

**FOOD DELIVERY MANAGEMENT SYSTEM**

Student: Francesca Dițulescu

Teacher: Viorica Chifu

Table of Contents

Assignment Objective

Main Objective

Sub-objectives

Problem analysis, modeling, scenarios, use cases

Problem analysis

Solution

Requirements

Use cases

Design

Overall System Design

Division into packages

Division into classes

UML Class Diagram

GUI Design

Implementation

Results

Conclusions

Bibliography

ASSIGNMENT OBJECTIVE

MAIN OBJECTIVE

The main objective of this assignment is to design and implement a food delivery management system with a dedicated graphical user interface in which the user can log in and perform different operations according to its role (administrator, regular employee, client) on the products and on the orders.

SUB-OBJECTIVES

* Analyze the problem and identify all the requirements
* Design the food delivery management system application
* Implement the food delivery management system application
  + Implement the log-in window
  + Implement graphical user interface for all types of users (client/employee/admin)
* Test the food delivery management system application

PROBLEM ANALYSIS

The purpose of this project is to solve the following problem: Ordering food on the phone has become an issue when there are multiple clients calling at the same time. Therefore, multiple clients should be able to order something without waiting on the phone and waste time.

Moreover, keeping track of orders and products on paper is really difficult, especially when there is an update that has to be made on one of the two.

Another point that has to be made here is regarding the delivery employees, which are having trouble looking through recipes and see all the details. They should be able to see and be notified about their orders more easily, so that they can deliver faster.

SOLUTION

This projects provides a faster and more interactive application that can help the user (according to its role) to manage the orders, the products and the whole delivery process in an easier way.

REQUIREMENTS

Functional requirements

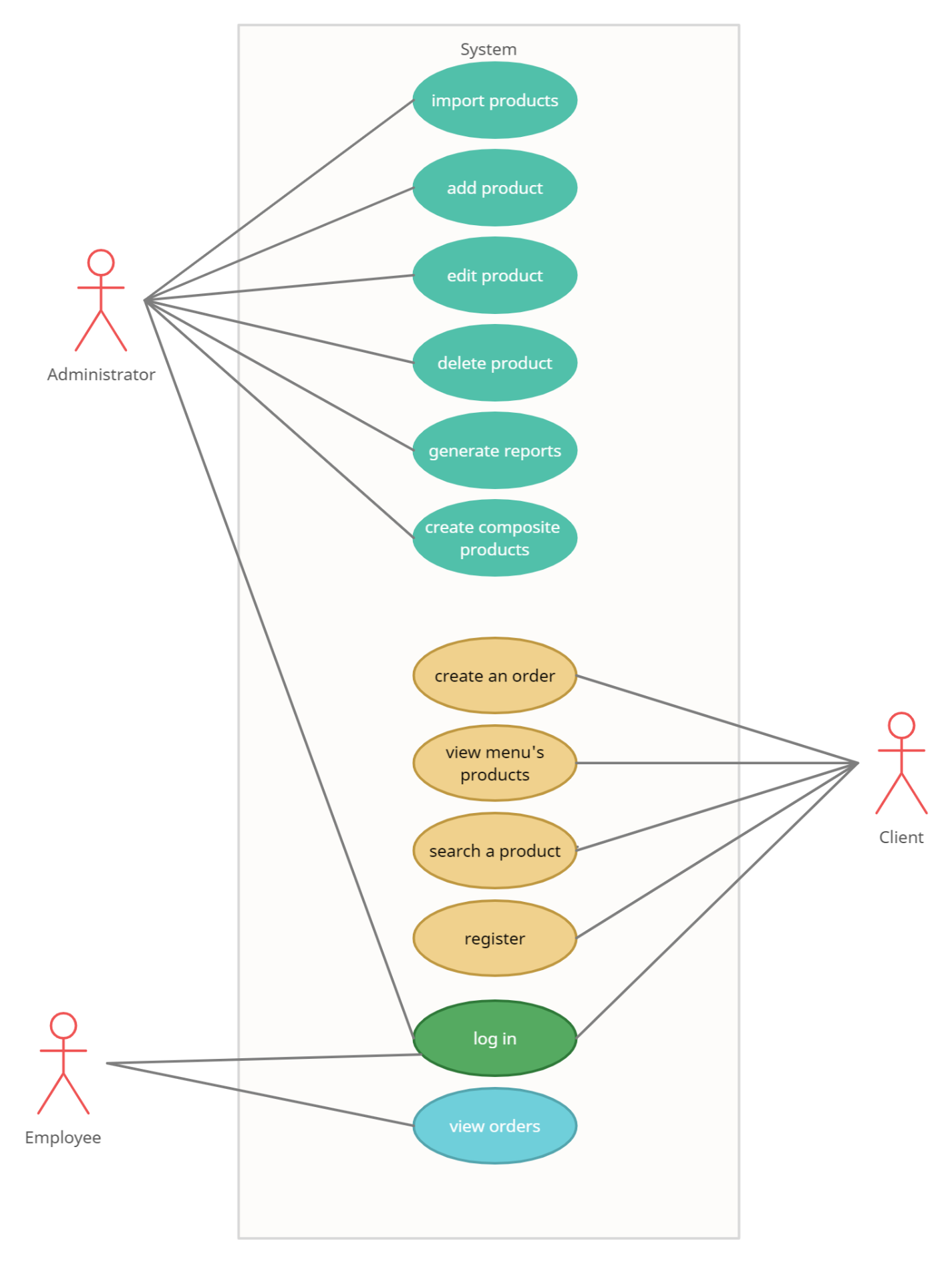
* The food delivery management system should allow the user to log in.
* The food delivery management system should allow the user to selects the operation he/she wants to perform.
* The food delivery management system should be able to import the products from a .csv file.
* The food delivery management system should be able to add/edit/delete a product.
* The food delivery management system should be able to add composed ingredients and select the elements from the products list.
* The food delivery management system should be able to generate reports about the orders performed based on the following criteria:
  + the time interval of the orders (select a period during which orders were made by inserting a start hour and an end hour)
  + the number of times a product has been ordered (specified by the user).
* The food delivery management system should be able to list all the orders made by a client.
* The food delivery management system should allow the client user to search through products based on one or multiple criteria (the fields of the product – title, rating, calories, proteins, fats, sodium, price).
* The food delivery management system should be able to create an order consisting of several products (either base or composed) and generate a bill for each order that contains the date, the list of products ordered and the total price of the order.
* The food delivery management system should be able to notify the employee regarding a newly placed order.

