O imagine care conține emblemă, simbol, siglă, Marcă comercială

Descriere generată automat

Gestiunea unui magazin online de produse de patiserie

Autor: Mierliță Ioana-Claudia

Profesor coordonator: Ciurea Cristian-Eugen

București, 2024

Cuprins

[**Scopul proiectului** 4](#_Toc181954199)

[**Structura proiectului** 4](#_Toc181954200)

[**Pachetul *com.products*** 5](#_Toc181954201)

[**Directorul Cart** 5](#_Toc181954202)

[**Clasa Cart** 5](#_Toc181954203)

[**Clasa CartView** 6](#_Toc181954204)

[**Directorul File** 6](#_Toc181954205)

[**Clasa FileManager** 7](#_Toc181954206)

[**Directorul Order** 7](#_Toc181954207)

[**Clasa Comandă** 8](#_Toc181954208)

[**Clasa OrderView** 8](#_Toc181954209)

[**Clasa StatusComanda** 9](#_Toc181954210)

[**Directorul Product** 9](#_Toc181954211)

[**Clasele BaseProductCard, UserProductCard, AdminProductCard** 9](#_Toc181954212)

[**Clasa ProductView** 9](#_Toc181954213)

[**Clasa Produs** 10](#_Toc181954214)

[**Clasa ProdusValidationException** 10](#_Toc181954215)

[**Clasa StarRatingComponent** 10](#_Toc181954216)

[**Directorul Report** 11](#_Toc181954217)

[**Clasa Raport** 11](#_Toc181954218)

[**Clasa ReportManagementView** 12](#_Toc181954219)

[**Clasa UserView** 13](#_Toc181954220)

[**Clasa AdminView** 13](#_Toc181954221)

[**Clasa Main** 13](#_Toc181954222)

[**Tabel de figuri** 15](#_Toc181954223)

[**Bibliografie** 16](#_Toc181954224)

# **Scopul proiectului**

Aplicația mea prezintă un sistem de gestiune al unui magazin online, realizat în Java și folosind JavaFX pentru interfața grafică. Proiectul pune accent pe operațiile CRUD(create, read, update, delete), oferind utilizatorilor posibilitatea de adăugare, modificare, șterege și vizualizare a produselor din cadrul magazinului online. Aplicația oferă posibilitatea utilizării ei atât de utilizatorii obișnuiți, cât și de administrator, fiecare dintre ei având meniuri distincte și având acces la operațiuni specifice fiecăruia în parte.

# **Structura proiectului**

Codul sursă al aplicației este inclus în **directorul *src*** și este organizat în pachete pentru a oferi modularitate aplicației și pentru a *respecta Single Responsability Principle.*

*O imagine care conține text, captură de ecran, Font

Descriere generată automat*

**Figură 1 - Structura directorului src**

Fișierele de intrare sunt organizate în directorul data, iar acestea sunt Produse.json – fișierul JSON care conține lista de produse disponibile pentru vânzare în cadrul magazinului online și Orders.txt, lista comenzilor trimise de către utilizatori.

