Logo

Description automatically generated

FOOD DELIVERY MANAGEMENT

SYSTEM

Facultatea de Automatică și Calculatoare

Departamentul Calculatoare

Curs: Tehnici de Programare Fundamentale

An universitar 2020-2021

Student:

Bîrluțiu Claudiu-Andrei

CTI-ro, An 2, Grupa 30226

Cuprins

[1.Problema și soluția dată 3](#_Toc70036932)

[2.Obiectivul temei 3](#_Toc70036933)

[3. Analiza problemei, modelare, scenarii, cazuri de utilizare 4](#_Toc70036934)

[4.Proiectare 8](#_Toc70036935)

[5.Implementare 12](#_Toc70036936)

[6.Rezultate 18](#_Toc70036937)

[7.Concluzii 19](#_Toc70036938)

[8.Bibliografie 19](#_Toc70036939)

ss

1.Problema și soluția dată

Având în vedere situația pandemică (2020-2021) cu restricțiile impuse cu scopul opririi răspândirii virusului, restaurantele au trebuit să se adapteze noilor condiții, ce necesită o mai bună gestiune a sistemului de catering.

Firmele de catering au reprezentat o alternativă atractivă, înregistrând câștiguri considerabile. Odată cu creșterea numărului de comenzi, este tot mai greu să se gestioneze o astfel de firmă fără existența unei aplicații care să ofere suport necesar atât staff-ului (administrator și angajați) cât și clienților care vor să consulte meniul și să plaseze comenzi.

În aceste condiții, o aplicație cu interfață intuitivă și prietenoasă ar înlesni munca angajaților și ar fluidiza procesul de delivery, astfel încât imediat după selecția produselor dorite și apăsarea butonului de finalizare comandă, angajații firmei să primească informațiile despre comandă și să o pregătească. De asemenea, aplicația va permite și modificări la nivelul meniului, operație ce va fi executată de administratorul restaurantului.

2.Obiectivul temei

Obiectivul principal se referă la proiectarea și implementarea unei aplicații de management a unui restaurant/firme de catering. Acesta va permite gestiunea comenzilor și a meniului pus la dispoziție clienților prin intermediul unei interfețe grafice. Accesul la anumite operații se va face prin logare cu username și parola. Vor exista 3 tipuri de useri, fiecare având acces la operațiuni specifice. Interfața grafică va permite userilor de tip administrator să importe dintr-un fișier .csv produsele de bază, va permite adăugare de produse de bază în meniu și de asemenea crearea de noi produse din cele existente și se vor face disponibile clienților. Administratorul poate să ceară prin intermediul interfeței și a criterilor pe care le introduce rapoarte despre produsele cele mai comandate, despre clienții fideli sau să obțină rapoarte la nivel de zi, interval orar. Userii de tip client pot să își creeze un cont nou prin introducerea unui username si parola, iar apoi se vor conecta cu acestea ori de cate ori vor. Aceștia pot vizualiza meniul cu mâncărurile disponibile și informațiile despre acestea din punct de vedere caloric, al aportului de sodiu etc., apoi pot să selecteze produsele dorite și să plaseze o comandă. Cel de-al treilea tip de user, angajatul va avea acces la o pagină unde îi sunt afișate comenzile pe care trebuie să le proceseze în ordinea plasării lor.

Obiectivele secundare, ce reprezintă pașii de urmat în îndeplinirea obiectivului principal, se regăsesc în următorul tabel. Acestea vor fi abordate în amănunt în următoarele capitole ale acestui document.

|  |  |  |
| --- | --- | --- |
| **Obiectiv** | **Scurtă descriere** | **Capitol** |
| Analiza problemei și identificarea cerințelor | -analiza se referă la descompunerea problemei în componente mai mici prin procesul de abstractizare.  -se poate vorbi și de o inginerie a cerințelor care stabilește serviciile cerute sistemului precum și constrângerile. | 3. Analiza problemei, modelare, scenarii, cazuri de utilizare |
| Proiectarea sistemului de management al firmei de catering | -obținerea unui sistem bun prin utilizarea unor resurse existente și noi resurse.  -reprezintă o activitate de tipul divide-and-conquer.  -presupune folosirea paradigmei OOP, realizarea de diagrame UML, proiectarea claselor etc. | 4. Proiectare |
| Implementarea sistemului de management al firmei de catering | -presupune dezvoltarea și scrierea codului în limbajul de programare Java pentru clasele și metodele acestora. | 5. Implementare |
| Testarea sistemului de management al firmei de catering | -realizarea unor operații prin intermediul interfeței și observarea modificărilor apărute în consecința la nivelul tabelelor cu produsele din meniu, comenzile de procesat din interfața angajaturlui | 6. Rezultate |

Tabel 1-Obiective secundare

3. Analiza problemei, modelare, scenarii, cazuri de utilizare

* 1. **Analiza problemei**

Este important să înțelegem problema și domeniul ei. Procesul de analiză are drept scop descompunerea problemei mari în componente mici pe care să le înțelegem prin asocieri din lumea reală.

Domeniul problemei prezente se referă la gestiunea unui restaurant de tip catering, care are nevoie de o aplicație prin care să se realizeze operațiile necesare fluidizării procesului de comandare.

Prin log in se înțelegere cererea conexiunii la o pagina autorizată prin intermediul unui username și o parolă unice.

