**TEHNICI DE PROGRAMARE**

**TEMA 3**

**WAREHOUSE MANAGEMENT**

**DOCUMENTATIE**

**Nume și prenume student: Marchis Raul**

**Profesor coordonator: Pop Cristina**

**Grupa: 30229**

1. **Cerinta temei………………………………………………………………………………………….…pag 3**
2. **Obiectivul temei………………………………………………………………………………….......pag 3**
   1. **Obiectivul principal…………………………………………………………………………pag 3**
   2. **Obiectivul secundar………………………………………………………………………..pag 4**
3. **Analiza problemei, modelare, scenarii, cazuri de utilizare…………………………..pag 4**
4. **Proiectare…………………………………………………………………………………………………..pag 6**
5. **Implementare…………………………………………………………………………………………….pag 11**
6. **Rezultate…………………………………………………………………………………………………….pag 13**
7. **Concluzii……………………………………………………………………………………………………..pag 13**
8. **Bibliografie………………………………………………………………………………………………….pag 14**
9. **Cerinta temei**

Consider an application **OrderManagement** for processing customer orders for a warehouse. Relational databases are used to store the products, the clients and the orders.   
  
Mod de rezolvare: Dupa conectarea la MySQL (baza de date), se va implementa o interfata grafica ce va cuprinde cele trei tabele necesare, impreuna cu implementarea operatiilor specifice : insert, update, delete, make order. Utilizatorul va putea vizualiza in timp real modificarea tabelelor si obiectele existente in ele.

1. **Obiectivul temei**
   1. **Obiectiv principal**Obiectivul principal al temei il reprezinta simularea cat mai reala a unui depozit, care este alcatuit din clienti, produse si comenzi. Asadar, un client poate comanda orice produs existent in depozit, de pe site-ul firmei : se selecteaza clientul dorit, produsul dorit, se verifica numarul necesar de bani pentru a putea achizitiona acel bun dupa care se da start comenzii.   
      Intreaga aplicatie se bazeaza pe afisarea tabelelor in interfata grafica si vizualizarea in timp real a modificarilor aduse acesteia.
   2. **Obiectiv secundar**

|  |  |  |
| --- | --- | --- |
| Obiectivele secundare | Descrierea obiectivelor | Capitolul |
| Diagrame si descrieri de use-case | Se vor propune posibile intrari de date (nu neaparat optime ) si se va explica motivul acestui lucru prin intermediul unui flow-chart. | Capitolul 2 |
| Structurile de date folosite | Se vor explica structurile de date utilizate si motivul pentru care au fost alese tocmai acestea. | Capitolul 3 |
| Deciziile de proiectare / implementare | Se va explica modul de gandire ales atat pentru interfata grafica cat si pentru implementare. | Capitolul 3 |
| Algoritmi utilizati | Se vor descrie algoritmii folositi pentru insert, delete cat si update, impreuna cu motivul utilizarii reflexiei. | Capitolul 3 |
| Pachete folosite | Se va descrie motivul alegerii pachetelor in care este structurata aplicatia. | Capitolul 3 |
| Rezultate | Se va discuta putin despre rezultatele obtinute in urma testarii mai multor intrari de date posibile. | Capitolul 5 |
| Concluzii | Se vor trage concluziile si vor fi subliniate lucrurile invatate in urma temei. | Capitolul 6 |

1. **Analiza problemei, modelare, scenarii, cazuri de utilizare**Use-case pentru posibilele date de intrare ale aplicatiei.   
   **1. Utilizatorul(actorul) introduce datele de intrare**  
    **Normal flow**: se introduc toate datele de intrare valide si se apasa butonul start.  
    **Bad flow**: utilizatorul introduce date gresite, partial gresite sau nu introduce date deloc, spre exemplu : acesta doreste sa insereze un nou client in baza de date, insa lasa field-ul dedicat numelui clientului gol.  
    **Tratarea bad flow-ului** / cazuri speciale: se va verifica fiecare field in parte, pus la dispozitia utilizatorului, astfel incat in momentul in care se gaseste o eroare in introducerea datelor de intrare ale programului, se va afisa un mesaj prin care acesta va fi instiintat despre acest lucru.

