**PROCESAREA COMENZILOR CLIENȚILOR PENTRU UN DEPOZIT**

**Nume prenume: Dulău Marius Cristian**

**Grupa: 30225**

**Profesor laborator: Asist. Antal Marcel**

Contents

[1. Cerințe funcționale 3](#_Toc476131445)

[2. Obiective 3](#_Toc476131446)

[2.1. Obiectiv principal 3](#_Toc476131447)

[2.2. Obective secundare 3](#_Toc476131448)

[3. Analiza problemei](#_Toc476131449) 4

[4. Proiectare](#_Toc476131450) 5

[4.1. Structuri de date](#_Toc476131451) 5

[4.2. Diagrama de clase](#_Toc476131452) 5

[4.3. Algoritmi](#_Toc476131453) 7

[5. Implementare](#_Toc476131454) 14

[6. Testare](#_Toc476131455) 15

[7. Concluzii si dezvoltări ulterioare](#_Toc476131456) 26

[8. Bibliografie](#_Toc476131457) 27

# Cerințe funcționale

Dezvoltați o aplicație care să proceseze comenzile clienților pentru un depozit. Aplicația este capabilă să:

* Autentifice un client sau administratorul
* Creeze conturi noi pentru clienți
* Inserarează informații noi într-o bază de date
* Preia informațiile din baza de date într-un tabel
* Afișează mesaje de eroare
* Prelucrează facturi, conținând datele comenzii
* Trimite mail la client cu factura

# Obiective

## Obiectiv Principal:

Propuneți, proiectați și implementați un sistem de procesare a comenzilor clienților pentru un depozit.

## Obective Secundare:

|  |  |  |
| --- | --- | --- |
| **Obiectiv Secundar** | **Descriere** | **Capitol** |
| Dezvoltarea de use case-uri și scenarii | Descrierea în limbaj natural a scenariilor de utilizare și funcționare aplicației, însoțite de o diagramă. | 3 |
| Alegerea structurilor de date | Pentru a construi un tabel a fost nevoie de o listă pentru a stoca toate obiectele corespunzătoare rândurilor din tabel. | 4 |
| Împărțirea pe clase | Clasele au fost structurate în mai multe pachete. Aceste pachete sunt : bll, bll.validators, connection, dao, model, presentation, start și mail. | 5 |
| Dezvoltarea algoritmilor | S-au dezvoltat algoritmi pentru selectare, inserare, modificare și ștergere de date. De asemenea au fost implementați algoritmi pentru a genera un tabel cu date. | 5 |
| Implementarea soluției | Implementare interfață grafică și a metodelor specifice operațiilor pe baze de date | 5 |
| Testare | Testarea efectuării operațiilor în interfața grafică | 6 |

# Analiza Problemei

Aplicația are 2 actori: cumpărătorul și administratorul. Cumpărătorul poate selecta un produs și cantitatea acestuia. Pe baza acestor date se generează o factură. Administratorul poate să adăuge un produs, să modifice detaliile unui produsului, să steargă un produs și să vizualizeze clienții.

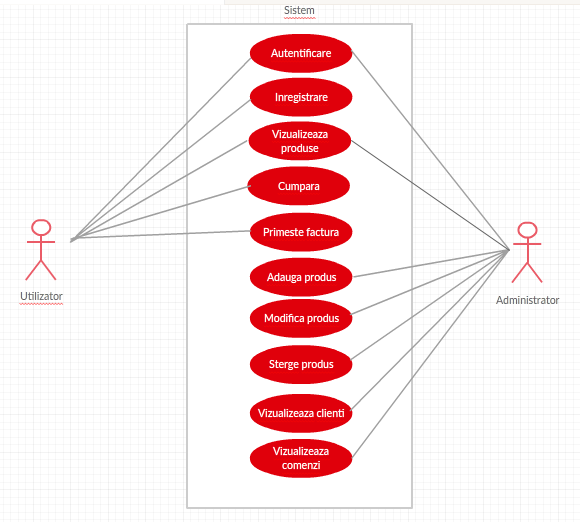


Fig. 1. Diagrama use case

# Proiectare

## Structuri de date

S-a folosit structura de date List pentru a stoca obiecte ce corespund liniilor dintr-un tabel din baza de date. Ulterior, această listă a fost utilizată pentru a forma un tabel.

## Diagrama de clase

1. **Baza de date**

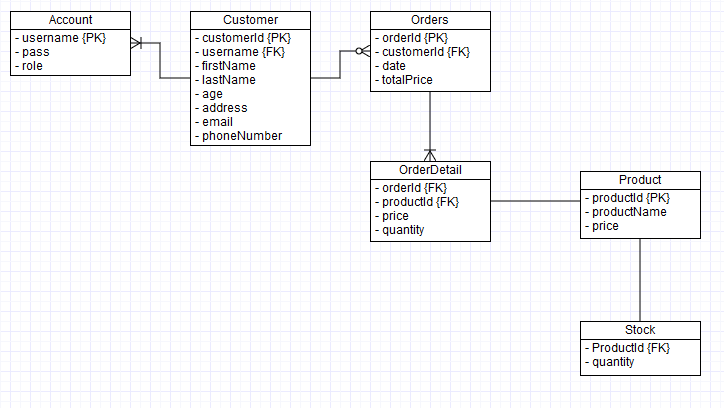
****

Fig. 2. Diagrama UML a bazei de date

1. **Java**

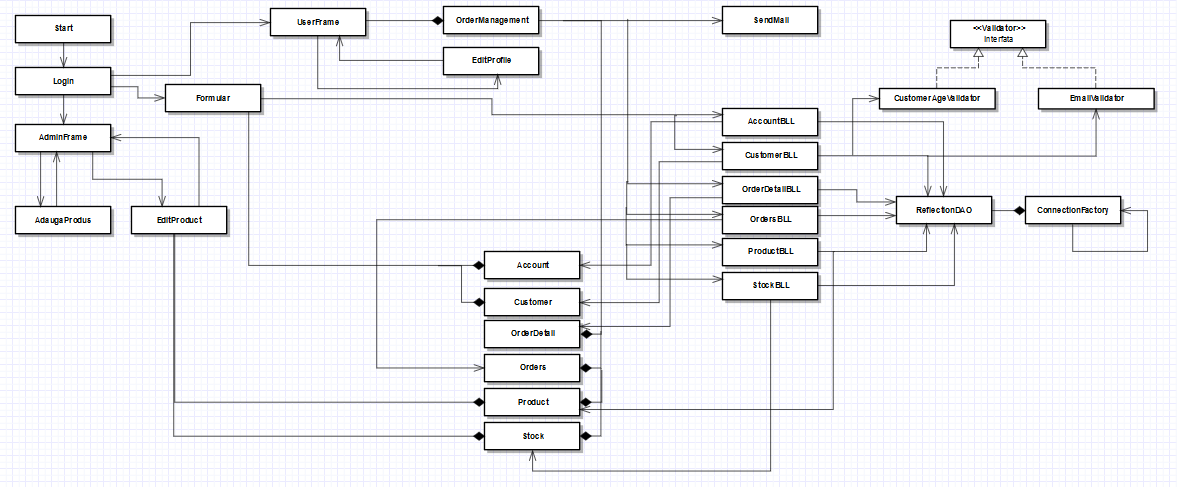
****

Fig. 3. Diagrama UML a aplicației Java

## Algoritmi

Pentru fiecare interogare SQL s-a creat o clasă care utilizează reflection și generează o interogare adecvată după tipul obiectului transmis ca parametru. S-au creat mai întâi metode pentru a genera un String potrivit interogării. Apoi s-au formulat metodele pentru executarea instrucțiunii din String. Exemplificăm mai jos pentru operația de inserare.

**public** **static** String insertQuery(Object object)

{

String insertStatementString = "insert into " + object.getClass().getSimpleName() + "(";

**int** nrFields = object.getClass().getDeclaredFields().length - 1;

**int** copyNrFields = nrFields;

**for**(Field field : object.getClass().getDeclaredFields())

{

field.setAccessible(**true**);

insertStatementString = insertStatementString + field.getName();

**if**(copyNrFields > 0)

{

insertStatementString = insertStatementString + ",";

copyNrFields--;

}

}

insertStatementString = insertStatementString + ") values (";

**for**(Field field : object.getClass().getDeclaredFields())

{

field.setAccessible(**true**);

insertStatementString = insertStatementString + "?";

**if**(nrFields > 0)

{

insertStatementString = insertStatementString + ",";

nrFields--;

}

}

insertStatementString = insertStatementString + ")";

**return** insertStatementString;

}

**public** **static** **void** insert(Object object)

{

Connection dbConnection = ConnectionFactory.*getConnection*();

PreparedStatement insertStatement = **null**;

String insertStatementString = *insertQuery*(object);

**try**

{

insertStatement = dbConnection.prepareStatement(insertStatementString);

**int** i = 1;

**for**(Field field : object.getClass().getDeclaredFields())

{

field.setAccessible(**true**);

Object value;

**try**

{

value = field.get(object);

**if**(value.getClass() == String.**class**) insertStatement.setString(i, (String)value);

**else** **if**(value.getClass() == Integer.**class**) insertStatement.setInt(i, (**int**)value);

**else** **if**(value.getClass() == Character.**class**) insertStatement.setString(i, String.*valueOf*(value));

**else** **if**(value.getClass() == Double.**class**) insertStatement.setDouble(i, (**double**)value);

i++;

}

**catch** (IllegalArgumentException e)

{

e.printStackTrace();

}

**catch** (IllegalAccessException e)

{

e.printStackTrace();

}

}

insertStatement.executeUpdate();

}

**catch**(SQLException e)

{

***LOGGER***.log(Level.***WARNING***, "ReflectionDAO : insert " + e.getMessage());

}

**finally**

{

ConnectionFactory.*close*(insertStatement);

ConnectionFactory.*close*(dbConnection);

}

}

Pentru a verifica cine se autentifică la baza de date avem un obiect Account în care punem valorile date în casetele de text pentru username și parolă. După ce verificăm cu baza de date obșinem obiectul cu toate câmpurile necesare. Dacă rolul său este de simplu utilizator se deschide fereastra pentru cumpărători unde se pot vizualiza produse și schimba datele personale. Dacă rolul utilizatorului este de administrator se va deschide o fereastră unde se pot vizualiza produse/clienți, adăuga/modifica/șterge produse.

btnLogin.addActionListener(**new** ActionListener(){

@Override

**public** **void** actionPerformed(ActionEvent e)

{

Account a = **new** Account(getUsername(), getPassword(), 'U');

AccountBLL accountBLL = **new** AccountBLL();

Object[] campuri = {"username", "pass"};

**try**

{

a = (Account)accountBLL.findByUser(a, campuri);

**if**(a.getRole()=='U')

{

userFrame = **new** UserFrame("User : " + a.getUsername(), a);

userFrame.setVisible(**true**);

}

**else**

{

adminFrame = **new** AdminFrame("Admin");

adminFrame.setVisible(**true**);

}

setVisible(**false**);

dispose();

}

**catch**(Exception ex)

{

JOptionPane.*showMessageDialog*(panel, ex.toString());

}

}

});

Atât la fereastra pentru client, cât și la cea pentru administrator avem un tabel care trebuie populat cu date. Se folosesc tehnici de reflection. Mai întâi se creează o listă cu obiectele ce se introduc în tabel.

**public** **static** List createListOfObject(Object obj)

{

List<Object> objects = **new** ArrayList<Object>();

Connection dbConnection = ConnectionFactory.*getConnection*();

PreparedStatement selectStatement = **null**;

String selectStatementString = *selectQuery*(obj, **null**);

ResultSet rs = **null**;

**try**

{

selectStatement = dbConnection.prepareStatement(selectStatementString);

rs = selectStatement.executeQuery();

Account a = **null**;

Customer c = **null**;

OrderDetail od = **null**;

Orders o = **null**;

Product p = **null**;

Stock s = **null**;

Object toReturn = **null**;

**while**(rs.next())

{

**int** index = 0;

Object[] variables = **new** Object[obj.getClass().getDeclaredFields().length];

**for**(Field field : obj.getClass().getDeclaredFields())

{

variables[index++] = rs.getString(field.getName());

}

**try**

{

**if**(obj.getClass().getSimpleName().compareTo("Account") == 0)

{

a = Account.**class**.getConstructor(String.**class**, String.**class**, **char**.**class**).newInstance(variables[0], variables[1], String.*valueOf*(variables[2]).charAt(0));

toReturn = a;

}

**else** **if**(obj.getClass().getSimpleName().compareTo("Customer") == 0)

{

c = Customer.**class**.getConstructor(**int**.**class**, String.**class**, String.**class**, String.**class**, **int**.**class**, String.**class**, String.**class**, String.**class**).newInstance(Integer.*parseInt*((String)variables[0]), variables[1], variables[2], variables[3], Integer.*parseInt*((String)variables[4]), variables[5], variables[6], variables[7]);

toReturn = c;

}

**else** **if**(obj.getClass().getSimpleName().compareTo("Product") == 0)

{

p = Product.**class**.getConstructor(**int**.**class**, String.**class**, **double**.**class**).newInstance(Integer.*parseInt*((String)variables[0]), variables[1], Double.*valueOf*((String)variables[2]));

toReturn = p;

}

**else** **if**(obj.getClass().getSimpleName().compareTo("Stock") == 0)

{

s = Stock.**class**.getConstructor(**int**.**class**, **int**.**class**).newInstance(Integer.*parseInt*((String)variables[0]), Integer.*parseInt*((String)variables[1]));

toReturn = s;

}

**else** **if**(obj.getClass().getSimpleName().compareTo("Orders") == 0)

{

o = Orders.**class**.getConstructor(**int**.**class**, **int**.**class**, String.**class**, **int**.**class**).newInstance(Integer.*parseInt*((String)variables[0]), Integer.*parseInt*((String)variables[1]), variables[2], Integer.*parseInt*((String)variables[3]));

toReturn = o;

