# DOCUMENTAȚIE

## TEMA 3

NUME STUDENT: Lupou Krisztián-Róbert
GRUPA: 30226

# CUPRINS

# 1. Obiectivul temei

## 1.1. Obiectiv principal

Obiectivul principal al acestui proiect constă în dezvoltarea și implementarea unui sistem de gestionare a comenzilor. Sistemul are rolul de a automatiza și eficientiza procesul de gestionare a comenzilor, de la înregistrarea acestora până la actualizarea și ștergerea lor din baza de date. Scopul este de a oferi o interfață intuitivă și funcționalități variate pentru a facilita gestionarea eficientă a comenzilor, inclusiv adăugarea, actualizarea, căutarea și ștergerea. Sistemul va avea o interfață grafică pentru utilizatori, permițându-le să lucreze într-un mod simplu și eficient cu comenzile.

## 1.2. Obiective secundare

- Implementarea unui sistem de gestiune a produselor, care să permită adăugarea, actualizarea și ștergerea produselor din baza de date.
- Implementarea unui sistem de gestiune a clienților, care să permită adăugarea, actualizarea și ștergerea informațiilor despre clienți din baza de date.
- Dezvoltarea unei funcționalități de căutare, care să permită utilizatorilor să găsească rapid și eficient comenzile în funcție de diferite criterii, cum ar fi ID-ul comenzii, ID-ul clientului sau ID-ul produsului.
- Implementarea unui sistem de validare și verificare a datelor introduse în sistem, pentru a asigura integritatea și coerența acestora.
- Realizarea unei interfețe grafice intuitive și prietenoase pentru utilizatori, care să faciliteze interacțiunea cu sistemul și să ofere o experiență plăcută utilizatorilor.

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

## 2.1. Cerințe funcționale și non-funcționale

Cerințele funcționale:

- Adăugarea unui client/produs/comandă în aplicație
- Actualizarea informațiilor unui client/produs/comandă
- Ștergerea unui client/produs/comandă din aplicație
- Afișarea tuturor clienților/produselor/comenzilor într-un tabel
- Validarea datelor introduse de utilizator în interfața grafică
- Utilizarea tehnicii de reflexie pentru popularea și actualizarea unui tabel cu date dinamice
- Crearea unei interfețe de meniu pentru navigarea între diferitele funcționalități ale aplicației
- Asigurarea conexiunii cu baza de date și gestionarea resurselor asociate

Cerințe non-funcționale:

- Interfața utilizator trebuie este intuitivă, prietenoasă și ușor de utilizat
- Sistemul este fiabil, robust și funcționează fără erori majore sau căderi

- Performanță optimă și capabil să răspundă rapid și eficient la operațiile de adăugare, actualizare și ștergere

## 2.2. Cazuri de utilizare

1. Gestionarea detaliilor clientului:

- Angajatul accesează interfața de administrare a sistemului.
- Angajatul vizualizează lista clienților înregistrați și detaliile acestora, cum ar fi numele, adresa, vârsta și adresa de e-mail.
- Angajatul poate adăuga un nou client în sistem, introducând informațiile necesare (nume, vârstă, adresă, adresă de e-mail).
- Angajatul poate modifica detaliile unui client existent (nume, vârstă, adresă, adresă de e-mail).
- Angajatul poate șterge un client din sistem, în cazul în care acesta nu mai este relevant sau solicită eliminarea datelor sale.

2. Actualizarea stocului de produse:

- Angajatul accesează interfața de administrare a sistemului.
- Angajatul vizualizează lista produselor disponibile împreună cu stocurile și prețurile acestora.
- Dacă un produs este în curs de epuizare sau are stocul insuficient, angajatul poate actualiza cantitatea disponibilă.
- Angajatul actualizează stocul produsului și salvează modificările în sistem.

