Documentație tema 4

Food delivery management system

Sigoiu George

Grupa 30224

Cuprins

[1. Obiectivul temei 4](#_Toc72612589)

[a) Obiectivul principal 4](#_Toc72612590)

[b) Obiective secundare 4](#_Toc72612591)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 5](#_Toc72612592)

[a) Cerințe funcționale 5](#_Toc72612593)

[b) Descriere cazuri de utilizare 5](#_Toc72612594)

[ Use-case: adunare, scădere, înmulțire, împărțire 6](#_Toc72612595)

[ Use-case: derivare, integrare 7](#_Toc72612596)

[3. Proiectare 8](#_Toc72612597)

[a) Diagrama UML 8](#_Toc72612598)

[b) Structuri de date 8](#_Toc72612599)

[c) Algoritmi 8](#_Toc72612600)

[d) Interfața utilizator 8](#_Toc72612601)

[4. Implementare 11](#_Toc72612602)

[A. Package business 11](#_Toc72612603)

[a) MenuItem 11](#_Toc72612604)

[b) BaseProduct 11](#_Toc72612605)

[c) CompositeProduct 11](#_Toc72612606)

[d) DeliveryServiceProcessing 11](#_Toc72612607)

[e) DeliveryService 11](#_Toc72612608)

[f) Order 11](#_Toc72612609)

[B. Package model 11](#_Toc72612610)

[a) Account 11](#_Toc72612611)

[b) Client 11](#_Toc72612612)

[c) Administrator 11](#_Toc72612613)

[d) Employee 11](#_Toc72612614)

[e) RegisteredUsers 11](#_Toc72612615)

[C. Package presentation 11](#_Toc72612616)

[a) AdministratorView 11](#_Toc72612617)

[b) ClientView 11](#_Toc72612618)

[c) CreateProductView 11](#_Toc72612619)

[d) EmployeeView 11](#_Toc72612620)

[e) LogIn 11](#_Toc72612621)

[f) ReportWindow 11](#_Toc72612622)

[g) SignUp 11](#_Toc72612623)

[D. Package dao 11](#_Toc72612624)

[a) FileWriter1 11](#_Toc72612625)

[b) ReadExcel 11](#_Toc72612626)

[c) Serializator 11](#_Toc72612627)

[5. Rezultate 12](#_Toc72612628)

[6. Concluzii 13](#_Toc72612629)

[7. Bibliografie 14](#_Toc72612630)

# Obiectivul temei

## Obiectivul principal

Obiectivul principal al aplicației este acela de a crea și implementa un sistem de gestiune a comenzilor pentru produse alimentare pentru o companie de caritate. Clienții pot comanda produse dintr-un meniu al companiei, dar trebuie să aibă un cont pe platforma companiei, iar dacă nu au, își pot crea unul. Platforma are 3 tipuri de utilizatori: administrator, angajat, client.

## Obiective secundare

|  |  |  |
| --- | --- | --- |
| Obiectiv secundar | Descriere | Capitol |
| Determinarea operațiilor | Descrierea cerințelor funcționale | [2](#_Cerințe_funcționale) |
| Dezvoltarea de use-case-uri și scenarii | Într-un sistem software un ‘use-case’ este o listă de acțiuni sau pași de eveniment care definesc în mod obișnuit interacțiunile dintre un rol ( cunoscut ca un actor în limbajul unificat de modelare ( UML ) ) și un sistem in atingerea unui obiectiv.  În cazul nostru, rolul este de a utiliza aplicația. | [2](#_Descriere_cazuri_de) |
| Alegerea structurilor de date | Structurile de date folosite pentru a duce la capăt obiectivul principal | [3](#_Structuri_de_date) |
| Interfața utilizator | Se descrie modul cum utilizatorul poate folosi aplicația | [3](#_Interfața_utilizator) |
| Împărțirea pe clase | Crearea mai multor clase pentru a ușura lucrul, clase pentru stocarea datelor, clase pentru efectuarea operațiilor, clase pentru interfața grafică | [4](#_Implementare) |
| Testare | Vor fi descrise câteva scenarii de testare a operaților pe polinoame, folosind ca instrument un Junit Test. | [5](#_Rezultate) |

# Analiza problemei, modelare, scenarii, cazuri de utilizare

## Cerințe funcționale

În aplicație se pot conecta 3 tipuri de utilizatori: administrator, angajat și client.