}

**else** **if**(obj.getClass().getSimpleName().compareTo("OrderDetail") == 0)

{

od = OrderDetail.**class**.getConstructor(**int**.**class**, **int**.**class**, **double**.**class**, **int**.**class**).newInstance(Integer.*parseInt*((String)variables[0]), Integer.*parseInt*((String)variables[1]), Double.*valueOf*((String)variables[2]), Integer.*parseInt*((String)variables[3]));

toReturn = od;

}

objects.add(toReturn);

}

**catch** (InstantiationException | IllegalAccessException | IllegalArgumentException | InvocationTargetException | NoSuchMethodException | SecurityException e)

{

e.printStackTrace();

}

}

}

**catch**(SQLException e)

{

***LOGGER***.log(Level.***WARNING***, "ReflectionDAO : find " + e.getMessage());

}

**finally**

{

ConnectionFactory.*close*(rs);

ConnectionFactory.*close*(selectStatement);

ConnectionFactory.*close*(dbConnection);

}

**return** objects;

}

Pentru a crea tabelul se folosește algoritmul descris mai jos:

**public** **static** JTable createTable(List<Object> objects)

{

Object[] columnNames = **new** Object[objects.get(0).getClass().getDeclaredFields().length];

Object[][] rowData = **new** Object[objects.size()][objects.get(0).getClass().getDeclaredFields().length];

**int** index = 0;

**for**(Field field : objects.get(0).getClass().getDeclaredFields())

{

columnNames[index++] = field.getName().toString();

}

index = 0;

**for**(**int** i = 0; i < objects.size(); i++)

{

index = 0;

**for**(Field field : objects.get(i).getClass().getDeclaredFields())

{

field.setAccessible(**true**);

Object value;

**try**

{

value = field.get(objects.get(i));

rowData[i][index++] = value;

}

**catch** (IllegalArgumentException e)

{

e.printStackTrace();

}

**catch** (IllegalAccessException e)

{

e.printStackTrace();

}

}

}

JTable table = **new** JTable(rowData, columnNames);

**return** table;

}

Pentru a ascunde datele de autentificare la baza de date s-a folosit un fișier care nu este inclus în proiect. S-au citit datele din fișier legate de autentificarea în baza de date.

**try**

{

scan = **new** Scanner(**new** File("C:/TP\_tema3.txt"));

*USER* = scan.nextLine();

*PASS* = scan.nextLine();

}

**catch**(FileNotFoundException e)

{

e.printStackTrace();

}

# Implementare

Aplicația este împățită în 8 pachete:

* bll – conține clase ce interacționează cu clasa de reflection; aceste clase sunt specifice tabelelor din baza de date
* bll.validators – conține o interfață ce e implementată de mai multe clase; acest pachet conține clase ce verifică datele ce urmează a fi introduse în baza de date
* connection – conține o clasă ce implementează conexiunea la baza de date
* dao – pachetul conține clasa ce implementează metode prin care se fac interogări asupra bazei de date
* model – clase ce ilustrează tabelele din baza de date
* presentation – conține clase ce implementează interfața grafică
* start – în acest pachet se află clasa cu metoda main a aplicației
* mail – pachetul conține clasa SendMail; aceasta are rolul de a transmite utilizatorului mail cu factura

Clasa ReflectionDAO din pachetul **dao** implementează metodele prin tehnici de reflection. Ea conține un singur câmp, ci anume un LOGGER pentru a afișa în consolă eventualele avertismente legate de interogarea bazei de date. Clasa conține metode specifice interogării unei baze de date.

Aceste metode sunt:

* **public** **static** String insertQuery(Object object) – generează comanda de inserare sub formă de String
* **public** **static** String selectQuery(Object object, Object[] campuri) – generează comanda de selecție asupra unui tabel sub formă de String
* **public** **static** String updateQuery(Object object) – generează comanda de update/modificare asupra unui tabel sub formă de String
* **public** **static** String deleteQuery(Object object) – generează comanda de delete/ștergere a unor date dintr-un tabel sub formă de String
* **public** **static** **void** insert(Object object) – efectuează inserarea datele unui obiect în tabelul din baza de date
* **public** **static** Object select(Object object, Object[] campuri) – efectuează selecția datelor unui obiect în tabelul din baza de date. Acest obiect este dat ca paramentru, precum și câmpurile după care se face căutarea
* **public** **static** **void** update(Object object) – efectuează modificarea de date a unui obiect dintr-un tabel din baza de date
* **public** **static** **void** delete(Object object) – efectuează inserarea datele unui obiect în tabelul din baza de date
* **public** **static** List createListOfObject(Object obj) – creează o listă cu toate datele dintr-un tabel. Tabelul este specificat prin intermediul obiectului dat ca parametru – tipul clasei sale
* **public** **static** JTable createTable(List<Object> objects) – preia o listă de obiecte și generează un tabel

Clasele din pachetul **model** conțin ca și variabile instanță cîmpurile din tabelul ce corespunde cu numele acelei clase. Metodele acestor clase sunt cele de get() și set().

În capitolul următor este prezentată testarea aplicației, precum și interfața grafică rezultată din clasele pachetului **presentation**.

# Testare

Pentru a urmări corectitudinea simulării, testarea a avut loc observând ceea ce se întâmplă în interfața grafică.

Fereastra de login conține o zonă pentru scrierea datelor de autentificare și două butoane (Fig. 4).

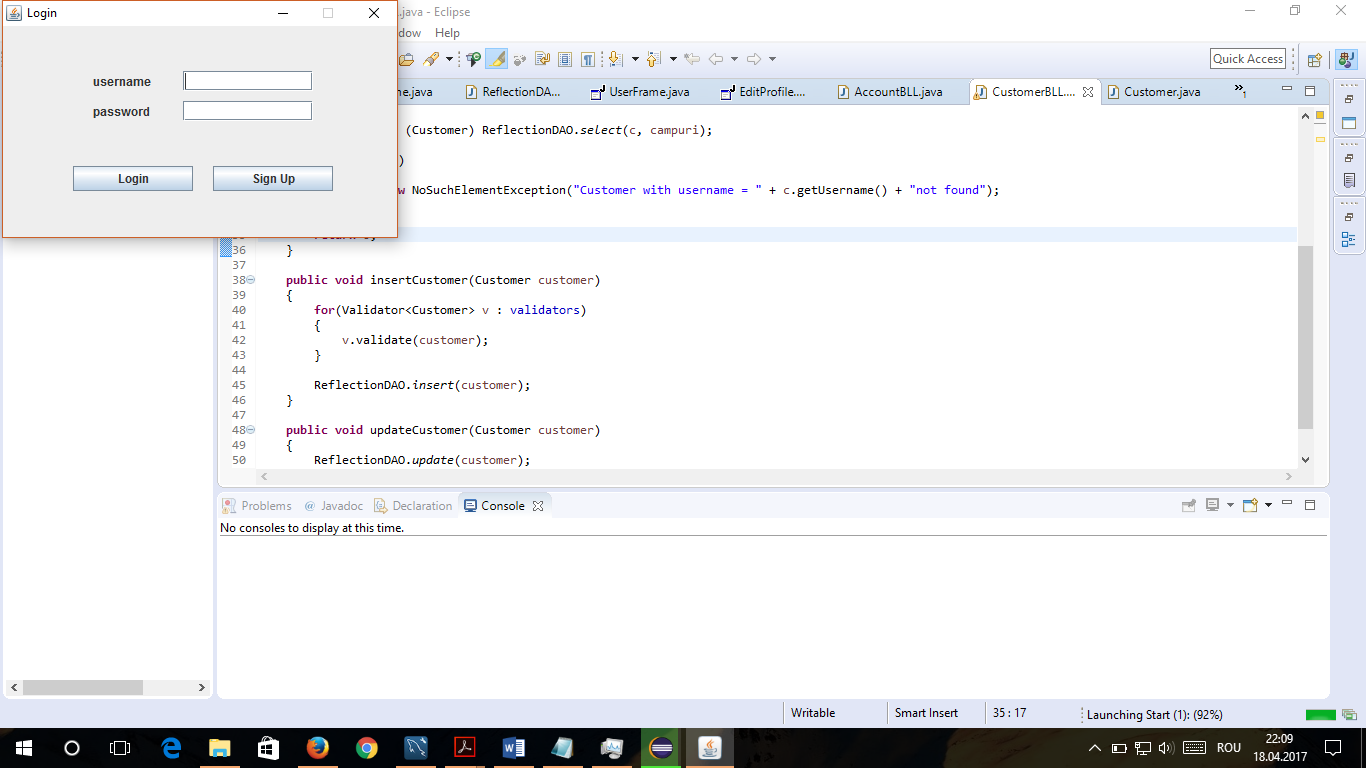


Fig. 4. Fereastra de login

În cazul în care selectăm opțiunea de “Sign Up” se va deschide o fereastră ce conține un formular (Fig. 5). După ce este completat și este apăsat butonul se va crea un cont pentru autentificare.

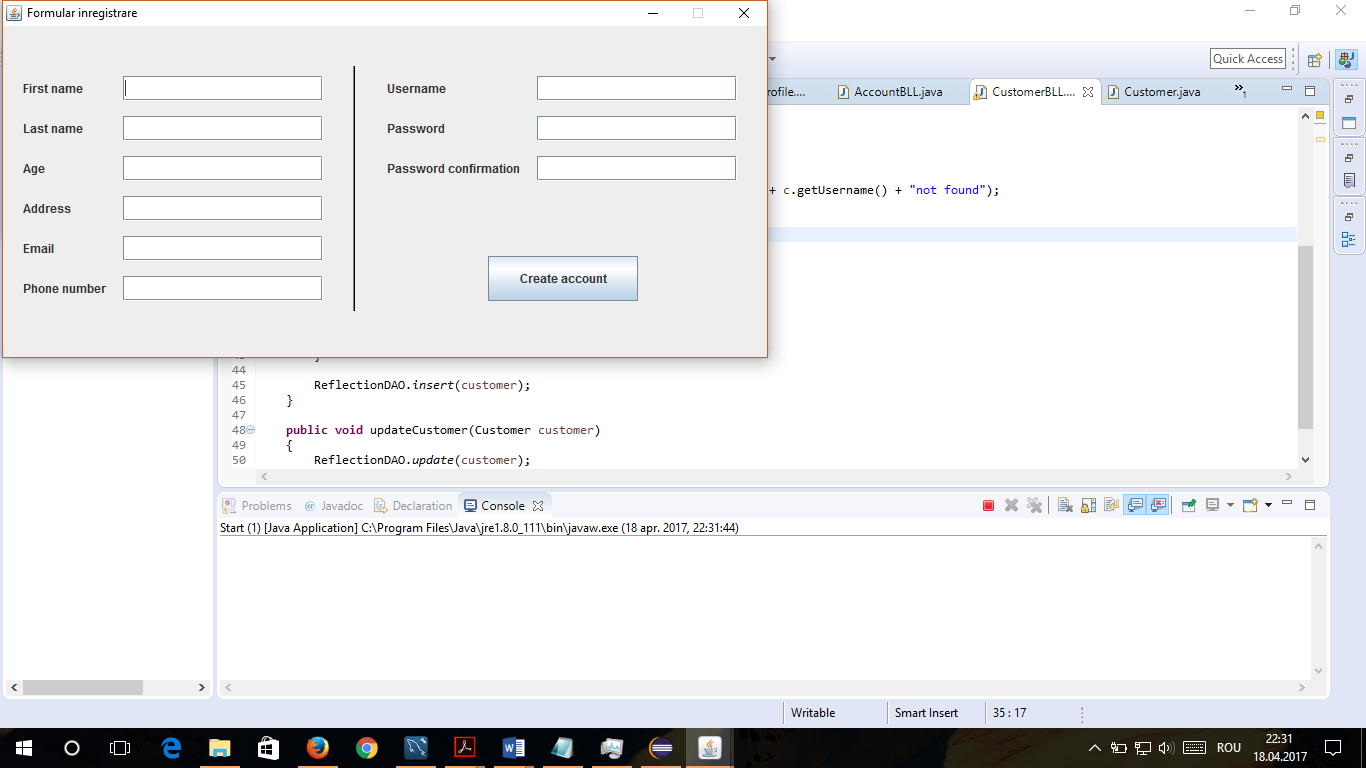


Fig. 5. Formular pentru crearea contului

Dacă ne autentificăm ca și administrator se va deschide o fereastră ce conține 4 butoane și un tabel (Fig. 6). Inițial sunt afișate în tabel produsele.