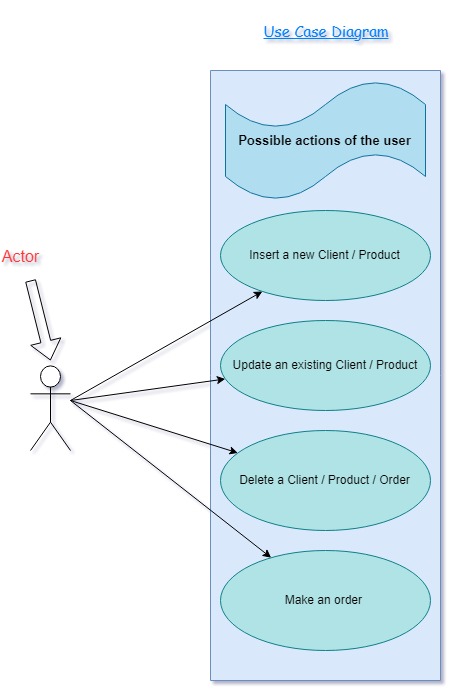
**2. Utilizatorul(actorul) apasa butonul insert / delete / update / add to order**  
 **Normal flow**: dupa introducerea corecta a datelor de intrare pentru a creea un nou obiect dorit, se apasa unul din butoanele **insert** / **update**. Pentru a sterge un obiect, se selecteaza un camp din tabel si se apasa butonul **delete**.

**Bad flow / cazuri speciale**: se apasa unul din butoanele respective dupa introducerea doar partiala a datelor de intrare ; se apasa butonul delete fara a selecta niciun obiect din tabela. Adaugarea unei comenzi, spre exemplu : produsul costa mai mult decat banii pe care ii detine clientul.

**Tratarea bad flow-ului / cazuri speciale:** pentru fiecare demers gresit al programului (cazurile descrise mai sus), am decis sa afisez un mesaj de eroare utilizatorului (fara ca aplicatia sa fie inchisa), care sa-l blocheze sa efectueze anumite lucruri “facute gresit”.   
 **3. Utilizatorul(actorul) analizeaza modificarile aduse bazei de date a warehouse-ului**

**Normal flow**: dupa finalizarea lucrului pe baza de date, utilizatorul urmareste modificarile aduse acesteia, verifica comenzile efectuate, etc.

