DOCUMENTATIE

TEMA 3

NUME STUDENT: VOICU LAURA-LUISA
GRUPA: 30226

CUPRINS

1. Obiectivul temei

Obiectivul unei aplicații de order management este de a gestiona cu succes toate comenzile primite de la clienți, de la primirea comenzii până la livrarea produselor către clienți. Această aplicație poate fi utilizată pentru a urmări și gestiona inventarul, pentru a urmări și analiza vânzările, precum și pentru a oferi o experiență de cumpărare mai bună clienților prin îmbunătățirea procesului de comandă și livrare.

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

Descrierea problemei:

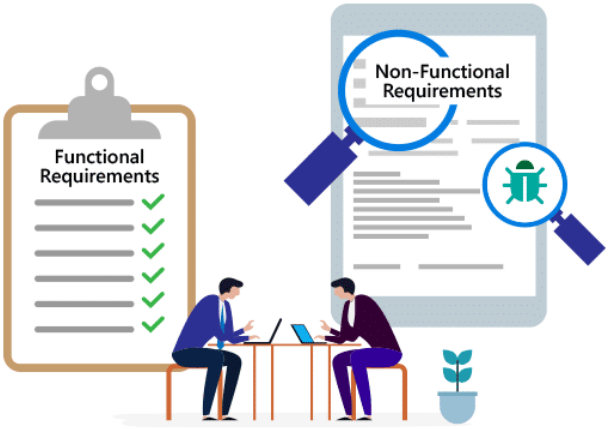
O aplicatie de order management se ocupa de gestionarea comenzilor primite de la clienti si de procesarea lor in mod eficient. Principalele probleme pe care aceasta aplicatie trebuie sa le gestioneze sunt stocarea informatiilor despre comenzile primite, gestiunea stocurilor de produse, procesarea platilor si monitorizarea livrarii. In plus, o astfel de aplicatie ar trebui sa permita o comunicare facila intre diferitele departamente ale companiei si sa ofere rapoarte utile despre starea comenzilor si a stocurilor.

➢ Cerinte functionale

- ***Inregistrarea comenzilor:***
  - *utilizatorii ar trebui sa poata introduce informatii despre comenzile primite, cum ar fi numele si adresa clientului, produsele si cantitatile comandate, precum si orice alte detalii relevante.*
- ***Managementul stocurilor:***
  - *Aplicatia ar trebui sa poata urmari cantitatile de produse disponibile in stoc si sa actualizeze stocurile dupa fiecare comanda procesata;*
- ***Generarea de rapoarte***
  - *Aplicatia ar trebui sa ofere utilizatorilor rapoarte utile despre starea comenzilor si a stocurilor pentru a ajuta la luarea deciziilor in timp util..*
- ***Administrarea bazei de date:***
  - *Utilizatorii trebuie sa poata administra baza de date, adaugand sau stergand informatii despre produse, clienti si comenzi.*
- ***Monitorizarea livrarilor:***
  - *Aplicatia trebuie sa poata urmari livrarile comenzilor si sa actualizeze informatiile despre statusul lor in baza de date.*

