**FOOD DELIVERY MANAGEMENT SYSTEM**

Tema 4

DOCUMENTAȚIE

**Gavrilescu Andreea-Lavinia**

**Grupa 30228**

# CUPRINS

[1. **Obiectivul temei**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297885) 1

[2. **Analiza problemei, modelare, scenarii, cazuri de utilizare**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297886) 2

[3. **Proiectare**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297887) 8

[4. **Implementare**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297888) 13

[5. **Rezultate**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297889) 31

[6. **Concluzii**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297890) 32

[7. **Bibliografie**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297891) 33

Obiectivul temei

**Obiective principale**

* Proiectarea și implementarea unei aplicații care are ca scop administrarea comenzilor pentru o firmă de catering.

**Obiective secundare**

* Analizarea problemei și identificarea cerințelor
* Proiectarea aplicației de administrare a comenzilor
* Implementarea aplicației de administrare a comenzilor
* Testarea aplicației de administrare a comenzilor

Analiza problemei, modelare, scenarii, cazuri de utilizare

**Analiza**

Administrator

FOOD DELIVERY MANAGEMENT SYSTEM

Client

Angajat

**Cerințe funcționale**

* Aplicația de administrare ar trebui să permită administratorului să își fac un cont
* Aplicația de administrare ar trebui să permită administratorului să intre în aplicație
* Aplicația de administrare ar trebui să permită administratorului să introducă un nou produs
* Aplicația de administrare ar trebui să permită administratorului să modifice un produs
* Aplicația de administrare ar trebui să permită administratorului să șteargă un produs
* Aplicația de administrare ar trebui să permită administratorului să creeze un meniu
* Aplicația de administrare ar trebui să permită clientului să își fac un cont
* Aplicația de administrare ar trebui să permită clientului să intre în aplicație
* Aplicația de administrare ar trebui să permită clientului să plaseze comenzi
* Aplicația de administrare ar trebui să permită angajatului să își fac un cont
* Aplicația de administrare ar trebui să permită angajatului să intre în aplicație
* Aplicația de administrare ar trebui să permită angajatului să vizualizeze comenzile

**Cerințe non-funcționale**

* Aplicația de administrare ar trebui să fie interactivă
* Aplicația de administrare ar trebui să fie ușor de folosit de către utilizator
* Aplicația de administrare ar trebui să facă deosebire între acțiunile pe care le poate executa un client/administrator/angajat

**Cazuri de utilizare**

* Caz de utilizare:log in

Actor principal: Client/Administrator/Angajat

Scenariu principal

1. Completează câmpul pentru username și parolă

2. Administratorul selectează opțiunea pentru “Client/Administrator/Angajat”

3. Administratorul apasă pe butonul de “REGISTER”

4. Aplicația deschide fereastra corespunzătoare

Scenariu alternativ

1. Nu introduce date în toate câmpurile necesare

2. Aplicația un mesaj de eroare corespunzător

3. Scenariul se întoarce la punctul 1

* Caz de utilizare:register

Actor principal: Client/Administrator/Angajat

Scenariu principal

1. Completează câmpul pentru username și parolă

2. Administratorul apasă pe butonul de “LOG IN”

3. Aplicația deschide fereastra corespunzătoare

Scenariu alternativ

1. Nu introduce date în toate câmpurile necesare

2. Aplicația un mesaj de eroare corespunzător

3. Scenariul se întoarce la punctul 1

* Caz de utilizare:adăugare produs

Actor principal: administrator

Scenariu principal

1. Administratorul completează câmpurile necesare cu informațiile produsului
2. Administratorul apasă pe butonul de “ADD”
3. Aplicația introduce produsul în meniu și afișează un mesaj corespunzător

Scenariu alternativ

1. Administratorul nu introduce date în toate câmpurile necesare
2. Aplicația un mesaj de eroare corespunzător
3. Scenariul se întoarce la punctul 1