O imagine care conține text, Font, captură de ecran, proiectare

Descriere generată automat

**Figură 2 - Directorul data**

# **Pachetul *com.products***

**Conține directoarele principale corespunzătoare aplicației:**

* [**Cart**](file:///C:\Users\mierl\OneDrive\Desktop\Proiect%20PPOO\JavaDoc\your.module.name\com\products\Cart\Cart.html) - Directorul Cart reprezintă funcționalitățile coșului de cumpărături al utilizatorului.
* [**FileManager**](file:///C:\Users\mierl\OneDrive\Desktop\Proiect%20PPOO\JavaDoc\your.module.name\com\products\File\FileManager.html)- Directorul FileManager implementează un singleton pentru gestionarea operațiilor cu fișiere.
* [**Comanda**](file:///C:\Users\mierl\OneDrive\Desktop\Proiect%20PPOO\JavaDoc\your.module.name\com\products\Order\Comanda.html)- Directorul care gestionează detaliile și procesarea unor comenzi în sistemul de vânzări.
* [**Prod****us**](file:///C:\Users\mierl\OneDrive\Desktop\Proiect%20PPOO\JavaDoc\your.module.name\com\products\Product\Produs.html) - Directorul Produs gestionează cardurile de produse și operațiile asupra acestora.
* [**Raport**](file:///C:\Users\mierl\OneDrive\Desktop\Proiect%20PPOO\JavaDoc\your.module.name\com\products\Report\Raport.html) - Directorul responsabil pentru generarea diferitelor tipuri de rapoarte privind vânzările, stocurile și performanța produselor.

## **Directorul Cart**

* este format din clasele **Cart** și **CartView**.

**Clasa Cart**

* **Clasa Cart utilizeaza un *HashMap* pentru a gestiona produsele comandate și cantitățile din acestea.**

De asemenea, este utilizat contructorul implicit pentru a inițializa lista de produse. Clasa conține o metodă pentru adăugarea produsului împreună cu cantitatea aferentă în map, întorcând o excepție definită de utilizator - **ProdusValidationException** în cazul în care cantitatea disponibilă din produsul respectiv este mai mică decât cantitatea solicitată de utilizator.

Utilizatorul are totodată posibilitatea de a modifica în coș canitatea produselor comandate, iar dacă noua cantitate introdusă este nulă produsul va fi eliminat din coș cu ajutorul metodei ***removeProduct()*** , iar în caz contrar, modificarea se va realiza cu succes.

**Clasa CartView**

* **Clasa CartView construiește interfața coșului de cumpărături primind referințe către interfața aferentă comenzilor - OrderView și coșul de cumpărături Cart. Clasa conține un constructor cu doi parametri – Cart și Orderview pe care îi inițializează.**

Metoda ***openCartView()***deschide fereastra aferentă coșului de cumpărături și inițializează interfața grafică, iar în cazul în care coșul este gol afișează o alertă de informare. Interfața coșului de cumpărături utilizează un container vertical din JavaFX, respectiv *Vbox*, lista produselor urmând de asemenea același layout. Interfața mai conține un buton pentru trimiterea comenzii care apelează metoda ***sendOrder()*** și închide fereastra coșului după trimiterea cu succes a comenzii.

Fiecare produs din coș este construit cu ajutorul metodei ***createProductBox()*** pe baza unui container orizonal *Hbox*, care conține numele produsului, cantitatea și butoane pentru actualizare și stergere produs care apelează metodele definite în clasa **Cart**.

## **Directorul File**

* este format din clasele FileManager și FileOperationException

### **Clasa FileManager**

* **Clasa FileManager implementează un singleton pentru gestionarea operațiilor cu fișiere. Aceasta se ocupă cu încărcarea și salvarea produselor în format JSON și a comenzilor în format text.**

Clasa are ca atribut static constant căile fișierelor si instanța singleton statică și privată. Clasa implementează totodată constructorul default privat pentru a nu permite instanțierea de mai multe ori a clasei, în cadrul căruia se realizează încărcarea produselor din fișierul *JSON* într-o listă cu ajutorul metodei ***loadFromFile().*** Produsele sunt adăugate în listă progresiv, utilizând *JSONObject* și *JSONArray* pentru a prelua fiecare obiect din fișier și pentru a construi un produs nou pe baza atributelor pe care acesta le stochează.

Clasa conține de asemenea o metodă statică pentru returnarea instanței, care verifică dacă aceasta este nulă și in caz afirmatic creează un nou **FileManager**, altfel returnează instanța existentă, unică.

Pentru salvarea produselor în fișierul *JSON* la închiderea aplicației, în urma modificărilor aduse asupra listei, clasa implementează metoda ***saveToFile()*** care aruncă o excepție definită de utilizator **FileOperationException** și utilizează tot *JSONArray* și *JSONObject* pentru a itera prin lista de produse și pentru a o salva ulterior.

