

# **DOCUMENTATIE**

## **TEMA 3**

NUME STUDENT: Igna Alexandra Andreea  
GRUPA: 30225

## CUPRINS

|    |  |    |
|----|--|----|
| 1. | Obiectivul temei.....  | 3  |
| 2. | Analiza problemei, modelare, scenarii, cazuri de utilizare ..... | 3  |
| 3. | Proiectare.....  | 5  |
| 4. | Implementare.....  | 6  |
| 5. | Rezultate.....   | 10 |
| 6. | Concluzii .....  | 14 |
| 7. | Bibliografie.....  | 14 |

## **Obiectivul temei**

Obiectiv principal: Proiectarea si implementarea unei aplicatii pentru gestionarea comenzilor de clienti pentru un depozit

Obiective secundare:

- Analiza problemei si identificarea cerintelor
- Proiectarea aplicatiei de gestionare a comenzilor
- Implementarea aplicatiei de gestionare a comenzilor
- Testarea aplicatiei de gestionare a comenzilor

## **Analiza problemei, modelare, scenarii, cazuri de utilizare**

Cerinte functionale:

- Adaugarea unui client nou in sistem
- Modificarea informatiilor unui client existent
- Stergerea unui client din sistem
- Vizualizarea listei de clienti intr-un tabel
- Adaugarea unui produs nou in sistem
- Modificarea informatiilor unui produs existent
- Stergerea unui produs din sistem
- Vizualizarea listei de produse intr-un tabel
- Crearea unei comenzi de produse

Cerinte non-functionale:

- Utilizarea unei arhitecturi stratificate
- Utilizarea JavaDoc pentru documentarea claselor
- Stocarea datelor intr-o baza de date relationala, cu cel putin trei tabele: Client, Product si Order
- Crearea unei interfete grafice user-friendly

Descrierea use-case-urilor:

Adaugarea unui client nou

- Utilizatorul introduce informatiile necesare in fereastra de adaugare a clientului

- Aplicatia valideaza datele introduse si le salveaza in baza de date

#### Modificarea informatiilor unui client existent

- Utilizatorul selecteaza un client din tabel
- Utilizatorul modifica informatiile clientului in fereastra de editare a clientului
- Aplicatia valideaza datele modificate si le salveaza in baza de date

#### Stergerea unui client din sistem

- Utilizatorul selecteaza un client din tabel
- Utilizatorul apasa butonul "Sterge client"
- Aplicatia sterge clientul selectat din baza de date

#### Crearea unei comenzi de produse

- Utilizatorul selecteaza un produs si un client in fereastra de creare a comenzii
- Utilizatorul introduce cantitatea dorita pentru produs
- Aplicatia verifica disponibilitatea stocului pentru produsul selectat
- Daca stocul este suficient, aplicatia creeaza comanda si actualizeaza stocul in baza de date
- Daca stocul nu este suficient, aplicatia afiseaza un mesaj de stoc insuficient

#### Adaugarea unui produs nou

- Utilizatorul introduce informatiile necesare in fereastra de adaugare a produsului
- Aplicatia valideaza datele introduse si le salveaza in baza de date

#### Modificarea informatiilor unui produs existent

- Utilizatorul selecteaza un produs din tabel
- Utilizatorul modifica informatiile produsului in fereastra de editare a produsului
- Aplicatia valideaza datele modificate si le salveaza in baza de date

#### Stergerea unui produs din sistem

- Utilizatorul selecteaza un produs din tabel
- Utilizatorul apasa butonul "Sterge produs"
- Aplicatia sterge produsul selectat din baza de date

#### Vizualizarea listei de clienti intr-un tabel

- Utilizatorul deschide fereastra de operatiuni client
- Aplicatia afiseaza lista de clienti intr-un tabel