Prin register se înțelege înregistrarea unui nou utilizator caracterizat prin cele două câmpuri unice din baza de date/memoria aplicației, username-ul și parola.

Pentru realizarea aplicației este de avut în vedere faptul că datele trebuie salvate la închiderea aplicației și se vor încărca la următoarele folosiri ale aplicației. Prin aceste operațiuni, memoria sistemului de management este mereu la zi. Metodele folosite sunt cele de serializare și deserializare a obiectelor.

Prin serializare se înțelege transformarea unui obiect într-o secvență de octeți, din care să poată fi refăcut ulterior obiectul original.

Procesul invers, de citire a unui obiect serializat pentru a-i reface starea originală, se numește deserializare.

Pentru unicitatea produselor și a userilor, s-au folosit structuri de date de tip hash table. Aceste structuri dau și un timp de căutare/acces destul de mic fapt îmbucurător având în vedere faptul că se fac căutări frecvente (căutarea de produse in meniu pe anumite criterii, verificarea existenței în memorie a user-ului cu care se încearcă conexiunea).

În ceea ce privește aspectul interfeței sistemului de management, interacțiunea cu utilizatorul trebuie sa fie cât mai prietenoasă și intuitivă, mai ales în cadrul ferestrei dedicate clienților, care trebuie în primă fază atrași vizual și de asemeanea procesul de comandare a mâncării să fie cât mai scurt și ușor de realizat. De asemenea, utilizatorii trebuie să fie atenționați în momentul în care introduc date invalide și să fie informați de greșelile pe care le-au făcut în timpul operației efectuate.

* 1. **Modelare**

După determinarea domeniului problemei, datele și procesele sunt translatate prin abstractizare în domeniul soluției ducând la modele de reprezentare, algoritmi și programe.

Folosind modelarea OOP se poate facilita alinierea obiectelor din lumea reală cu obiectele software, deoarece obiectele cu relațiile și interacțiunile din lumea reală relaționează și interacționează similar și într-o aplicație software. Clasele din implementare pot fi inspirate din domeniul problemei (nume similare, relații și responsabilități similare) ceea ce va asigura o mai bună înțelegere a designului și a implementării.

În cadrul sistemului de management al firmei de delivery s-au creat clase cu denumiri specifice: *ClientView*, *MenuItem, Order, User, EmployeeView, AdministratorView* . Relațiile dintre aceste clase s-au corelat cu situația din lumea reală. În cadrul aplicației, clasa Order conține o listă de produse și id-ul clientului care le-a comandat. Prețul total al comenzii se calculează ca sumă a preturilor tuturor produselor din comandă.

Denumirea sugestivă a claselor facilitează înțelegerea lor și definirea atributelor corespunzătoare.

* 1. **Scenarii și cazuri de utilizare**

Identificarea cerințelor și setarea lor clară sunt esențiale și trebuie realizate riguros înainte de proiectarea și implementare programului software de față. Se poate vorbi chiar de o inginerie a cerințelor care se ocupă cu stabilirea serviciilor cerute sistemului și a constrângerilor( acest proces este unul ciclic). În continuare mă voi axa pe două etape ale acestui proces: elicitarea cerințelor și specificarea cerințelor în cadrul temei de laborator.

*Elicitarea cerințelor*: descoperirea cerințelor sistemului: activități precum interviuri și analiză bazată pe scenarii.

*Specificarea cerințelor*: implică modelarea riguroasă a cerințelor: use-case, diagrame UML

Din enunțul problemei reies următoarele cerințe pe care trebuie să le îndeplinească sistemul:

1. Existența unei interfețe grafice prin care administratorul poate să importe produsele de bază dintr-un fișier .csv, să modifice sau să adauge noi produse, fie ele de bază sau compuse ( „Meniul zilei” de exemplu), să obțină rapoarte cu privire la comenzile procesate (clienți fideli, produse cumpărăte într-o anumită zi etc. )
2. Existența unei interfețe grafice prin un utilizator de tip client să consulte meniul, să poată căuta anumite mâncăruri după anumite criterii (număr calorii, cantitate sodiu , preț) și să poată să creeze o comandă.
3. Existența unei interfețe grafice pentru angajat, unde acesta poate să observe noile comenzi ce trebuie procesate.
4. Aplicația trebuie să conțină mecanismul de logare a diferiților utilizatori la paginile destinate operațiilor specifice.
5. Aplicația trebuie să mențină consistența datelor chiar și după închiderea ei.

În următorul tabel sunt definite cerințele sistemului:

|  |  |
| --- | --- |
| Cerințe | |
| Cerințe funcționale  (descriu ce trebuie să facă sistemul) | * Sistemul de management permite accesul a 3 tipuri de utilizatori: administrator, client și angajat * Sistemul de management permite utilizatorului să importe produse dintr-un fișier .csv la primare rulare a aplicației * Sistemul de management permite administratorului să adauge sau să modifice anumite caracteristici ale unui produs, fie el de baza sau compus * Sistemul de management permite administratorului să șteargă un produs din meniul restaurantului * Sistemul de management permite administratorului să genereze rapoarte pe baza comenzilor procesate. (raport cu privire la comenzile dintr-un interval orar, produsele frecvent cumpărate, clienți fideli -care au comandat de mai multe ori peste o sumă dată ca referință, de asemenea se poate obține și un raport cu produsele cumpărate într-o anumită zi) * Sistemul de management permite clientului să vizualizeze întregul meniu al restaurantului, sub forma unui tabel pentru produsele de bază, unde poate vedea și caracteristicele nutriționale (proteine, calorii, grăsimi etc. ),precum și prețul fiecăruia. De asemenea, poate să vizualizeze și o listă cu produsele compuse (de exemplu meniul zilei). * Aplicația îi permite clientului să realizeze căutări pe mai multe criterii astfel încât acesta să poată vizualiza doar produsele astfel filtrate. * Sistemul de management permite clientului să creeze o comandă nouă după ce a selectat produsele dorite. * Sistemul de management al unui depozit angajaților să vizualizeze comenzile ce trebuie procesate în ordinea plasării lor. * Aplicația permite înregistrarea unui nou client prin introducerea unui username și a unei parole valide. * Aplicația va salva actualizările la noile lansări ale acesteia cu scopul păstrării consistenței datelor. |
| Cerințe non-funcționale  (descriu cu ce calitate ar trebui să funcționeze sistemul) | * Sistemul de management al restaurantului să fie intuitiv și ușor de folosit * Semnalizarea unei nereguli la introducerea de date sau la conectarea unui user, să se facă într-un fel cât mai prietenos și ușor de identificat de către utilizator. Se vor avea în vedere restricțiile la nivelul dimensiunii username-ului și parolei, ale datelor introduse pentru noile produse sau pentru cele existente, precum și pentru criteriile rapoartelor |
| Constrângeri  (descriu limitele de dezvoltare a sistemului) | * La un moment dat poate fi conectat doar un user. Nu avem sincronizare intre operatații |

Tabel Definirea cerințelor sistemului

**Use-case UML** capturează grafic actorii sistemului, cazurile de utilizare și relațiile dintre acestea.

Actorii- orice tip de utilizator

Diagram

Description automatically generated

Figure 1 Use-Case UML

Cazurile de utilizare ale aplicației:

Înainte de a face anumite operații este nevoie ca utilizatorii să se conecteze prin intermediul username-ului și a aprolei unice.

**Use case**: conectare

**Actor principal**: utilizator

**Principalul scenariu de succes**:

1. Utilizatorul selectează din fereastra dedicată logării tipul său: Client, Administrator sau Angajat
2. Utilizatorul introduce username-ul și parola
3. Utilizatorul apasă butonul *Log in* din fereastra ClientWindow

**Secvențe secundare**

1. Utilizatorul introduce unul sau ambele câmpuri greșit sau nu are un cont pentru accesul la aplicația de food delivery. Va fi atenționat printr-un mesaj de informare și va putea să facă un număr nelimitat de încercări.

**Use case**: înregistrare client

**Actor principal**: utilizator

**Principalul scenariu de succes**:

1. Utilizatorul selectează din fereastra dedicată logării tipul *Client*
2. Utilizatorul introduce un username și o parolă validă sau nefolosite
3. Utilizatorul apasă pe butonul *Register Client*

**Secvențe secundare**

* Utilizatorul introduce un username folosit. Va primi un mesaj de informare și poate introduce un alt username.
* Utilizatorul nu respectă condițiile pentru dimensiunea celor două câmpuri. În consecință va fi informat printr-un mesaj și poate introduce alte valori.

**Use case**: importare produse din fișierul .csv

**Actor principal**: administrator

**Principalul scenariu de succes**:

1. Administratorul selectează tab-ul *Import products*
2. Administratorul apasă pe butonul *Import products*
3. Administratorul va primi un mesaj de informare cu privire la import: în cazul în care au fost deja importate produsele nu se va mai face import-ul. Admin-ul va fi informat în acest caz.

**Secvențe secundare**

**Use case**: adăugare produs bază

**Actor principal**: administrator

**Principalul scenariu de succes**:

1. Administratorul selectează tab-ul *Add product*
2. Administratorul introduce informațiile despre produsul de bază în câmpurile puse la dispoziție
3. Administratorul apasă pe butonul *Add new product*
4. Administratorul va primi un mesaj de informare

**Secvențe secundare**

* Administratorul introduce date invalide și va fi informat asupra acestui aspect printr-un mesaj. El poate să modifice datele care nu sunt în conformitate cu tipurile de informații necesare pentru un produs de bază.

**Use case**: modificare/ștergere produs bază

**Actor principal**: administrator

**Principalul scenariu de succes**:

1. Administratorul selectează tab-ul *Modify product*
2. Administratorul introduce informațiile despre produsul de bază pe care dorește să îl modifice în câmpurile din tabel. Și va apăsa tasta ENTER. Va lăsa selectată linia corespunzătoare produsului.
3. În cazul ștergerii se va selecta linia cu produsul dorit de șters.
4. Administratorul va apăsa butonul de *Modify product* sau *Delete Product*
5. Administratorul va primi un mesaj de informare cu privire la succesul operației efectuate.

**Secvențe secundare**

* Administratorul introduce date invalide și va fi informat asupra acestui aspect printr-un mesaj.
* Administratorul nu selectează nicio linie din tabel.

**Use case**: ștergere produs compus

**Actor principal**: administrator

**Principalul scenariu de succes**:

1. Administratorul selectează tab-ul *Modify product*
2. Administratorul selectează din lista dedicată produselor compuse produsul pe care dorește să îl șteargă.
3. Administratorul va apăsa butonul *Delete Product*
4. Administratorul va primi un mesaj de informare cu privire la succesul operației efectuate.

