

Order Management

Documentatie – Tema 3

Czako Zoltan

Grupa 30225

Contents

1.Obiectivul temei	3
2. Analiza problemei , modelare , scenarii , cazuri de utilizare.....	3
3. Proiectare	4
3.1 Usecase UML	5
3.2 Sequence diagram 1	6
3.3 Sequence diagram 2	7
3.4 Activiry diagram	8
3.5 Diagrama UML pentru baza de date.....	9
3.6 Diagrama UML pentru clasele Java.....	10
3.7 Interfata grafica	13
4. Testare	15
5. Rezultate.....	16
6. Concluzii	16
7. Bibiliografie	17

1. Obiectivul temei

Enunțul temei:

„Consider an application OrderManagement for processing customer orders. The application uses (minimally) the following classes: Order, OPDept (Order Processing Department), Customer, Product, and Warehouse. The classes OPDept and Warehouse use a BinarySearchTree for storing orders.

- a) Analyze the application domain, determine the structure and behavior of its classes, identify use cases and generate use case diagrams, an extended UML class diagram and two sequence diagrams.*
- b) Implement the application classes. Use javadoc for documenting classes.*
- c) Implement a system of utility programs for reporting such as: under-stock, over-stock, totals, filters, etc.*
- d) Write the appropriate test drivers”*

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

Avem de făcut un program folosite foarte des în zilele noastre, de firmele care vând produsele pe internet, dar dar și de firmele obișnuite, care au un system pentru prelucrarea comenzilor, verificarea stockului din depozit.

Programul va afisa produsele din depozit (Warehouse). Langă asta, putem adauga produse noi , putem efectuacomenzi , afișa comenzile procesate, neprocesate. Despre un produs cunoaștem numele produsulei, tipul produsului , prețul , cantitatea din depozit. Despre o comanda cunoaștem, numele clientului care a efectuat comanda, ce a comandat , cate bucați , si prețul total a comenzii.

Enunțul temei impune niște, și ne obligă folosirea acestora (Order, OPDept, Costumer, Product și Warehouse). Lângă clasele impuse de enunț putem folosi și alte clase, daca consideram că sunt necesare. Lângă aceste clase vom folosi o clasa View, care are scopul de a desena interfața grafica, un controller care face legătura intre interfața grafică și programul care prelucrează datele date de la utilizator. De asemenea vom avea și clase de test conform JUnit.

3. Proiectare (diagrama UML, structuri de date, proiectare clase , interfete , relații, packages, algoritmi, interfața utilizator)

Pentru rezolvarea acestei probleme am folosit atat limbajul de programare Java, cat si un limbaj pentru crearea si manipularea bazelor de date, adica limbajul MySQL.

Pentru stocarea datelor am folosit baza de date numit Restorante, deoarece aceasta aplicatie este o aplicatie de gestiune a produselor in cadrul unui restaurant (aceasta aplicatie poate fi folosita ca si o aplicatie pentru catering).

Baza de date contine sase tabele, si anume:

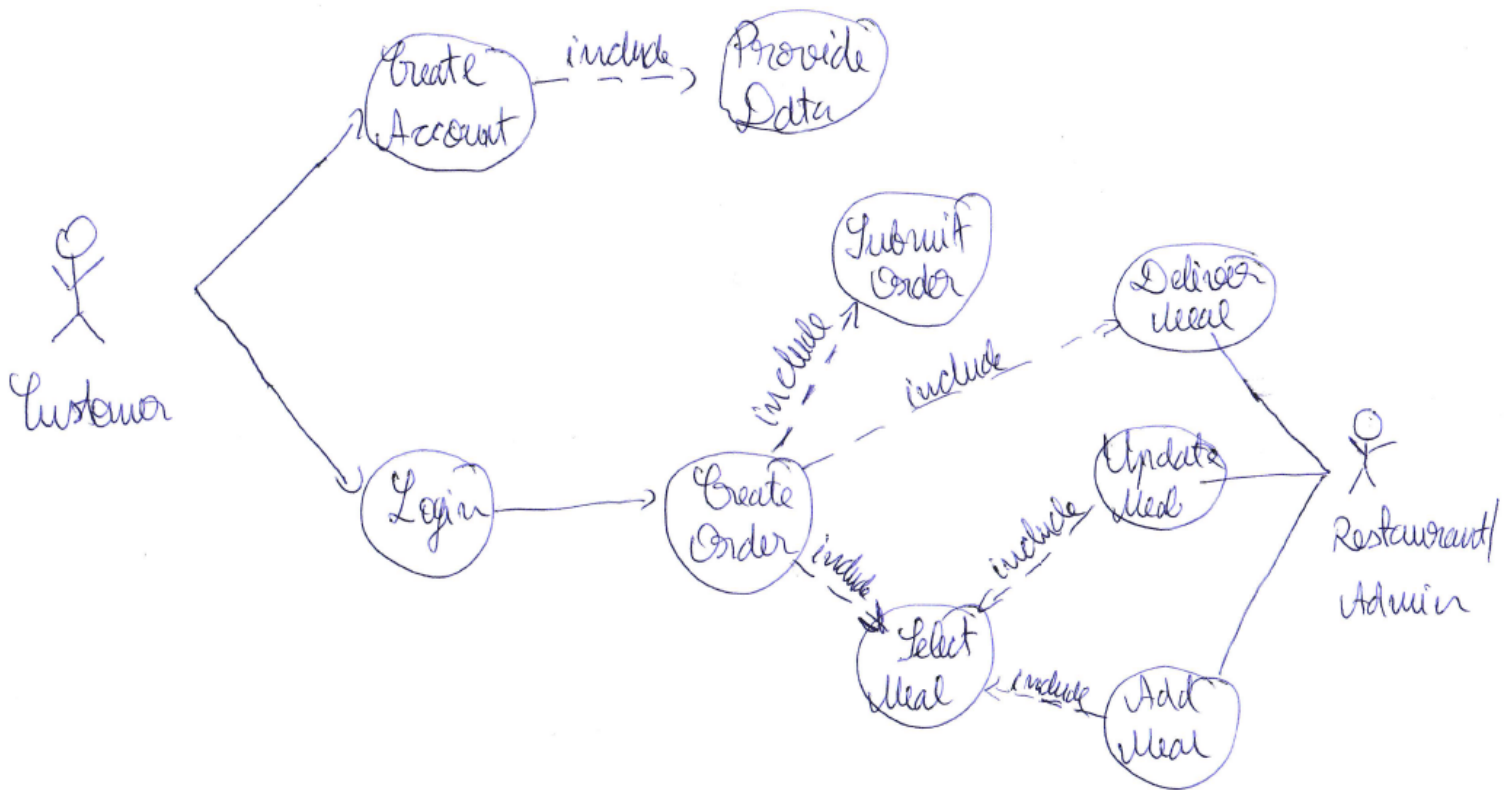
- Product – tabelul care contine date referitoare la un produs (in cazul nostru o mancare) si anume pretul produsului si numele produsului
- Warehouse – tabelul care contine date referitoare la stocare produselor, adica locul unde se gaseste produsul si cantitatea produselor
- Customer – tabelul care contine date referitoare la un client/utilizator, adica numele clientului, numele utilizator, parola, adresa clientului (city, str, nr), numarul de telefon mobil al clientului, adresa de mail si codul contului bancar, folosit pentru platirea produselor comandate
- Admin – tabelul care contine date referitoare la administratorul bazei de date si a aplicatiei, adica numele administratorului, numele de utilizator si parola
- Orders – tabelul folosit pentru despartirea relatiei many – to – many in doua relatii one – to many, adica contine un foreign key compus din id-ul produsului comandat si id-ul cumparatorului, care a facut comanda
- OPDept – tabelul folosit pentru stocarea comenzilor, contine un foreign key la tabelul orders si contine data cumpararii si data maxima pentru livrare

