**FOOD DELIVERY MANAGEMENT SYSTEM**

**Neag Dragoș-Ion**

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.

**Sub-objectives**

* Analyze the problem and identify requirements.
* Design the order system and choose an appropriate representation of the clients, products and orders.
* Implement the management system.
* Test the food delivery management system.

1. **Problem analysis**
2. Modeling

The purpose of the problem is creating a manager, able to handle the database and mold it around the user interface.

To achieve the purpose, a series of requirements had to be met:

* Functional requirements
* The delivery management system should allow users to register new clients
* The delivery management system should allow the administrator to insert new products
* The delivery management system should allow the administrator to delete products
* The delivery management system should allow the administrator to edit products
* The delivery management system should allow the administrator to view products
* The delivery management system should allow the administrator to generate reports
* The delivery management system should allow users to add products to an order
* The delivery management system should allow users to place an order
* Non-Functional requirements
* The delivery management system should be intuitive and easy to use by the user
* The delivery management system should indicate the user when the format he typed in is wrong

1. Scenarios

Diagram

Description automatically generated

**Use Case:** register client

**Primary Actor:** user

**Main Success Scenario:**

Step1. The user presses the “create new user” button

Step2. The user inserts the fields of the client, which are the username, password, name, phone number, and adress in the graphical user interface.

Step3. The user presses the “create” button

Step4. The manager inserts the client into the memory.

**Alternative Sequence:** Incorrect input values

* The user inserts incorrect input values (e.g. inserts digits in the name, some fields are empty,)
* The scenario returns to step 2

**Use Case:** insert product in order and place order

**Primary Actor:** user

**Main Success Scenario:**

Step1. The user logs into the application

Step2. The user selects an item by clicking on it

Step2. The user presses the “add” button

Step3. The user repeats the first two steps if desired

Step4. The user presses the “order” button

Step5. The manager inserts the order into the memory.

**Alternative Sequence:** Incorrect input values

* The admin selects more than one product at once
* The scenario returns to step 2

**Alternative Sequence:** Incorrect input values

* The admin does not add any product to the order
* The scenario returns to step 2

**Use Case:** insert product

**Primary Actor:** administrator

**Main Success Scenario:**

Step1. The admin logs into the application

Step2. The admin presses the “add” button

Step3. The admin inserts the fields of the product, which are the name, rating, calories, proteins, fats, sodium and price in the graphical user interface.

Step4. The admin presses the “add” button

Step5. The manager inserts the product into the memory.

**Alternative Sequence:** Incorrect input values

* The admin inserts incorrect input values (e.g. inserts digits in the name, some fields are empty,)
* The scenario returns to step 2

**Use Case:** delete product

**Primary Actor:** administrator

**Main Success Scenario:**

Step1. The admin logs into the application

Step2. The admin selects a product by clicking on it

Step3. The admin presses the “delete” button

Step4. The manager deletes the product from the memory.

**Alternative Sequence:** Incorrect input values

* The admin selects more than one product
* The scenario returns to step 2

**Use Case:** edit product

**Primary Actor:** administrator

**Main Success Scenario:**

Step1. The admin logs into the application

Step2. The admin presses the “edit” button

Step3. The admin selects a product by clicking on it

Step4. The admin presses the “display” button

Step5. The admin edits the fields of the product, which are the name, rating, calories, proteins, fats, sodium and price in the graphical user interface.

Step6. The manager edits the product in the memory.

**Alternative Sequence:** Incorrect input values

* The admin selects more than one product
* The scenario returns to step 2

**Alternative Sequence:** Incorrect input values

* The admin inserts incorrect input values (e.g. inserts digits in the name, some fields are empty)
* The scenario returns to step 2

**Use Case:** generate report

**Primary Actor:** administrator

**Main Success Scenario:**

Step1. The admin logs into the application

Step2. The admin inserts the data into the fields related to generating reports

Step3. The admin presses the “generate” button

Step4. The manager generates the corresponding report.

**Alternative Sequence:** Incorrect input values

* The admin inserts incorrect input values (e.g. inserts digits in the name, some fields are empty)
* The scenario returns to step 2

