Universitatea Tehnica din Cluj-Napoca

Facultatea de Automatica si Calculatoare

Calculatoare

Tehnici de Programare



Food Delivery Management System

Student: Balog Helg

Grupa: 30227

Cuprins

[1.Cerințe Funcționale 2](#_Toc1272178168)

[2.Obiective 2](#_Toc456410600)

[1.Obiective principale 2](#_Toc339347870)

[2.Obicetive Secundare 2](#_Toc1413033578)

[3.Analiza Problemei 3](#_Toc229504159)

[4.Proiectare și Implementare 3](#_Toc996163185)

[4.1 Proiectare 3](#_Toc1377413651)

[4.2 Pachete și Clase 4](#_Toc177343409)

[5.Funcționalitate 10](#_Toc816216496)

[6.Concluzii și Dezvoltări Ulterioare 11](#_Toc825554085)

[7.Bibliografie 11](#_Toc1652735509)

# 1.Cerințe Funcționale

Proiectați și implementați o aplicație care se ocupă de managementul unei companii de catering. Aplicația trebuie să comunice cu utilizatorul prin interfețe grafice. La primul pas, utilizatorul trebuie să se logheze pe aplicație și în funcție de rolul acestuia, administrator, angajat, client, se va afișa interfața corespunzătoare. Dacă clientul nu are cont, se poate înscrie. Dacă utilizatorul este administrator, atunci acesta poate efectua următoarele operații: să importe meniul inițial, să afișeze meniul, să adauge/editeze/șteargă anumite produse și să genereze patru tipuri de rapoarte. Dacă utilizatorul este un client, atunci acesta poate să vizualizeze meniul, să caute în meniu după anumite criterii și să plaseze o comandă. Angajatul trebuie să primească o notificare tot timpul când o comandă nouă este plasată.

La al doilea pas, aplicația trebuie să urmeze anumite metode de implementare, cum ar fi: să folosească expresii lambda și stream-uri, să se folsoească de Composite și Observer Design Pattern, să serializeze anumite informații, etc.

Aplicația trebuie să ofere o funcționalitate eficientă și corectă.

# 2.Obiective

## 1.Obiective principale

Obiectivul principal al aplicației este ca un utilizator să se poată loga și în funcție de rolul acestuia să efectueze anumite acțiuni, cum ar fi: să vizualizeze meniul, să plaseze o comandă, să adauge produse, să genereze rapoarte, etc,

## 2.Obicetive Secundare

Pentru a atinge obiectivul principal, trebuie să parcurgem anumite obiective secundare, precum: să importăm produsele inițiale, să implementăm acțiunile ce se pot efectua de utilizatori, să salvăm informațiile despre utilizatori, produse și despre comenzi, astfel încât la sfârșitul execuției acestea să nu se piardă. Să creăm interfețele grafice corespunzătoare pentru fiecare utilizator.

# 3.Analiza Problemei

Acest proiect susține următoarea scenă de utilizare, în funcție de rolul utilizatorului:

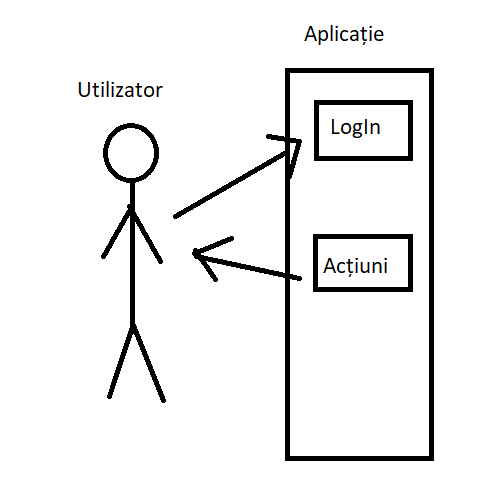
-administrator: se loghează și mai departe poate efectua operațiile menționate mai sus;

-client: se loghează și mai departe poate efectua operațiile de mai sus;

-angajatul, după logare, va primi notificările legate de comenzi.