Non-functional requirements

* The food delivery management system should be intuitive and easy to use by the user.
* The food delivery management system should have a nice and pleasant graphical user interface.

USE CASES

The Use Case Diagram is on the following page.



1. Log In

Primary Actor: User (admin/client/employee)

Main Success Scenario:

1. The user selects the button corresponding to its role.
2. The user introduces the log in credentials
3. The user selects the log in button
4. The application performs the log in and opens a new window with the operations that the user can perform according to its role.

Alternative sequence:

1. The user introduces an invalid log in credential (username/password)
2. The scenario returns to step 1.

2. Import products

Primary Actor: Administrator

Main Success Scenario:

1. The admin selects the *admin* button from the main window
2. The admin logs in successfully
3. The admin selects the *import products* button.
4. The application performs the import products operation and displays a message that it was successful.

Alternative sequence – None

3. Add a product

Primary actor: Administrator

Main Success Scenario:

1. The admin selects the *admin* button from the main window
2. The admin logs in successfully
3. The admin introduces the data about the new product
4. The admin selects the *add* button.
5. The application performs the insert operation and displays a message that it was successful.

Alternative sequence:

1. The admin introduces an invalid input
2. The scenario returns to step 3.

4. Edit a product

Primary actor: Administrator

Main Success Scenario:

1. The admin selects the *admin* button from the main window
2. The admin logs in successfully
3. The admin introduces the data he/she wants to update/edit
4. The admin selects the *edit* button.
5. The application performs the update operation and displays a message that it was successful.

Alternative sequence:

1. The admin introduces an invalid input
2. The scenario returns to step 3.

5. Delete a product

Primary actor: Administrator

Main Success Scenario:

1. The admin selects the *admin* button from the main window
2. The admin logs in successfully
3. The admin introduces the ID of the product he/she wants to delete
4. The admin selects the *delete* button.
5. The application performs the delete operation and displays a message that it was successful.

Alternative sequence:

1. The admin introduces a non-existing ID
2. The scenario returns to step 3.

6. Generate reports

Primary actor: Administrator

Main Success Scenario:

1. The admin selects the *admin* button from the main window
2. The admin logs in successfully
3. The admin fills the required fields in order to generate a report
4. The admin selects the *generate* button
5. The application performs the generate reports operation.