1. **Design**

Level 1: Overall system design

Diagram

Description automatically generated

Level 2: Division into packages

Diagram

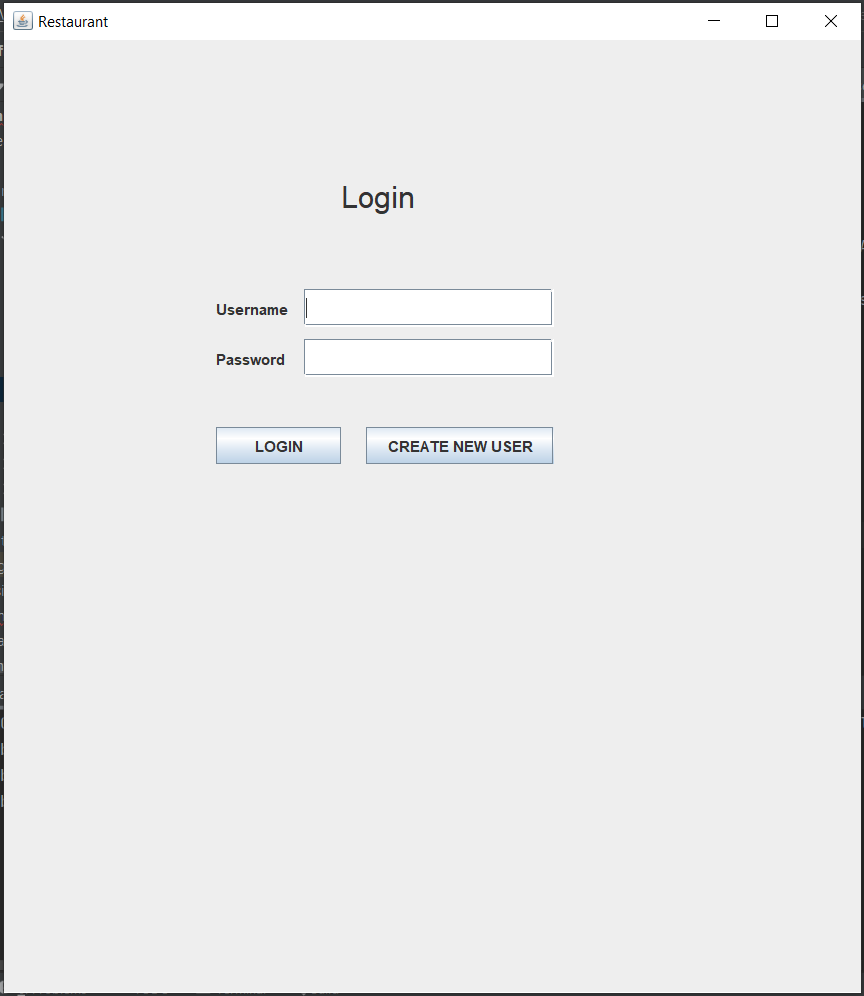