* Caz de utilizare:modificare produs

Actor principal: administrator

Scenariu principal

1. Administratorul completează câmpurile necesare cu noile informații ale produsului

2. Administratorul apasă pe butonul de “MODIFY”

3. Aplicația introduce produsul în meniu și afișează un mesaj corespunzător

Scenariu alternativ

4. Administratorul nu introduce date în toate câmpurile necesare

5. Aplicația un mesaj de eroare corespunzător

6. Scenariul se întoarce la punctul 1

* Caz de utilizare:ștergere produs

Actor principal: administrator

Scenariu principal

1. Administratorul selectează produsul

2. Administratorul apasă pe butonul de “DELETE”

3. Aplicația șterge produsul din meniu și afișează un mesaj corespunzător

Scenariu alternativ

4. Administratorul nu introduce date în toate câmpurile necesare

5. Aplicația un mesaj de eroare corespunzător

6. Scenariul se întoarce la punctul 1

* Caz de utilizare:creare meniu

Actor principal: administrator

Scenariu principal

1. Administratorul completează câmpul pentru titlul meniului

2. Administratorul selectează produsele dorite

3. Administratorul apasă pe butonul de “CREATE”

4. Aplicația introduce un nou meniu și afișează un mesaj corespunzător

Scenariu alternativ

1. Administratorul nu introduce date în toate câmpurile necesare

2. Aplicația un mesaj de eroare corespunzător

3. Scenariul se întoarce la punctul 1

* Caz de utilizare:generare raport

Actor principal: administrator

Scenariu principal

1. Administratorul completează câmpul pentru numărul raportului dorit

2. Administratorul introduce un parametru sau doi în funcție de raport

3. Administratorul apasă pe butonul de “REPORT”

4. Aplicația afișează informațiile corespunzătoare în fișier

Scenariu alternativ

1. Administratorul nu introduce date în toate câmpurile necesare

2. Aplicația un mesaj de eroare corespunzător

3. Scenariul se întoarce la punctul 1

* Caz de utilizare:căutare produse

Actor principal: client

Scenariu principal

1. Clientul completează câmpul pentru criteriul dorit

2. Clientul introduce un număr/nume după care să se facă căutarea

3. Clientul apasă pe butonul de “SEARCH”

4. Aplicația afișează produsele corespunzătoare

Scenariu alternativ

1. Administratorul nu introduce date în toate câmpurile necesare

2. Aplicația un mesaj de eroare corespunzător

3. Scenariul se întoarce la punctul 1

* Caz de utilizare:plasare comandă

Actor principal: client

Scenariu principal

1. Clientul completează câmpul pentru ID-ul comenzii

2. Clientul selectează produsele dorite

3. Clientul apasă pe butonul de “CREATE”

4. Aplicația introduce comanda și afișează un mesaj corespunzător

Scenariu alternativ

1. Administratorul nu introduce date în toate câmpurile necesare

2. Aplicația un mesaj de eroare corespunzător

3. Scenariul se întoarce la punctul 1

* Caz de utilizare:vizualizare comandă

Actor principal: angajat

Scenariu principal

1. Angajatul vizualizează comenzile plasate de clienți

Proiectare

**Design general**

Angajat

Comandă

Meniu

Date

Administrator

Client

**Divizare în pachete**

Conține clasele care implementează interfața

Presentation Layer

Business Logic

Conține clasele care implementează operațiile

Model

Conține clasele care conțin query-urile și conexiunea cu baza de date

Data Layer

Conține clasele care modelează datele aplicației relaționate cu baza de date

Pentru afișarea rezulatelor potrivite confrom datelor introduse de utilizatori prin interfața grafică voi folosi un Model-View-Controller pattern care împarte aplicația în 3 categorii: procesare, input și output.

* Componentele Model – încapsulează datele și funcționalitatea
* Componentele View – afișează informația utilizatorului – obține datele afișate din model
* Controller – fiecare view are un controller asociat; acesta primește un input, de obicei un eveniment care denotă mișcarea mouse-ului, apăsarea mouse-ului sau a tastaturii; aceste evenimente sunt traduse ca apeluri care sunt trimise către model sau view.

Controller-ul modifică apeluri View-ului în funcție de modificările din Model

Datele din Model sunt folosite în View

Model

View-ul poate cere date din Model

View

Controller

Interacțiunea cu View-ul face apeluri către Controller

**Divizare în clase**

BaseProduct

CompositeProduct

DeliveryService

IDeliveryServiceProcessing

Business

MenuItem

Observable

Order

User

UserService

Data

MyFileWriter

ClientController

AdministratorController

Controller

EmployeeController

RegisterController

ClientView

AdministratorView

View

EmployeeView

RegisterView

Utils

MyComparator

Start

Main

Observable

O imagine care conține text

Descriere generată automat

Implementare

**1. Clasa BaseProduct** – extinde clasa MenuItem, având ca atribute private, numele și caracteristicile unui produs, însoțite de metodele standard de get() și set(), 2 constructori, metoda computePrice() din clasa abstractă MenuItem, precum si cea de toString().

private String title;  
private double rating;  
private double calories;  
private double protein;  
private double fat;  
private double sodium;  
private double price;  
  
*/\*\*  
 \* Constructor pentru clasa BaseProduct  
 \*  
 \** ***@param*** *title numele produsului  
 \** ***@param*** *price pretul produsului  
 \*/*public BaseProduct*(*String title, double price*) {* super*(*title, price*)*;  
*}  
  
/\*\*  
 \* Constructor pentru clasa BaseProduct  
 \*  
 \** ***@param*** *title numele produsului  
 \** ***@param*** *rating rating-ul produsului  
 \** ***@param*** *calories numarul de calorii ale produsului  
 \** ***@param*** *protein numarul de proteine ale produsului  
 \** ***@param*** *fat numarul de grasimi ale produsului  
 \** ***@param*** *sodium numarul de sodiu al produsului  
 \** ***@param*** *price pretul produsului  
 \*/*public BaseProduct*(*String title, double rating, double calories, double protein, double fat, double sodium, double price*) {* super*(*title, rating, calories, protein, fat, sodium, price*)*;  
 this.title = title;  
 this.rating = rating;  
 this.calories = calories;  
 this.protein = protein;  
 this.fat = fat;  
 this.sodium = sodium;  
 this.price = price;  
*}  
  
/\*\*  
 \* Metoda pentru calcularea pretului pentru un BaseProduct  
 \*  
 \** ***@return*** *pretul  
 \*/*@Override  
public double computePrice*() {* return super.getPrice*()*;  
*}*

**2. Clasa CompositeProduct** – extinde clasa MenuItem, având ca atribute private, numele, prețul și o listă de însoțite de metodele standard de get(), set(), computePrice() din clasa abstractă MenuItem.

