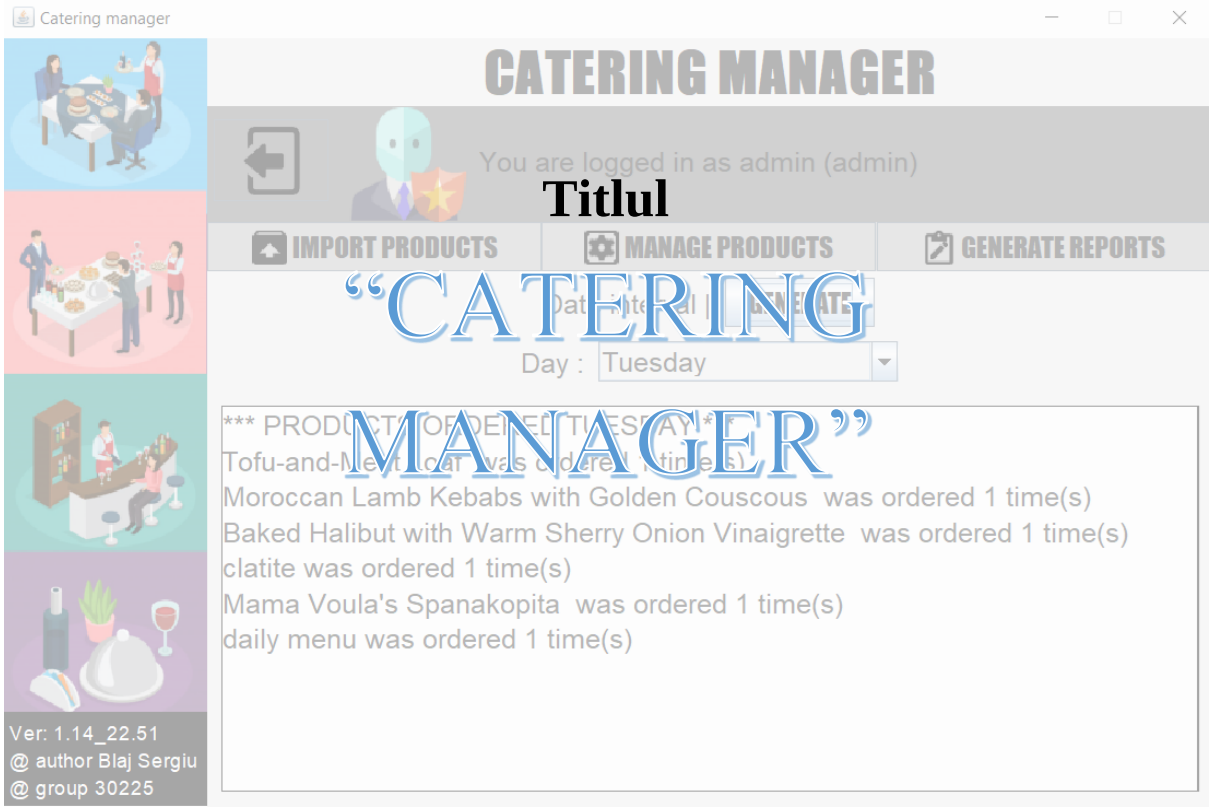
MINISTERUL EDUCAȚIEI ȘI CERCETĂRII

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

Proiect la disciplina

Tehnici de programare fundamentale



Titlul “CATERING MANAGER”

An universitar

2020 – 2021

Blaj Sergiu-Emanuel

An II, grupa 30225

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

Cuprins

1. Obiectivul temei

Obiectivul principal al temei îl reprezintă proiectarea și implementarea unei aplicații de simulare care procesează comenzile unor clienți. Suplimentar, acest simulator trebuie să asigure:

- utilizarea conceptelor de programare orientată pe obiecte;
- stocarea clineților, a produselor și a comenzilor folosind fișiere serialziate;
- dezvoltarea sa folosind un pattern arhitectural;
- dezvoltarea sa folosind minim patru pachete: business logic, data access, presentation și model;
- utilizarea JavaDoc pentru documentarea claselor;
- utilizarea design pattern-urilor: design by contract pattern, observer pattern și composite pattern;
- utilizarea stream-urilor și a expresiilor lambda;
- interacțiunea utilizatorului cu simulatorul printr-o interfață grafică user-friendly, pentru a facilita navigarea prin aplicație: o fereastră pentur client, în care poate vedea sau căuta produse și a plasa comenzi, o fereastră pentru administrator, unde poate prelucra produsele sau poate genera rapoarte și una în care poate genera rapoarte privind produsele;
- emiterea unui bon fiscal pentru fiecare comandă plasată, în fișierul “bill.txt”;
- testarea și depanarea programului, pentru a verifica funcționalitatea aplicației.

Figura 1

2. Analiza problemei

2.1. Cerințe funcționale

Aplicația dezvoltată trebuie să permită logarea celor trei tipuri de utilizatori: clienți, administratori și angajați. În cazul în care un utilizator are deja cont, acesta se poate loga, în panoul din dreapta; în caz contrar, acesta își poate face un cont nou, în panoul din stânga.

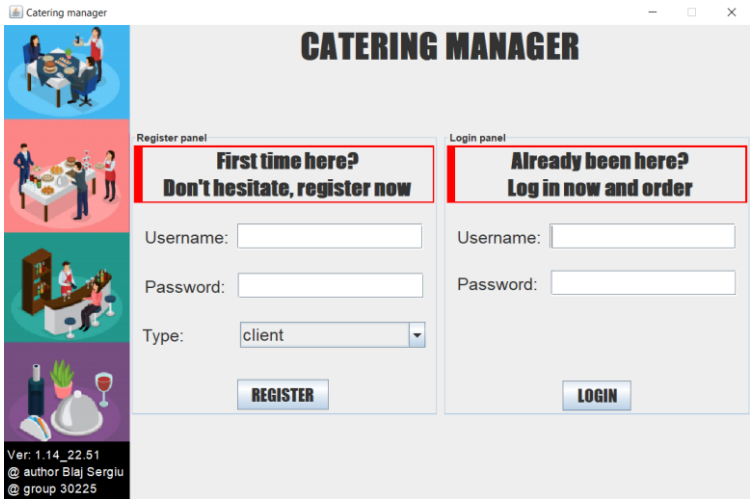


Figura 2.1.1

Administratorul trebuie să poată să importe din fișierul *products.csv* lista de produse. Poate face aceasta în panoul prezentat mai jos.