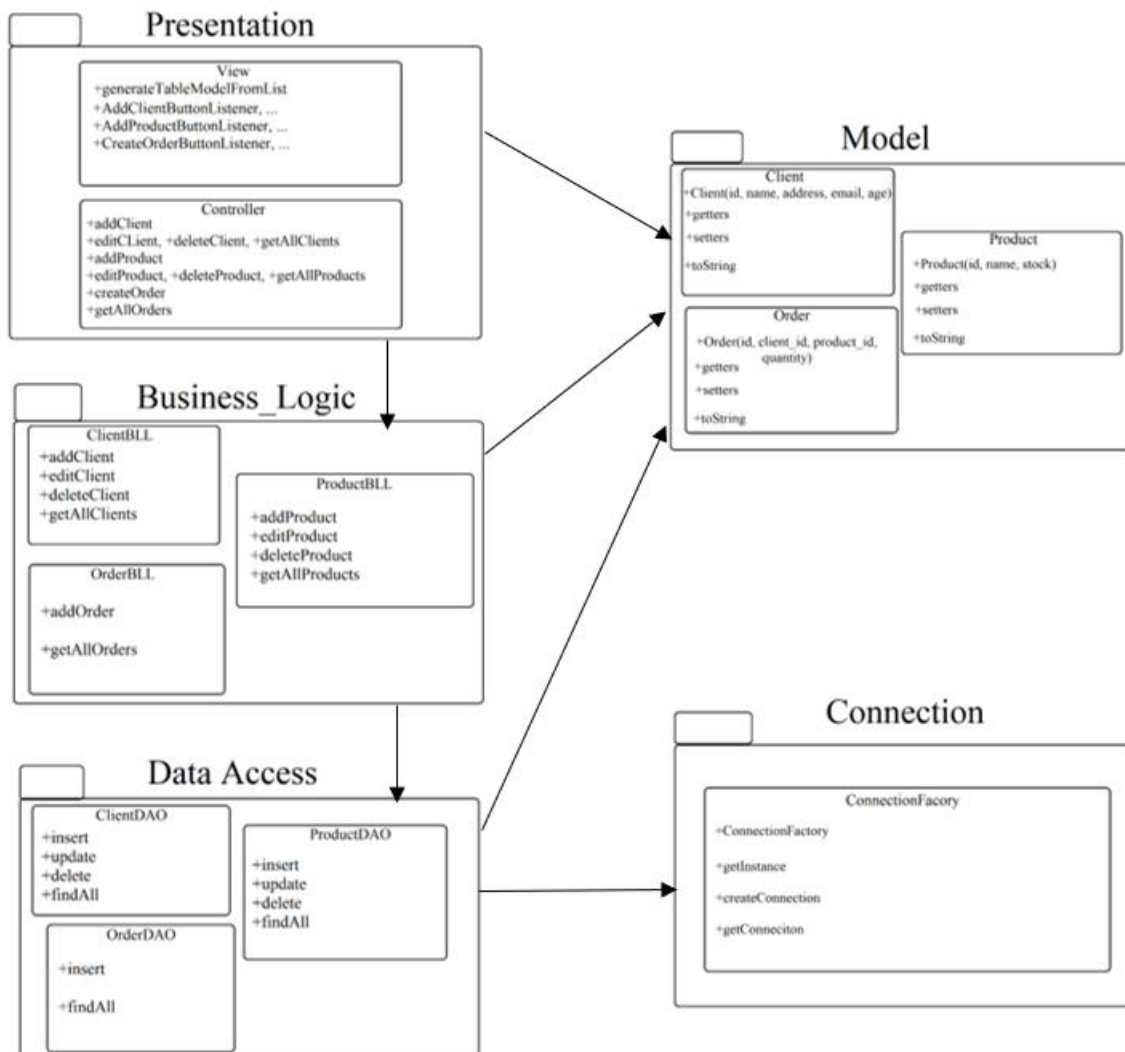
#### Vizualizarea listei de produse intr-un tabel

- Utilizatorul deschide fereastra de operatiuni produs

- Aplicatia afiseaza lista de produse intr-un tabel

## Proiectare

### Diagrama UML



### Structurile de date folosite

Pentru a stoca si manipula datele in cadrul aplicatiei, vom folosi urmatoarele structuri de date:

Liste de obiecte pentru reprezentarea clientilor, produselor si comenzilor din baza de date.

Matrice bidimensionale pentru reprezentarea datelor in tabelele JTable.

## Interfetele definite

In aplicatie, interactiunea intre pachete si clase va fi realizata prin intermediul metodelor publice definite in fiecare clasa. De exemplu:

Clasele BLL vor expune metode pentru adaugarea, actualizarea, stergerea si interogarea datelor.

Clasele DAO vor expune metode pentru efectuarea operatiilor specifice fiecarei entitati.

## Algoritmi utilizati

In cadrul aplicatiei, vom utiliza algoritmi pentru a valida si procesa datele, precum:

Algoritmi de validare a datelor introduse in formularele de adaugare si modificare a clientilor si produselor.

Algoritmi pentru verificarea disponibilitatii stocului inainte de a crea o comanda.

Algoritmi pentru actualizarea stocului dupa finalizarea unei comenzi.