Pentru încărcarea comenzilor din fișierul text, este implementată metoda ***loadOrders()*** care utilizează clasele *FileReader* și *BufferedReader* pentru citirea fișierului. Totodată această clasă utilizează metoda ***parseOrderData()*** pentru a parsa o comandă.

## **Directorul Order**

* este format din clasele Comanda, OrderView și StatusComanda

**Clasa Comandă**

* **Clasa Comandă este clasa care gestionează detaliile și procesarea unei comenzi.**

Conține lista de produse incluse în comandă, valoarea totală a comenzii, data și ora la care a fost plasată și statusul curent al ei. Are un constructor ce are ca paramtru lista de produse și calculează totalul comenzii și atribuie data comenzii care espre preluată cu ajutorul funcției ***now()*** din clasa *LocalDateTime* în momentul în care utilizatorul trimite comanda. Statusul comenzii este ales din enum-ul **StatusComandă** și este setat pe IN\_PROCESARE by default. Pe lângă constructoriu, getteri și setteri mai conține o metodă ***getDetails()*** care utilizează un *StringBuilder* pentru a stoca detaliile unei comenzi.

### **Clasa OrderView**

* **Clasa OrderView gestionează interfața de vizualizare a comenzilor.**

Conține un constructor ce primește ca parametru un *Cart* și îl inițializează, împreună cu lista de comenzi. Metoda ***openOrdersView()*** inițializează interfața grafică și creează un *TabelView* al comenzilor cu ajutorul metodei ***createOrdersTableView().*** Pentru gestionarea statusului am folosit un *ComboBox* din *JavaFX* care permite administratorului să modifice stadiul comenzii prin intermediul metodei ***updateItem()*** care setează statusul utilizând **StatusComanda**. De asemenea, clasa conține și metoda menționată anterior*,* ***sendOrder(),*** folosită pentru a trimite comanda din coșul de cumpărături. Cu ajutorul unui atribut boolean se verifică dacă comanda este trimisă deja și în caz afirmativ apare o eroare, altfel comanda este adăugată în listă, produsului îi este incrementat numărul de cumpărări, coșul este golit și valoarea aatributului orderSent este setată pe true, deoarece a fost trimisă.

### **Clasa StatusComanda**

* **Clasa StatusComanda reprezintă enumerația pentru stările posibile ale unei comenzi( în procesare, finalizată, expediată).**

## **Directorul Product**

* este format din clasele BaseProductCard, UserProductCard, AdminProductCard, ProductView, Produs, ProdusValidationException și StarRatingComponent

### **Clasele BaseProductCard, UserProductCard, AdminProductCard**

* **Clasele BaseProductCard, UserProductCard, AdminProductCard sunt responsabile pentru crearea cardurilor de produs utilizate atât în interfața utilizatorului, cât și în interfața de administrator.**

Clasele **UserProductCard** și **AdminProductCard** sunt derivate din **BaseProductCard**. UserProductCard onține o metodă ***createUserProductCard()*** care returnează un *Pane* din *JavaFX* și reprezintă un card specific unui produs. Acest card este construit cu ajutorul unui *Vbox* și conține denumirea produsului, sistemul de rating, descrierea sa, prețul și un câmp de introducere a cantității produsului în cazul în care acesta e achiziționat. Totodată cardul conține un buton ce apelează metoda ***addToCart()*** din **CartView** pentru a adăuga produsul în coș. În clasa AdminProductCard regăsim metoda ***createAdminProductCard()*** este metoda specifică cardurilor administratorilor și conține butoane de modificare sau ștergere produse.