Administratorul poate să:

* importe setul inițial de produse cu care se v-a popula meniul companiei de caritate (dintr-un fișier excel)
* gestioneze produsele din meniu, adică să adauge produse noi, să le editeze sau să le șteargă pe cele existente și să creeze produse noi alcătuite din mai multe produse (exemplu pentru un nou produs care poate fi numit „Meniu zilnic” și să fie alcătuit dintr-o supă, o friptură, garnitură și un desert)
* genereze rapoarte cu privire la anumite aspecte referitoare la comenzile efectuate de către clienți, dar și referitoare la clienți. Rapoartele trebuie să conțină:
  + comenzile dintr-un interval orar, neluând în considerare ziua
  + produsele comandate de mai mult de un număr specificat de ori
  + clienții care au comandat de mai mult decât un număr specificat de ori, iar fiecare comandă să aibă o valoare totală mai mare decât o valoare specificată
  + produsele comandate într-o anumită zi, împreună cu numărul echivalent de câte ori au fost comandate fiecare în parte

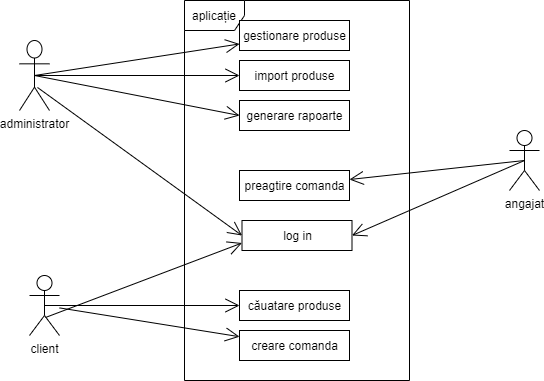
Clientul poate să:

* se înregistreze în aplicație și să-și creeze un cont pentru a se conecta în aplicație
* vizualizeze produsele din meniu
* caute produsele bazându-se pe un criteriu de căutare cum ar fi: titlu (cuvânt sau cuvinte care să se regăsească în denumirea unui produs), rating, numărul de calorii / proteine / grăsimi sau preț.

Angajatului îi este specificat de fiecare dată când o nouă comandă este creată de către un client astfel încât acesta să poată pregăti produsele.

## Descriere cazuri de utilizare

Fiecare dintre cele 3 tipuri de utilizatori trebuie să se logheze întâi pentru a avea acces la aplicație.



# Proiectare

## Diagrama UML

Diagram

Description automatically generated

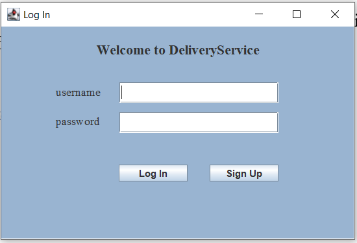
## Structuri de date

Pentru stocarea produselor în memorie am folosit o structură BaseProduct (String title, double rating, int calories, sodium, fat, protein, price), pentru stocarea produselor noi definite de administrator am folosit ca structură CompositeProduct (conține un titlu și o listă de produse), iar pentru comenzi am folosit un Map<Order, List<MenuItem>>, unde Order conține datele comenzii efectuate (orderID, clientID și data când a fost înregistrată comanda) iar lista conține toate produsele comandate de către client. MenuItem este o clasă abstractă extinsă de către BaseProduct și CompositeProduct.

## Algoritmi

Algoritmii folosiți au fost cei din expresiile lambda pentru sortare si filtrarea datelor, precum si cei de transformare a unui stream în List sau Set.

## Interfața utilizator

Interfața pentru logare conține 2 câmpuri pentru scrierea datelor de utilizator și două butoane, unul pentru verificarea datelor introduse și intrarea în aplicație dacă datele sunt corecte, iar altul pentru crearea unui cont nou de client.

Graphical user interface

Description automatically generatedInterfața pentru înregistrare conține 4 câmpuri pentru introducerea datelor persoanele pentru fiecare client, necesare pentru livrarea comenzii și 2 câmpuri pentru crearea numelui de utilizator și a unei parole asociate numelui de utilizator anterior ales.