## Implementare

### Clasa **Client**:

Aceasta clasa reprezinta un client cu proprietatile sale asociate. Campurile includ client\_id, name, address, email si age. Exista un constructor care seteaza toate proprietatile unui client si un constructor fara parametri. De asemenea, exista un constructor care seteaza proprietatile unui client, fara ID. Clasa include metode pentru obtinerea si modificarea acestor campuri, precum getId, setId, getName, setName, getAddress, setAddress, getEmail, setEmail, getAge si setAge. La final, exista o metoda toString care returneaza o reprezentare textuala a obiectului Client.

### Clasa **Order**:

Aceasta clasa reprezinta o comanda in sistem. Campurile includ id, client\_id, product\_id si quantity. Exista un constructor implicit si un constructor cu parametri care seteaza ID-ul clientului care face comanda, ID-ul produsului comandat si cantitatea de produs comandata. Clasa include metode pentru obtinerea si modificarea acestor campuri, precum getId, setId, getClient\_id, setClient\_id, getProduct\_id, setProduct\_id, getQuantity si setQuantity. La final, exista o metoda toString care returneaza o reprezentare textuala a obiectului Order.

### Clasa **Product**:

Aceasta clasa reprezinta un produs in sistem. Campurile includ product\_id, name, price si current\_stock. Exista un constructor cu parametri care seteaza toate campurile, un constructor implicit si un constructor fara ID. Clasa include metode pentru obtinerea si modificarea acestor campuri, precum getId, setId, getName, setName, getPrice, setPrice, getCurrent\_stock si setCurrent\_stock. La final, exista o metoda toString care returneaza o reprezentare textuala a obiectului Product.

### Clasa **ClientBLL**:

Aceasta clasa gestioneaza operatiunile pentru clienti si include metode precum addClient, editClient, deleteClient si getAllClients, care adauga, modifica, sterge si returneaza o lista cu toti clientii din baza de date, respectiv. Clasa are un constructor implicit care initializeaza un obiect de tip ClientDAO.

### Clasa **OrderBLL**:

Aceasta clasa gestioneaza operatiunile pentru comenzi si include metode precum createOrder si getAllOrders, care adauga o comanda noua si returneaza o lista cu toate comenzile din baza de date, respectiv. Clasa are un constructor implicit care initializeaza un obiect de tip OrderDAO.

### Clasa **ProductBLL**:

Aceasta clasa gestioneaza operatiunile pentru produse si include metode precum addProduct, editProduct, deleteProduct, getAllProducts si getProductById, care adauga, modifica, sterge, returneaza o lista cu toate produsele si cauta un produs dupa ID in baza de date, respectiv. Clasa are un constructor implicit care initializeaza un obiect de tip ProductDAO.

### Clasa **ClientDAO**:

Aceasta clasa se ocupa de manipularea datelor clientilor din baza de date si include metode precum insert, update, delete si findAll. Metodele permit inserarea unui client nou in baza de date, actualizarea datelor unui client existent, stergerea unui client dupa ID si returnarea unei liste cu toti clientii. Clasa utilizeaza constant un set de string-uri SQL pentru a efectua interogările necesare.

### **Clasa `OrderDAO`:**

Aceasta clasa se ocupa de manipularea datelor comenzilor din baza de date si include metode precum `insert` si `findAll`. Metodele permit inserarea unei comenzi noi in baza de date si returnarea unei liste cu toate comenzile. La fel ca si clasa `ClientDAO`, clasa `OrderDAO` utilizeaza constant un set de string-uri SQL pentru a efectua interogările necesare.

### **Clasa `ProductDAO`:**

Aceasta clasa se ocupa de manipularea datelor produselor din baza de date si include metode precum `findProductById`, `insert`, `update`, `delete` si `findAll`. Metodele permit gasirea unui produs dupa ID, inserarea unui produs nou in baza de date, actualizarea datelor unui produs existent, stergerea unui produs dupa ID si returnarea unei liste cu toate produsele. Clasa utilizeaza constant un set de string-uri SQL pentru a efectua interogările necesare.

### **Clasa `ConnectionFactory`:**

Aceasta clasa este responsabila pentru crearea si inchiderea conexiunilor cu baza de date. Este implementata folosind pattern-ul Singleton, ceea ce inseamna ca exista doar o singura instanta a acestei clase in aplicatie. `ConnectionFactory` utilizeaza JDBC (Java Database Connectivity) pentru a stabili conexiuni cu baza de date si expune metoda statica `getConnection()` pentru a obtine o conexiune. Clasa stocheaza informatiile necesare pentru a stabili conexiunea, precum driver-ul JDBC, URL-ul bazei de date, numele de utilizator si parola.