Pentru gestiunea corecta a datelor din baza de date, am folosit un trigger, care verifica daca cantitatea din Warehouse a produselor este mai mare decat 0 (nu putem sa avem un numar negativ de produse) si daca numrul produselor ajunge la 0, automat sterge produsul din warehouse si implicit din tabelul Product, astfel cumparatorul nu va mai vedea aceasta mancare in meniu.

Codul acestui trigger este:

```
USE restaurante;
DELIMITER $$
CREATE TRIGGER `trg_stock_cantity_update` BEFORE UPDATE ON warehouse
FOR EACH ROW
BEGIN
    IF ( NEW.pieces <= 0) THEN
        -- bad data
        SET @ID_P = NULL;
        SELECT id_Product INTO @ID_P FROM warehouse WHERE pieces = 0;
        DELETE FROM product WHERE id = @ID_P;
    END IF;
END$$
DELIMITER ;
```

Usecase Diagram:

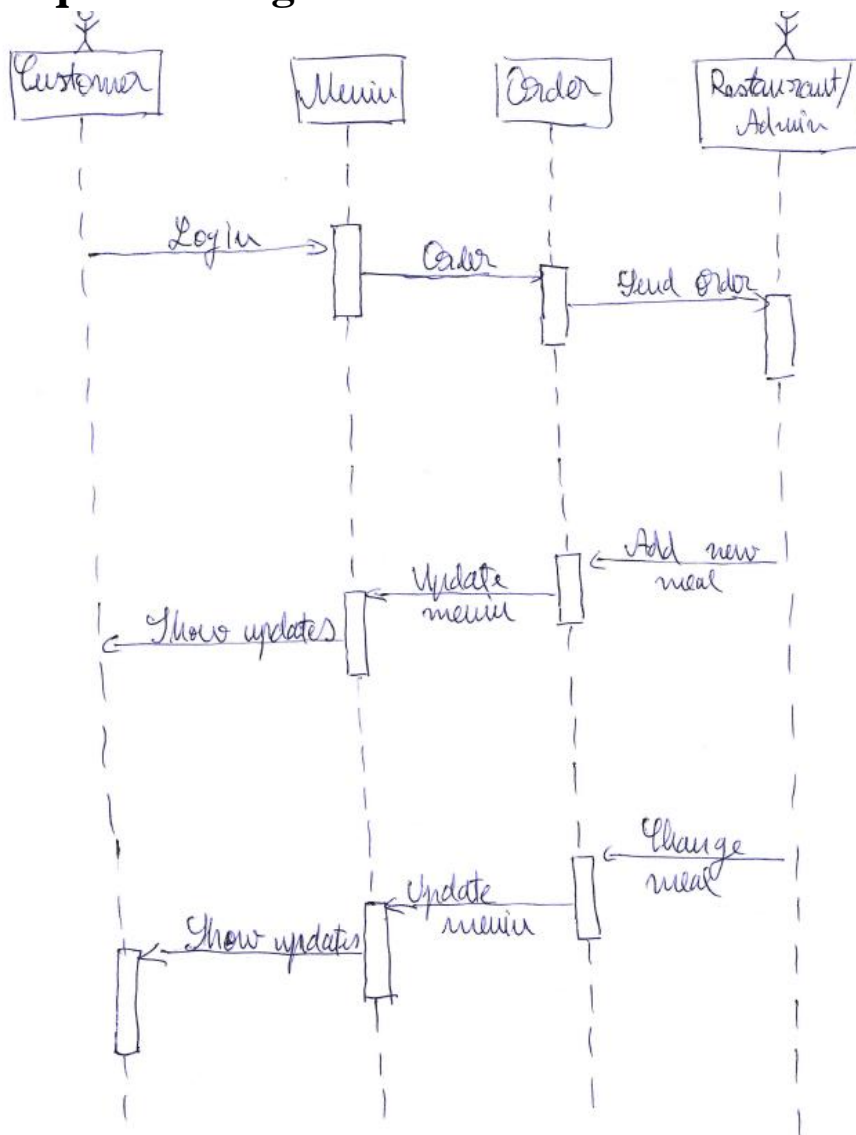


Explicatii:

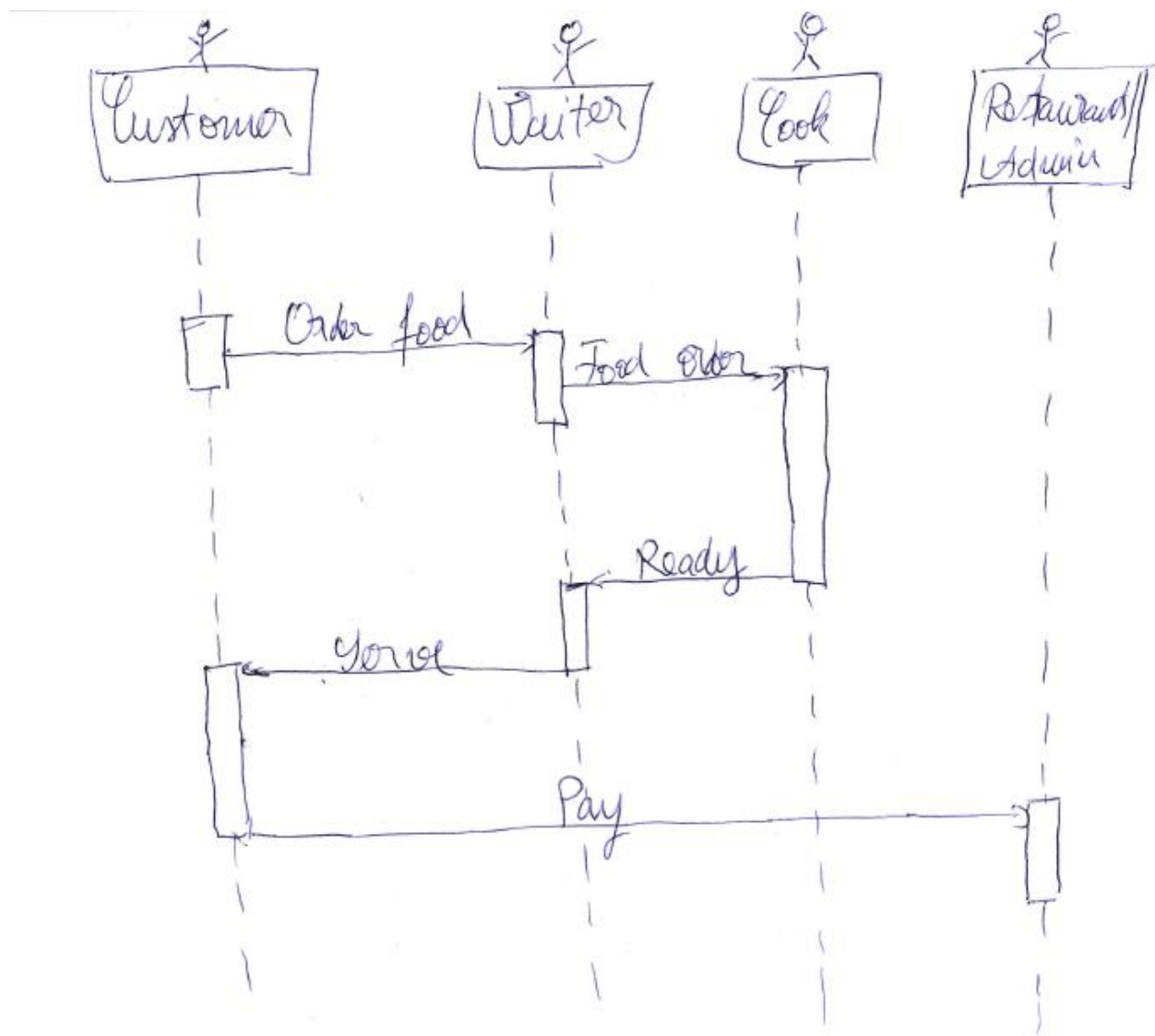
Utilizatorul, in cazul in care nu are un cont, poate sa creeze un cont de utilizator, adaugand astfel date personale, folosite pentru livrarea produselor si pentru plata produselor.

Dupa crearea contului, utilizatorul poate sa logeze in contul sau propriu, in care va vedea pe partea stanga lista produselor si pretul fiecarui produs, iar in partea dreapta apare propriul cos de vanzare, un poate sa adauge sa poate sa elimine produse. La finalul cumpararii, utilizatorul transmite lista comenzilor la restaurant sau la admin. Adminul la randul sau poate sa adauge produse noi, poate sa elimine produse sau poate sa modifice datele unui produs.