### **Clasa ProductView**

* **Clasa ProductView este responsabilă pentru interfața utilizatorilor, oferindu-le acestora psoibilitatea de a vizualiza produsele.**

Aceasta conține o referință la clasa **AdminView** și lista de produse, pe care le inițializează într-un constructor cu parametri. Clasa conține metoda ***saveProduct()*** care apelează metoda ***updateProduct(),*** dacă produsul nu există în listă, iar dacă acesta există apelează ***addNeewProduct()*** pentru a adăuga un nou produs în listă. De asemenea, clasa mai conține și metoda de ștergere a unui produs, la finalul fiecărei metode de actualizare a listei de produse apelându-se metoda ***refreshProductGrid()*** care face să se reflecte în interfață noile modificări. Pentru a gestiona validările este utilizată metoda ***showAlert()*** din clasa **Alert**, care primește ca parametri titlul, mesajul de afișat și tipul alertei(de infromare, de confirmare sau de eroare).

### **Clasa Produs**

* **Clasa Produs reprezintă un produs din cadrul magazinului online de patiserie și conține atribute precum preț, descriere, cantitateDisponibilă, nume, dataAdăugării produsului, dataExpirare, rating, nrRatinguri și numarCumpărăi, toate aceste atribute fiind inițializate într-un constructor cu parametri.**

Clasa conține de asemenea metode accesor și metoda ***toString()***pentru parsarea detaliilor unor produs, dar și metodele de incrementare a numărului de cumpărări și ratinguri.

### **Clasa ProdusValidationException**

* **Clasa ProdusValidationException este o excepție custom utilizată pentru validarea datelor produselor.**

### **Clasa StarRatingComponent**

* **StarRatingComponent este componenta de evaluare a ratingului unui produs și e practic un container orizontal format din cinci stele reprezentate printr-un vector unidimensional de obiecte de tip SVGPath ce este format dintr-un șir de caractere ce descriu coordonatele și liniile necesare pentru a desena o stea.**

Clasa conține de asemenea un buton care confirmă ratingul oferit de utilizator și actualizează ratingului produsului fiind actualizat permanent. De asemenea, clasa conține o metodă ***updateStars()*** pentru colorarea stelelor – cu auriu dacă sunt selectate și cu gri în caz contrar.

## **Directorul Report**

* este format din clasele Raport și RaportManagementView

### **Clasa Raport**

* **Clasa Raport este responsabilă pentru generarea diferitelor tipuri de rapoarte privind vânzările, stocurile și performanța produselor.**

Clasa conține lista de produse și comenzi și denumirea raportului și un constructor ce le inițializează. Clasa prezintă metode pentru generarea rapoartelor precum :

|  |  |  |
| --- | --- | --- |
| **Metodă** | **Tip retur** | **Descriere** |
| genereazaRaportFrecventaComenzi(String filename) | void | Generează un raport cu frecvența comenzilor pe zile calendaristice și ore. |
| genereazaRaportGeneralStoc(String filename) | void | Generează un raport cu referire la stocurile de produse. |
| genereazaRaportProduseAdaugateRecent(String filename) | void | Generează un raport cu produsele adăugate în ultimele 30 de zile. |
| genereazaRaportProduseAproapeExpirate(String filename) | void | Generează un raport cu produsele care vor expira în următoarele 30 de zile. |
| genereazaRaportProdusePeStoc(String filename) | void | Generează un raport cu starea curentă a stocului pentru toate produsele. |
| genereazaRaportRatinguriProduseMatrice(String filename) | void | Generează un raport de ratinguri pentru toate produsele utilizând o matrice. |
| genereazaRaportTendinteVanzari(String filename) | void | Generează un raport cu tendințele vânzărilor pe zile. |
| genereazaRaportTotalVanzari(String filename) | void | Generează un raport cu totalul vânzărilor. |

**Figură 3 - Tabel rapoarte**

De asemenea, fiecare metodă de scriere în fișier este apelată la închiderea aplicației.