### **Clasa `Controller`:**

Aceasta clasa reprezinta controlerul aplicatiei si gestioneaza operatiunile de baza pentru obiectele `Client`, `Product` si `Order`. Controlerul utilizeaza obiecte `ClientBLL`, `ProductBLL` si `OrderBLL` pentru a efectua operatiuni asupra bazelor de date corespunzatoare.

Metodele disponibile in clasa `Controller` includ adaugarea, modificarea si stergerea clientilor, produselor si comenzilor, precum si obtinerea listelor cu toti clientii, produsele si comenzile din baza de date.

Folosind aceste metode, aplicatia poate efectua operatiuni CRUD (Create, Read, Update, Delete) pentru clienti, produse si comenzi, iar aceste operatiuni sunt gestionate de clasele de logica de afaceri (`ClientBLL`, `ProductBLL` si `OrderBLL`) si clasele de acces la date (`ClientDAO`, `ProductDAO` si `OrderDAO`).

### **Clasa `View`:**

Aceasta clasa extinde `JFrame` si reprezinta interfata grafica a utilizatorului (GUI) pentru gestionarea comenzilor, clientilor si produselor. Ea contine componente precum butoane si panouri pentru a permite utilizatorului sa interactioneze cu aplicatia.



Constructorul clasei View primește un obiect Controller și initializează componentele, cum ar fi combourile pentru clienți și produse, câmpul pentru cantitate și setează proprietățile ferestrei.

Metoda `generateTableModelFromList()` este utilizată pentru a genera un tabel model dintr-o listă de obiecte și este folosită pentru a afișa informațiile despre clienți, produse și comenzi într-un tabel.

Metodele `initQuantityTextField()`, `initClientComboBox()` și `initProductComboBox()` sunt utilizate pentru a inițializa câmpul de text pentru cantitate și combourile pentru clienți și produse.

Metoda `initComponents()` inițializează toate componentele, precum butoanele pentru adăugarea, actualizarea și ștergerea clienților, produselor și comenzilor, și setează ascultătorii de acțiuni pentru fiecare dintre ele.

Metoda `setUpLayout()` este utilizată pentru a seta layout-ul și adăuga componentele în fereastra aplicației.

Clasele interne **AddClientButtonListener**, **UpdateClientButtonListener** și celelalte clase similare pentru produse și comenzi sunt ascultători de acțiuni pentru butoane. Acestea definesc comportamentul care va fi executat atunci când butoanele respective sunt apasate. De exemplu, când se apasă butonul "Add Client", se deschide un panou de introducere a informațiilor despre client, iar dacă utilizatorul apasă OK, un nou client este adăugat în baza de date prin metoda `controller.addClient(client)`.

**DeleteClientButtonListener:** Această clasă ascultă acțiunile de pe butonul "Delete Client". Când se face clic pe buton, se afișează o casetă de dialog pentru introducerea ID-ului clientului care va fi șters. După aceea, clientul este șters folosind metoda `controller.deleteClient()`.

**ViewAllClientsButtonListener:** Această clasă gestionează acțiunile pentru butonul "View All Clients". Când se apasă acest buton, aceasta preia lista tuturor clienților prin `controller.getAllClients()`, apoi creează un tabel cu acești clienți și îl afișează într-o casetă de dialog.

**AddProductButtonListener:** Această clasă gestionează acțiunile pentru butonul "Add Product". Când se apasă acest buton, se deschide o casetă de dialog care solicită informații despre produsul ce urmează a fi adăugat (nume, preț, stoc). După aceea, produsul este adăugat folosind metoda `controller.addProduct()`.

**UpdateProductButtonListener:** Această clasă gestionează acțiunile pentru butonul "Update Product". Când se apasă acest buton, se deschide o casetă de dialog care solicită ID-ul produsului ce urmează a fi actualizat și informațiile actualizate. Apoi, produsul este actualizat folosind metoda `controller.editProduct()`.

**DeleteProductButtonListener:** Această clasă gestionează acțiunile pentru butonul "Delete Product". Când se face clic pe acest buton, se deschide o casetă de dialog pentru

introducerea ID-ului produsului ce urmeaza a fi sters. Dupa aceea, produsul este sters folosind metoda `controller.deleteProduct()`.

**ViewAllProductsButtonListener:** Aceasta clasa gestioneaza actiunile pentru butonul "View All Products". Cand se apasa acest buton, aceasta preia lista tuturor produselor prin `controller.getAllProducts()`, apoi creeaza un tabel cu aceste produse si il afiseaza intr-o caseta de dialog.