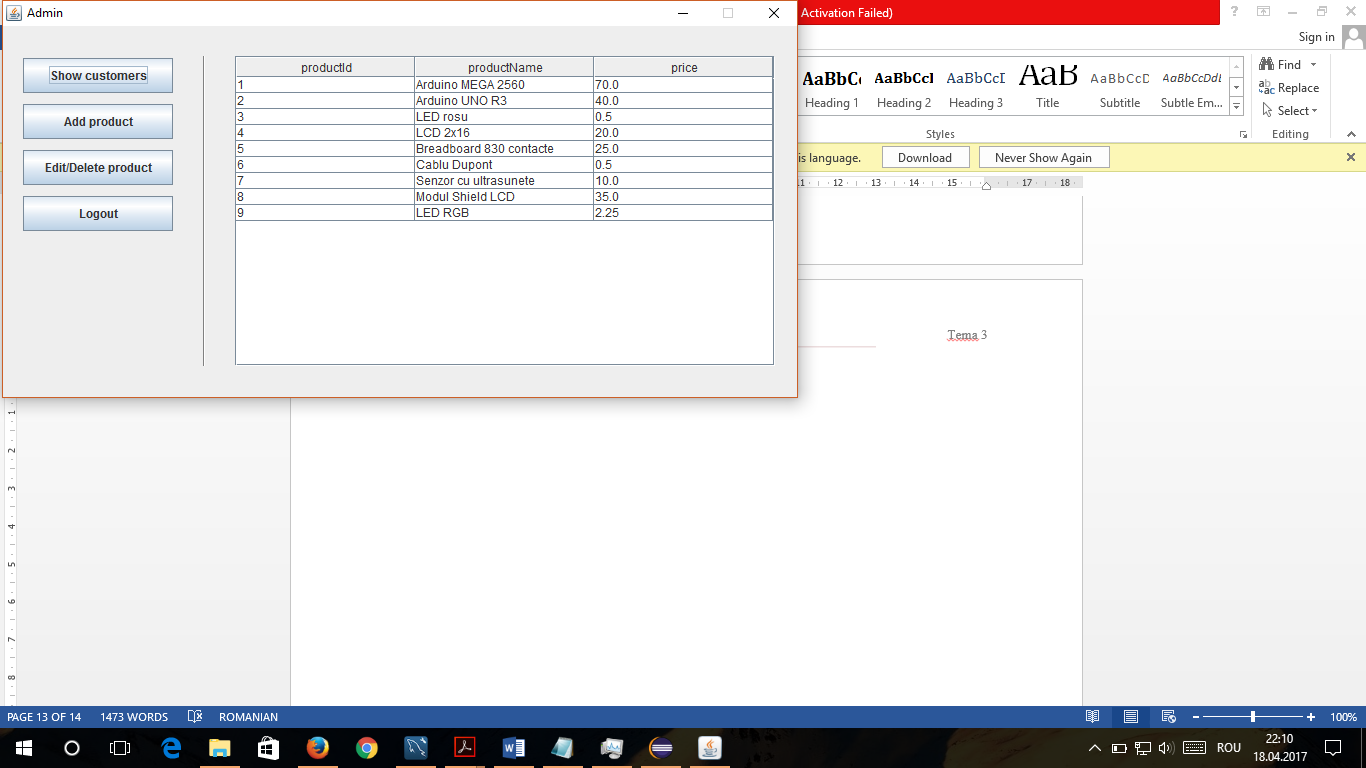


Fig. 6. Fereastra administratorului

Conținutul tabelului corespunde cu tabelul din MySQL (Fig. 7).

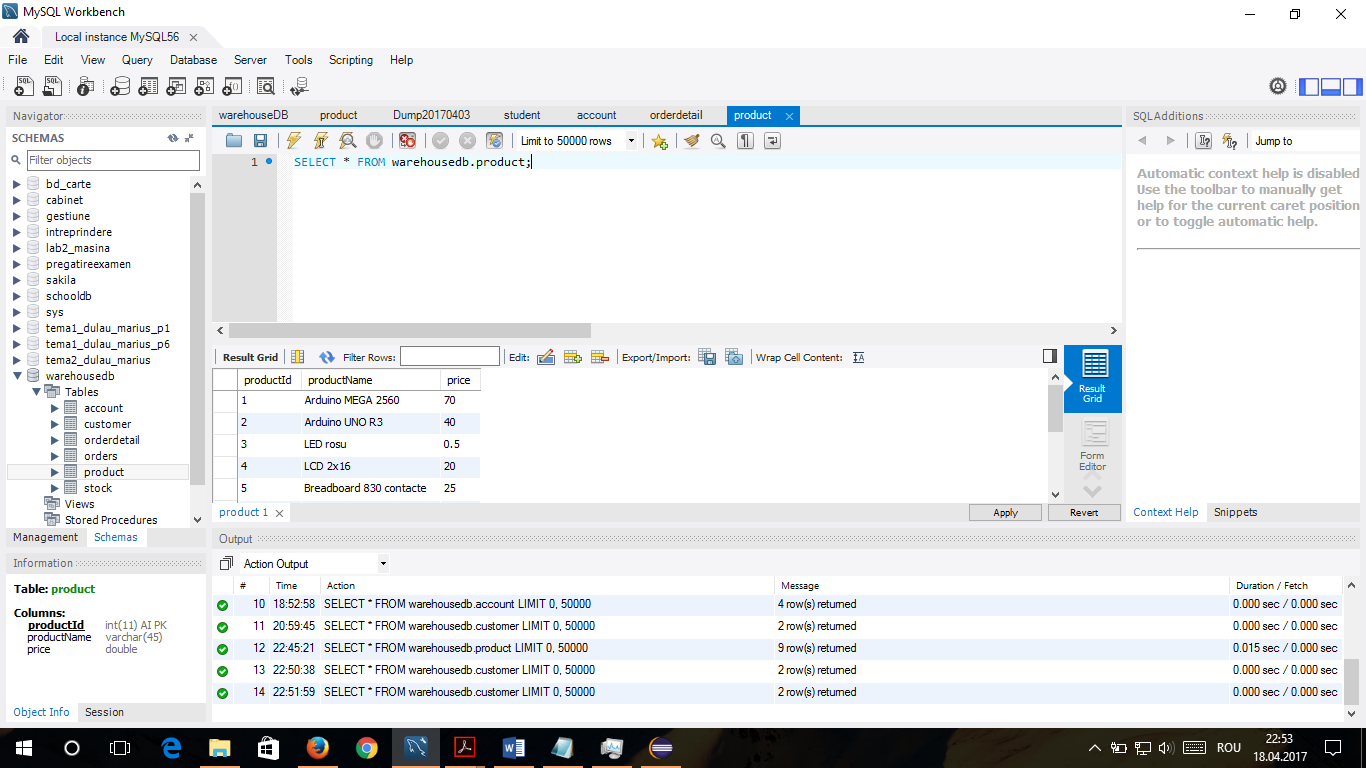


Fig. 7. Tabelul Product din MySQL

Dacă apăsăm pe butonul “Show customers” în tabel se vor afișa clienții, iar textul butonul se va schimba în “Show products” (Fig. 8).

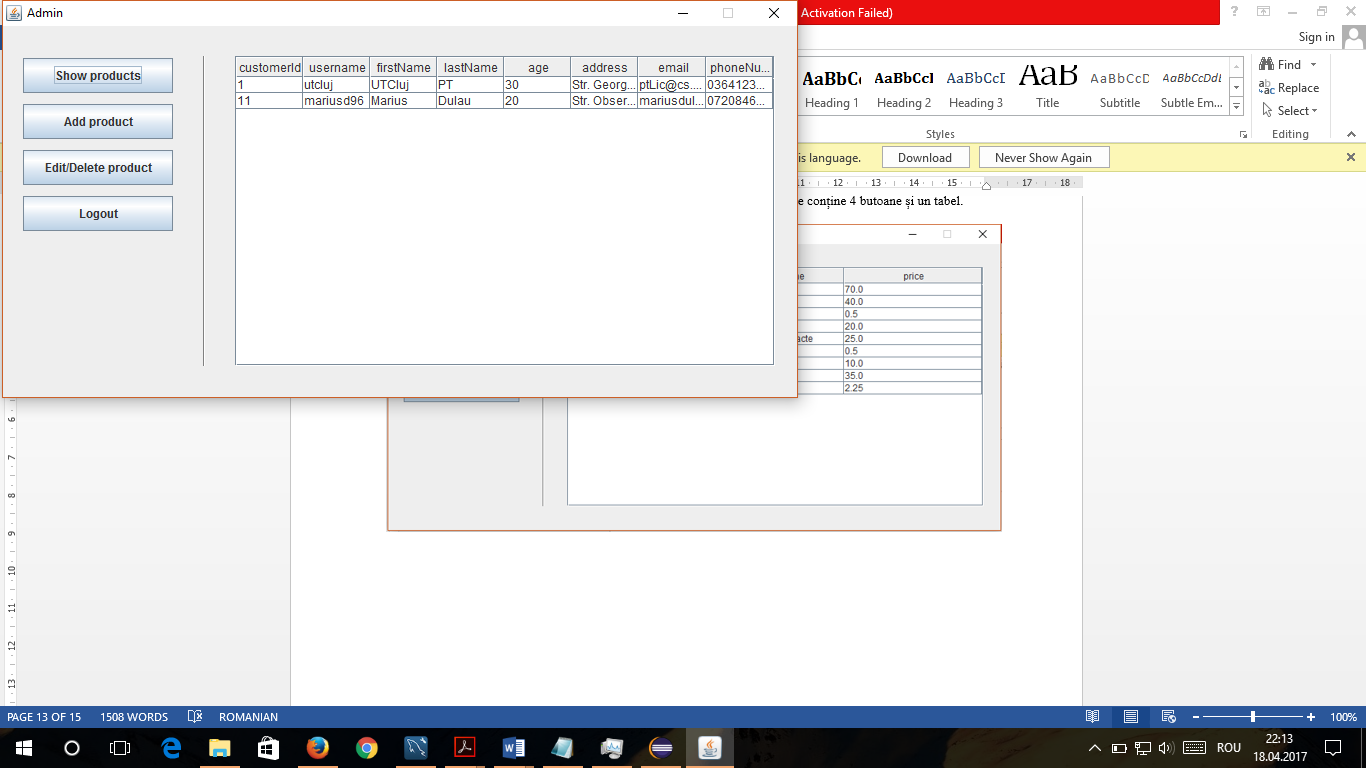


Fig. 8. Tabelul Clienți din MySQL ilustrat în aplicație

Ceea ce apare în tabelul din interfața grafică corespunde cu tabelul din MySQL(Fig. 9).

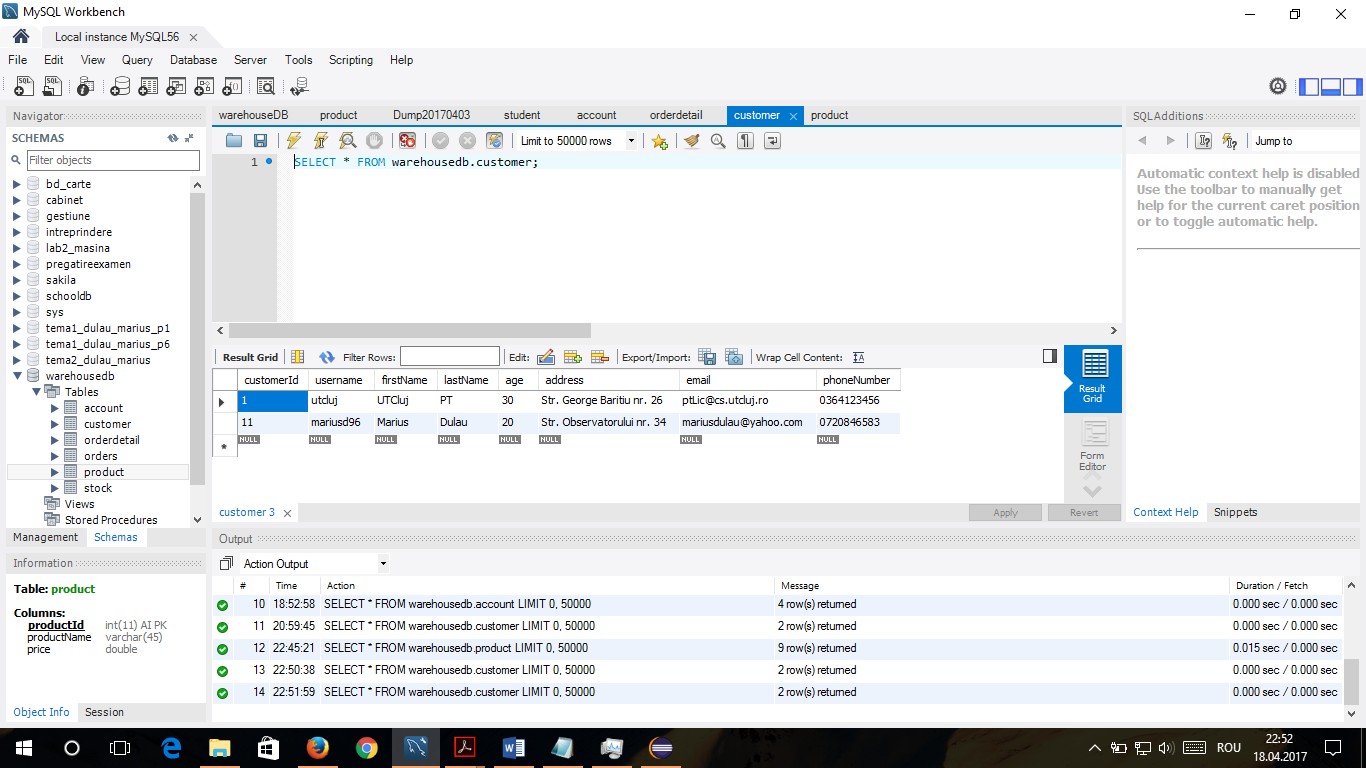


Fig. 9. Tabelul Customer din MySQL

Dacă se dă click pe butonul “Add product” se va deschide o fereastră pentru a adăuga produse noi (Fig. 10).

