DOCUMENTAȚIE

TEMA 3

Nume Student: IRINI KARINA

Grupa: 30229

CUPRINS:

1. Obiectivul temei

Obiectivul principal al acestei teme este de a proiecta și implementa o aplicație numită "Gestionare Comenzi" (Orders Management) care să permită gestionarea comenzilor clienților pentru un depozit. Aceasta implică utilizarea unor baze de date relaționale pentru stocarea informațiilor despre produse, clienți și comezile plasate. Aplicația va fi proiectată conform modelului architectural stratificat (layered architecture pattern) și va include clasele necesare pentru modelele de date, logica aplicației, interfața utilizatorului și accesul la date.

Scopul principal al aplicației este de a oferi funcționalități pentru adăugarea, actualizarea și ștergerea produselor, clienților și comenzilor, precum și plasarea de comenzi noi și vizualizarea facturilor. De asemenea, interfața grafică trebuie să permit introducerea datelor de intrare și vizualizarea informațiilor din bazele de date într-un mod intuitiv și eficient.

Pentru a îndeplini obiectivul principal al temei, trebuie să ne axăm și pe obiectivele secundare. Este important să începem cu analiza problemei și identificarea cerințelor. Această etapă ne ajută să ne concentrăm asupra funcționalităților de bază ale aplicației de gestionare a comenzilor, aspect detaliat în capitolul următor. Proiectarea aplicației, descrisă în cel de-al treilea capitol, este esențială pentru a începe implementarea programului. Aceasta constă în introducerea datelor în bazele de date corespunzătoare, editarea, ștergerea și vizualizarea acestora, precum și a unei interfețe grafice. Este necesară realizarea acesteia într-un mod intuitiv, pentru a asigura o utilizare cât mai plăcută a software- ului. Ultimul pas, dezvoltat în capitolul cinci, asigură buna funcționare a aplicației, fiind reprezentat de testarea codului creat.

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

Cerințele funcționale sunt cele care descriu funcționalitățile pe care programul trebuie să le îndeplinească, iar cerințele non funcționale sunt cele care descriu caracteristicile programului.

Cerința problemei constă în realizarea unei aplicații de gestionare a comenzilor pentru procesarea comenzilor clienților. Pentru aceasta, este necesară utilizarea unei baze de date relaționale pentru a stoca informațiile despre produse, clienți și comenzi. Structura aplicației trebuie să respecte următoarele pachete:

1. Pachetul Model: Acesta conține clasele model care reprezintă modelele de date ale aplicației. Atributele acestor clase trebuie să fie identice cu cele din baza de date, iar numele claselor trebuie să coincidă cu numele tabelelor.
2. Pachetul BusinessLogic: Acesta conține clasele care implementează logica aplicației. Aceste clase manipulează obiecte din pachetul Model. Aici vor fi definite metode pentru crearea, modificarea și ștergerea comenzilor.
3. Pachetul Presentation: Acesta conține clasele care alcătuiesc interfața utilizator, inclusiv partea de vizualizare și conectarea dintre logica aplicației și partea vizuală.
4. Angajații vor putea vizualiza datele, introduce noi elemente, le pot modifica sau șterge, precum și plasa comenzi noi. În acest pachet vor fi incluse componentele necesare pentru realizarea acestor operații.

5. Pachetul DataAccess: Acesta conține clasele responsabile de accesul la baza de date. Aceste clase furnizează operații CRUD (Create, Read, Update, Delete) pentru entități. De exemplu, clasa ProductDAO oferă metode pentru salvarea, căutarea și ștergerea produselor din baza de date.

Proiectarea arhitecturii aplicației Orders Management poate fi abordată într-un mod simplu, însă detaliile specifice și tehnologiile utilizate vor influența design-ul final. Câteva aspecte importante ale aplicației sunt: performanța, reacția rapidă la interacțiunea utilizatorului, ușurința în utilizare și executarea rapidă a comenzilor.