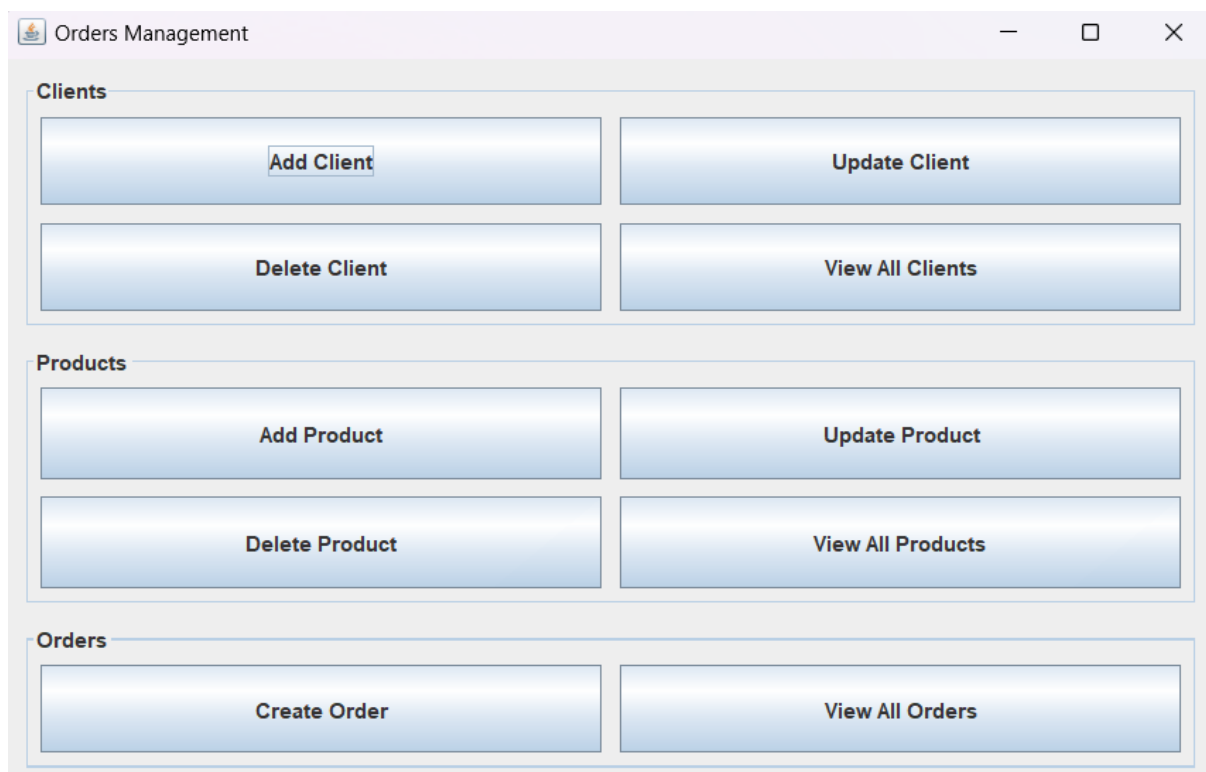
**CreateOrderButtonListener:** Aceasta clasa gestioneaza actiunile pentru butonul "Create Order". Cand se apasa acest buton, se deschide o caseta de dialog pentru a alege clientul, produsul si cantitatea dorita. Daca stocul este suficient, se creeaza o noua comanda prin metoda `controller.createOrder()` si se actualizeaza stocul produsului.

**ViewAllOrdersButtonListener:** Aceasta clasa gestioneaza actiunile pentru butonul "View All Orders". Cand se apasa acest buton, aceasta preia lista tuturor comenzilor prin `controller.getAllOrders()`, apoi creeaza un tabel cu aceste comenzi si il afiseaza intr-o caseta de dialog.

Metoda **main** din clasa View creeaza o instanta a clasei Controller si o instanta a clasei View. Apoi, seteaza fereastra principala a aplicatiei ca vizibila, astfel incat utilizatorul sa poata interactiona cu ea.

## Rezultate: Scenariile pentru testare

La deschiderea aplicatiei, vedem interfata cu toate butoanele, si diverse scenarii:



### Scenariul 1: Adaugarea unui client

Deschideti aplicatia si faceti clic pe butonul "Add Client".

In caseta de dialog care apare, introduceti datele necesare (nume, adresa, email, varsta).

Faceti clic pe butonul OK si verificati daca clientul a fost adaugat cu succes.

