**DOCUMENTAȚIE TEMA 4**

**FOOD DELIVERY MANAGEMENT SYSTEM**

**Student: Raul - Alexandru Gorgan**

**Grupa: 30227**

**Profesor Laborator: Dan Mitrea**

Contents

[1. Cerinte Functionale 3](#_Toc476131445)

[2. Obiective 3](#_Toc476131446)

[2.1. Obiectiv Principal: 3](#_Toc476131447)

[2.2. Obective Secundare: 3](#_Toc476131448)

[3. Analiza Problemei 3](#_Toc476131449)

[4. Proiectare 3](#_Toc476131450)

[4.1. Structuri de date 3](#_Toc476131451)

[4.2. Diagrama de clase 3](#_Toc476131452)

[4.3. Algoritmi 3](#_Toc476131453)

[5. Implementare 4](#_Toc476131454)

[6. Testare 4](#_Toc476131455)

[7. Concluzii si Dezvoltari Ulterioare 4](#_Toc476131456)

[8. Bibliografie 4](#_Toc476131457)

# Cerințe Funcționale

Se cere implementarea unei aplicații de management al comenzilor unei firme de catering. În cadrul acestei aplicații sunt 3 tipuri de useri: administrator, clienți și angajați. Un administrator este cel mai privilegiat și poate să facă următoarele acțiuni: adăugarea meniului, adăugare de produse simple sau compuse, modificare, ștergere produse, generare rapoarte. Un client se poate înregistra în aplicație cu username și parola și poate efectua comenzi. Angajatul este notificat când se plasează o comandă nouă, pentru a știi să pregătească comanda. Pentru fiecare comadă se compune o factură care conține produsele comandate și prețul total.

# Obiective

## Obiectiv Principal:

Obiectivul principal al acestui proiect este de propunere, proiectare și implementare al unei aplicații de management al comenzilor unei firme de catering. Este o aplicație cu o interfață grafică în care fiecare user se poate conecta sau înregistra și efectua diferite operații. Administratorul se ocupă de intreținerea meniului, clientul efectuează comenzi iar angajatul este notificat să știe să pregătească comanda. Toate aceste operații se fac în interfața grafică, pentru a face mai ușoară și mai atractivă comunicarea, utilizatorul fiind cel care introduce datele dorite și efectuează operațiile dorite.