Sequence Diagram 1:



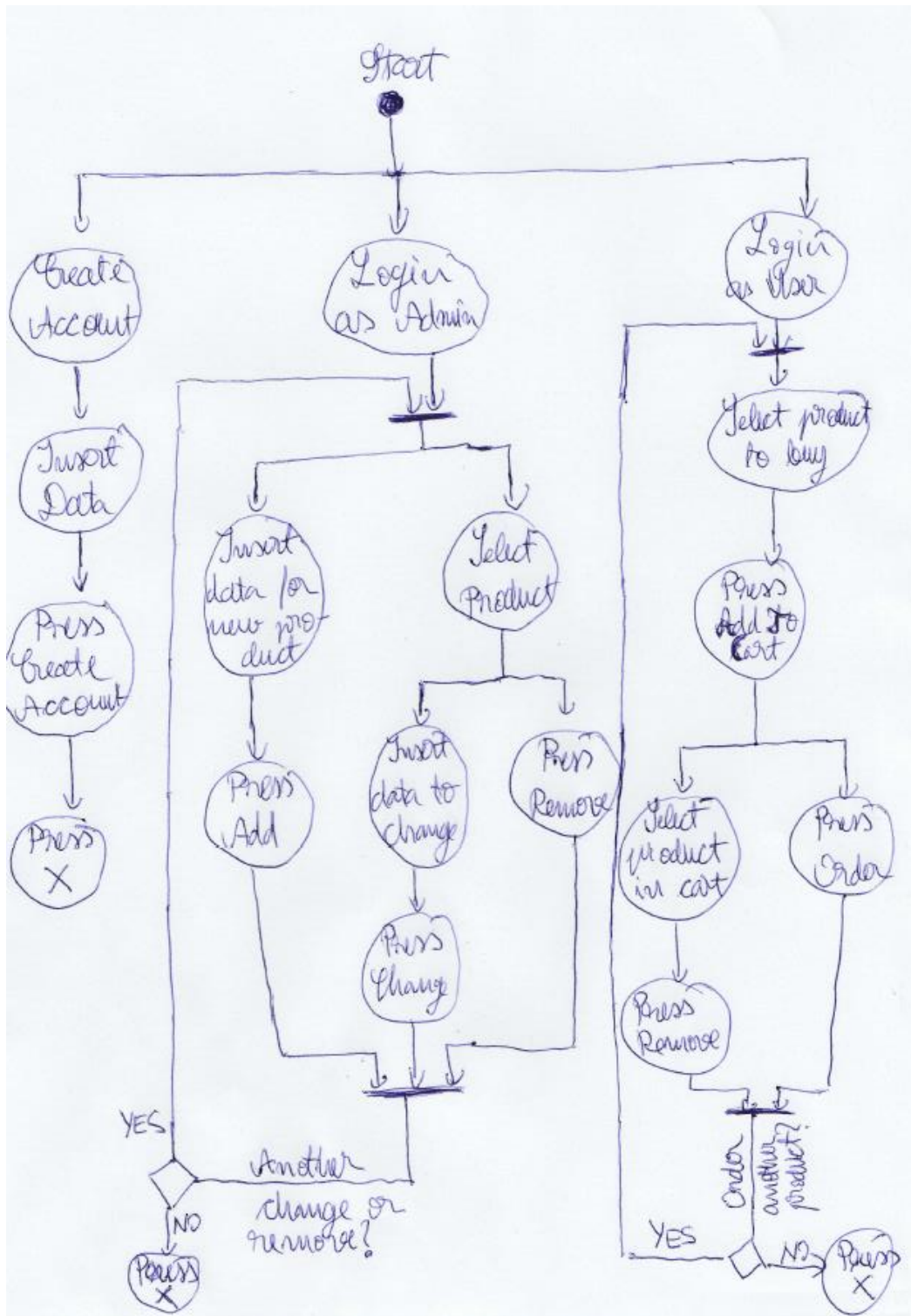
Sequence diagram 2:



Explicatii:

Cumparatorul face o comanda, chelnerul livreaza comanda la bucatar, care dupa ce mancarea e gata facuta, il cheama pe chelner, care livreaza mancarea la client. Clientul, dupa ce a mancat, plateste restaurantului (aici admin) pentru mancarea gustoasa si pleaca.

Activity Diagram:



Explicatii:

Dupa pornirea aplicatiei, utilizatorul poate sa-si creeze un cont. In acest context, utilizatorul introduce datele personale si apoi, dand click pe Create Account un cont nou va fi creat. Dupa apasarea X-ului din partea dreapta sus a aplicatiei, utilizatorul ajunge la pagina de Login. Dupa introducerea username – ului si a parolei corespunzatoare, utilizatorul intra pe pagina principal, unde poate sa creeze niste comenzi. Dupa alegerea produsului dorit, daca apasa butonul Add, produsul respectiv va fi adaugat la cosul propriu de cumparaturi. Daca urilizatorul vrea sa elimine un produs din cosul de cumparaturi, trebuie sa aleaga produsul din cosul de cumparaturi si sa dea click pe butonul Remove. Dand click pe Order, comanda va fi inregistrata in baza de date. La final, utilizatorul poate sa iasa din aplicatie, dand click pe X-ul din partea dreapta – sus al aplicatiei.