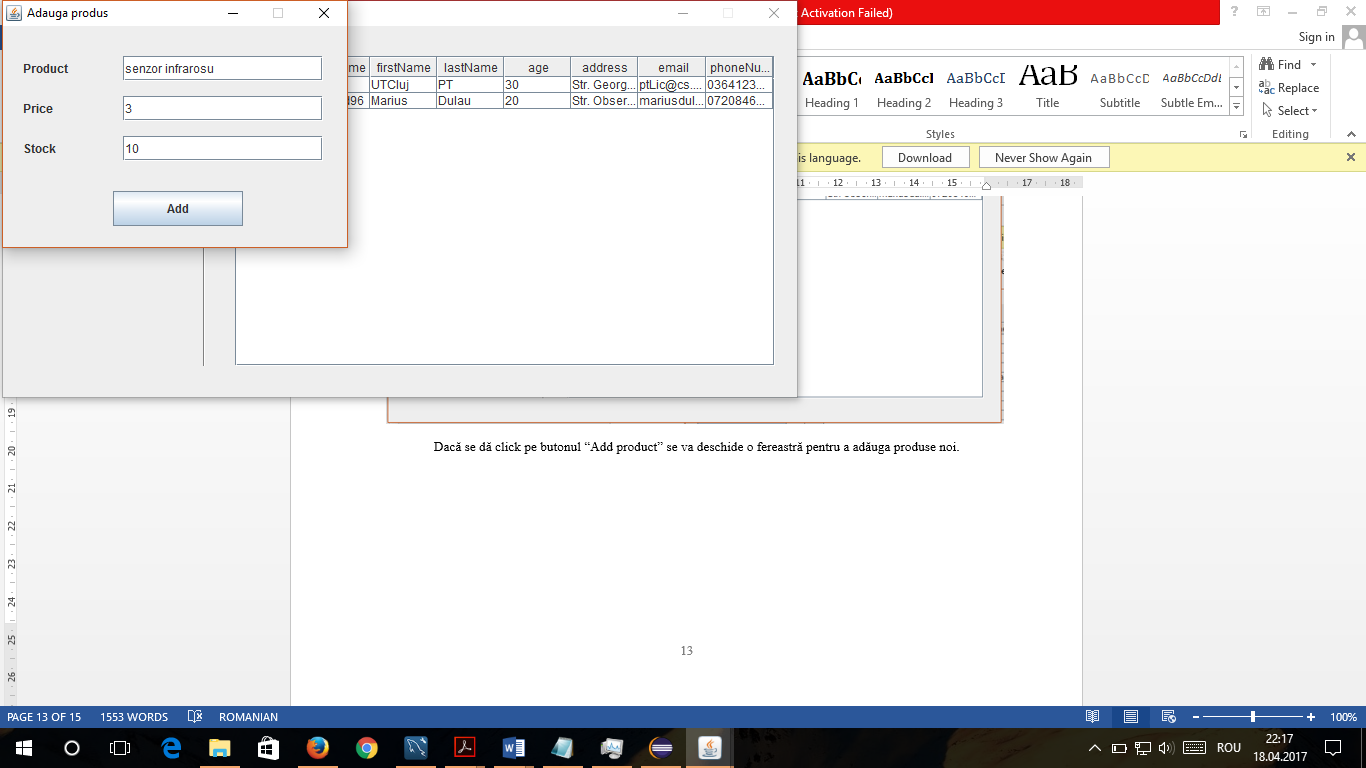


Fig. 10. Fereastra de adăugare a produselor noi

Produsul nou adăugat va apărea în tabelul principal și în tabelul din MySQL (Fig. 11 și Fig. 12).

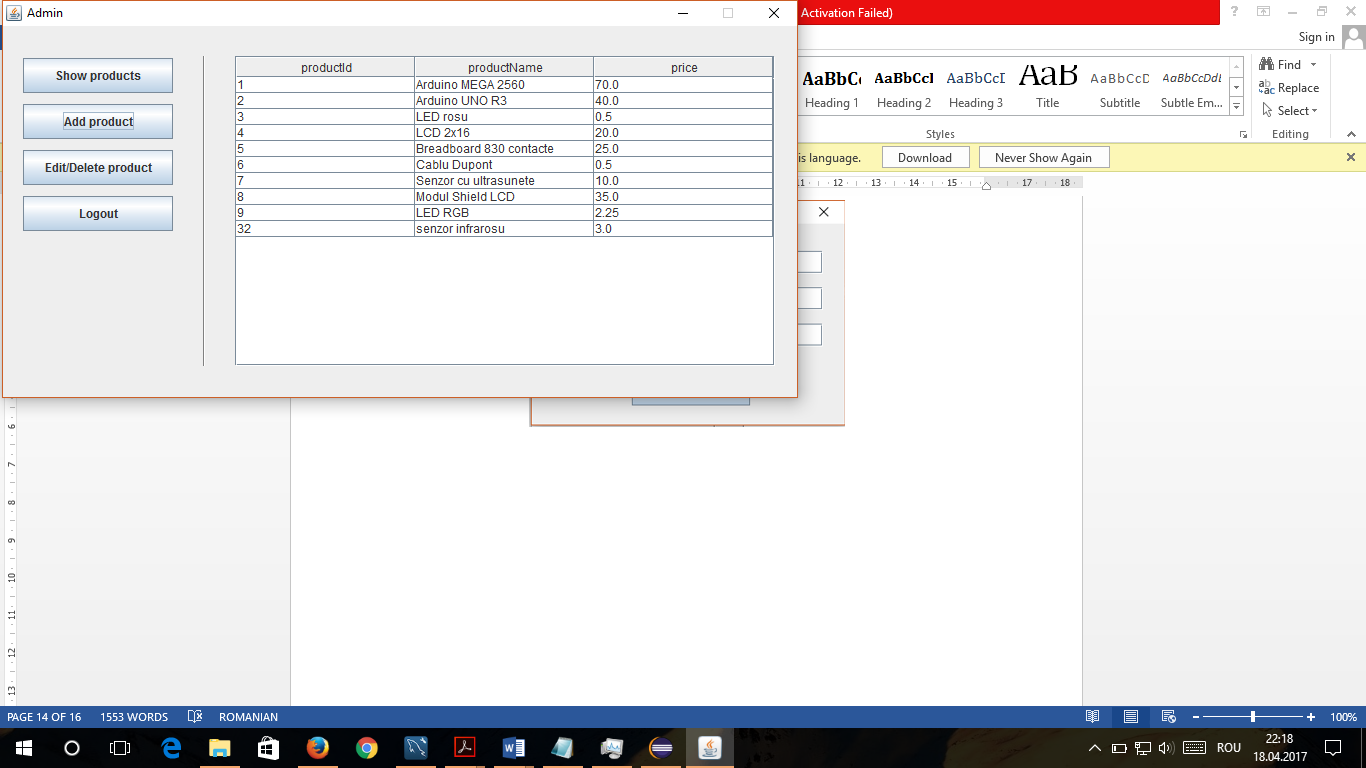


Fig. 11. Tabelul din fereastra administratorului actualizat

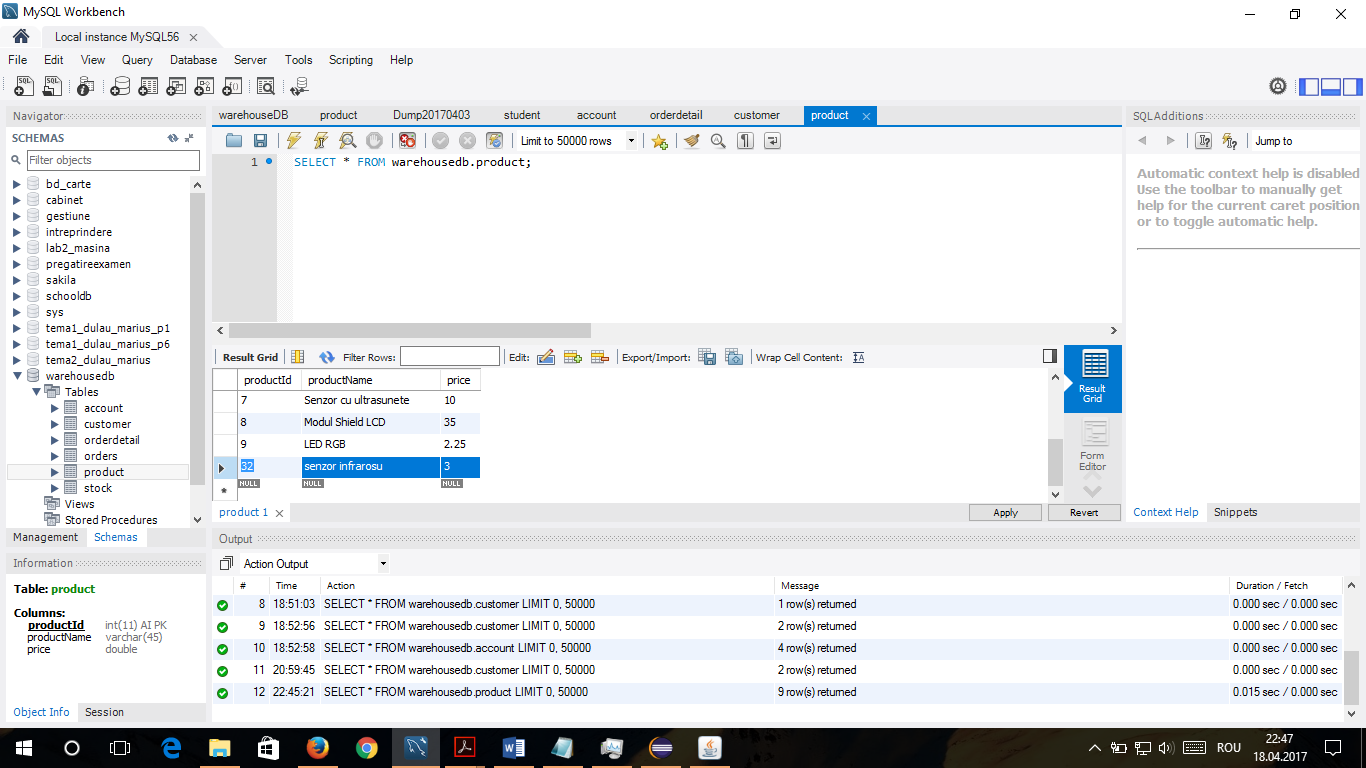


Fig. 12. Tabelul Product actualizat în MySQL

Efectuarea unui click pe butonul “Edit/Delete product” va deschide o fereastră în care se alege un produs și se pot efectua operații de modificare sau chiar ștergerea produsului (Fig. 13).

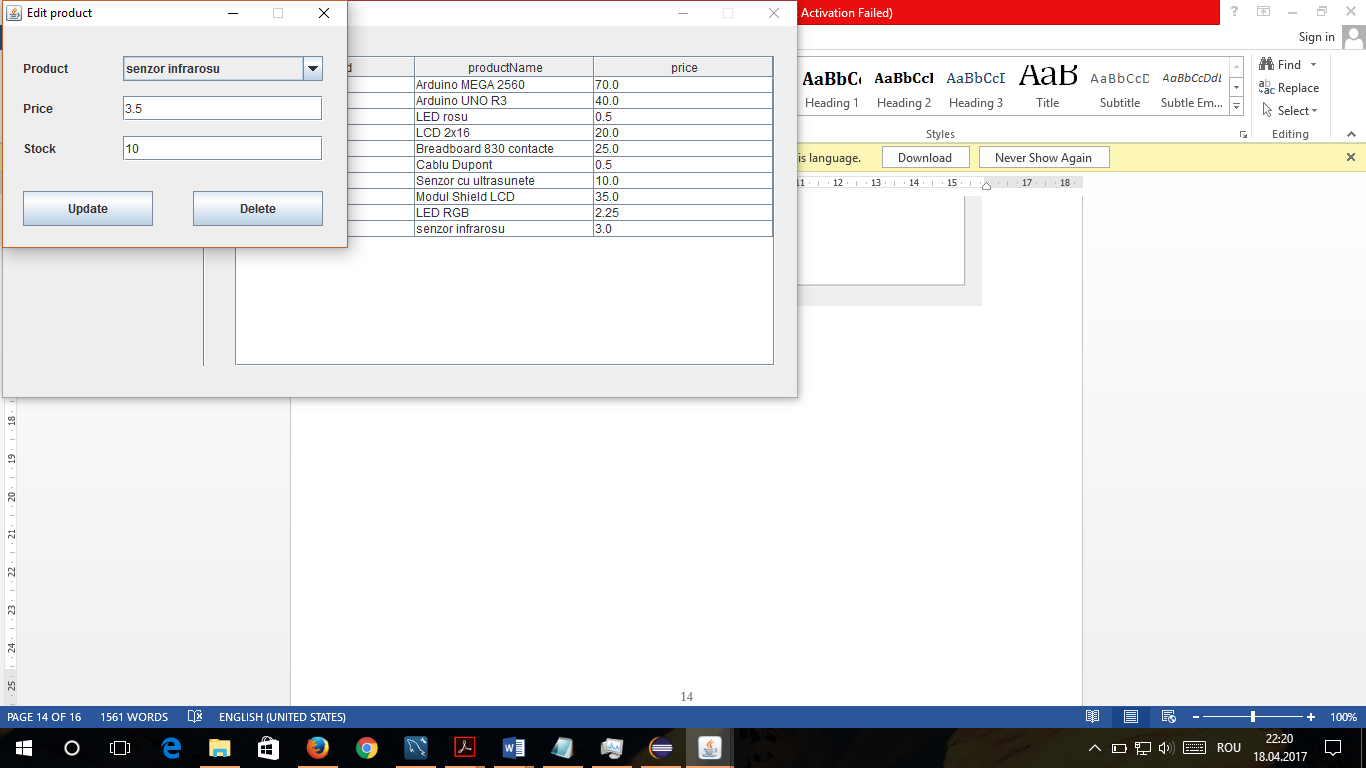


Fig. 12. Tabelul Product actualizat în MySQL

Dacă dăm click pe butonul “Update” prețul produsului sensor infraroșu se va modica (Fig. 14).