private String title;  
private double price;  
private Collection*<*MenuItem*>* menuItems;  
  
*/\*\*  
 \* Constructor pentru clasa CompositeProduct  
 \*  
 \** ***@param*** *title numele produsului  
 \** ***@param*** *items pretul produsului  
 \*/*public CompositeProduct*(*String title, Collection*<*MenuItem*>* items*) {* super*(*title, 0*)*;  
 this.title = title;  
 this.menuItems = items;  
 this.price = computePrice*()*;  
*}  
  
/\*\*  
 \* Metoda de calculare a pretului unui CompositeProduct  
 \*  
 \** ***@return*** *pretul  
 \*/*@Override  
public double computePrice*() {* double total = 0;  
  
 for *(*MenuItem m : menuItems*) {* total += m.getPrice*()*;  
 *}* super.setPrice*(*total*)*;  
  
 return total;  
*}*

3. **Clasa DeliveryService** – are ca atribute private, o listă de observers, un hashMap de comenzi și un treeSet de produse din meniu, însoțite de 1 constructor, metodele standard de get() și set(), precum și:

*/\*\*  
 \* Metoda de creare a unor comenzi  
 \*/*

void generateOrders*()*

* Hardcodare unor comenzi

*/\*\*  
 \* Metoda care importa informatiile despre produse din fisier  
 \*  
 \** ***@throws*** *IOException exceptie  
 \*/*@Override  
public void importProducts*()* throws IOException *{* BufferedReader csvReader = new BufferedReader*(*new FileReader*(*"D:\\Facultate\\UTCN\\An II\\Semestrul 2\\Tehnici de programare\\Laboratoare\\PT2022\_30228\_Gavrilescu\_AndreeaLavinia\_4\\src\\main\\resources\\products.csv"*))*;  
 String row;  
  
 row = csvReader.readLine*()*;  
  
 while *((*row = csvReader.readLine*())* != null*) {* String*[]* data = row.split*(*","*)*;  
  
 String title;  
 double rating, calories, protein, fat, sodium, price;  
 title = data*[*0*]*.strip*()*;  
 rating = Double.*parseDouble(*data*[*1*])*;  
 calories = Double.*parseDouble(*data*[*2*])*;  
 protein = Double.*parseDouble(*data*[*3*])*;  
 fat = Double.*parseDouble(*data*[*4*])*;  
 sodium = Double.*parseDouble(*data*[*5*])*;  
 price = Double.*parseDouble(*data*[*6*])*;  
  
 MenuItem item = new BaseProduct*(*title, rating, calories, protein, fat, sodium, price*)*;  
  
 menuItems.add*(*item*)*;  
 *}* csvReader.close*()*;  
*}*

Metodele de adăugare/modificare/ștergere a unui produs:

*/\*\*  
 \* Metoda de adaugare a unui produs  
 \*  
 \** ***@param*** *product produsul care trebuie adaugat  
 \*/*@Override  
public void addProduct*(*MenuItem product*) {* if *(*!menuItems.contains*(*product*))* menuItems.add*(*product*)*;  
*}  
  
/\*\*  
 \* Metoda de stergere a unui produs  
 \*  
 \** ***@param*** *product produsul care trebuie sters  
 \*/*@Override  
public void deleteProduct*(*MenuItem product*) {* for *(*MenuItem m : menuItems*) {* if *(*m.getTitle*()*.equals*(*product.getTitle*())) {* menuItems.remove*(*m*)*;  
 return;  
 *}  
 }  
}  
  
/\*\*  
 \* Metoda de modificare a unui produs  
 \*  
 \** ***@param*** *oldItem vechiul produs  
 \** ***@param*** *newItem noul produs  
 \*/*@Override  
public void modifyProduct*(*MenuItem oldItem, MenuItem newItem*) {* oldItem.setTitle*(*newItem.getTitle*())*;  
 oldItem.setRating*(*newItem.getRating*())*;  
 oldItem.setCalories*(*newItem.getCalories*())*;  
 oldItem.setProtein*(*newItem.getProtein*())*;  
 oldItem.setFat*(*newItem.getFat*())*;  
 oldItem.setSodium*(*newItem.getSodium*())*;  
 oldItem.setPrice*(*newItem.getPrice*())*;  
*}*

Metoda de creare a unei comenzi:

*/\*\*  
 \* Metoda de creare a unei comenzi  
 \*  
 \** ***@param*** *orderID ID-ul comenzii  
 \** ***@param*** *clientID ID-ul clientului  
 \** ***@param*** *orderDate data comenzii  
 \** ***@param*** *items produsele continute de comanda  
 \*/*@Override  
public void createOrder*(*int orderID, int clientID, LocalDateTime orderDate, Collection*<*MenuItem*>* items*) {* Order order = new Order*(*orderID, clientID, orderDate*)*; // cream o comanda   
  
 double price = 0; // intial pretul e 0  
   
 for *(*MenuItem i : items*) {* price += i.getPrice*()*; // calculam pretul total al comenzii  
 *}* order.setPrice*(*price*)*; // setam pretul comenzii  
  
 orders.put*(*order, items*)*; // punem comanda in lista de comenzi  
  
 this.notifyObservers*(*order*)*; // trimitem o notificare catre angajati  
*}*

Metoda de căutare a produselor după nume:

*/\*\*  
 \* Metoda de cautare a unui produs dupa titlu  
 \*  
 \** ***@param*** *title titlul pe care il cautam  
 \** ***@return*** *produsele corespunzatoare cautarii  
 \*/*@Override  
public MenuItem searchProducts*(*String title*) {* for *(*MenuItem m : menuItems*) {* if *(*m.getTitle*()*.equals*(*title*))* // cautare dupa nume  
 return m;  
 *}* return null;  
*}*

Metodele de adăugare/ștergere/notificare observers:

*/\*\*  
 \* Metoda de adaugare a unui observator  
 \*  
 \** ***@param*** *o observatorul care trebuie adaugat  
 \*/*@Override  
public void addObserver*(*Observer o*) {* this.observers.add*(*o*)*;  
*}  
  
/\*\*  
 \* Metoda de stergere a unui observator  
 \*  
 \** ***@param*** *o observatorul care trebuie sters  
 \*/*@Override  
public void removeObserver*(*Observer o*) {* this.observers.remove*(*o*)*;  
*}  
  
/\*\*  
 \* Metoda de notificare a tuturor observatorilor  
 \*  
 \** ***@param*** *order comanda in functie de care se vor trimite notificari  
 \*/*@Override  
public void notifyObservers*(*Order order*) {* for *(*Observer o : this.observers*)* o.update*(*order*)*;  
*}*

Urmează metodele de filtrare folosind stream-uri:

*/\*\*  
 \* Metoda de filtrare dupa titlu  
 \*  
 \** ***@param*** *keyword titlul dupa care se va face filtrarea  
 \** ***@return*** *lista de produse corespunzatoare  
 \*/*private List*<*MenuItem*>* filterByTitle*(*String keyword*) {* return this.menuItems  
 .stream*()* .filter*(*item -> item.getTitle*()*.toLowerCase*()*.contains*(*keyword.toLowerCase*()))* .collect*(*Collectors.*toList())*; // collects the result of the filter into a List<MenuItem>  
*}*private List*<*MenuItem*>* filterByRating*(*String keyword*) {* try *{* double rating = Double.*parseDouble(*keyword*)*;  
  
 return this.menuItems  
 .stream*()* .filter*(*item -> item.getRating*()* == rating*)* .collect*(*Collectors.*toList())*;  
 *}* catch *(*NumberFormatException ex*) {* return null;  
 *}  
}  
  
/\*\*  
 \* Metoda de filtrare dupa numarul de calorii  
 \*  
 \** ***@param*** *keyword numarul de calorii dupa care se va face filtrarea  
 \** ***@return*** *lista de produse corespunzatoare  
 \*/*private List*<*MenuItem*>* filterByCalories*(*String keyword*) {* try *{* double calories = Double.*parseDouble(*keyword*)*;  
  
 return this.menuItems  
 .stream*()* .filter*(*item -> item.getCalories*()* == calories*)* .collect*(*Collectors.*toList())*;  
 *}* catch *(*NumberFormatException ex*) {* return null;  
 *}  
}  
  
/\*\*  
 \* Metoda de filtrare dupa numarul de proteine  
 \*  
 \** ***@param*** *keyword numarul de proteine dupa care se va face filtrarea  
 \** ***@return*** *lista de produse corespunzatoare  
 \*/*private List*<*MenuItem*>* filterByProteins*(*String keyword*) {* try *{* double proteins = Double.*parseDouble(*keyword*)*;  
  
 return this.menuItems  
 .stream*()* .filter*(*item -> item.getProtein*()* == proteins*)* .collect*(*Collectors.*toList())*;  
 *}* catch *(*NumberFormatException ex*) {* return null;  
 *}  
}  
  
/\*\*  
 \* Metoda de filtrare dupa numarul de grasimi  
 \*  
 \** ***@param*** *keyword numarul de grasimi dupa care se va face filtrarea  
 \** ***@return*** *lista de produse corespunzatoare  
 \*/*private List*<*MenuItem*>* filterByFats*(*String keyword*) {* try *{* double fats = Double.*parseDouble(*keyword*)*;  
  
 return this.menuItems  
 .stream*()* .filter*(*item -> item.getFat*()* == fats*)* .collect*(*Collectors.*toList())*;  
 *}* catch *(*NumberFormatException ex*) {* return null;  
 *}  
}  
  
/\*\*  
 \* Metoda de filtrare dupa numarul de sodiu  
 \*  
 \** ***@param*** *keyword numarul de sodiu dupa care se va face filtrarea  
 \** ***@return*** *lista de produse corespunzatoare  
 \*/*private List*<*MenuItem*>* filterBySodium*(*String keyword*) {* try *{* double sodium = Double.*parseDouble(*keyword*)*;  
  
 return this.menuItems  
 .stream*()* .filter*(*item -> item.getSodium*()* == sodium*)* .collect*(*Collectors.*toList())*;  
 *}* catch *(*NumberFormatException ex*) {* return null;  
 *}  
}  
  
/\*\*  
 \* Metoda de filtrare dupa pret  
 \*  
 \** ***@param*** *keyword pretul dupa care se va face filtrarea  
 \** ***@return*** *lista de produse corespunzatoare  
 \*/*private List*<*MenuItem*>* filterByPrice*(*String keyword*) {* try *{* double price = Double.*parseDouble(*keyword*)*;  
  
 return this.menuItems  
 .stream*()* .filter*(*item -> item.getPrice*()* == price*)* .collect*(*Collectors.*toList())*;  
 *}* catch *(*NumberFormatException ex*) {* return null;  
 *}  
}*