➢ Cerinte non-functionale

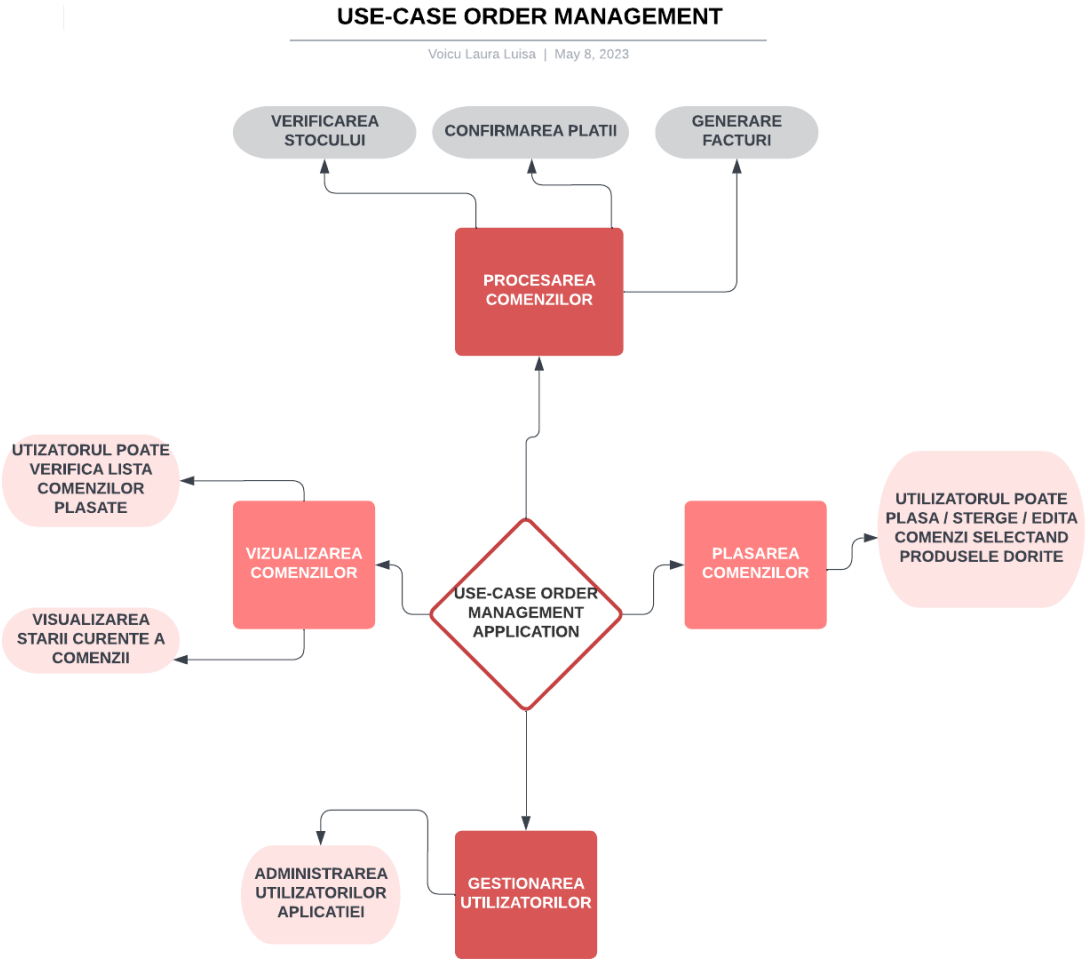
- ***Salabilitate*** *:*
  - *Capacitatea aplicatiei de a gestiona o crestere a volumului de date si a numarului de utilizatori trebuie sa fie luata in considerare si sa fie asigurata o capacitate adecvata de scalare.*



- ***Performanta:***
  - *Aplicatia trebuie sa aiba o performanta ridicata, astfel incat sa poata gestiona eficient volumul mare de date generate de comenzile primite si sa ofere un timp de raspuns rapid utilizatorilor.*

- ***Fiabilitate:***
  - *Aplicatia trebuie sa fie fiabila si sa ofere o experienta fara erori sau intarzieri pentru utilizatori, astfel incat sa nu afecteze procesul de gestionare a comenzilor si sa asigure un nivel ridicat de satisfactie a clientilor.*
- ***Usurinta de utilizare:***
  - *sistemul trebuie sa fie usor de utilizat pentru clienti, astfel incat acestia sa poata intelege si sa foloseasca cu usurinta interfata utilizatorului.*

