# DOCUMENTATIE

## TEMA *3*

NUME STUDENT: Jarda Adina-Ionela
GRUPA:grupa 30222

# CUPRINS

## 1. Obiectivul temei

În dezvoltarea unei aplicații, trecem prin mai multe etape cruciale pentru a ne asigura că produsul final îndeplinește cerințele și oferă o experiență plăcută utilizatorilor.

Prima etapă este proiectarea interfeței grafice. Aceasta este faza în care definim cum arată și cum funcționează aplicația noastră, pentru a fi intuitivă și ușor de folosit.

Următoarea etapă implică implementarea funcționalităților de bază, precum adăugarea, modificarea și ștergerea clienților și produselor. Aici ne asigurăm că datele sunt gestionate corect și că interacțiunea utilizatorului cu aplicația decurge fără probleme.

După aceea, integrăm un sistem de plasare a comenzilor, care să faciliteze procesul de cumpărare și să gestioneze automat stocul de produse și facturarea.

Testarea și validarea sunt esențiale pentru a ne asigura că totul funcționează așa cum ne așteptăm. Prin testarea fiecărei componente și a interacțiunilor dintre ele, identificăm și rezolvăm problemele înainte ca aplicația să fie lansată.

În final, analizăm rezultatele și tragem concluzii privind beneficiile și limitele aplicației, oferind sugestii pentru îmbunătățiri viitoare. Această etapă ne ajută să înțelegem mai bine cum putem face aplicația noastră să fie mai bună și mai eficientă în viitor.

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

### *Cerințe funcționale:*

- Adăugarea, modificarea și ștergerea clienților și produselor.
- Plasarea comenzilor și generarea automată a facturilor.

### *Cerințe non-funcționale:*

- Interfața intuitivă și ușor de folosit.
- Scalabilitatea sistemului pentru un volum mare de date.
- Securitatea datelor prin autentificare și autorizare.

### *Cazurile de utilizare:*

- Adăugare client: Utilizatorul introduce datele, iar clientul este adăugat după validare.
- Modificare client: Datele sunt actualizate după introducerea noilor informații.
- Ștergere client: Confirmarea acțiunii este solicitată înainte de ștergere.
- Adăugare produs: Datele sunt validate și produsul este adăugat în baza de date.
- Modificare produs: Datele sunt actualizate după introducerea noilor informații.
- Ștergere produs: Confirmarea acțiunii este solicitată înainte de ștergere.
- Plasare comandă: Verificarea disponibilității în stoc și înregistrarea comenzii.