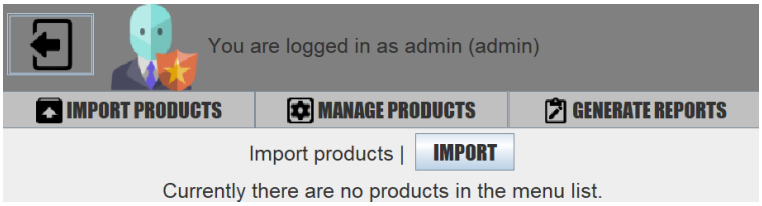


Figura 2.1.2

Apoi, administratorul poate adăuga, modifica sau șterge produse din lista de produse.

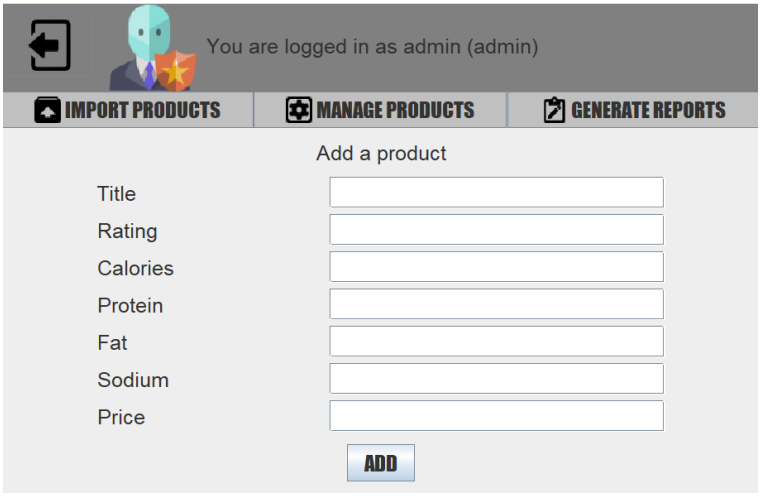


Figura 2.1.3

O altă atribuție a administratorului este de a genera varii rapoarte în funcție de anumite valori. Acest lucru este posibil în panoul de mai jos.

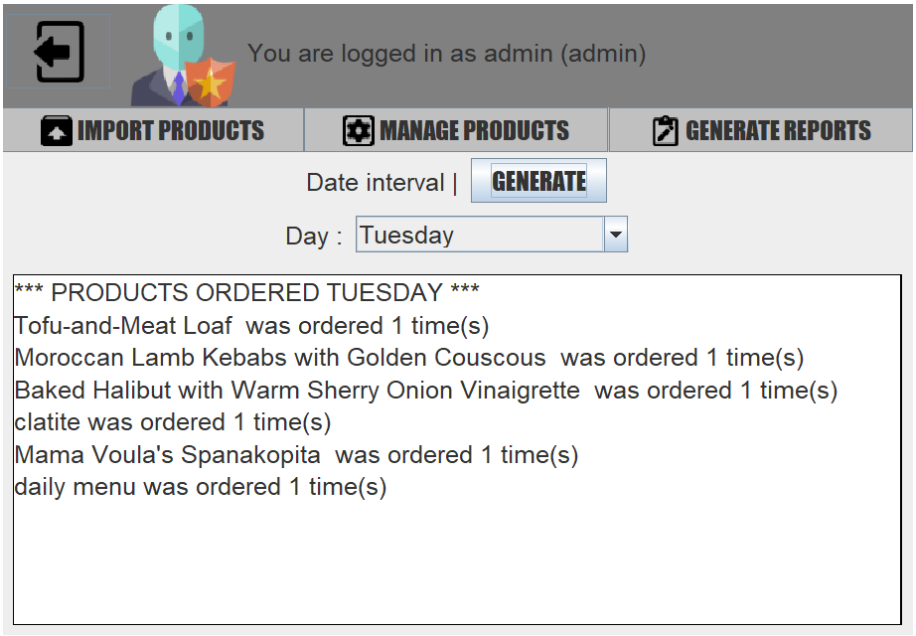


Figura 2.1.4

Clientul trebuie să aibă posibilitatea de a se loga în cont sau de a se înregistra. Aces lucru a fost prezentat în figura 2.1.1. Tot clientul este cel care poate vedea lista de produse sau căuta produse în această listă.