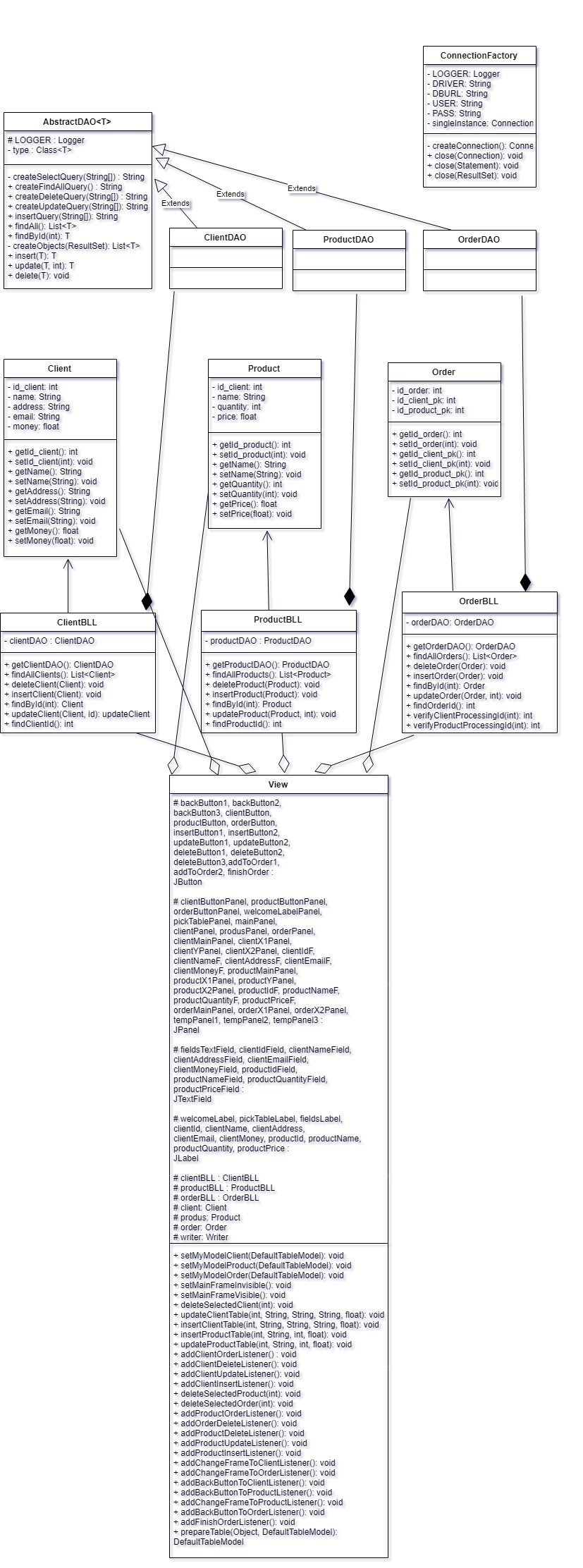
**Bad flow / cazuri speciale**: -  
 **Tratarea bad flow-ului / cazuri speciale :** -



