**Documentație Tema 3**

**Managementul comenzilor**

**Student: Tocilă Emilia**

**Grupa 302210**

**Profesor laborator : Antal Marcel**

**Cuprins**

1. Obiectivul temei...........................................................................3
2. Analiza problemei, modelare, scenarii, cazuri de utilizare..........3
3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfața utilizator)........................................................................4
4. Implementare................................................................................4
5. Rezultate.....................................................................................11
6. Concluzii....................................................................................16
7. Bibliografie.................................................................................16

* **Obiectivul temei**
* **Obiectivul principal**

Considerați o aplicație pentru procesarea comenzilor pentru o fabrică. Folosiți baze de date pentru a stoca clienții, produsele și comenzile.

* **Obiective secundare**
* Dezvoltarea de use case-uri si scenarii (cap. 2) – descrisă execuția programului
* Alegerea structurilor de date (cap. 3) – descrise structurile care pot implementa facil problema
* Împărțirea pe clase (cap. 3) – clasele ce vor dezvolta implementările
* Dezvoltarea algoritmilor (cap. 4) – implementarea algoritmilor & metodelor
* Implementarea soluției (cap. 4)
* Rezultate (cap. 5) – verificare
* **Analiza problemei, modelare, scenarii, cazuri de utilizare**

Scopul bazelor de date este de a stoca informații despre diferite obiecte, de a efectua operații pe aceste obiecte, de a stabili relații între acestea. Pentru a facilita realizarea aplicației, am stratificat arhitectura în pachete: model, bll (business logic), presentation, dao, connection, start. Pentru a defini operațiile comune pentru fiecare tabel, am creat clasa AbstractDAO, ce se bazează pe tehnica reflexiei, folosind query-uri generice, completând câmpurile cu informațiile din tabela dorită.

Efectuarea unei operații dorite pe baza de date

**Diagrama use-case:**

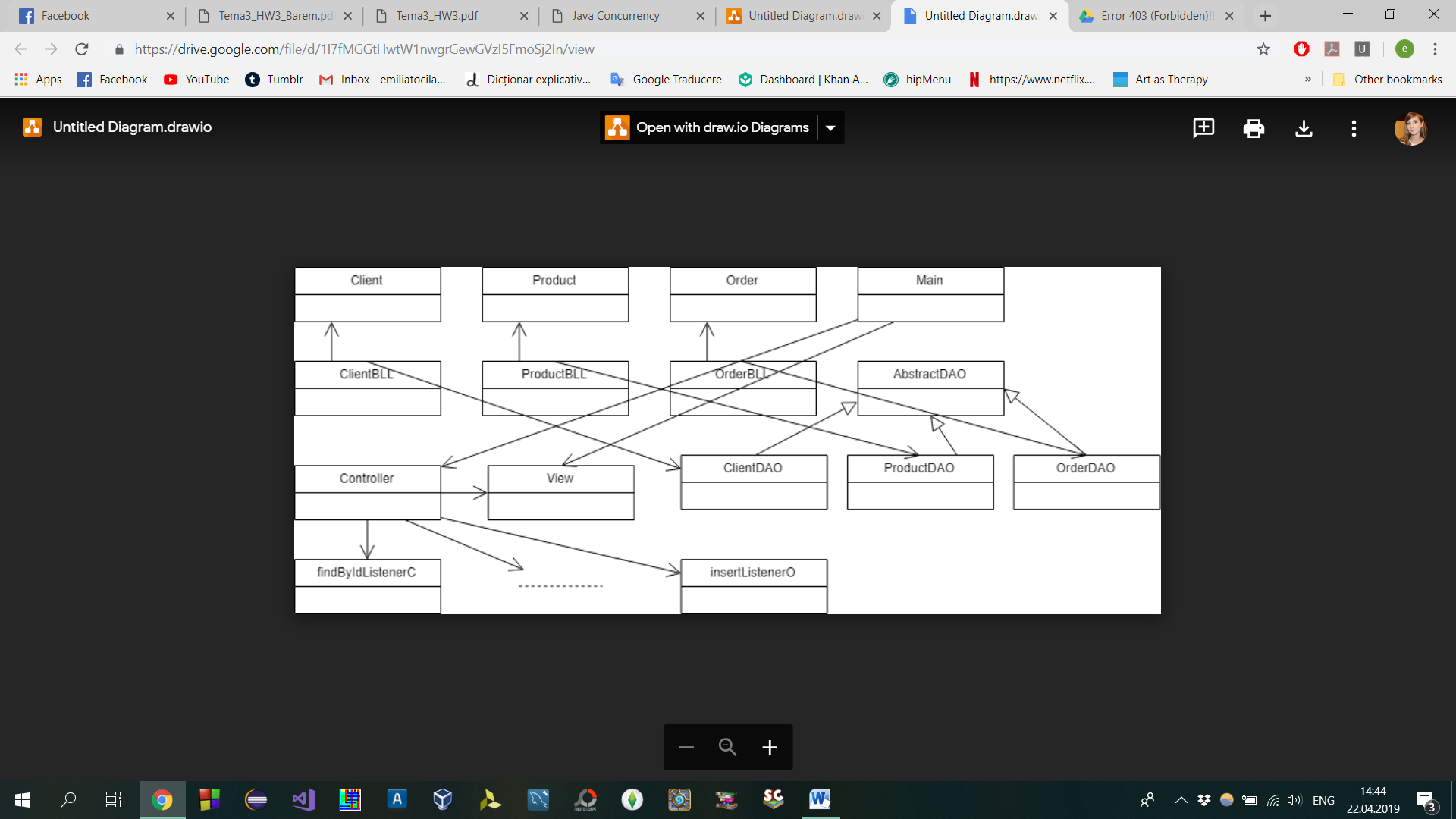
Utilizator

Conectarea la baza de date, executarea query-ului

**Scenariu:**

1. Utilizatorul alege operația dorită pe unul din cele 3 tabele.
2. Utilizatorul introduce datele de intrare dacă acestea sunt necesare.
3. Se realizează executarea query-ului.
4. Se modifică elementele în tabele în baza de date în cazul schimbărilor.
5. Se afișează rezultatul pe interfața grafică.

* **Proiectare**
* **Diagrama UML**



* **Decizii de proiectare**

În pachetul model există clasele Client, Product și Order. Pachetul dao conține clasele AbstractDAO, ClientDAO, ProductDAO, OrderDAO. Pachetul bll: ClientBLL, ProductBLL, OrderBLL. Pachetul presentation: Controller, View. Pachetul connection: ConnectionFactory. Pachetul start: Main. Aceste pachete ”stratificate” au câte un rol special și apelează funcții din straturile de sub acestea. Pachetul model conține clasele mapate după tabelele bazei de date. Pachetul presentation conține clasele care definesc interfața grafică. Pachetul bll conține clasele care încapsulează logica aplicației. Pachetul dao conține clasele ce se ocupă de query-uri și conexiunea la baza de date.

* **Structuri de date**

Principala structură de date folosită a fost ArrayList<T>, pentru a păstra diferite liste: de obiecte de tip Client, Product, Order etc.

* **Implementare**

**Clasa Client:**

Clasa Client este clasa ce reprezintă tabela client din MySqlWorkBench. Câmpurile acestei clase reprezintă coloanele tabelei: id, name, address, email, age. În baza noastră de date, id-ul este primary-key. Această metodă constă într-un constructor, getter-e și setter-e, și se asemenea un toString folosit pentru verificare.

**Clasa Product:**

Clasa Product este clasa ce reprezintă tabela product din MySqlWorkBench. Câmpurile acestei clase reprezintă coloanele tabelei: id, name, price, stock. În baza noastră de date, id-ul este primary-key. Această metodă constă într-un constructor, getter-e și setter-e, și se asemenea un toString folosit pentru verificare.

**Clasa Order:**

Clasa Order este clasa ce reprezintă tabela order din MySqlWorkBench. Câmpurile acestei clase reprezintă coloanele tabelei: orderId, productId, clientId, quantity. În baza noastră de date, orderId-ul este primary-key. Această metodă constă într-un constructor, getter-e și setter-e, și se asemenea un toString folosit pentru verificare.

**Clasa ConnectionFactory:**

Această clasă conține numele driver-ului, inițializat prin reflexie, locația bazei de date (DBURL), user-ul și parola pentru a accesa serverul MySQL. Conexiunea la baza de date va fi ținută într-un obiect Singleton. Clasa conține metode pentru crearea unei conexiuni, obținerea unei conexiuni active și închiderea conexiunii, unui Statement sau unui ResultSet. Pentru ca aplicația Java să interacționeze cu baza de date, o librărie special .jar a fost adăugată. Această librărie trimite query-uri serverului folosind Statements și primește rezultatele query-urilor ca ResultSet.

**Clasa AbstractDAO:**

Clasa AbstractDAO se ocupă de query-uri și de conexiunea la baza de date.

Metoda createSelectQuery crează query-ul general SELECT. Metoda primește ca parametru un String care va fi folosit pentru selecție (id/orderId) și returnează String-ul corespunzător: ”SELECT \* FROM [tableName] WHERE [field] = ?”.

Metoda createSelectAllQuery crează query-ul SELECT ALL. Returnează String-ul: ”SELECT \* FROM [tableName]”.

Metoda createInsertQuery crează query-ul pentru inserție. Se obțin numele coloanelor parcurgând variabilele instanță ale clasei în care se inserează, și de asemenea se adaugă ”?” pentru fiecare. String-ul returnat este: ”INSERT INTO [tableName] ([columnNames]) VALUES (?,..?).

Metoda createDeleteQuery crează query-ul pentru ștergere. String-ul returnat este ”DELETE \* FROM [tableName] WHERE id = ?”.