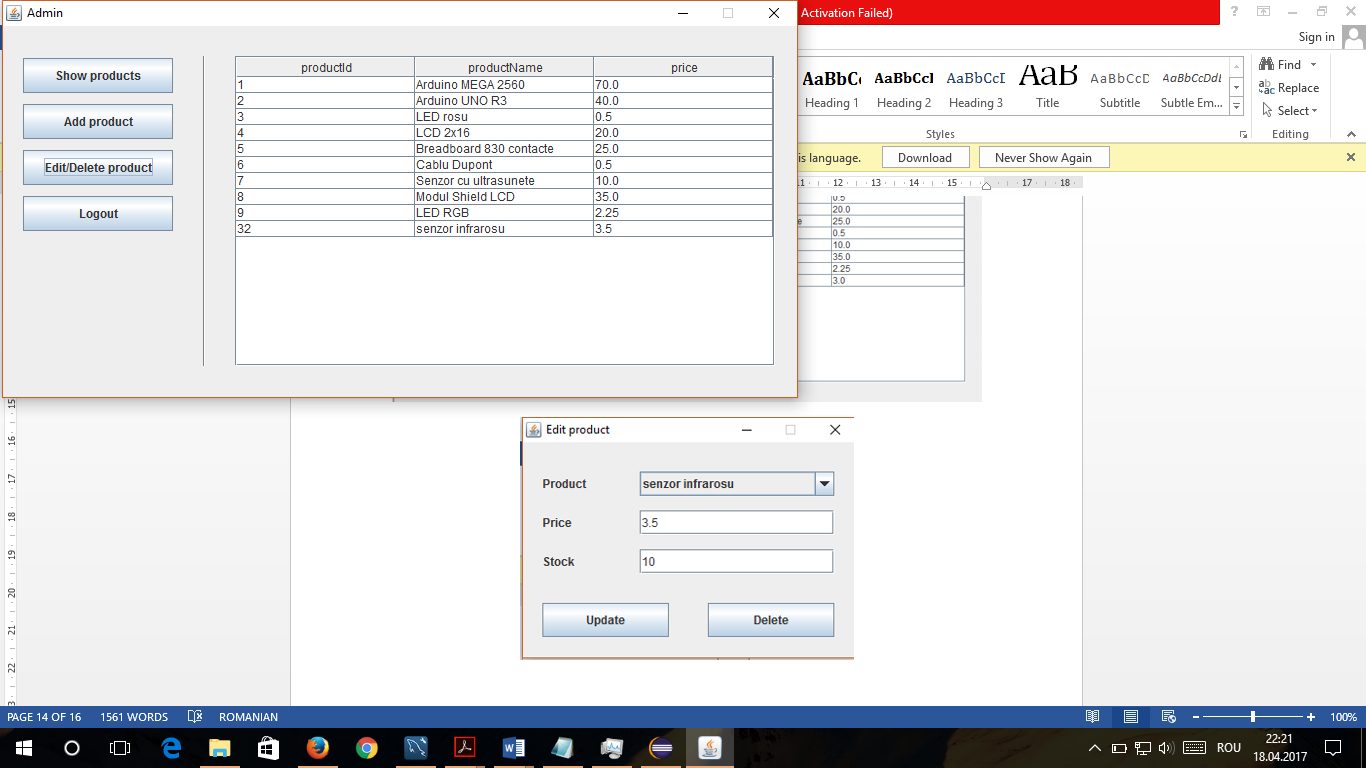


Fig. 14. Tabelul cu produse actualizat

Dacă se apasă pe butonul “Delete” acel produs este șters din baza de date, iar tabelul din interfața grafică se actualizează. După fiecare operație efectuată se va afișa un mesaj de confirmare (Fig. 15 și Fig. 16).

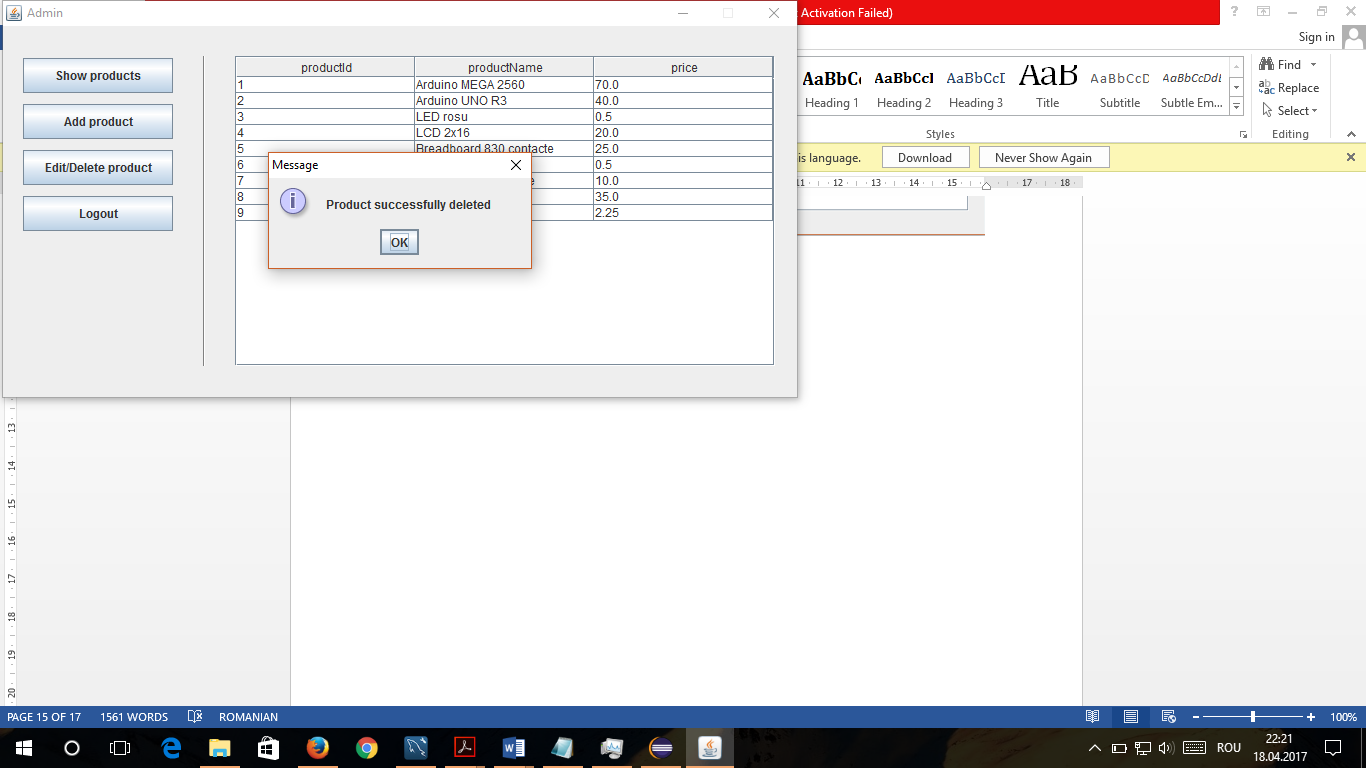


Fig. 15. Tabelul reactualizat în aplicație în urma ștergerii produsulu senzor infraroșu

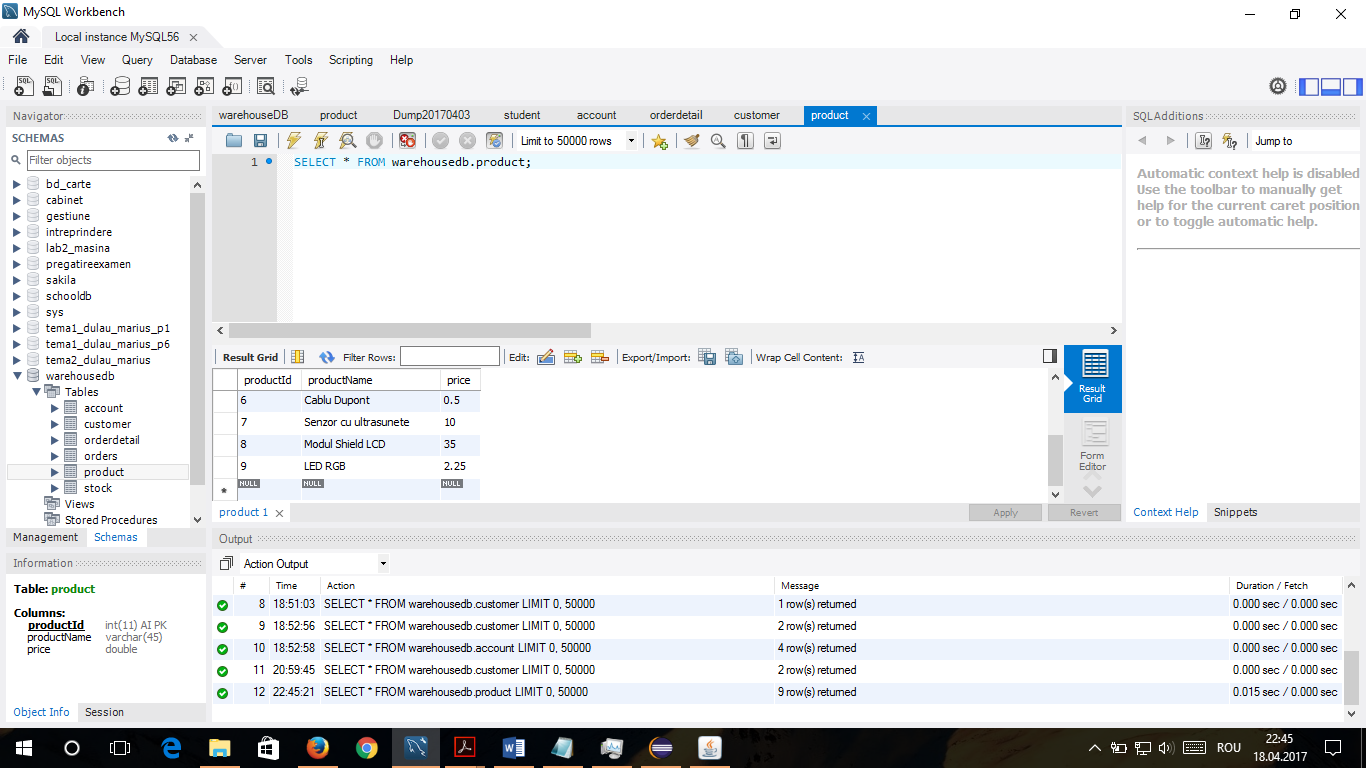


Fig. 16. Tabelul reactualizat în aplicație în urma ștergerii produsulu senzor infraroșu

Dacă ne autentificăm ca și un simplu client se va deschide o fereastră ce conține 4 butoane și un tabel(Fig. 17). În tabel afișate doar produsele. Dând click pe butonul “Edit profile data” se deschide o fereastră cu toate datele utilizatorului care se pot modifca (Fig. 18). Apăsând pe butonul “Delete account” se va șterge din baza de date contul utilizatorului (Fig.19).