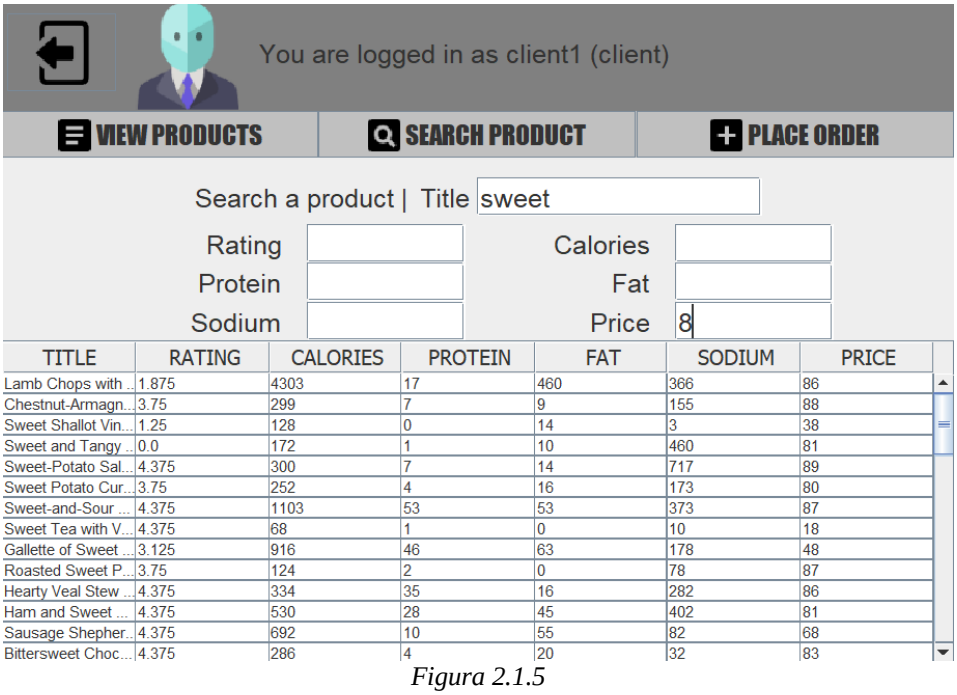


Figura 2.1.5

După ce a vizualizat produsele, clientul trebuie să aibă posibilitatea de a plasa o comandă. Acest lucru se realizează în panoul de mai jos.

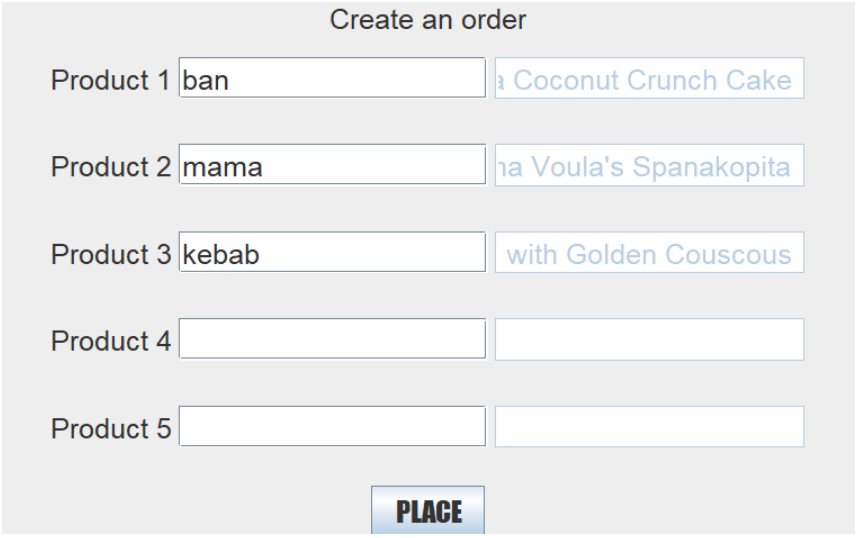


Figura 2.1.5

Ultimul, dar nu cel din urmă utilizatorul este angajatul, care trebuie să fie notificat de fiecare dată când este plasată o nouă comandă.

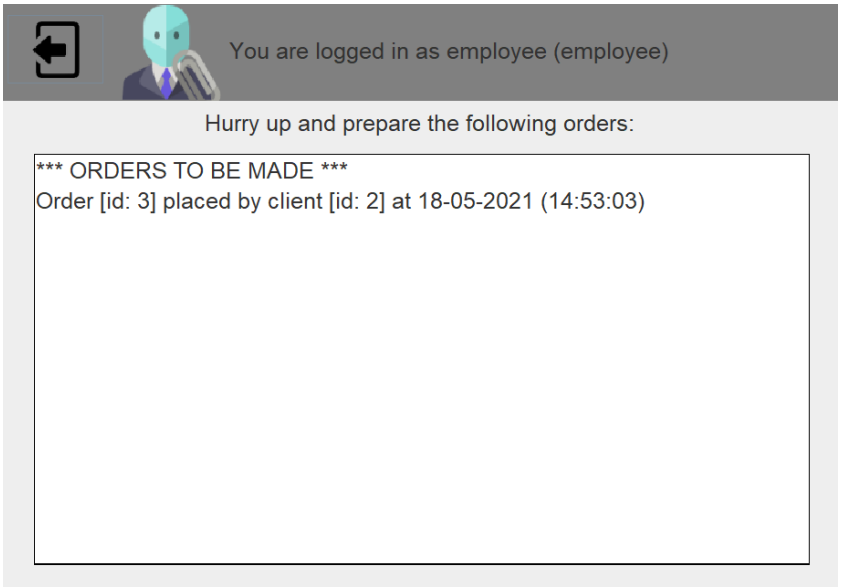


Figura 2.1.6

2.2. Cerințe non-funcționale

Pe lângă cerințele de bază, am considerat că următoarele cerințe vor asigura o mai bună interacțiune între calculatorul de polinoame și utilizator. Prima dintre ele, poate chiar cea mai importantă, este ca interfața grafică să fie una intuitivă, astfel ca și cuiva care nu obișnuiește să folosească o aplicație pe calculator să îi fie simplu să o folosească.

În figura 2.2.1 se poate observa că aplicația îndeplinește acest criteriu. În partea stângă, este prezentă o imagine legată de tematica aplicației cât și date despre dezvoltator. În partea principală, apare titlul aplicației, urmat de zonele interactive.