Diagrama UML pentru baza de date

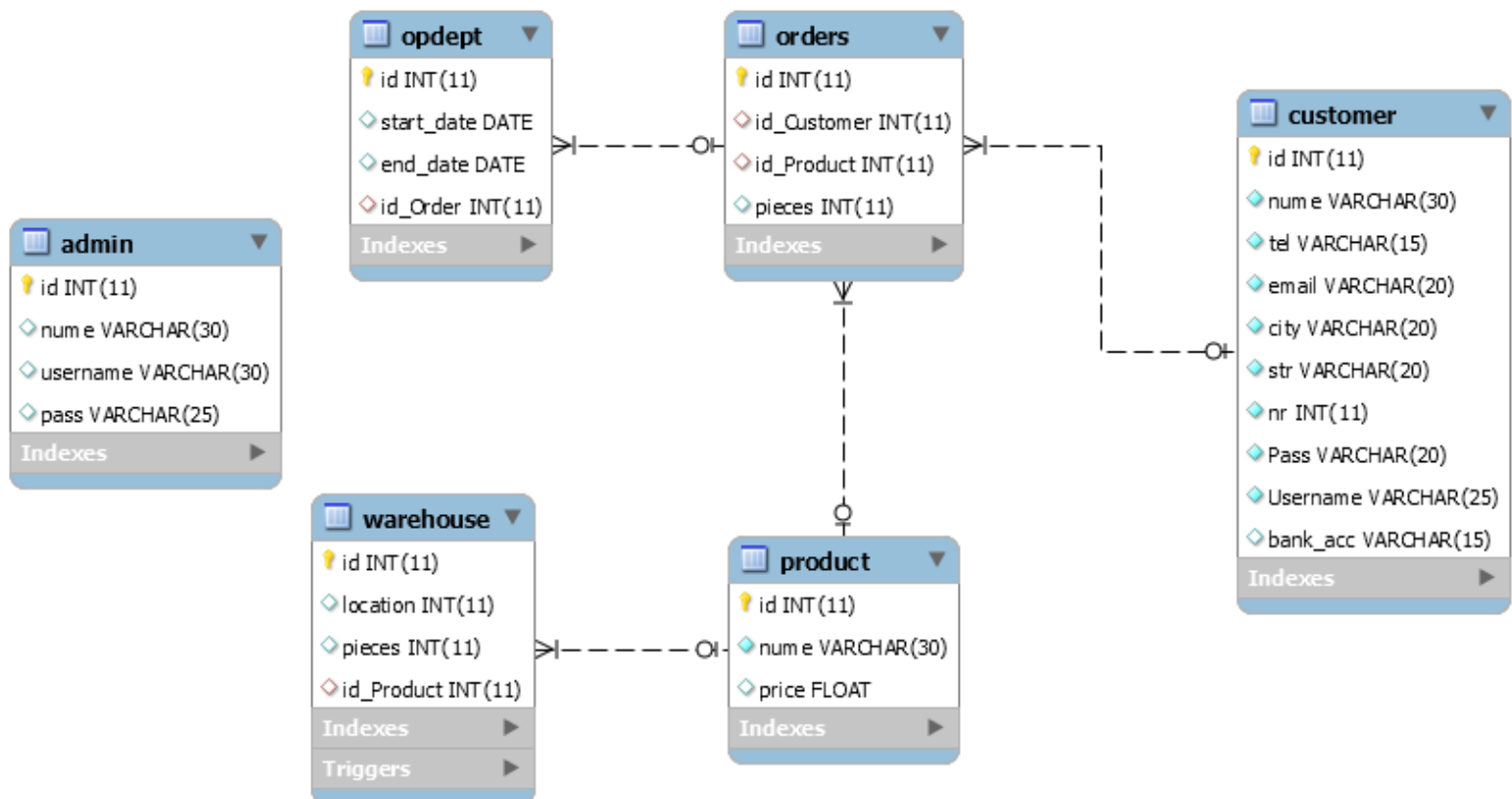
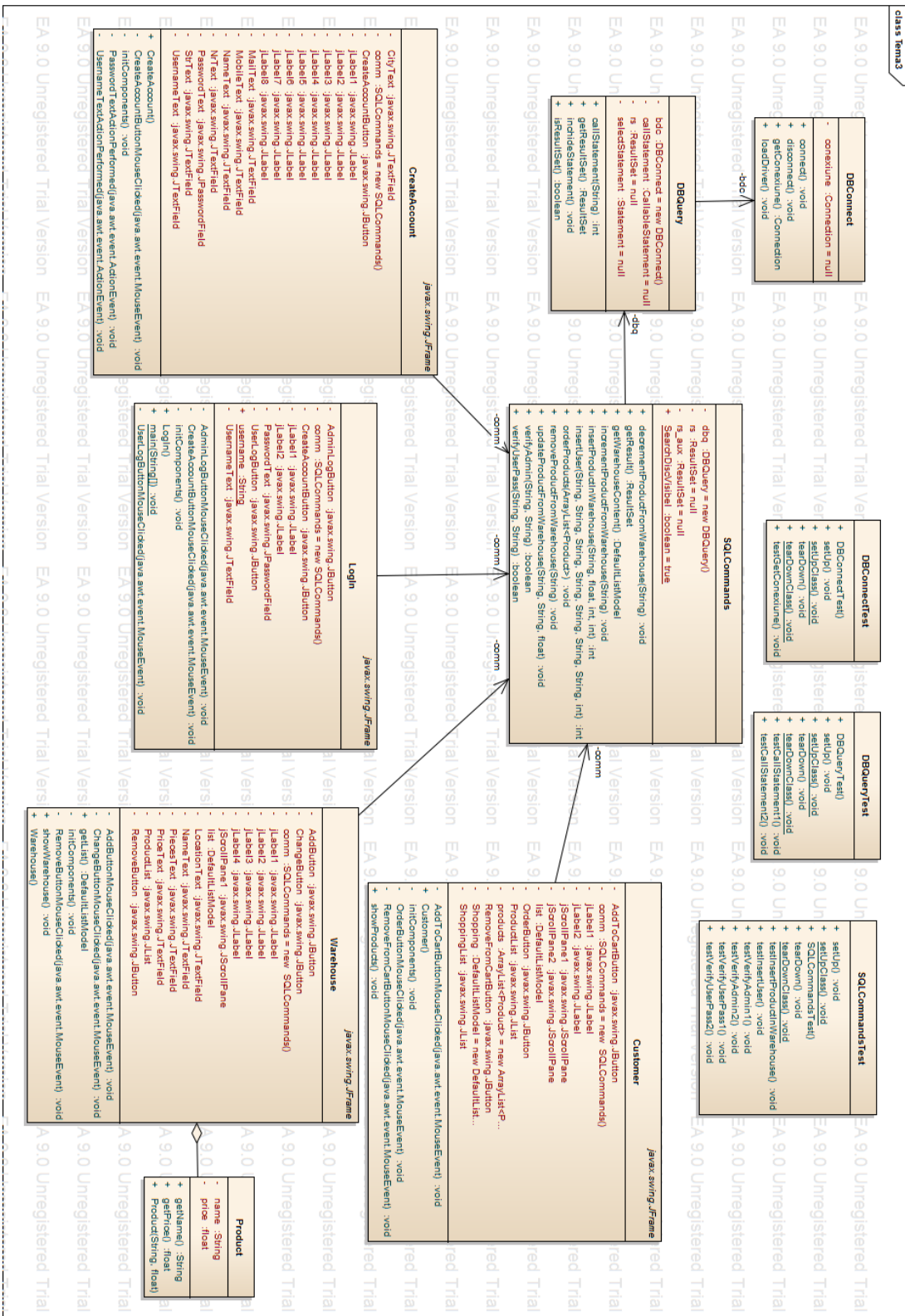


