**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE**

**DEPARTAMENTUL CALCULATOARE**

**TEMA 4**

**FOOD DELIVERY MANAGEMENT SYSTEM**

**Documentație**

**Nume: Litu Saviana**

**Grupa: 30221**

**An academic 2020 – 2021**

**Cuprins**

[**Obiectivul temei** 3](#_Toc73385687)

[**Analiza problemei, modelare, scenarii, cazuri de utilizare** 4](#_Toc73385688)

[**Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfața utilizator)** 5](#_Toc73385689)

[**Implementare** 9](#_Toc73385690)

[**Rezultate** 15](#_Toc73385691)

[**Concluzii** 19](#_Toc73385692)

[**Bibliografie** 19](#_Toc73385693)

# **Obiectivul temei**

Obiectivul acestei teme este de a propune și implementa o aplicație Java ce are scopul de a realiza un sistem de management al unei firme de catering. Se va folosi un fisier csv din care se preia meniul. Aplicația va avea trei tipuri de utilizator, și anume administator, angajat și client. Clientul va putea să comande produse din meniul firmei, iar administratorul va putea să realizeze anumite operații asupra produselor din meniu și să genereze anumite report-uri. În plus, aplicația va fi structurată în pachete, conform unei arhitecturi stratificate.

Pentru a realiza acest obiectiv principal, definim următoarele obiective secundare:

|  |  |  |
| --- | --- | --- |
| **Obiectiv** | **Descriere** | **Capitol** |
| Analiza problemei | Se analizează enunțul problemei și se dezvoltă un design inițial | **2** |
| Alegerea structurilor de date potrivite | Se aleg structuri de date care să faciliteze salvarea comenzilor și a datelor din meniu | **3** |
| Proiectare | Se vor crea clase și pachete în concordanță cu design-ul dorit | **3** |
| Implementare | Se va realiza preluarea datelor din fișierul inițial folosind stream-uri și expresii lambda, dar și operațiile de serializare și deserializare a datelor | **4** |
| Realizarea interfeței grafice | Se va realiza interfața utilizator utilizând JavaFX | **4** |
| Testarea | Se vor efectua teste pentru diferite input-uri | **5** |

# **Analiza problemei, modelare, scenarii, cazuri de utilizare**

Un stream reprezintă o secvență de elemente și suportă diferite tipuri de operații care să evalueze elementele respective. Operațiile cu stream-uri sunt intermediare sau terminale. Operațiile intermediare returnează un stream, deci putem înlănțui mai multe operații intermediare fără a folosi punct și virgulă. Operațiile terminale sunt ori de tipul void, ori returnează un rezultat care nu este tot un stream. Tipuri de operații intermediare sunt filter, map și sorted, iar un exemplu de operație terminală este forEach. Cele mai multe operații cu stream-uri acceptă un fel de lambda expresii ca parametrii.

Expresiile Lambda profită de capacitățile de proces paralele ale mediilor multi-core, așa cum se vede cu sprijinul operațiunilor de conducte pe date în API-ul Stream. Sunt metode anonime (metode fără nume) utilizate pentru a implementa o metodă definită de o interfață funcțională. O interfață funcțională este o interfață care conține o singură metodă abstractă.

Expresiile Lambda introduc noul operator săgeată -> în Java. Împarte expresiile lambda în două părți. Partea din stânga specifică parametrii solicitați de expresie, care ar putea fi, de asemenea, goi dacă nu sunt necesari parametri. Partea dreaptă este corpul lambda care specifică acțiunile expresiei lambda.

Analizând enunțul temei propuse, observăm că trebuie să începem prin a citi datele din fișierul de tip csv care constituie meniul cu care vom lucra, folosind stream-uri și expresii lambda. Apoi, vom implementa funcțiile de serializare și deserializare pentru a putea salva meniul într-un fișier separat, pe care să îl putem citi și în care putem adăuga noi intrări. Apoi implementăm funcționalitățile pentru tipurile de utilizatori client și administrator. Administratorul va putea să importe datele din fișierul inițial, să adauge, să șteargă și să modifice produse din meniu, dar și să genereze report-uri în funcție de anumite criterii. Clientul se poate înregistra în aplicație, creându-și un cont, se poate loga, poate vedea meniul, poate căuta anumite produse din meniu în funcție de anumite criterii și să plaseze comenzi, în urma cărora va primi o chitanță.

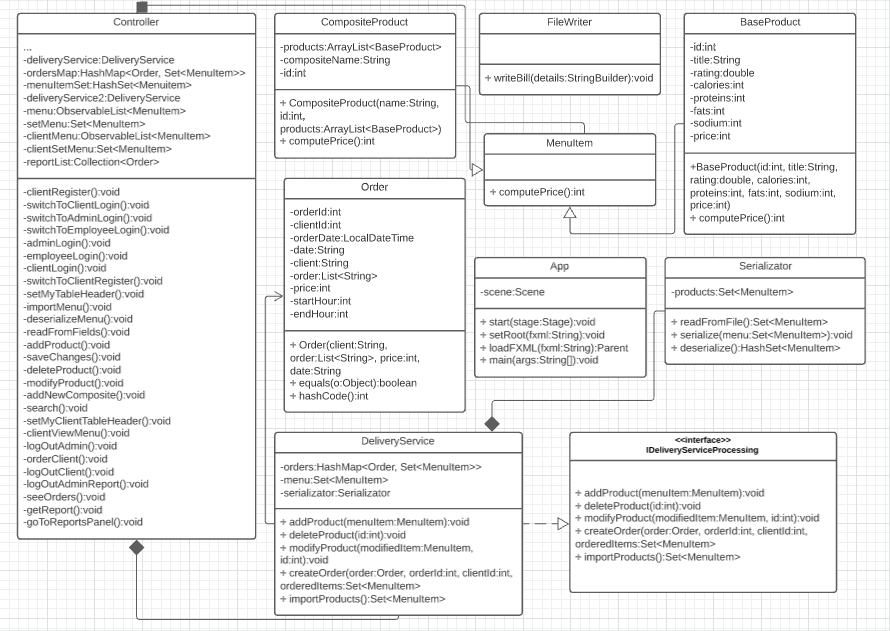
Trecând la partea de modelare, vom respecta diagrama de clase și pachete prestabilită în cerința proiectului. Așadar, vom avea pachetele: presentation (care va conține clasa Controller, care se ocupă de interfața grafică), businessLayer (care va conține clasele Order, MenuItem, BaseProduct, CompositeProduct, interfața IDeliveryServiceProcessing și clasa DeliveryService) și dataLayer (care va conține clasele Serializator și FileWriter).

În continuare, voi prezenta cazul de utilizare ideal:

1. Utilizatorul alege panel-ul în care vrea să lucreze: Client Register, Client Login, Admin Login sau Employee Login.
2. Utilizatorul își poate crea un cont dacă a ales să fie client sau se poate loga dacă are deja un cont de client sau este admin/angajat.
3. Dacă utilizatorul este client, atunci poate folosi toate funcționalitățile de care dispune clientul: poate vedea meniul, poate căuta produse, poate plasa o comandă și va primi o chitanță.
4. Dacă utilizatorul este administrator, poate folosi toate funcționalitățile de care dispune adminul: poate importa meniul din fișierul csv, poate sa adauge, să șteargă și să modifice produse din meniu și să genereze un report.
5. Atunci când dorește, utilizatorul are și opțiunea de log out, în urma căreia se poate loga pe un alt cont, poate crea un nou cont sau își poate schimba statutul (client/admin).