**Secvențe secundare**

**Use case**: modificare produs compus

**Actor principal**: administrator

**Principalul scenariu de succes**:

1. Administratorul selectează tab-ul *Modify product*
2. Administratorul selectează din lista dedicată produselor compuse produsul pe care dorește să îl modifice.
3. Administratorul va apăsa butonul *Update Composite* , iar în partea de jos va apărea un tabel cu produsele de bază din care e format produsul de compus cu câmpurile pe care poate să i le modifice.
4. Administratorul va introduce în câmpurile produselor de bază modificările dorite.
5. Administratorul selectează tab-ul *Modify* și va primi un mesaj de inforare.

**Secvențe secundare**

* Administratorul introduce date invalide; va fi informat asupra acestui aspect printr-o fereastră de mesaje.

**Use case**: creare produs compus

**Actor principal**: administrator

**Principalul scenariu de succes**:

1. Administratorul selectează tab-ul *Create product*
2. Administratorul introduce un nume pentru noul produs
3. Administratorul selectează din cele două liste de produse existente în meniu, produsele pe care vrea să le conțină noul produs și le va adăuga prin apăsarea celor două butoane *Add Base Product* și *Add Composite Product*
4. Administratorul apasă butonul *Finalize*
5. În caz de succes se va afișa un mesaj ce conține informațiile produsului noului creat.

**Secvențe secundare**

* Administratorul nu introduce un nume valid. În acest caz va primi un mesaj de informare.
* Administratorul nu a adăugat niciun produs pentru noul produs compus.

**Use case**: generare raport cu comenzile dintr-un interval orar

**Actor principal**: administrator

**Principalul scenariu de succes**:

1. Administratorul selectează tab-ul *Generate reports*
2. Administratorul adaugă intervalul orar
3. Administratorul apasă butonul *Generate report* aflat imediat sub câmpurile dedicate acestui raport

**Secvențe secundare**

* Orice în neregulă legată de consistența datelor introduse va fi semnalizată printr-un mesaj astfel încât utilizatorul să realizeze modificările corespunzătoare.

**Use case**: generare raport cu produsele comandate de un anumit număr de ori

**Actor principal**: administrator

**Principalul scenariu de succes**:

1. Administratorul selectează tab-ul *Generate reports*
2. Administratorul adaugă numărul minim de comandări ale produselor
3. Administratorul apasă butonul *Generate report* aflat imediat sub câmpul dedicat acestui raport

**Secvențe secundare**

* Orice în neregulă legată de consistența datelor introduse va fi semnalizată printr-un mesaj astfel încât utilizatorul să realizeze modificările corespunzătoare.

**Use case**: generare raport cu clienții care au comandat de un număr de ori peste o valoare dată

**Actor principal**: administrator

**Principalul scenariu de succes**:

1. Administratorul selectează tab-ul *Generate reports*
2. Administratorul adaugă numărul minim de comenzi și valoarea minimă per comandă
3. Administratorul apasă butonul *Generate report* aflat imediat sub câmpurile dedicate acestui raport

**Secvențe secundare**

* Orice în neregulă legată de consistența datelor introduse va fi semnalizată printr-un mesaj astfel încât utilizatorul să realizeze modificările corespunzătoare.

**Use case**: generare raport cu produsele comandate într-o anumită zi

**Actor principal**: administrator

**Principalul scenariu de succes**:

1. Administratorul selectează tab-ul *Generate reports*
2. Administratorul adaugă data formată din zi , lună și an
3. Administratorul apasă butonul *Generate report* aflat imediat sub câmpurile dedicate acestui raport

**Secvențe secundare**

* Orice în neregulă legată de consistența datelor introduse va fi semnalizată printr-un mesaj astfel încât utilizatorul să realizeze modificările corespunzătoare.

**Use case**: consultarea meniului

**Actor principal**: client

**Principalul scenariu de succes**:

1. Clientul selectează tab-ul *View Menu*
2. Clientul apasă butonul *Show Menu*
3. Se va afișa meniul cu produsele de bază (sub formă de tabel) și produsele compuse( sub formă de listă) dacă există meniul.

**Secvențe secundare**

* În cazul în care nu există meniul se va afișa un mesaj de informare în acest scop

**Use case**: căutare produse

**Actor principal**: client

**Principalul scenariu de succes**:

1. Clientul selectează tab-ul *Search*
2. Clientul introduce valori în câmpurile dedicate criteriilor de căutare
3. Un câmp gol reprezintă faptul că se va ignora în procesul de filtrare a produselor
4. Se vor afișa sub formă de tabel și listă produsele filtrate.

**Secvențe secundare**

* În cazul în care nu există meniul se va afișa un mesaj de informare în acest scop

**Use case**: plasare comandă

**Actor principal**: client

**Principalul scenariu de succes**:

1. Clientul selectează tab-ul *New Order*
2. Clientul selectează produsele dorite și le adaugă comenzii.
3. Clientul va apăsa butonul *Finalize* și se va plasa comanda dorită.

**Secvențe secundare**

* În cazul în care nu există meniul se va afișa un mesaj de informare în acest scop

**Use case**: vizualizare comenzi de procesat

**Actor principal**: angajat

**Principalul scenariu de succes**:

1. Angajatul va avea acces la o pagină în care vor apărea comenzile pe care trebuie să le proceseze în ordinea plasării lor.