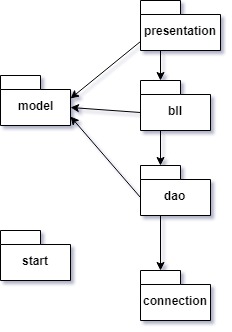
1. **Proiectare**

**Decizii de proiectare** Aplicatia este impartita in sase pachete : **presentation, bll, dao, connection, model, start.**   
 Pachetul **presentation** va contine clasa **View** care reprezinta intreaga interfata grafica. Am decis ca in locul folosirii mai multor clase diferite pentru a realiza interfata grafica, sa folosesc mai multe JFrame-uri pentru fiecare tabel in parte.  
 Pachetul **bll** va contine implementarea logica (un fel de controller) pentru fiecare clasa din pachetul **model**, in parte.   
 Pachetul **dao** este alcatuit din extinderile clasei **AbstractDAO**, folosita pentru a insera, sterge, updata, cauta dupa id, etc un obiect de orice tip (clasa **AbstractDAO** contine metodele de baza ce vor fi mostenite de aceste clase din pachetul dao, fapt pentru care, folosind reflexia, totul va fi floare la ureche.  
 Pachetul **connection** contine clasa **ConnectionFactory** ce face legatura cu aplicatia java si baza de date in MySQL.  
 Pachetul **model** este alcatuit din clasele de baza ale aplicatiei : **Client**, **Produs**, **Comanda**, prin intermediul carora putem adauga elemente noi in tabele.  
 Pachetul **start** contine doar clasa main() pentru ca programul sa poata fi verificat si rulat rapid.  
**Proiectare clase** Pentru inceput, am considerat ca cel mai util ar fi sa realizez conectarea cu succes la baza mea de date si sa pot efectua operatii simple pe aceste date: un select dintr-o tabela, o stergere dintr-o tabela, un update dintr-o tabela. Astfel, folosindu-ma de template-ul oferit la laborator, clasa ***ConnectionFactory*** realizeaza acest lucru, fapt care chiar a fost foarte util si de ajutor. Totodata, pentru a putea realiza query-uri simple, am folosit din plin clasa ***AbstractDAO***, care implementeaza si arata modul de creeare a unui query, impreuna cu datele dorite pentru a executa cu succes query-ul. Primul lucru a fost sa implementez operatiile de **insert**, **update**, **delete** si **findById**, intrucat pe acestea se bazeaza intreaga aplicatie. Am hotarat sa le fac in continuare in functie de un parametru generic, pentru a-mi usura munca mai tarziu.  
 In stransa legatura cu aceasta clasa AbstractDAO, se afla clasele ***ClientDAO***, ***ProductDAO*** si ***OrderDAO***. Acestea mostenind clasa abstracta, folosesc metodele deja definite in interiorul ei. Apeland o metoda cu un obiect de tipul **ClientDAO**, variabila instanta **type** a clasei **AbstractDAO**, impreuna cu metoda **getDeclaredFields**() returneaza toate variabilele instanta a clasei cu care a fost apelata metoda. Astfel, construirea generica a campurilor unui nou obiect de inserat spre exemplu, devine foarte usoara. In acelasi mod, valorile pentru inserarea unui nou obiect se obtin prin metoda **get** a clasei **Field** (metoda **getDeclaredFields**() returneaza un array de Field-uri).   
Asadar, avand la dispozitie aceste clase descrise mai sus, am reusit sa efectuez operatii simple pe baza de date cu succes.  
 Pe urma, intrucat aplicatia este organizata pe Layere, am decis sa implementez alte trei clase, cu rol de controller pentru fiecare tabel in parte. Clasele ***ClientBLL***, ***ProductBLL*** si ***OrderBLL*** implementeaza metode aferente metodelor din clasele lor specifice, ce extind **AbstractDAO**.  
Aceste trei clase contin fiecare cate un obiect de tipul tabelei aferente (obiect de tipul clasei ***Client / Product / Order*** : clase ce descriu proprietatile unui Client, a unui Produs si a unei Comenzi) .   
 Ultima clasa folosita, dar nicidecum cea din urma, intrucat este printre cele mai importante, o reprezinta clasa ***View***. Aceasta clasa implementeaza intreaga interfata grafica afisata utilizatorului pentru a putea vizualiza tabelele si pentru a putea opera pe ele.  
Clasa contine o metoda implementata generic, numita **prepareTable,** care creeaza generic un tabel prin intermediul unei liste de obiecte. Sunt creeate trei astfel de tabele, cate unul pentru fiecare clasa in parte : Client, Product, Order. Pe langa faptul ca aceasta clasa acopera interfata grafica, de la ea pornesc toate operatiile efectuate de catre utilizator pe tabelele implementate : am implementat cate un Listener pentru fiecare buton in parte, care mai departe, in functie de operatia dorita, verifica datele introduse, urmand ca pe urma sa fie folosite metode din Layerele imediat urmatoare.  
 Mai multe detalii despre interfata grafica vor fi discutate intr-o sectiune urmatoare.

**Diagrama UML de clase**



**Diagrama UML de pachete**

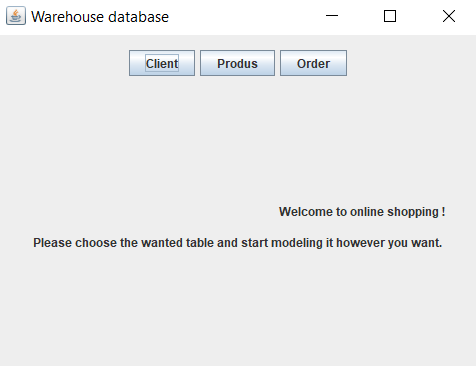


**Structuri de date** Structurile de date folosite pentru dezvoltarea aplicatiei sunt urmatoarele:  
 - Clasa Connection – pentru a realiza conectarea la baza de date  
 - Clasa singleInstance – pentru a creea un singur obiect de tip Connection  
 - Clasa PreparedStatement – pentru a putea seta valorile pentru inserarea unui nou obiect intr-un tabel (pentru crearea query-ului), folosind metoda setInt, setString, etc.  
 - StringBuilder – folosit pentru a creea string-ul folosind metoda implicita append  
 - PropertyDescriptor – ca sa putem obtine o metoda anume (methodName, class)  
 - Method – clasa folosita pentru a putea obtine metoda returnata de metoda getWriteMethod din clasa propertyDescriptor  
 - List – spre exemplu, pentru a putea retine toate elementele de tip Product, returnate de metoda findAll din clasa productDAO.