Alternative sequence:

1. The admin leaves the fields blank
2. The scenario returns to step 3.

7. Add a composite product

Primary actor: Administrator

Main Success Scenario:

1. The admin selects the *admin* button from the main window
2. The admin logs in successfully
3. The admin selects the products he/she wants to add to the composite product
4. The admin selects the *add* button.
5. The admin provides a name for the new composite product
6. The admin selects the *create* button.
7. The application performs the insert operation and displays a message that it was successful.

Alternative sequence:

1. The admin introduces an invalid input
2. The scenario returns to step 3.

8. Create an order

Primary actor: Client

Main Success Scenario:

1. The client selects the *client* button from the main window.
2. The client logs in successfully.
3. The client selects the products he/she wants to add in the order
4. The client selects the *place* button from the client window.
5. The application performs the create operation.

Alternative sequence:

1. The client tries to create an empty order.
2. The scenario returns to step 3.

9. View menu’s products

Primary actor: Client

Main Success Scenario:

1. The client selects the *client* button from the main window.
2. The client logs in successfully.
3. The client selects the *view* button.
4. The application performs the operation and displays a table with all the products from the menu.

Alternative sequence – None.

10. Search a product

Primary actor: Client

Main Success Scenario:

1. The client selects the *client* button from the main window.
2. The client logs in successfully.
3. The client introduces the name of the product he/she wants to find
4. The application performs the search operation and lists the results (the names of the products containing the introduced keyword)

Alternative sequence:

1. The client introduces an invalid word (one that is not found or is not spelled correctly)
2. The scenario returns to step 3.

11. Register

Primary actor: Client

Main Success Scenario:

1. The client selects the *client* button from the main window.
2. The client selects the button *register* from the log in window.
3. The client introduces his/her username and password.
4. The application performs the register operation and saves the data introduced by the client.

Alternative sequence – None.

12. View orders

Primary actor: Employee

Main Success Scenario:

1. The employee selects the *employee* button from the main window.
2. The employee logs in successfully.
3. The employee selects the button *view* from the Employee Window.
4. The application performs the view operation and lists all the newly created orders.

Alternative sequence:

1. The employee introduces invalid log in credentials.
2. The scenario returns to step 1.

DESIGN

Level 1: Overall system design



Level 2: Division into sub-systems/packages

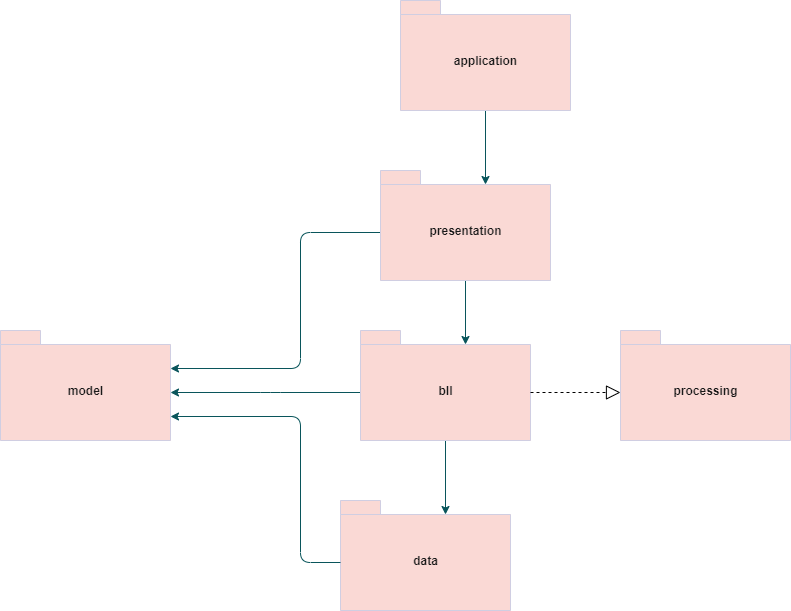
LAYERED ARCHITECTURE

The architecture used for this project is the layered architecture style. By using this style, the components/modules that have the same functionalities are organized into horizontal layers, meaning that each layer has a specific role in the application. Each layer uses the functions provided by the one below and so on.

When the user interacts with the application, the very first thing he/she sees is the Presentation Layer (the graphical user interface). After that, the Business Layer Logic takes care of the execution of the operations that the user wants to perform on the application (it coordinates the application, processes commands and makes logical decisions and evaluations). Since the Business Layer Logic requires data to work with, it will get the respective data from the Data Access Layer who is responsible for using serialization and deserialization to obtain the information about the DeliveryService object.