## Obiective Secundare:

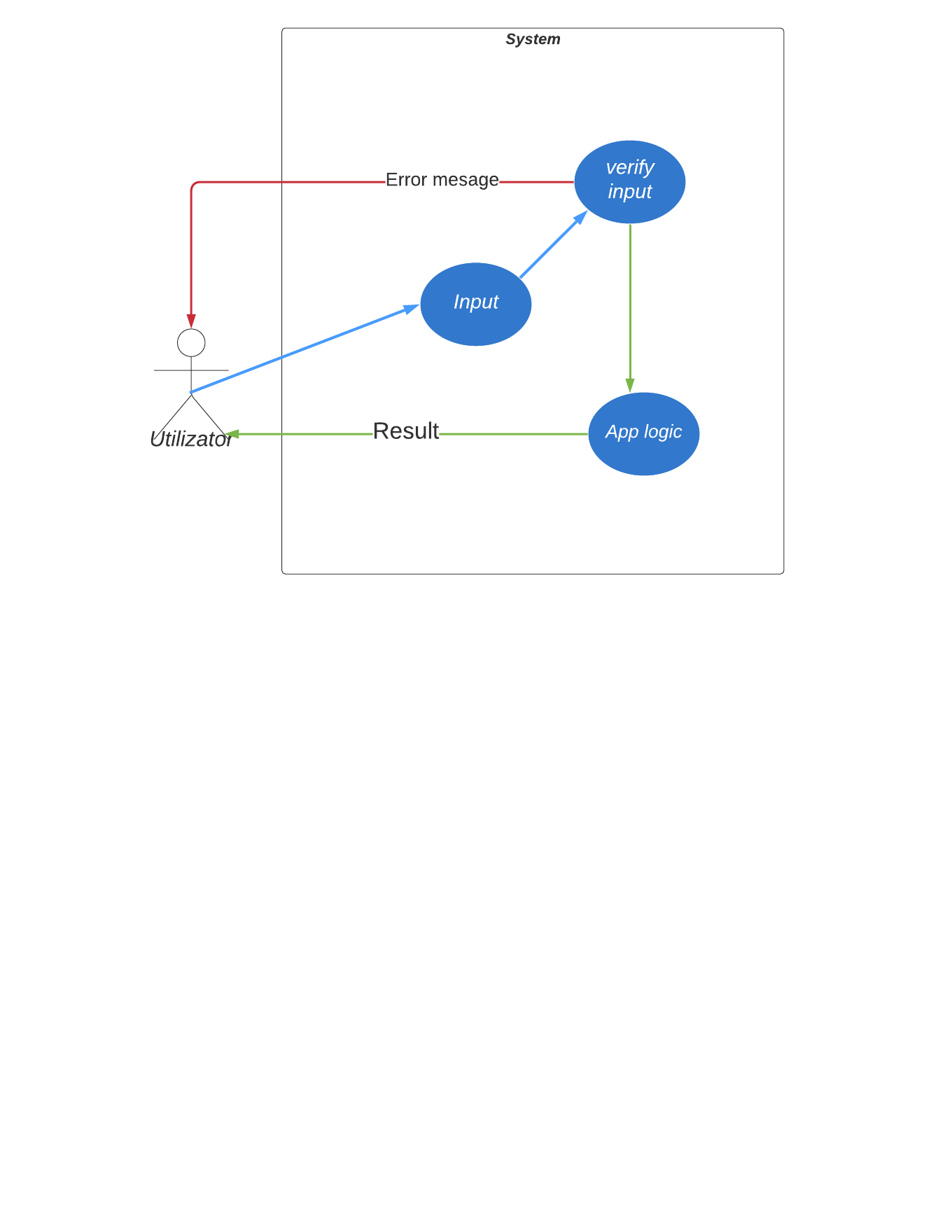
Pentru a putea îndeplinii obiectivul principal al acestui proiect, este nevoie de stabilirea unor obiective secundare care ne vor conduce la realizarea lui. Stabilirea acestora este importantă înainte de implementarea unei aplicații, deoarece o proiectare bună poate aduce multe beneficii, atât pentru proiectant cât și pentru utilizator. O bună alegere a soluției poate face ca aplicația să fie mai ușor de scris, întreținut, chiar și de promovat. Utilizatorul dorește să aibă o aplicație care efectuează calculele într-un timp minim, să fie user-friendly, să lucreze ușor și intuitiv. Aceste avantaje pot fi rezultatul unei analize în detaliu și a unei proiectări de calitate.

Un obiectiv secundar care intră în discuție la acest proiect este analiza problemei prin dezvoltarea de use case-uri și de scenarii, care vor fi prezentate în capitolul numărul 3. Acesta ne va ajuta la îndeplinirea unui alt obiectiv secundar și anume alegerea structurilor de date folosite, alegerea fiind argumentată la capitolul 4. Tot la capitolul 4 vor fi dezvoltate și celelalte obiective secundare: împărțirea pe clase, respectând paradigmele programării orientate pe obiecte și dezvoltarea algoritmilor esențiali. După analiza în detaliu a problemei, la capitolul 5 se descrie pe larg implementarea soluției, aceasta fiind urmată de faza de testare, ultimul obiectiv secundar în vederea finalizării aplicației.

# Analiza Problemei

Analiza problemei începe cu identificarea use case-urilor și a scenariilor. Niciodată nu este garantat că utilizatorul o să introducă datele exact în formatul specificat sau o să urmeze pașii în ordinea corectă, așa că este nevoie de analiza tuturor cazurilor de utilizare și a cazurilor limită. Aplicația trebuie să acopere toate aceste cazuri și să afișeze mesaje sugestive utilizatorului.