# 3. Proiectare

## 3.1. Structuri de date

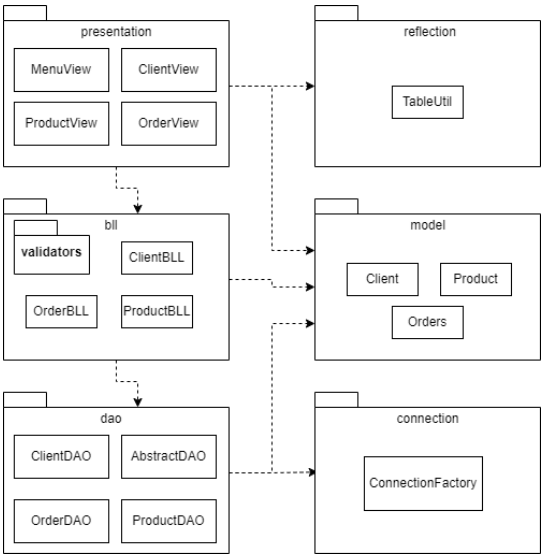
- ArrayList/List: Această colecție (numită și container) este un obiect care grupează mai multe elemente într-o singură unitate.

- JTable: Este o clasă din biblioteca Java Swing și reprezintă o componentă grafică utilizată pentru a afișa date într-un format tubular, similar cu un tabel. Structura principală folosită de JTable este matricea bidimensională de obiecte care reprezintă datele afișate în tabel.

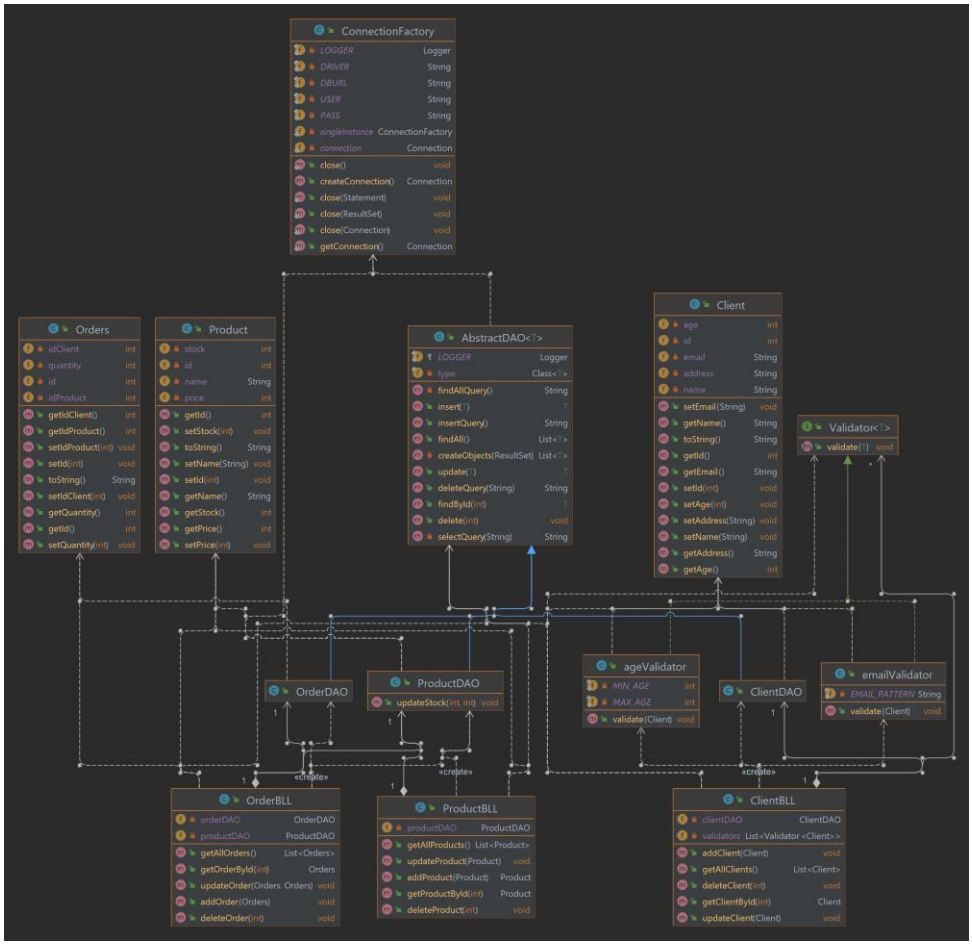
- ResultSet: Este o interfață din Java care reprezintă o structură de date tubulară utilizată pentru a reține rezultatele unei interogări către o bază de date relațională.

- StringBuilder: Utilizat pentru a construi șirurile de caractere pentru diferitele interogări SQL.