The big advantage when it comes to the layered architecture is that it provides a clear abstract view of the system as a whole while also providing enough detail to understand the roles and responsibilities of individual layers and the relationship between them.

My package diagram is the following:



Level 3: Division into packages

a) APPLICATION PACKAGE – contains the Main class with the main method that starts the application.

b) BLL PACKAGE – contains the DeliveryService class which deals with the business logic behind each operation performed on the application.

C) DATA PACKAGE – contains 3 classes:

* FileReader – used to extract and import all the entries from the .csv file in the DeliveryService.
* FileWriter – used to write the bill generated after placing an order.
* Serializator – used to read/write objects in the deserialization/serialization process in order to keep track of the state of the DeliveryService object.

D) MODEL PACKAGE – contains the classes which represent the model of the application:

* Order – the order placed by the client and stored in the DeliveryService’s hashMap
* Client – the user which has to register and provides its own data fields (username, password, a generated ID)
* MenuItem – an abstract class which models the products from the menu (because in the menu we can have base products and also composite products and they have in common the 7 fields)
* BaseProduct – a base product from the menu (which is inherited by MenuItem, so it is also a MenuItem)
* CompositeProduct – a composite product from the menu (which is inherited by MenuItem, so it is also a MenuItem)

E) PRESENTATION PACKAGE – contains all the classes which represents the windows for the graphical user interface:

* AppMainWindow – the first window that appears when we start the application and asks the user to select his/her role (admin, client or employee).
* LogInWindow: the second window – it requires the user to log in according to its role (for admin and employee there are pre-defined credentials) or permits a new client to register (create an account in the application).
* ClientWindow – the widow which contains all the buttons of the operations that the client can perform
* AdministratorWindow – the window which contains all the buttons of the operations that the client can perform
* EmployeeWindow – the window which contains all the buttons of the operations that the employee can perform
* Controller – it manages how the windows appear and initializes their buttons

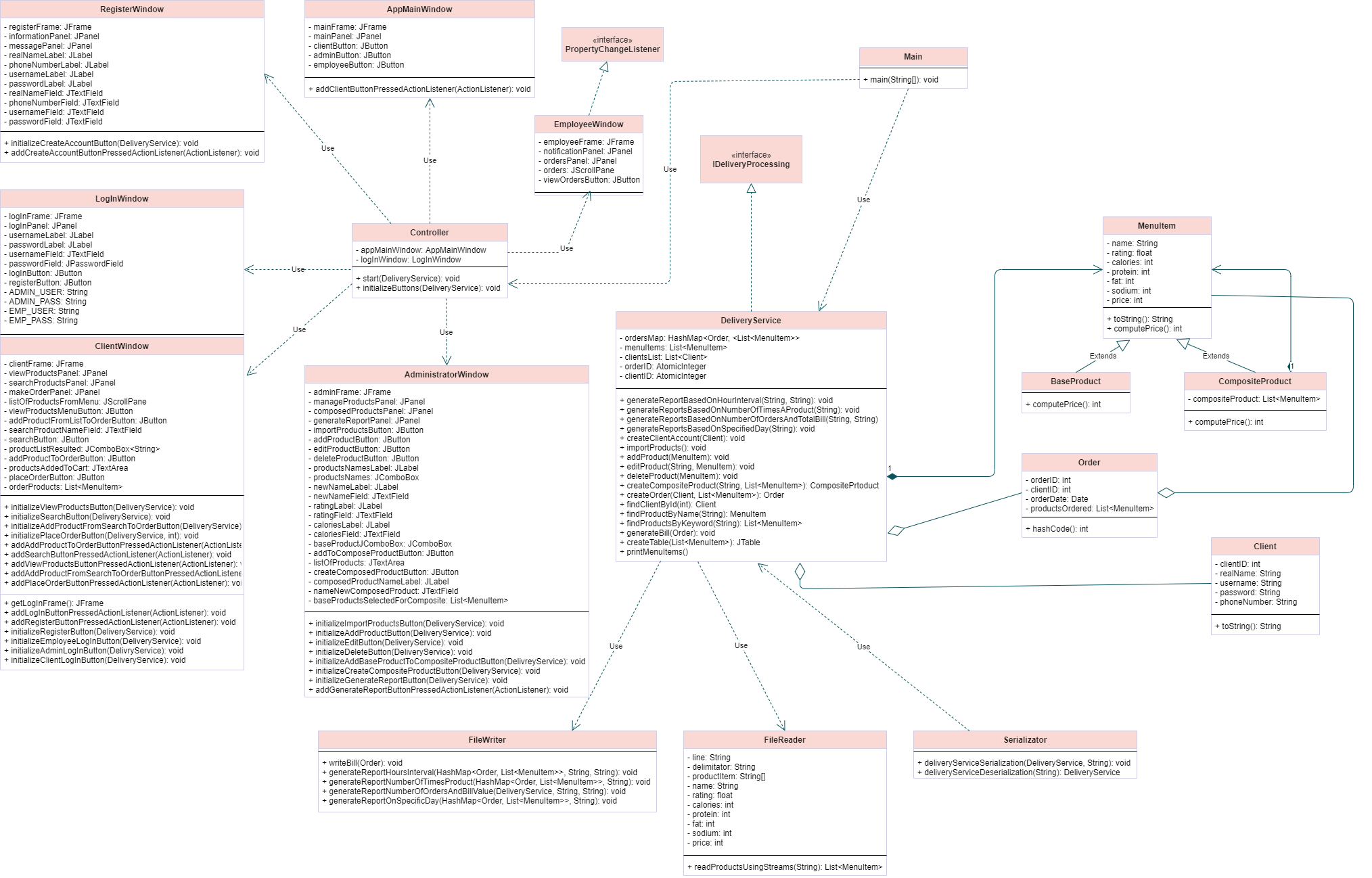
F) PROCESSING PACKAGE – contains the interface IDeliveryProcessing which contains the abstract declarations of the methods that the DeliverService has to deal with while processing a placemet of an order or a registration of a client.

