



Documentație

Tema 3: Aplicație pentru gestionarea unor clienți, produse și comenzi



Cuprins

1. Obiective	3
2. Analiza problemei	4
3. Proiectare	5
4. Implementare	7
5. Concluzii și dezvoltări ulterioare	13
6. Bibliografie	13



1. Obiective

1.1 Obiectiv principal

Luați în considerare o aplicație de gestionare a comenzilor pentru procesarea comenzilor clienților pentru un depozit.

Bazele de date relaționale ar trebui utilizate pentru a stoca produsele, clienții și comenzile.

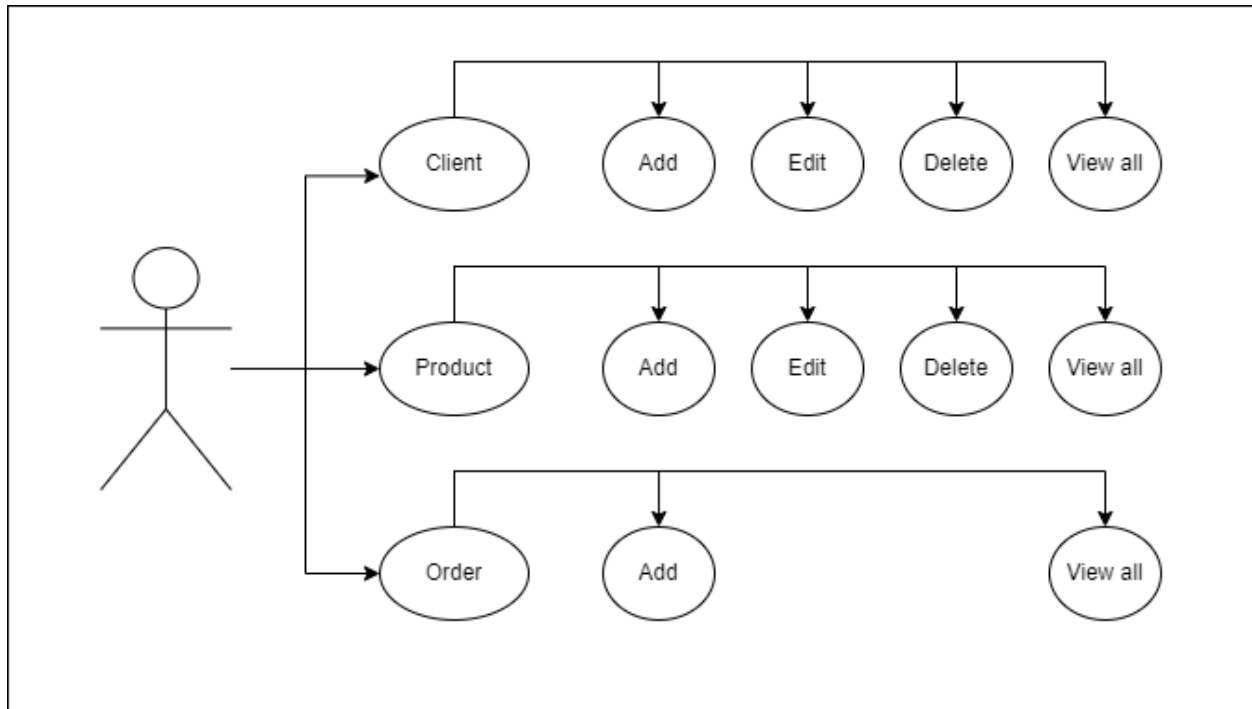
Aplicația ar trebui să fie proiectată conform modelului de arhitectură stratificată și ar trebui să utilizeze (minim) următoarele clase:

- Clasele model - reprezintă modelele de date ale aplicației
- Clasele businessLogic- conțin logica aplicației
- Clasele de prezentare - clase legate de interfața grafică a utilizatorului
- Clasele de dataAccess - clase care conțin accesul la baza de date