Description automatically generated

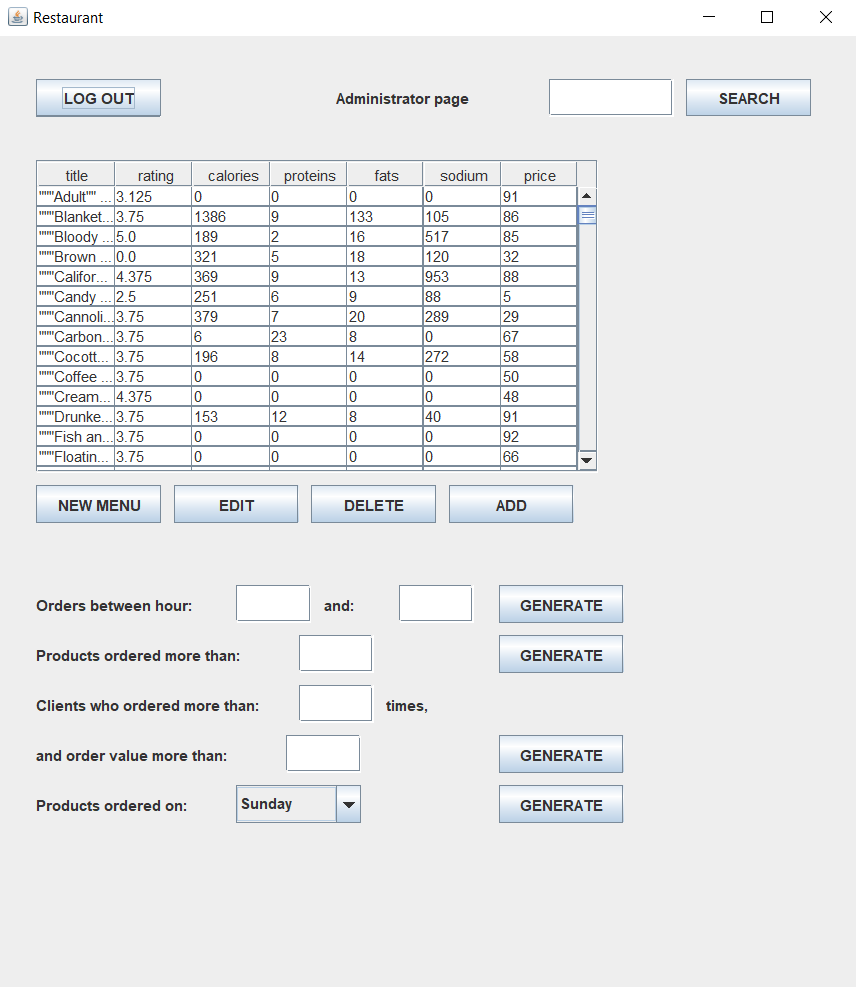
Diagram

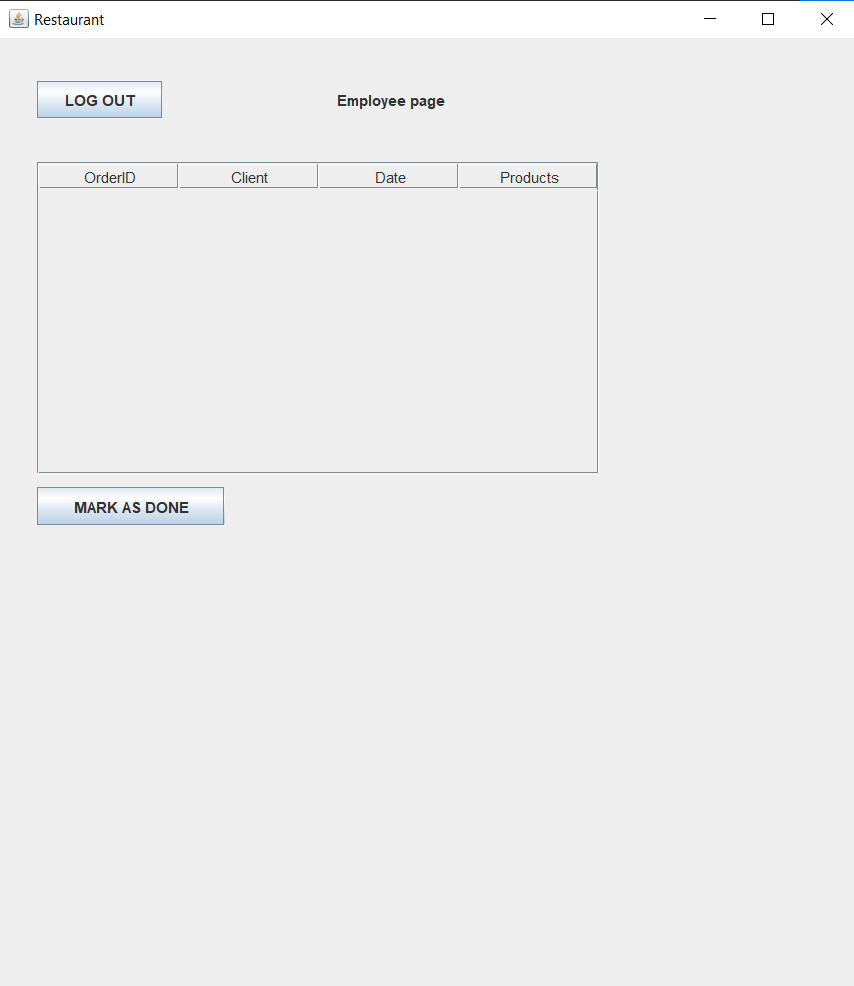
Description automatically generatedLevel 3: Division into classes