Metoda createUpdateQuery crează query-ul pentru actualizare. Se obțin numele coloanelor parcurgând variabilele instanță ale clasei și se adaugă ”=?” pentru fiecare. Query-ul returnat va fi: ”INSERT INTO [tableName] SET [columnName] = ?, …”.

Metoda createObjects crează o listă de obiecte de tip T. Un obiect e creat obținând toate numele câmpurilor clasei T, căutând getter-ul (sau orice metodă write) pentru fiecare câmp și invocându-l. Metoda primește ca parametru resultSet-ul care va fi transformat în lista de obiecte.

Metoda findById returnează un obiect de clasă T obținut în urma query-ului rezultat din metoda createSelectQuery. Ca parametri, metoda primește un String ce reprezintă câmpul după care va fi căutat obiectul (id, orderId) și un int ce reprezintă efectiv id-ul ca și număr. Acest int este înlocuit în ”?” din query și acesta este executat. Rezultatul query-ului executat este stocat în resultSet :

statement = connection.prepareStatement(query);

statement.setInt(1, id);

resultSet = statement.executeQuery();

ArrayList<T> found = createObjects(resultSet);

Se apelează meotda createObjects(resultSet) care returnează o listă, însă noi avem nevoie de un singur obiect. Înainte de acest lucru, verificăm dacă s-a returnat o listă goală (nu există obiectul cu acel id), iar dacă aceasta nu este goală, returnăm primul (și singurul element din listă), altfel, null:

if (found.size() != 0)

return found.get(0);

Metoda insert returnează un int (0 sau un număr pozitiv în caz de succes, -1). Este creat query-ul de insert. Valorile câmpurilor obiectului care va fi inserat sunt introduse parcurgând câmpurile clasei respective și invocând setter-ele acestora. În locul ”?” din query vor fi înlocuite aceste valori. Se execută query-ul.

Metoda delete returnează un int (0 succes, -1), primind ca parametru un int ce repezintă id-ul după care se va efectua ștergerea. Se crează query-ul pentru delete, se completează acesta cu id-ul aferent și se execută.

Metoda update returnează un int (0 succes, -1), primind ca parametru obiectul cu noile valori care vor fi actualizate și id-ul corespunzător acelui obiect căruia i se face update-ul. Se crează query-ul și se parcurg câmpurile clasei obiectului și se invocă setter-ele. Valorile obținute se completează în ”?” din query, în afară de câmpul id (acela nu se poate schimba). Se execute query-ul.

Metoda viewAll returnează o listă de obiecte de tipul T, ca rezultat al query-ului SELECT \*.

**Clasa ClientDAO:**

Această clasă extinde clasa AbstractDAO<Client>. Deși nu conține nimic, este important să existe deoarece se transmite ca parametru clasa Client ce face posibilă modelarea metodelor moștenite pe cele trei clase.

**Clasa ProductDAO:**

Această clasă extinde clasa AbstractDAO<Product>. Deși nu conține nimic, este important să existe deoarece se transmite ca parametru clasa Client ce face posibilă modelarea metodelor moștenite pe cele trei clase.

**Clasa OrderDAO:**

Această clasă extinde clasa AbstractDAO<Order>. Deși nu conține nimic, este important să existe deoarece se transmite ca parametru clasa Client ce face posibilă modelarea metodelor moștenite pe cele trei clase.

**Clasa ClientBLL:**

Această clasă conține logica operațiilor. Metoda findClientBy id primește ca parametru id-ul după care se va face căutarea și returnează un ArrayList<Client>. Se crează un obiect de tipul ClientDAO și se apelează metoda findById. Dacă nu a fost găsit niciun client cu id-ul corespunzător, se returnează null, altfel, acesta se adaugă în lista noastră și se returnează lista care conține un singur obiect. Motivul pentru care returnăm un ArrayList și nu un obiect este că vom avea nevoie de o structură de tip ArrayList în view când vom dori să afișăm tabelul cu o singură linie al rezultatului căutării.

Metoda viewAll returnează tot un ArrayList de clienți. Se apelează metoda viewAll. Dacă nu există clienți, se returnează null, altfel, se returnează lista returnată de apelul metodei.

Metoda insert primește ca parametru un obiect de tipul Client care va fi introdus. Generarea id-ului se va face automat : se caută cel mai mic id posibil care nu a fost folosit de alt client și se setează acest id noului client. Se apelează metoda insert.

Metoda delete primește ca parametru un obiect de tipul Client care va fi șters. Se apelează metoda delete ce primește ca parametru id-ul clientului (c.getId()).

Metoda update primește ca parametru un obiect de tipul Client căruia îi vor fi actualizate câmpurile (în afară de id). Se apelează metoda update care primește ca parametru id-ul clientului.

**Clasa ProductBLL:**

Această clasă conține logica operațiilor. Metoda findProductBy id primește ca parametru id-ul după care se va face căutarea și returnează un ArrayList<Product>. Se crează un obiect de tipul ProductDAO și se apelează metoda findById. Dacă nu a fost găsit niciun produs cu id-ul corespunzător, se returnează null, altfel, acesta se adaugă în lista noastră și se returnează lista care conține un singur obiect.

Metoda viewAll returnează tot un ArrayList de produse. Se apelează metoda viewAll. Dacă nu există produse, se returnează null, altfel, se returnează lista returnată de apelul metodei.

Metoda insert primește ca parametru un obiect de tipul Product care va fi introdus. Generarea id-ului se va face automat : se caută cel mai mic id posibil care nu a fost folosit de alt produs și se setează acest id noului produs. Se apelează metoda insert.

Metoda delete primește ca parametru un obiect de tipul Product care va fi șters. Se apelează metoda delete ce primește ca parametru id-ul produsului (p.getId()).

Metoda update primește ca parametru un obiect de tipul Product căruia îi vor fi actualizate câmpurile (în afară de id). Se apelează metoda update care primește ca parametru id-ul produsului.

**Clasa OrderBLL:**

Această clasă conține logica operațiilor. Metoda findProductBy id primește ca parametru id-ul după care se va face căutarea și returnează un ArrayList<Order>. Se crează un obiect de tipul OrderDAO și se apelează metoda findById. Dacă nu a fost găsit nicio comandă cu id-ul corespunzător, se returnează null, altfel, acesta se adaugă în lista noastră și se returnează lista care conține un singur obiect.

Metoda viewAll returnează tot un ArrayList de comenzi. Se apelează metoda viewAll. Dacă nu există comenzi, se returnează null, altfel, se returnează lista returnată de apelul metodei.

Metoda insert primește ca parametru un obiect de tipul Order care va fi introdus. Generarea id-ului se va face automat : se caută cel mai mic id posibil care nu a fost folosit de altă comandă și se setează acest id noii comenzi. Dacă nu se găsește niciun produs cu id-ul corespunzător, se returnează -2 (astfel încât să nu se poată face o comandă pe un produs care nu există). Dacă nu există niciun client cu id-ul corespunzător, se returnează -3.

if (productDAO.findById("id", o.getProductId()) == null)

return -2;

if (clientDAO.findById("id", o.getClientId()) == null)

return -3;

Se crează un obiect de tipul produs și primește valoarea produsului găsit în urma apelării metodei findProductById din clasa ProductBLL. Se face același lucru și pentru client. În variabila locală productStock păstrăm stock-ul produsului returnat mai sus. Se verifică daca se dorește plasarea unei comenzi ale unui produs cu cantitatea mai mare decât stocul disponibil. În cazul acesta, se returnează -1.

Product p = productBll.findProductById(o.getProductId()).get(0);

Client c = clientBll.findClientById(o.getClientId()).get(0);

int productStock = p.getStock();

if (productStock < o.getQuantity())

return -1;

În cazul în care nu există nicio problem menționată mai sus, se apelează metoda insert și se decrementează stock-ul produsului, apoi se face update la acest produs. Se returnează 0.

else {

orderDAO.insert(o);

p.setStock(productStock - o.getQuantity());

productBll.update(p);

return 0;

}

**Clasa View:**

Se ocupă cu aspectul interfeței cu utilizatorul. Interfața va avea 3 tab-uri: Client, Product, Order care vor fi adăugate unui JTabbedPanel.

Metoda generateTable primește un ArrayList<? extends Object> (o listă de obiecte de un tip generic) și returnează un JTable.

String[] columnNames = new String[list.get(0).getClass().getDeclaredFields().length];

Object[][] data = new Object[list.size()][list.get(0).getClass().getDeclaredFields().length];

Vectorul de String-uri va columnNames va conține numele câmpurilor (header-ul tabelei), ce sunt obținute prin reflexie parcurgând câmpurile clasei.

Obiectele, cu câmpurile sale, vor fi obținute prin reflexive astfel: se parcurge lista de obiecte, apoi încă într-un for each se parcurg câmpurile obiectului, se setează acestea ca fiind accesibile și se încearcă scrierea în matricea data la rândul și coloana corespunzătoare obiectul respective (fie el Integer, String etc).

Pentru a seta prima celula să nu fie editabilă (cea cu id-urile), se suprascrie metoda isCellEditable din clasa DefaultTableModel. Se returnează tabelul obținut.

Metoda createTable(C, P, O) crează un tabel astfel: se crează un ArrayList de tipul obiectului, se apelează metoda viewAll din clasa –BLL ce returnează și invocă metoda generateTable ce primește ca parametru Array-ul returnat de metoda viewAll, iar tabelul rezultat este introdus într-un JScrollPane pentru a face posibilă vizualizarea header-ului. La final, JScrollPane-ul se adaugă JPanel-ului corespunzător clasei (clientPanel, productPanel, orderPanel).