1.2 Obiective secundare

Obiectiv	Detaliere	Prezentat în capitolul
Dezvoltarea de <i>use cases</i> și scenarii	Reprezentarea diagramelor <i>use case</i> , precum și exemplificarea unor operații implementate	2
Structura de date și clasele	Explicarea și detalierea metodelor folosite în implementarea proiectului	3, 4
Implementarea soluției	Prezentarea funcționalităților interfeței din perspectiva utilizatorului	4

2. Analiza problemei



Urmărind diagrama *use case* de mai sus, utilizatorul are opțiunea de a alege asupra cărui obiect să facă operații: **Client**, **Produs** sau **Comandă**. Operațiile posibile sunt de adăugare a unui nou client/produs - în cazul comenzii, de a face o nouă comandă -, de editare a datelor unui client/produs anume, de ștergerea unui client/produs anume, și de vizualizare a tuturor clienților/produselor/comenzilor din baza de date.

Pentru toate operațiile în care se cer manual date de la utilizator, formatarea acestora este validată înaintea începerii execuției, pentru a nu apărea probleme pe parcurs. Spre exemplu, verificarea stocului disponibil pentru produsul cerut în momentul în care se forește a se plasa comanda, alertând clientul faptul că nu există disponibilă cantitatea cerută, dar este și anunțat de cantitatea actuală disponibilă, în eventualitatea în care dorește o comandă mai mică.



3. Proiectare

3.1 Introducere

Scopul principal al acestui proiect este de a dezvolta o aplicație care vizează gestiunea unei baze de date cu clienți, produse și comenzi.

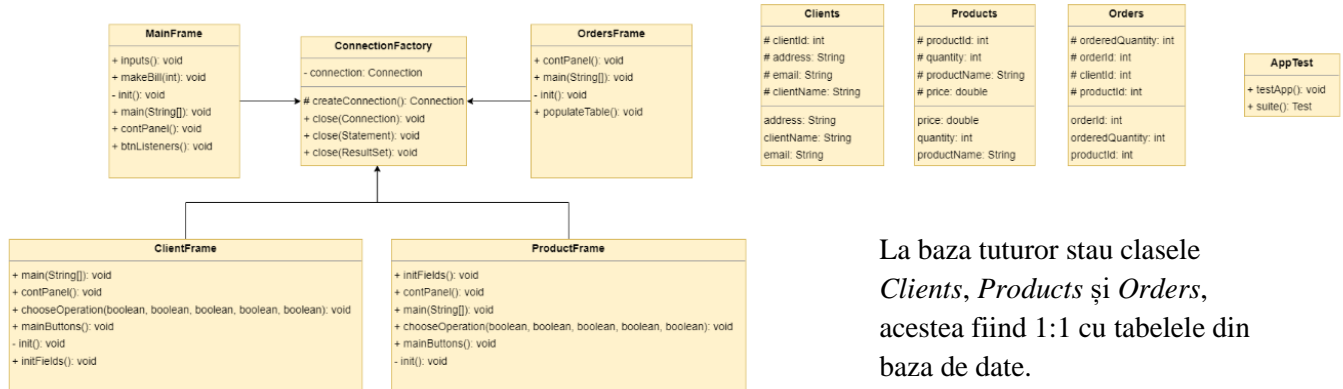
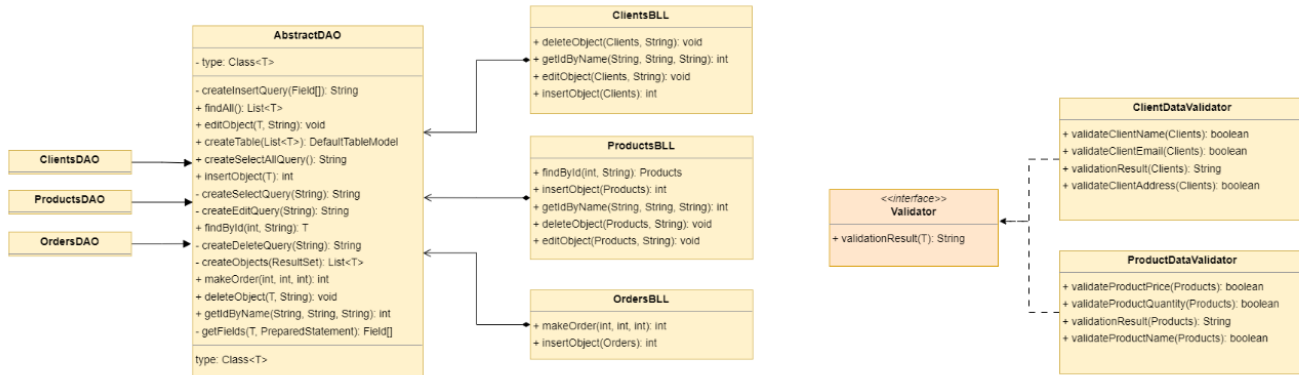
Proiectarea unei astfel de aplicații a avut ca origine modelul din viața reală: cum un depozit are în vedere să rețină și eventual să actualizeze datele despre clienții care fac comenzi, pentru a ști unde să livreze aceste comenzi, să rețină și să actualizeze în timp real datele despre produse, dacă mai sunt în stoc sau nu, și să afișeze mesaje de eroare corespunzătoare cu ceea ce se întâmplă. Această idee de actualizare are la bază o bază de date în MySQL cu ajutorul căreia am putut reține clienții, produsele și comenzile efectuate. Scrierea de cod MySQL a avut loc în MySQL Workbench, unde doar am creat tabelele necesare și făcut niște inserări manuale de test, restul codului fiind scris în Java.