# **Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfața utilizator)**

* Diagrama UML



* Structuri de date

Structurile de date folosite sunt:

* Set
* HashSet
* HashMap
* ArrayList
* ObservableList
* Collection
* TextField, TextArea, TableView și TableColumn – specifice JavaFX, folosite pentru a declara text field-urile, respectiv tabelele și coloanele acestora
* Proiectarea claselor

*Clasa App*: clasă în care se află metoda main, deci din care se rulează aplicația.

*Clasa Order*: clasă în care se află atributele specifice unei comenzi – orderId, clientId, orderDate, date, client, order, price, startHour, endHour.

*Clasa MenuItem*: clasă abstractă care va fi extinsă de clasele BaseProduct și CompositeProduct prin implementarea metodei computePrice().

*Clasa BaseProduct*: clasă extinde MenuItem și în care se află atributele specifice unui produs din meniu, și anume id, title, rating, calories, proteins, fats, sodium, price. Acestea se citesc din fișierul inițial în ordinea menționată anterior. De asemenea, aici se implementează metoda computePrice().

*Clasa CompositeProduct*: clasă care extinde MenuItem și în care se salvează o listă de BaseProduct deoarece un produs compus conține mai multe produse de bază. De asemenea, implementează metoda computePrice().

*Interfața IDeliveryServiceProcessing*: interfață care va fi implementată de clasa DeliveryService și în care sunt definite metodele addProduct, deleteProduct, modifyProduct, createOrder și importProducts.

*Clasa DeliveryService*: clasă care implementează interfața IDeliveryServiceProcessing prin metodele menționate anterior. De asemenea, aici se declară un HashMap în care vom reține comenzile și un Set în care vom reține meniul. De asemenea, am declarat și un obiect al clasei Serializator pentru a putea face importarea produselor din fișierul inițial într-un alt fișier text.

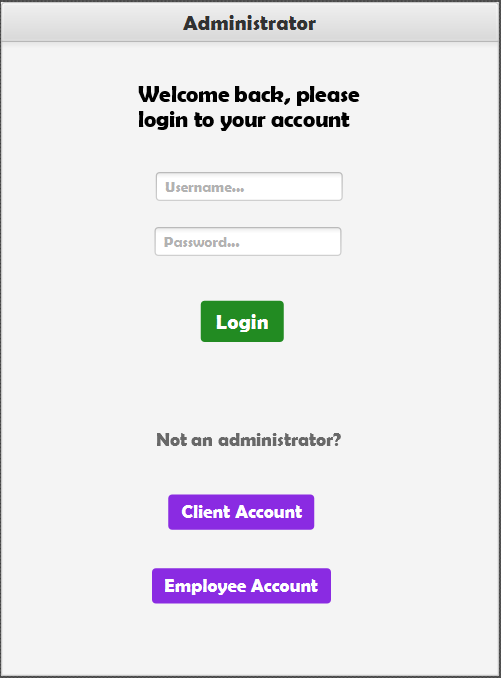
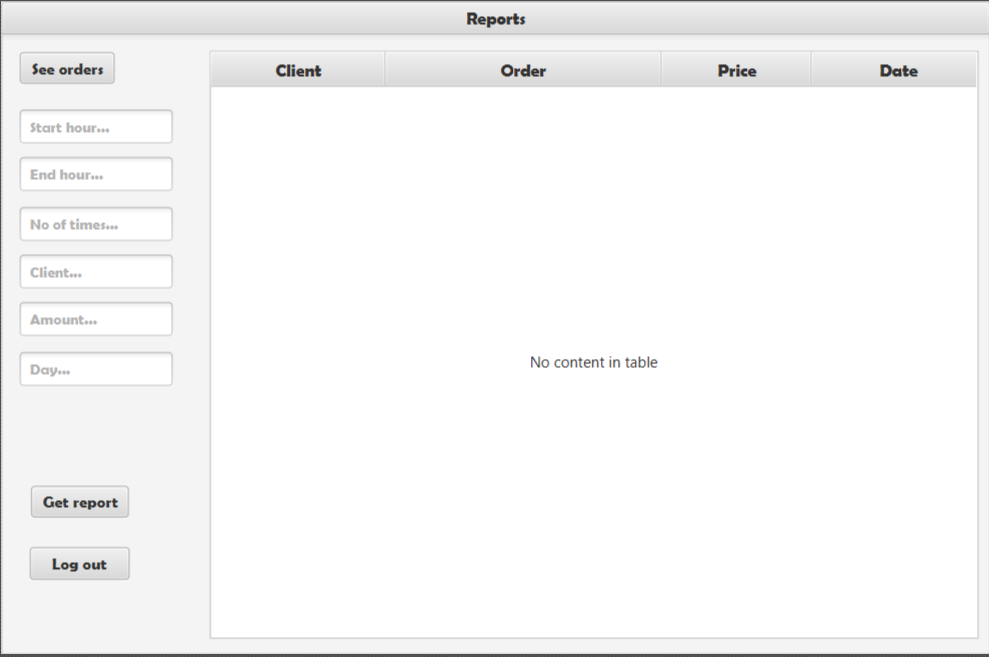
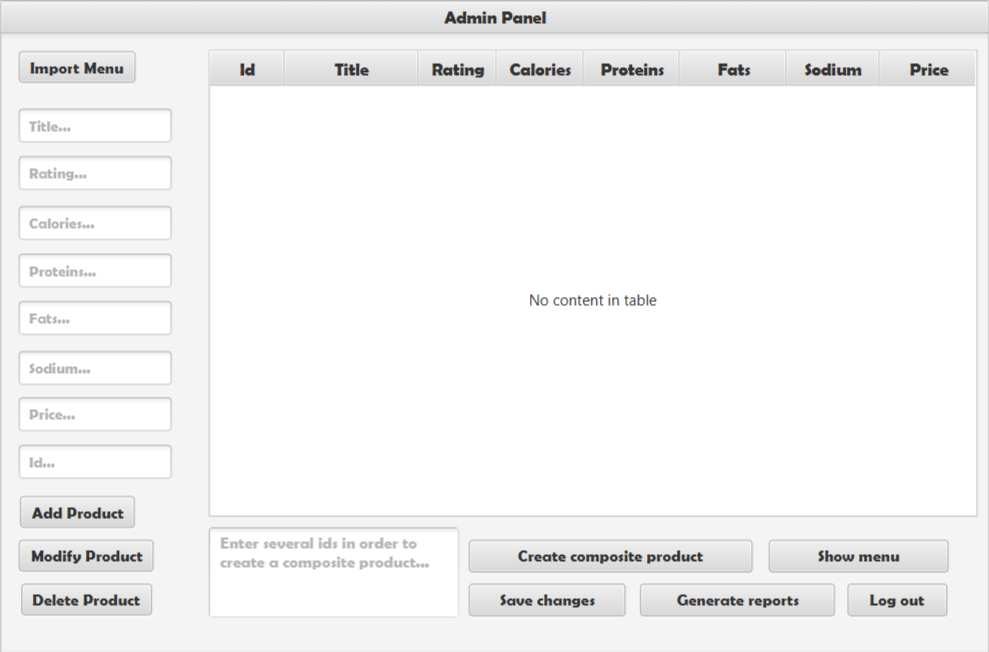
*Clasa FileWrite*: clasă în care implementez scrierea în fișier a chitanței pe care o primește clientul după ce plasează o comandă.