➢ USE-CASE

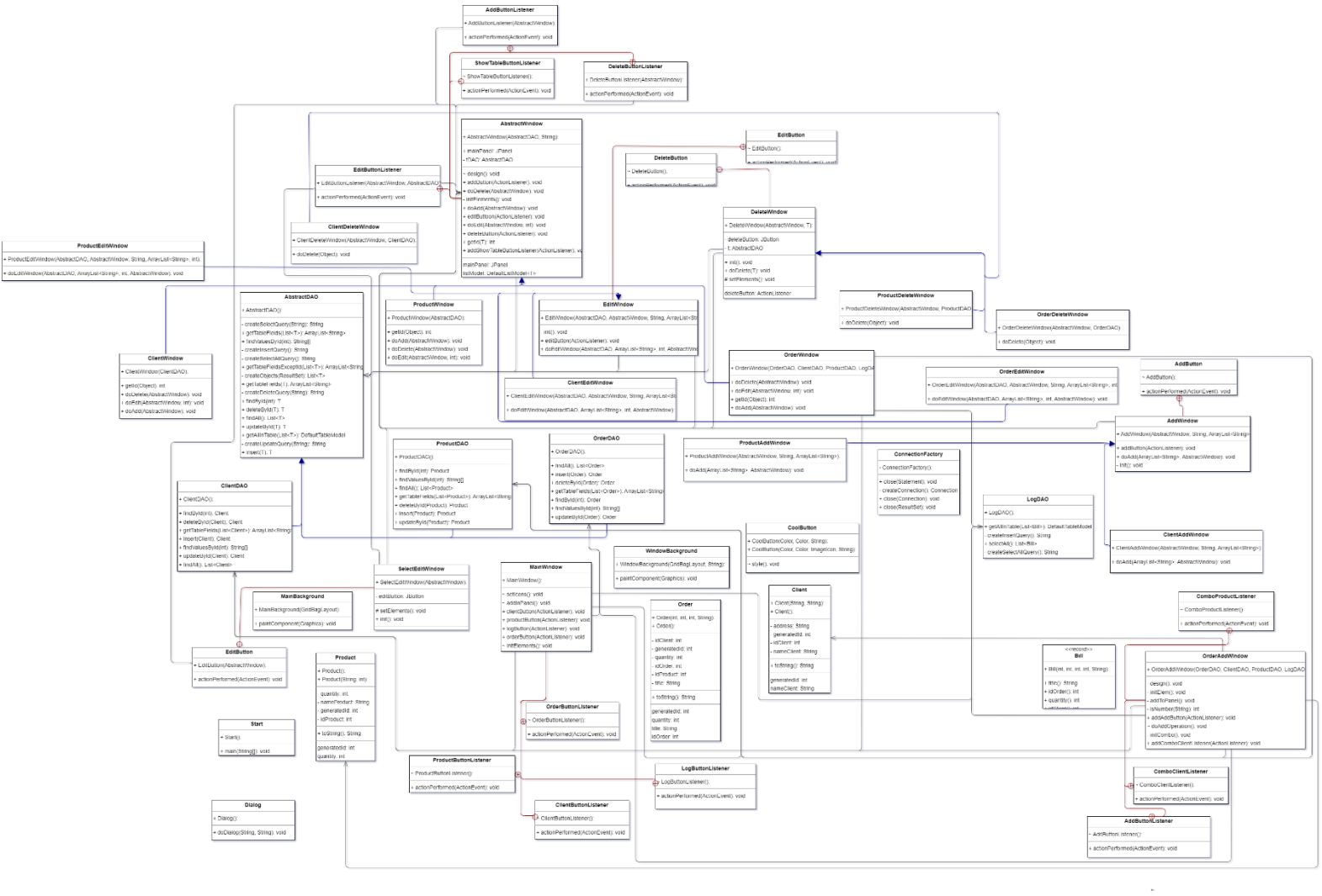


3. Proiectare

➢ Proiectarea OOP

- Aplicatia este construita dupa arhitectura Layered Architecture, fiind astfel alcatuita din package-urile principale dataAccessLayer, businessLayer, model, presentation.
- Descrierea package-urilor:
  - **connection :** alcatuira dintr-o singura clasa ce realizeaza conexiunea cu baza de date
  - **dao:** Data Abstract Objects → in cadrul pachetului se regaseste o clasa generica ce e utilizata pentru implementarea query-urilor . Cu ajutorul reflexiei clasa este folosita ca “baza” pentru implementarea query-urilor specifice fiecarei tabele din baza de date (**client, product, order, log)**
  - **model:** pachetul contine clasele specifice fiecarei tabele ; observatie: aici se regaseste si clasa **Bill** ce este o clasa imutabila (implementata ca **record**) folosita la inregistrarea comenzilor (fara a se permite modificarea acestora)
  - **presentation:** clasa in sine este alcatuita din subpachete si clase cu ajutorul carora este construita interfata aplicatiei (si aici se va folosi conceptul de **reflection**)
  - **start:** in cadrul acestui pachet se afla clasa ce “porneste” aplicatia

Descrierea detaliata a fiecarei clase poate fi regasita in sectiunea **4.**



- **Structurile de date** principale folosite in program sunt:
  - **ArrayList** generic

- Printre algoritmii folositi se regasesc :
  - Algoritm de verificare a corectitudinii input-ului introdus de utilizator prin intermediul bibliotecii **REGEX**.
  - Algoritmi elementari : calculul cantitatii ramase unui produs in urma efectuari unei comenzi.