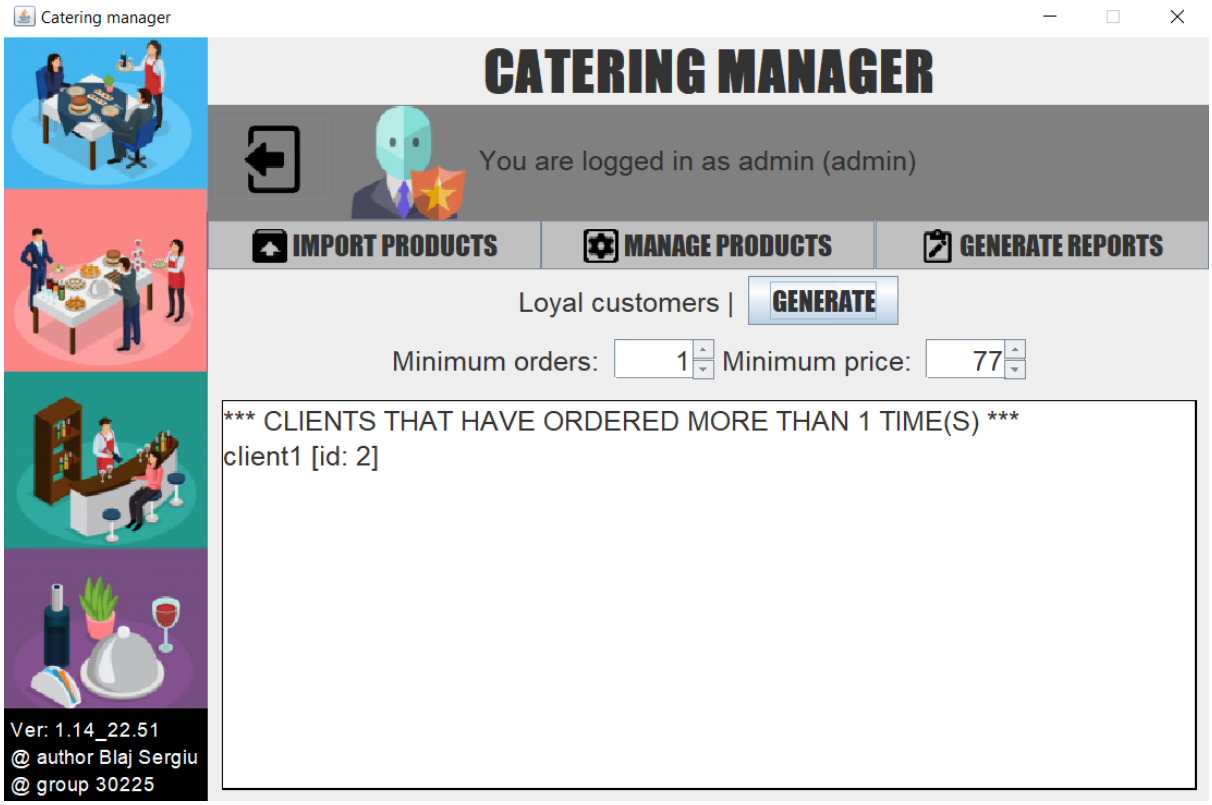


Figura 2.2.1

O altă cerință non-funcțională, derivată din cea menționată mai sus, o reprezintă aspectul plăcut al butoanelor și imaginile reprezentative pe care acestea le conțin.

Figura 2.2.2

2.3. Scenarii de utilizare

Figurile următoare descriu posibilele scenarii de utilizare ale aplicației. Erorile care pot apărea în acest sunt reprezentate de introducerea incorectă a datelor despre client, produse sau comenzi, precum vârsta incorectă, email invalid, sau, în cazul unei comenzi, un id de client sau produs care nu există. Pentru simplitate, vom considera numai operațiile aplicate pe clienți, cele pe produse sau comenzi fiind identice.

Figura 2.3.1 descrie cazul de utilizare al aplicației în care utilizatorul dorește să își facă cont nou.

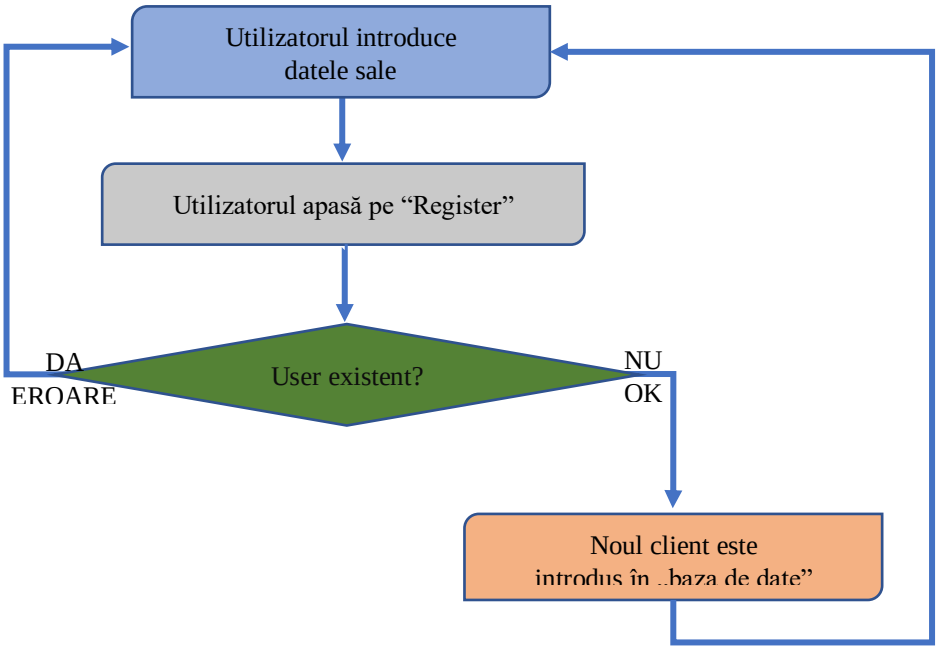


Figura 2.3.1

Iar în figura de mai jos este descris scenariul de utilizare pentru logarea clientului în aplicație.