Programul oferă atât o interfață grafică ușor de utilizat și intuitivă, cât și un rezultat într-un format familiar tuturor: fișierului PDF. După plasarea comenzii, se va genera automat un fișier .pdf care va conține datele despre comanda abia plasată, precum ziua curentă, datele de contact ale clientului și costul total al acesteia.

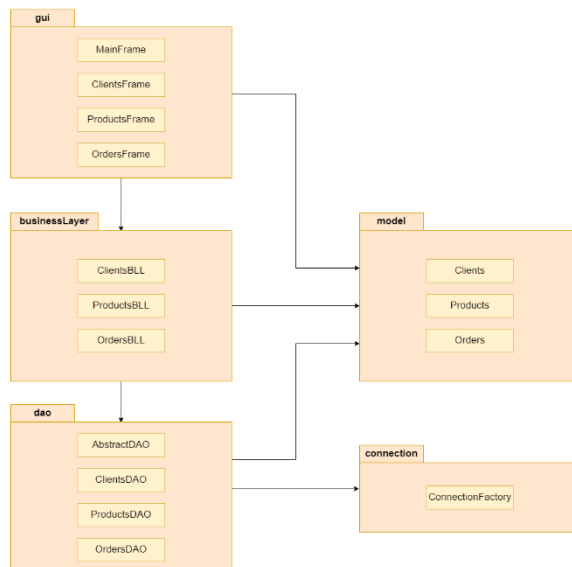


3.3 Diagrama de clase

În diagrama de clase următoare, sunt prezentate schematic clasele care au realizat această aplicație de gestionare a comenzilor, clienților și produselor.



La baza tuturor stau clasele *Clients*, *Products* și *Orders*, acestea fiind 1:1 cu tabelele din baza de date.



Repartizarea claselor în pachete



4. Implementarea

4.1 Descrierea claselor și metodelor

Datele sunt structurate în următoarele pachete și clase:

- **businessLayer**, cu clasele *ClientsBLL*, *ProductsBLL* și *OrdersBLL*
- **dataAccessLayer**, având mai departe:
 - **connection**, cu clasa *ConnectionFactory*
 - **dao**, cu clasele *AbstractDAO<T>*, *ClientsDAO*, *ProductsDAO* și *OrdersDAO* care extind fiecare clasa *AbstractDAO* cu parametrul T aferent
 - **validators**, cu interfața *Validator* și clasele care o implementează *ClientDataValidator* și *ProductDataValidator*
- **gui**, cu clasele *MainFrame*, *ClientFrame*, *ProductFrame* și *OrdersFrame*
- **model**, cu clasele *Clients*, *Products* și *Orders*

businessLayerLogic - BLL

Clasele *ClientsBLL*, *ProductsBLL* și *OrdersBLL* sunt responsabile de apelarea metodelor cerute de interfața grafică. Fiecare din ele creează un nou obiect de tipul DAO, pe care mai apoi apelează metodele respective.