**Secvențe secundare**

* În cazul în care nu există meniul se va afișa un mesaj de informare în acest scop

4.Proiectare

Obiectivul etapei de design sau proiectare este obținerea unui sistem bun prin utilizarea eficientă resurselor existente, precum și a unor resurse noi. De asemenea, scopul ei este de a sparge complexitatea sistemului folosind fie abordări de tipul top-down sau bottom-up. S-a folosit paradigma de programare orientată pe obiect.

Se pot defini mai multe nivele în cadrul etapei de design:

*Design level 1*

În cadrul primului nivel regăsim imaginea globală a sistemului. Aceasta se reprezintă sub forma unei „cutii negre” asupra căreia se trimit date de intrare și procesele ascunse din interiorul cutiei furnizează rezultatele așteptate. Principalul scop al sistemului de management al firmei de catering este gestiune produselor și a comenzilor plasate de clienți. Sistemul este unul determinist.

Text

Description automatically generated

Figure 2 Cutia neagră a sistemului

*Design level 2*

Sistemul de management este divizat în 5 subsisteme dintre care unul este cel care se ocupă cu lansarea aplicației (conține funcția *main*). Această arhitectură pe layere va întări proprietate de reutilizare a codului și minimizează dependențele/interacțiunile între subsisteme evitându-se relațiile circulare între acestea.

presentation- conține clasele care definesc interfața grafică cu controllerele asociate fiecărui window. Vor exista 4 ferestre: fereastra principală de logare, fereastra dedicată administratorului, clientului și cea a angajatului. Fiecărei ferestre îi corespunde un controller care vor descrie evenimentele asociate butoanelor din interfață pentru diferitele operații cerute de utilizatori

businesslogic– conține clasele care încapsulează logica aplicației. În cadrul acestuia sunt definite clasele pentru produse, care respectă un pattern composite.

* Prin patternul arhitectural composite s-a putut implementa o formă de recursivitate, mai ales pentru calcularea prețului produselor.
* Un alt pattern folosit este cel de design by contract la nivelul metodelor definite în interfața *IDeliveryServiceProcessing*, metode ce vor fi implementatede *clasa DeliveryService.* Prin acest pattern se va specifica comportamentul metodelor, eliminând ambiguitatea execuției anumitor metode asigurând constrângerile necesare execuției corecte a codului. Se vor regăsi astefle precondiții, postcondiții și invarianți care trebuie satisfăcuți în momentul execuției metode conform contractului definit.
* În cadrul acestui sistem se regăsește și clasa principală care implementează toate operațiile specifice unui sistemului de catering, *DeliveryService*

*model*- conține clasa *User* și enum-ul UserType în care sunt definite cele 3 tipuri de utilizatori: CLIENT, ADMIN și EMP (angajatul)

*data*- acest sistem se va ocupa salvarea și încărcarea datelor aplicației precum și scrierea în fișier pentru chitanță comandă și rapoarte, precum se va ocupa și de citirea fișierului .csv

Diagram

Description automatically generated

Figure 3 Diagrame pachete

*Design level 3*

La acest nivel se definesc clasele fiecărui subsistem și interacțiunile dintre ele. Denumirea claselor, e sugestivă și inspirată din domeniul problemei *(Order, BaseProduct, MenuItem, DeliveryService, User*).

.

Diagram, timeline

Description automatically generated

Figure 4 Diagrame UML

*Design-ul de nivel 4*, care se referă la împărțirea claselor în rutine este ilustrat în cadrul diagramei UML. Există câteva metode private care ajută la îndeplinirea unor task-uri din interiorul unor metode publice. De exemplu, în clasa *FileWriter* regăsim metoda statică private *BufferedWriter getBuffer(String name)*, metodă ce va crea un buffer de scriere asociat unui fișier nou cu numele *name,* deoarece această operație este efectuată de majoritatea claselor statice ale clasei FileWriter.

Având partea de proiectare finalizată se poate trece implementarea claselor cu funcționalitățile acestora.

5.Implementare

* 1. **Clase și metode**

În ceea ce privește denumirea claselor și a metodelor s-au ales nume sugestive care să reflecte domeniul problemei (sistemul de management al unei firme de catering), un exemplu fiind cel prezentat în subcapitol *Modelare*. Prin folosirea uni pattern arhitectural (pe layere) s-a crescut proprietatea de reutilizare a codului, astfel încât clase precum *MenuItem, Order, User, Serializator* etc. pot fi folosite în mod independent fiindcă s-au minimizat dependențele circulare. În continuare sunt prezentate clasele din cele 5(7) pachete ale sistemului împreună cu câmpurile și capabilitățile lor importante.

1. **Clase din pachetul „model**”

Enum*-ul* ***UserType –*** în cadrul acestei enumerații sunt definite cele 3 tipuri de utilizatori: client, administrator și angajat (*CLIENT*,*ADMIN*, *EMP*).

*Clasa* ***User*** -este clasa care descrie modelul real de utilizator. Există un câmp de tipul UserType prin care se face distincția între tipurile de utilizator. Un obiect de tipul User va fi caracterizat prin id propriu, username și parolă. Această clasa implementează interfața *Serializable* pentru a putea fi salvate datele despre useri la închiderea aplicației și de asemenea încărcarea obiectelor de acest tip prin procesul invers de deserializare.