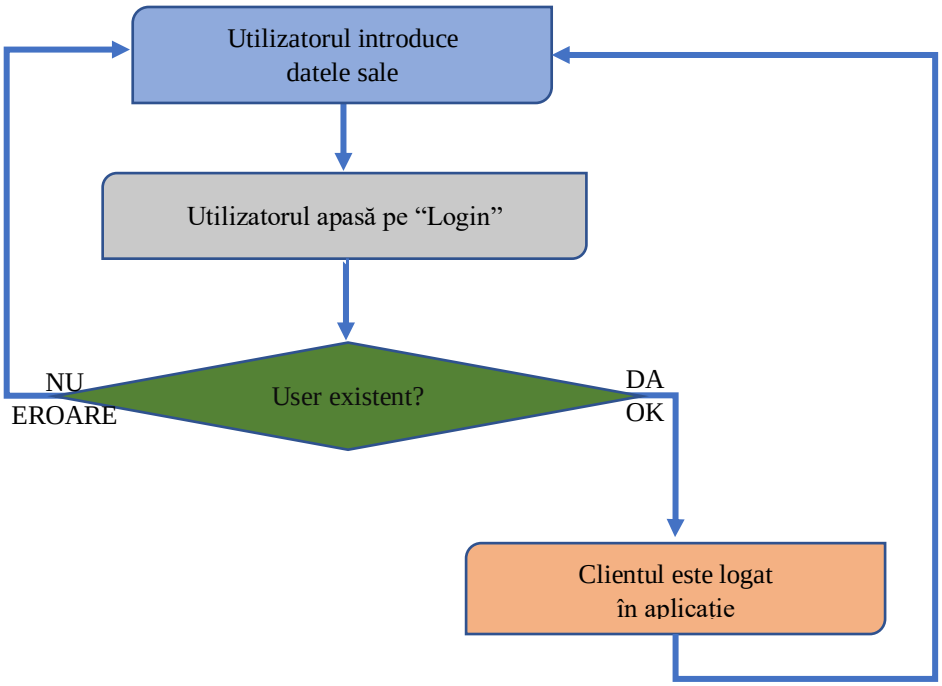


Figura 2.3.2

Figura 2.3.3 descrie scenariul de utilizare în cazul în care administratorul dorește să adauge, să editeze, să șteargă sau să creeze un nou produs.

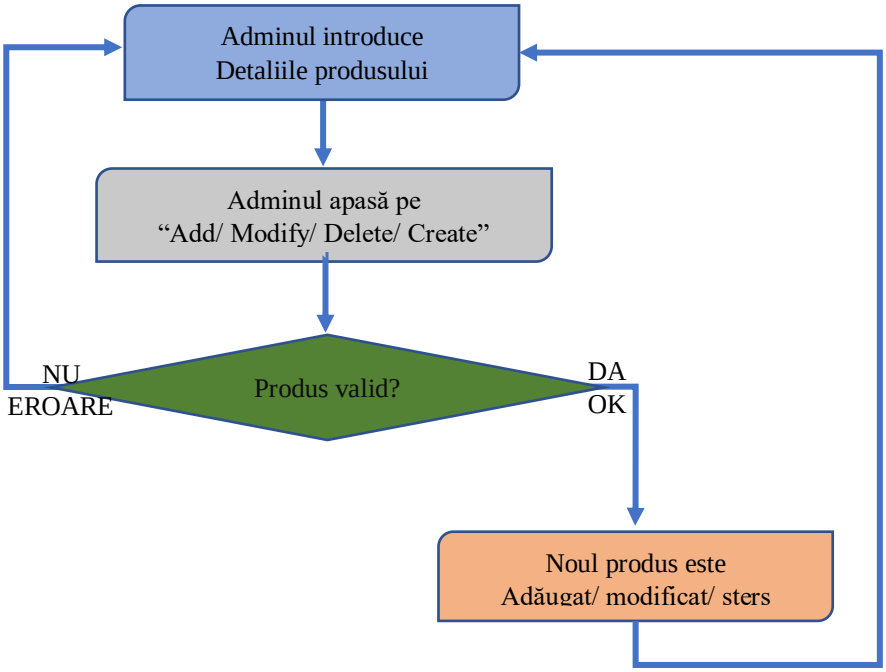


Figura 2.3.3

Figura 2.3.4 descrie scenariul de utilizare în cazul în care administratorul dorește să genereze rapoarte.

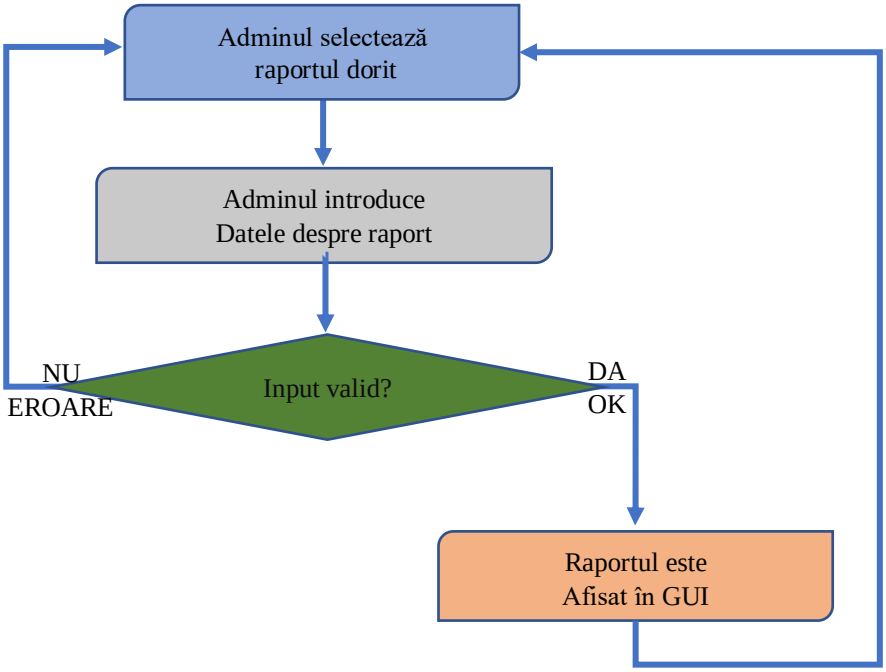


Figura 2.3.4

Figura 2.3.5 descrie scenariul de utilizare în cazul în care clientul dorește să caute un produs sau să plaseze o comandă.