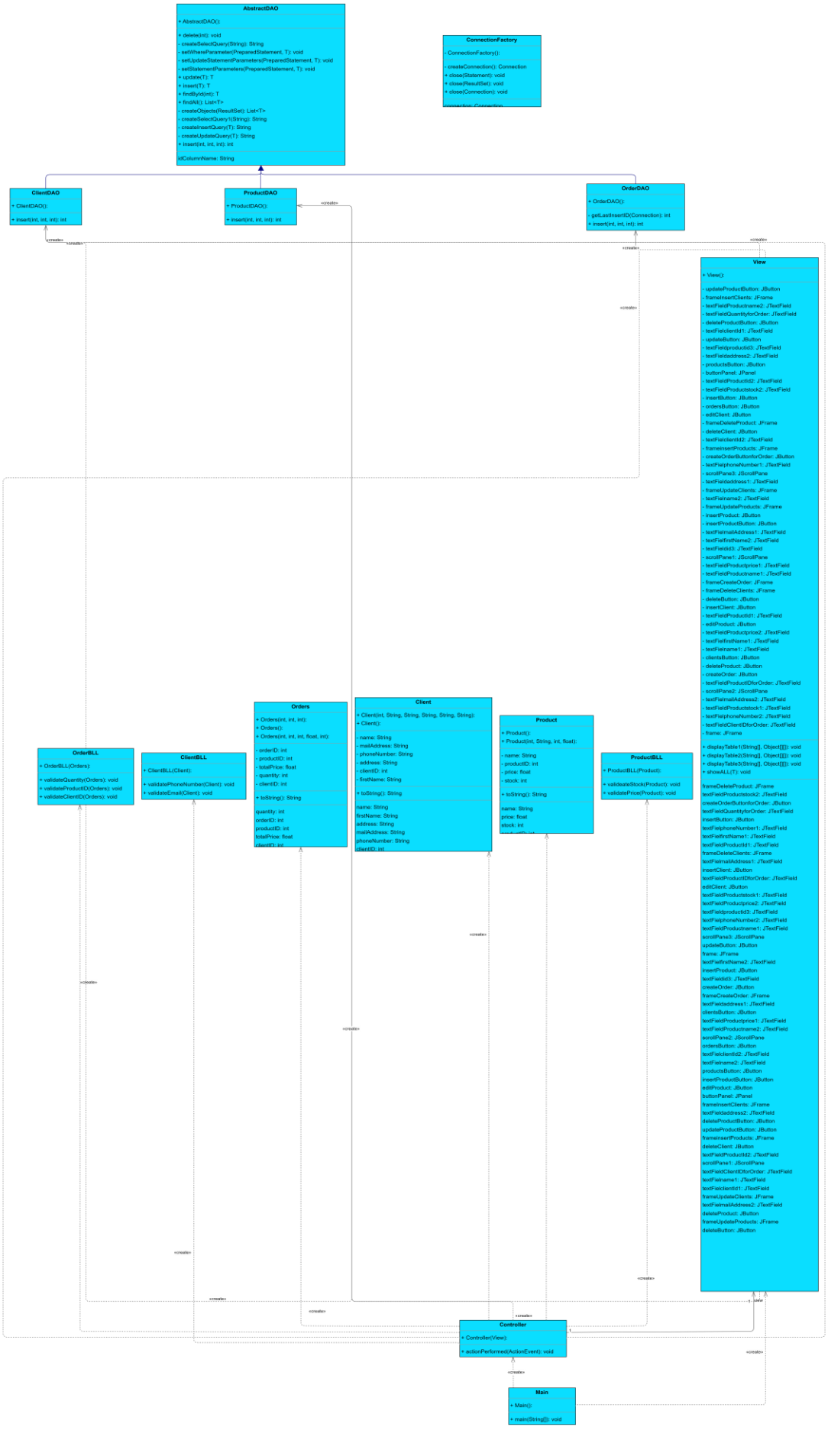
## 3. Proiectare

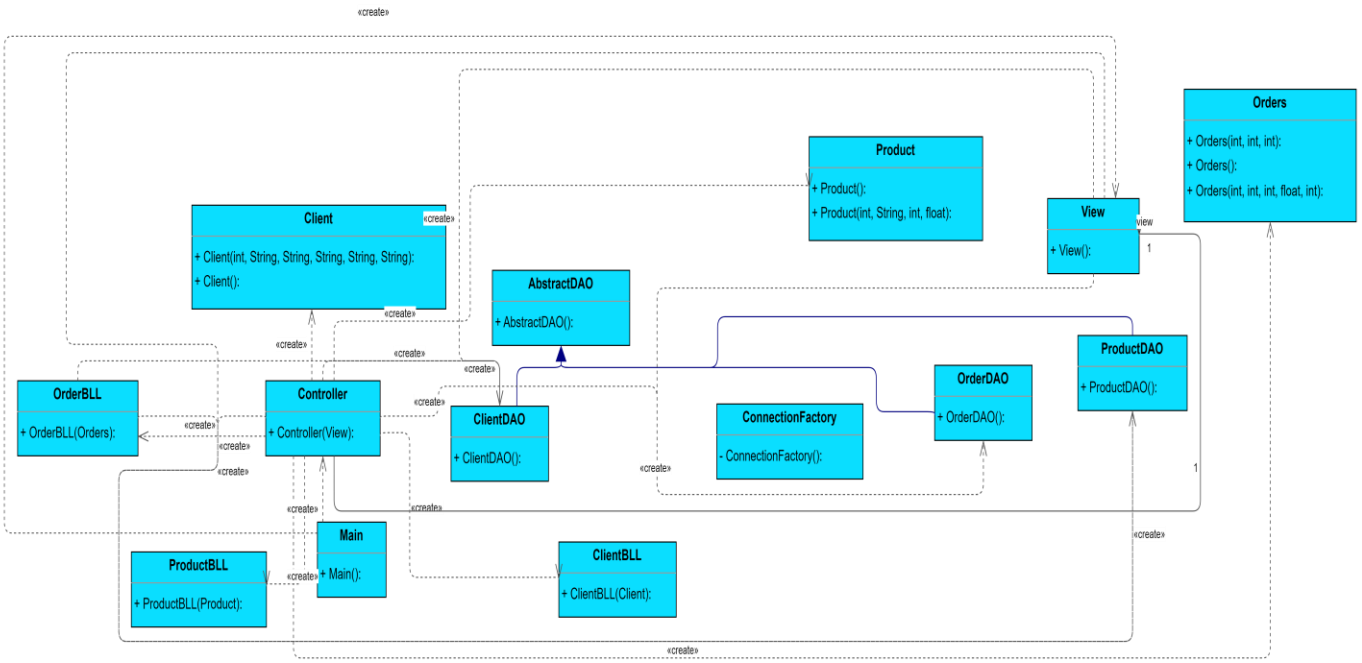
### *Proiectarea Orientată pe Obiecte a Aplicației*