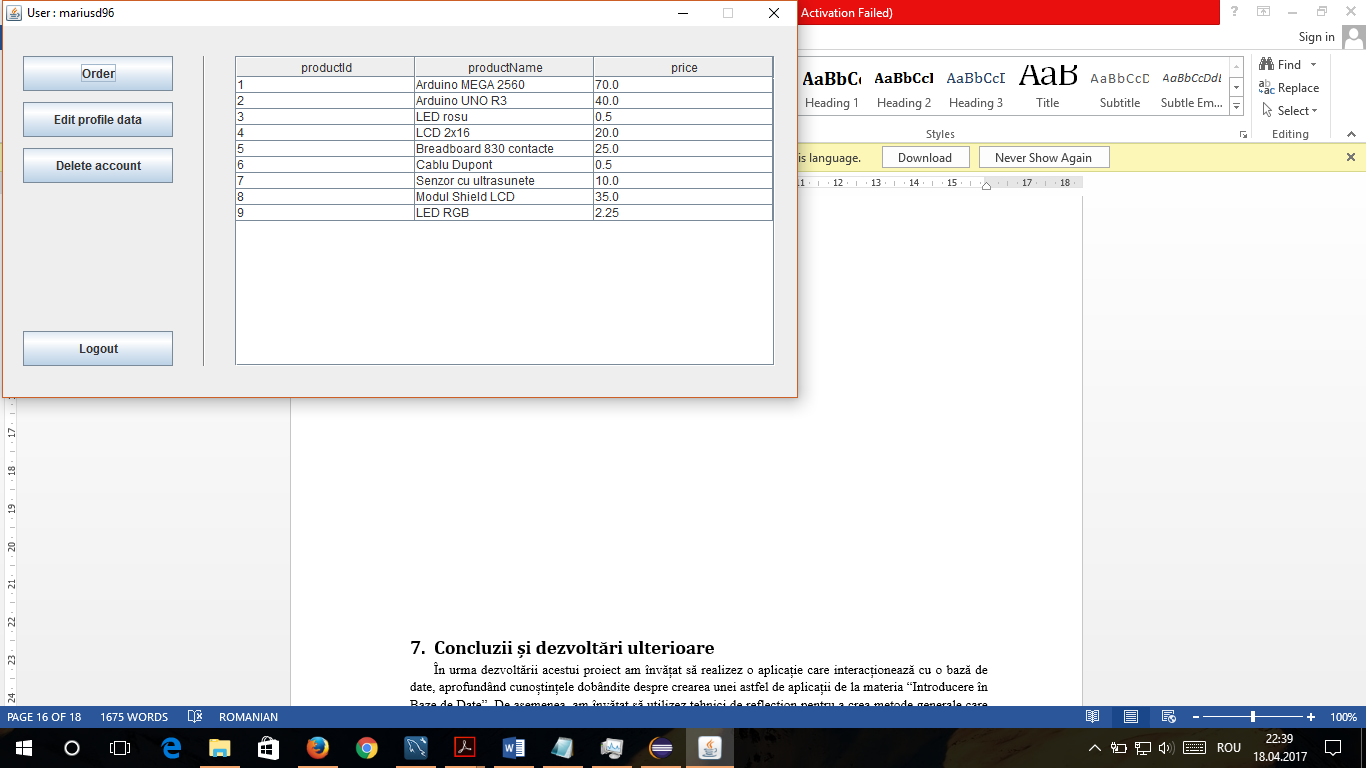


Fig. 17. Fereastra utilizatorului

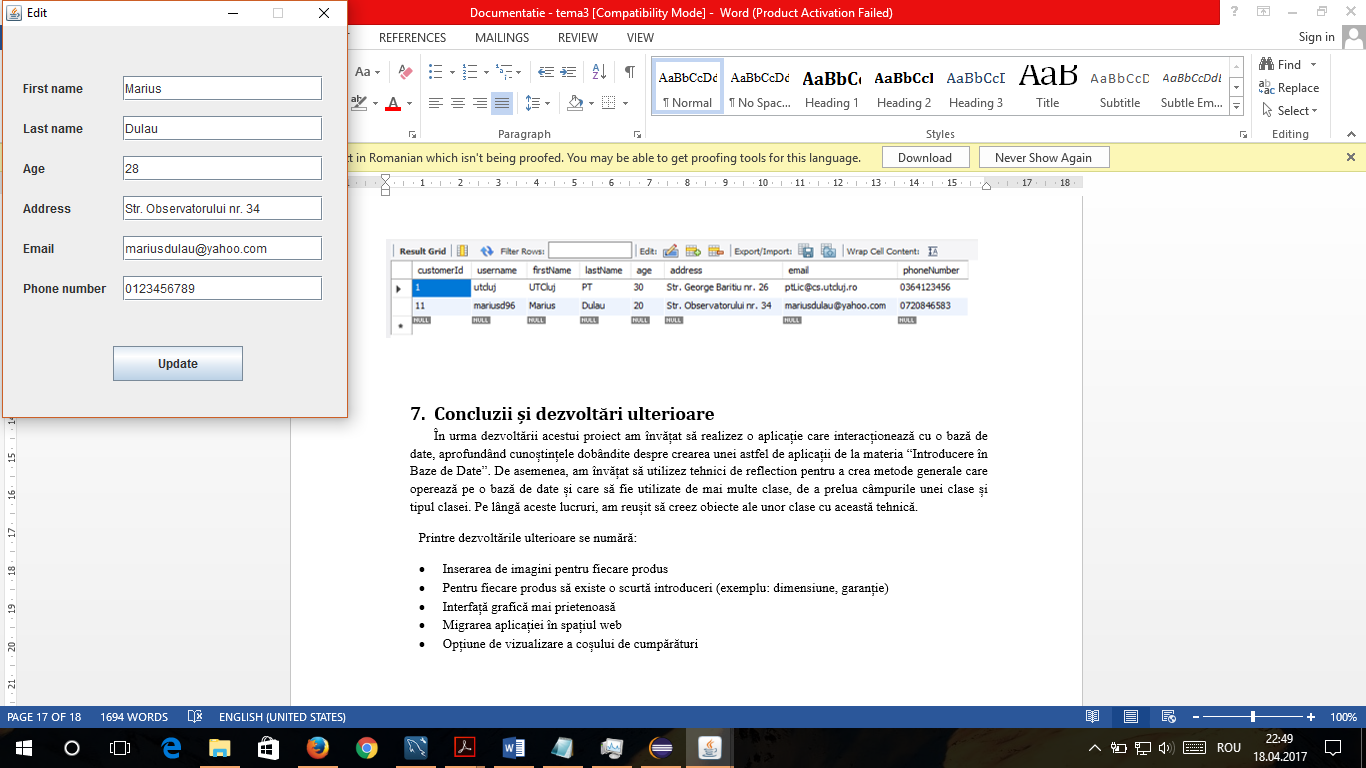


Fig. 18. Fereastra de editare a datelor utilizatorului

După efectuarea unui click pe butonul “Update” se vor modifica informațiile în baza de date și în aplicație se va afișa un mesaj de confirmare(Fig. 19).

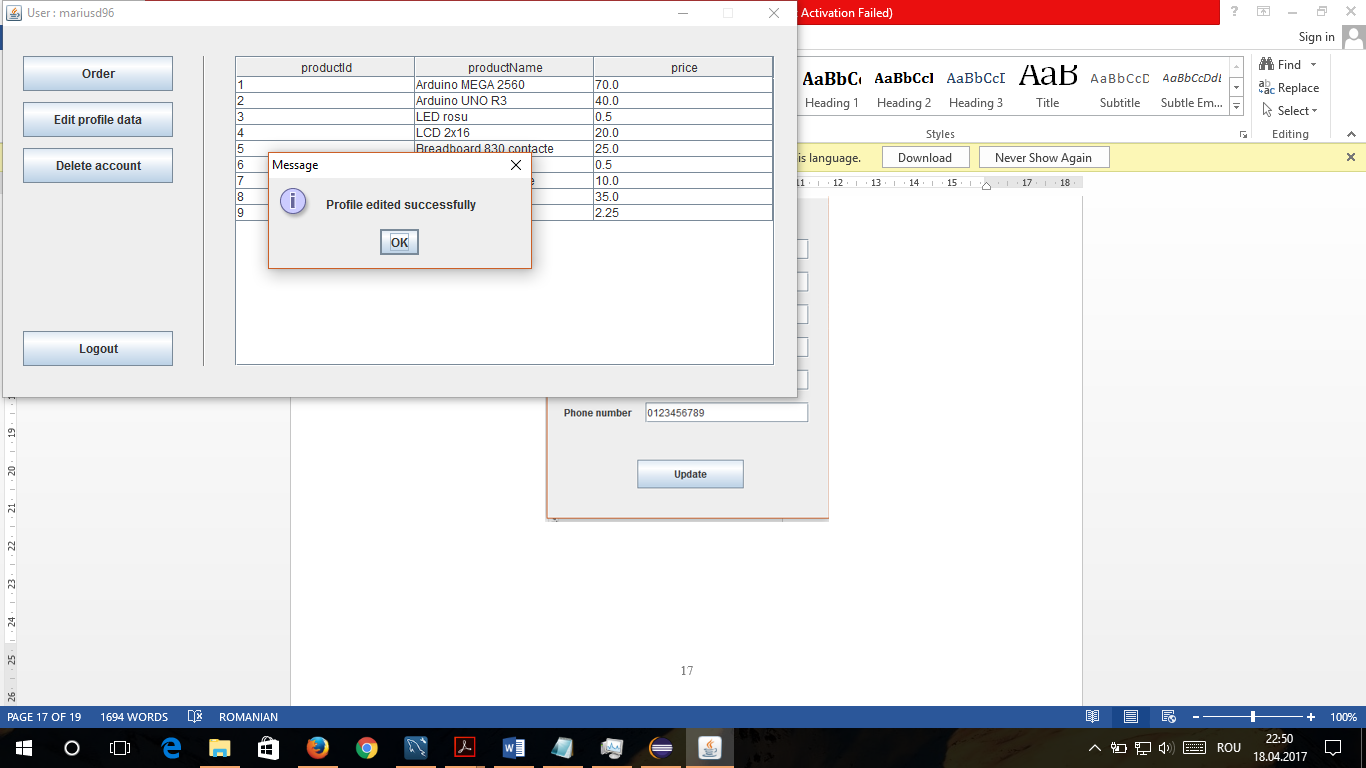


Fig. 19. Mesaj de confirmare a modificării datelor personale

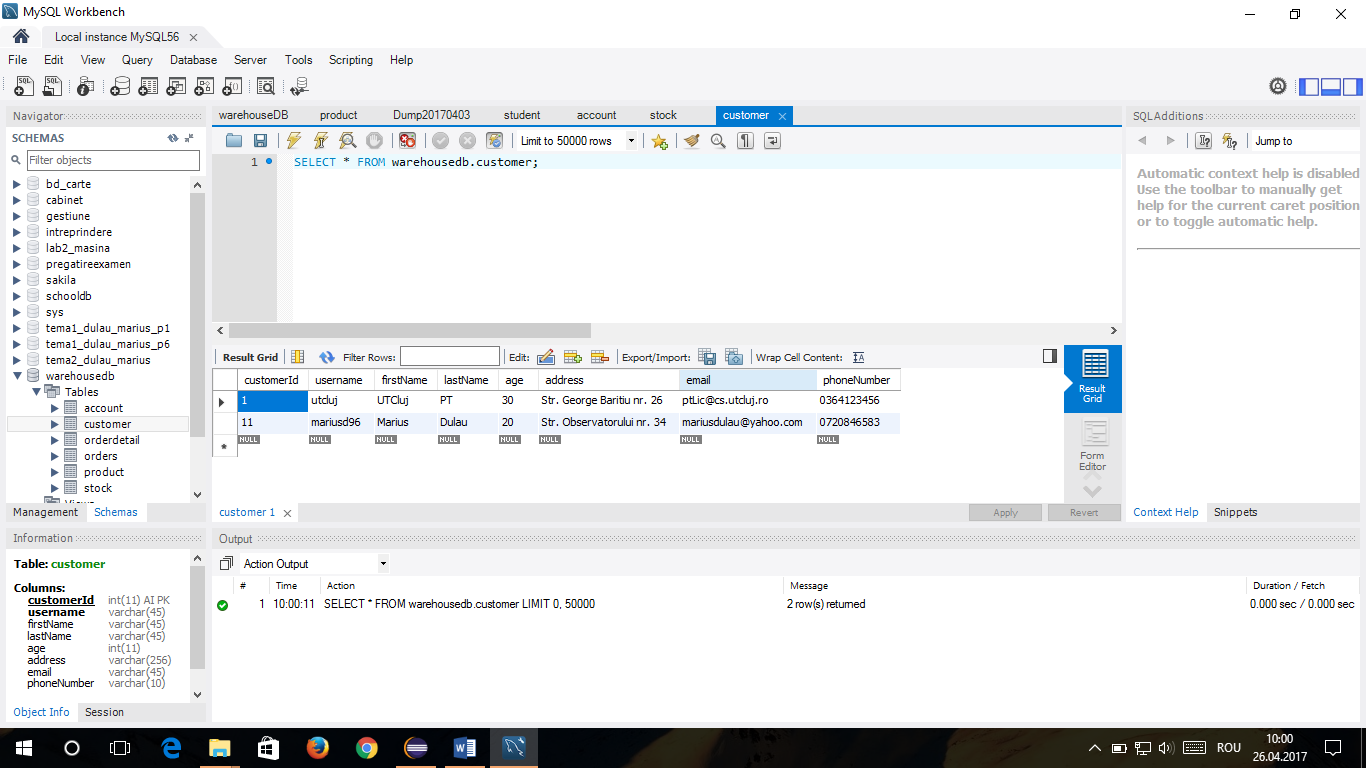


Fig. 20.a. Tabelul în baza de date înainte de modificare

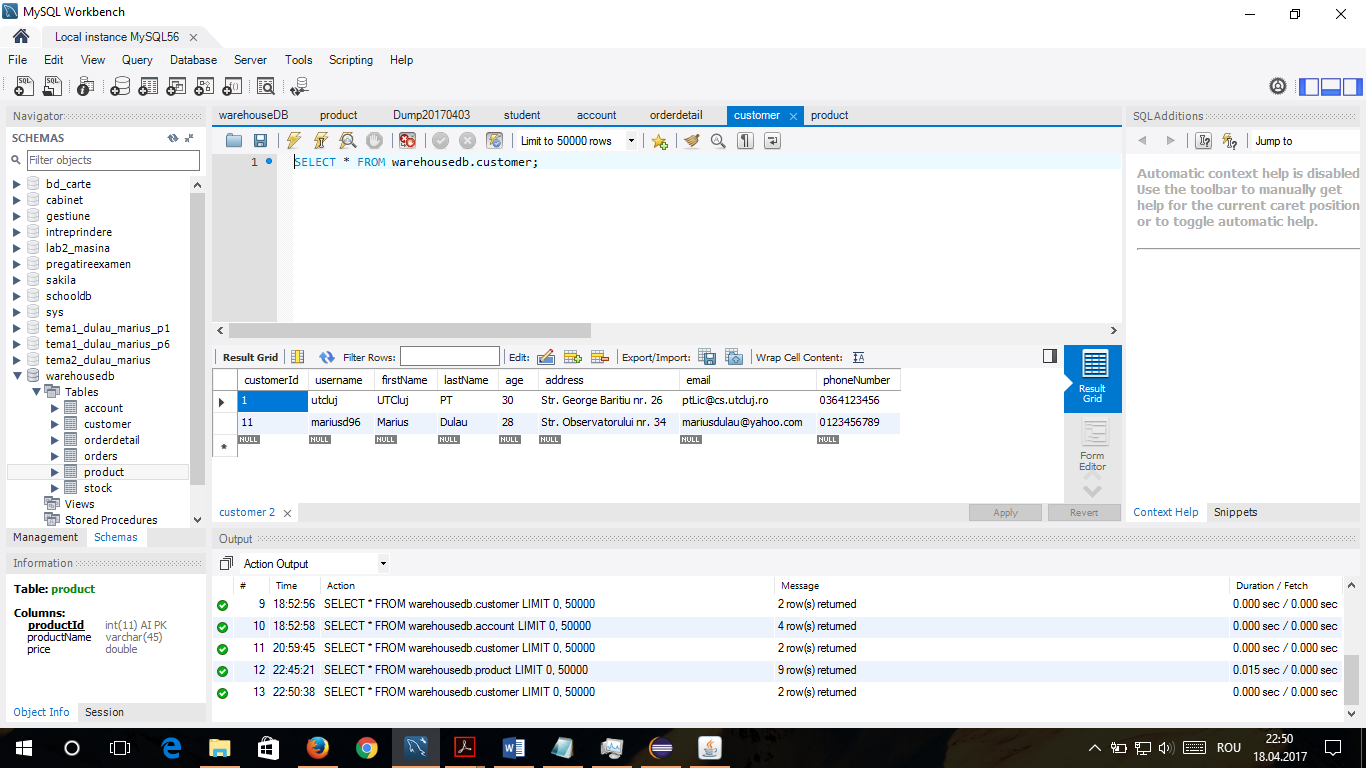


Fig. 20.b. Tabelul în baza de date după de modificare

Actiunea butonului “Order” va deschide o fereastră ce conține o zonă cu detaliile unui produs și cantitatea dorită și o altă zonă cu coșul de cumpărături propriu-zis (Fig. 21). Butonul “Add” va introduce datele clientului în factură și într-un ArrayList. Se introduc date în ArrayList pentru că s-ar putea anula comanda prin închiderea ferestrei și actualizarea în baza de date ar fi eronată.