Diagramele de use case sunt o variantă vizuală a pașilor care se parcurg la o simulare:



Aceasta este diagrama generală de use case. Parcursul albastru e comun celor două scenarii (scenariul încheiat cu succes și scenariul alternativ / încheiat cu afișarea unui mesaj de eroare). Dacă inputul e corect, parcursul va urma linia verde, dacă nu, va urma linia roșie.

Use case adăugare comandă:

Actorul principal: utilizatorul

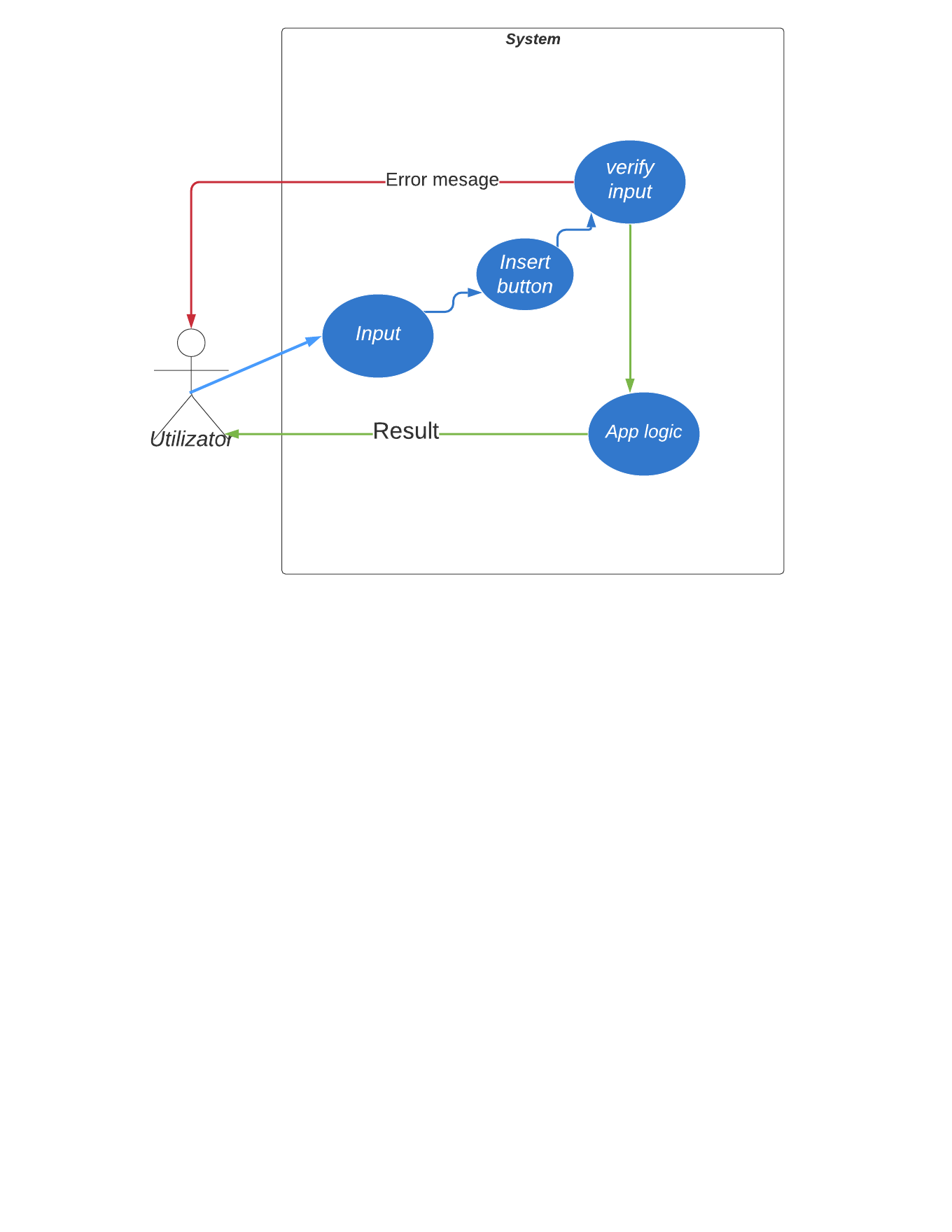
Scenariul de success:

1. Utilizatorul intră în sistem cu username și parolă
2. Utilizatorul deschide meniul
3. Utilizatorul selectează din tabel produsele dorite
4. Aplicația extrage datele de la utilizator și plasează comanda
5. Utilizatorului îi este scrisă o factură

Scenarii alternative:

* Utilizatorul nu selectează niciun produs
* În acest caz, îi va fi afișat un mesaj de eroare iar execuția aplicației va continua întorcându-se la pasul 3.

Use-case pentru place order:



# Proiectare

## Structuri de date

Pornind de la cerință și de la analiza problemei, sistemul necesită o fereastră principală în care fiecare utilizator se poate înregistra sau loga în sistem pe baza unui username și a unei parole. În funcție de tipul de user, se vor deschide interfețe diferite, corespunzătoare administratorului, clientului și angajatului. Clienții adaugă comenzi iar administratorul și angajatul trebuie să lucreze cu ele, deci datele unei comenzi se vor stoca și ele.

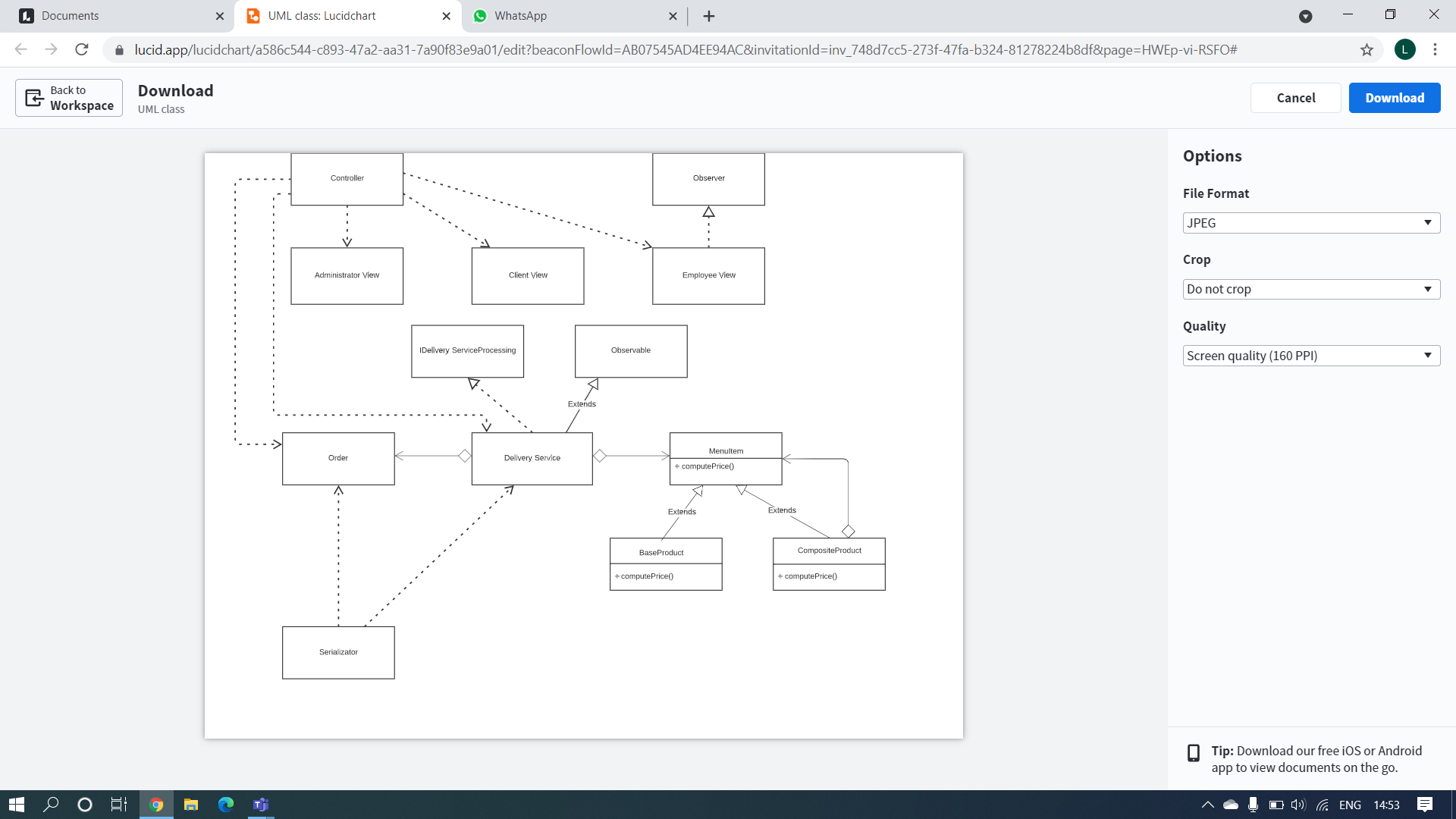
Așadar, este nevoie de trei modele pentru aplicație User (care conține câmpurile username, password și role) și Order (care ține evidența clientului care comandă și data și ora comenzii).