A dialog box titled "Add Client" with a close button (X) in the top right corner. On the left, there is a green square icon with a white question mark. To its right, there are four labels: "Name:", "Address:", "Email:", and "Age:". Each label is followed by a text input field. At the bottom of the dialog, there are two buttons: "OK" and "Cancel".

|          |  |
|----------|--|
| Name:    |  |
| Address: |  |
| Email:   |  |
| Age:     |  |

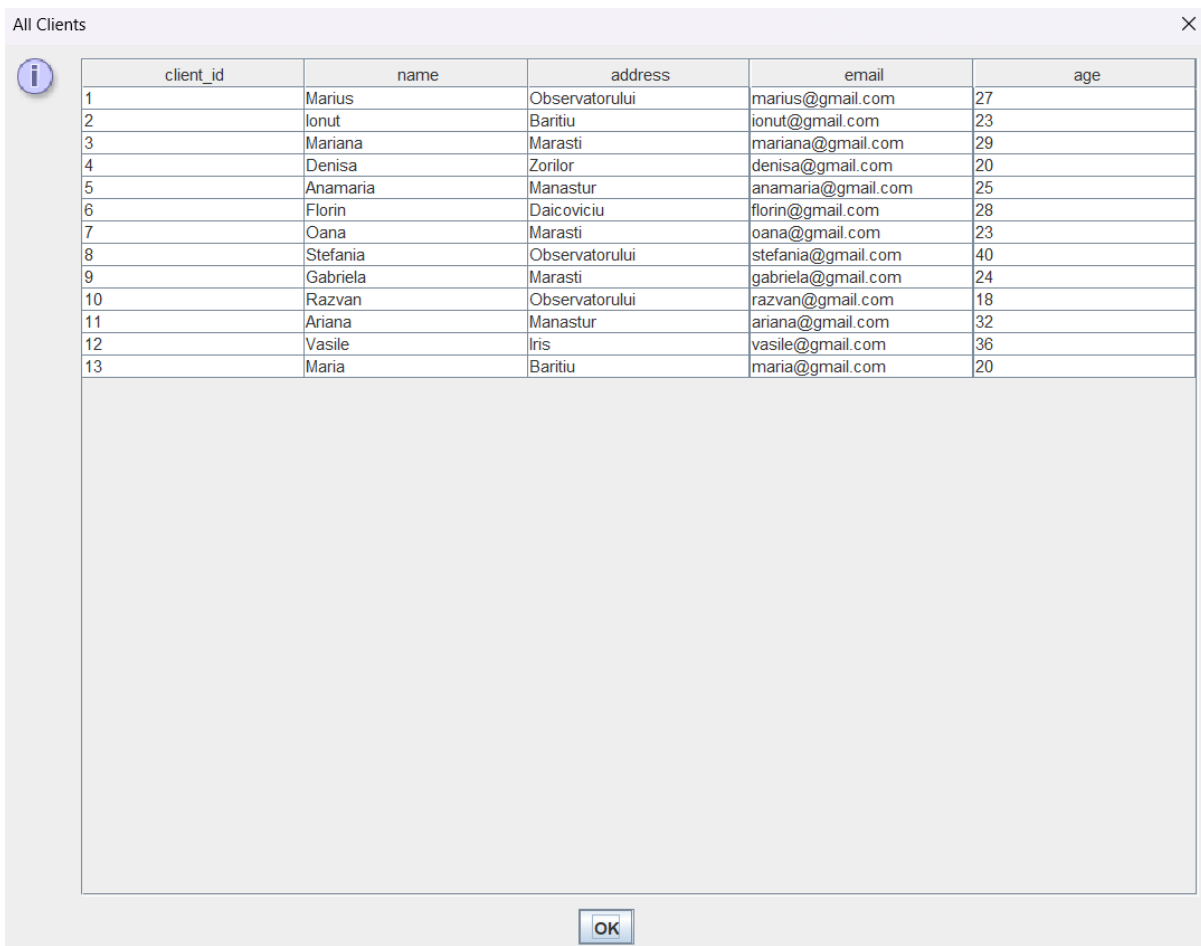
OK Cancel

### Scenariul 2: Vizualizarea tuturor clientilor

Asigurati-va ca aplicatia contine mai multi clienti.

Faceti clic pe butonul "View All Clients".

Verificati daca se afiseaza o lista cu toti clientii existenti in sistem.