Ca și client, acesta va primi confirmarea trimiteri comenzii , dacă plasează una, altfel poate vizualiza meniul. Administratorul va primi anumite informații despre comenzile făcute până în acel moment și va putea modifica meniul. Angajatul va pregăti comenzile.

În imaginea de mai jos se poate observa diagrama de utilizare.



# 4.Proiectare și Implementare

## 4.1 Proiectare

Aplicația ‘Food Delivery Management System’, descris în limbajul java, are o arhitectură de tip Layered, adică are trei straturi semnificative. Primul strat, cel mai e sus, este cel de vizualizare, care se ocupă de realizarea interfețelor grafice și de comunicarea dintre interfețe cu restul proiectului.

Al doilea strat este cel de logică, adică Controllerul va transmite datele citite la acest strat unde se va decide ce se face mai departe și cum. De exemplu: la crearea unui cont nou pentru un utilizator se vor lua datele înscrise și se vor salva, pentru a lucra cu acestea mai târziu.

Al treilea strat, cel mai de jos, se ocupă cu accesarea datelor din lista pentru produse, comenzi și pentru utilizatori. Aceste informații se salvează într-un fișier, ca la sfârșitul execuției programului să nu se piardă datele utilizate.

## 4.2 Pachete și Clase

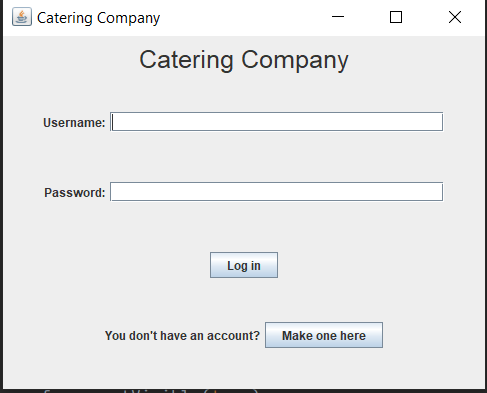
Pachetul cateringComapny

Acest pachet conține o singură clasă, Main,cu metoda main, din care pornește execuția programului. Aici se instanțiază un obiect de tipul Controller.

Pachetul presentation

Acest pachet reprezintă primul strat din arhitectura de tip Layered al aplicației, astfel , cum s-a precizat și mai sus , el se ocupă de interfețele grafice ale programului și de comunicarea dintre interfețe si program.

**Clasa View**, implementează interfața prin care utilizatorul se loghează. Puteți observa interfața, în imaginea de mai jos:

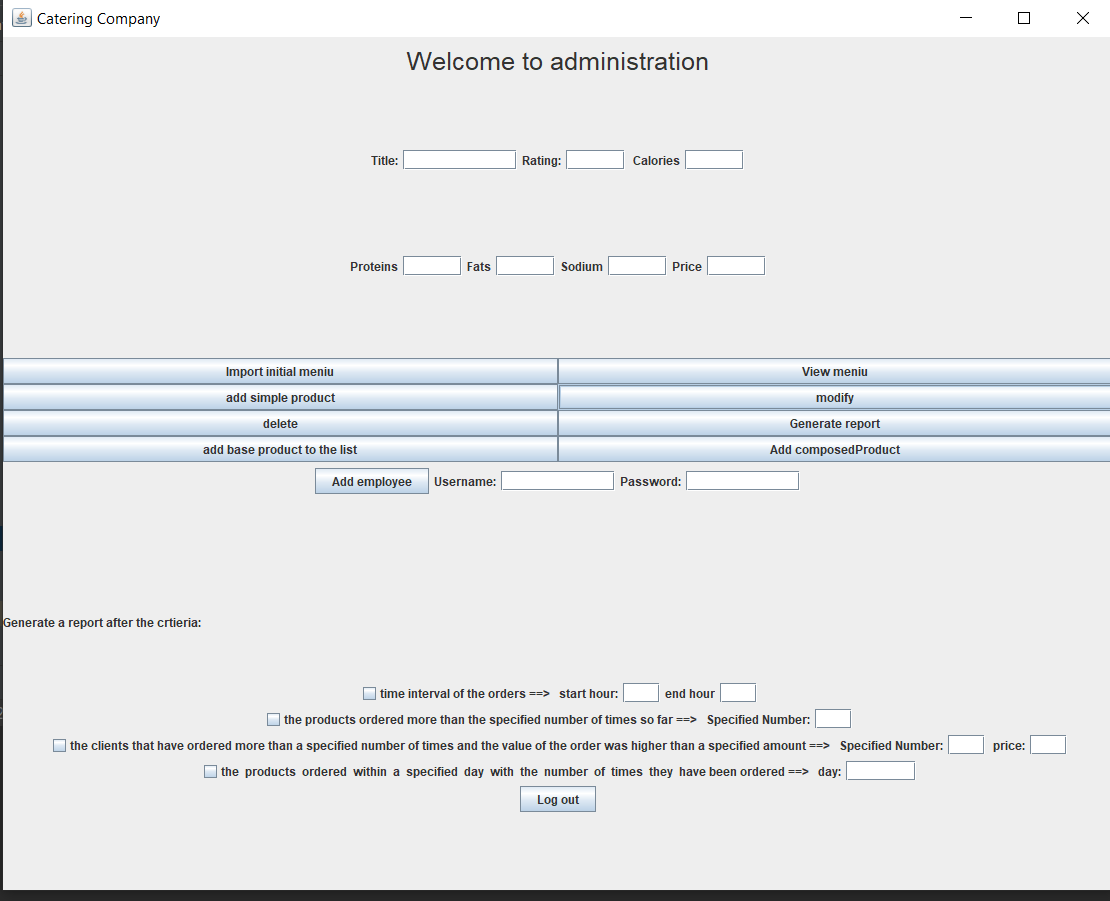


Utilizatorul trebuie să își scrie username-ul și parola, după care să apese pe buton de ‘Log in’. Dacă utilizatorul există în datele aplicației, se va afișa mai departe, interfața corespunzătoare, altfel se va afișa un mesaj de avertizare despre inexistența acelui cont.

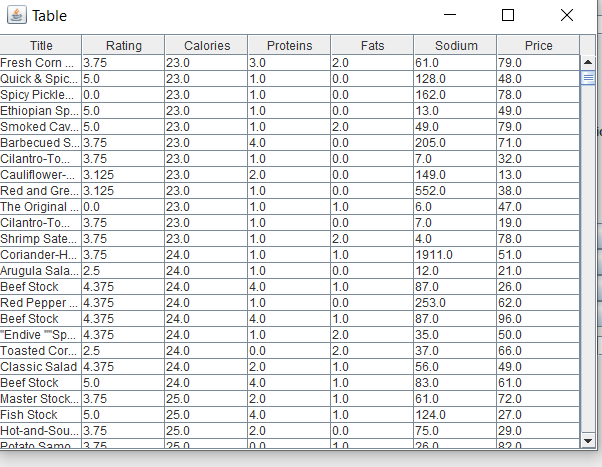
În imaginea următoare puteți observa un astfel de mesaj de avertizare. Clientul pe

numele Albert nu are cont și este sfătuit să își creeze unul prin apăsarea butonului de ‘Make one here’.

**Clasa AdministratorView** implementează interfața pentru administrator. În următoarea imagine se poate vizualiza aspectul acesteia.



În primul rând, dacă se apasă buton de ‘Import initial menu’ se va importa din fișierul products.csv meniul inițial. Însă acest pas nu mai este necesar , deoarece aceste produse s-au salvat deja într-un fișier. La apăsarea butonul de ‘View Menu’ se va afișa sub forma unui tabel meniul companiei. În următoarea imagine se poate observa o parte din acest meniu.