Graphical user interface, application, Word

Description automatically generated

Interfața administratorului conține o fereastră în care se pot vizualiza toate produsele (BaseProduct sau CompositeProduct în funcție de care dintre cele 2 butoane de la baza ferestrei este selectat), un buton import care efectuează înlocuirea produselor deja existente în baza de date cu alte noi dintr-un fișier ales la apăsarea butonului, un buton pentru generarea de rapoarte (în funcție de care checkbox-uri sunt bifate) și 2 butoane pentru adăugarea unui produs simplu în baza de date, respectiv pentru crearea unui produs compus. Butonul de edit permite administratorului să modifice produsele după ce a fost apăsat.

Graphical user interface, application, table

Description automatically generated

Aceasta este interfața pentru crearea unui produs compus în care produsele pot fi căutate după mai multe criterii. Produsele selectate din fereastra din stânga sunt adăugate în fereastra din dreapta (care reprezintă o previzualizare a unei posibile comenzi). Produsele din fereastra din dreapta pot fi șterse pe rând sau toate odată, dacă se dorește eliminarea unui produs sau mai multor produse din coșul creat.

Graphical user interface, text, application

Description automatically generated Fereastra angajatului este una simplă care conține doar un panel unde vor apărea comenzile nou comandate de către oricare client.

Graphical user interface, application

Description automatically generated

Interfața clientului conține, asemenea administratorului, o fereastră care îi permite clientul vizualizarea tuturor produselor existente în meniu, o fereastră care îi permite observarea produselor din posibila comandă și 2 butoane care permit ștergerea elementelor selectate din comandă, respectiv plasarea comenzii.

# Implementare

## Package business

### MenuItem

Este o clasă abstractă care implementează Serializable

abstract public int computePrice();

### BaseProduct

Clasă care extinde MenuItem și implementează Serializable. Conține ca atribute următoarele: String title, double rating, int calories, protein, fat, sodium, price.

Ca și metode, clasa BaseProduct conține doar getter-e și setter-e, suprascrierea metodei toString() și metoda public int computePrice() care returnează valoarea produsului.

### CompositeProduct

Este o clasă care extinde MenuItem și implementează Serializable care simbolizează un produs compus din mai multe produse simple. Conține ca atribute un String title și o listă de produse simple ce reprezintă produsele din care este alcătuit.

Ca și metode, clasa BaseProduct conține doar getter-e și setter-e, suprascrierea metodei toString() și metoda public int computePrice() care returnează valoarea totala produsului.

### DeliveryServiceProcessing

Este o interfață ce conține metodele care trebuie implementate pentru ca funcțiile administratorului și ale clientului să funcționeze.

public void importProducts(String pathFile);

* importă produse dintr-un fișier excel care se află la locația din pathFile.

public void addNewProduct(MenuItem p);

* adaugă un nou produs simplu în meniu

public void deleteProduct(MenuItem p);

* șterge un produs din meniu

public void editProduct(List<BaseProduct> newProducts);

* primește o listă de produse modificate și le înlocuiește pe cele existente

public void generateTimeIntervalOrderReport(int startHour,int endHour);

* generează un raport cu privire la comenzile efectuate într-un anumit interval orar

public void generateOrderReport(int times,int minValue);

- generează un raport privind numărul de comenzi comandate care au valoarea peste minValue  
public void generateOrderReport(int times);

- generează un raport privind produsele comandate de un anumit număr de ori  
public void generateOrderReport(String dt);

- generează un raport privind comenzile dintr-o anumită zi

public void createOrder(Client client, List<MenuItem>products);

* creeaza o camandă creată de un client

### DeliveryService

Este o clasă care extinde Observable și implementează DeliveryServiceProcessing, clasă care reprezintă meniu companiei de caritate. În această clasă se stochează produsele și comenzile efectuate de către clienți și o variabilă care contorizează toate comenzile efectuate în aplicație.

Pe lângă metodele interfeței pe care o implementează și pe lângă getter-e și setter-e, clasa conține și:

public static void addProduct(int i, MenuItem p)

* adaugă un produs în meniu

public static void setProducts(int i, List<MenuItem> produse)

* schimbă lista de produse

### Order

Clasa Order reprezintă datele unei comenzi, conține id-ul clientul care efectuează comanda, data la care se efectuează comanda și numărul comenzii.