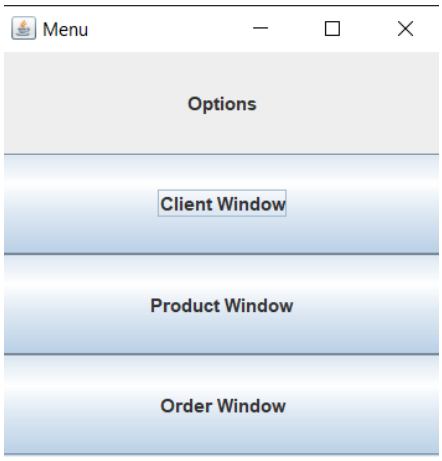
## 3.2. Diagrama UML a pachetelor



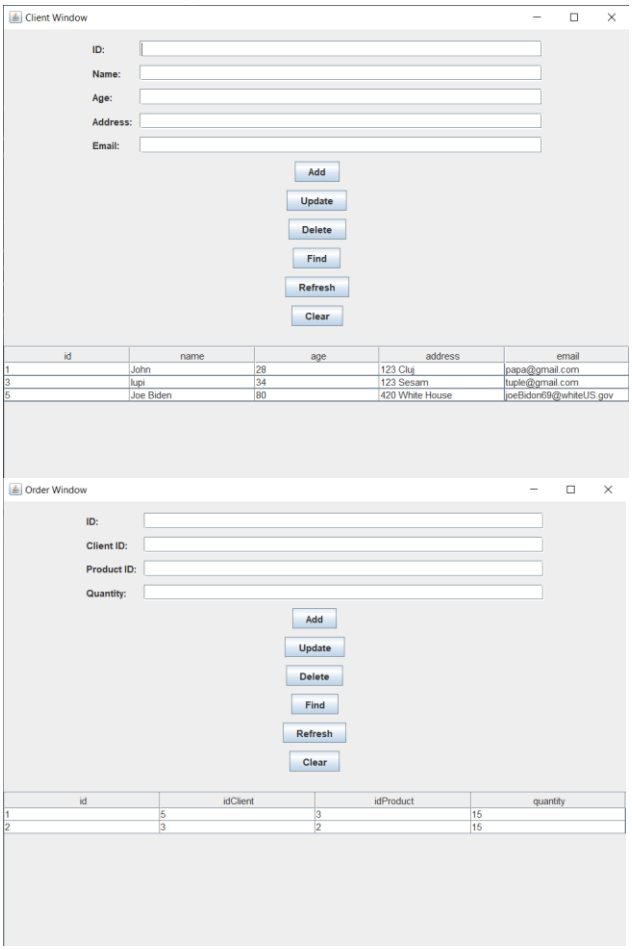

## 3.3. Diagrama UML

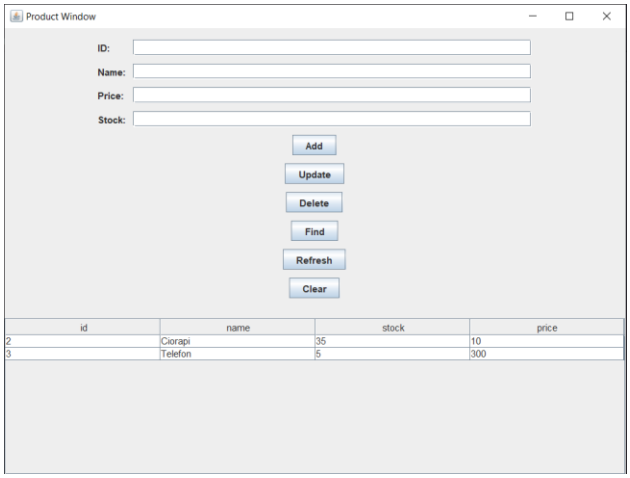


## 3.4. Interfața utilizator




Prima interfață este reprezentaată de o fereastră cu 3 butoane. Aceste butoane reprezintă cele 3 tabele pe care se vor efectua operațiile de adăugare, actualizare, căutare și ștergere.



Fiecare dintre cele 3 ferestre are butoanele pentru adăugare (Add), update (Update), ștergere (Delete), căutare după ID (find), reîmprospătare a tabelului (Refresh) și un Clear pentru golirea căsuțelor de tip text. La toate cele 3 ferestre sub se afișează tabelul corespunzător ferestrei.