## Diagrama de clase

Organizarea claselor în pachete îl reprezintă nivelul al doilea de design. Aplicația este structurată pe nivele, așadar vor fi 3 pachete principale care se ocupă de buna funcționare. Pachetul pentru presentationLayer conține clasele care implementează interfețele grafice din cadrul aplicației și clasa care le controlează. Pachetul bussinessLayer conține clasele care modelează toate structurile necesare și toate operațiile efectuate. Pachetul dataLayer conține clasele care se ocupa de serializare și scriere în fișiere. Acest tip de structură oferă multe avantaje de proiectare, fiind ușor de întreținut.

Nivelul al treilea în design-ul unei aplicații îl reprezintă împărțirea pachetelor în clase. Acest lucru trebuie făcut după o analiză a necesităților și a resurselor. Prin urmare, pachetul presentationLayer conține următoarele clase: LoginView (fereastra de login și register), AdministratorView (interfața administratorului sistemului), ClientView (intefața utilizată de client să plaseze comenzi), EmployeeView (fereastra în care este notificat angajatul că s-a plasat o comandă nouă) și clasa Controller (care se ocupă de controlul întregii aplicații). Pachetul businesLayer conține următoarele clase: MenuItem (clasa care inglobează un produs simplu sau compus), clasa BaseProduct (care extinde MenuItem și reprezintă un obiect simplu), clasa CompositeProduct (care extinde MenuItem și reprezintă un produs compus din mai multe produse simple), interfața IDeliveryServiceProcessing (care conține toate metodele folosite la operațiile administratorului și clientului, neimplementate), clasa DeliveryService (care conține metodele din interfața IDeliveryServiceProcessing implementate), clasa Order (modelează o comandă) și clasa User (modelează un utilizator al aplicației). Pachetul dataLayer conține clasa Serializator care conține metodele de serializare și deserializare, pentru a ține datele salvate și la închiderea aplicației, clasa BillCompute care efectuează scrierea facturii unei comenzi și clasa RaportCompute care scrie rapoartele pentru administrator. Clasele adăugate suplimentar sunt cele pentru a notifica angajatul că s-a plasat o comandă. Se folosește Observer Design Pattern deci clasele necesare adăugate sunt Subject și Observer.