Diagrama UML pentru clasele Java:



Clasele folosite:

Create Account: clasa pentru cu interfata grafica, care conctine textfield-uri pentru introducerea datelor personale al utilizatorului, adica textfield – uri pentru nume, adresa (city, strada, nr), adresa de mail, numarul de telefon mobil, numele utilizatorului, parola, contul bancar.

Customer: clasa cu interfata grafica, cu ajutorul careia se realizeaza cumpararea produselor, contine doua liste de produse, prima lista e lista produselor din stoc iar a doua lista contine produsele adaugate in cosul de cumparaturi. Cu ajutorul butoanelor Add si Remove se poate adauga sau elimina produse din cosul de cumparaturi, iar cu butonul Order se salveaza comanda in baza de date.

DBConnect: clasa folosita pentru realizarea conexiunii intre baza de date MySql si intre aplicatia scrisa in Java. Contine metode pentru conectare, deconectare si incarcarea driverului.

DBQuery – clasa folosita pentru creare interogarilor MySql. Are ca metode, callStatement, pentru realizarea interogarii, inchideStatement pentru inchiderea interogarii si getResultSet pentru returnarea rezultatelor gasite.

LogIn – clasa cun interfata grafica care contine textfielduri pentru username si pentru password, si aici se poate dechide fereastra pentru crearea unui cont nou.

Product – clasa pentru stocarea datelor pentru produse, contine doua fielduri, si anume numele produsului si pretul produsului

Warehouse – clasa cu interfata grafica, care realizeaza gestiunea stocului de catre administratorul bazei de date si a aplicatiei.

SQLCommands – clasa care contine toate comenzile pentru gestiunea datelor in baza de date, interogari, inserari, actualizari, stergeri etc.

Exemple de metode din SQLCommands:

```
/**
 * Calculeaza numarul total de produse in Warehouse
 * @return int - numar total de produse
 */

public int findTotalInWarehouse()
{
    dbq.callStatement( " SELECT SUM ( pieces ) AS total FROM warehouse " );
    int total = 0 ;
    try
    {
```

```

        rs = dbq.getResultSet( ) ;
        if( rs.next( ) )
        {
            total = rs.getInt( "total" ) ;
        }
    }
    catch( SQLException ex )
    {
        JOptionPane.showMessageDialog( null , ex.getMessage( ) ,
                                        "Error", JOptionPane.ERROR_MESSAGE);
    }

    return total;
}

/**
 * Verifica daca numele utilizatorului si parola corespunde cu admin
 * @return true - admin existent, false - admin inexistent
 */
public boolean verifyAdmin(String username, String pass)
{
    dbq.callStatement("SELECT id FROM Admin WHERE username = '" + username + "'
AND pass = '" + pass + "'");
    if(dbq.isResultSet()) return true;
    else return false;
}

/**
 * Insereaza un produs in Warehouse.
 * @return void
 */
public int insertProductInWarehouse(String name, float price, int location, int pieces)
{
    dbq.callStatement("INSERT INTO Product(num,price) VALUES('" + name + "'," + price
+ ")");
    dbq.callStatement("SELECT id FROM Product WHERE num LIKE '" + name + "%'");
    rs = dbq.getResultSet();
    int succes = 0;
    try
    {
        rs.next();
        int index = rs.getInt("id");
    }
}

