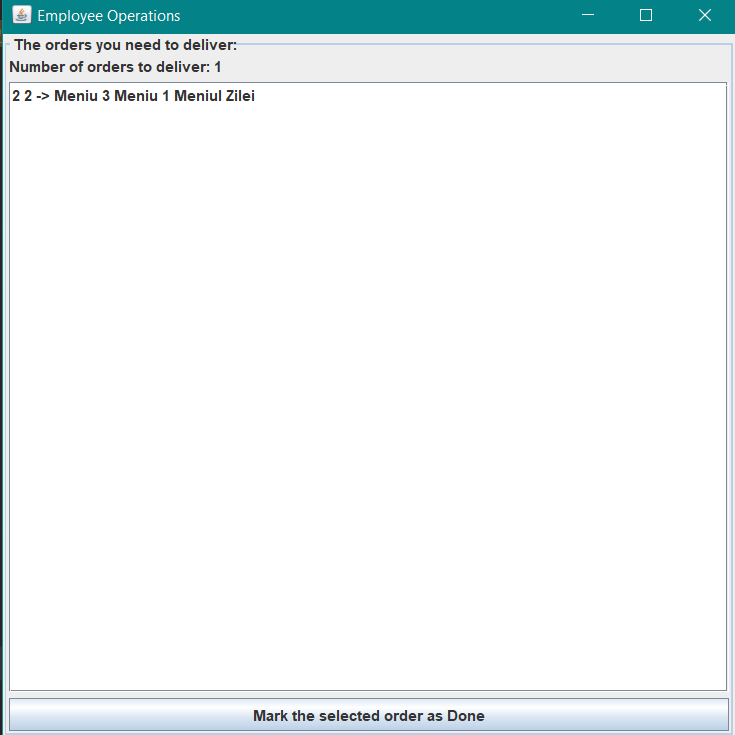
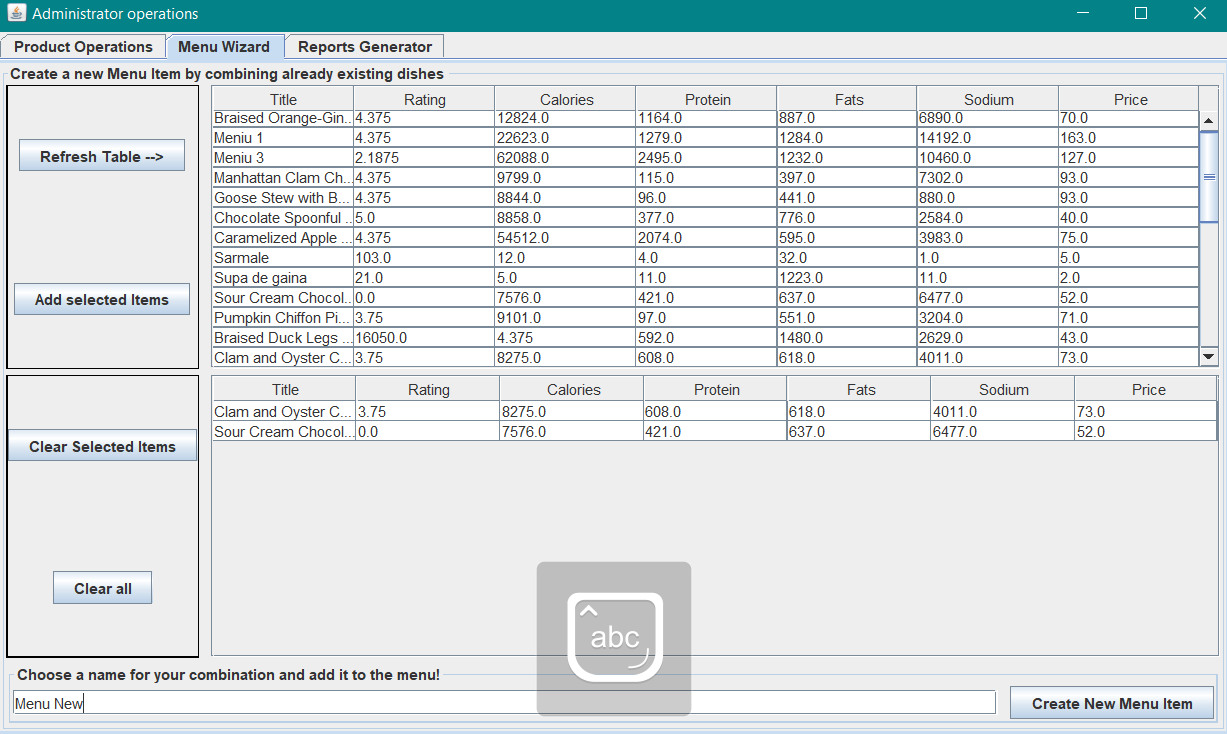
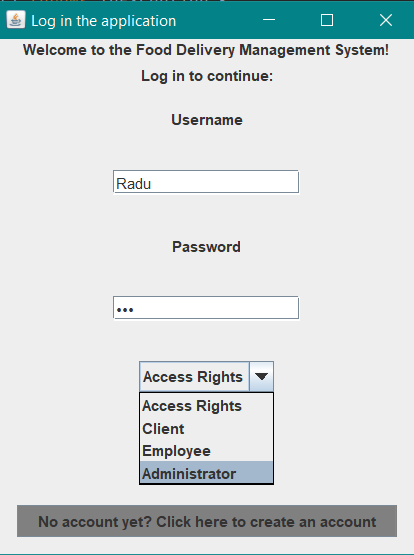
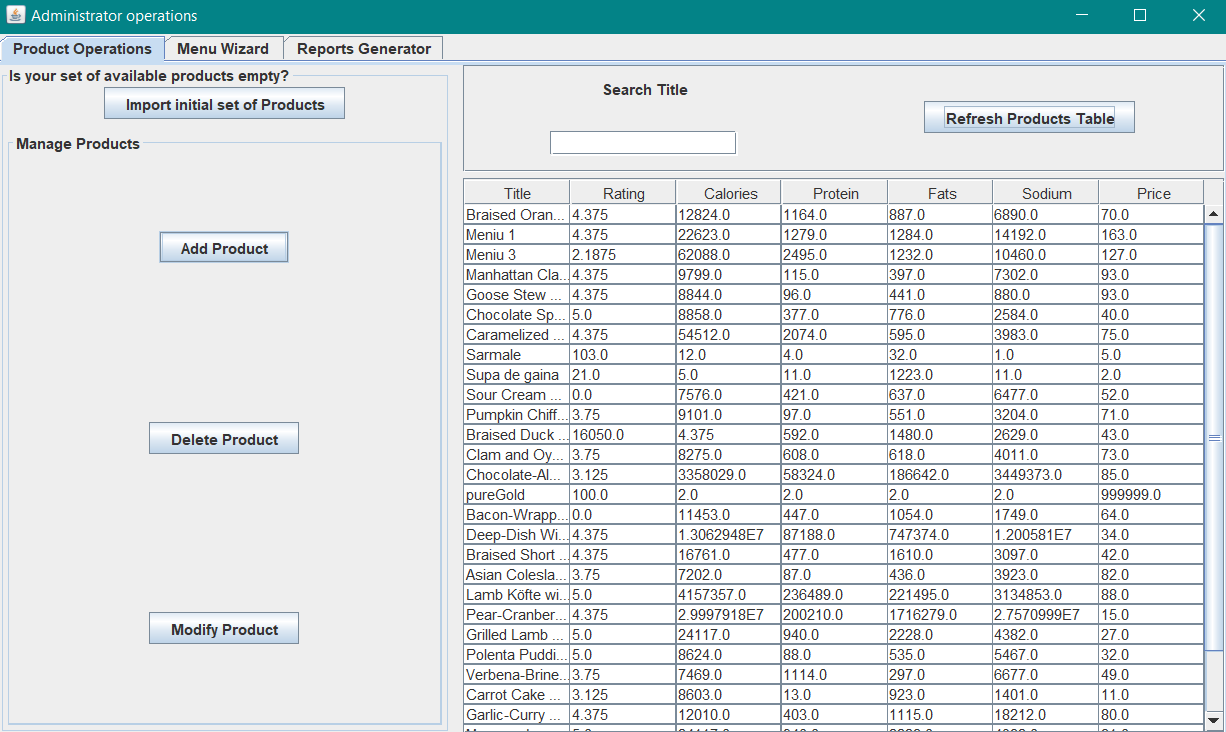
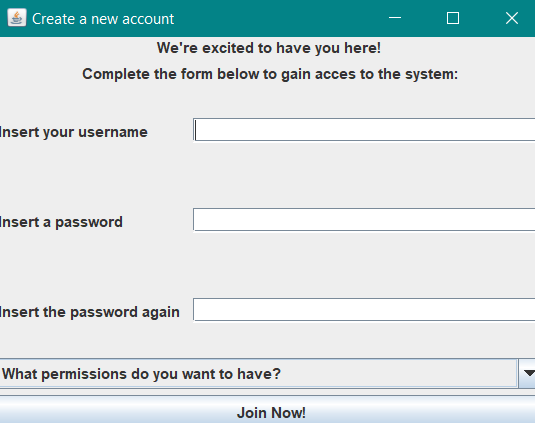
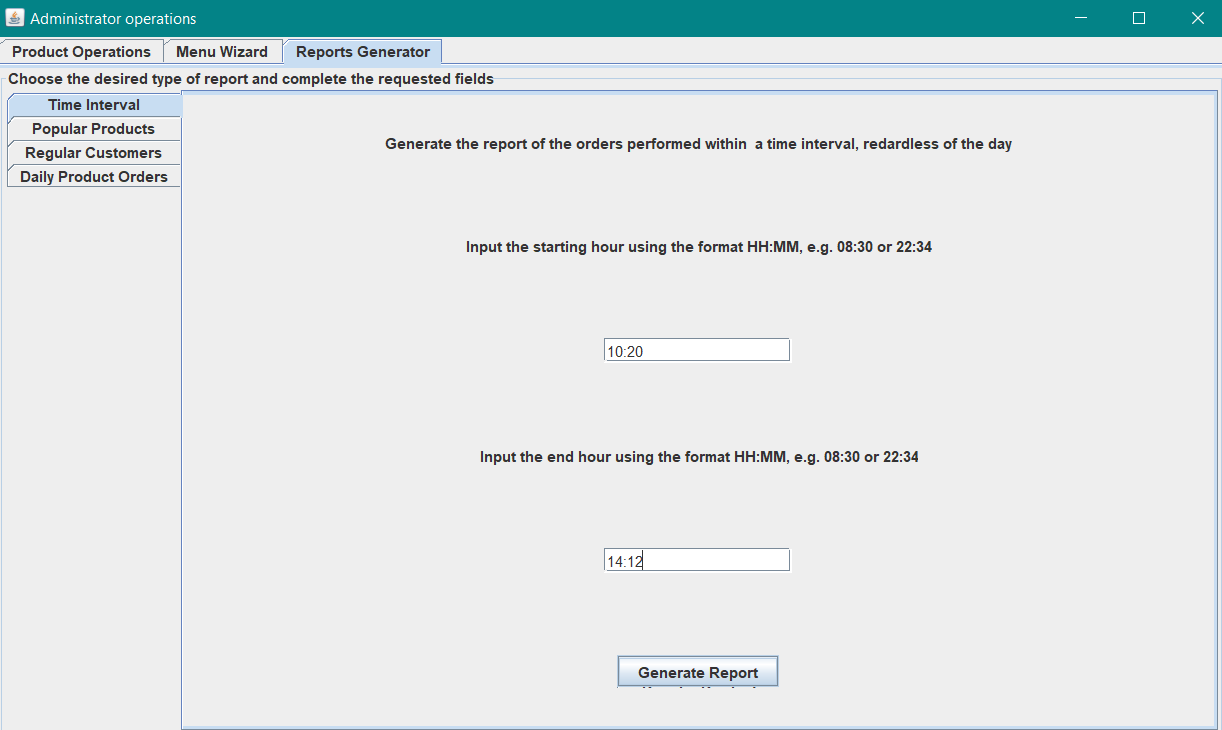
This design pattern is used for the process of managing the orders by the customers. When the observable object (DeliveryService object) creates a new order, the observer object (EmployeeController object) is notified in order to update the list of orders to be served. The implementation is supported by the javaBeans PropertyChange mechanism.

2. Composite Design Pattern

This design pattern is used in order to have the same “treatment” for a BaseProduct and a Composite Product, even though the later one contains multiple MenuItem objects.

**Graphical User Interface**

The Graphical User Interface was designed to allow the user to easily switch between the functionalities of the application. Initially a login window appears. The flow of the windows is determined by the accessing rights of the user: AdminWindow, ClientWindow, EmployeeWindow. An image of each of these windows is presented below.



**5. Results.**

The functionality of the application was tested for various scenarios proving to work successfully. Moreover, a well-formed method has been added to the DeliveryService class in order to check if the information within itself was correctly organized.

**6. Conclusions. Takeaways. Further Development**

The application was a true challenge imposing a steep learning curve. The most important outcome of the development was the understanding of the relationships between classes, lambda expressions, streams, the Composite Design Pattern and the Observer Design Pattern. A very important aspect that needs to be further developed is the graphical user interface. Also, the reports should be generated in a more detailed manner.

**7. Bibliography**

1. Distributed systems research laboratories - TUCN, resources.

2. Stack overflow: streams.

3. Java Docs: Streams, Lambda Expressions, Calendar, Date, PropertyChange.

4. GeeksForGeeks: Composite Design Pattern.