Metoda createTableFindById(C, P, O) este asemănătoare, cu singura diferență că metoda apelată din clasa –BLL nu este viewAll, ci find(Client/Product/Order)byId) cu parametrul id, transmis metodei createTableFindById.

Prin metoda init se inițializează toate cele 3 taburi, invocând la rândul ei metodele init(C, P, O). Acestea verifică dacă se crează un table pentru operația findById sau pentru viewAll, și se crează tabelul corespunzător. De asemenea se adaugă butoanele corespunzătoare (Show all, Find by id, Insert, Delete, Update).

Metoda clearPane șterge conținului JPanel-ului claselor (clientPanel etc).

De asemenea, clasa view conține getter-e și setter-e (getFindById(C, P, O), set---, getMainPane, getJTable) și metode ce crează listener pentru butoane.

**Clasa Controller:**

Se ocupă cu corelarea logicii din spatele programului cu partea cu care utilizatorul va interacționa direct. Clasele ---Listener (--- : showAll-, findById-, insert-, delete-, update- (C, P, O)) care implementează ActionListener au metoda actionPerformed, cu parametrul ActionEvent e, fac întocmai acest lucru.

Metoda din clasele showAll Listener apelează metoda clearPane și reinițializează panel-urile.

Metoda din clasele findByIdListener deschid un JDialog unde utilizatorul introduce id-ul căutat. Dacă nu există acel id, se afișează un mesaj de eroare. Altfel, se semnalează view-ului id-ul, se șterg panel-urile și se reinițializează, generând tabelele corespunzătoare operației. După acest lucru se setează pe 0 faptul că se face operația de findAll.

Metoda din clasele insertListener deschide un JDialog unde utilizator introduce datele noului obiect, se apelează metoda insert din clasa –BLL, și se reinițializează tabelele.

Metoda din clasele deleteListener funcționează astfel: utilizatorul selectează cu un click rândul obiectului pe care dorește să îl șteargă, apasă butonul și obiectul este șters.

ClientBLL clientBll = new ClientBLL();

JTable table = view.getJTable("client");

int row = table.getSelectedRow();

if (table.isRowSelected(table.getSelectedRow())){

Client c = new Client((int)table.getValueAt(row, 0), table.getValueAt(row, 1).toString(), table.getValueAt(row, 2).toString(), table.getValueAt(row, 3).toString(), (int)table.getValueAt(row, 4));

clientBll.delete(c);

view.clearPane();

view.init();

}

else

JOptionPane.showMessageDialog(view.getMainPane(), "Select a client to be deleted!", "Try again", JOptionPane.PLAIN\_MESSAGE);

Metoda din clasele updateListener funcționează astfel: utilizatorul selectează rândul pe care dorește să îl actualizeze și cu un dublu click, selectează celula pentru modificare, o modifică și apasă enter. După acest lucru, apasă butonul de update.

ClientBLL clientBll = new ClientBLL();

JTable table = view.getJTable("client");

int row = table.getSelectedRow();

if (table.isRowSelected(table.getSelectedRow())){

if (table.isEditing())

table.getCellEditor().stopCellEditing();

Client c = new Client((int)table.getValueAt(row, 0), table.getValueAt(row, 1).toString(), table.getValueAt(row, 2).toString(), table.getValueAt(row, 3).toString(), Integer.parseInt(table.getValueAt(row, 4).toString()));

clientBll.update(c);

view.clearPane();

view.init();

}

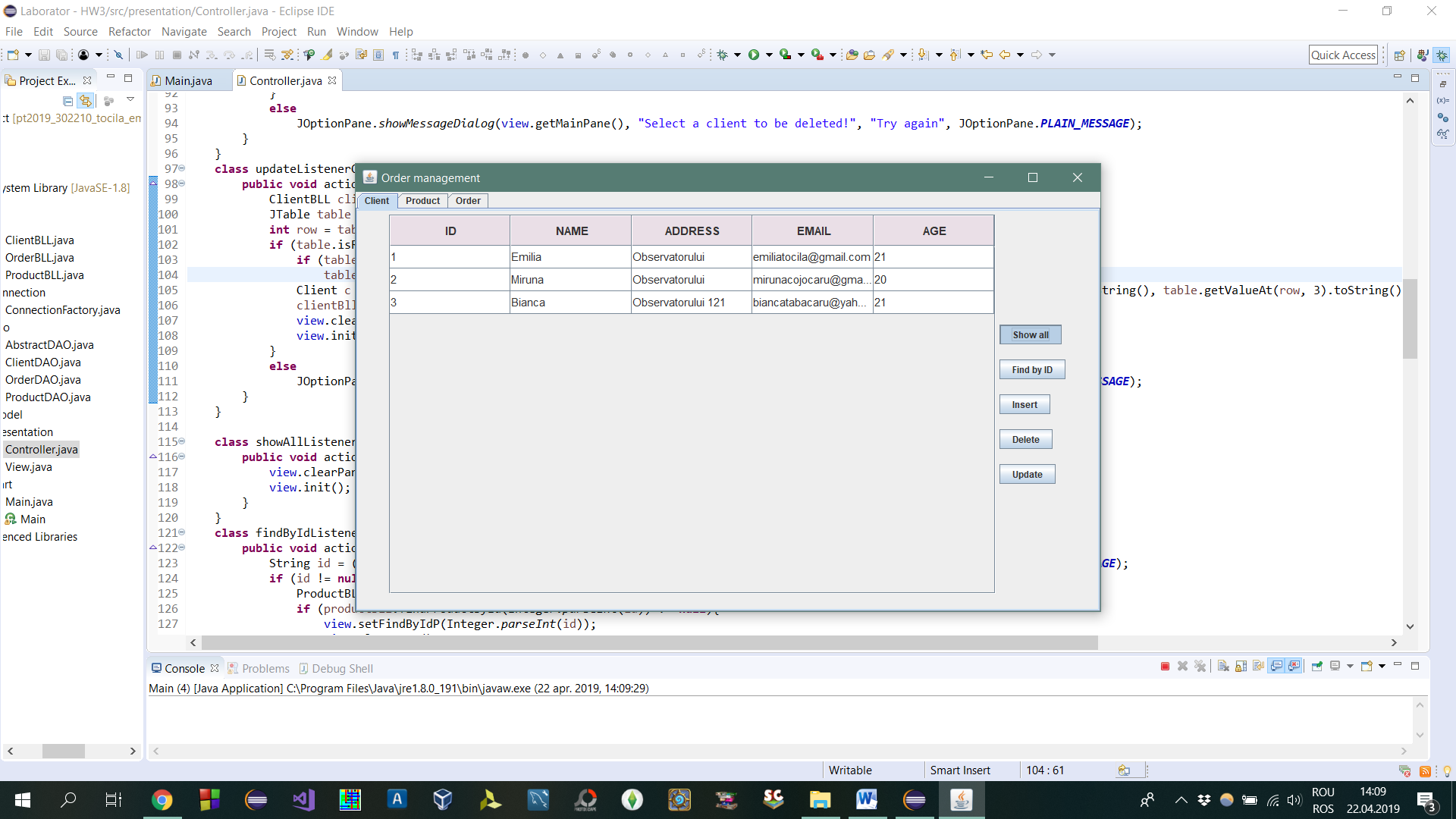
else

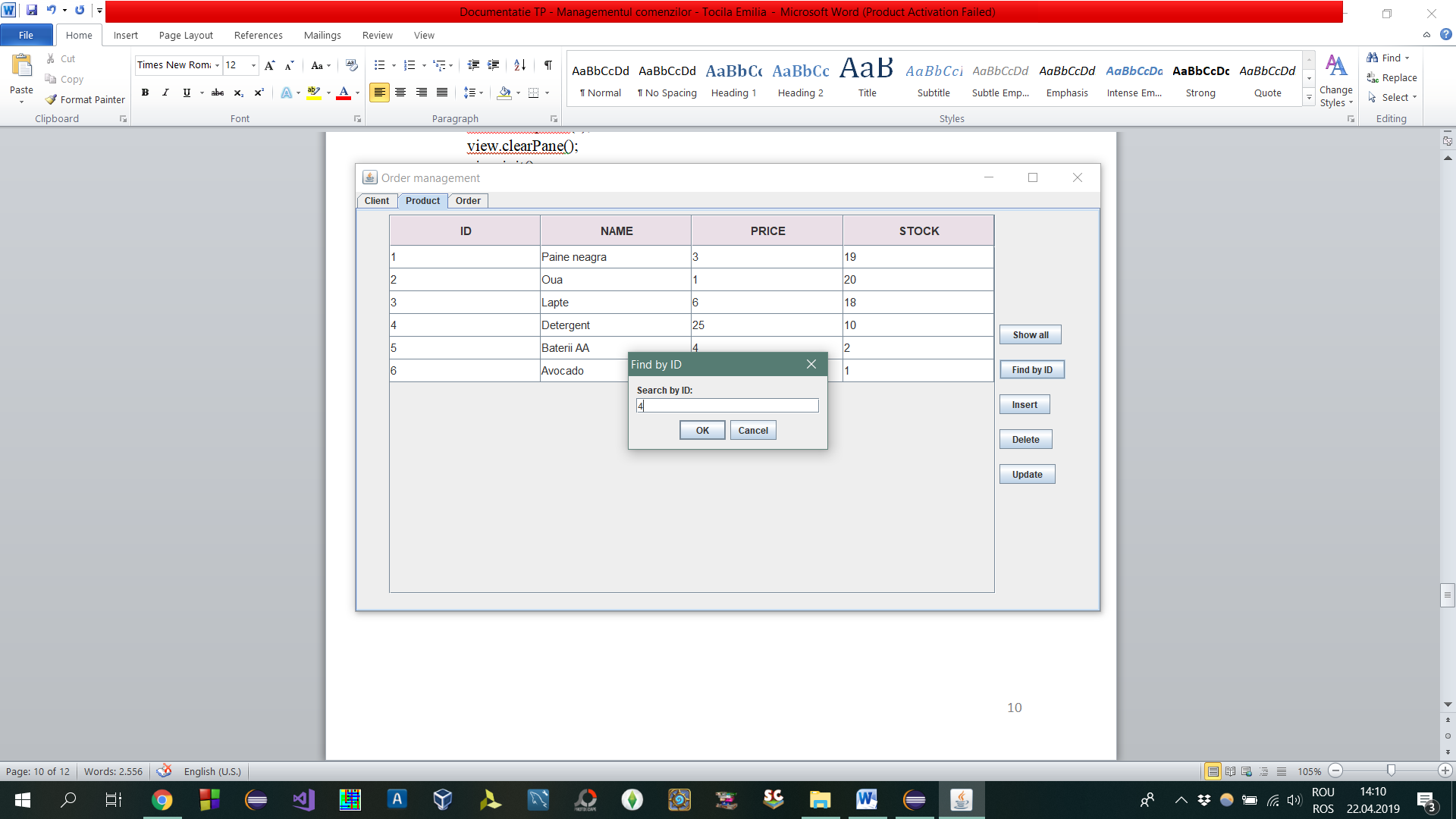
JOptionPane.showMessageDialog(view.getMainPane(), "Select a client to be updated!", "Try again", JOptionPane.PLAIN\_MESSAGE);