```

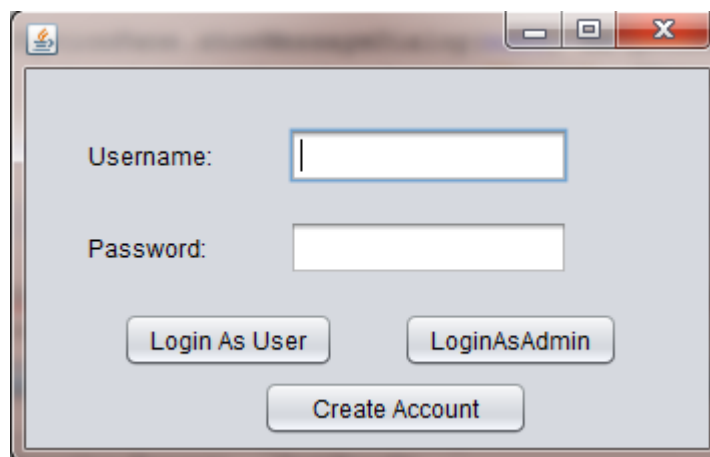
```

        succes = dbq.callStatement("INSERT INTO Warehouse(id_Product,location,pieces)
VALUES(" + index + "," + location + "," + pieces + ")");
    }
    catch(SQLException ex)
    {
        JOptionPane.showMessageDialog(null, ex.getMessage(),
        "Error", JOptionPane.ERROR_MESSAGE);
    }
    return succes;
}

```

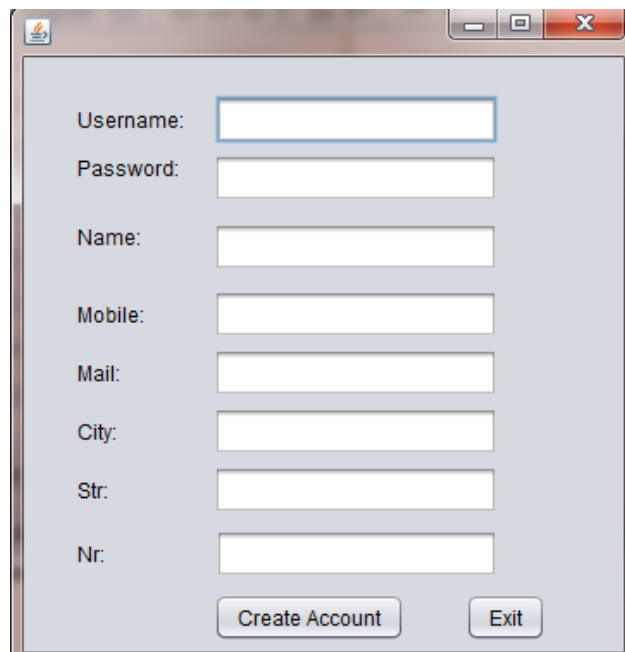
Interfata grafica:

La pornirea aplicatiei apare fereastra de LogIn, unde utilizatorul trebuie sa logheze in contul sau. Daca nu are cont, poate sa-si creeze dand click pe buton Create Account.



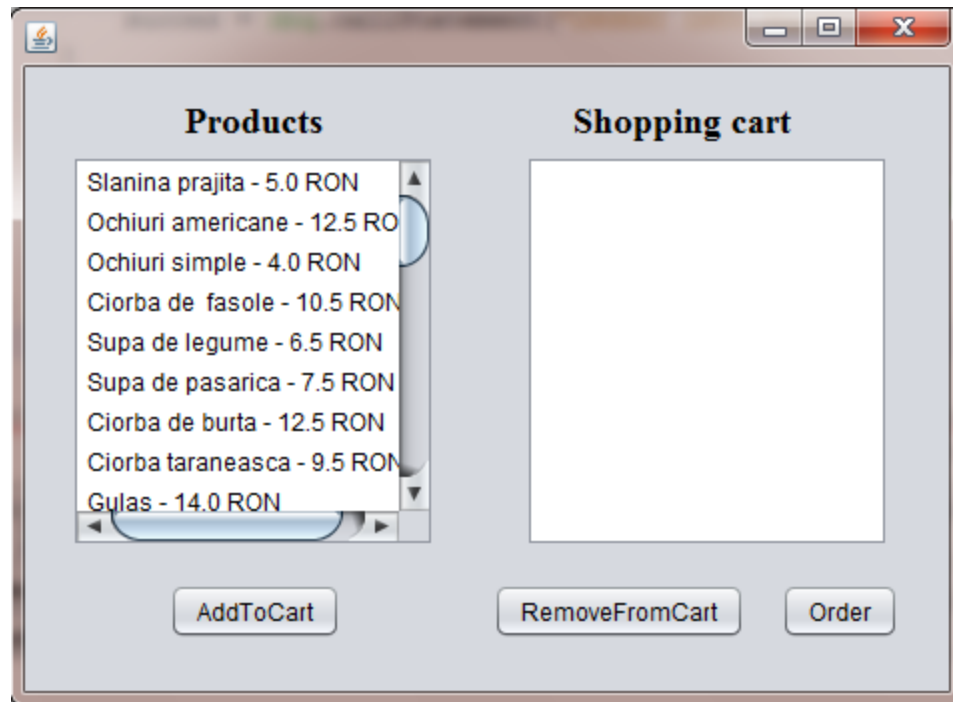
The screenshot shows a login window with a light blue background. It has two text input fields: 'Username:' and 'Password:'. Below the password field are three buttons: 'Login As User', 'LoginAsAdmin', and 'Create Account'. The window has a standard Windows-style title bar with minimize, maximize, and close buttons.

Pentru creare unui cont nou, utilizatorul trebuie sa introduca datele personale si apoi click pe buton Create Account.