DATA STRUCTURES USED

In DeliveryService class I have used ArrayList<MenuItem> in which I store all the items from the menu (after importing the products from the .csv file I add there all the new products created by the administrator in the application).

Also in DeliveryService class I have used a HashMap<Order, ArrayList<MenuItem>> to store all the orders taken by the clients. The key of the map is the order, computed with the function hashCode(OrderId, ClientId, Date). The ArrayList<MenuItem> in the values side is for storing all the products that compose the order placed by the client.

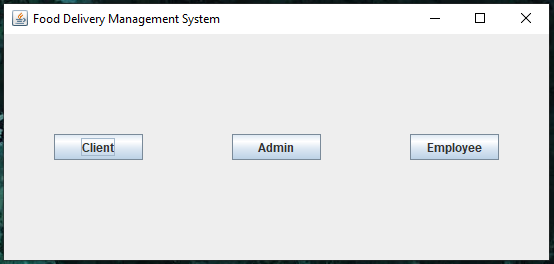
UML CLASS DIAGRAM



GUI DESIGN

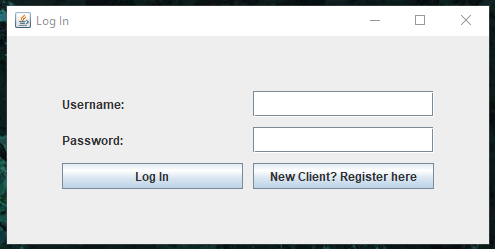
MAIN WINDOW

* Contains 3 buttons which represents the role of the user (each one leads to a log in window)