*Clasa Serializator*: clasă în care implementez citirea din fișierul inițial folosind stream-uri și lambda expressions, dar și metodele de serializare și deserializare a datelor.

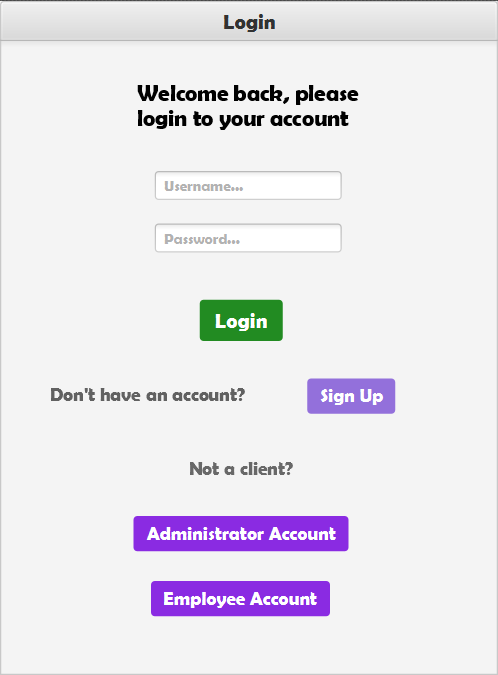
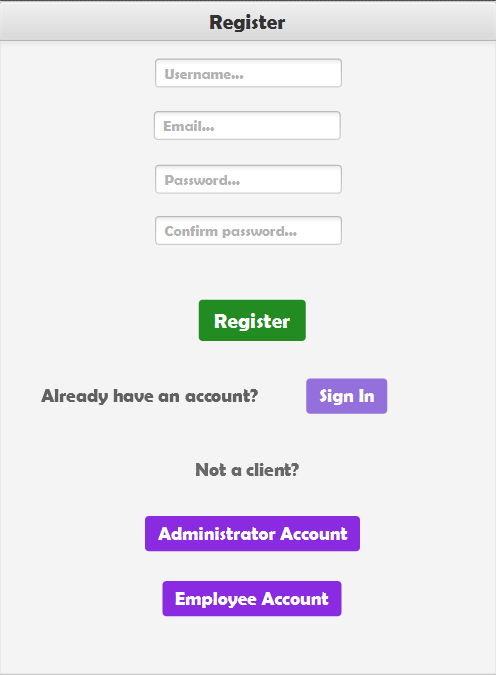
*Clasa Controller*: clasă în care se implementează funcționalitatea butoanelor și text field-urilor din interfața grafică. Atributele de tip TextField sunt folosite pentru ca utilizatorul să poată să își introducă username-ul și parola, pentru a căuta anumite detalii despre produse și a filtra meniul în funcție de acestea, pentru a introduce datele unui nou produs care urmează să fie adăugat în meniu etc. Tot aici implementez și tabelele de tipul TableView, cu coloane de tipul TabelColumn, tabele în care voi stoca datele citite și importate din fișierul inițial și pe care le voi afișa pentru fiecare tip de utilizator.

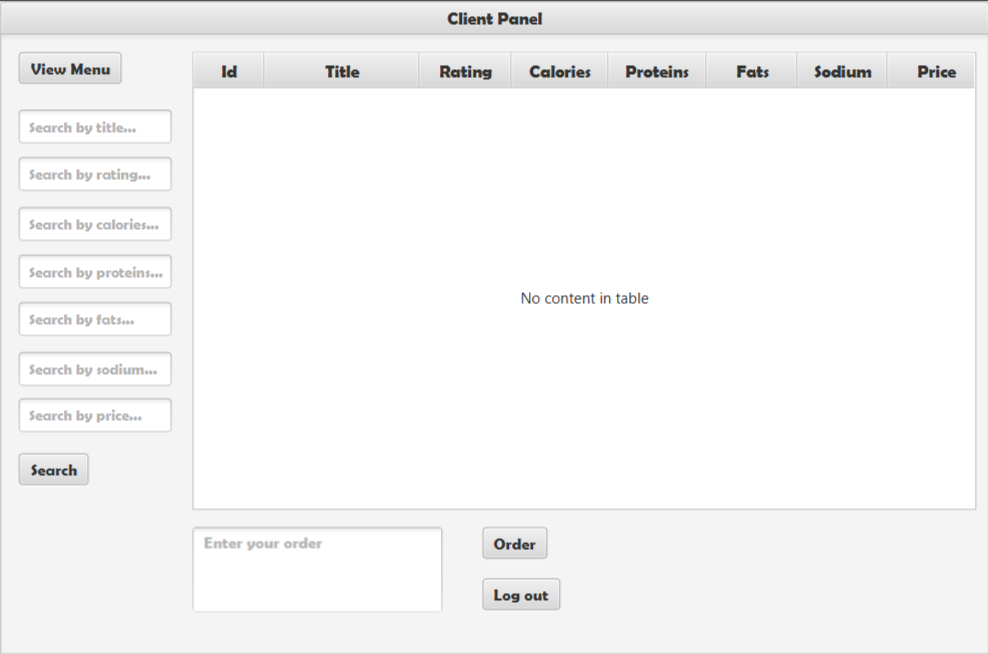
* Interfața utilizator

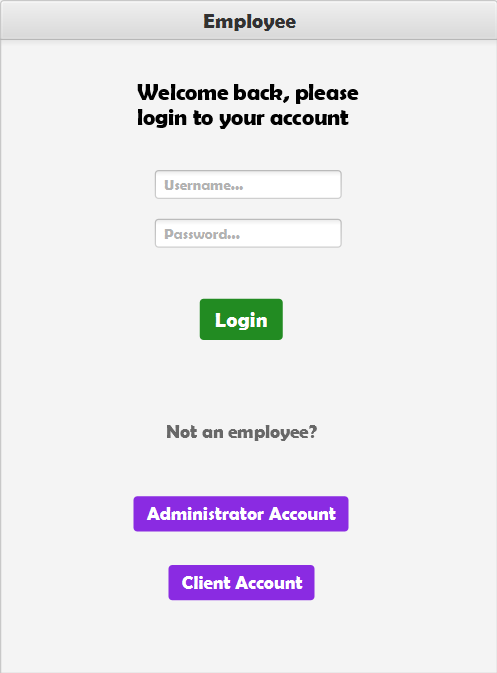
Pentru a realiza interfața grafică folosind JavaFX, am creat un proiect Java de tip Maven și am dezvoltat interfața incluzând dependințele specifice, apoi modelând aspectul în SceneBuilder. Am ales cadranul care să stea la bază, de tipul TitledPane, apoi am adăugat un tabel de tipul TableView și în jurul său TextField-uri, TextArea și butoane pentru schimbarea window-ului sau pentru efectuarea operațiilor.

Window-ul pentru logarea administratorului și window-ul în care administratorul genereaza report-urile.

Window-ul pentru panel-ul administratorului. Aici admin-ul poate să realizeze operații supra produselor din meniu, să le salveze și să genereze report-uri.

Window-urile pentru crearea unui nou cont și logarea în aplicație în calitate de client. Un client se poate înregistra și apoi loga sau se poate loga cu un cont deja existent.