Apoi avem metodele de generare a raporturilor folosind stream-uri:

*/\*\*  
 \* Metoda care ofera un raport al comenzilor in functie de un interval orar  
 \*  
 \** ***@param*** *start ora de inceput  
 \** ***@param*** *end ora de sfarsit  
 \*/*private void timeReport*(*String start, String end*) {* try *{* int startHour = Integer.*parseInt(*start*)*;  
 int endHour = Integer.*parseInt(*end*)*;  
  
 List*<*Order*>* result = this.orders  
 .keySet*()* .stream*()* .filter*(*order -> *(*order.getOrderDate*()*.getHour*()* >= startHour && order.getOrderDate*()*.getHour*()* <= endHour*))* .collect*(*Collectors.*toList())*;  
  
 String content = "";  
 for *(*Order o : result*)* content += o.toString*()* + "\n";  
  
 MyFileWriter.*writeToFile(*content, "report.txt"*)*;  
 *}* catch *(*NumberFormatException ex*) {* return;  
 *}  
}  
  
/\*\*  
 \* Metoda care verifica de cate ori a fost comandat un produs  
 \*  
 \** ***@param*** *param1 numele produsului pentru verificare  
 \*/*private void numberOrderedReport*(*String param1*) {* try *{* int number = Integer.*parseInt(*param1*)*;  
  
 Map*<*MenuItem, Integer*>* orderMap = new HashMap*<>()*;  
  
 for *(*Order o : this.orders.keySet*()) {* for *(*MenuItem i : this.orders.get*(*o*)) {* if *(*orderMap.containsKey*(*i*))* orderMap.put*(*i, orderMap.get*(*i*)* + 1*)*;  
  
 else  
 orderMap.put*(*i, 1*)*;  
 *}  
 }* String content = "Items:\n";  
 for *(*MenuItem i : orderMap.keySet*()) {* if *(*orderMap.get*(*i*)* > number*)* content += i.toString*()* + "\n";  
 *}* MyFileWriter.*writeToFile(*content, "report.txt"*)*;  
 *}* catch *(*NumberFormatException ex*) {* return;  
 *}  
}  
  
/\*\*  
 \* Metoda care scrie in fisier cati clienti au avut o coamnda cu pret mai mare decat o valoare data ca parametru  
 \*  
 \** ***@param*** *param1 numarul de comenzi  
 \** ***@param*** *param2 valoarea  
 \*/*private void clientsReport*(*String param1, String param2*) {* try *{* int numberOfOrders = Integer.*parseInt(*param1*)*;  
 int valueAmount = Integer.*parseInt(*param2*)*;  
  
 List*<*Order*>* orders = this.orders  
 .keySet*()* .stream*()* .filter*(*order -> order.getPrice*()* > valueAmount*)* .collect*(*Collectors.*toList())*;  
  
 Map*<*Integer, Integer*>* mapClientsOrders = new HashMap*<>()*;  
 for *(*Order o : orders*) {* if *(*mapClientsOrders.containsKey*(*o.getClientID*()))* mapClientsOrders.put*(*o.getClientID*()*, mapClientsOrders.get*(*o.getClientID*())* + 1*)*;  
  
 else  
 mapClientsOrders.put*(*o.getClientID*()*, 1*)*;  
 *}* String content = "IDs:\n";  
 for *(*Integer id : mapClientsOrders.keySet*())* if *(*mapClientsOrders.get*(*id*)* > numberOfOrders*)* content += String.*valueOf(*id*)* + "\n";  
  
  
 MyFileWriter.*writeToFile(*content, "report.txt"*)*;  
 *}* catch *(*NumberFormatException ex*) {* return;  
 *}  
}  
  
/\*\*  
 \* Metoda care scrie in fisier cate produse au fost comandate intr-o zi  
 \*  
 \** ***@param*** *param1 ziua data ca parametru  
 \*/*private void dayReport*(*String param1*) {* try *{* int day = Integer.*parseInt(*param1*)*;  
  
 Map*<*MenuItem, Integer*>* orderMap = new HashMap*<>()*;  
  
 for *(*Order o : this.orders.keySet*()) {* if *(*o.getOrderDate*()*.getDayOfMonth*()* == day*) {* for *(*MenuItem i : this.orders.get*(*o*)) {* if *(*orderMap.containsKey*(*i*))* orderMap.put*(*i, orderMap.get*(*i*)* + 1*)*;  
  
 else  
 orderMap.put*(*i, 1*)*;  
 *}  
 }  
 }* String content = "Items and Times Ordered:\n";  
 for *(*MenuItem i : orderMap.keySet*()) {* content += i.toString*()* + "\nOrdered: " + orderMap.get*(*i*)* + " times\n";  
 *}* MyFileWriter.*writeToFile(*content, "report.txt"*)*;  
 *}* catch *(*NumberFormatException ex*) {* return;  
 *}  
}*

**4.** **Interfața IdeliveryService –** conține operațiile pe care le pot face un administrator și client.

**5.** **Clasa MenuItem –** este o clasă abstractă care are ca atribute private caracteristicile unui produs din meniu, 2 constrcutori, metodele de get() și set(), metoda abstractă computePrice(), precum și metoda equals().