Aplicația este proiectată folosind principii solide de OOP pentru a organiza și gestiona eficient entitățile și operațiile acestora. Aceasta urmărește să ofere o structură modulară și ușor de întreținut.

### *Diagrama UML de Clase*

Diagrama UML de clase oferă o imagine de ansamblu asupra structurii și relațiilor dintre clasele aplicației. Ea cuprinde clasele principale precum Client, Product, Order, dar și clasele de acces la date ClientDAO, ProductDAO, OrderDAO, împreună cu clasele de logică a afacerilor ClientBLL, ProductBLL, OrderBLL. Relațiile dintre aceste clase sunt modelate pentru a reflecta fluxul de date și operațiile desfășurate.


AbstractDAO
+ AbstractDAO():
+ delete(int): void
- createSelectQuery(String): String
- setWhereParameter(PreparedStatement, T): void
- setUpdateStatementParameters(PreparedStatement, T): void
- setStatementParameters(PreparedStatement, T): void
+ update(T): T
+ insert(T): T
+ findById(int): T
+ findAll(): List<T>
- createObjects(ResultSet): List<T>
- createSelectQuery(String): String
- createInsertQuery(T): String
- createUpdateQuery(T): String
+ insert(int, int, int): int
idColumnName: String
ConnectionFactory
- ConnectionFactory():
- createConnection(): Connection
+ close(Statement): void
+ close(ResultSet): void
+ close(Connection): void
connection: Connection
ClientDAO
+ ClientDAO():
+ insert(int, int, int): int
ProductDAO
+ ProductDAO():
+ insert(int, int, int): int
OrderDAO
+ OrderDAO():
- getLastInsertID(Connection): int
+ insert(int, int, int): int
«create»
View
+ View():
- updateProductButton: JButton
- frameInsertClients: JFrame
- textFieldProductname2: JTextField
- textFieldQuantityforOrder: JTextField
- deleteProductButton: JButton
- textFieldclientId1: JTextField
- updateButton: JButton
- textFieldproductid3: JTextField
- textFieldaddress2: JTextField
- productsButton: JButton
- buttonPanel: JPanel
- textFieldProductId2: JTextField
- textFieldProductstock2: JTextField
- insertButton: JButton
- ordersButton: JButton
- editClient: JButton
- frameDeleteProduct: JFrame
- deleteClient: JButton
- textFieldclientId2: JTextField
- frameinsertProducts: JFrame
- createOrderButtonforOrder: JButton
- textFieldphoneNumber1: JTextField
- scrollPane3: JScrollPane
- textFieldaddress1: JTextField
- frameUpdateClients: JFrame
- textFieldname2: JTextField
- frameUpdateProducts: JFrame
- insertProduct: JButton
- insertProductButton: JButton
- textFieldmailAddress1: JTextField
- textFieldfirstName2: JTextField
- textFieldId3: JTextField
- scrollPane1: JScrollPane
- textFieldProductprice1: JTextField
- textFieldProductname1: JTextField
- frameCreateOrder: JFrame
- frameDeleteClients: JFrame
- deleteButton: JButton
- insertClient: JButton
- textFieldProductId1: JTextField
- editProduct: JButton
- textFieldProductprice2: JTextField
- textFieldfirstName1: JTextField
- textFieldname1: JTextField
- clientsButton: JButton
- deleteProduct: JButton
- createOrder: JButton
- textFieldProductIDforOrder: JTextField
- scrollPane2: JScrollPane
- textFieldmailAddress2: JTextField
- textFieldProductstock1: JTextField
- textFieldphoneNumber2: JTextField
- textFieldClientIDforOrder: JTextField
- frame: JFrame
+ displayTable1(String[], Object[][]): void
+ displayTable2(String[], Object[][]): void