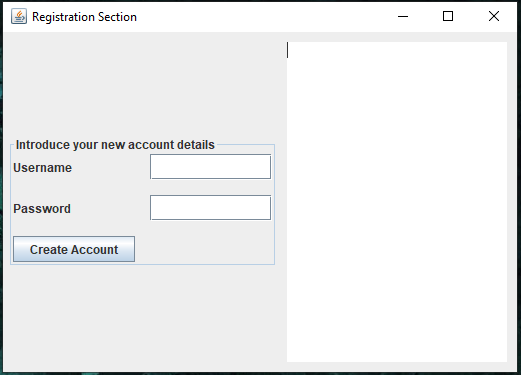


LOG IN WINDOW

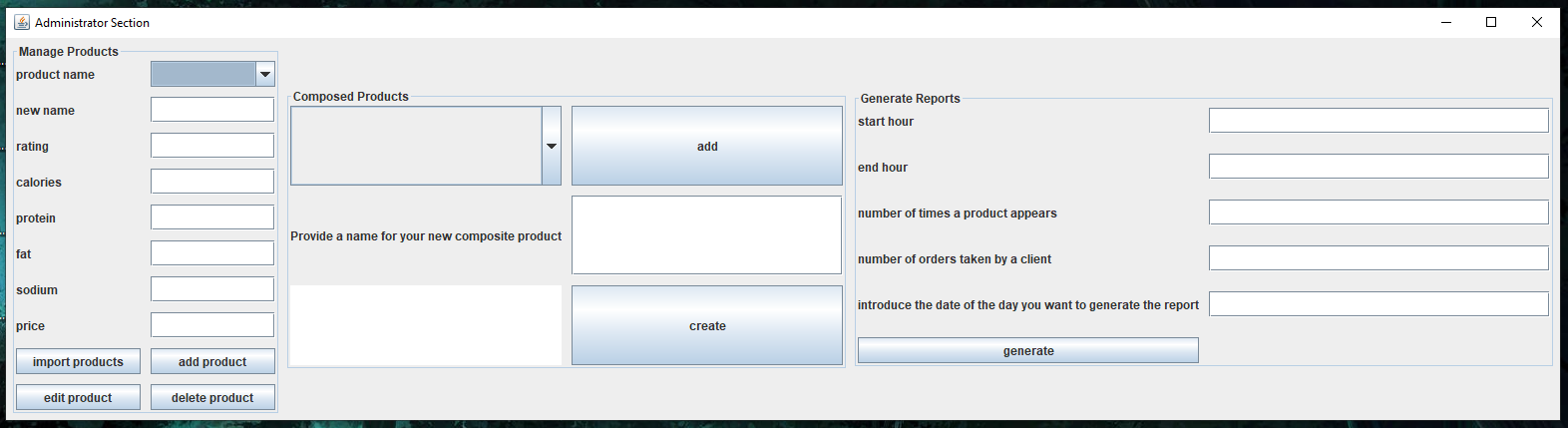
* Requires an username and a password to login (according to what button has been pressed before in the main window)
* Has a register button for the client (in case the user is an admin or an employee that button does not work, it’s just in the frame)