Dacă se apasă buton de ‘add base product’ se vor lua valorile înscrise mai sus și se vor adăuga în lista de produse, adică în meniu. Este foarte important ca datele introduse să fie corecte, de exemplu titlul să fie un șir de caractere, numerele să fie pozitive și nenule, deoarece în caz contrat nu se va efectua inserarea.

Dacă se selectează un rând din tabel și se apasă pe butonul de ștergere, se va șterge acel produs din meniu.

La fel, dacă se selectează un produs, și se apasă de data aceasta butonul de editare, se vor modifica câmpurile produsului cu datele înscrise. În cazul acesta nu este necesar ca toate câmpurile să fie nenule, ca și la adăugare, ci doar cele pe care dorim să le modificăm trebuie să fie corecte.

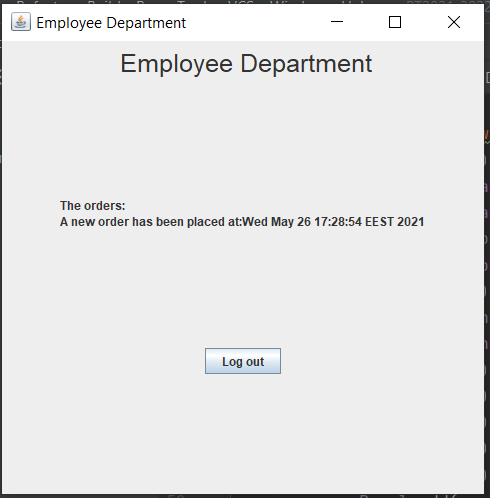
Dacă se apasă buton de ‘add base product to list’ înseamnă că se creează o listă de produse simple, pentru un produs compus, precum Meniul zilei de azi, etc. După ce se creează această listă, trebuie să se apese buton de ‘add composed product’, altfel nu se va crea.

În al doilea rând, deoarece suntem la interfața pentru administrator, putem genera și patru tipuri de rapoarte. Tot ce trebuie să facem este să bifăm raportul pe care îl dorim, să completăm corect câmpurile necesare și să apăsăm pe butonul de generare. Se va afișa un mesaj în cazul în care am avut succes cu crearea raportului.

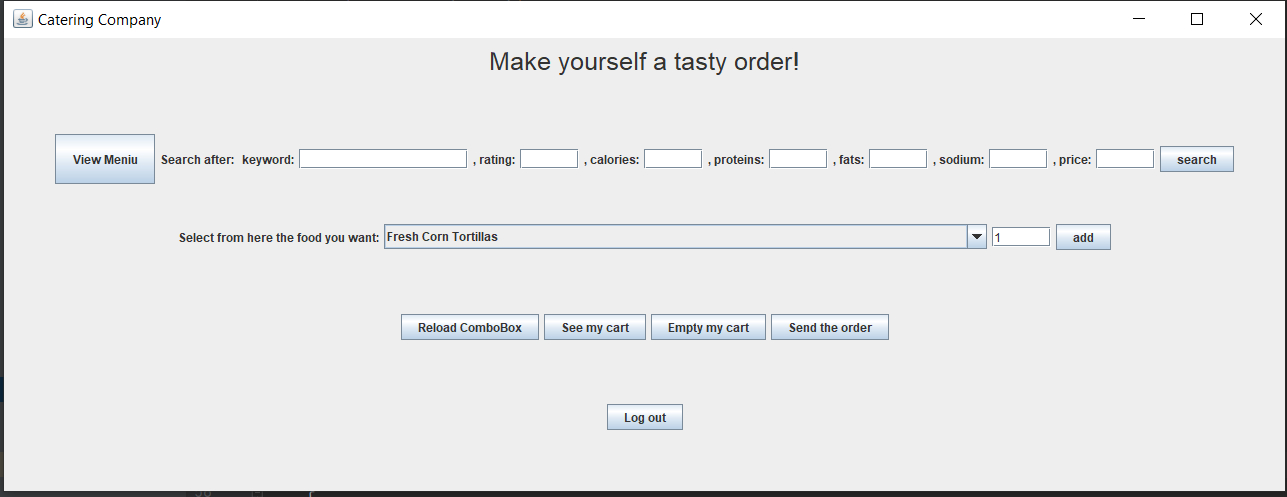
Pentru ca administratorul să mai adauge angajați, trebuie să completeze numele și parola acestora și să apese pe butonul de adăugare. Aceasta este singura metoda prin care putem

adăuga angajați.