+ displayTable3(String[], Object[][]): void
+ showALL(T): void
frameDeleteProduct: JFrame
textFieldProductstock2: JTextField
createOrderButtonforOrder: JButton
textFieldQuantityforOrder: JTextField
insertButton: JButton
textFieldphoneNumber1: JTextField
textFieldfirstName1: JTextField
textFieldProductId1: JTextField
frameDeleteClients: JFrame
textFieldmailAddress1: JTextField
insertClient: JButton
textFieldProductIDforOrder: JTextField
editClient: JButton
textFieldProductstock1: JTextField
textFieldProductprice2: JTextField
textFieldproductid3: JTextField
textFieldphoneNumber2: JTextField
textFieldProductname1: JTextField
scrollPane3: JScrollPane
updateButton: JButton
frame: JFrame
textFieldfirstName2: JTextField
insertProduct: JButton
textFieldId3: JTextField
createOrder: JButton
frameCreateOrder: JFrame
textFieldaddress1: JTextField
clientsButton: JButton
textFieldProductprice1: JTextField
textFieldProductname2: JTextField
scrollPane2: JScrollPane
ordersButton: JButton
textFieldclientId2: JTextField
textFieldname2: JTextField
productsButton: JButton
insertProductButton: JButton
editProduct: JButton
buttonPanel: JPanel
frameInsertClients: JFrame
textFieldaddress2: JTextField
deleteProductButton: JButton
updateProductButton: JButton
frameinsertProducts: JFrame
deleteClient: JButton
textFieldProductId2: JTextField
scrollPane1: JScrollPane
textFieldClientIDforOrder: JTextField
textFieldname1: JTextField
textFieldclientId1: JTextField
frameUpdateClients: JFrame
textFieldmailAddress2: JTextField
deleteProduct: JButton
frameUpdateProducts: JFrame
deleteButton: JButton
Orders
+ Orders(int, int, int):
+ Orders():
+ Orders(int, int, int, float, int):
- orderID: int
- productID: int
- totalPrice: float
- quantity: int
- clientID: int
+ toString(): String
quantity: int
orderID: int
productID: int
totalPrice: float
clientID: int
Client
+ Client(int, String, String, String, String, String):
+ Client():
- name: String
- mailAddress: String
- phoneNumber: String
- address: String
- clientID: int
- fristName: String
+ toString(): String
name: String
firstName: String
address: String
mailAddress: String
phoneNumber: String
clientID: int
Product
+ Product():
+ Product(int, String, int, float):
- name: String
- productID: int
- price: float
- stock: int
+ toString(): String
name: String
price: float
stock: int
OrderBLL
+ OrderBLL(Orders):
+ validateQuantity(Orders): void
+ validateProductID(Orders): void
+ validateClientID(Orders): void
ClientBLL
+ ClientBLL(Client):
+ validatePhoneNumber(Client): void
+ validateEmail(Client): void
ProductBLL
+ ProductBLL(Product):
+ validateStock(Product): void
+ validatePrice(Product): void
Controller
+ Controller(View):
+ actionPerformed(ActionEvent): void
1
1..*
Main
+ Main():
+ main(String[]): void




## 4. Implementare

***Descriere a claselor și a interfeței utilizator***

**Clasa ClientBLL**

- **Descriere:** Această clasă furnizează operațiile logice de afaceri legate de clienți, inclusiv validarea adreselor de email și a numerelor de telefon.
- **Atribute:**
  - **PHONE_PATTERN**: Șablon pentru validarea numerelor de telefon.
  - **EMAIL_PATTERN**: Șablon pentru validarea adreselor de email.
- **Metode:**
  - **validateEmail(Client client)**: Validează adresa de email a unui client.