Prin urmare, clasele rezultate în urma analizei pot fi vizualizate într-o diagramă de clase:



## Algoritmi

După ce s-a realizat împărțirea în clase, design-ul problemei continuă cu analiza metodelor. Trebuie proiectați algoritmii folosiți în cadrul operațiilor făcute de către administrator și de către client. Principalii algoritmi sunt prezenți în clasa DeliveryService:

Algoritmul pentru importarea meniului constă în obținerea căii spre fișierul din care se extrag produsele, apoi filtrarea acestora cu stream-uri. Într-o colecție de tip Set se ia fiecare linie din fișier. Apoi din fiecare linie se iau câmpurile și se creează un obiect nou de tip MenuItem. Acest obiect va fi adaugat apoi în lista de produse din meniu.

Algoritmii de management al produselor de către administrator și anume adăugare, modificare, ștergere, constau în colectarea datelor de la utilizator și efectuearea operației dorite. La adăugare, dacă se adaugă un produs simplu, se scriu datele produsului în interfață, aplicația ia datele și inserează în tabel și în lista de produse noul produs. Dacă este produs compus, se selectează produsele care în alcătuiesc, se scrie in interfață numele noului produs și se adaugă la fel. Dacă se dorește modificarea, se selectează în tabel produsul de modificat și se scriu în interfață noile câmpuri. La ștergere, se selectează în tabel produsul de șters și se șterge prin buton.

Algoritmii de generare a rapoartelor pentru utilizator constau în filtrarea informațiilor și generarea unor fișiere text cu rapoartele. La primul raport se parcurg comenzile și se iau acele comenzi din intervalul orar dorit. La al doilea raport se parcurg comenzile și se iau acele comenzi care au fost comandate de mai multe ori decât numaărul dat în interfață. Toate rapoartele sunt efectuate cu ajutorul stream-urilor și lambda expression.

# Implementare

În urma analizei problemei și a implementării design-ului, au reieșit clasele descrise în secțiunea 4.2.

**Clasa Order**

Această clasă modelează comenzile puse de clienți și conțin numele clientului care a pus comanda și data și ora la care a fost efectuată. Clasa conține gettere și settere, constructor și metoda hashCode.

**Clasa User**

Această clasă conține datele despre utilizatori: conține variabilele instanță username, password și role. Pe baza rolului, la logare se deschide interfața potrivită.

**Clasa MenuItem**

Este clasa care înglobează un item din meniu care poate fi atât BaseProduct cât și CompositeProduct. Variabilele instanță sunt câmpurile unui produs din csv. O metoda importantă este metoda computePrice, care returnează prețul produsului.

**Clasa BaseProduct și CompositeProduct**

Clasa BaseProduct extinde clasa MenuItem, deci moștenește totul din MenuItem și nu suprascrie metoda computePrice. Clasa CompositeProduct moștenește și ea clasa MenuItem, dar suprascrie metoda computePrice care returnează suma prețurilor produselor simple care compun produsul compus.

**Interfața IDeliveryServiceProcessing**

În aceasă interfață este descris antetul metodelor principale care se ocupă de operațiile efectuate de client și administrator. Aceste motode sunt: importMenu, addProduct, modifyProduct, deleteProduct, filter, addOrder și metodele de generare de rapoarte.

**Clasa DeliveryService** comține implementarea metodelor din interfața IDeliveryServiceProcessing.

**Clasa BillCompute** este clasa în care se alcătuiește factura pentru fiecare comandă adaugată.

**Clasa RaportCompute** conține metodele de scriere a rapoartelor în fișiere text.

**Clasa Serializator** conține metodele pentru serializarea și desealizarea obiectelor din aplicație: comenzi, utilizatori, produse.