Funcționarea aplicației poate fi descrisă prin mai multe cazuri de utilizare. Utilizatorul aplicației de gestionare a comenzilor este un angajat, iar aplicația trebuie să ofere următoarele funcționalități:

- Introducerea de clienți noi: Angajatul poate introduce informații despre clienți noi în aplicație.
- Modificarea datelor unui client: Angajatul are posibilitatea de a modifica informațiile existente despre un client.
- Ștergerea unui client: Angajatul poate șterge înregistrarea unui client din aplicație.
- Vizualizarea datelor despre clienți: Angajatul poate vizualiza informațiile despre clienți stocate în aplicație.

Aceleași operații sunt disponibile și pentru gestionarea produselor:

- Introducerea de produse noi: Angajatul poate adăuga informații despre produse noi în aplicație.
- Modificarea datelor unui produs: Angajatul poate modifica informațiile existente despre un produs.
- Ștergerea unui produs: Angajatul poate elimina un produs din aplicație.
- Vizualizarea datelor despre produse: Angajatul poate vizualiza informațiile despre produsele stocate în aplicație.

În ceea ce privește plasarea comenzilor, angajatul poate selecta un produs, un client și cantitatea dorită, astfel creând o nouă comandă în aplicație. Procesul de plasare a comenzii va genera automat o factură, care poate fi accesată doar de persoanele autorizate cu acces la baza de date a magazinului sau depozitului.

Caz de utilizare: Adăugare produs

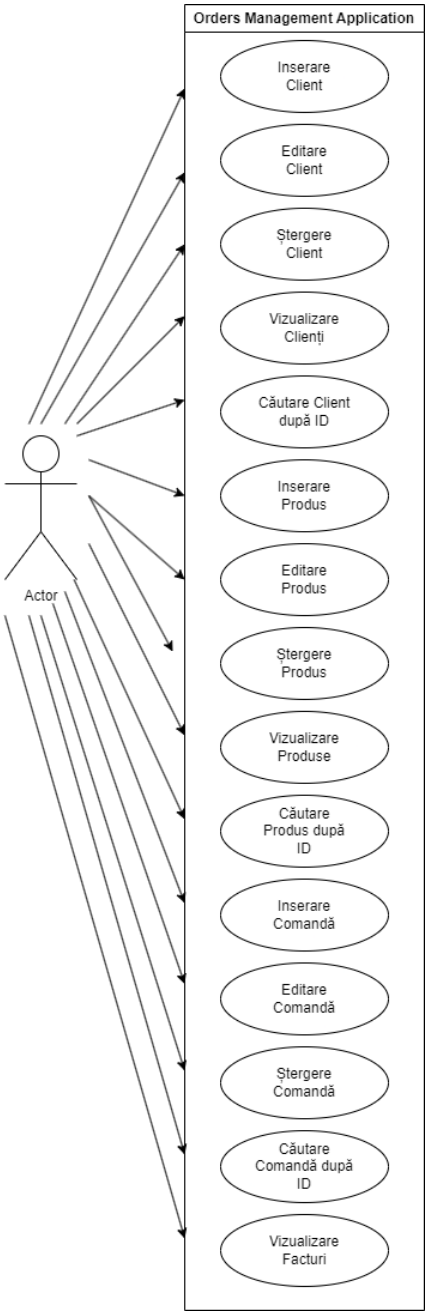
Actor principal: angajat

Scenariul principal de succes:

1. Angajatul selectează opțiunea de adăugare a unui produs nou.
2. Aplicația afișează un formular în care trebuie introduse detaliile produsului.
3. Angajatul introduce numele produsului, prețul și stocul curent.
4. Angajatul apasă pe butonul "Adăugare".
5. Aplicația stochează datele produsului în baza de date și afișează un mesaj de confirmare.

Secvență alternativă: Valori nevalide pentru datele produsului

- Utilizatorul introduce o valoare negativă pentru stocul produsului.
- Aplicația afișează un mesaj de eroare și solicită utilizatorului să introducă un stoc valid.
- Scenariul revine la pasul 3.