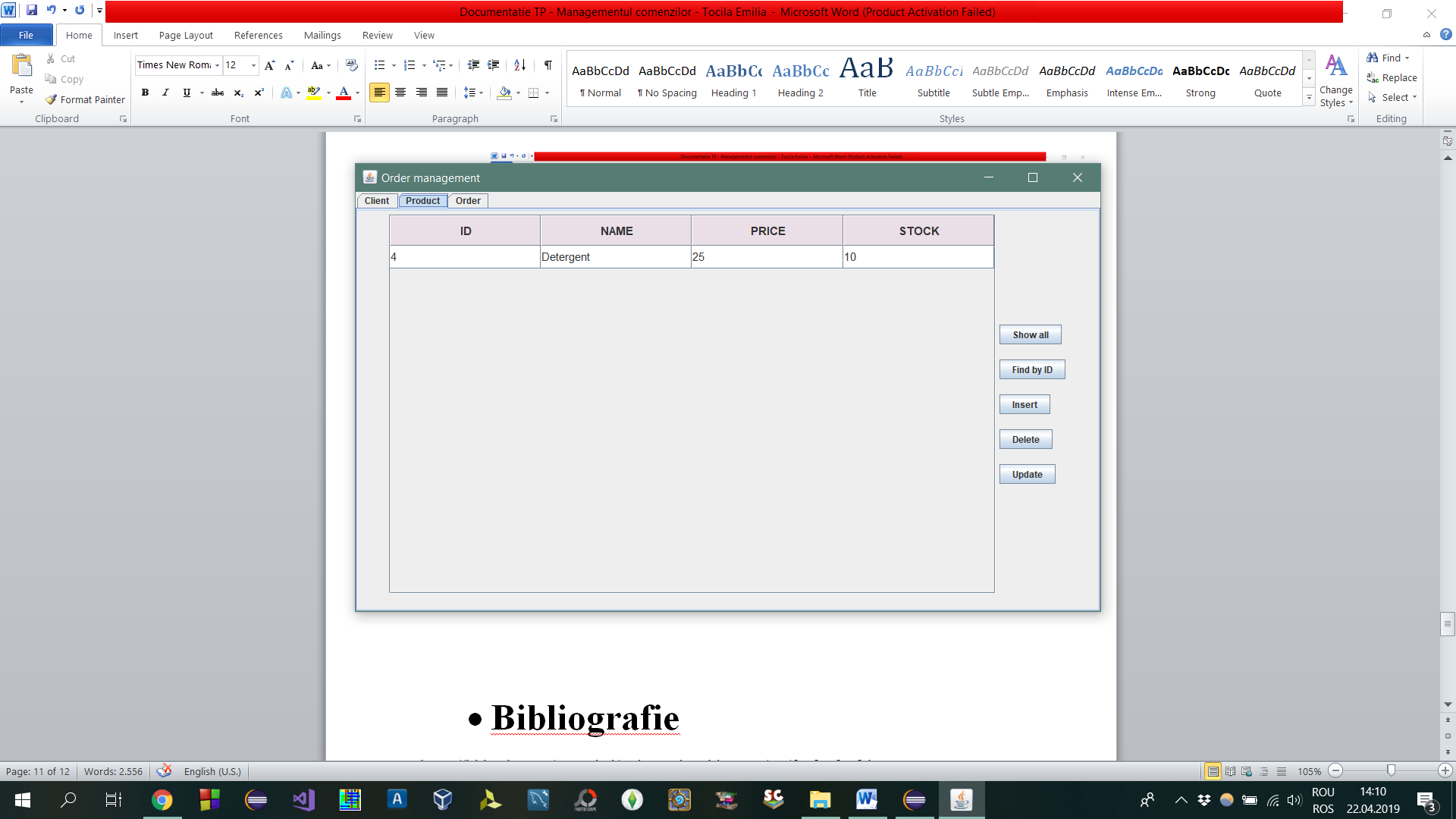
**Clasa Main:**

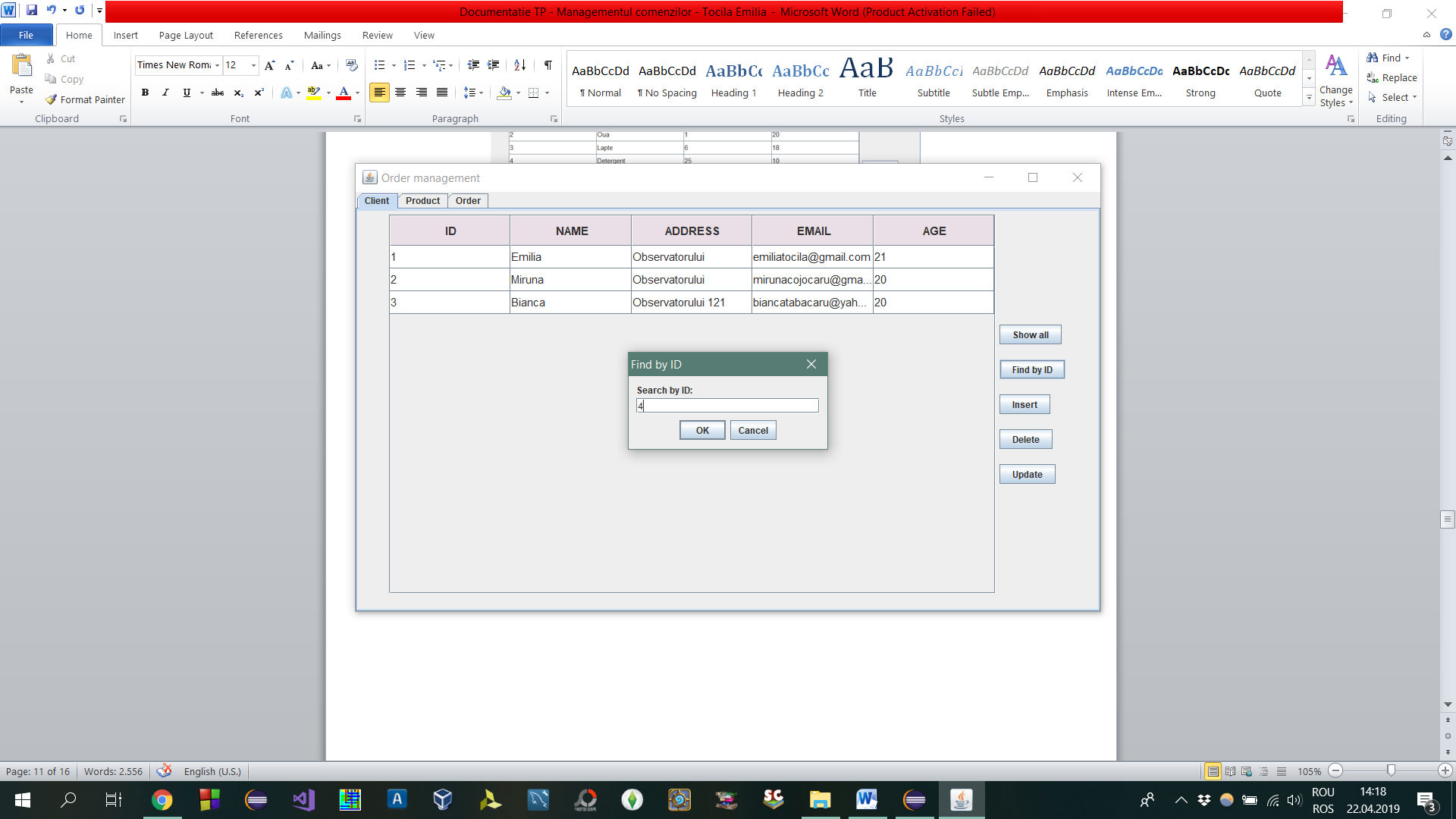
Clasa Main leagă View-ul de Controller și de model.