Clasa Client are doi **constructori**:

|  |  |
| --- | --- |
| Constructori | Descriere |
| public User(Integer userId, String userName, String password, UserType type) | -se vor seta câmpurile cu parametrii corespunzători |

Pe lângă metodele mutatoare și accesoare s-au suprascris metodele de equals și hascode pe campurile de username și password astfel încât useri se pot introduce într-o structura de tip hash table, mai precis *HashSet*.

1. **Clase din pachetul *businesslogic***

*Clasa abstractă MenuItem*-este clasa care descrie un produs din meniu. Aceasta clasă face parte din pattenul composite MenuItem-BaseProduct-CompositeProduct, prin care se permite o recursivitatea la nivelul produselor compuse care pot fi formate atat in produse de bază, cât și din alte produse compuse. Există doar metoda abstractă *computePrice(),* metodă necesară pentru calculul recursiv al prețului pentru produsele de tip composite.

Clasa Baseproduct – descrie un produs de bază din meniu moștenind clasa *MenuItem*. Câmpurile fac referire la numele produsului, rating-ul acestuia, aportul de calorii, sodiu, proteine și grăsimi și de asemenea prețul produsului.

Sunt mai mulți constructori pentru această clasă, definiți cu scopul de a ușura implementarea aplicației. Dintre acești constructori voi aminti pe cel folosit direct de metodele care citesc din interfață sau fișier valori sub formă de string pentru obiectele de tip BaseProduct

|  |  |
| --- | --- |
| Constructori | Descriere |
| public BaseProduct(String title,String rating,String calories,  String protein,String fat, String sodium, String price ) throws NumberFormatException | -se vor seta câmpurile cu parametrii parsați în conformitate cu tipurile cerute; în caz că nu se poate realiza parsarea se va arunca o excepție |

Pe lângă metodele mutatoare și accesoare s-au suprascris metodele de equals și hashcode pe câmpul title și de asemnea se ține cont și de produsele compuse. Această implementare permite crearea unui HashSet cu produse pentru meniul restaurantului. Implementarea funcției *computePrice()* reprezintă returnarea câmpului pentru prețul produsului.

*Clasa CompositeProduct* extinde clasa MenuItem. Conține un câmp de tipul Set<MenuItem>, containedProducts astfel încât un produs de tip compus poate conține atât produse de bază, cât și alte produse compuse (caracter recurent). Implementarea funcției *computePrice()* constă în parcurgerea mulțimii produselor componente și însumarea prețului acestora ( se va apela din nou metoda computePrice() pe produsele iterate).

Și aceasta clasa va suprascrie cele 2 metode equals() și hashCode() la fel ca în cazul clasei *BaseProduct.*

*Clasa Order* descrie modelul real de comandă cu câmpurile necesare descrierii acestuia: id-ul comenzii, lista de produse comandate, data și id-ul clientului. Această clasa va implementa interfața Serializable pentru salvare comenzilor după închiderea aplicației și încărcare lor de asemenea, și interfața Comparable pentru a putea fi sortate după id comnezile. Și aceasta clasa va suprascrie metodele equals și hashCode, luând în considerare toate cele 3 câmpuri.

Interfața IDeliveryServiceProcessing-conține metodele specifice administratorului și clientului pe care clasa DeliveryService trebuie să le implementeze după patternul design by contract.

Exemplu de design by contract:

/\*\*

\* Functia de import produse administrator

\* @pre true

\* @post menu!=null and user.size()==user.size()@pre and orders.size()==orders.size()@pre

\* @return daca s-au importat sau nu

\* @invariant isWellFormed()

\*/

public boolean importProducts();

Clasa DeliveryService va prezenta metodele și logica de funcționare a restaurantului. Această clasă va conține un câmp pentru meniul restaurantului care va avea urmatorul tip: Set<MenuItem>, un câmp pentru toate comenzile restaurantului de tipul: Map<Order, Collection<MenuItem>>, un camp pentru mulțimea userilor care au acces la aplicație: Set<User> și încă 2 câmpuri care țin ultimele id-uri folosite atât pentru comenzi, cât și pentru useri.

*Constructorul* : inițializeză câmpurile obiectului sau nu, în funcție dacă găsește fițiere corespunzătoare deserializării câmpurilor specifice.

|  |  |  |
| --- | --- | --- |
| Metode(implementate din interfața IDeliveryServiceProcessing) | | Descriere |
| public boolean importProducts(); | -operație specifică administratorului  -se vor importa produsele din fisierul .csv în Map-ul definit în DeliveryService pentru meniul restaurantului  -se va face apelul metodei statice *initializeList()* din FileWriter  -se folosesc stream-uri și expresii lamba de tip predicate | |
| public boolean deleteProduct(MenuItem menuItem); | - operație specifică administratorului  -va șterge produsul transmis ca parametru din meniu | |
| public boolean editProduct(MenuItem menuItem); | - operație specifică administratorului  -va actualiza produsul din menu (cu acelasi nume) cu noile informații | |
| public boolean addProduct(MenuItem menuItem); | -operație specifică administratorului  -se va adăuga un nou produs in meniu | |
| public boolean generateReportTime(int startHour, int endHour); | -se va genera raportul cu privire la comenzile din intervalul orar;  -se face apel pe metoda statică specifică din FileWriter  - se folosesc stream-uri și expresii lamba de tip predicate | |
| public boolean generateReportProducts(int number); | -se va genera raportul cu privire la produsele comandate peste un anumit număr de ori  -se folosesc stream-uri și expresii lamba de tip predicate  -se face apel pe metoda statică specifică din FileWriter | |
| public boolean generateReportClient(int numberOrders, int value); | -se va genera raportul cu privire la clienții care au comandat de un anumit număr de ori peste o valoare dată  -se face apel pe metoda statică specifică din FileWriter | |
| public boolean generateReportDay(int day, int month, int year); | -se va genera raportul cu privire la produsele comandate într-o anumită zi;  - se folosesc stream-uri și expresii lamba de tip predicate  -se face apel pe metoda statică specifică din FileWriter | |
| public void createNewOrder(User user, List<MenuItem> menuItems); | -operație specifică clientului  -crearea unei noi comenzii și apelul metodei statice specifice din FileWriter pentru scrierea în fișier txt a comenzii  -se folosesc stream-uri și expresii lamba de tip *predicate* | |
| public Set<MenuItem> searchProduct(String title, Double rating, Integer calories, Integer proteins, Integer fats, Integer sodium, Double price ); | -operație specifică clientului  -folosirea de stream-uri pentru filtrarea anumitor produse după criteriile transmise ca parametru; | |