3. Proiectare

Programarea orientată pe obiecte (OOP) este o paradigmă de programare ce se bazează pe conceptul de obiecte, componente software care incorporează atât atributele cât și operațiile care se pot efectua asupra câmpurilor. Unul dintre principiile fundamentale ale OOP este încapsularea datelor, ascunderea detaliilor unui obiect și, astfel, protejarea informațiilor. Prin utilizarea principiilor OOP se beneficiază de o dezvoltare mai rapidă și mai eficientă a software-ului dorit, cu o structură clară și organizare bună a datelor. Acest aspect implică identificarea obiectelor și a interacțiunilor dintre acestea, definirea claselor și a relațiilor dintre ele, implementarea metodelor și a datelor pentru fiecare clasă, precum și testarea codului.

Pentru acest proiect, am utilizat 4 clase care reprezintă modelul aplicației și corespund tabelelor din baza de date. Aceste clase sunt fundamentale pentru tema proiectului și sunt specifice tehnicii de programare alese.

De asemenea, am utilizat și tehnica reflection în acest proiect, care ne permite să lucrăm cu clase, interfețe, atribute și metode la runtime, fără a cunoaște numele lor în prealabil. Cu ajutorul acestei tehnici, putem instantia noi obiecte, apela metode și obține sau schimba valoarea atributelor. Reflection-ul este extrem de util în acest caz. De exemplu, l-am folosit pentru a accesa baza de date, creând metode de reflection care să realizeze operațiile de inserare, ștergere, editare și interogare specifice, care pot fi utilizate pentru orice tabel din baza de date. Astfel, nu a fost nevoie să implementăm aceste metode de fiecare dată.

Proiectul a fost structurat în 4 pachete:

- Pachetul Model: Acesta conține clasele model care reprezintă modelele de date ale aplicației.
- Pachetul BusinessLogic: Aici se găsesc clasele care implementează logica aplicației. Aceste clase manipulează obiecte din pachetul Model.
- Pachetul Presentation: Acesta conține clasele care compun interfața utilizator, inclusiv partea de vizualizare și conexiunea între logica și aspectul vizual. Angajatul are astfel posibilitatea de a vizualiza datele, introduce elemente noi, le poate modifica sau șterge, și poate plasa comenzi noi.
- Pachetul DataAccess: Aici se găsesc clasele care accesează baza de date. Acestea furnizează operații CRUD (Create, Read, Update, Delete) asupra entităților.
- Pachetul Connection: Acesta se ocupă de realizarea conexiunii cu baza de date.

Proiectul este de asemenea împărțit în mai multe pachete:

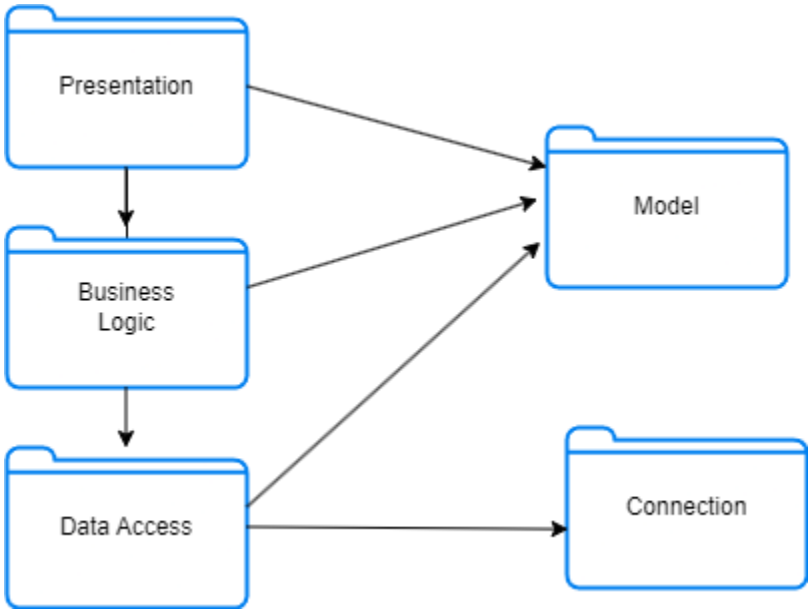
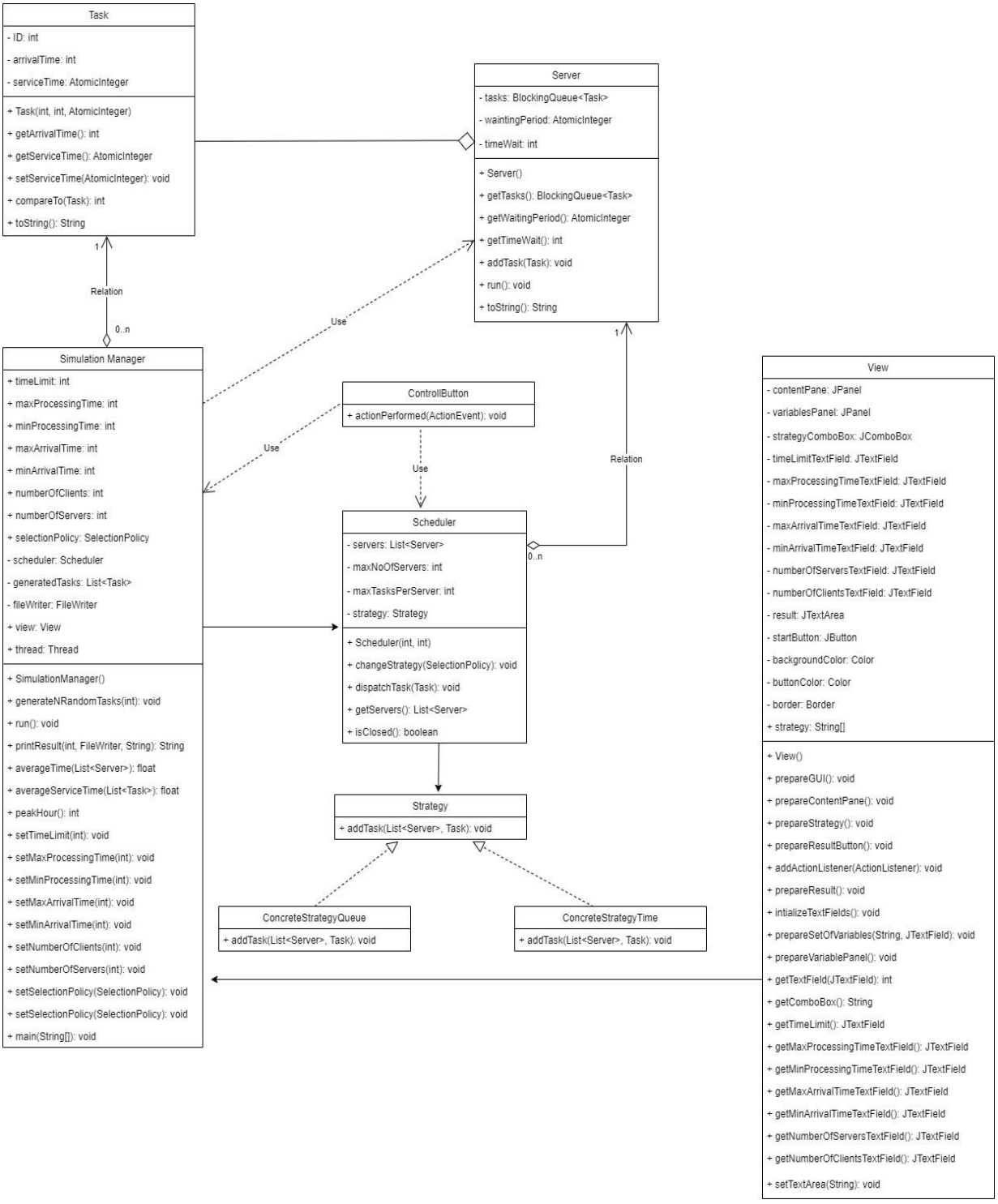


Diagrama UML de clase, din care se poate observa relațiile dintre acestea, este următoarea:



4. Implementare

Clasa Client: Această clasă are 4 atribute: nume, id (unic), adresă și email. Metodele din această clasă sunt doar settere și gettere. Clasa Client corespunde tabelului "client" din baza de date.