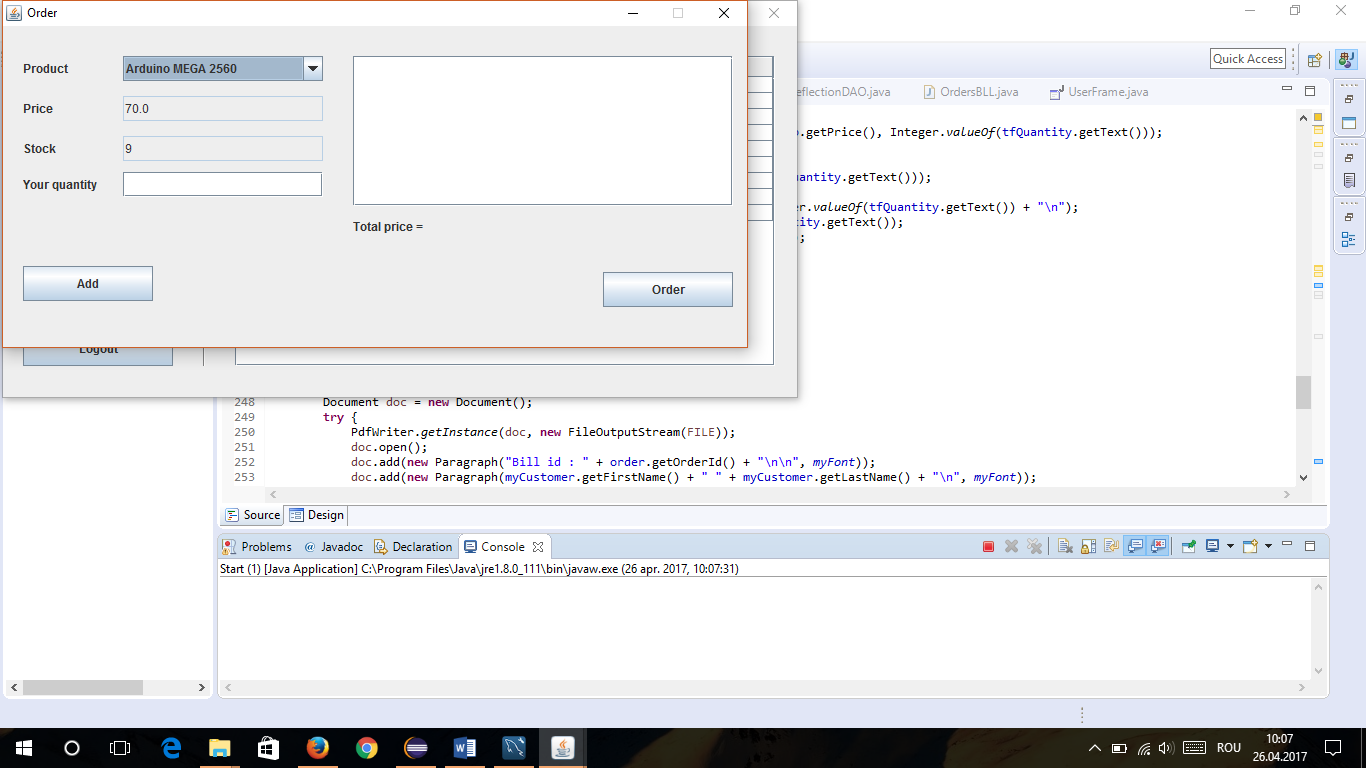


Fig. 21. Fereastra de efectuarea unei comenzi

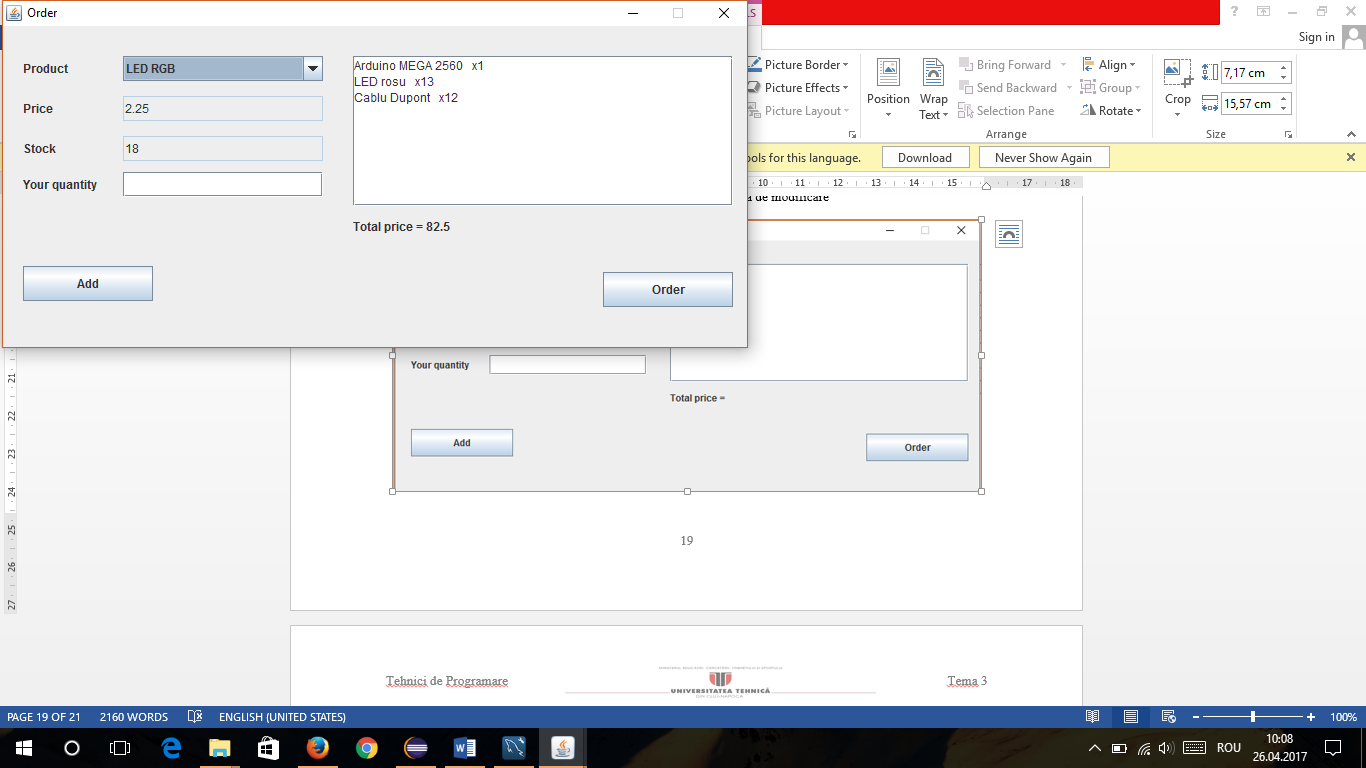


Fig. 22. Coșul de cumpărături aflat în partea stângă

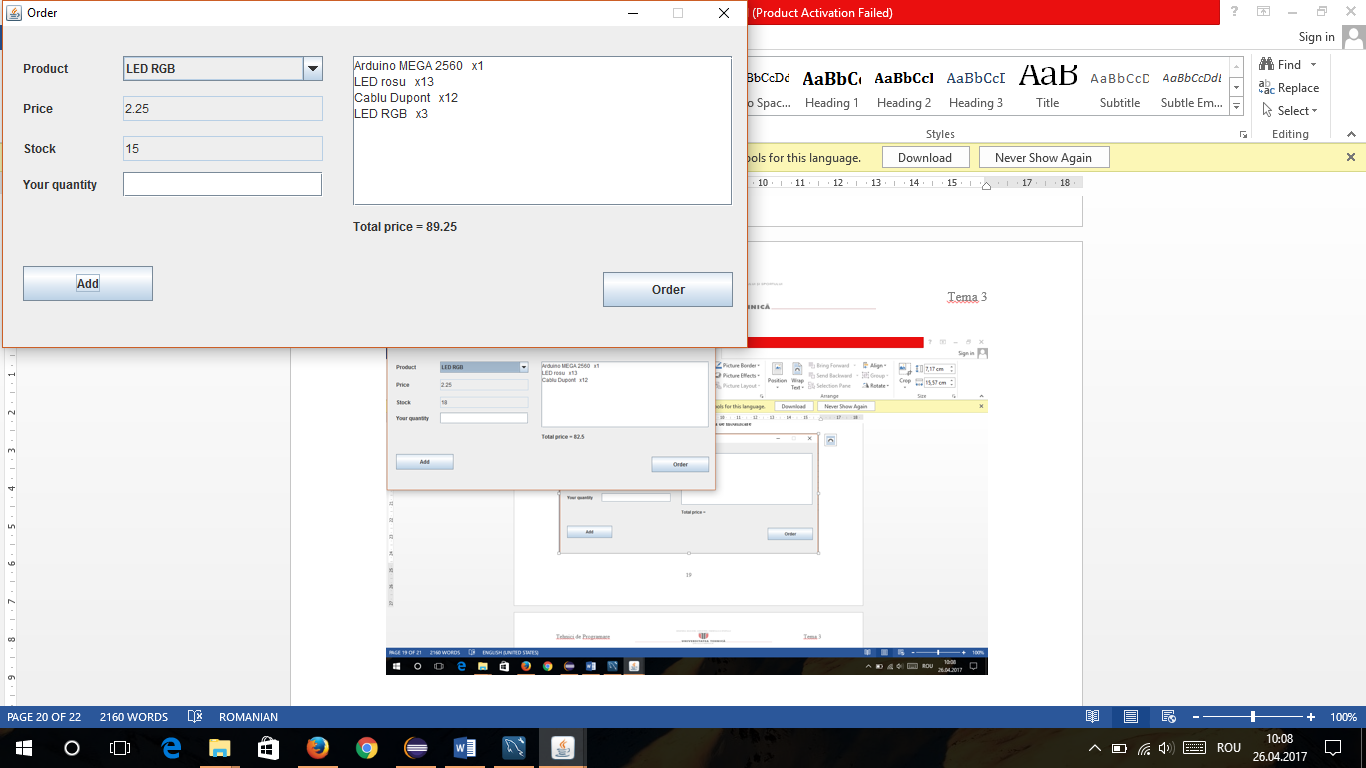


Fig. 23. Modificarea stocului produsului LED RGB după ce a fost adăugat în coșul de cumpărături

Dacă utilizatorul introduce o cantitate mai mare decăt limita stocului disponibil se generează un mesaj de avertizare (Fig. 24).

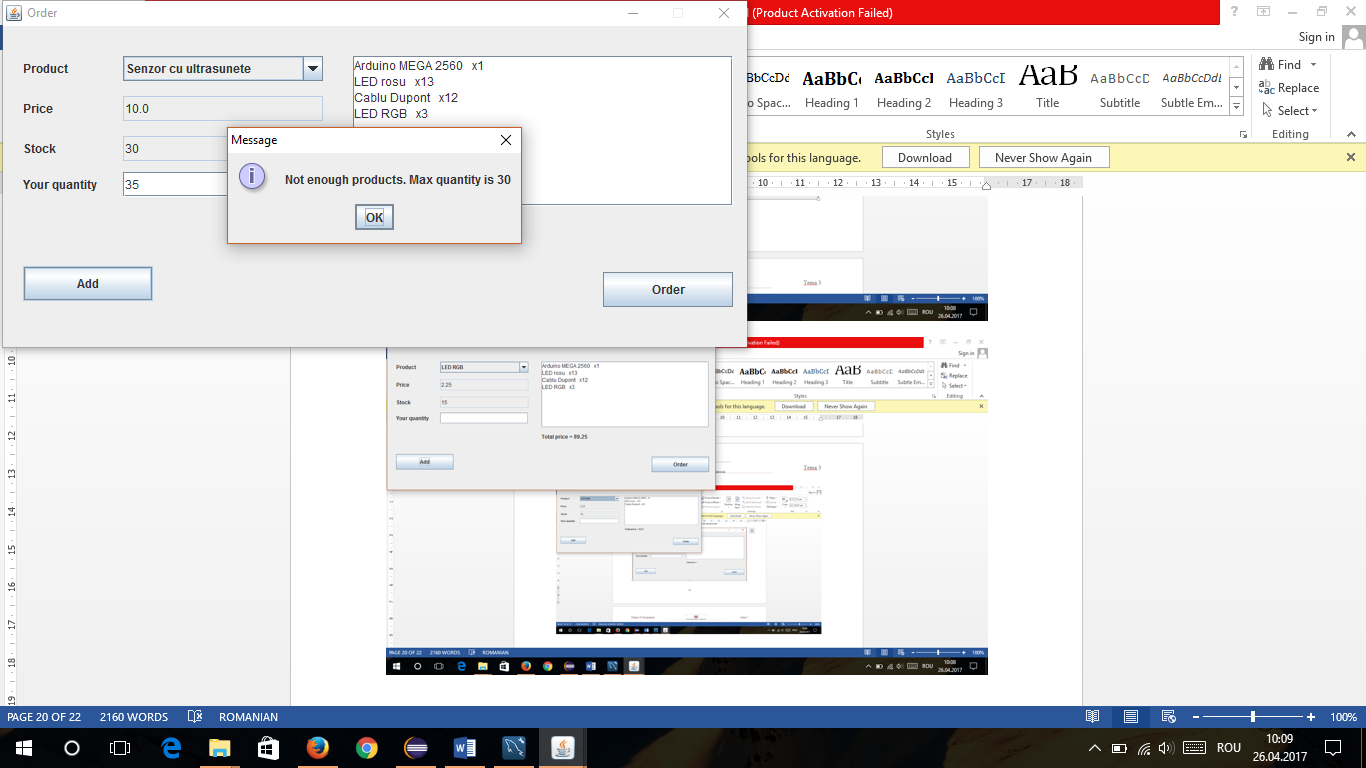


Fig. 24. Mesaj de eroare în cazul unui stoc mai mic decât cantitatea cerută

În cazul în care cantitatea este mai mică decât stocul, acel produs va fi adăugat în coșul de cumpărături (Fig. 25) și se va modifica stocul la prima vedere.