# 4. Implementare

- Clasele Cient/Product/Order:

  În cadrul aplicației, clasa `Client` este utilizată pentru a reprezenta informațiile despre clienți, cum ar fi numele, vârsta, adresa și adresa de email.
  Clasa `Product` este utilizată pentru a reprezenta produsele disponibile în magazin. Atributele sale, cum ar fi numele, stocul și prețul, sunt utilizate pentru a afișa informații despre produse și a gestiona stocul.
  Clasa `Orders` este utilizată pentru a gestiona comenzile plasate de clienți. Atributele sale, cum ar fi id-ul comenzii, id-ul clientului, id-ul produsului și cantitatea, permit înregistrarea și urmărirea comenzilor.

- Clasele AbstractDAO<T>/ClientDAO/ProductDAO/OrderDAO:

  Clasa AbstractDAO<T> este responsabilă de funcționalitățile de bază comune ale operațiilor CRUD, precum găsirea tuturor obiectelor, găsirea după ID, inserarea, actualizarea și ștergerea unui obiect. Aceasta este o clasă generică, unde parametrul de tip T reprezintă tipul obiectelor gestionate de DAO.

  Acestea sunt principalele metode și funcționalități ale clasei AbstractDAO:

  1. Constructorul AbstractDAO: Aceasta initializează clasa cu ajutorul reflecției pentru a obține tipul obiectului gestionat (type).
  2. Metode pentru construirea interogărilor SQL:
     - selectQuery(String field): Construiește o interogare de tip SELECT pentru a selecta obiectele care au o valoare specificată într-un anumit câmp.
     - findAllQuery(): Construiește o interogare de tip SELECT pentru a selecta toate obiectele din tabela corespunzătoare tipului obiectului gestionat.
     - insertQuery(): Construiește o interogare de tip INSERT pentru a insera un obiect în baza de date.
     - deleteQuery(String field): Construiește o interogare de tip DELETE pentru a șterge obiectele care au o valoare specificată într-un anumit câmp.
  3. Metode de bază pentru operațiile CRUD:
     - findAll(): Returnează o listă cu toate obiectele din baza de date corespunzătoare tipului obiectului gestionat.
     - findById(int id): Returnează obiectul cu ID-ul specificat.
     - insert(T t): Inserează un obiect în baza de date.
     - delete(int id): Șterge obiectul cu ID-ul specificat.
     - update(T t): Actualizează un obiect în baza de date.

4. Metode de ajutor pentru crearea și manipularea obiectelor:
    - createObjects(ResultSet resultSet): Transformă rezultatul unei interogări într-o listă de obiecte.

Clasa ProductDAO extinde clasa AbstractDAO<Product> și adaugă o metodă suplimentară numită updateStock care permite actualizarea stocului pentru un produs specificat. Această metodă este specifică pentru clasa Product și oferă o funcționalitate suplimentară pentru a actualiza stocul produsului în baza de date.

- Clasa ConnectionFactory:

Clasa ConnectionFactory este utilizată pentru a crea și gestiona conexiunea la baza de date. Are câteva metode importante:

- createConnection(): Această metodă este responsabilă de crearea conexiunii la baza de date. Utilizează informațiile de configurare precum driverul, URL-ul bazei de date, numele utilizatorului și parola pentru a stabili o conexiune. În cazul în care apare o excepție în timpul conectării, se înregistrează un mesaj de avertizare și se afișează stiva de urmărire a excepțiilor.
- getConnection(): Este o metodă statică care returnează conexiunea creată prin apelarea metodei createConnection(). Această metodă permite obținerea conexiunii în alte clase, fără a fi nevoie să se creeze o instanță a clasei ConnectionFactory.
- metodele close(): Aceste metode sunt responsabile de închiderea conexiunii, declarațiilor și setului de rezultate. Acestea iau ca argument obiectele asociate (de exemplu, o conexiune, o declarație sau un set de rezultate) și încearcă să le închidă. În cazul în care apare o excepție în timpul închiderii, se înregistrează un mesaj de avertizare.