Clasa Product: Această clasă are 4 atribute: nume, id (unic), număr produse și preț. Metodele din această clasă sunt doar settere și gettere. Clasa Product corespunde tabelului "product" din baza de date.

Clasa Command: Această clasă are 4 atribute: id (unic), id client, id produs și preț. Metodele din această clasă sunt doar settere și gettere. Clasa Command corespunde tabelului "command" din baza de date.

Clasa Bill: Această clasă este o clasă imutabilă construită folosind Java record. Are 3 atribute: id order, preț, id client și id. Clasa Bill corespunde tabelului "bill".

Clasa AbstractDAO: Această clasă conține metode principale pentru operațiile CRUD pe baza de date. Este o clasă generică în Java și servește ca bază pentru clasele DAO specifice diferitelor entități din aplicație. Clasa implementează metode generale pentru accesul la date într-o bază de date relațională. Utilizează reflecția pentru a obține tipul entității pe care o gestionează. Metodele acestei clase permit interogarea și manipularea datelor într-un mod general și facilitează implementarea claselor DAO specifice pentru diverse entități. Clasele specifice care extind clasa AbstractDAO sunt: ProductDAO, ClientDAO și OrderDAO.

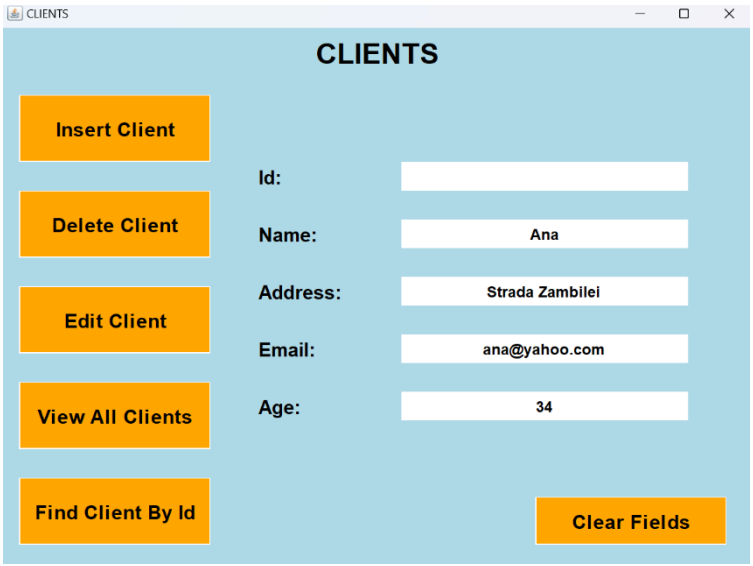
Clasele ClientBLL, ProductBLL și OrderBLL manipulează obiectele din pachetul model. Aici sunt apelate metodele din clasele DAO corespunzătoare.