Spre exemplu, dacă dorim inserarea unui noi client, vom apela metoda de `insertObject` asupra câmpului `clientsAbstractDAO`, și vom transmite parametrul primit. Astfel, nu vom apela niciodată în mod direct metodele din *AbstractDAO*, ci clasele BLL vor servi ca mediatori.

dataAccessLayer

→ dao

DAO (Data Access Object) reprezintă obiectul generic care furnizează metodele comune aplicate asupra bazei de date. Caracterul generic sau abstract (deși nu este o clasă abstractă) este dat de faptul că aceasta lucrează cu orice obiect de orice tip. Această clasă are implementate metode de extragere a anumitor date, inserare/editare/ștergere/vizualizare sau chiar găsirea unui obiect, în două situații: ne este dat id-ul și dorim câmpul de nume, sau invers. De asemenea, *AbstractDAO* mai oferă și alte metode pentru lucrul cu baze de date, și anume extragerea în mod universal a tuturor atributelor unui tabel, iar mai apoi, crearea unui tabel cu aceste atribute, și dacă tabelul este deja populat, va face la fel.

```
private Field[] getFields(T t, PreparedStatement preparedStatement)
public DefaultTableModel createTable(List<T> objectsList)
```



→ validators

Clasele de validatori au rolul de a verifica datele primite de la utilizator, deoarece și oamenii mai greșesc. Nu permite, spre exemplu, ca numele unui client să conțină altceva în afară de litere și eventual spații, sau la produs, cantitatea să fie altceva în afară de un număr întreg. Validatorul final ia în considerare fiecare validare anterioară separată a fiecărui câmp și le combină, astfel că și dacă un câmp e greșit, execuția nu continuă, și este semnalată eroarea.

```
@Override
public String validationResult(Clients client) {
    if(!validateClientName(client)) {
        return ("Nume client incorect!");
    }
    if(!validateClientAddress(client)) {
        return ("Adresa client incorecta!");
    }
    if(!validateClientEmail(client)) {
        return ("Email client incorect!");
    }
    return ("Date client valide!");
}
```

→ connection

Clasa *ConnectionFactory* realizează conexiunea cu baza de date. În interiorul acesteia avem toate datele pentru conectare: user-ul necesar, parola, url-ul bazei de date pentru a ști care mai exact trebuie accesată, și Driver-ul care realizează conexiunea.

gui - interfața cu utilizatorul

Clasele acestui pachet creează fiecare câte o fereastră în funcție de "obiectul" care dorește a fi accesat. Clasa *MainFrame* este cea care se deschide când începe execuția, iar ea rămâne deschisă cât timp programul rulează. Restul claselor sunt accesate prin apăsarea unor butoane aferente din *MainFrame*, iar închiderea acestora nu rezultă în închiderea programului, dat fiind faptul că acestea au opțiunea setată `setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE)`;