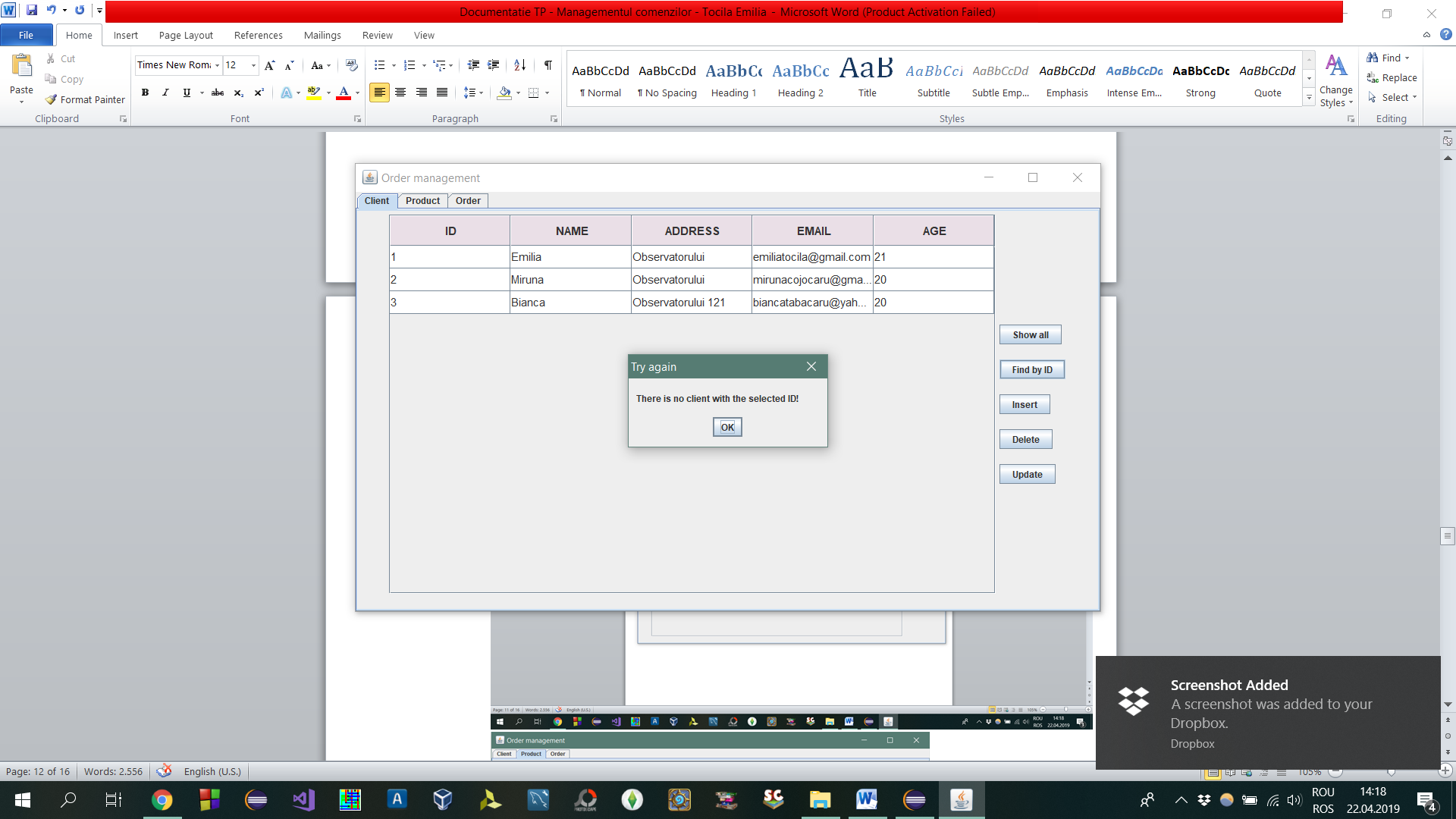
* **Rezultate**

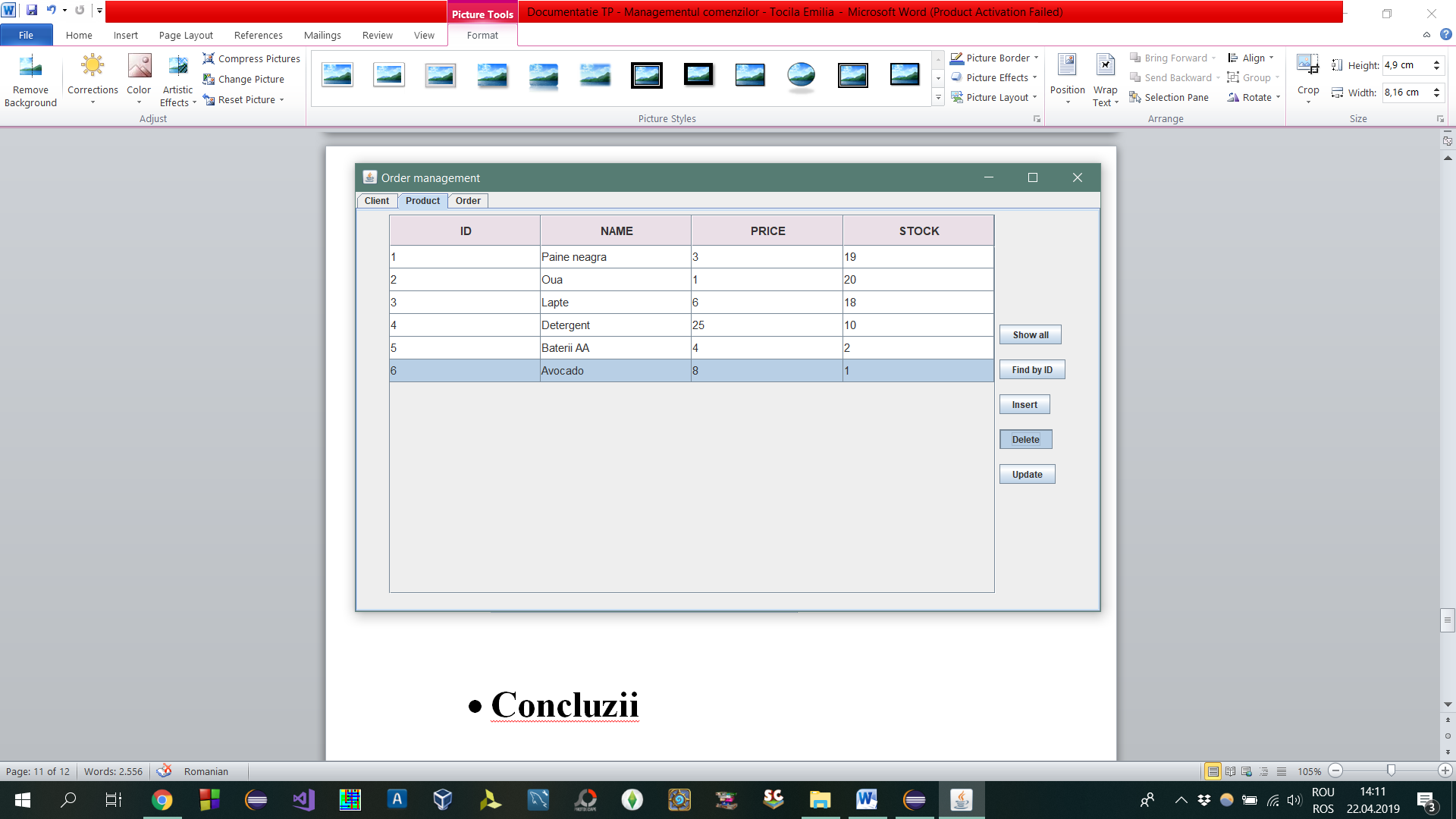