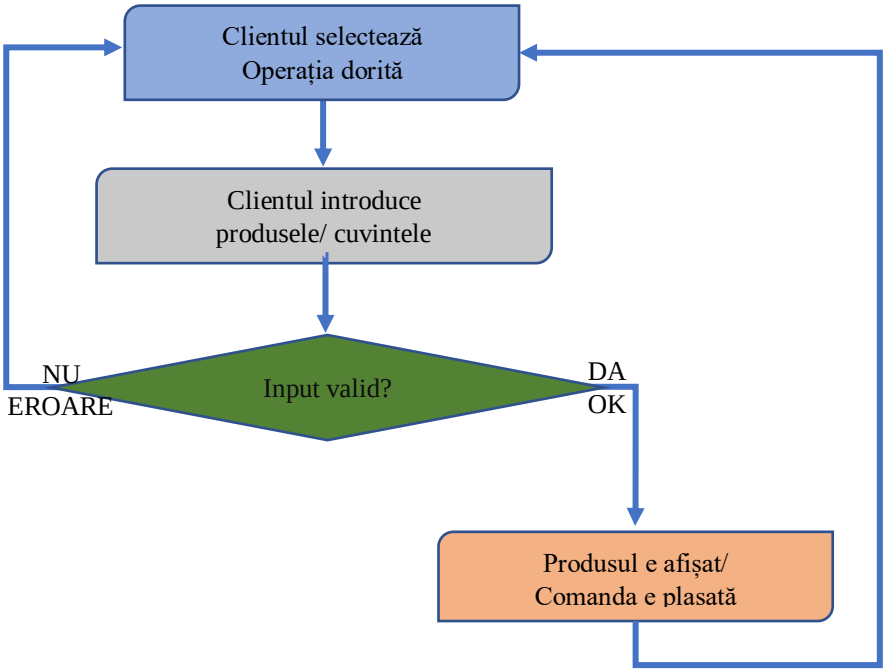


Figura 2.3.4

3. Proiectare

3.1. Concepte ale programării orientate pe obiect folosite

Aplicația creată a fost implementată în limbajul de programare Java. Fiind dezvoltată într-un limbaj de programare orientat pe obiecte, aplicația modelează obiecte, precum *Client, Product* sau *Order*, utilizează concepte ale programării orientate pe obiect, precum moștenirea, polimorfismul, suprascrierea metodelor, supraîncărcarea metodelor (în special a constructorilor), implementarea de interfețe (în cadrul claselor "ascultători", spre exemplu *CreateObjectListener*, care implementează interfața *ActionListener*, utilă pentru interacțiunea utilizatorului cu interfața grafică.

Deoarece formatul instrucțiunilor SQL asupra manipulării obiectelor de tip *Client, Product* sau *Order* sunt asemănătoare, dar și panourile din interfața grafică referitoare la acestea, a fost folosită tehnica reflexiei. De asemenea, aplicația a fost organizată într-o arhitectură "pe straturi", ce utilizează mai multe pachete, care va fi prezentată în cele ce urmează.

3.2. Pachetul Presentation

În figura 3.2 este reprezentată structura pachetului *Presentation*. Aici au fost considerate clasele care alcătuiesc interfața grafică a aplicației, cât și panourile care conțin operațiile pe clienți, produse sau comenzi.

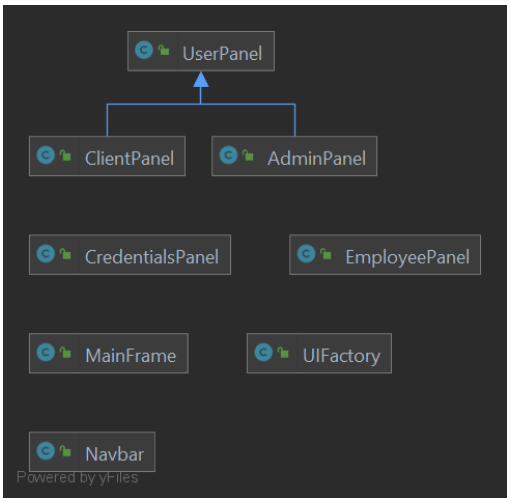


Figura 3.2

3.3. Pachetul Model

Este pachetul care conține cele trei clase care sunt mapate în „baza de date", cele cu care lucrează aplicația.

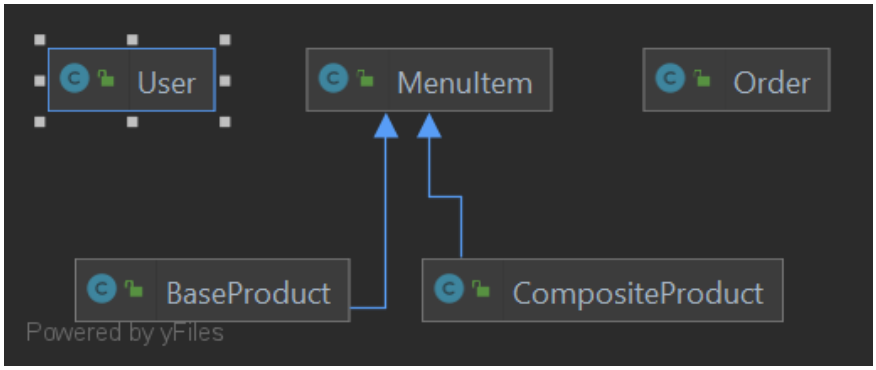


Figura 3.3

3.4. Pachetul Business Logic

Este pachetul care conține logica aplicației, și anume clasele *ascultători*, cele care asigură tratarea evenimentelor în momentul apăsării butoanelor, dar și clasele *IdeliveryServiceProcessing* și *DeliveryService*, cea care conține operațiile executate de către client și administrator, precum crearea unor produse sau generarea de rapoarte.