- Clasele ClientBLL/ProductBLL/OrderBLL:

Aceste clase reprezintă stratul de logică de business al aplicației și se ocupă de prelucrarea și validarea datelor, interacțiunea cu baza de date și asigurarea conformității cu regulile specifice domeniului de activitate. Pachetul "business logic" conține următoarele clase:

- ClientBLL: Această clasă gestionează operațiunile specifice pentru entitatea "Client". Are o listă de validatoare care verifică diverse aspecte ale obiectelor de tip Client, cum ar fi validitatea adresei de email sau vârsta. Clasa utilizează obiectul ClientDAO pentru a accesa și manipula datele despre clienți în baza de date. Metodele sale includ: getAllClients() pentru a obține toți clienții, getClientById() pentru a obține un client după ID, addClient() pentru a adăuga un client nou, updateClient() pentru a actualiza un client existent și deleteClient() pentru a șterge un client.
- ProductBLL: Această clasă gestionează operațiunile specifice pentru entitatea "Product". Utilizează obiectul `ProductDAO` pentru a accesa și manipula datele

despre produse în baza de date. Metodele sale includ: `getAllProducts()` pentru a obține toate produsele, `getProductById()` pentru a obține un produs după ID, `addProduct()` pentru a adăuga un produs nou, `updateProduct()` pentru a actualiza un produs existent și `deleteProduct()` pentru a șterge un produs.

- OrderBLL: Această clasă gestionează operațiunile specifice pentru entitatea "Order". Folosește obiectele OrderDAO și ProductDAO pentru a accesa și manipula datele despre comenzile și produsele asociate în baza de date. Metodele sale includ: getAllOrders() pentru a obține toate comenzile, getOrderById() pentru a obține o comandă după ID, addOrder() pentru a adăuga o comandă nouă (actualizând și cantitatea de produs disponibil), deleteOrder() pentru a șterge o comandă (actualizând și cantitatea de produs disponibil) și updateOrder() pentru a actualiza o comandă (actualizând și cantitatea de produs disponibil).

- Clasa TableUtil:

Clasa TableUtil este o clasă utilitară care oferă o metodă statică populateTable() pentru popularea unui obiect JTable cu date dintr-o listă de obiecte de tipul T. Metoda populateTable() primește ca parametri un obiect JTable și o listă de obiecte de tipul T. În interiorul metodei, se obține modelul tabelului utilizând table.getModel(). În cazul în care tabelul are un model de tip DefaultTableModel, acesta este salvat într-o variabilă de tip DefaultTableModel numită model. Se resetează numărul de rânduri și numărul de coloane ale modelului la zero utilizând model.setRowCount(0) și model.setColumnCount(0), astfel încât să se poată adăuga noile date în tabel. Dacă lista de obiecte nu este goală, se procedează la popularea tabelului. Se obține clasa obiectelor din lista utilizând list.get(0).getClass(). Aceasta este salvată în variabila clazz. Se obțin toate câmpurile (fields) declarate ale clasei clazz utilizând clazz.getDeclaredFields(). Aceste câmpuri reprezintă coloanele tabelului. Se iterează prin fiecare câmp și se adaugă numele câmpului ca și nume de coloană în modelul tabelului utilizând model.addColumn(fieldName). Pentru fiecare obiect t din listă, se creează un vector de obiecte rowData de dimensiunea numărului de câmpuri. Se iterează prin fiecare câmp și se accesează valoarea acestuia pentru obiectul t utilizând field.get(t). Această valoare este salvată în vectorul rowData. Se adaugă vectorul rowData ca rând în modelul tabelului utilizând model.addRow(rowData). Astfel, tabelul este populat cu datele din lista de obiecte.

Clasa TableUtil este utilă în scenariile în care se dorește afișarea datelor dintr-o listă de obiecte într-un tabel grafic, precum un tabel Swing. Metoda populateTable() simplifică procesul de creare și populare a tabelului cu date, eliminând nevoia de a scrie cod repetitiv.

- Clasele MenuView/ClientView/ProductView/OrderView:

Clasa MenuView reprezintă o interfață de meniu în care sunt afișate trei butoane: clientWin, productWin și orderWin. Această clasă deschide fereastra meniului și ascultă acțiunile utilizatorului. La apăsarea butoanelor, se deschid ferestrele corespunzătoare: ClientView, ProductView sau OrderView.