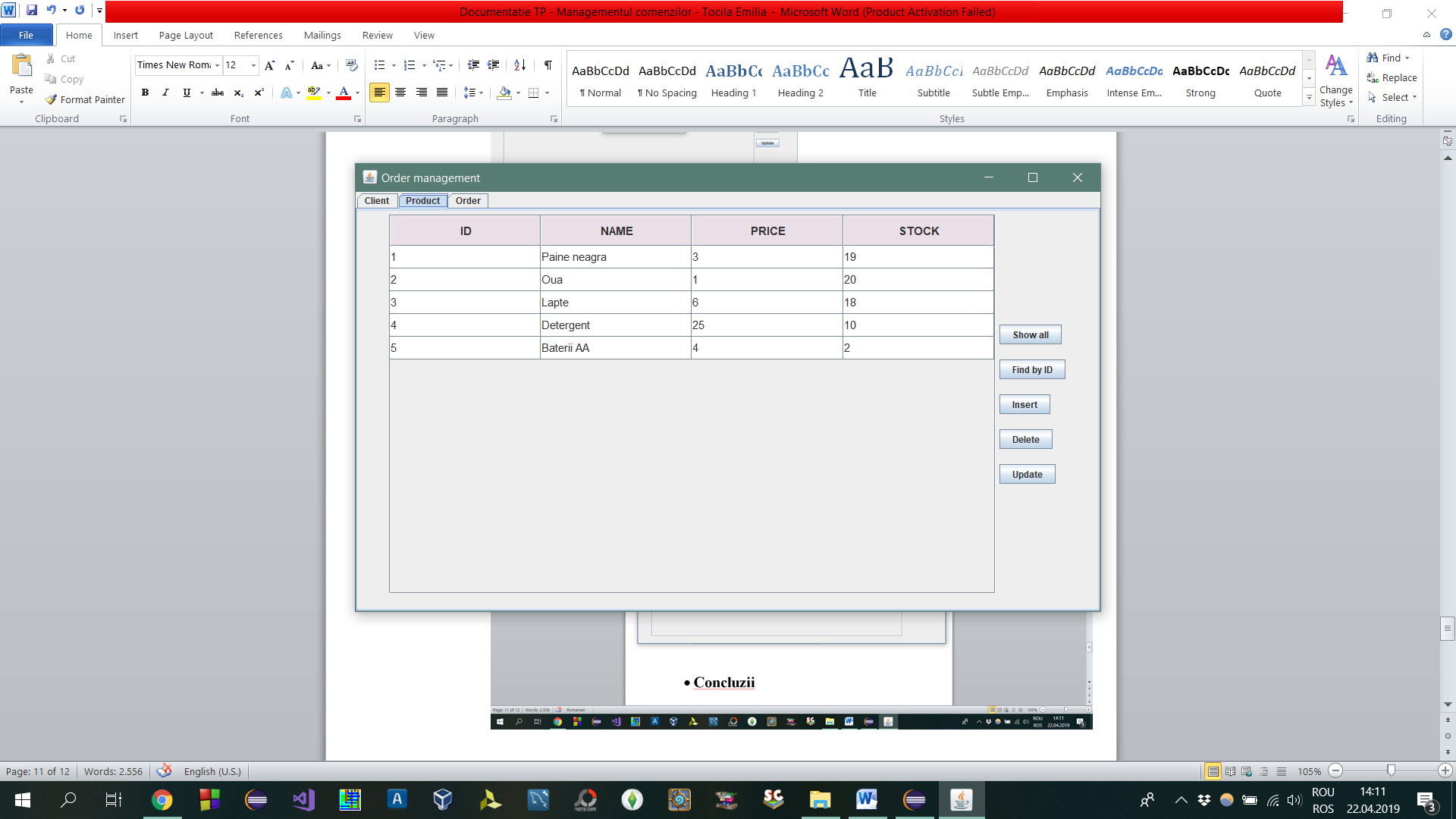


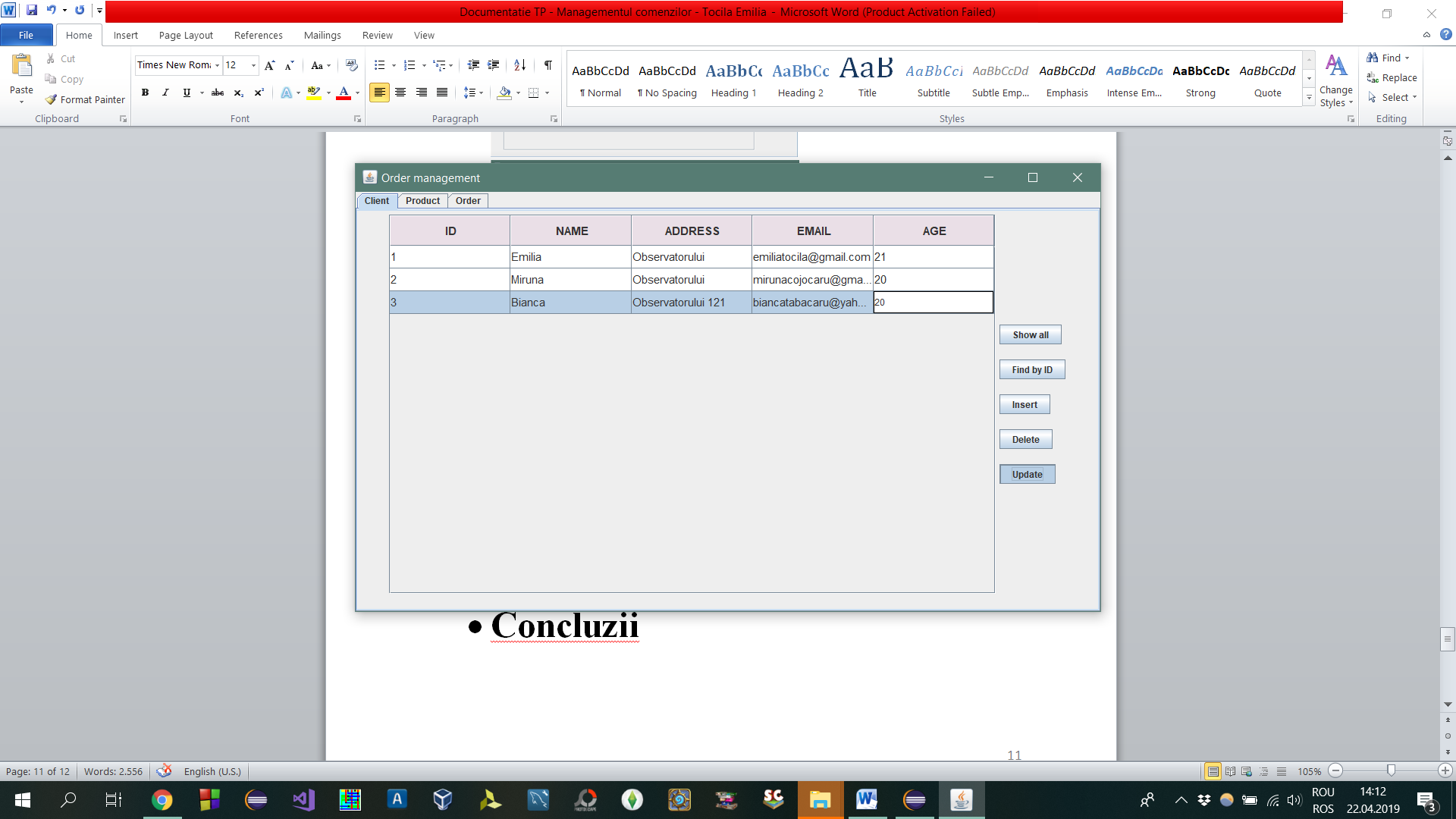


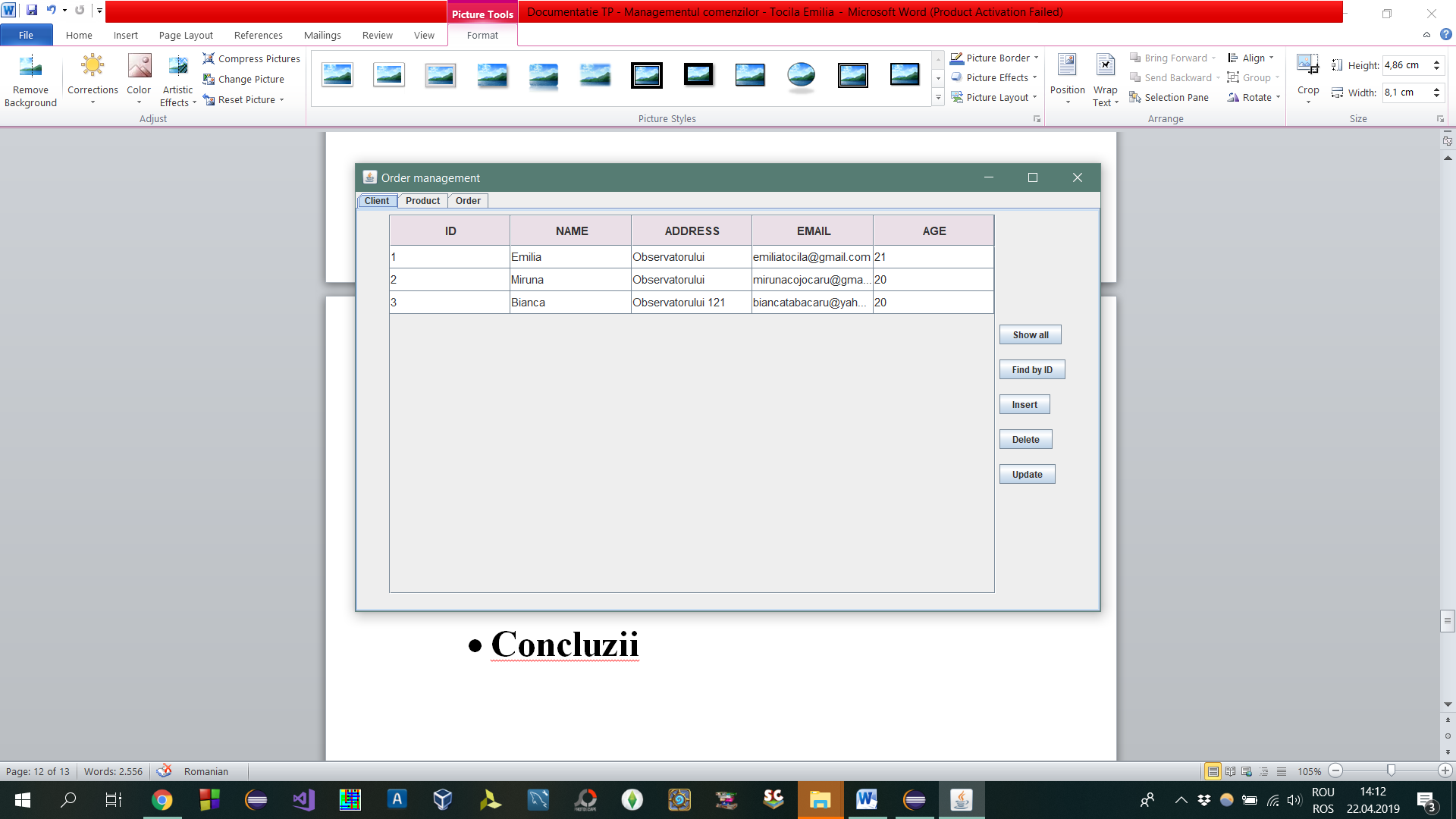