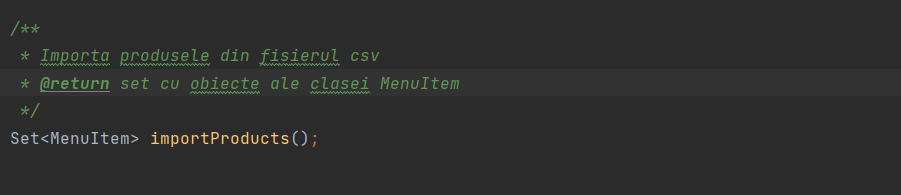
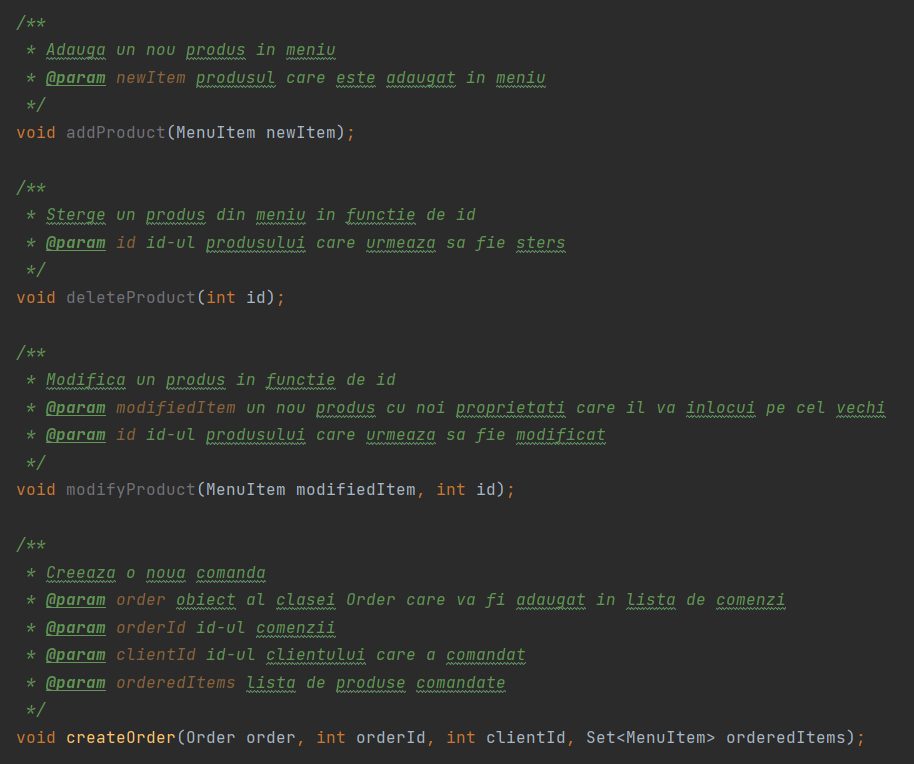
Window-ul pe care îl acceseaza clientul după logare. Aici clientul poate să vadă meniul, poate să caute unele produse din meniu în funcție de anumite criterii și poate să plaseze o comandă, scriind id-urile produselor pe care dorește să le comande.



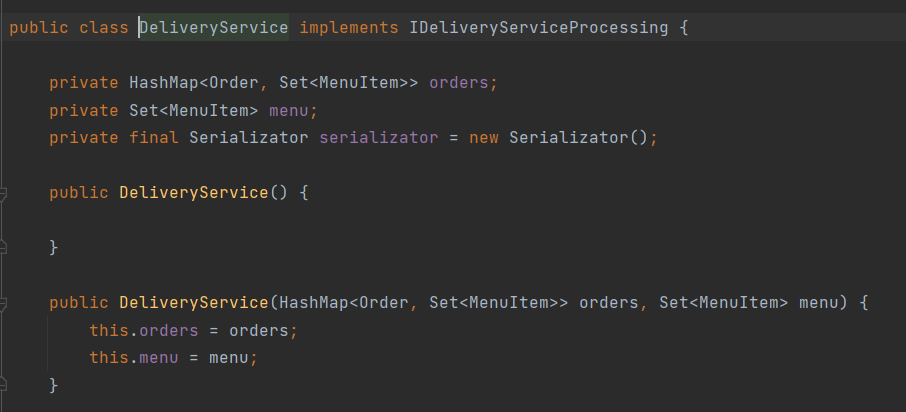
Window-ul pentru utilizatorul de tip angajat. Funcționalitățile acestui tip de utilizator nu sunt încă implementate, însă contul cu care cineva se poate loga ca angajat este: (username)angajat și (parola)angajat.

# **Implementare**

Încep prin a prezenta interfața IDeliveryServiceProcessing. Această interfață conține metodele care vor fi implementate în clasa DeliveryService care o implementează. Acestea sunt: addProduct, deleteProduct, modifyProduct, createOrder, importProducts. Toate metodele au fost documentate folosind JavaDoc.

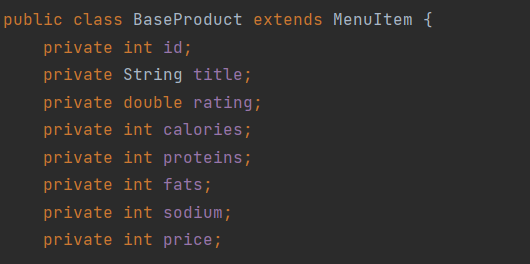


Clasa DeliveryService implementează interfața IDeliveryServiceProcessing prin metodele addProduct, deleteProduct, modifyProduct, createOrder, importProducts. Primele trei metode corespund funcționalităților de care dispune utilizatorul de tip administrator. Metoda createOrder corespunde utilizatorul de tip client. Ultima metodă corespunde administratorului, iar prin aceasta administratorul importă datele din fișierul csv inițial.

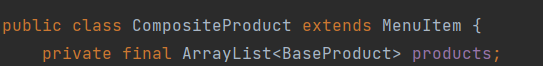


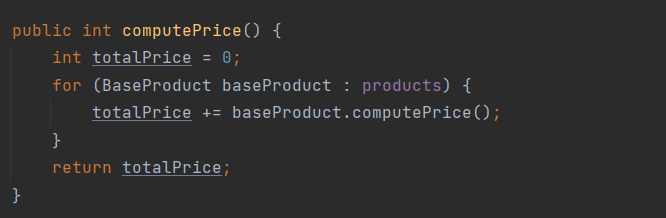
Clasa MenuItem este o clasă abstractă care va fi extinsă de clasele BaseProduct și CompositeProduct, prin metoda computePrice. Aceasta este folositoare pentru obiectele din clasa CompositeProduct, deoarece prețul total se calculează adunând prețurile tuturor produselor de bază care alcătuiesc produsul compus.

Clasa BaseProduct este o clasă care extinde clasa abstractă MenuItem prin metoda computePrice. Aceasta are următoarele field-uri: id, title, rating, calories, proteins, fats, sodium, price. Acestea sunt folosite în ordinea în care au fost citite din fișierul inițial și importate de către administrator.