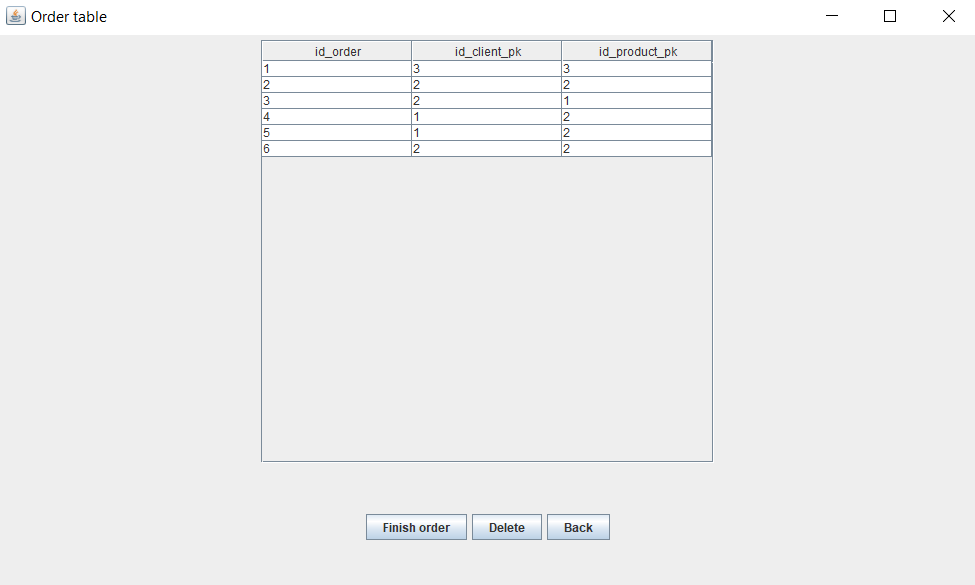
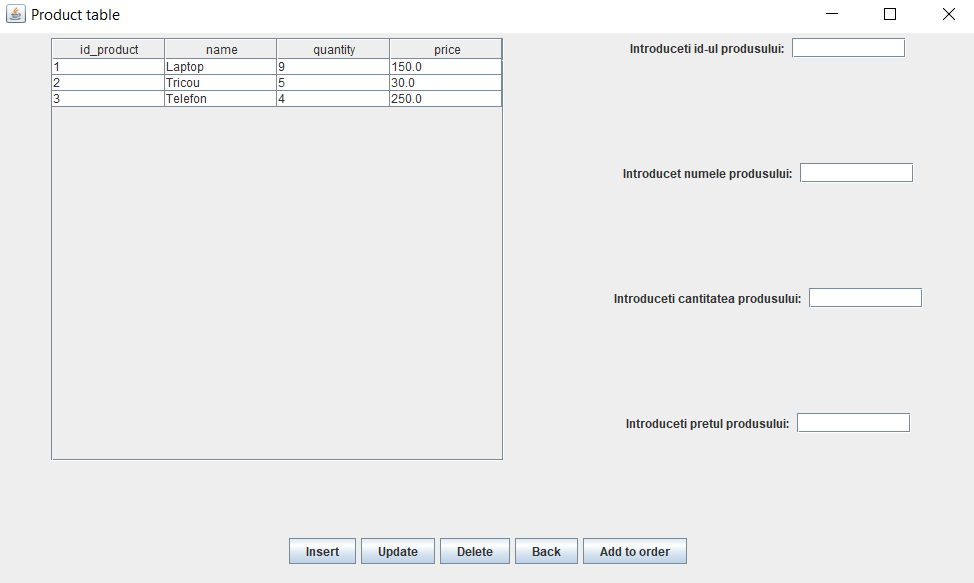
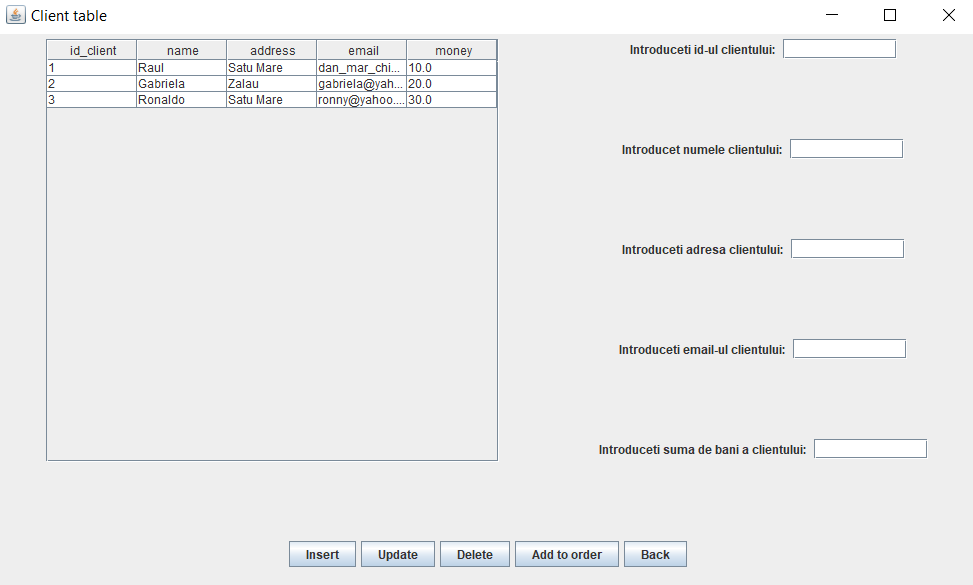
**Pachete folosite** Aplicatia este impartita pe 6 pachete. Pachetul **presentation,** contine clasa View care implementeaza intreaga interfata grafica si metodele aferente pentru operatiile executate de catre utilizator pe tabele. Acest lucru se face prin apel la clasele din pachetul **bll** (business logic), care este alcatuit din clasele **ClientBLL**, **ProductBLL**, **OrderBLL**. De aici, metodele implementate in fiecare dintre aceste clase apeleaza alte metode, din clasele ce se afla in pachetul **dao** (data access object), clase ce extind **AbstractDAO**. Pentru a putea realiza operatii pe tabelele din baza de date, aceste clase din pachetul dao vor avea nevoie de conexiunea la baza de date, lucru ce are loc in pachetul **connection**.  
Totodata, pachetul **model** contine clasele necesare aplicatiei, si anume Client, Product si Order.