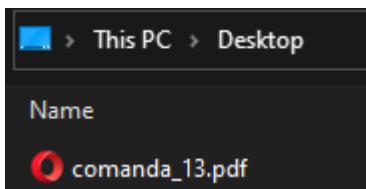
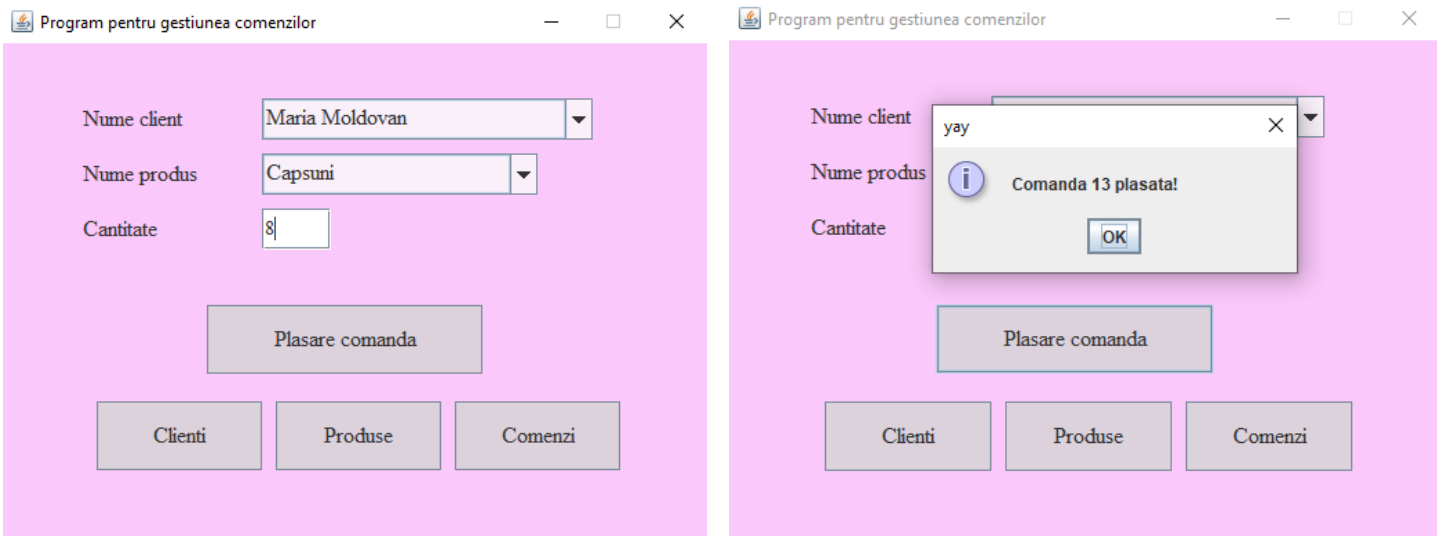
model

Clasele acestui pachet sunt o replică 1 la 1 cu tabelele din baza de date.

4.2 Interfața grafică

Utilizatorul trebuie să introducă anumite valori, fiecare având denumirea în stânga casetelor de input. La apăsarea butonului dorit, are loc validarea datelor. Erorile posibile sunt semnalate de un pop-up cu mesajul corespunzător.

Situație de succes: comanda a fost plasată. Programul ne spune și id-ul comenzii, iar apoi putem deschide tabela apăsând butonul *Comenzi*. Alternativ, putem deschide pdf-ul generat în path-ul specificat în metoda `makeBill`, implementată în *MainFrame*.



28-05-2023

Client: Maria Moldovan
Date de contact:
Str. Observatorului, Nr. 64
mariamoldovan@email.com

Pdf-ul are titlul **comanda_orderId.pdf**, conține data la care a fost plasată comanda, datele clientului, precum și datele comenzii: detaliile produsului și prețul total, în funcție de prețul/unitate de măsură și cantitatea comandată.

Produs	Cantitate	Pret
Capsuni	8	12.0

Cost total: 96.0



Ca și prezentare generală a ferestrelor pentru client și produs, aici sunt cele ale clientului, iar pentru produs singura diferență este numele câmpurilor, implementarea fiind la fel:

Am ales să implementez editarea și ștergerea cu comboBox-uri, deoarece este mai user-friendly doar să aleagă clientul după nume

clientId	clientName	address	email
1	Maria Moldovan	Str. Observatorului,...	mariamoldovan@...
2	Iulia Moldovan	Str. Fabricii de zah...	iuliamoldovan@e...
3	Catalin Robotin	Str. Orastie, Nr. 24	catalinrobotin@em...
4	Mihaela Moldovan	Str. Stefan cel Mare...	mihaela@a.a
6	Olivia A	Str Beius, nr. 3	olivia@email

Erori posibile în introducerea datelor:

The 'Produse' window has a yellow background and a toolbar with 'Adaugare' (Add), 'Editare' (Edit), 'Stergere' (Delete), and 'Afisare' (Display). The form contains three input fields: 'Nume' (Name) with 'Test', 'Cantitate' (Quantity) with '6.1', and 'Pret' (Price) with '123'. An 'Adauga produs' (Add product) button is at the bottom. An error dialog box is displayed over the 'Cantitate' field, stating: 'Error: Cantitatea trebuie sa fie numar intreg' (The quantity must be an integer).