*/\*\*  
 \* Metoda de verificare a titlului  
 \*  
 \** ***@param*** *obj obiectul dupa al carui titlu verificam  
 \** ***@return*** *true sau false in functie de caz  
 \*/*@Override  
public boolean equals*(*Object obj*) {* if *(*!*(*obj instanceof MenuItem*))* return false;  
  
 else *{* MenuItem item = *(*MenuItem*)* obj;  
 return item.title.equals*(*this.title*)*;  
 *}  
}*

**6.** **Interfața Observable –** conține operațiile pe care le poate face un observer.

**7. Clasa Order -** are ca atribute private caractersiticile unei comenzi, un constructor, metodele de get() și set(), equals() și cea de hashCode().

**8. Clasa User -** are ca atribute private caractersiticile unui utilizator, un constructor, precum și metodele de get() și set().

**9. Clasa UserService –** conține implementarea logicii din spatele operațiilor de pe interfața pentru autentificare.

*/\*\*  
 \* Constructor pentru clasa UserService()  
 \*/*public UserService*() {* User u1 = new User*(*1, "admin", "admin", "administrator"*)*;  
 User u2 = new User*(*2, "ion", "ion", "client"*)*;  
 User u3 = new User*(*3, "angajat", "angajat", "employee"*)*;  
 User u4 = new User*(*4, "alin", "alin", "client"*)*;  
  
 this.users = new HashMap*<>()*;  
  
 users.put*(*u1.getUsername*()*, u1*)*;  
 users.put*(*u2.getUsername*()*, u2*)*;  
 users.put*(*u3.getUsername*()*, u3*)*;  
 users.put*(*u4.getUsername*()*, u4*)*;  
*}*

*/\*\*  
 \* Metoda de adaugare a unui user  
 \*  
 \** ***@param*** *user user-ul care trebuie adaugat  
 \*/*public void addUser*(*User user*) {* this.users.put*(*user.getUsername*()*, user*)*;  
*}  
  
/\*\*  
 \* Metoda de verificare a credentialelor  
 \*  
 \** ***@param*** *username username  
 \** ***@param*** *password parola  
 \** ***@return*** *user-ul  
 \** ***@throws*** *Exception exceptie  
 \*/*public User checkCredentials*(*String username, String password*)* throws Exception *{* if *(*!this.users.containsKey*(*username*))* throw new Exception*(*"Nu exista un utilizator cu acest username!"*)*;  
  
 else *{* User user = this.users.get*(*username*)*;  
  
 if *(*!user.getPassword*()*.equals*(*password*))* throw new Exception*(*"Parola gresita!"*)*;  
  
 return user;  
 *}  
}*

**10. Clasa MyFileWriter** – conține o metodă care scrie în fișier datele generate de rapoarte.

*/\*\*  
 \* Metoda in care se va scrie in fisier fiecare comanda adaugata  
 \*  
 \** ***@param*** *content ce trebuie scris in fisier  
 \** ***@param*** *file fisierul in care trebuie scris  
 \*/*public static void writeToFile*(*String content, String file*) {* try *{* String filePath = "D:\\Facultate\\UTCN\\An II\\Semestrul 2\\Tehnici de programare\\Laboratoare\\PT2022\_30228\_Gavrilescu\_AndreeaLavinia\_4\\src\\main\\resources\\orders\\" + file;  
 BufferedWriter bw = new BufferedWriter*(*new FileWriter*(*filePath*))*;  
 bw.write*(*content*)*;  
 bw.close*()*;  
 *}* catch *(*IOException ex*) {* ex.printStackTrace*()*;  
 *}  
}*

**11.** **Clasa ClientView** – impementează elementele interfeței grafice pentru Client

– conține atributele necesare în casete text, a etichetelor corepunzătoare acestora, precum și a butoanelor de adăugare, modificare, ștergere, căutare produse și plasare comenzi.

În constructor vom poziționa toate elementele interfeței, pe lângă metodele de get() și set() pentru castele text, avem metode de adăugare a unor listeners pentru a verifica dacă unul dintre butoane este apăsat, caz în care se trimite un apel către controller pentru a afișa rezulatul corepunzător.

*/\*\*  
 \* Metoda de adaugare a unui listener pe butonul de cautare  
 \*  
 \** ***@param*** *actionListener actionListener  
 \*/*public void addCautareListener*(*ActionListener actionListener*) {* this.searchButton.addActionListener*(*actionListener*)*;  
*}*

**12.** **Clasa AdministratorView** – impementează elementele interfeței grafice pentru Administrator

– conține atributele necesare în casete text, a etichetelor corepunzătoare acestora, precum și a butoanelor de adăugare, modificare, ștergere, creare meniu și generare rapoarte.

În constructor vom poziționa toate elementele interfeței, pe lângă metodele de get() și set() pentru castele text, avem metode de adăugare a unor listeners pentru a verifica dacă unul dintre butoane este apăsat, caz în care se trimite un apel către controller pentru a afișa rezulatul corepunzător.

**13.** **Clasa EmployeeView** – impementează elementele interfeței grafice pentru Employee

– conține un textArea în care apare o notificare când un client plasează o comandă.

În constructor vom poziționa toate elementele interfeței, pe lângă metodele de get() și set() pentru castele text, avem metode de adăugare a unor listeners pentru a verifica dacă unul dintre butoane este apăsat, caz în care se trimite un apel către controller pentru a afișa rezulatul corepunzător.