- **validatePhoneNumber(Client client)**: Validează numărul de telefon al unui client.

**Clasa OrderBLL**

- **Descriere:** Această clasă furnizează operațiile logice de afaceri legate de comenzile efectuate, inclusiv validarea ID-urilor de produs și client, precum și a cantității.
- **Metode:**
  - **validateProductID(Orders order)**: Validează ID-ul produsului dintr-o comandă.
  - **validateClientID(Orders order)**: Validează ID-ul clientului dintr-o comandă.
  - **validateQuantity(Orders order)**: Validează cantitatea dintr-o comandă.

**Clasa ProductBLL**

- **Descriere:** Această clasă furnizează operațiile logice de afaceri legate de produse, inclusiv validarea prețului și a stocului.
- **Metode:**
  - **validatePrice(Product product)**: Validează prețul unui produs.
  - **valideateStock(Product product)**: Validează stocul unui produs.

**Clasa AbstractDAO<T>**

- **Descriere:** Această clasă este o clasă abstractă care servește ca șablon pentru toate clasele DAO specifice entităților din baza de date.
- **Atribute:**
  - **type**: Este un obiect de tip **Class<T>** care reprezintă tipul entității cu care operează clasa.
- **Metode importante:**

- **findAll()**: Returnează o listă de toate obiectele din tabel.
- **findById(int id)**: Găsește un obiect în baza de date după ID-ul specificat.
- **insert(T t)**: Inserează un nou obiect în baza de date.
- **update(T t)**: Actualizează un obiect în baza de date.
- **delete(int id)**: Șterge un obiect din baza de date pe baza ID-ului specificat.
- **createObjects(ResultSet resultSet)**: Creează un obiect de tipul specificat pe baza datelor din **ResultSet**.

**Clasa ClientDAO**

- **Descriere:** Această clasă gestionează accesul la date asociate obiectelor de tip Client în baza de date.
- **Metode importante:**
  - **insert(int clientID, int productID, int quantity)**: Metodă abstractă care trebuie implementată pentru a insera o relație între un client, un produs și o cantitate.

**Clasa OrderDAO**

- **Descriere:** Această clasă gestionează accesul la date asociate comenzilor în baza de date.
- **Atribute:**
  - **LOGGER**: Logger pentru înregistrarea mesajelor de avertizare sau erori.
  - **insertStatementString**: Declarația pentru procedura stocată de inserare a unei comenzi.
- **Metode importante:**
  - **insert(int clientID, int productID, int quantity)**: Inserează o nouă comandă în baza de date, utilizând informațiile furnizate.

- **getLastInsertID(Connection dbConnection)**: Returnează ID-ul ultimei înregistrări inserate în baza de date.

**Clasa ProductDAO**

- **Descriere:** Această clasă gestionează accesul la date asociate produselor în baza de date.
- **Metode importante:**
  - **insert(int clientID, int productID, int quantity)**: Metodă abstractă care trebuie implementată pentru a insera o relație între un client, un produs și o cantitate.

**Clasa Client:**

- **Atribute**:
  - **clientID**: ID-ul clientului (int)
  - **name**: Numele de familie al clientului (String)
  - **firstName**: Prenumele clientului (String)
  - **phoneNumber**: Numărul de telefon al clientului (String)
  - **mailAddress**: Adresa de email a clientului (String)
  - **address**: Adresa fizică a clientului (String)
- **Metode**:
  - Constructori:
    - **Client()**: Constructor fără parametri.
    - **Client(int clientID, String name, String firstName, String phoneNumber, String mailAddress, String address)**: Constructor cu parametri pentru inițializarea unui obiect **Client**.
  - Metode pentru accesarea și modificarea atributelor:

- Metode **get** și **set** pentru fiecare atribut: **getClientID()**, **getName()**, **getFirstName()**, **getPhoneNumber()**, **getMailAddress()**, **getAddress()**, **setClientID()**, **setName()**, **setFirstName()**, **setPhoneNumber()**, **setMailAddress()**, **setAddress()**.