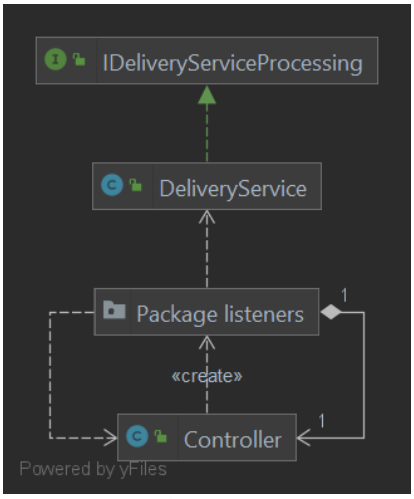


Figura 3.4

3.5. Pachetul Service

În acest pachet sunt conținute clasele care efectuează persistența datelor referitoare la useri, comenzi și produse, prin serializarea acestora în fișiere.

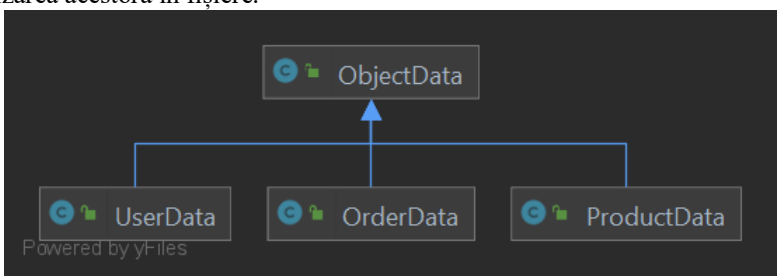


Figura 3.5

4. Implementare

În cele ce urmează, vor fi descrise clasele folosite de aplicație și modul lor de funcționare. Pentru o mai bună organizare, acestea vor fi descrise în cadrul secțiunii pachetului din care acestea fac parte.

4.1. Pachetul Presentation

Pachetul conține clasele care alcătuiesc interfața grafică a aplicației; panourile care conțin operațiile pe clienți, produse sau comenzi.

Clasa *MainFrame* este responsabilă pentru afișarea interfeței grafice. Clasa conține constante String și int, pentru a evita folosirea "numerelor magice" în program, apoi o listă de panouri și butoane. Construcția ferestrei a fost împărțită în mai mulți pași, fiecare pas fiind dictat de constructorul clasei. Spre exemplu, *void setUpFrame()* este responsabilă de a pregăti JFrame-ul (fereastra aplicației), setând dimensiunea ei, titlul ș.a.m.d., astfel că fiecare funcție face ceea ce conține în nume.

Sunt adăugate, rând pe rând, partea din stânga a interfeței, cea care conține imaginea legată de tematica aplicației, informații despre autor, titlul aplicației precum și altele legate de simulare.

Urmează partea interesantă. Observând că fiecare zonă corespunzătoare efectuării operațiilor de login sau register conține aceleași informații, am creeat clasa *CredentialsPanel*, care extinde clasa JPanel și conține o listă de JLabel-uri și de JTextField-uri care vor ține locul textului potrivit.

Cu ajutorul diferitelor metode, se creează iluzia existenței mai multor paneluri interioare: În funcție de operația selectată, șterge anumite texte din panou și le afișează pe altele (de exemplu, metoda *void showAddProductPanel()* va afișa toate inputurile legate de un produs de bază, precum titlu, cantitatea de sodiu, numărul caloriilor, prețul etc).

Pentru ca fiecare utilizator să-și vadă informațiile sale, am creeat clasa *Navbar* care conține o poză cu userul (diferită de la client la admin sau angajat), numele, funcția și un buton de logout din aplicație.

4.2. Pachetul Model

Pachetul conține clasele mapate în baza de date, și anume *Order, User, MenuItem, BaseProduct* și *CompositeProduct*. Fiecare este definită de variabile instanță specifice obiectului: pentru un utilizator vor fi reținute numele și parola, pentru o comandă va fi reținut id-ul acesteia, data, precum și clientul care a plasat-o, iar pentru produse vor fi reținute titlul, cantitatea de grăsimi, sodiu, caloriile, prețul etc.

Pe lângă getteri și setteri, fiecare clasă suprascrie metodele *hashCode și equals*, deoarece plasarea lor în cadrul tabelelor de dispersie corespunzătoare va fi făcută apelând metodele mai sus menționate. De precizat este faptul că metoda *hashCode* returnează un hash în funcție de numele obiectului, acesta fiind singurul mod în care pot fi distinse două produse la fel sau doi clienți (numele, tipul, parola și id-ul fiind diferite, în eventualul caz al înregistrării).

4.3. Pachetul Business Logic

Pachetul conține clasele care asigură logica aplicației. Printre acestea, menționăm clasele care modifică obiectele din baza de date. Aceastea primesc obiecte create de clasele din pachetul *Presentation* cu ajutorul *Controllerului.* Înainte de a manipula baza de date, aceste obiecte trebuie să fie validate, fiecare în funcție de tipul său.

Tot în acest pachet au fost definite clasele ce vor asigura interacțiunea utilizatorului cu programul. Clasa principala, *Controller*, asignează fiecărui buton din interfața grafică, o utilizare. Așadar, fiecare buton din program are câte o clasă ce corespunde unui "ascultător" diferit (de exemplu, *AddProductListener* este adăugat butonului de adăugare, care, odată apăsat, va prelua valorile din zonele dedicate introducerii acestora, le va valida, iar în caz că acestea sunt corecte, va introduce un nou produs în baza de date).