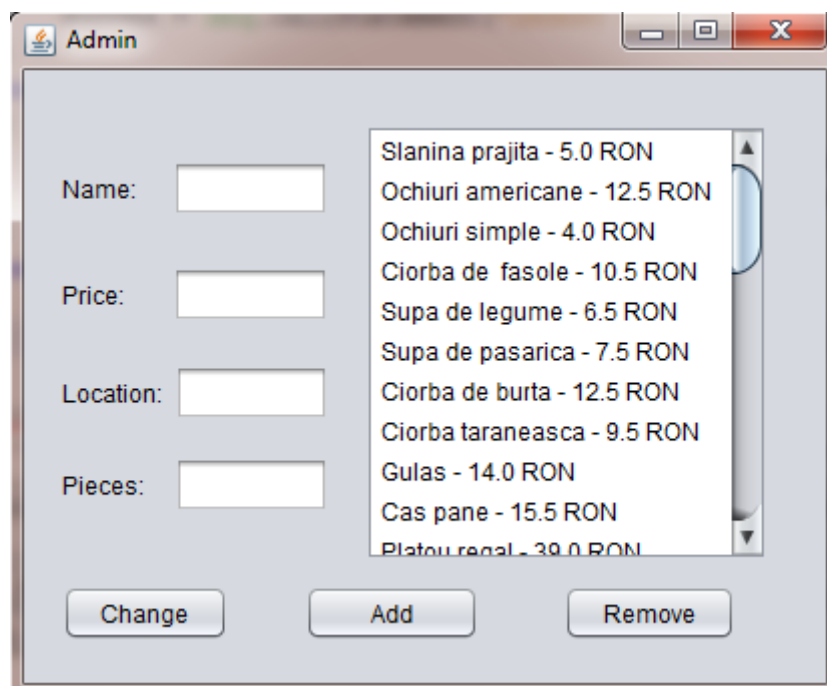


The screenshot shows a 'Create Account' window with a light blue background. It contains eight text input fields: 'Username:', 'Password:', 'Name:', 'Mobile:', 'Mail:', 'City:', 'Str:', and 'Nr:'. At the bottom are two buttons: 'Create Account' and 'Exit'. The window has a standard Windows-style title bar with minimize, maximize, and close buttons.

Dupa crearea contului, utilizatorul poate sa logeze in contul sau propriu, in care va vedea pe partea stanga lista produselor si pretul fiecarui produs, iar in partea dreapta apara propriul cos de vanzare, un poate sa adauge sa poate sa elimine produse. La finalul cumpararii, utiliztorul transmite lista comenzilor la restaurant sau la admin. Adminul la randul sau poate sa adauge produse noi, poate sa elimine produse sau poate sa modifice datele unui produs.



Fereastra pentru admin:



Dupa introducerea datelor, administratorul poate sa modifice datele produsului selectat, poate sa adauge un produs nou, dand click pe Add sau poate sa elimine un produs, dand click pe Remove.

4. Testare

Simulatorul a fost testată de mai multe ori, pentru a vedea, dacă funcționează corect, si returnează rezultate corecte. Pentru testare am folosit un driver de testare, care testează fiecare metoda din program.

Am implementat trei clase de test, prima clasa, numita DBConnectTest testeaza conectarea corecta, clasa a doua DBQueryTest testeaza corectitudinea sql statement-urilor, iar a treia clasa, numita SQLCommandsTest testeaza gestiunea datelor din baza de date, verificand astfel corectitudinea interactiunii aplicatiei cu clientul si cu administratorul.

Exemple de test:

Pentru out-of-stock:

```
/**
 * Test2 of findProductByID method, of class SQLCommands. out-of-stock
 */
@Test
public void testfindProductByID2() {
    System.out.println("findProductByID");
    int id = 3;
    SQLCommands instance = new SQLCommands();
    Product expResult = null;
    Product result = instance.findProductByID(id);
    assertEquals(expResult, result);
}
```

Pentru in-stock:

```
/**
 * Test2 of findProductByID method, of class SQLCommands. in-stock
 */
@Test
public void testfindProductByID1() {
    System.out.println("findProductByID");
    int id = 1;
    SQLCommands instance = new SQLCommands();
    String expResult = "Slanina prajita";
    String result = instance.findProductByID(id).getName();
    assertEquals(expResult, result);
}
```

```
}
```

Pentru total:

```
/**
 * Test2 of findProductByID method, of class SQLCommands. out-of-stock
 */
@Test
public void testfindTotalInWarehouse() {
    System.out.println("findTotalInWarehouse");
    SQLCommands instance = new SQLCommands();
    int expResult = 63;
    int result = instance.findTotalInWarehouse();
    assertEquals(expResult, result);
}
```

5. Rezultate

După testarea programului vedem că programul funcționează perfect, verifică datele de intrare , afișează rezultatele pe interfața grafică, și efectuează comenzile primite de la utilizatorul programului.

6. Concluzii, ce s-a invatat din tema, dezvoltari ulterioare

Lucrand la acest proiect am invatat realizarea conexiunii unei baze de date cu Java, am invatat folosirea interogarilor, insert-urilor si folosirea metodelor update si delete in MySql.

Se mai pot introduce rapoarte, exporturi și importuri de date in formate xls, cvs, pdf, json, etc.

Calcularea mai multor profituri, si totaluri in cadrul depozitului un sistem de siguranta la plata , aplicatia incepand sa tina cont de pretul produselor cumparate .

Implementarea programului pentru mai multe firme, afisând produsul de la firma care vinde cu cel mai mic pret.

Adăugarea unui câmpuri pentru fiecare produs, unde clientii pot lăsa comentariile, părerile despre produse.

7. Bibliografie

Carte:

Effective Java, 2nd edition

Internet:

<https://www.youtube.com/> -tutoriale

<http://stackoverflow.com/>

<http://docs.oracle.com/javase/tutorial/>

http://www.coned.utcluj.ro/~salomie/PT/Lab/Tema3_HW3.pdf