- **toString()**: Metodă suprascrisă pentru a returna o reprezentare sub formă de șir de caractere a informațiilor despre client.

**Clasa Orders:**

- **Atribute**:
  - **orderID**: ID-ul comenzii (int)
  - **clientID**: ID-ul clientului care a plasat comanda (int)
  - **productID**: ID-ul produsului din comandă (int)
  - **totalPrice**: Prețul total al comenzii (float)
  - **quantity**: Cantitatea de produse comandate (int)

- **Metode**:
  - Constructori:
    - **Orders()**: Constructor fără parametri.
    - **Orders(int orderID, int productID, int clientID, float totalPrice, int quantity)**: Constructor cu parametri pentru inițializarea unei comenzi.
    - **Orders(int productID, int clientID, int quantity)**: Constructor alternativ cu parametri pentru inițializarea unei comenzi fără ID-ul comenzii.
  - Metode **get** și **set** pentru fiecare atribut: **getOrderID()**, **getClientID()**, **getProductID()**, **getTotalPrice()**, **getQuantity()**, **setOrderID()**, **setClientID()**, **setProductID()**, **setTotalPrice()**, **setQuantity()**.

- **toString()**: Metodă suprascrisă pentru a returna o reprezentare sub formă de șir de caractere a informațiilor despre comandă.

**Clasa Product:**

- **Atribute**:
  - **productID**: ID-ul produsului (int)
  - **name**: Numele produsului (String)
  - **stock**: Stocul disponibil pentru produs (int)
  - **price**: Prețul produsului (float)
- **Metode**:
  - Constructori:
    - **Product()**: Constructor fără parametri.
    - **Product(int productID, String name, int stock, float price)**: Constructor cu parametri pentru inițializarea unui produs.
  - Metode **get** și **set** pentru fiecare atribut: **getProductID()**, **getName()**, **getStock()**, **getPrice()**, **setProductID()**, **setName()**, **setStock()**, **setPrice()**.
  - **toString()**: Metodă suprascrisă pentru a returna o reprezentare sub formă de șir de caractere a informațiilor despre produs.

**Clasa Controller:**

- **Atribute**:
  - **view**: O referință către obiectul de tip **View** cu care interacționează controlerul.

- **Constructor**:
  - **Controller(View view)**: Constructorul primește o referință către obiectul **View** și inițializează interacțiunea cu butoanele din interfața grafică.
- **Metode**:
  - **actionPerformed(ActionEvent e)**: Metodă care gestionează evenimentele de acțiune generate de interacțiunea utilizatorului cu interfața grafică. În funcție de butonul apăsat, această metodă decide ce acțiune trebuie să întreprindă, cum ar fi afișarea datelor despre clienți, produse, crearea sau editarea comenzilor etc.

**Clasa ConnectionFactory:**

- **Atribute**:
  - **LOGGER**: Obiect pentru înregistrarea mesajelor de tipul Logger.
  - **DRIVER**: Șir de caractere care specifică driver-ul JDBC pentru conexiunea la baza de date MySQL.
  - **DBURL**: Șir de caractere care specifică adresa URL a bazei de date MySQL.
  - **USER**: Numele utilizatorului pentru conectarea la baza de date MySQL.
  - **PASS**: Parola utilizatorului pentru conectarea la baza de date MySQL.
  - **singleInstance**: Instanță unică a clasei **ConnectionFactory**.
- **Metode**:
  - **createConnection()**: Metodă privată care creează și returnează o conexiune la baza de date.

- **getConnection()**: Metodă statică care returnează o conexiune la baza de date folosind instanța unică a clasei.
- **close(Connection connection)**, **close(Statement statement)**, **close(ResultSet resultSet)**: Metode statice pentru închiderea conexiunii, a obiectului **Statement** și a obiectului **ResultSet**.

**Clasa Main:**

- **Metoda main(String[] args)**:
  - Punctul de intrare în aplicație.
  - Creează o instanță a clasei **View** pentru a afișa interfața grafică.
  - Inițializează un obiect **Controller** pentru a gestiona interacțiunea utilizatorului cu interfața grafică.