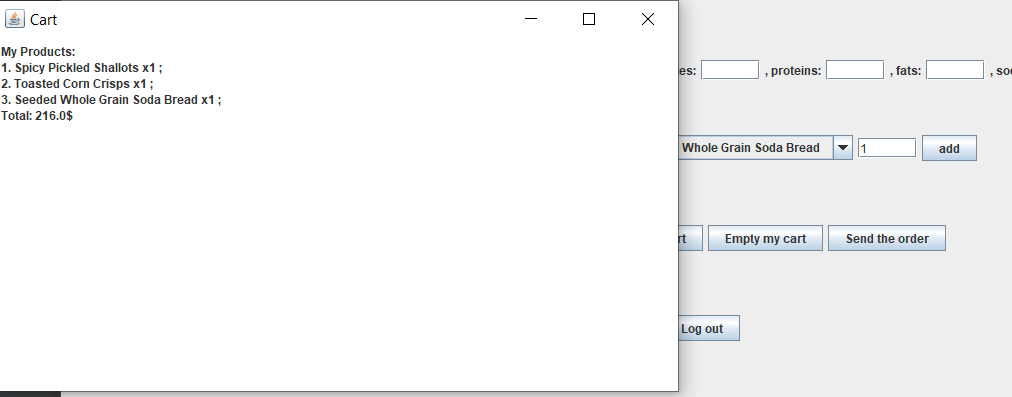
**Clasa EmployeeView** implementează interfața pentru angajați. Această interfață nu conține altceva decât comenzile care s-au plasat ulterior. În imaginea de mai jos, este reprezentată interfața pentru angajați cu o comandă nouă.



**Clasa ClientView** implementează interfața pentru clienți. Aceasta, la fel ca și interfața pentru administrator, are mai multe funcționalități.



Clientul poate vizualiza meniul, dacă apasă pe butonul de ‘View Menu’. Pentru a căuta un produs, sau mai multe produse, pe baza unor criterii, completăm câmpurile după care vrem să ne orientăm și apăsăm butonul de căutare. Rezultatele se vor afișa în comboBoxul de mai jos, de unde se poate alege produsul, cantitatea și la final se poate adăuga în coș. Coșul este de forma prezentată în imaginea de mai jos, unde se înșiruie produsele, cantitățile și suma totală de plătit. Dacă clientul este satisfăcut, se poate plasa comanda, altfel se poate goli coșul de cumpărături.



De fiecare dată când se plasează o nouă comandă, angajații sunt notificați pentru a putea pregăti comanda. Acest fapt se realizează datorită implementării de Observer Design Pattern. EmployeeView extinde **clasa Observer**, care lucrează și cu **clasa Subject**. Aceste trei clase formează pattern-ul de observer.

**Clasa Controller** specifică ce acțiune să se execute în funcție de butonul apăsat. Această clasă face legătura cu interfețele aplicației și programul din spate. El decide dacă programul poate să continue cu datele inserate, sau dacă utilizatorul trebuie avertizat în legătură cu ceva/

Pachetul businessLogic

Acest pachet reprezintă stratul din mijloc din arhitectura folosită și se ocupă de logica din spatele acțiunilor.

**Clasa User** reprezintă utilizatorii. Aceasta conține un nume, o parolă și un rol. Ulterior, vom avea o listă cu acești utilizatori. Această listă este esențială, din mai multe puncte de vedere. În primul rând, când se loghează un utilizator nou, se verifică dacă acesta apare în listă, și dacă da, cu ce rol. Altfel se va adăuga în listă. În al doilea rând, datele acestei liste, sunt salvate tot timpul, astfel un cont deja creat nu se va pierde niciodată.