Alte metode importante:

* public void saveData() – Metoda pentru salavarea câmpurilor din delivery service la închiderea aplicației
  + - se vor instantia obiecte de tip serilizator<tip> pe care se va apela metoda save a obectului de tip Serializator
    - exemplu pentru câmpul *menu*
      * *Serializator<Set<MenuItem>> serializatorMenu=new Serializator<>();*
      * *serializatorMenu.save(menu,"menu");*
* *protected boolean isWellFormed() -*pentru invariantul clasei;este bună sau nu starea obiectului de tip DeliveryService
* *public User getConnection(User user) -*se va căuta userul cu numele și parola celui transmis ca parametru în mulimea userilor înregistrați în aplicație, și în caz afirmativ îl va returna

**Clase din pachetul data**

*Clasa FileWriter* conține metode statice de importare a produselor din fișierul .csv folosind stream-uri și expresii lambda , funcția de generare a fișierului cu chitanța unei comenzi și metodele de creare a fisierilor pentru clee 4 tipuri de rapoarte. Mai multe detalii în Javadocț

Clasa Serializator este o clasă generică, care conține două metode *save****()-pentru serializarea unui obiect*** *și loadData(String name****)-pentru procesul de deserializare****.*

*Exemplu*: deserilazare în cazul constructorului clasei DeliveryService

Serializator<Set<User>> serializatorUsers=new Serializator<>();

users=serializatorUsers.loadData("users");

**Clase din pachetul presentation**

Conține 4 clase pentru cele 4 window-uri: (+clasele pentru Controller)

* LogView -pentru window-ul de logare și înregistrare
  + Constructorul: în cadrul lui se definește poziționarea componentelor grafice
  + Metode importante:
    - public void addClosingListener(DeliveryService deliveryService)
      * la închiderea ferestrei se va apela metoda de serializare a câmpurilor obiectului deliveryService
    - public void addLogListener(ActionListener e) - Adaugare listener log in
    - public void addRegListener(ActionListener e) - Adaugare listener register
    - gettere pentru cele două câmpuri și tipul de utilizator
  + Are asociat un controller de tip ControllerLogin
    - In constructor se apelează adaugarea de listeneri pentru obiectul de tip LogView asociat controlerrului
    - Subclasele pentru implementarea metodei de event pentru butoanele din fereastra LogView
      * public class LogListener implements ActionListener -metoda suprascrisă va extrage câmpurile dininterfață și vor face apelul metodei getConnection pe obiectul de tip DeliveryService pentru a verifica dacă există utilizatorul
      * public class RegListener implements ActionListener- înregistrarea unui nou client cu valorile introduse în interfața LogView dacă sunt valide și dacă nu exită un alt utilizator cu aceleași date deja inregistrat
* AdministratorView pentru window-ul specific administratorului
  + Constructorul: în cadrul lui se definește poziționarea componentelor grafice
  + Conține metode prin care se actualizează componentele grafice (panou-rile) în momentul când se fac modificări la nivelul datelor
    - setModifyPanel(DeliveryService deliveryService) -actualizarea tabelului cu produsele de baza modificate
    - public void setModifyPanelComposite(CompositeProduct compositeProduct)- actualizarea listei cu produsele modificate
    - public void setCreateProdPanel(DeliveryService deliveryService)-actulizarea panoului pentru crearea de produse noi
  + Are asociat un controller de tip AdministratorController
    - Va adăuga listenerii pentru butoanele din window-ul administratorului prin crearea unor subclase care implementează interfața ActionListener. Metodele public void actionPerformed(ActionEvent e) suprascrise vor prelua datele din interfața și va face apelurile metodelor asociate din obiectul de tip *DeliveryService*
    - *În javadoc se regăsesc informații suplimentare*
* ClientView -pentru window-ul specific administratorului
  + Constructorul: în cadrul lui se definește poziționarea componentelor grafice
  + Conține metode prin care se actualizează componentele grafice (panou-rile) în momentul când se cere din metodele listenerilor asociați butoanelor din interfață
  + Are asociat un controller de tip ClientController care are subclase ce implementează interfața ActionListener
    - De exemplu: subclasa *SearchListener implements ActionListener* va suprascrie metoda *actionPerformed* prin care se preiau câmpurile din interfața clientului în care acesta a introdus criteriile de căutare și se va face apelul metodei *searchProduct* a obiectului deliveryService, care va returna lista produselor din meniu filtrate. După se va face apelul metodei de actualizare a panelului dedicat rezultatelor a interfeței *clientView*
    - Celelalte sublclase se regăsesc în JavaDoc
* EmployeeView – această clasă va fi un Observer, iar subiectul de observat este obiectul de tip DeliveryService: în momentul în care se apelează funcția *createNewOrder* pe obiectul de tip DeliveryService se va trimite un mesaj de informare observatorilor că s-a generat o nouă comandă. În

metoda update din EmployeeView se va actualiza lista cu comenzile de procesat (se adaugă noua comandă)

Clasa Tables conține metode statice de generare a unui tabel de tip JTabel cu produse de bază și metode de generare a unei liste de tipul JList pentru produsele compuse.