4. Implementare

Aplicatia , construita dupa pattern-ul Layered Architecture, implementeaza clasele din cele 4 pachete principale astfel:

- **connetion :**
  - **ConnectionFactory**
    - **Metode: getConnection** (folosita pentru a genera conexiunea cu baza de date), **close** (folosita pentru a inchide conexiunea la baza de date)
- **dao**
  - **AbstractDAO**
    - **Metode:**

      - **createSelectAllQuery→** folosita la compunerea unui string ce contine intructiunea SQL de selectare a tuturor datelor dintr-o tabela

- **createSelectQuery→** folosita la compunerea unui string ce contine intructiunea SQL de selectare a datelor dintr-o tabela in functie de id
- **createInsertQuery →** folosita la compunerea unui string ce contine instructiunea SQL de inserare a unui element (generic, ulterior particularizat de catre fiecare clasa ce va mosteni clasa AbstractDAO) in baza de date
- **createUpdateQuery →** folosita la compunerea unui string ce contine instructiunea SQL de editare a datelor din baza de date, mai putin id-ul
- **getTableFields→** folosita pentru a determina field-urile unei tabele
- **findValueById→** folosita pentru a gasi valorile de pe o linie din baza de date ce corespunde unui id dat
- **findById →** folosita pentru a gasi obiectul (nu valorile) corespunzatoare unui id
- **updateById →** metoda in cadrul careia se modifica obiectul, atat in baza de date, cat si in interfata

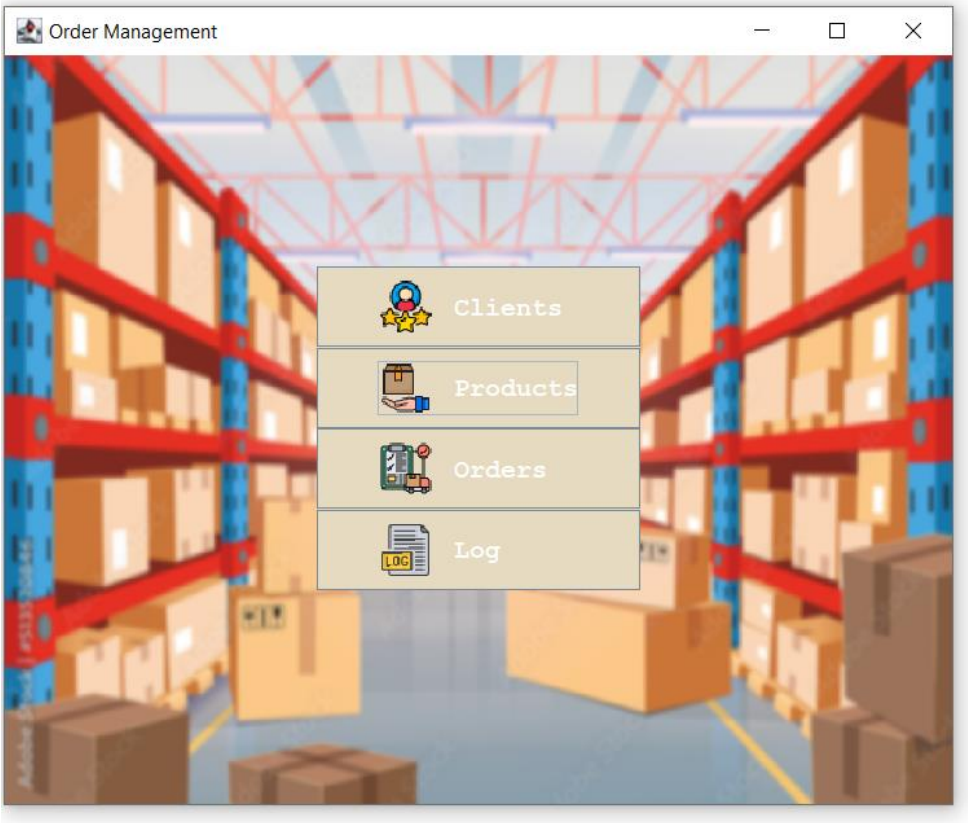
  - **ClientDAO , ProductDAO, OrderDAO :** clase ce mostenesc clasa **AbstractDAO** si suprascriu metodele (toate mai putin cele de tip createXQuery care sunt private) astfel incat sa se plieze pe cerintele fiecarui obiect
  - **BillDAO :** aceasta clasa nu foloseste reflexia, intrucat nu e necesara decat implementarea operatiilor de insert si select