- **Clasa View:** Această clasă este o fereastră principală a aplicației și extinde **JFrame**. Ea conține câmpuri pentru diferite componente ale interfeței grafice, precum butoane pentru gestionarea clienților, produselor și comenzilor, precum și pentru inserarea, editarea și ștergerea acestora. De asemenea, conține și tabele pentru afișarea datelor despre clienți, produse și comenzi.
  - **Câmpuri:**
    - **clientsButton**, **productsButton**, **ordersButton**: Butoane pentru comutarea între vizualizarea clienților, produselor și comenzilor.
    - **insertClient**, **editClient**, **deleteClient**: Butoane pentru inserarea, editarea și ștergerea clienților.
    - **insertProduct**, **editProduct**, **deleteProduct**: Butoane pentru inserarea, editarea și ștergerea produselor.
    - **createOrder**: Buton pentru crearea unei comenzi noi.

- **table1**, **table2**, **table3**: Tabele pentru afișarea datelor despre clienți, produse și comenzi.
- Alte câmpuri pentru gestionarea ferestrelor și a altor componente grafice.

- **Metode:**
  - **showALL(T t)**: Metodă pentru afișarea tuturor datelor dintr-o anumită entitate în tabelul corespunzător.
  - **displayTable1(String[] columnNames, Object[][] data)**: Metodă pentru afișarea datelor în tabelul corespunzător entității Client.
  - **displayTable2(String[] columnNames, Object[][] data)**: Metodă pentru afișarea datelor în tabelul corespunzător entității Product.
  - **displayTable3(String[] columnNames, Object[][] data)**: Metodă pentru afișarea datelor în tabelul corespunzător entității Orders.
  - Metode de acces pentru obținerea diferitelor componente ale interfeței grafice, cum ar fi butoanele, tabelele și ferestrele.
- **Constructor:**
  - Inițializează toate componentele interfeței grafice și stabilește aspectul și comportamentul inițial al ferestrei principale.

## 5. Rezultate

Nu a fost cazul.

## 6. Concluzii

Scenarii de Testare Simplificate:

1. ***Adăugarea unui Client Nou:***

   - Utilizatorul introduce informațiile despre client și apasă "Adăugare".
   - Se verifică adăugarea cu succes în baza de date.
   - Lista clienților este actualizată în interfață.

2. ***Editarea unui Client Existenta:***

   - Utilizatorul selectează un client și actualizează informațiile.
   - Apăsând "Editare", detaliile sunt actualizate în baza de date.
   - Lista clienților în interfață se actualizează.

3. ***Ștergerea unui Client:***

   - Utilizatorul selectează un client și apasă "Ștergere".
   - Se verifică ștergerea cu succes din baza de date.
   - Lista clienților în interfață este actualizată fără clientul șters.

4. ***Adăugarea unui Produs Nou:***

   - Utilizatorul completează informațiile despre produs și apasă "Adăugare".
   - Se verifică adăugarea cu succes în baza de date.
   - Lista produselor este actualizată în interfață.

5. ***Editarea unui Produs Existenta:***

   - Utilizatorul selectează un produs și actualizează informațiile.
   - Apăsând "Editare", detaliile sunt actualizate în baza de date.
   - Lista produselor în interfață se actualizează.

6. ***Ștergerea unui Produs:***

- Utilizatorul selectează un produs și apasă "Ștergere".
- Se verifică ștergerea cu succes din baza de date.
- Lista produselor în interfață este actualizată fără produsul șters.

7. ***Plasarea unei Comenzi:***

- Utilizatorul selectează un client, un produs și introduce cantitatea.
- Apăsând "Plasează comanda", se verifică disponibilitatea și se adaugă comanda.
- Utilizatorul primește un mesaj de succes și o factură în format PDF este generată automat.

## 7. Bibliografie

*https://dsrl.eu/courses/pt/materials/PT2024_A3_S1.pdf*
*https://dsrl.eu/courses/pt/materials/PT2024_A3_S2.pdf*
*https://dsrl.eu/courses/pt/materials/PT2024_A3.pdf*
*https://gitlab.com/utcn_dsrl/pt-layered-architecture*
*https://gitlab.com/utcn_dsrl/pt-reflection-example*