Conține și o metoda private statică de obținere a unei matrice de stringuri pentru liniile tabelului (fiecare linie prezintă informațiile despre un produs).

**Clase din pachetul startapp**

Conține metoda main de lansare aplicației.

* 1. **Interfața grafică cu utilizatorul (GUI)**

Interfața grafică cu utilizatorul este destul de intuitivă și folosește elemente din pachetul *javax.swing*. Am încercat să ofer un aspect cât mai prietenos pentru utilizator pentru ca acesta să își dea seama cât mai ușor de modul de funcționare, fie el administrator, client sau angajat.. În cazul în care va introduce valori necorespunzătoare acesta va fi informat.

Scurtă prezentare:

La pornirea aplicației va apărea fereastra de log in.

Graphical user interface, application

Description automatically generated

Figure 5 Fereastra principală

Fereastra administratorului coține tab-uri asociate operațiilor pe care le poate executa:

Graphical user interface, application, table, Word

Description automatically generatedGraphical user interface, application, Word

Description automatically generated

Figure 6 Fereastra Adminstrator- import

Figure 7 Fereastra Adminstrator- adaugare produs baza

Graphical user interface, application

Description automatically generatedGraphical user interface, table

Description automatically generated

Figure 10 Fereastra Adminstrator- generare rapoarte

Figure 8 Fereastra Adminstrator- modificare produse

Graphical user interface, text, application

Description automatically generated

Figure 9 Fereastra Adminstrator- creare produs compus

Graphical user interface, table

Description automatically generated with medium confidenceGraphical user interface, table

Description automatically generated

Figure 12 Fereastra Client -cautare produse

Figure 11 Fereastra Client -consultare meniu

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Figure 14 Fereastra Employee -vizualizare comenzi

Figure 13 Fereastra Client -plasare comanda

Text

Description automatically generated

Figure 15 Fisier .txt pentru chitanță

6.Rezultate

Modul de testare a constat în verificări repetate a actualizărilor de la nivelul interfeței grafice a tabelelor și listelor cu produse după operații diverse. De asemenea, s-a verificat modul de căutare a produselor, operație specifică clienților, și de asemenea s-a vizualizat fișierul cu chitanța pentru o comandă făcând verificări la nivelul consistenței datelor dorite și calcul prețului total de achitat.

7.Concluzii

În concluzie, prin realizarea acestei teme de laborator, s-a realizat un sistem de management al unei firme de catering (food delivery), prin intermediul căruia utilizator se poate loga în calitate de admin, client sau angajat și poate executa operațiile specifice, menționate în capitolele precedente.

Pentru proiectarea aplicației s-a folosit paradigma orientată pe obiect, folosindu-se layered Architectures, care îi oferă proprietate de reutilizare a codului/ claselor în alte aplicații. S-au folosit mai multe tipuri de pattern-uri: design by contract, composite design, observer design. Întreținerea codului este mai ușor de realizat, iar modelul poate fi reutilizat de alte aplicații. De asemenea și dezvoltarea aplicației de față se poate efectua independent la nivel de clase, fie la nivelul interfeței grafice, fie în cadrul modelului.

Pentru cel care a realizat aplicația, dezvoltarea ei l-a familiarizat mai mult cu conceptul de stream-uri și expresii lambda (cu conceptul de programare declarativă), cu modul de modelare, proiectare și implementare a unei aplicații folosind paradigma de programare OOP. De asemenea, l-a obișnuit cu urmarea pașilor logici în procesul de dezvoltare a unei aplicații software , de la analiza problemei, fixarea cerințelor, proiectarea, implementarea și testarea ei. Și-a îmbunătățit lucrul cu colecții, cu structuri de tip HashTable și a învățat mai mult despre tipurile de design(menționate anterior) pe care poate să le folosească pentru structurarea aplicației.

Ca posibilă dezvoltare ar fi crearea posibilității de a lucra în paralel utilizatorii, adică să fie poată fi vizibile toate ferestrele dedicate utilizatorilor- va necesita tehnici de sincronizare.

8.Bibliografie

* „ASSIGNMENT 4-SUPPORT PRESENTATION” , material auxiliar ( furnizat pe platforma Microsoft Teams)
* Cursuri de la disciplina Tehnici de Programare Fundamentală, studiată în semestrul al 2-lea al anului universitar 2020-2021
* <https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#tag>
* <https://intellij-support.jetbrains.com/hc/en-us/community/posts/207014815-How-to-enable-assert>
* <http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html>
* <http://www.tutorialspoint.com/java/java_serialization.htm>
* <https://howtodoinjava.com/java8/java-stream-distinct-examples/>
* <https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html>