**Interfața Grafică**

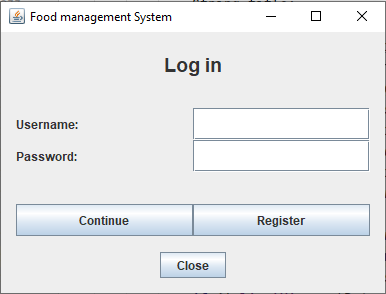
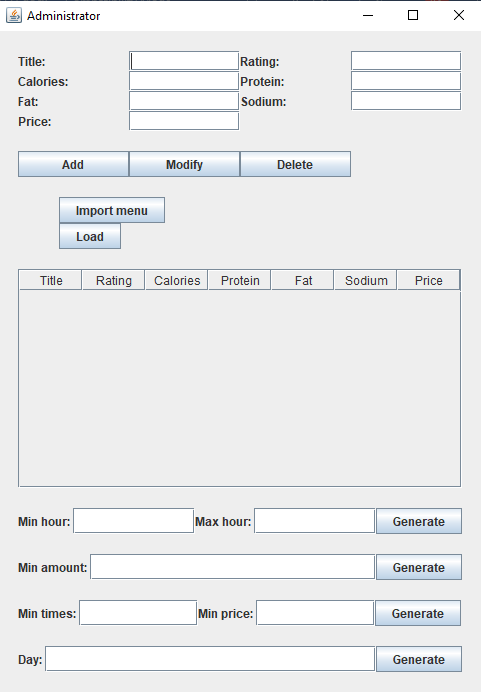
Pentru realizarea interfeței am folosit Java Swing. Clasele care o implementează sunt: clasa LoginView (interfața principală în care utilizatorii se loghează sau se înregistrează), clasa AdministratorView (interfața în care administratorul efectuează operațiile dorite), ClientView (interfața în care clientul efectuează comenzi) și EmployeeView (fereastra în care angajatul este notificat că s-a plasat o comandă).

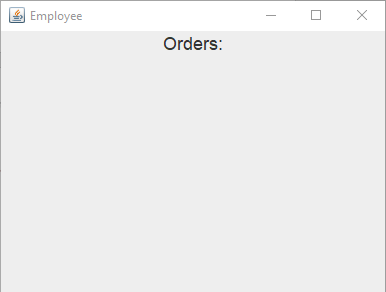
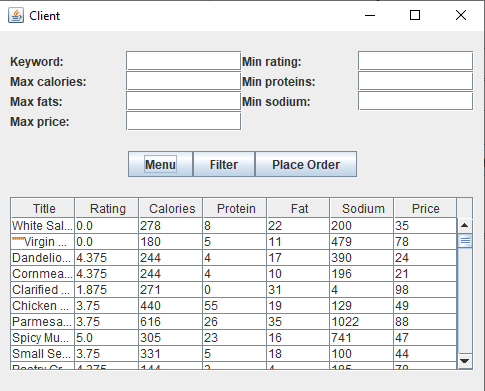
Fereastra principală (LoginView) conține câmpurile unde utilizatorul completează datele pentru logare și poate să aleagă să se logheze sau să se înregistreze în sistem. Dacă încearcă să se logheze și nu are cont, i se va cere să se înregistreze. După ce s-a înregistrat, acesta poate sa se logheze cu datele furnizate și poate efectua comenzi.

În interfața administratorului sunt toate componentele necesare pentru implementarea operațiilor necesare. Produsele sunt văzute într-un tabel. Adăugarea produselor se face prin completarea câmpurilor sau prin selectarea din tabel a produselor dorite. Modificarea se face prin selectarea produsului dorit pentru modificare și completarea datelor ce se doresc a fi schimbate, în interfață și se apasă butonul Modify. Ștergerea se face tot prin selectarea produsului dorit pentru șters și apăsarea butonului pentru Delete. Meniul se importa din fisierul csv prin apăsarea butonului Import. Datele aplicației se deserializează prin intermediul butonului Load din interfață. Generarea rapoartelor se face într-un mod ușor în interfață, completând câmpurile necesare. La apăsarea butonului pentru generarea raportului, se va crea un fișier text în care este completat raportul.