**Clasa Order** reprezintă comenzile. Această clasă conține numele clientului, id-ul comenzii, data în care s-a plasat comanda și suma totală de plătit. La fel ca și la clasa de utilizatori, vom avea o lista de comenzi, tot importantă.

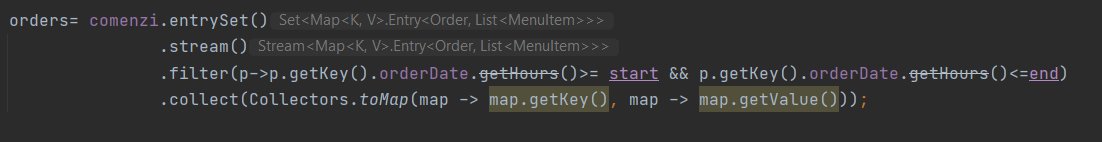
**Clasa BaseProduct** și **CompositeProduct** extind  **clasa MenuItem.** Aceste trei clase respectă principiile impuse de Composite Design Pattern. Ulterior, o să avem o listă de produse, în care o să avem atât produse de bază cât și produse compuse. În momentul în care administratorul adaugă un produs nou, indiferent de ce fel, acesta se va adăuga în această listă. La fel, căutarea/ generarea rapoartelor/ ștergerea/editarea se vor executa pe seama acestei liste.

**Clasa ItemsAndQuantities** este o clasă ajutătoare, de care programul se folosește la completarea coșului de cumpărături. Aceasta conține un produs și cantitatea acesteia.

**Interfața IDeliveryServiceProcessing** declară metodele care trebuie implementate de calsa care o să o extindă.

**Clasa DeliveryService** este clasa principală din acest substrat din arhitectură. Ea extinde interfața menționată mai sus și se ocupă cu logica din spatele sistemului. Clasa are o listă de utilizatori, de produse și de comenzi. Pe baza acestora se efectuează operațiile posibile pentru administrator și clienți, cum ar fi: adăugarea de produse, ștergerea de produse, editarea, generarea rapoartelor, căutarea în lista de produse în funcție de anumite criterii, plasarea comenzii și alte multe funcții e care se folosește aplicația. Această clasă, de asemenea, generează o factură cu numele clientului și cu totalul de plată la plasarea unei noi comenzi.

Unele operații dintre cele menționate mai sus, se implementează folosind teoria expresiilor lambda și a stream-urilor. De exemplu la generarea primului timp de raport:



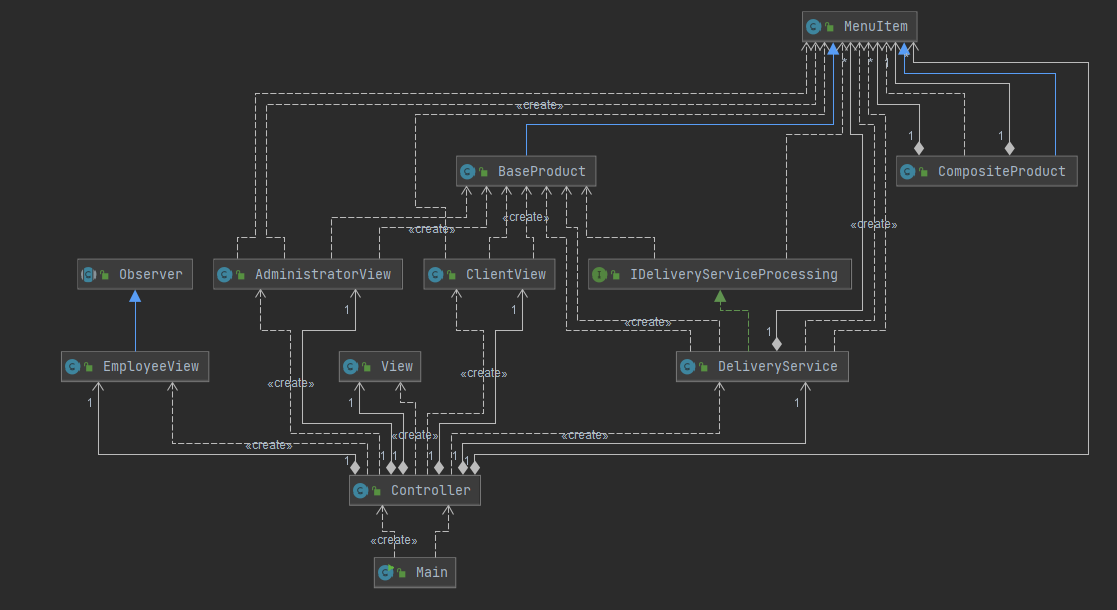
Orders este lista rezultată după filtrarea comenzilor. Se poate observa cum se ia fiecare comandă și se verifică dacă ora la care s-a plasat se situează între cele două limite specificate de administrator. La final, aceste comenzi se scriu într-un raport de tip text.