❖ **model:**

- **Bill :** clasa de tip **record** ce asigura imutabilitatea acesteia
- **Client, Order, Product:**
  - Metode: doar gettere si settere
  - Proprietati: **“**copii” a field-urilor din baza de date
    - **Observatie:** toate acestea au o proprietate “**generatedID”** ce genereaza id-ul pentru fiecare tabela astfel incat procesarea si ordinea lor sa fie mai usor de interpretat si manipulat

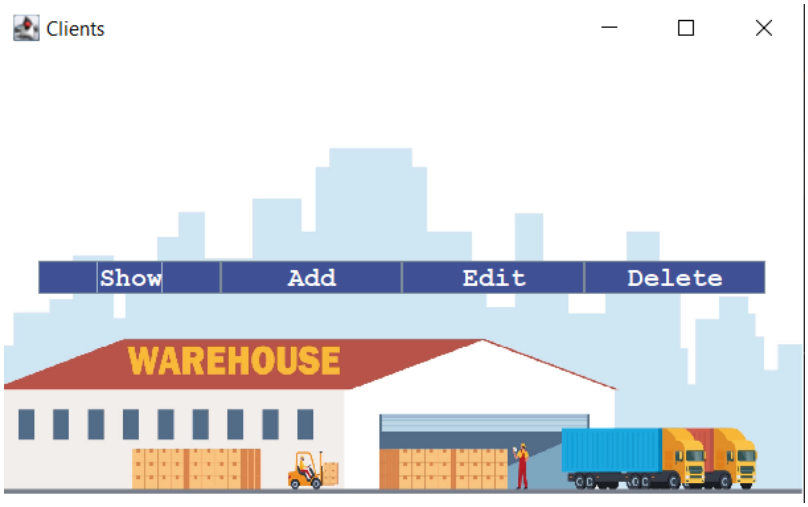
❖ **presentation:**

Clasa **MainWindow** ce creeaza o fereastra la inceperea aplicatiei din care utilizatorul poate sa selecteze asupra carei tabele vor fi efectuate urmatoarele query-uri