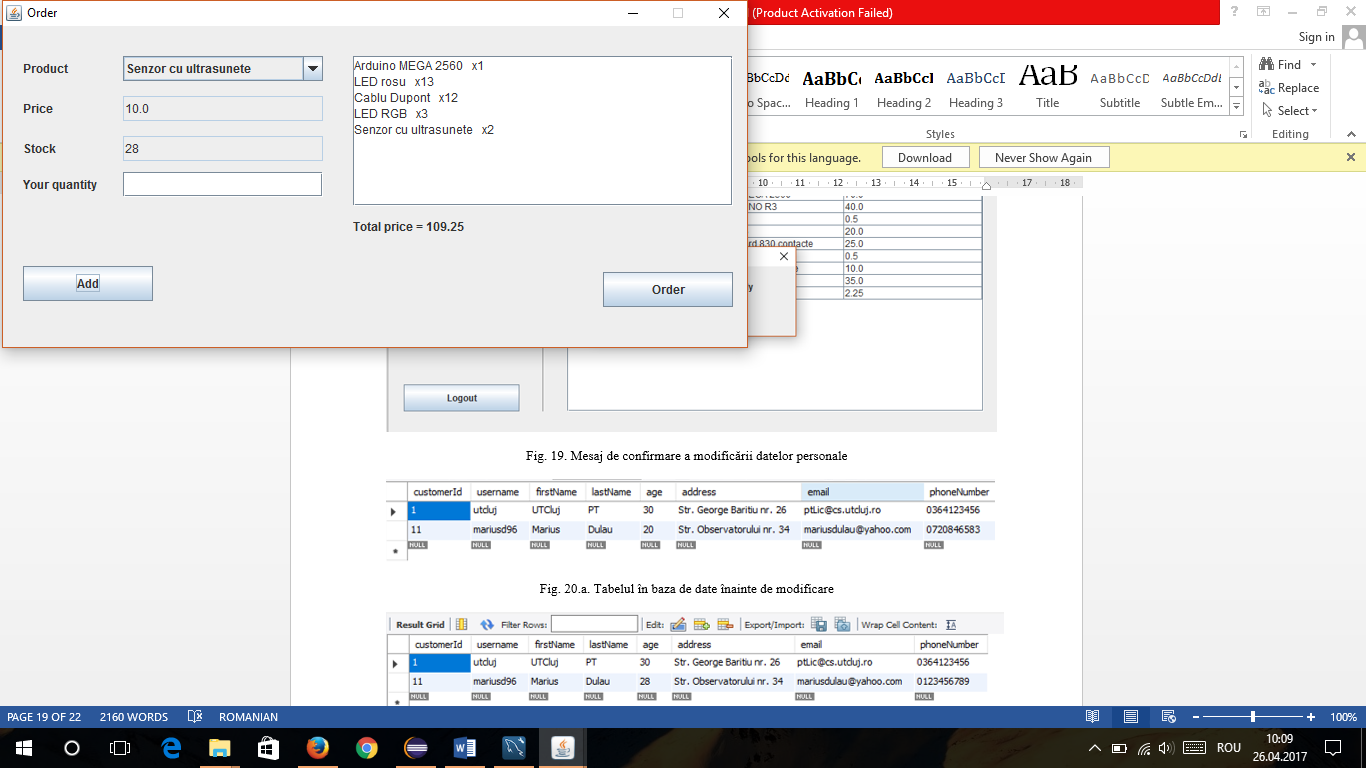


Fig. 25. Modificarea stocului produsului senzor cu ultrasunete

După ce se apasă pe butonul “Order” se actualizează baza de date și se creează un fișier .pdf cu factura(Fig. 26). De asemenea, se trimite un mail utilizatorului cu factura curentă(Fig. 27).

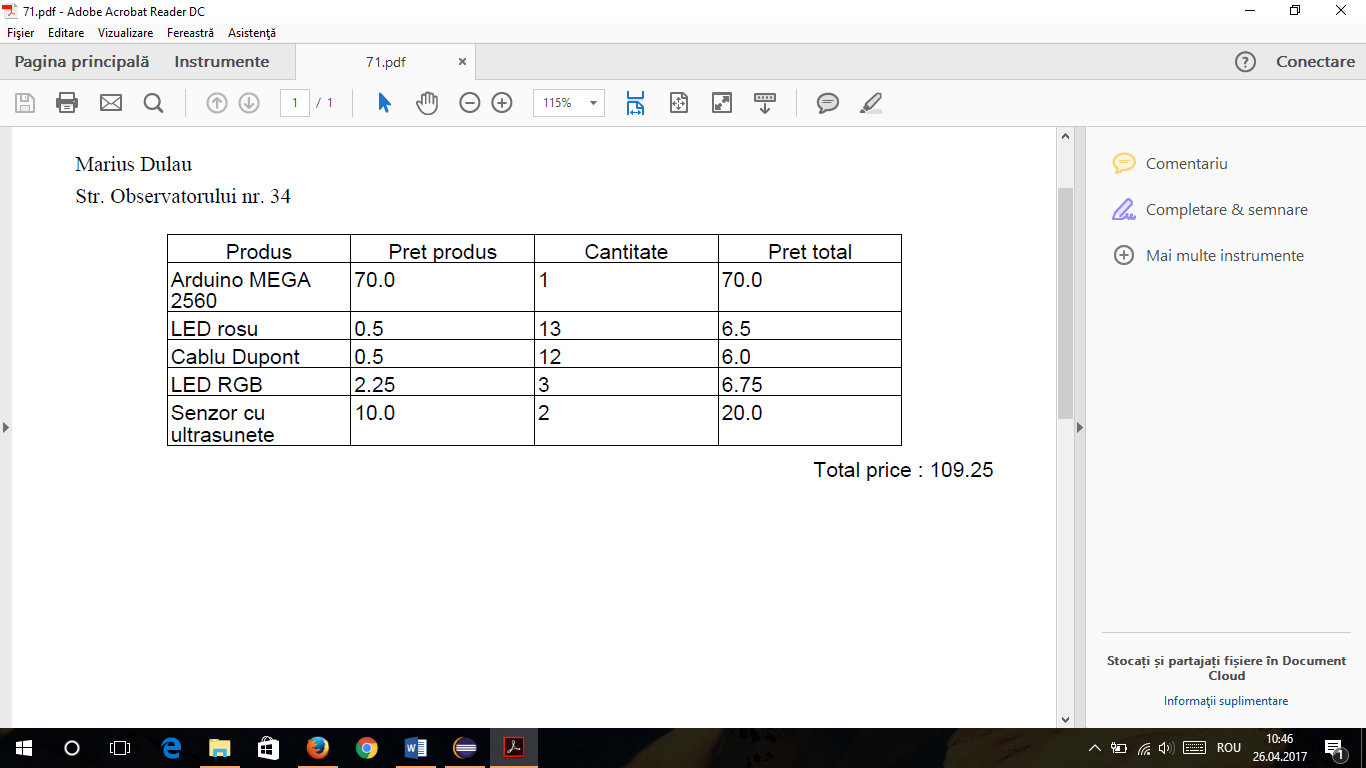


Fig. 26. Factura în format .pdf

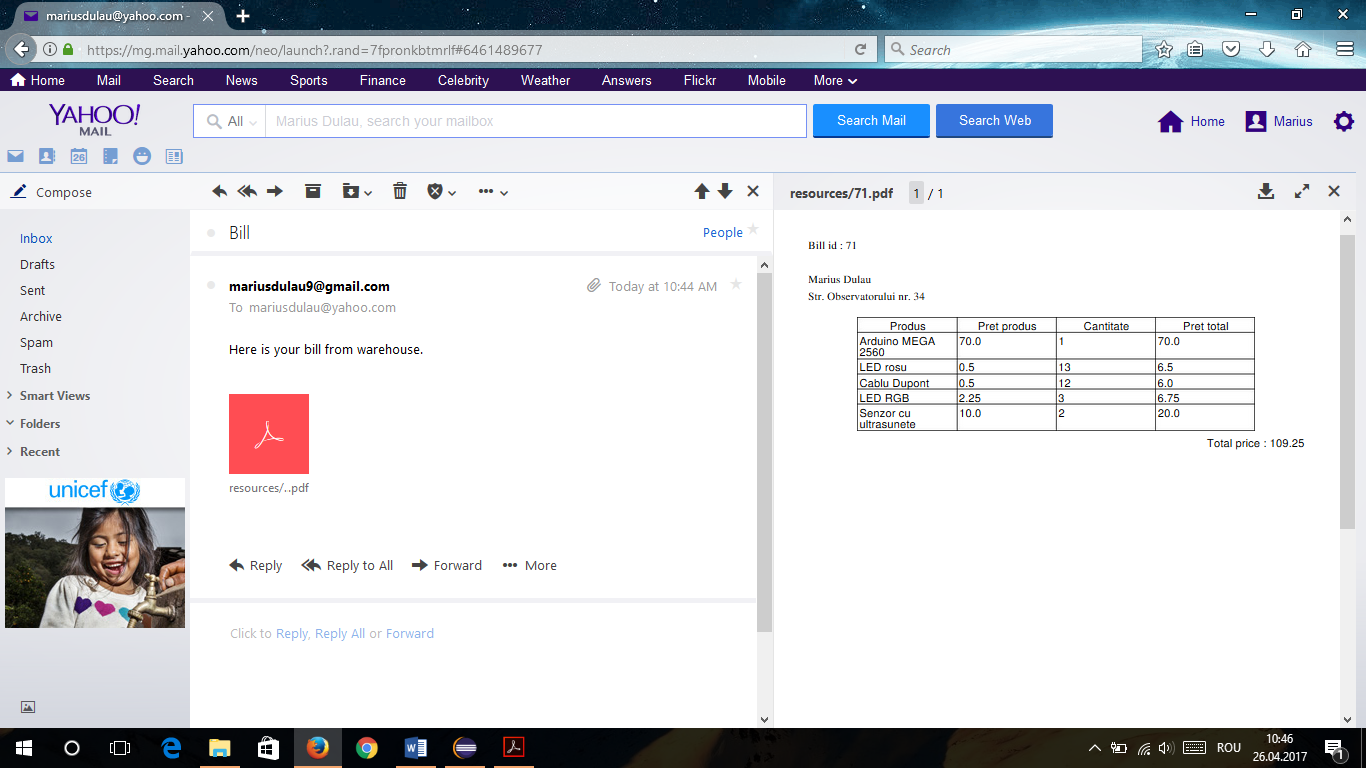


Fig. 27. Mail cu factura în format .pdf

# Concluzii și dezvoltări ulterioare

În urma dezvoltării acestui proiect am învățat să realizez o aplicație care interacționează cu o bază de date, aprofundând cunoștințele dobândite despre crearea unei astfel de aplicații de la materia “Introducere în Baze de Date”. De asemenea, am învățat să utilizez tehnici de reflection pentru a crea metode generale care operează pe o bază de date și care să fie utilizate de mai multe clase, de a prelua câmpurile unei clase și tipul clasei. Pe lângă aceste lucruri, am reușit să creez obiecte ale unor clase cu această tehnică.

Printre dezvoltările ulterioare se numără:

* Inserarea de imagini pentru fiecare produs
* Pentru fiecare produs să existe o scurtă introduceri (exemplu: dimensiune, garanție)
* Interfață grafică mai prietenoasă
* Migrarea aplicației în spațiul web
* Opțiune de vizualizare a coșului de cumpărături
* Clientul să vizualizeze comenzile anterioare

# Bibliografie

<http://stackoverflow.com/questions/10531513/how-to-identify-object-types-in-java>

<http://stackoverflow.com/questions/6271417/java-get-the-current-class-name>

<http://stackoverflow.com/questions/10470263/create-new-object-using-reflection>

<http://stackoverflow.com/questions/14162225/getting-text-from-password-field>

<http://stackoverflow.com/questions/5868369/how-to-read-a-large-text-file-line-by-line-using-java>

<http://alvinalexander.com/java/java-mysql-select-query-example>

<http://www.java2s.com/Tutorial/Java/0240__Swing/CreatingaJTable.htm>

<http://stackoverflow.com/questions/21585851/jtable-cant-see-column-names>

<http://stackoverflow.com/questions/28250698/how-to-refresh-jtable>

<https://www.mkyong.com/java/java-how-to-get-current-date-time-date-and-calender/>

<https://www.youtube.com/watch?v=2852BXYimj4>

<http://www.vogella.com/tutorials/JavaPDF/article.html>

<http://www.codejava.net/java-ee/javamail/send-e-mail-with-attachment-in-java>

<http://stackoverflow.com/questions/27586940/ruby-gmail-uncaught-exception-534-5-7-14-https-accounts-google-com-continue>

<http://stackoverflow.com/questions/14939607/java-to-send-an-email-via-gmail>