**Algoritmi** Legat de algoritmii folositi, nu pot spune ca sunt chiar “algoritmi”, dar sunt moduri de rezolvare a unor cerinte si voi incerca sa-I descriu in limbaj natural.  
 Pentru operatiile de inserare in baza de date, utilizatorul va avea la dispozitie, in fiecare din cele trei tabele, niste field-uri pentru a introduce datele despre un posibil nou “obiect” de inserat. Aceste valori vor fi extrase si se va verifica corectitudinea lor cat si completarea totala a acestora. In caz afirmativ, persoana este introdusa in tabel, asadar si in baza de date. La nivel de implementare, generic se extrag field-urile din clasa cu care este apelata metoda insert, si se retine atat numele lor cat si valorile de introdus. Se apeleaza o metoda pentru constructia query-ului, se trimit valorile la acest query si se face un update la baza de date.  
 Pentru operatiile de stergere din baza de date, utilizatorul va trebui sa selecteze un obiect dintr-o tabela, iar la o apasare a butonului delete, daca si numai daca clientul / produsul respectiv, nu face parte dintr-o comanda ce are loc, se va sterge din tabel, deci si din baza de date.  
 Pentru operatiile de update, utilizatorul va introduce noile date pentru obiectul care se doreste a fi modificat, impreuna cu id-ul acestuia obligatoriu : se va cauta in baza de date obiectul cu id-ul respectiv, se vor updata datele pe acesta si va fi modificat atat in tabele, cat si in baza de date.  
 Pentru a realiza o comanda, utilizatorul este trebuie sa selecteze intai un client, pe urma un produs, pentru care va trebui sa spuna cantitatea dorita din acesta pentru a realiza comanda. Daca clientul are destui bani pentru a achita comanda, apasand butonul “Finish order” din tabelul Order, va fi realizata comanda si se vor updata datele folosite (clientului i se va scadea suma totala de bani, iar produsului i se va decrementa cantitatea).