Alte două clase care trebuie menționate aici sunt clasele *IDeliveryServiceProcessing* și *DeliveryService*. Acestea conține metodele care sunt folosite de clienți sau de administratori pentru a gestiona produsele și comenzile. Printre acestea trecem în revistă operațiile de creare, adăugare, modificare sau ștergere a unui produs, respectiv cele de generare a rapoartelor, în cazul administratorilor, sau cele de plasare a unei comenzi, în cazul clienților. Toate aceste metode au fost create folosind stream-uri și expresii lambda.

4.4. Pachetul Service

Pachetul conține clasele care asigură persistența datelor, prin serializarea acestora. Clasa *ObjectData* se ocupă cu serializarea și deserializarea obiectelor, iar *OrderData, ProductData* și *UserData* folosesc apelează acele două metode pentru adăugarea, ștergerea, căutarea etc a unui obiect în cele serializate în fișiere.

5. Rezultate

În cele ce urmează vom trece în revistă câteva dintre mesajele de alertă pentru a demonstra corectitudinea aplicației. La crearea unui client, în cazul în care utilizatorul utilizează un nume de client deja existent, acesta va fi semnalat printr-o eroare. În caz contrar, câmpurile se validează cu succes iar noul client va fi inserat în baza de date.

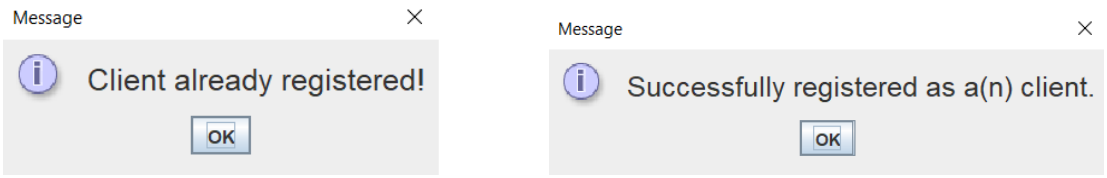


Figura 5.1

Fiecare utilizator se loghează în aplicație cu un nume și o parolă. Acestea trebuie să fie valide, în caz contrar, una din erorile de mai jos vor apărea.

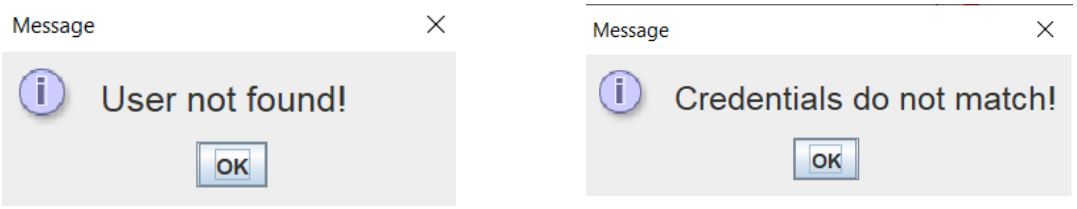


Figura 5.2

În caz contrar, logarea se face cu succes. Dacă userul dorește să se delogheze, acesta o poate face dând click pe iconița de logout. Un mesaj de alertă va fi afișat.

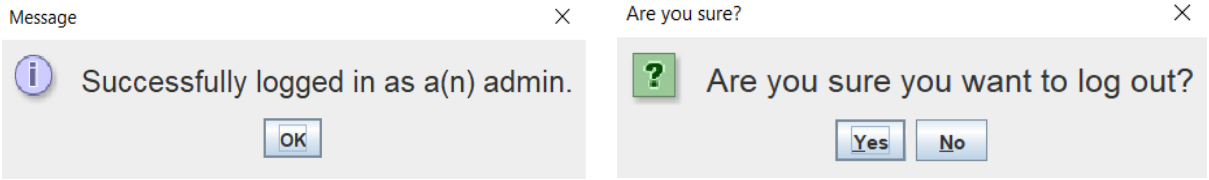


Figura 5.3

6. Concluzii

În aproape fiecare afacere este nevoie ca beneficiarii activității prestate de un antreprenor, clienții, să fie menținuți într-o bază de date, pentru eventuale prelucrări sau în scopuri de marketing. De asemenea, o afacere care oferă produse destinate vânzării trebuie să țină evidența acestora, iar cer mai simplu ar fi păstrarea lor într-o bază de date sau fișiere serializate. Astfel, în cazul în care un client dorește să achiziționeze un anumit produs, va creea o comandă, care va fi inserată în baza de date și care va ține anumite detalii, precum clientul care a plasat comanda, produsul comandat, data plasării comenzii, cantitatea comandată, prețul comenzii etc.

În urma acestei teme, consider că mi-am îmbunătățit stilul de programare, respectând convențiile de numire Java sau de organizare a codului în pachete și subpachete. De asemenea, mi-am consolidat cunoștințele deja existente legate despre programarea orientată pe obiecte, precum moștenirea, polimorfismul sau elementele de interfață grafică din pachetul Java.swing dar și tipurile generice, punând bazele unor cunoștințe noi de Java, cum ar design pattern-urile, streamurile și expresiile lambda.

Pe viitor, ar putea fi dezvoltată o versiune în care produsele vor avea imagini sugestive, nu doar nume și descirere. La fel s-ar putea proceda și pentru clienți, fiecare având posibilitatea de a avea un avatar.

O altă posibilă dezvoltare ulterioară ar putea permite căutarea unui client după nume, adresă sau email, la fel și în cazul unui produs sau, mai important, a unei comenzi. Luând ca exemplu lumea reală, fiecare client ar putea avea o anumită sumă de bani, care, asemenea stocului produselor în cazul plasării unor noi comenzi, se decrementează, la fel s-ar putea întâmpla și cu această sumă de bani. În cazul unui client care plasează comenzi regulat, sau de o valoare mai mare, acesta ar putea primi un anumit discount.

7. Bibliografie

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/

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

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

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

https://intellij-support.jetbrains.com/hc/en-us/community/posts/207014815-How-toenable-assert

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