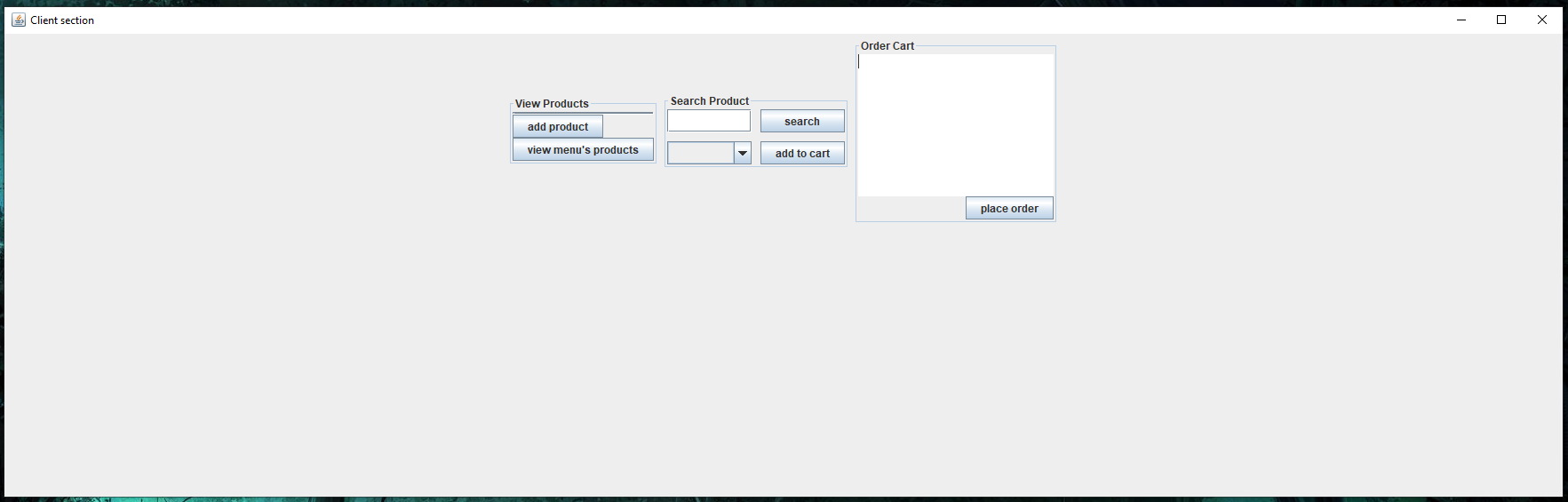
REGISTER WINDOW



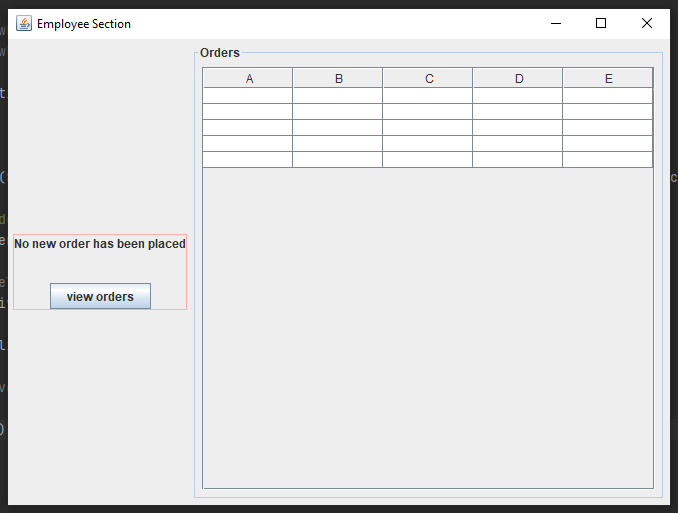
ADMIN WINDOW



CLIENT WINDOW



EMPLOYEE WINDOW



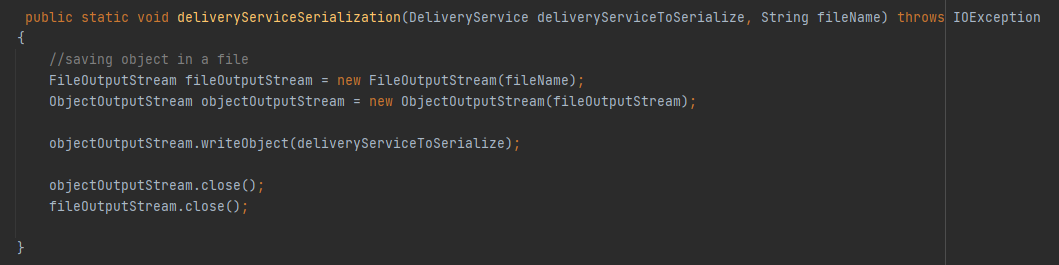
IMPLEMENTATION

For this assignment it was required to use serialization, stream processing and lambda expressions as new things to learn while programming in Java.

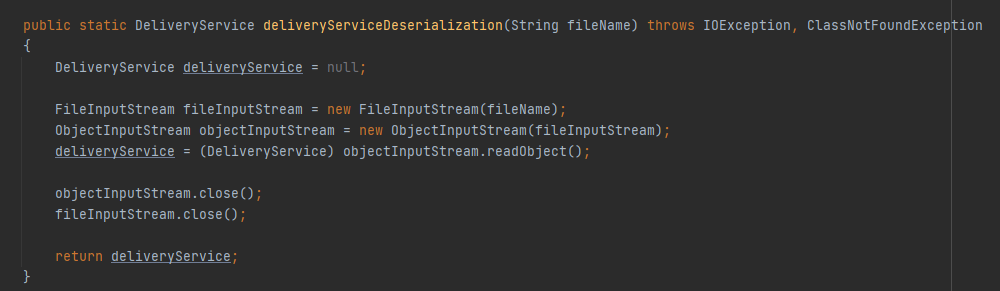
Serialization represents the way in which an objects is persisted from memory to a sequence of bits, while deserialization is the opposite – reads the data from the disk (or from the file where the object was serialized) to create an object.

The advantages provided by serialization is that it keeps track of an objects state if the object’s class implements the Serializable interface. Counting on serialization, it is possible to skip the part where a database connection was required to execute the insertion, delete and update operation on the products. Why? Because all the operations performed on the products list from the serializable DeliveryService object is stored on the generated file .ser and are known at deserialization when the application is run again. That is why it can be said that an objects is persisted (because it continues to exist past some time and past some actions).

Below is the method for serializing an object (saving its state into a file).