În interfața clietului sunt componente prin care se efectuează operațiile necesare. Meniul se poate vedea prin apăsarea butonului Menu. Adăugarea unei comenzi se face prin selectarea produselor dorite din tabel și apăsarea butonului Place Order. La plasarea comenzii, se va genera un fișier text cu factura pentru comandă care conține clientul, produsele comandate și prețul total al produselor. Prin completarea câmpurilor dorite se pot filtra produsele din meniu, pe baza a mai multor criterii.

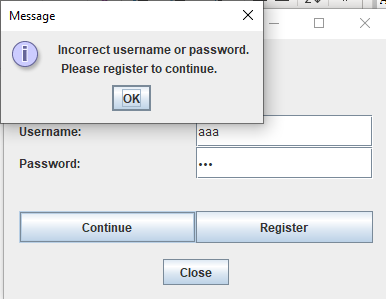
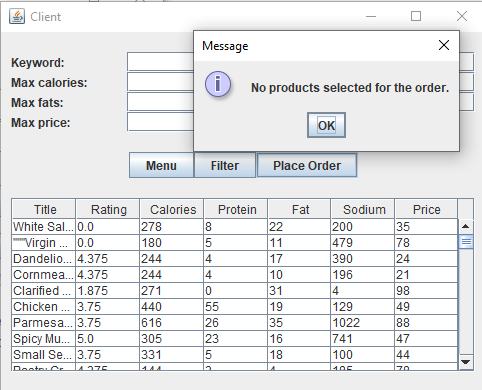
Ferestrele aplicației:

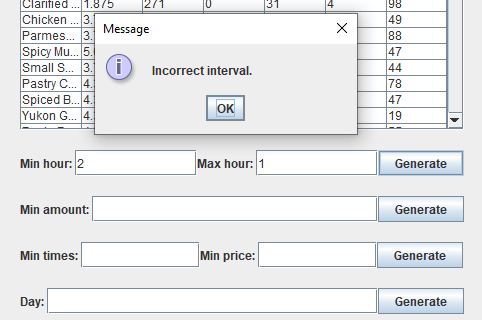
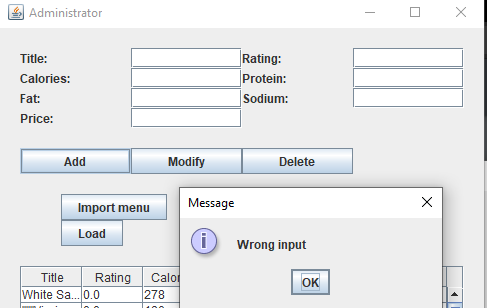
 

# Testare

Am testat aplicația cu mai multe seturi de intarare și s-au verificat și anumite excepții:

# Concluzii si Dezvoltări Ulterioare

În concluzie, înainte de implementarea propriu zisă a unei aplicații, e nevoie de o analiză în detaliu și de luarea unor decizii cu privire la structură și la modelul ales, astfel încât să fie cel mai potrivit cerințelor. Acest lucru conferă o implementare mai ușoară, mai logică, mai organizată și cu mai puțin efort. Poate fi un avantaj și pentru client/utilizator dacă aplicația e mai intuitivă, organizată și atractivă.

Ca dezvoltare ulterioară, utilizatorul poate să facă mai multe rapoarte, să ofere mai multe facilități, să aibă o interfață mai complexă (ex. afișare mai interactivă, culori, buton de back din ferestre)

# Bibliografie

1. <http://tynerblain.com/blog/2007/04/09/sample-use-case-example/>
2. <https://lucid.app/>
3. ASSIGNMENT\_4\_SUPPORT\_PRESENTATION.pdf
4. <http://www.tutorialspoint.com/java/java_serialization.htm>
5. <https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>