### **Clasa ReportManagementView**

* **Clasa ReportManagementView este responsabilă pentru interfața utilizatorului și primește o referință la un obiect de tip Raport.**

Conține un constructor și metoda ***openReportManagementView()*** pentru a construi view-ul cu ajutorului unui *Vbox* format din mai multe *Hbox*, câtre unul corespunzător fiecărui raport care conține denumirea raportului și butoane de genereare și vizualizare raport. Metoda ***generateReport()*** conține un *switch* ce tratează index-ul rapoartelor, în funcție de acesta apelând metodele corespunzătoare raportului respectiv din clasa **Raport**. Metoda de vizualizare a unui raport – ***viewReport()*** deschide fișierul text cu raportul, utilizând metoda ***getDesktop()*** din clasa *Desktop*.

Directorul com.products conține și alte clase specifice, cum ar fi clasa **Alert** menționată anterior necesară creării alertelor, **ControllerManipulation** – care gestionează metode de creare și stilizare a butoanelor, imaginilor, etc. De asemenea, conține și clasele **UserView** și **AdminView** care gestionează meniurile fiecăruia dintre utilizatori.

## **Clasa UserView**

* **Clasa UserView conține lista de produse, comenzi și referințe la Cart, CartView și OrderView.**

Are de asemenea în componență metoda ce deschide vizualizarea meniului cu butoane specifice ce conduc spre view-urile corespunzătoare – spre exemplu, butonul *viewProductsButton* conduce către pagina de vizualizare produse construită cu ajutorul clasei **ProductView**.

## **Clasa AdminView**

* **Clasa AdminView conține lista de produse și referințe la OrderView și ReportManagementView.**

Metoda ***openAdminMenu()*** deschide meniul adminisratorului și conține butoane specifice fiecărei acțiuni, la fel ca meniul utilizatorului obișnuit. Aceste metode sunt apelate asupra butonelor prin metoda *setOnAction*.

## **Clasa Main**

* **Clasa Main este clasa principală a aplicației de gestiune a produselor de patiserie. Extinde clasa Application pentru a oferi interfața grafică principală a aplicației și gestionează inițializarea aplicației și oferă punctul de intrare pentru diferite moduri de utilizare.**

Aceasta primește ca atribute lista statică de produse și lista statică de comenzi și obiectul pentru manipularea controalelor UI **ControllersManipulation**. De asemenea, conține metoda ***start(),*** metoda principală de inițializare a interfeței grafice, care utilizează instanța unică **FileManager** prin metoda ***getInstance()*** pentru a inițializa listele de produse și comenzi din fișiere. Clasa utilizează două butoane pentru a putea ghida utilizatorii în alegerea interfeței corespunzătoare – de administrator sau utilizator obișnuit. O altă metodă disponibilă în cadrul clasei este ***setOnCloseRequest()***, un *handler* care configurează închiderea aplicației și salvează toate informațiile despre comenzi și produse în fișiere. De asemenea, în cadrul acestei metode sunt generate și rapoartele.

# **Tabel de figuri**

[**Figură 1 - Structura directorului src** 4](#_Toc181953657)

[**Figură 2 - Directorul data** 5](#_Toc181953658)

[**Figură 3 - Tabel rapoarte** 12](#_Toc181953659)

# **Bibliografie**

1. **Vos, J., Gao, W., Chin, S., Weaver, J. (2021). "Pro JavaFX 9: A Definitive Guide to Building Desktop, Mobile, and Embedded Java Clients". Apress.**
2. **Sharan, K. (2022). "Java: A Beginner's Guide", 9th Edition. McGraw-Hill Education. "Online Shopping System Java Project" -** [**https://projectsgeek.com/java-project/online-shopping-system**](https://projectsgeek.com/java-project/online-shopping-system)
3. **"Mastering JavaFX Controls" de Hendrik Ebbers (2019)**
4. **"Inventory Management System" -** [**https://www.geeksforgeeks.org/inventory-management-system/**](https://www.geeksforgeeks.org/inventory-management-system/)