**14.** **Clasa RegisterView** – impementează elementele interfeței grafice pentru fereastra de autentificare

– conține atributele necesare în casete text, a etichetelor corepunzătoare acestora, precum și a butoanelor de log in și register.

În constructor vom poziționa toate elementele interfeței, pe lângă metodele de get() și set() pentru castele text, avem metode de adăugare a unor listeners pentru a verifica dacă unul dintre butoane este apăsat, caz în care se trimite un apel către controller pentru a afișa rezulatul corepunzător.

**15.** **Clasa ClientController** – primește un apel când este apăsat un buton de pe interfață, iar după efectuarea operației corespunzătoare butonului, , returnează rezultatul în tabel sau afișează un mesaj corespunzător (restul butoanelor).

Constructorul primește ca parametru view-ul și deliveryService-ul, iar acesta are rolul de a lega butoanele de metoda care apelează operația corepunzătoare.

*/\*\*  
 \* Constructor pentru clasa ClientController  
 \*  
 \** ***@param*** *view view-ul  
 \** ***@param*** *deliveryService service-ul  
 \*/*public ClientController*(*ClientView view, DeliveryService deliveryService*) {* this.view = view;  
 this.deliveryService = deliveryService;  
  
 this.view.addCreareListener*(*new CreareListener*())*;  
  
 this.view.addCautareListener*(*new CautareListener*())*;  
  
 this.loadTableData*()*;  
*}*

Apoi avem metodele care implementeaza interfața ActionListener prin care se verifică inputul */\*\*  
 \* Clasa pentru implementarea logicii din spatele butonului de creare  
 \*/*class CreareListener implements ActionListener *{* @Override  
 public void actionPerformed*(*ActionEvent e*) {* try *{* List*<*Integer*>* selectedRows = new ArrayList*<>()*;  
  
 for *(*int row : view.getProductsTable*()*.getSelectedRows*()) {* selectedRows.add*(*row*)*;  
 *}* int i = 0, added = 0;  
  
 List*<*MenuItem*>* items = new ArrayList*<>()*;  
 for *(*MenuItem item : deliveryService.getMenuItems*()) {* if *(*selectedRows.contains*(*i*)) {* items.add*(*item*)*;  
 added++;  
  
 if *(*added == selectedRows.size*())* break;  
 *}* i++;  
 *}* if *(*view.getOrderTextField*()*.getText*()*.equals*(*""*))* throw new Exception*(*"Nu s-a introdus un ID pentru comanda!"*)*;  
  
 int id = Integer.*parseInt(*view.getOrderTextField*()*.getText*())*;  
  
 User client = view.getLoggedInClient*()*;  
 deliveryService.createOrder*(*id, client.getId*()*, LocalDateTime.*now()*, items*)*;  
  
 String fileName = "order" + id + ".txt";  
 String content = "ID: " + id + "\nClient ID: " + client.getId*()* + "\nDate Order: " + LocalDateTime.*now()* + "\nItems: " + items;  
 MyFileWriter.*writeToFile(*content, fileName*)*;  
  
 view.getProductsTable*()*.clearSelection*()*;  
  
 JOptionPane.*showMessageDialog(*null, "Comanda a fost creata cu succes!"*)*;  
 *}* catch *(*Exception ex*) {* JOptionPane.*showMessageDialog(*null, ex.getMessage*())*;  
 *}  
 }*

Pentru tabel:

*/\*\*  
 \* Metoda de punere a datelor in tabel  
 \*  
 \** ***@param*** *items produse din meniu  
 \*/*private void loadTableData*(*List*<*MenuItem*>* items*) {* DefaultTableModel tableModel = new DefaultTableModel*()*;  
  
 tableModel.addColumn*(*"Title"*)*;  
 tableModel.addColumn*(*"Rating"*)*;  
 tableModel.addColumn*(*"Calories"*)*;  
 tableModel.addColumn*(*"Protein"*)*;  
 tableModel.addColumn*(*"Fat"*)*;  
 tableModel.addColumn*(*"Sodium"*)*;  
 tableModel.addColumn*(*"Price"*)*;  
  
 for *(*MenuItem item : items*) {* List*<*String*>* values = new ArrayList*<>()*;  
  
 values.add*(*item.getTitle*())*;  
 values.add*(*String.*valueOf(*item.getRating*()))*;  
 values.add*(*String.*valueOf(*item.getCalories*()))*;  
 values.add*(*String.*valueOf(*item.getProtein*()))*;  
 values.add*(*String.*valueOf(*item.getFat*()))*;  
 values.add*(*String.*valueOf(*item.getSodium*()))*;  
 values.add*(*String.*valueOf(*item.getPrice*()))*;  
  
 tableModel.addRow*(*values.toArray*())*;  
 *}* this.view.getProductsTable*()*.setModel*(*tableModel*)*;  
*}*

**16.** **Clasa AdministratorController** – primește un apel când este apăsat un buton de pe interfață, iar după efectuarea operației corespunzătoare butonului, returnează rezultatul în tabel sau afișează un mesaj corespunzător (restul butoanelor).

Constructorul primește ca parametru view-ul și deliveryService-ul, iar acesta are rolul de a lega butoanele de metoda care apelează operația corepunzătoare.