A dialog box titled "All Clients" with a close button (X) in the top right corner. On the left, there is a blue circular icon with a white 'i'. The main area contains a table with 5 columns: "client\_id", "name", "address", "email", and "age". The table has 13 rows of data. Below the table, there is a large empty rectangular area. At the bottom center, there is an "OK" button.

| client_id | name     | address        | email              | age |
|-----------|----------|----------------|--------------------|-----|
| 1         | Marius   | Observatorului | marius@gmail.com   | 27  |
| 2         | Ionut    | Baritiu        | ionut@gmail.com    | 23  |
| 3         | Mariana  | Marasti        | mariana@gmail.com  | 29  |
| 4         | Denisa   | Zonilor        | denisa@gmail.com   | 20  |
| 5         | Anamaria | Manastur       | anamaria@gmail.com | 25  |
| 6         | Florin   | Daicoviciu     | florin@gmail.com   | 28  |
| 7         | Oana     | Marasti        | oana@gmail.com     | 23  |
| 8         | Stefania | Observatorului | stefania@gmail.com | 40  |
| 9         | Gabriela | Marasti        | gabriela@gmail.com | 24  |
| 10        | Razvan   | Observatorului | razvan@gmail.com   | 18  |
| 11        | Ariana   | Manastur       | ariana@gmail.com   | 32  |
| 12        | Vasile   | Iris           | vasile@gmail.com   | 36  |
| 13        | Maria    | Baritiu        | maria@gmail.com    | 20  |

OK

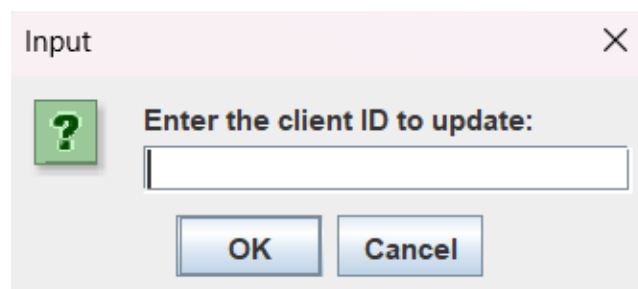
### *Scenariul 3: Actualizarea unui client*

Deschideti aplicatia si faceti clic pe butonul "Update Client".

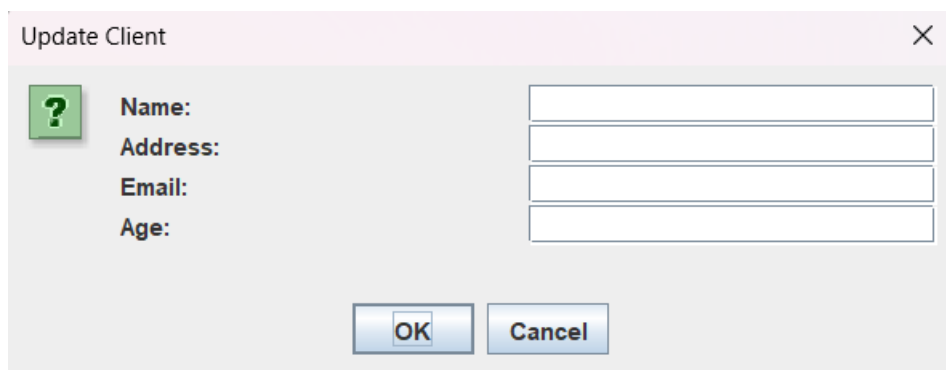
In caseta de dialog care apare, introduceti id-ul clientului.

In caseta de dialog care apare, introduceti datele necesare (nume, adresa, email, varsta).

Faceti clic pe butonul OK si verificati daca clientul a fost actualizat cu succes.



A dialog box titled "Input" with a close button (X) in the top right corner. On the left, there is a green square icon with a white question mark. To the right of the icon, the text "Enter the client ID to update:" is displayed. Below the text is a single-line text input field. At the bottom of the dialog, there are two buttons: "OK" and "Cancel".



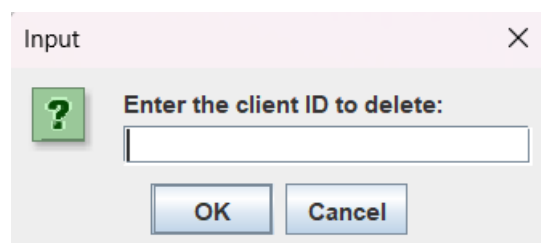
A dialog box titled "Update Client" with a close button (X) in the top right corner. On the left, there is a green square icon with a white question mark. To the right of the icon, the labels "Name:", "Address:", "Email:", and "Age:" are listed vertically. To the right of these labels are four single-line text input fields, one for each label. At the bottom of the dialog, there are two buttons: "OK" and "Cancel".

### *Scenariul 4: Stergerea unui client*

Deschideti aplicatia si faceti clic pe butonul "Delete Client".

In caseta de dialog care apare, introduceti id-ul clientului.

Faceti clic pe butonul OK si verificati daca clientul a fost sters cu succes.