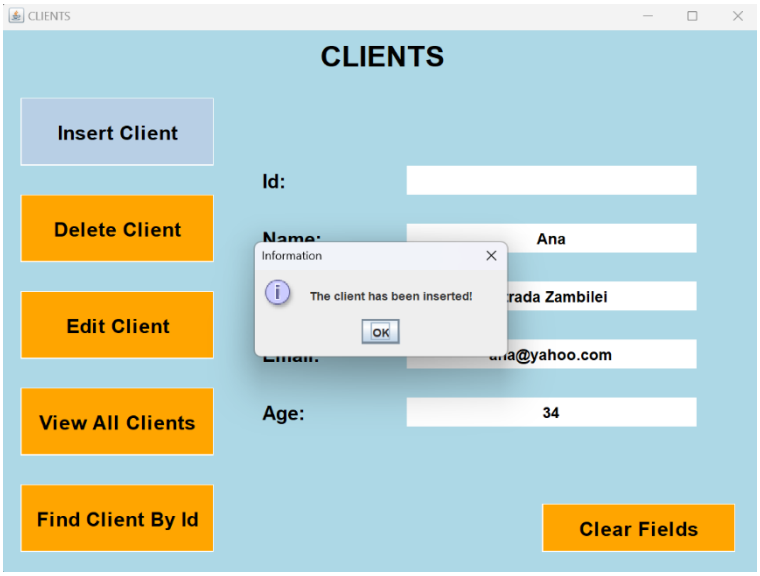
Pentru realizarea interfeței grafice au fost folosite multiple clase pentru a oferi funcționalitățile dorite. Interfața permite adăugarea, ștergerea și modificarea clienților și produselor, precum și plasarea de comenzi. Aceste operații necesită utilizarea de label-uri, textfield-uri și butoane.

5. Rezultate

După rularea aplicației, vom obține rezultatele dorite. Pentru a asigura funcționalitatea corectă, am efectuat teste pentru fiecare scenariu posibil, inclusiv inserarea și editarea multiplelor obiecte, ștergerea acestora și vizualizarea rezultatelor.

Rezultatele obținute în urma rulării includ următoarele:





CLIENTS

id	name	address	email	age
1	Maria	Strada Lalelelor	maria@gmail.com	45
2	Cristian	Strada Republicii	cristian.cristi@ya...	25
3	Carmen	Strada Manastur	carmen@gmail.co...	45
4	Daniel	Strada Cocosul de..	daniel.lofi@hotma...	38
5	Ana	Strada Zambilei	ana@yahoo.com	34

Modificările realizate în interfața aplicației pot fi observate și în baza de date MySQL. Dacă am adăugat un nou produs, acesta va fi vizibil în tabelul corespunzător, iar dacă l-am șters, acesta va fi eliminat din tabel.

6. Concluzii

Această temă mi-a oferit oportunitatea de a exersa conceptele de Programare Orientată pe Obiect (POO) și de a învăța diverse tehnici și concepte noi pentru mine, precum legătura dintre Java și bazele de date relaționale, metodele Reflection și elementele SQL.

Dezvoltând această aplicație, am avut multe de învățat și de explorat. Am înțeles importanța structurării aplicației în componente distincte, precum Model, Business Logic, Presentation și Data Access. Am realizat importanța folosirii claselor model pentru a reprezenta datele într-un mod care să corespundă structurii tabelelor din baza de date. Utilizarea clasei AbstractDAO pentru a efectua operațiile CRUD (Create, Read, Update, Delete) asupra entităților a fost de asemenea o lecție valoroasă. Am utilizat tehnica Reflection pentru a obține informații despre entitățile pe care le gestionăm și pentru a apela metodele corespunzătoare în funcție de operația dorită.

De asemenea, am învățat cât de importantă este testarea și validarea funcționalităților implementate, asigurându-ne că aplicația funcționează corect în diferite scenarii și că rezultatele obținute sunt în conformitate cu așteptările.

Această experiență ne-a ajutat să înțelegem mai bine structura și organizarea unui proiect software și să dobândim cunoștințe esențiale în dezvoltarea aplicațiilor de gestionare a comenzilor și a bazelor de date. Deși inițial am întâmpinat dificultăți în înțelegerea construirii metodelor utilizând reflection, la finalul temei simt că am acumulat o cantitate semnificativă de cunoștințe și că am învățat multe lucruri noi.

Această temă a reprezentat un pas important în dezvoltarea mea ca programator și m-a inspirat să continui să explorez și să învăț mai multe despre aceste concepte și tehnici în viitor.

7. Bibliografie

1. https://dsrl.eu/courses/pt/
2. https://app.diagrams.net/
3. https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html
4. https://dzone.com/articles/layers-standard-enterprise
5. https://www.baeldung.com/javadoc
6. https://dzone.com/articles/layers-standard-enterprise