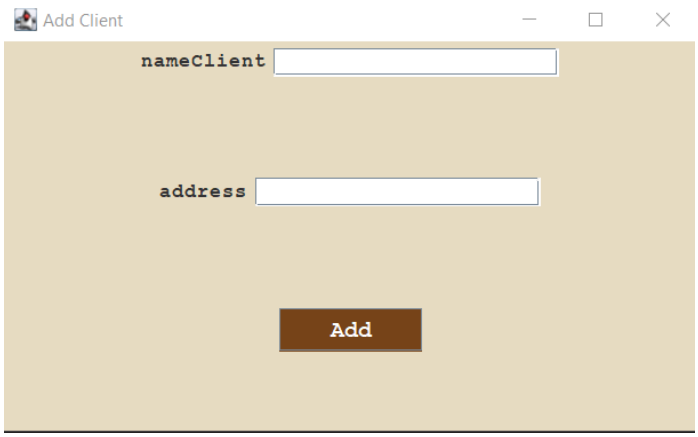
Sub-package-uri**:**

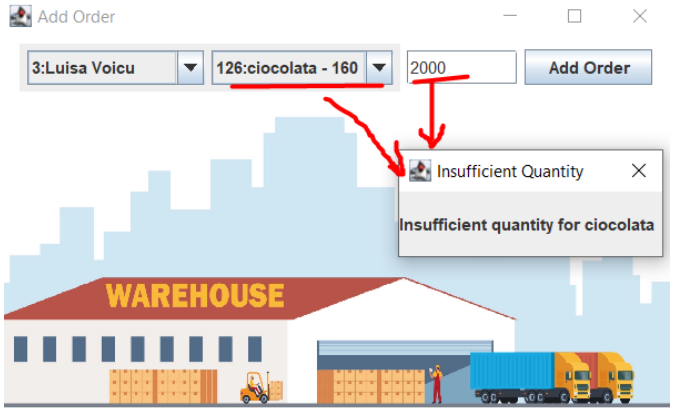
- **model.window:**
  - **AbstractWindow :** clasa generica ce defineste metode folosite la crearea butoanelor, tabelelor, panel-urilor ferestrei . Utilizatorul poate accesa 4 ferestre independente specifice fiecarei tabele unde poate alege ce operatii vor fi efectuate prin apasarea butoanelor. La apasarea unui buton se va deschide o noua fereasta cu specificatii (detalii mai jos)
  - **ProductWindow , ClientWindow, OrderWindow** mostenesc si suprascriu metode din AbstractWindow pliabile pe datele fiecarei tabele
  - **BillWindow** are doar optiunea de afisare a unei copii a tabelului din baza de date

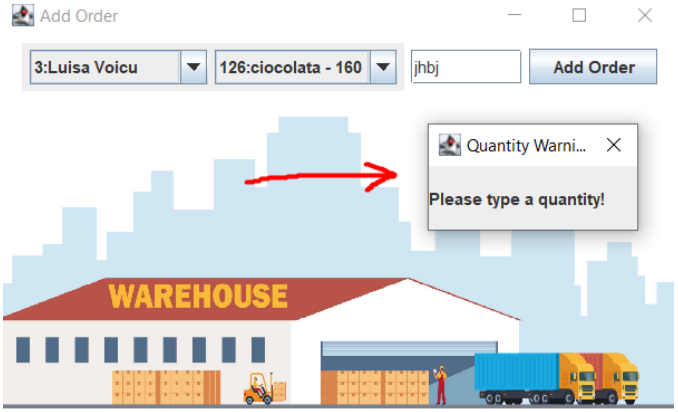


Ferestra ClientWindow *alcatuita din butoane corespunzatoare query-urilor Select * , Insert, Update, Delete*

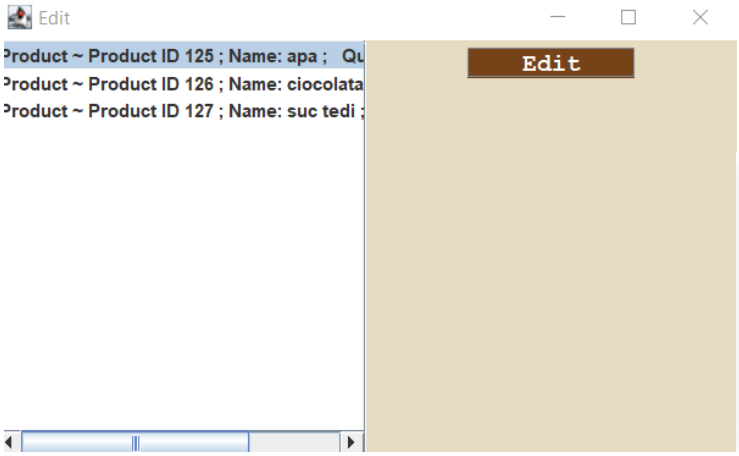
- **add:**
  - la fel ca si pentru model.window, pachetul add e alcatuit din clasa **AddWindow** mostenita de clasele ClientAddWindow, ProductAddWindow, OrderWindow.
  - In cadrul metodei AddWindow vor fi implementate metode ce creeaza Label-uri specifice field-urilor fiecarei tabele, TextBox-uri in care sunt introduse de utilizator date si buton ce transmite finalizarea completarii de catre utilizator
  - **AddOrderWindow** difera de structura generala deoarece adaugarea unei noi comenzi se realizeaza prin selectarea unui client, selectarea unei comenzi si introducerea unei cantitati (se poate observa in imaginea de mai jos din partea dreapta) . De asemenea, in cazul in care utilizatorul nu selecteaza produsul sau clientul, ori cantitatea introdusa e necorespunzatoare (fie este solicitata o cantitate mai mare decat cea inregistrata in baza de date, fie nu sunt introduse numere, ci caractere) aplicatia genereaza o fereastra pop-up care anunta utilizatorul de greseala facuta, iar comanda nu e inregistrapa pana ce acesta corecteaza.