Clasa ClientView reprezintă o fereastră în care utilizatorul poate interacționa cu operațiile CRUD (Create, Read, Update, Delete) pentru entitatea Client. Aici sunt afișate câmpurile de introducere pentru ID, nume, vârstă, adresă și email ale unui client. Utilizatorul poate adăuga, actualiza, șterge, găsi sau reîmprospăta datele clientului. De asemenea, există o tabelă pentru afișarea tuturor clienților.

Clasa ProductView reprezintă o fereastră similară cu ClientView, dar pentru operațiile CRUD pe entitatea Product. Utilizatorul poate adăuga, actualiza, șterge, găsi sau reîmprospăta datele unui produs. Există câmpuri pentru ID, nume, preț și cantitate, precum și o tabelă pentru afișarea tuturor produselor.

Clasa OrderView reprezintă o fereastră pentru gestionarea comenzilor. Utilizatorul poate adăuga, actualiza, șterge sau găsi o comandă. Sunt afișate câmpuri pentru ID-ul comenzii, ID-ul clientului, ID-ul produsului și cantitatea. Există, de asemenea, o etichetă care indică dacă stocul produsului este suficient pentru comandă. De asemenea, există o tabelă pentru afișarea tuturor comenzilor.

Toate cele trei ferestre (ClientView, ProductView și OrderView) extind clasa JFrame și utilizează obiecte BLL (Business Logic Layer) corespunzătoare (ClientBLL, ProductBLL și OrderBLL) pentru a gestiona operațiile cu baza de date. Există, de asemenea, o clasă utilitară TableUtil care ajută la popularea tabelului cu date. Aceste clase fac parte din pachetul "view" și reprezintă interfețele grafice ale aplicației care permit utilizatorului să interacționeze cu datele.

# 5. Rezultate

Funcționalitatea aplicației este perfectă, prezentând o interfață grafică intuitivă pentru introducerea parametrilor și afișând în mod corespunzător rezultatele dorite. În primul rând, utilizatorul se va confrunta cu meniul principal al aplicației, unde va putea alege interfața grafică pe care dorește să o utilizeze. Alegerile disponibile includ interfețele pentru Client, Produs și Comandă. Fiecare interfață va dispune de câmpuri de introducere a datelor specifice, urmate de o serie de butoane care permit adăugarea, actualizarea, ștergerea și căutarea după Id, precum și un buton pentru reîmprospătarea tabelului. În cazul comenzilor, se va asigura că atât Id-ul clientului, cât și Id-ul produsului există și că cantitatea disponibilă în stoc este suficientă. În momentul plasării comenzii, cantitatea corespunzătoare va fi dedusă din stocul total al produsului respectiv. Aspectul interfeței poate fi vizualizat în secțiunea 3.4.

# 6. Concluzii

În concluzie, prin această temă am aprofundat foarte multe cunoștințe despre lucrul cu o bază de date în Java, și anume MySql. Am învățat cum să mă conectez la o bază de date și cum să implementez operațiile de bază, precum crearea, actualizarea, ștergerea și citirea datelor. De asemenea, am învățat să implementez o metodă reflection pentru popularea tabelelor cu date.

Proiectul poate fi dezvoltat în viitor prin adăugarea funcționalităților avansate, precum sortarea și filtrarea datelor, implementarea unui sistem de autentificare și autorizare, integrarea cu alte aplicații sau servicii terțe, implementarea unui sistem de raportare și analiză, precum și îmbunătățirea interfeței utilizator pentru a oferi o experiență mai intuitivă și atractivă. Aceste adăugiri ar contribui la îmbunătățirea eficienței, extinderea funcționalității și satisfacția utilizatorilor.

# 7. Bibliografie


1. Connect to MySql from a Java application:

   o https://www.baeldung.com/java-jdbc

   o http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/

2. Layered architectures:

   o https://dzone.com/articles/layers-standard-enterprise

3. Reflection in Java:

   o http://tutorials.jenkov.com/java-reflection/index.html

4. ASSIGNMENT 3 – SUPPORT PRESENTATION (PART 1) - https://www.dsrl.eu/courses/pt/materials/PT2023_A3_S1.pdf

5. ASSIGNMENT 3 – SUPPORT PRESENTATION (PART 2) - https://www.dsrl.eu/courses/pt/materials/PT2023_A3_S2.pdf