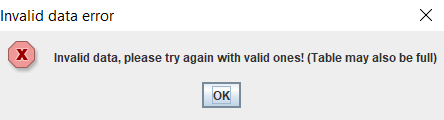
1. **Implementare** In principiu, metodele importante au fost discutate in subsectiunea **algoritmi**, asadar voi descrie **implementarea interfetei utilizator.** La prima rulare a aplicatiei, va aparea urmatoarea interfata grafica.   
    Utilizatorul va fi intampinat cu un mesaj de informare asupra aplicatiei si rolul acesteia, dupa care ii vor fi puse la dispozitie trei butoane, care la o simpla apasare, vor afisa un nou “window” reprezentand tabelul si campurile aferente acestuia. Din fiecare din aceste trei tabele, utilizatorul va avea la dispozitie un buton de “back” care il va aduce inapoi pe pagina principala, intrucat spre exemplu, la realizarea unei comenzi, acesta trebuie sa intre intai in tabelul pentru clienti, pe urma in tabelul pentru produse si in final in tabelul pentru comenzi.



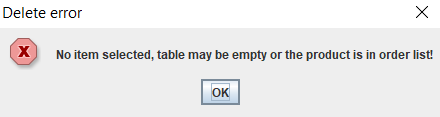
In urmatoarele poze, voi prezenta aspectul fiecarui tabel in parte.



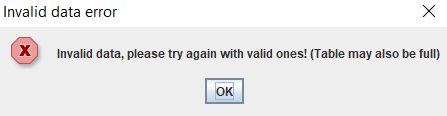
***Tratarea posibilelor exceptii***   
 **Introducerea de caractere de catre utilizator in locul pretului**



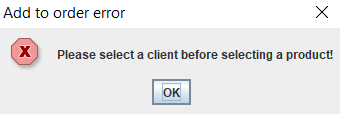
**Apasarea butonului de stergere fara a selecta un camp**



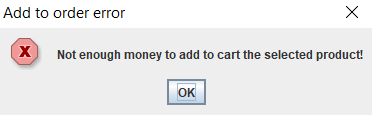
**Apasarea butonului de inserare folosind date invalide**



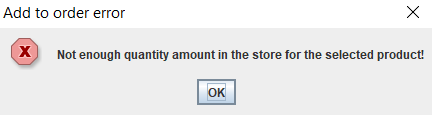
**Adaugarea la comanda a unui produs, fara a selecta un client**



**Creearea unui comenzi cand clientul nu are destui bani**



**Achizitionarea unui produs care nu mai este in stoc**



Interfata contine cate un ascultator pentru fiecare buton in parte, pentru a putea realiza operatia dorita de catre utilizator.

1. **Rezultate** Utilizatorul poate urmari in timp real modificarea tabelelor, introducerea de noi campuri in fiecare dintre aceste, stergerea anumitor campuri cat si adaugarea comenzilor. Totodata, intr-un fisier text am scris fiecare comanda in parte pentru ca acestea sa poata fi urmarite de catre utilizator
2. **Concluzii**

O prima concluzie ar fi ca nu am lucrat niciodata cu o baza de date “adevarata”, in sensul in care dintr-o interfata grafica sa pot modela fiecare tabel in parte asa cum imi doresc. Big up, cea mai frumoasa tema de pana acum, a fost o placere imensa sa lucrez la acest proiect si sa vad pas cu pas cum totul prinde forma.

1. **Bibliografie**

Generarea diagramelor : [www.draw.io](http://www.draw.io)