Ca și metode conține getter-e și setter-e.

## Package model

### Account

Conține ca atribute username și password de tipul String care reprezintă contul de conectare la aplicație pentru fiecare utilizator.

### Client

Conține un clientID (int), un account (Account) și atribute care repzintă datele personale ale clientului de tipul String (name, email, phone, address).

### Administrator

Conține doar un account (de tipul Account) care conține datele pentru logare în aplicație.

### Employee

Conține doar un account (de tipul Account) care conține datele pentru logare în aplicație.

### RegisteredUsers

Reprezintă structura folosită pentru stocarea datelor de logare pentru fiecare tip de utilizator. Conține 3 liste, una pentru fiecare tip de utlizator.

Metodele clasei sunt câte un getter pentru fiecare listă și metode care adaugă un utilizator în lista respectivă.

## Package presentation

### AdministratorView

### ClientView

### CreateProductView

### EmployeeView

### LogIn

### ReportWindow

### SignUp

## Package dao

### FileWriter1

Clasa reprezintă un mecanism de scriere în fișiere.

public static void generateBill(Client client, Order order, List<MenuItem> products)

* metoda care generează un bon fiscal pentru fiecare comandă efectuată

public static void generateRaport(String filename, String mesaj)

* metodă ca strie într-un fisier dat ca argument un mesaj dat ca argument

### ReadExcel

Conține o singură metodă care citește datele dintr-un fișier excel și returnează o listă de produse citite din fișierul respectiv.

### Serializator

Este o clasă care se ocupă cu serializarea și deserializarea obiectelor din package-ul model.

# Rezultate

Pentru a testa aplicația s-a realizat verificarea fiecărei funcționalități a aplicației, neavând date inserate în aplicație pentru a rezolva posibilele erori/excepții ce ar putea apărea. Pe urmă s-au introdus date în aplicație pe rând, adică prima data s-au importat datele dintr-un fișier excel (folosind opțiunea de import din interfața administrator) și s-au testat din nou toate funcționalitățile aplicației, pentru fiecare tip de utilizator.

# Concluzii

Proiectul realizat deși este unul complex, are totuși o mulțime de noi dezvoltări anterioare. Ca dezvoltare pentru interfața administratorului se poate introduce opțiunea pentru a vizualiza toate comenzile efectuate în aplicație, precum și realizarea unei căutări ale comenzilor în funcție de numele clientului, de valoarea totala a comenzii sau în funcție de un anumit produs pe care să-l conțină o comandă, de asemenea și realizarea afișării comenzilor dintr-o anumită perioadă de timp ar fi utilă. Pentru interfața client o posibilă dezvoltare anterioară ar fi posibilitatea ca fiecare client sa-și vadă toate comenzile comandate, iar pentru interfața angajat, posibilitatea de a marca comenzile livrate pentru a-i fi mai ușor în gestionarea comenzilor către clienți.

Realizarea acestei teme m-a ajutat să înțeleg cât de ușor este să folosim expresiile lambda în lucrul nostru și să învăț de asemenea să folosesc expresiile lambda. Totodată am învățat și să folosesc serializarea obiectelor, un lucru foarte util, metodă ce poate înlocui lucrul cu baze de date (pentru aplicații mici care nu necesită stocarea a multor tipuri date).

# Bibliografie

**Lambda expressions and stream processing**

<https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>

<https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html>

<https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8-streams.html>

<https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/>

<https://howtodoinjava.com/java8/java-stream-distinct-examples/>

**Java serialization**

<http://www.tutorialspoint.com/java/java_serialization.htm> <https://www.baeldung.com/java-serialization>

<https://www.geeksforgeeks.org/serialization-in-java/>

<https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html>

**Java HashMap**

<http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html>

**Java assert**

<http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html>

<http://javarevisited.blogspot.ro/2012/01/what-is-assertion-in-java-java.html>

<http://stackoverflow.com/questions/11415160/how-to-enable-the-java-keyword-assert-in-eclipse-program-wise>

<https://intellij-support.jetbrains.com/hc/en-us/community/posts/207014815-How-to-enable-assert>

**Adding custom tags to javadoc**

<https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#tag>