**

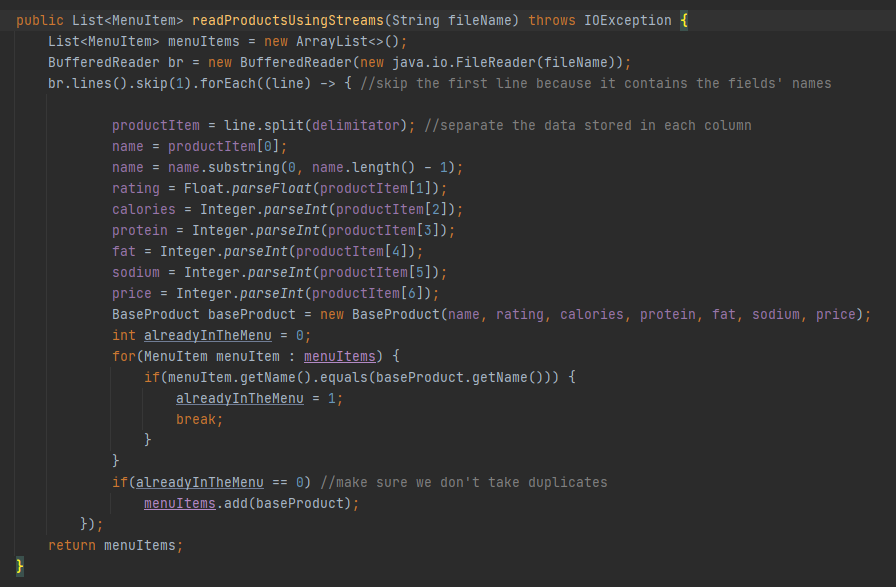
Below is the method for deserializing an object (get its data from the file).



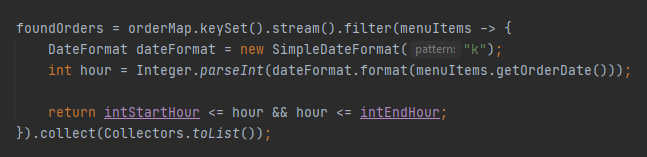
A Stream represents a sequence of elements on which one or more operations can be performed. Stream operations are either intermediate or terminal. While terminal operations return a result of a certain type, intermediate operations return the stream itself so you can chain multiple method calls in a row. Stream operations can either be executed sequential or in parallel.

Another advantage that stream processing brings in Java is the code’s clean and short look. It simplifies a lot how the code looks. Together with lambda expressions, it is possible to replace anonymous objects with a simple parameter.

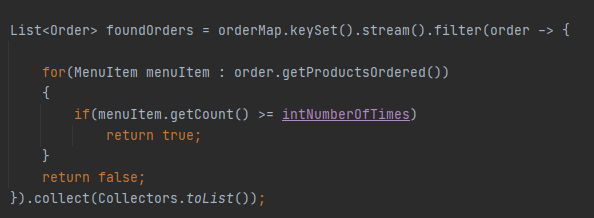
Below is an example where I have used streams: (in parsing the elements from the .csv file)



Another examples:



In the example above, I have used streams to find the orders from the deliveryService’s HashMap which were placed between a specific time interval (which limits were determined by 2 parameters – startHour and endHour).



In this example, I have used streams to find the orders which had products that have been ordered more than a specific time (e.g an order which had a product that has been ordered more than 5 times so far, not just in that respective order).

CONCLUSION

This assignment as a whole was more complex than the others and have introduced us to the streams and serialization approaches. You can see their utility when you have to present your application on a different device (and still want to have the previous operations’ results that you performed, so that they can be seen after deserialization on a device) and when you have to develop a bigger project (like this one) and to simplify the way you write code (as lambda expressions and streams make it easier to have a clean and short code).

BIBLIOGRAPHY

<https://stackoverflow.com/questions/46380073/observer-is-deprecated-in-java-9-what-should-we-use-instead-of-it>

<https://stackabuse.com/the-observer-design-pattern-in-java/>

<https://www.javatpoint.com/how-to-read-csv-file-in-java>

<https://stackoverflow.com/questions/3429921/what-does-serializable-mean>

<https://blog.codeleak.pl/2014/05/parsing-file-with-stream-api-in-java-8.html>

<https://winterbe.com/posts/2014/03/16/java-8-tutorial/>

<https://www.baeldung.com/java-maps-streams>

<https://beginnersbook.com/2017/10/java-8-filter-a-map-by-keys-and-values/>

<https://www.tutorialspoint.com/how-to-compare-two-dates-in-java>

<https://www.techiedelight.com/count-frequency-elements-list-java/>

<https://www.baeldung.com/java-check-two-dates-on-same-day>