The 'Cienti' window has a yellow background and a toolbar with 'Adaugare' (Add), 'Editare' (Edit), 'Stergere' (Delete), and 'Afisare' (Display). The form contains three input fields: 'Nume' (Name) with 'Test', 'Adresa' (Address) which is empty, and 'Email' with 'test@te'. An 'Adauga client' (Add client) button is at the bottom. An error dialog box is displayed over the 'Adresa' field, stating: 'Error: Campul de adresa nu poate fi gol!' (The address field cannot be empty!).

The 'Program pentru gestiunea comenzilor' window has a pink background. It contains three input fields: 'Nume client' (Client name) with 'Maria Moldovan', 'Nume produs' (Product name) with 'Banane', and 'Cantitate' (Quantity) with '100'. There are buttons for 'Comenzi' (Orders) and 'Comenzi' (Orders). An error dialog box is displayed, stating: 'Error: Nu avem in stoc 100 Banane. Disponibil: 67' (We do not have 100 Bananas in stock. Available: 67).



Tabela de comenzi la care se ajunge prin apăsarea butonului *Comenzi* are la spate un query MySQL puțin mai complex față de cel din findA11() folosit pentru clienți și produse, deoarece am vrut ca acest tabel să fie mai ușor de urmărit, nu doar numere pentru id-uri:

```
select Orders.orderId, Clients.clientName, Products.productName, Orders.orderedQuantity as quantity,
       Products.price, (Orders.orderedQuantity * Products.price) as totalPrice
from Orders
      join Clients on Orders.clientId = Clients.clientId join Products on Orders.productId = Products.productId order by orderId asc;
```

 Comenzi — □ ×

Order ID	Client Name	Product Name	Quantity	Price	Total Price
1	Mihaela Moldovan	Cola	1	2.3	2.3
2	Maria Moldovan	Banane	2	6.0	12.0
3	Maria Moldovan	Mere	4	4.0	16.0
4	Catalin Robotin	Capsuni	1	12.0	12.0
5	Iulia Moldovan	Banane	1	6.0	6.0
6	Catalin Robotin	Ciocolata	50	11.0	550.0
7	Mihaela Moldovan	Cola	5	2.3	11.5
8	Maria Moldovan	Paine	2	7.0	14.0
9	Iulia Moldovan	Pere	1	5.5	5.5
10	Catalin Robotin	Ciocolata	55	11.0	605.0
11	Olivia A	Rosii	1	6.22	6.22
12	Olivia A	Rosii	98	6.22	609.56
13	Maria Moldovan	Capsuni	8	12.0	96.0

5. Concluzii și dezvoltări ulterioare

În concluzie, am luat deciziile de implementare cele mai eficiente, în limita timpului disponibil. Am acumulat multe cunoștințe noi, și aș avea câteva dezvoltări ulterioare sau îmbunătățiri.

Una din ele ar fi posibilitatea de a adăuga mai multe produse la o comandă. Aceasta a fost una din ideile mele inițiale, cu un câmp extra de verificare a statusului comenzii, însă a fost mai dificil față de cât mă așteptam. O altă idee ar fi ca pdf-ul generat să fie trimis automat prin email clientului, deoarece avem datele deja furnizate. O ultimă idee, probabil cea mai utilă și care se leagă de prima, ar fi posibilitatea de editare sau ștergere a comenzii, deoarece un client se poate răzgândi, și cât timp comanda nu e încheiată, ar trebui să aibă posibilitatea aceasta.

6. Bibliografie

- <https://dsrl.eu/courses/pt/>
- <https://introcs.cs.princeton.edu/java/3Ooop/>
- <https://www.javatpoint.com/java-string-format>
- <https://stackoverflow.com>
- https://gitlab.com/utcn_dsrl/pt-reflection-example - pentru înțelegerea conceptelor de Reflection și LayeredArchitecture, folosite în acest proiect
- <https://www.javatpoint.com/java-get-current-date>
- <https://itextpdf.com> – pentru generarea de PDF
- <https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html> - pentru studierea scrierii unor JavaDoc comments