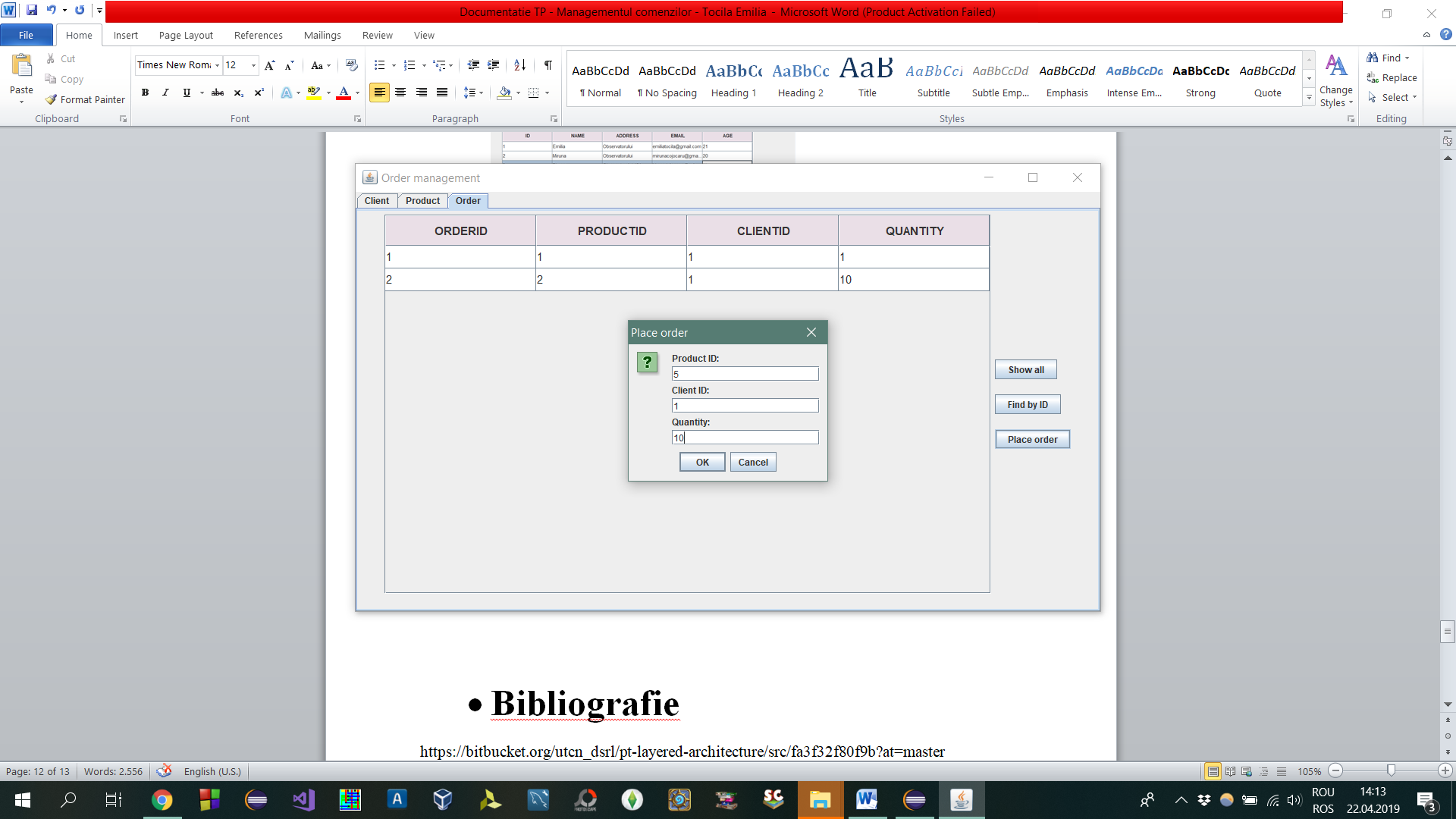


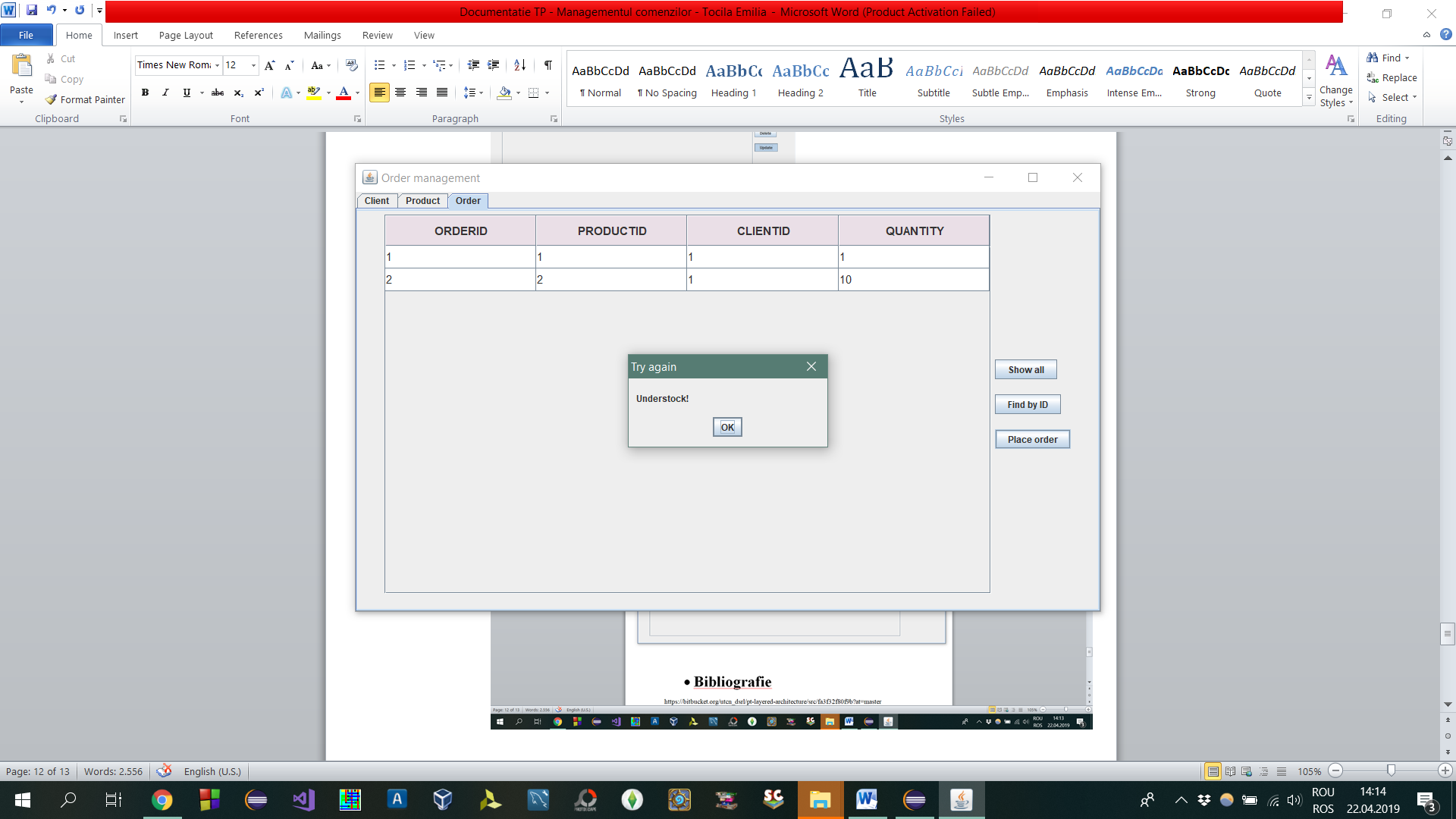