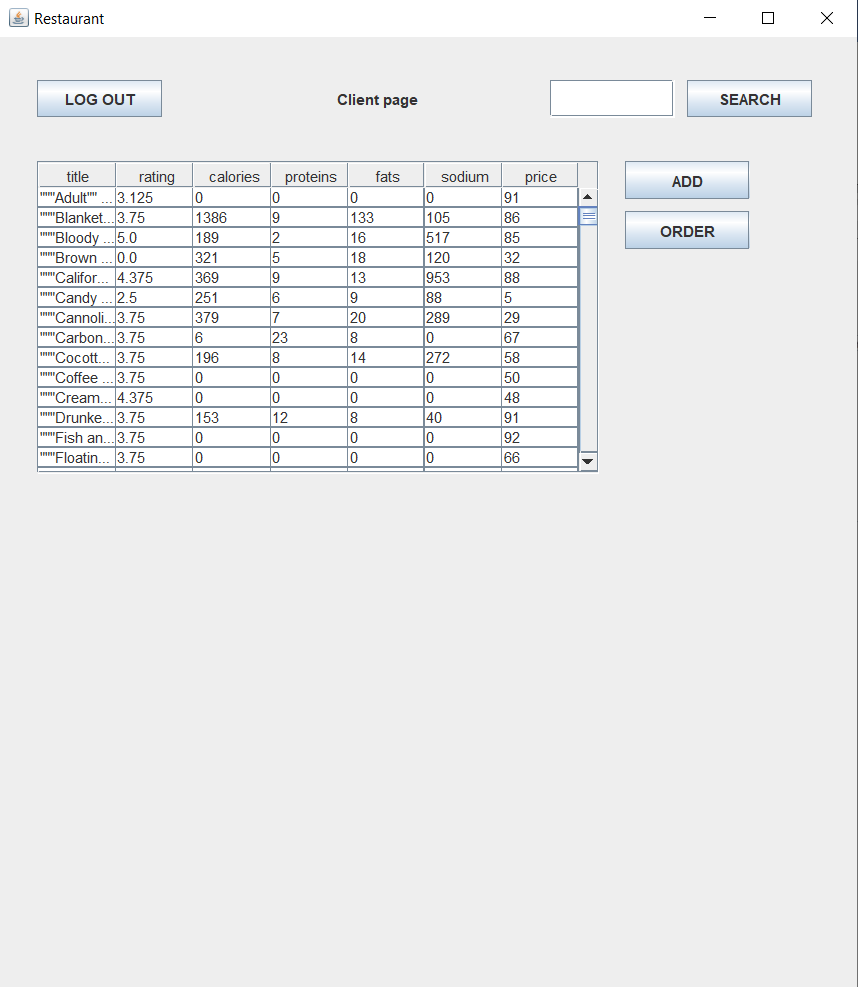
* Main package
* Start – initializes the view class, meaning it starts the GUI
* Presentation package
* MainView – the main part of the GUI. It holds the principal frame and it initializes the button listeners. It is also responsible with the client frame, the product frame and the order frame, as it performs operations, validating the data before attempting to enter the stored data.
* Employee – represents an entity, corresponding to an employer, but mainly used to communicate with the Observer
* Observer – interface related to the Observer Design Pattern, with only one method, update
* Data package
* Client – represents an entity, with all its fields: id, name, username, password, phone, address
* FileWriter – responsible with serializing the products, the clients and the orders every time a change takes place in the deliveryService
* TableManager – as its name suggests, the class manages tables, that meaning it takes the data and it displays it into tables, for the products as well as for the orders
* Serializator – this class is responsible with managing the “database”. It transforms the objects into text and stores it into a text file. That file is later accessible again, when the deserialization takes place, the text being converted back into objects.
* Validator - is responsible with checking the format of the provided client or product object
* Business package
* IDeliveryServiceProcessing – contains the main operations that can be executed by the administrator and client, as follows: the administrator: import products, manage the products from the menu, generate reports and the client: create new order which implies computing the price for an order and generating a bill in .txt format, searching for products based on several criteria.
* DeliveryService – implements the IDeliveryServiceProcessing, and it contains the main operations executed by the administrator and the client
* MenuItem - represents an entity, part of the Composite Design Pattern
* BaseProduct – represents an entity, with all its fields: name, rating, calories, proteins, fats, sodium and price, and it is part of the Composite Design Pattern
* CompositeProduct – represents an entity, with all its fields: title, menuItemsList, and it is part of the Composite Design Pattern, as it is a product composed of other base or composed products
* Order - represents an entity, with all its fields: orderID, clientID, orderDate, menuItemList
* Observable - responsible of informing all the observers about any change, as well as adding or removing any observers

**GUI Design**









1. **Implementation**

* DeliveryService implementation
* Edit product from the stored ones

Text

Description automatically generated

* Generate report using lambda expressions

Text

Description automatically generated

* IDeliveryServiceProcessing interface

Text

Description automatically generated

* Serializator implementation
* Serializing a list of objects

Text

Description automatically generated

* FileWriter implementation
* Method that serializes every set of objects when a change occurs

Text

Description automatically generated

* TableManager implementation
* Method that fills up the table of the orders seen by the employee

Text

Description automatically generated

* Search method depending on a keyword, implemented with lambda

Text

Description automatically generated

* Observable class
* All the important methods used to add/remove and notify observers

Text

Description automatically generated

1. **Results**

* Registering a new client

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Table

Description automatically generated

* Successfully placing an order as a client

Graphical user interface, table

Description automatically generated

Graphical user interface, application

Description automatically generated

* The administrator editing a product

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

* The administrator generating a report based on the products ordered on monday

Graphical user interface

Description automatically generated

* Bill.txt

A picture containing text

Description automatically generated

* Report4.txt

Scatter chart

Description automatically generated with low confidence

1. **Conclusions**

In the end, I would say the application turned out pretty good and can be useful in certain situations. Although is far from perfect, I think is a very effective homework, even though it is pretty basic.

As when it comes to further development, I don’t think it is necessary from me to mention it, as this type of apps are very popular nowadays, new ones being launched every day.

This project helped me learn something new and made me understand how to build a database and an application on top of it on my own. I will definitely use these concepts in the future.