**17.** **Clasa AdministratorController** – primește un apel când este plasată o comandă de către un client.

Constructorul primește ca parametru view-ul și deliveryService-ul, iar acesta are rolul de a lega butoanele de metoda care apelează operația corepunzătoare.

**18.** **Clasa RegisterController** – primește un apel când este apăsat un buton de pe interfață, iar după efectuarea operației corespunzătoare butonului, deschide fereastra corespunzătoare.

Constructorul primește ca parametru view-ul și deliveryService-ul, iar acesta are rolul de a lega butoanele de metoda care apelează operația corepunzătoare.

**19. Clasa Main** – conține atributele necesare controllere-lor astfel încât să realizeze legătura între interfață și logică.

private Map*<*String, String*>* administrators = new HashMap*<>()*;  
  
private Map*<*String, String*>* clients = new HashMap*<>()*;  
  
private Map*<*String, String*>* employees = new HashMap*<>()*;  
  
*/\*\*  
 \* Metoda in care se va executa programul  
 \*  
 \** ***@param*** *args argumentele din linia de comanda  
 \** ***@throws*** *IOException exceptie  
 \*/*public static void main*(*String*[]* args*)* throws IOException *{* DeliveryService deliveryService = new DeliveryService*()*;  
 deliveryService.importProducts*()*;  
 deliveryService.generateOrders*()*;  
  
 UserService userService = new UserService*()*;  
  
 RegisterView registerView = new RegisterView*()*;  
  
 AdministratorView administratorView = new AdministratorView*()*;  
 EmployeeView employeeView = new EmployeeView*()*;  
 ClientView clientView = new ClientView*()*;  
  
 AdministratorController administratorController = new AdministratorController*(*administratorView, deliveryService*)*;  
 ClientController clientController = new ClientController*(*clientView, deliveryService*)*;  
 EmployeeController employeeController = new EmployeeController*(*employeeView, deliveryService*)*;  
 RegisterController registerController = new RegisterController*(*registerView, administratorView, clientView, employeeView, userService*)*;  
*}*

**20. Interfața Observer –** conține metoda de update() care permite primirea notificărilor de către angajați.

**21. Clasa MyComparator –** clasa care conține metoda de comparare a două produse.

*/\*\*  
 \* Metoda de comparare a doua produse  
 \*  
 \** ***@param*** *m1 parametrul1  
 \** ***@param*** *m2 parametrul2  
 \** ***@return*** *rezultatul comparatiei  
 \*/*@Override  
public int compare*(*MenuItem m1, MenuItem m2*) {* return *(*m1.getTitle*())*.compareTo*(*m2.getTitle*())*;  
*}*

Rezultate

Un administrator poate genera diferite tipuri de rapoarte.

Exemplu:

report.txt

Items and Times Ordered:  
BaseProduct{title='"""Blanketed"" Eggplant "', rating=3.75, calories=1386.0, protein=105.0, fat=98.0, sodium=578.0, price=88.0}  
Ordered: 1 times  
BaseProduct{title='"""Bloody Mary"" Tomato Toast with Celery and Horseradish "', rating=5.0, calories=189.0, protein=2.0, fat=16.0, sodium=517.0, price=16.0}  
Ordered: 1 times  
BaseProduct{title='"""Brown on Blonde"" Blondies "', rating=0.0, calories=321.0, protein=5.0, fat=18.0, sodium=120.0, price=96.0}  
Ordered: 1 times

De asemenea, se generează fișiere text de fiecare dată când este plasată o comandă.

Exemplu:

Order1.txt

ID: 1  
Client ID: 2  
Date Order: 2022-04-29T20:39:37.627497500  
Items: [BaseProduct{title='"""Blanketed"" Eggplant "', rating=3.75, calories=1386.0, protein=105.0, fat=98.0, sodium=578.0, price=88.0}, BaseProduct{title='"""Bloody Mary"" Tomato Toast with Celery and Horseradish "', rating=5.0, calories=189.0, protein=2.0, fat=16.0, sodium=517.0, price=16.0}, BaseProduct{title='"""Brown on Blonde"" Blondies "', rating=0.0, calories=321.0, protein=5.0, fat=18.0, sodium=120.0, price=96.0}, BaseProduct{title='"""California Roll"" Salad "', rating=4.375, calories=369.0, protein=9.0, fat=13.0, sodium=953.0, price=67.0}]

Concluzii

Acest proiect mi-a consolidat cunoștințele teoretice și practice de Java pe care le-am învățat în semestrul 1 prin volumul de lucru mult mai intens și avansat pe care l-am depus în scopul finalizării acestei teme.

Ca dezvoltare ulterioară, se mai poate îmbunătăți aspectul interfeței ca să arate mai interesant, se mai poate adăuga tot felul de butoane de back pentru a naviga mai ușor între interfețe, precum și butoane care să asigure o conexiune mai buna între acestea, putând fi utlizată eventual pentru comenzi online.

Bibliografie

* <https://dsrl.eu/courses/pt/materials/A4_Support_Presentation.pdf>
* <https://dsrl.eu/courses/pt/materials/PT2021-2022_Assignment_4.pdf>
* <http://javahungry.blogspot.com/2013/08/hashing-how-hash-map-works-in-java-or.html>
* <http://javahungry.blogspot.com/2013/08/how-sets-are-implemented-internally-in.html>
* <https://www.geeksforgeeks.org/serialization-in-java/>