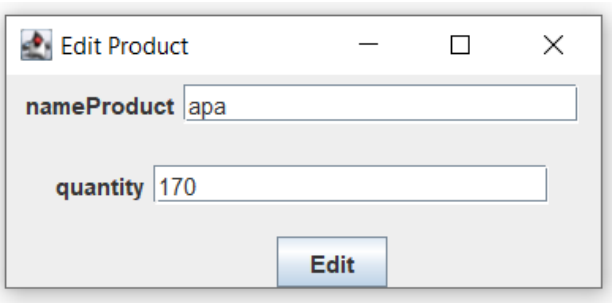






- **selectedit / edit**
  - pachetele implementeaza clasele specifice ferestrelor ce vor aparea pe ecran in urma selectarii butonului “Edit” .
    - clasa **SelectEditWindow** creeaza o ferestra alcatuita intr-un JList ce permite selecatarea unui obiect si un buton . Dupa apasarea butonului userul este redirectat care o alta fereastra (implementata de clasele din pachetul **edit** → clasa principala generalizata: **EditWindow** ce defineste metode ulterior implementate de clasele **OrderEdit,ClientEdit,ProductEdit**) unde userul poate introduce datele dorite.





- **delete**
  - pachetul contine clasele **DeleteWindow (generalizata) , ClientWindow , ProductWindow, OrderWindow** care folosesc un JList pentru selectarea obiectului ce se doreste a fi sters din baza de date si un buton ce efectueza aceasta operatie

5. Rezultate

Testarea functionalitatii aplicatiei a fost constituita din teste din perspectiva userului cu date ale caror rezultat era cunoscut, introduse direct din interfata grafica sau din baza de date.

6. Concluzii

In concluzie, aplicatia ofera o modalitate eficienta si usor de utilizat pentru efectuarea operatiilor asupra unei baze de date. Proiectul in sine m-a ajutat sa ma familiarizez cu software-ul GitHub (crearea unui repository, push, commit, pull, etc) si cu IDE-ul IntelliJ IDEA. Crearea aplicatiei m-a ajutat la extinderea cunostintelor in ceea ce priveste design-ul OOP, crearea unei interfete grafice folosind Java Swing, lucrul cu expresii regulate, dar si aspecte referitoare la organizarea codului (utilizarea pachetelor, conventii de nume,etc).

Particularitatea acestui proiect este data de lucrul si conexiunea la baza de date, dar si utilizarea reflexei si claselor de tip record.

Posibile imbunatatiri:

- Adaugarea unei functionalitati de urmarire a comenzilor pentru clienti - aceasta ar putea include furnizarea de informatii despre starea comenzii, istoricul comenzii, estimari ale timpului de livrare si notificari de livrare.

- Integrarea cu un sistem de plata securizat - astfel, clientii ar putea plati comenzile direct prin aplicatie, oferind o experienta de cumparare mai fluida si mai sigura.

- Implementarea unui sistem de recenzii si evaluari pentru clienti - aceasta ar putea ajuta la imbunatatirea experientei clientilor prin furnizarea de feedback si sugestii de imbunatatire.

- Adaugarea de functionalitati de analiza a datelor - astfel, afacerile ar putea obtine o mai buna intelegere a comportamentului clientilor, a preferintelor de cumparare si a trendurilor de vanzare, ajutand la imbunatatirea strategiilor de marketing si de afaceri.

- Dezvoltarea unei aplicatii mobile - astfel, clientii ar putea accesa si plasa comenzile direct prin aplicatia mobila, permitand o mai mare accesibilitate si o experienta mai prietenoasa pentru utilizator.

7. Bibliografie

I. FUNDAMENTAL PROGRAMMING TECHNIQUES – curs : https://dsrl.eu/courses/pt/
II. Programare Orientata pe Obiecte – Ionel Giosan : https://users.utcluj.ro/~igiosan/teaching_poo.html
III. https://www.flaticon.com/packs/animal-27