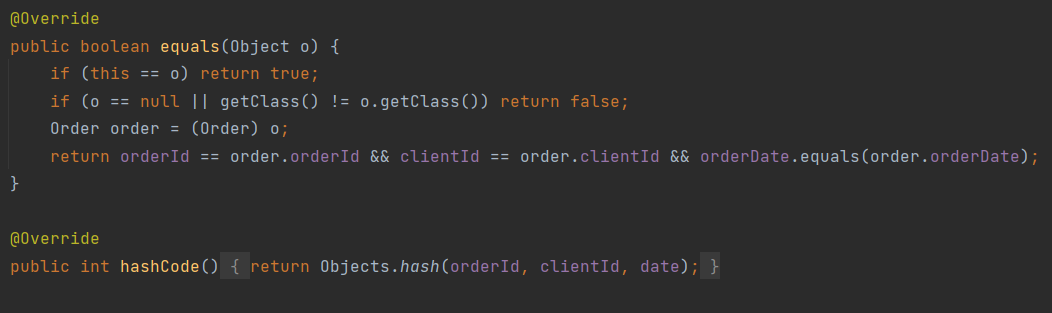


Clasa CompositeProduct este o clasă care extinde clasa abstractă MenuItem prin metoda computePrice. Aceasta este folosită pentru produsele compuse care sunt alcătuite din mai multe produse de bază. Metoda computePrice calculează prețul final adunând prețurile produselor de bază care alcătuiesc produsul compus.





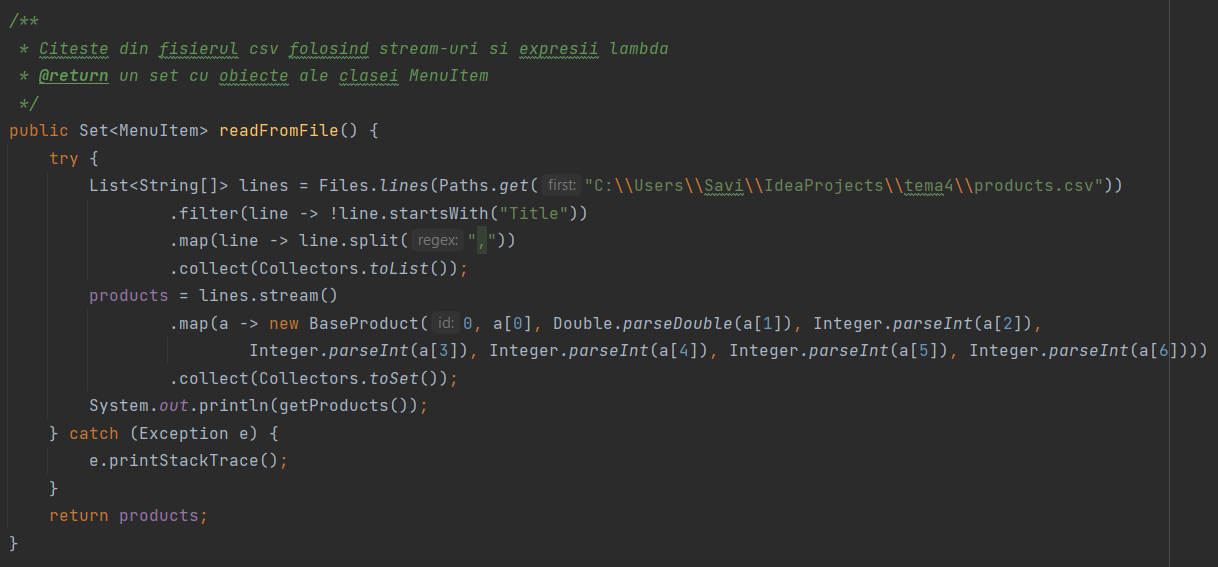
Clasa Order este o clasă care ajută la salvarea comenzilor realizate de clienți. În această clasă folosim următoarele field-uri: orderId, clientId, orderDate, date, client, order, price, startHour și endHour. De asemenea, aici aici am suprascris metodele equals și hashCode.

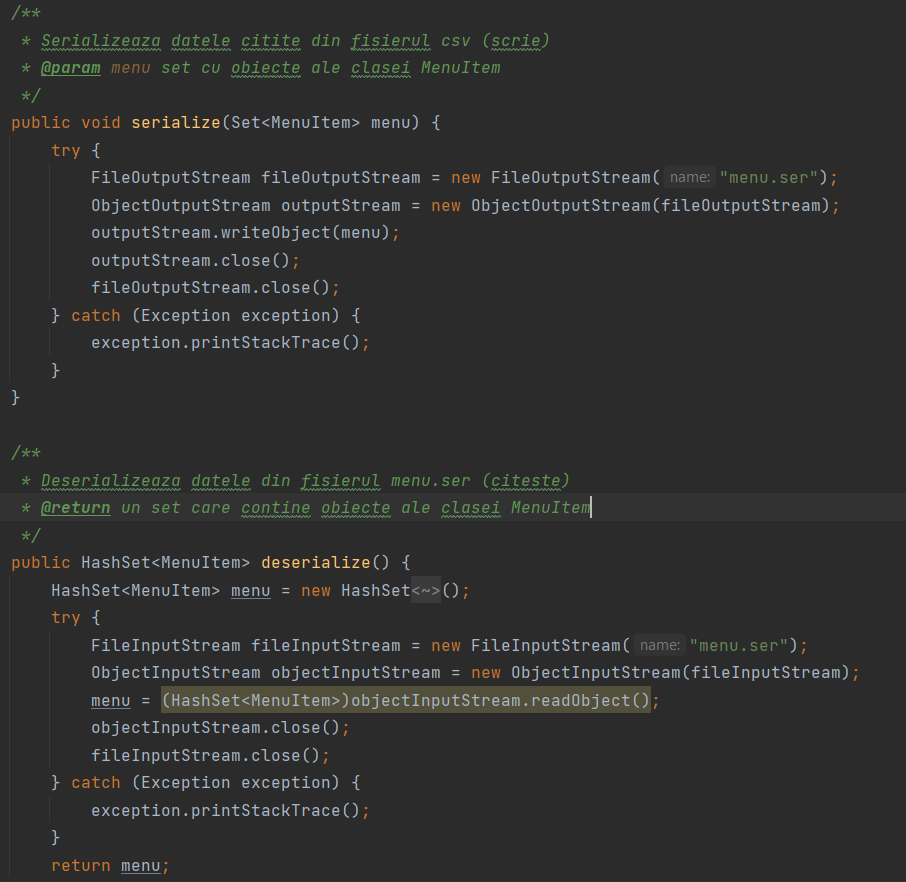


Clasa FileWrite este o clasă în care implementez metoda prin care realizez scrierea chitanței într-un fișier text, după ce un client realizează o comandă. Chitanța este rescrisă în același fișier după fiecare comandă plasată. Această metodă a fost documentată și folosind JavaDoc.

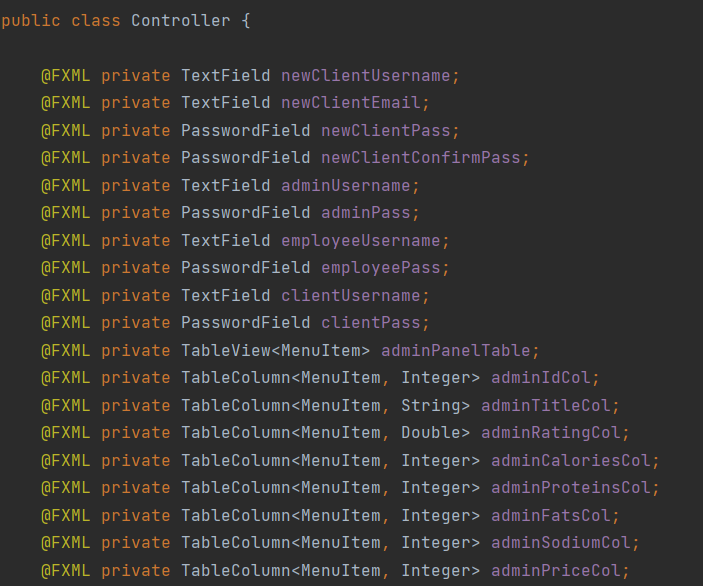


Clasa Serializator este o clasă în care se implementează metodele pentru: importarea meniului din fișierul products.csv, folosind stream-uri și expresii lambda, serializarea datelor și deserializarea acestora.