Pachetul Data

Acest pachet corespunde cu cel mai de jos strat din arhitectura folosită. Astfel acesta se ocupă de salvarea datelor la finalul execuției și la citirea datelor la începutul execuției programului.

**Clasa Serializator** are șase metode. Trei pentru citirea datelor, lista de utilizatori, lista de produse și lista de comenzi, și trei metode pentru scrierea datelor din listele menționate.

În controller, la începutul programului se citesc datele necesare în liste , iar în momentul în care se închide fereastra se scriu datele în fișier. Este de menționat că datele se țin în trei fișiere diferite, un fișier text pentru fiecare listă.



# 5.Funcționalitate

**Administrator:** se loghează pe aplicație după care își efectuează sarcinile: pe baza rapoartelor, adaugă sau șterge produse și eventual adaugă angajați

**Angajat:** se loghează și își verifică comenzile pe care le are de pregătit.

**Client:** în cazul în care nu are cont, își creează unul și intră pe acesta, după care caută produsele de care este interesat și le adaugă în coșul de cumpărături. La final plasează comanda si așteaptă ca angajații să o pregătească.

# 6.Concluzii și Dezvoltări Ulterioare

În concluzie, aplicația Food Delivery Mangement System îsi atinge atât obiectivele secundare cât și pe cele principale. Programul oferă o funcționalitate corectă și eficientă în acest sens.

Ca și dezvoltări ulterioare, s-ar putea adăuga o verificare la apăsarea butonului de importare a produselor, astfel încât dacă acele produse există deja, să nu se mai adauge.

La fel s-ar putea extinde interfața pentru angajați, astfel încât aceștia să bifeze comenzile pregătite. Acest lucru trebuie sincronizat deoarece mai mulți angajați pot pregăti aceeași comandă, ceea ce nu este corect. Un angajat trebuie să pregătească o singură comandă pe rând, iar o comandă trebuie să fie pregătită de un singur angajat.

S-ar putea adăuga testări pentru a verifica dacă suma totală a comenzii depășește o sumă impusă, de la care se oferă reduceri sau alte oferte.

O dezvoltare ulterioară necesară ar fi ca utilizatorii în cazul în care își pierd parola, să aibă posibilitatea de a o recupera.

# 7.Bibliografie

1.https://www.baeldung.com/java-collectors-tomap

2. <https://stackoverflow.com/questions/43890986/java-8-lambda-expression-for-finding-contains/43891099>

3. <https://www.baeldung.com/java-collectors-tomap>

4. <https://en.wikipedia.org/wiki/Composite_pattern>

5. <https://www.tutorialspoint.com/design_pattern/observer_pattern.htm>

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

7. <https://stackoverflow.com/questions/2677564/how-to-create-custom-javadoc-tags>

8. <https://docs.oracle.com/cd/E19683-01/806-7930/assert-13/index.html>