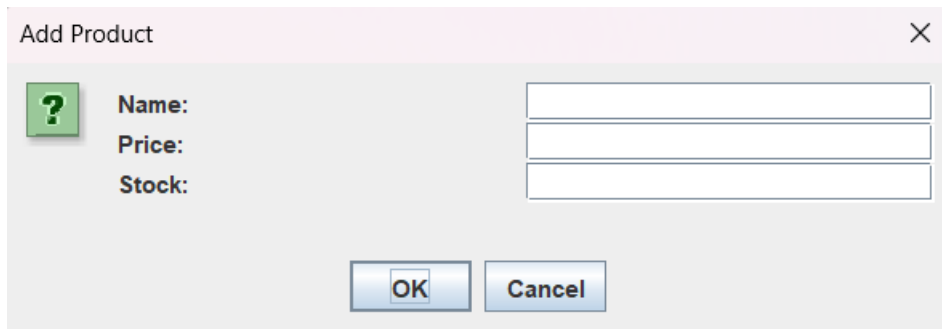
A dialog box titled "Input" with a close button (X) in the top right corner. On the left, there is a green square icon with a white question mark. To the right of the icon, the text "Enter the client ID to delete:" is displayed. Below the text is a single-line text input field. At the bottom of the dialog, there are two buttons: "OK" and "Cancel".

### *Scenariul 5: Adaugarea unui produs*

Faceti clic pe butonul "Add Product".

Introduceti datele necesare (nume, pret, stoc) in caseta de dialog.

Faceti clic pe butonul OK si verificati daca produsul a fost adaugat cu succes.



The image shows a dialog box titled "Add Product" with a close button (X) in the top right corner. On the left side, there is a green square icon with a white question mark. To the right of the icon, the labels "Name:", "Price:", and "Stock:" are listed vertically. To the right of these labels are three empty text input fields. At the bottom of the dialog box, there are two buttons: "OK" and "Cancel".

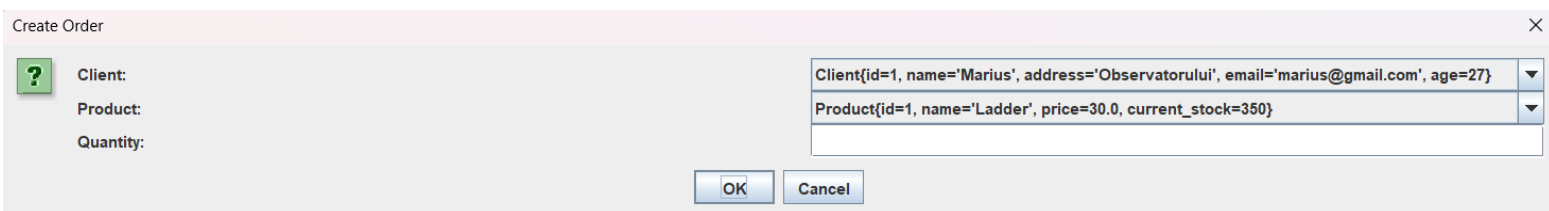
### *Scenariul 6: Crearea unei comenzi*

Asigurati-va ca aplicatia contine cel putin un client si un produs.

Faceti clic pe butonul "Create Order".

Alegeti un client, un produs si introduceti o cantitate in caseta de dialog.

Faceti clic pe butonul OK si verificati daca comanda a fost creata cu succes.



The image shows a dialog box titled "Create Order" with a close button (X) in the top right corner. On the left side, there is a green square icon with a white question mark. To the right of the icon, the labels "Client:", "Product:", and "Quantity:" are listed vertically. To the right of these labels are two dropdown menus and one text input field. The first dropdown menu is labeled "Client" and contains the text "Client{id=1, name='Marius', address='Observatorului', email='marius@gmail.com', age=27}". The second dropdown menu is labeled "Product" and contains the text "Product{id=1, name='Ladder', price=30.0, current\_stock=350}". The third text input field is labeled "Quantity:". At the bottom of the dialog box, there are two buttons: "OK" and "Cancel".

## Concluzii

Dezvoltand aceasta aplicatie de comenzi, am invatat importanta organizarii codului si utilizarea programarii orientate pe obiecte. Am folosit Model-View-Controller pentru a separa functionalitatea, facand aplicatia mai usor de intretinut. Pe viitor, aplicatia poate fi imbunatatita prin adaugarea de noi functii si optimizarea interfetei.

## Bibliografie

<https://www.baeldung.com/java-jdbc>

<http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/3>.

<https://dzone.com/articles/layers-standard-enterprise>

<https://www.baeldung.com/javadoc>

<https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html>

Prezentarile suport din <https://dsrl.eu/courses/pt/>