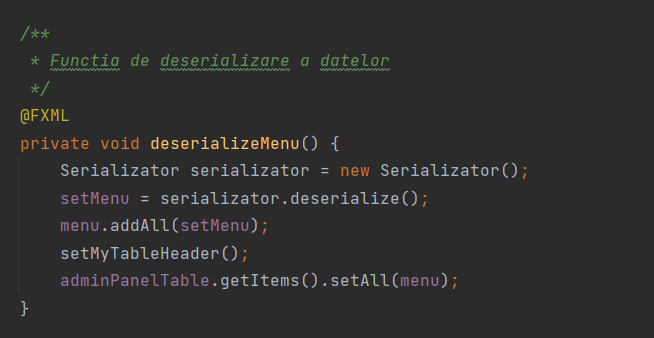




Clasa Controller este o clasă în care implementez toate metodele care se află la baza interfeței grafice, atât cele pentru partea de funcționalități pentru administrator, cât și partea de funcționalități pentru client. Aici am declarat toate TextField-urile, TextArea, TableView-urile și TableColumn-urile folosite, dar și metodele care corespund butoanelor din interfață. Toate acestea pot fi recunoscute după adnotația @FXML. Pe lângă acestea, am mai folosit și alte metode adiționale care ajută la lizibilitatea codului.





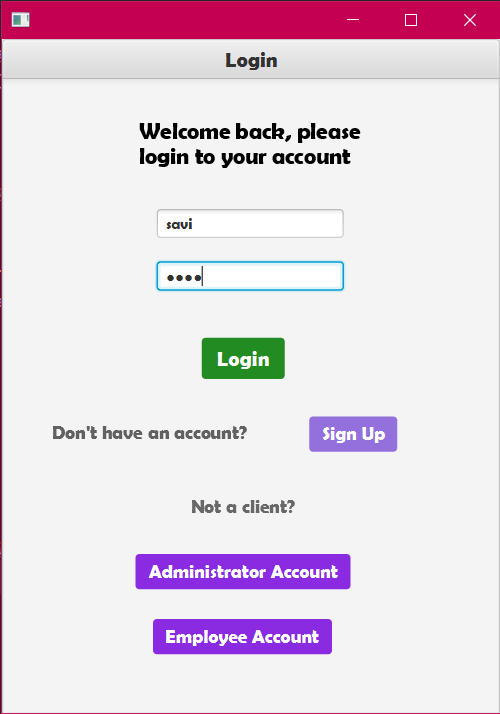




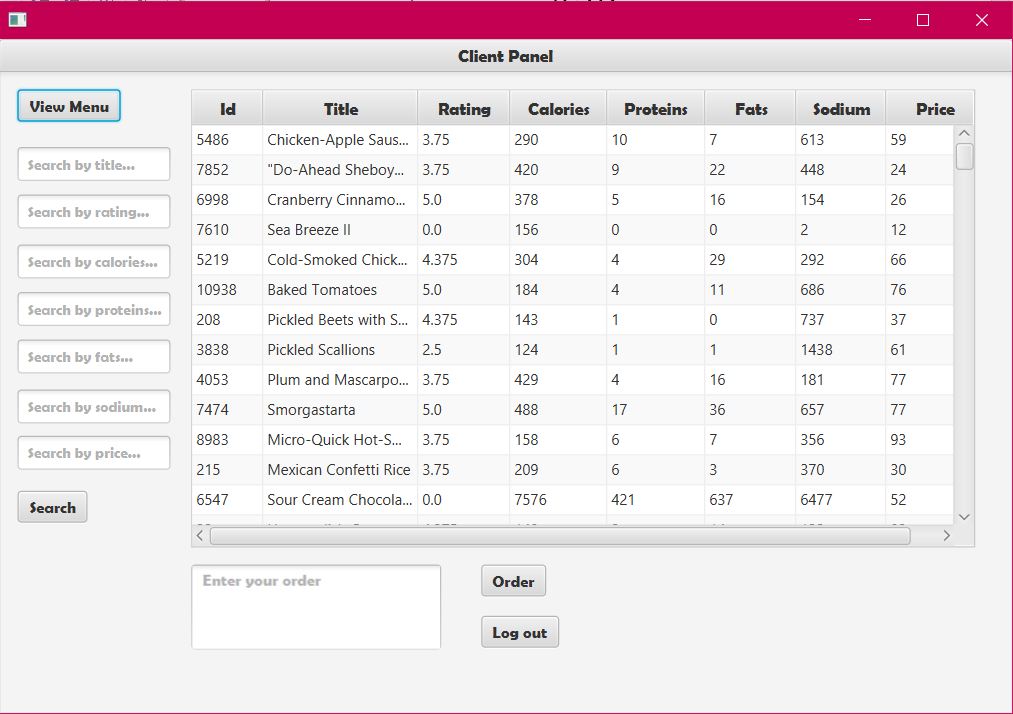
# **Rezultate**

Am testat aplicația verificând toate funcționalitățile implementate. Voi prezenta câteva screenshot-uri ale rezultatelor obținut în interfața care corespunde clientului, pentru început.

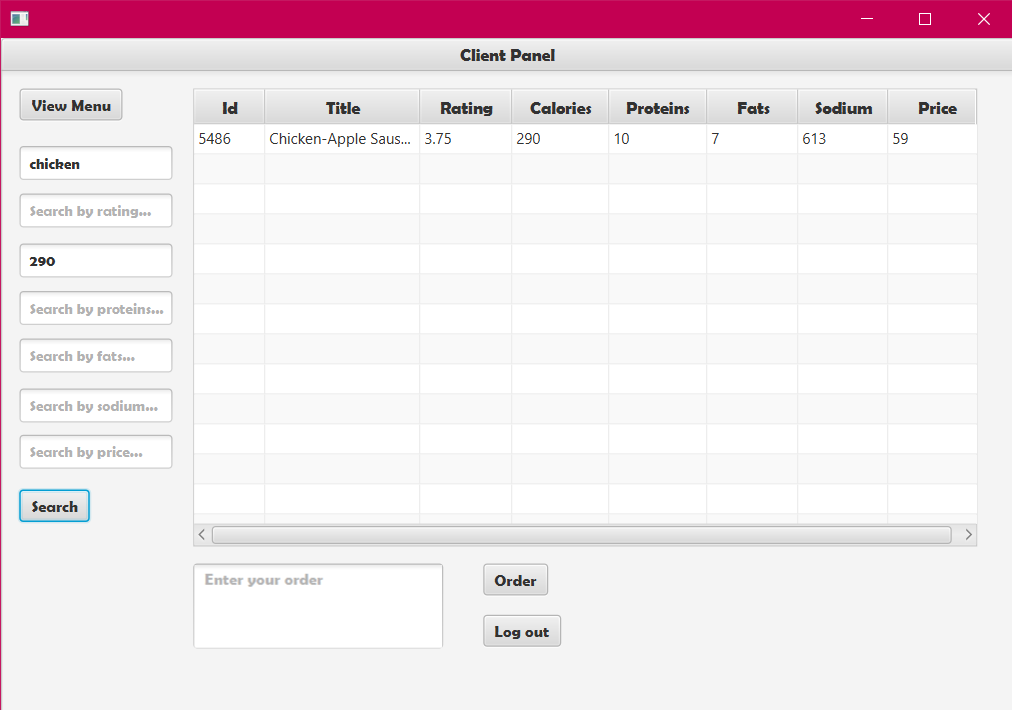
Utilizatorul alege să se logheze cu un cont deja existent, introduce username-ul și parola, apoi apasă butonul de login. Apoi, acesta va fi redirecționat către panel-ul clientului. Logarea clientului în aplicație:



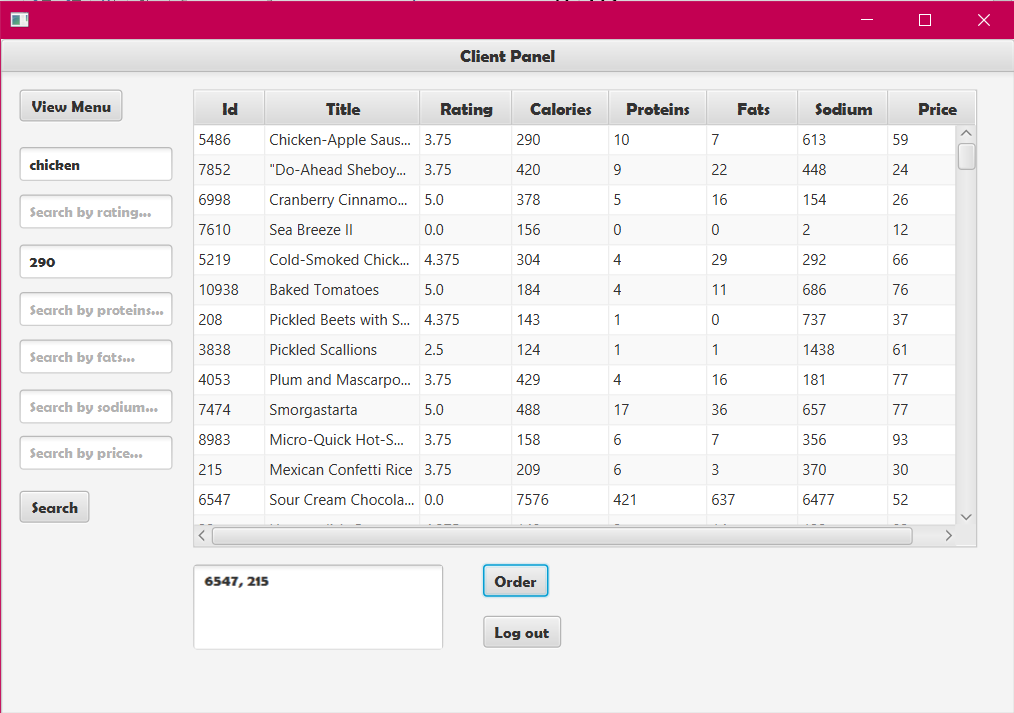
Odată ajuns în client panel, utilizatorul apasă butonul View Menu și poate vedea toate produsele din meniu. Vizualizarea meniului:



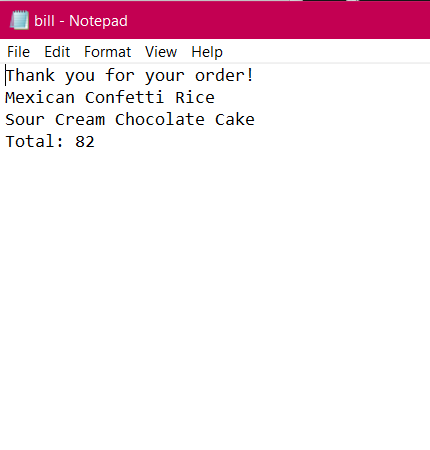
Apoi, utilizatorul caută un produs din meniu folosind un cuvânt cheie care se regăsește în titlul produsului și, de exemplu, un număr de calorii, după care obține un rezultat. Căutarea unui produs după câteva criterii:



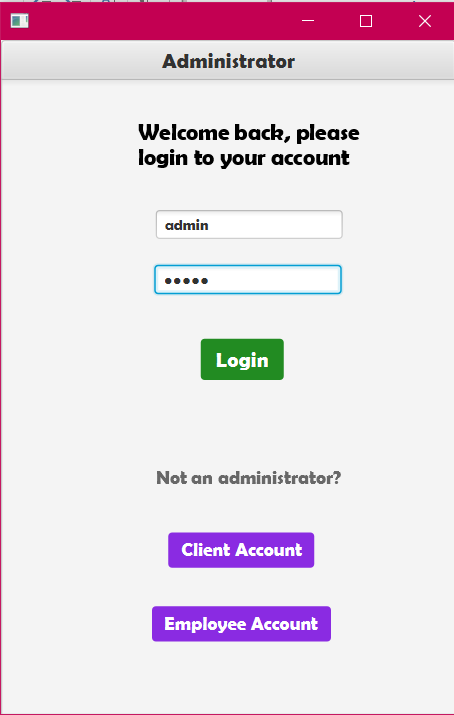
Apoi, utilizatorul poate sa introducă o serie de id-uri ale produselor pe care vrea să le comande, despărțindu-le prin virgulă. Plasarea unei comenzi:



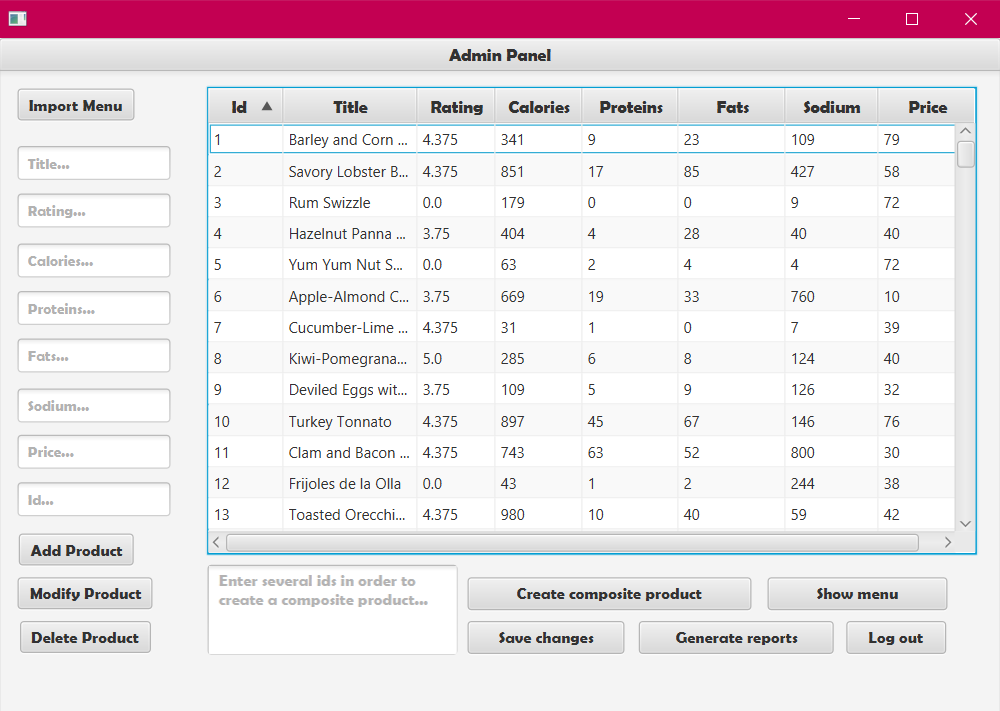
După plasarea comenzii, în fișierul text bill.txt se vor scrie următoarele date: mesajul ”Thank you for your order!”, denumirile produselor comandate pe câte un rând și prețul total. Acest fișier este întotdeauna rescris pentru fiecare comandă nouă. Obținerea chitanței:



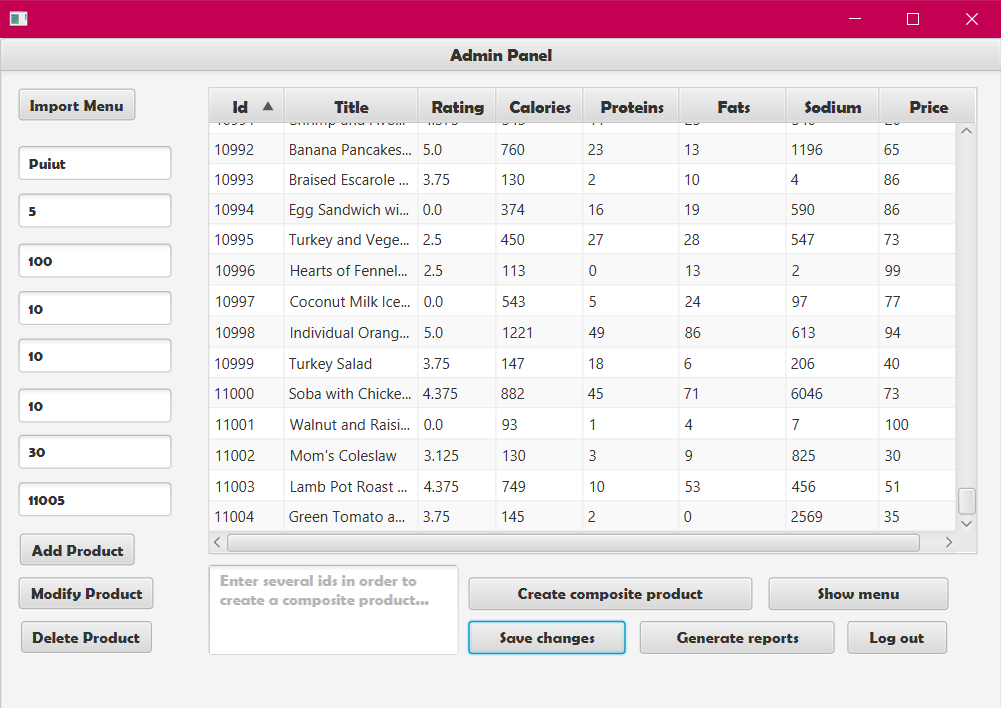
După ce termină toate operațiile pe care vrea să le efectueze, utilizatorul poate apăsa butonul de logout și să își schimbe statutul, din client în administrator. Alegând să fie administrator, acesta se va loga folosind username-ul admin și parola admin.

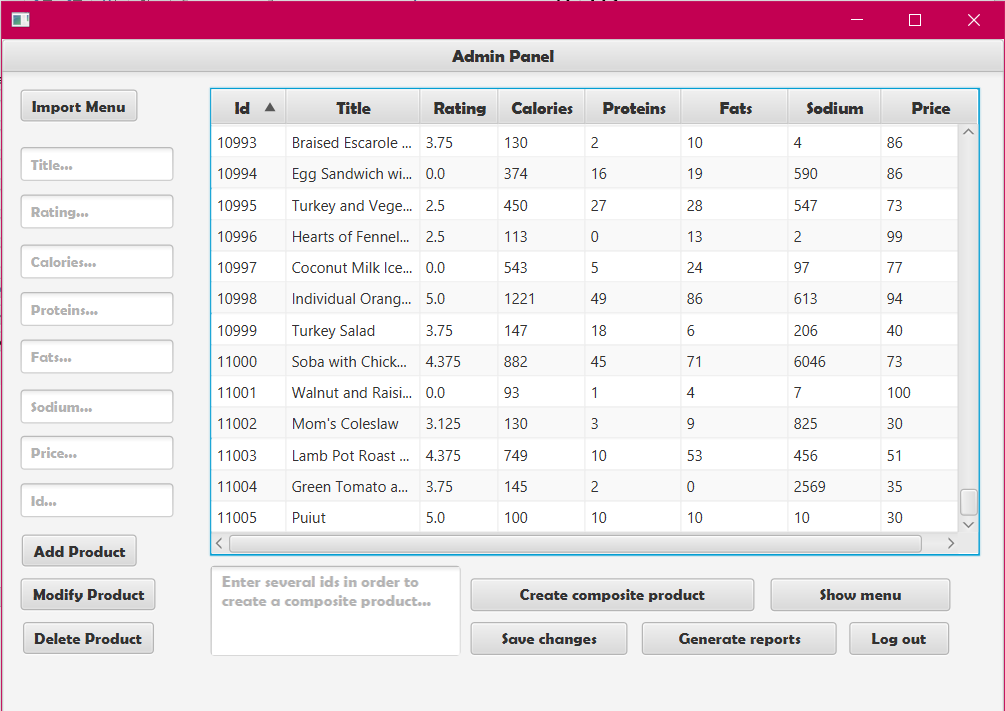


După ce se loghează, utilizatorul va vedea admin panel-ul, unde pentru început va importa datele din fișierul products.csv apăsând butonul ”Import Menu”.



Apoi, utilizatorul introduce un nou produs în meniu, completând field-urile de stânga, apoi apasă ”Add Product”, ”Save changes”. Pentru a vedea modificările, va trebuie să se relogheze:





# **Concluzii**

În concluzie, realizând această temă am reușit să mă familiarizez cu stream-urile, expresiile lambda și operațiile de serializare și deserializare a datelor, care facilitează atât scrierea, cât și citirea datelor. Toate aceste noi cunoștinte mi se par folositoare pentru viitoarele proiecte. Totuși, cel mai mult mi-a plăcut să mă gândesc și să lucrez la interfața grafică.

Ca dezvoltare ulterioară, aș vrea să implementez și funcționalitatea utilizatorului de tip angajat, pe care nu am reușit să o fac deocamdată și să îmbunătățesc importarea datelor, deoarece este cam lentă în acest moment.

# **Bibliografie**

<https://lucid.app/lucidchart>

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

<https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html>

<https://stackify.com/streams-guide-java-8/>

<https://www.w3schools.com/java/java_lambda.asp>

<https://dzone.com/articles/how-to-read-a-big-csv-file-with-java-8-and-stream>

<https://openjfx.io>

<https://www.oracle.com/java/technologies/javase/javafx-overview.html>

<https://www.oracle.com/java/technologies/javase/javafx-overview.html>