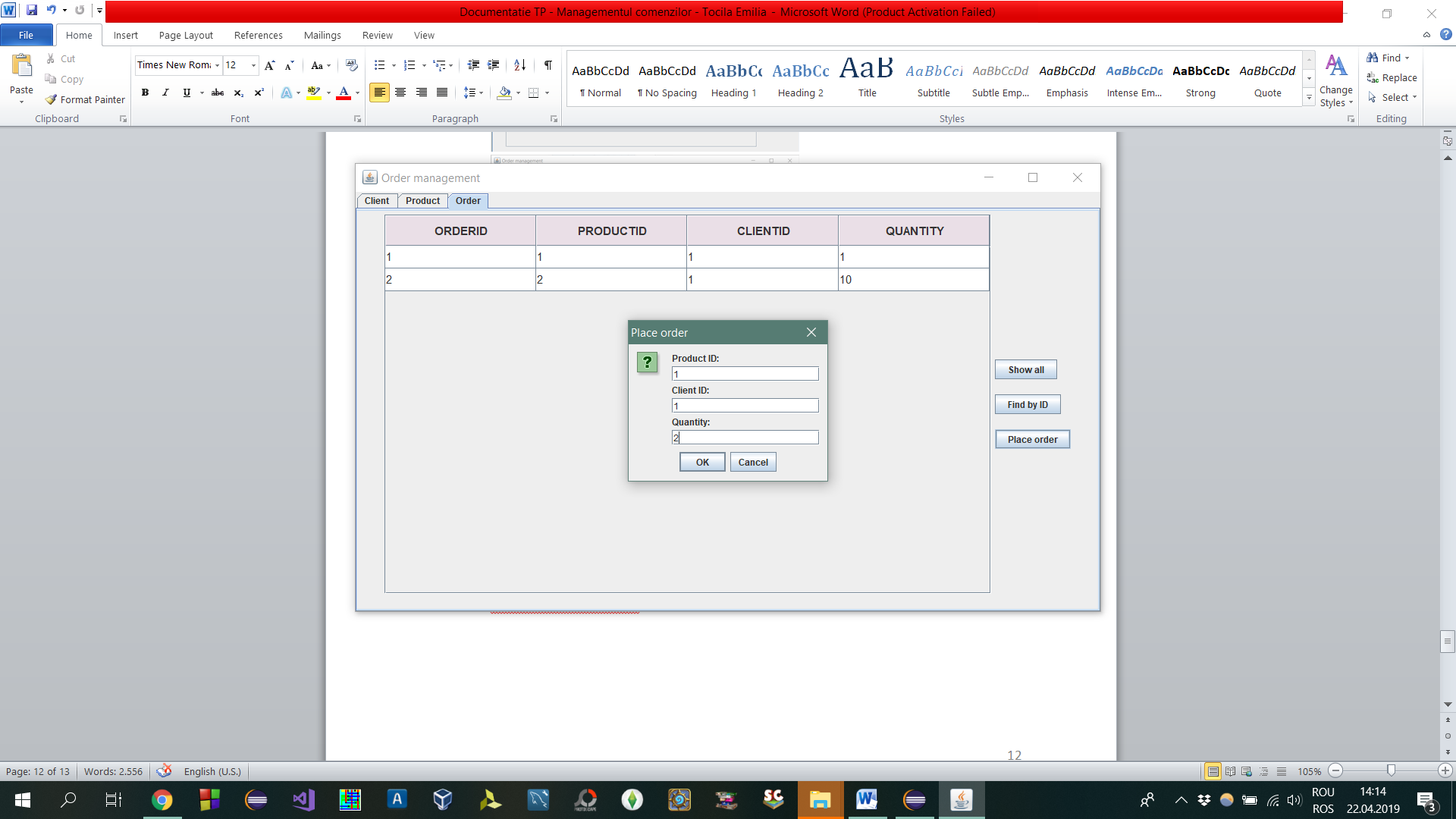


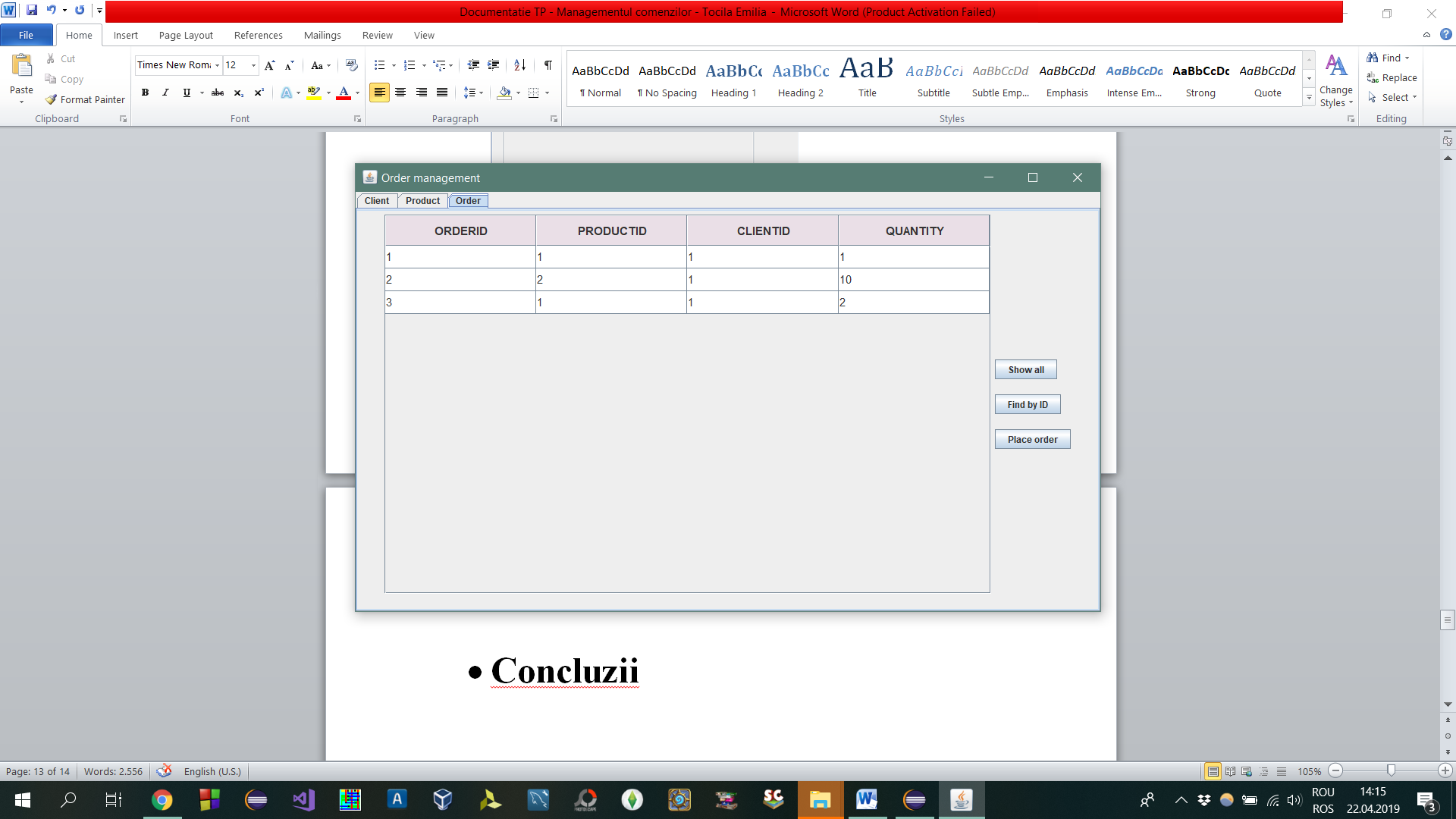


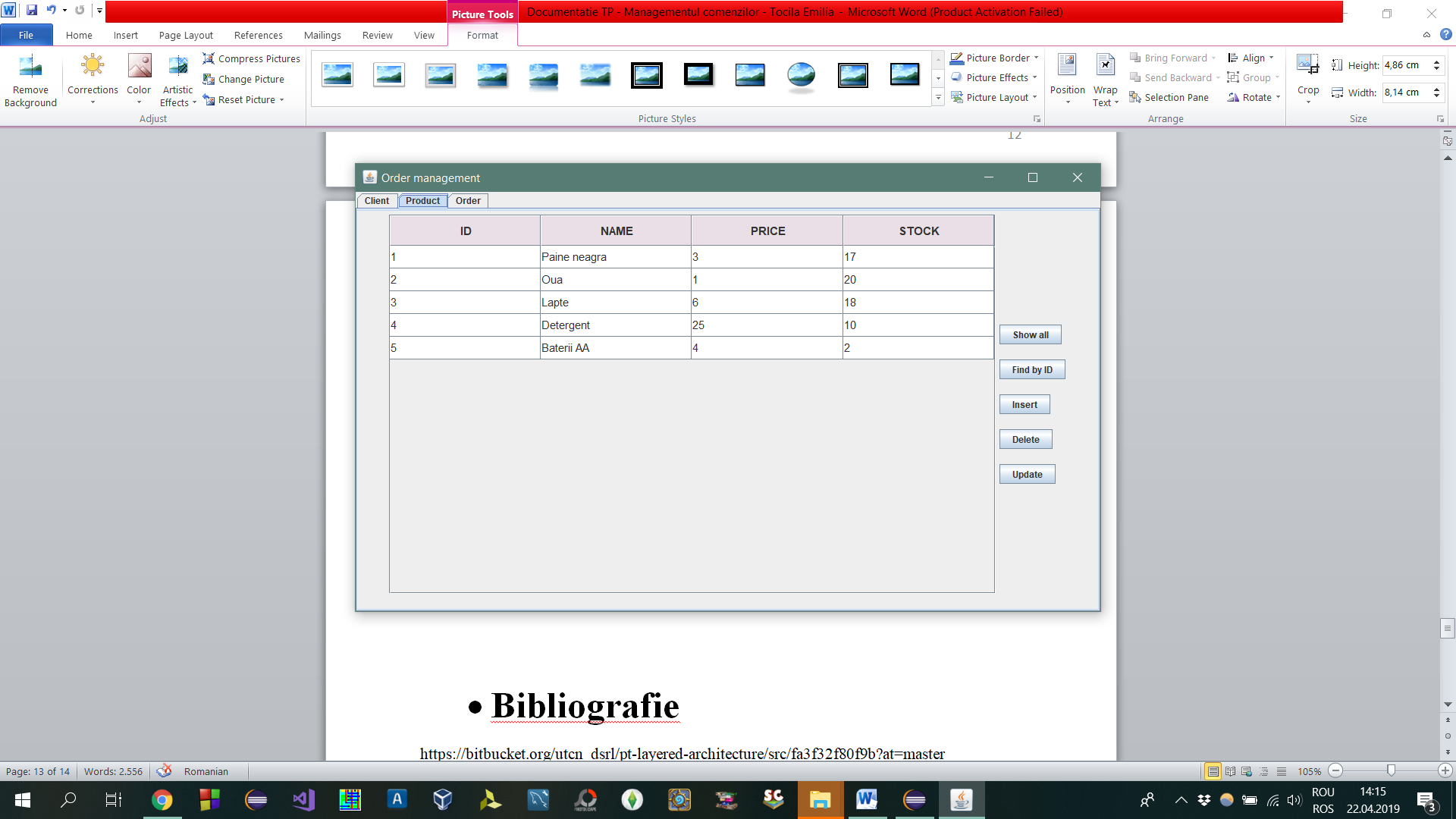


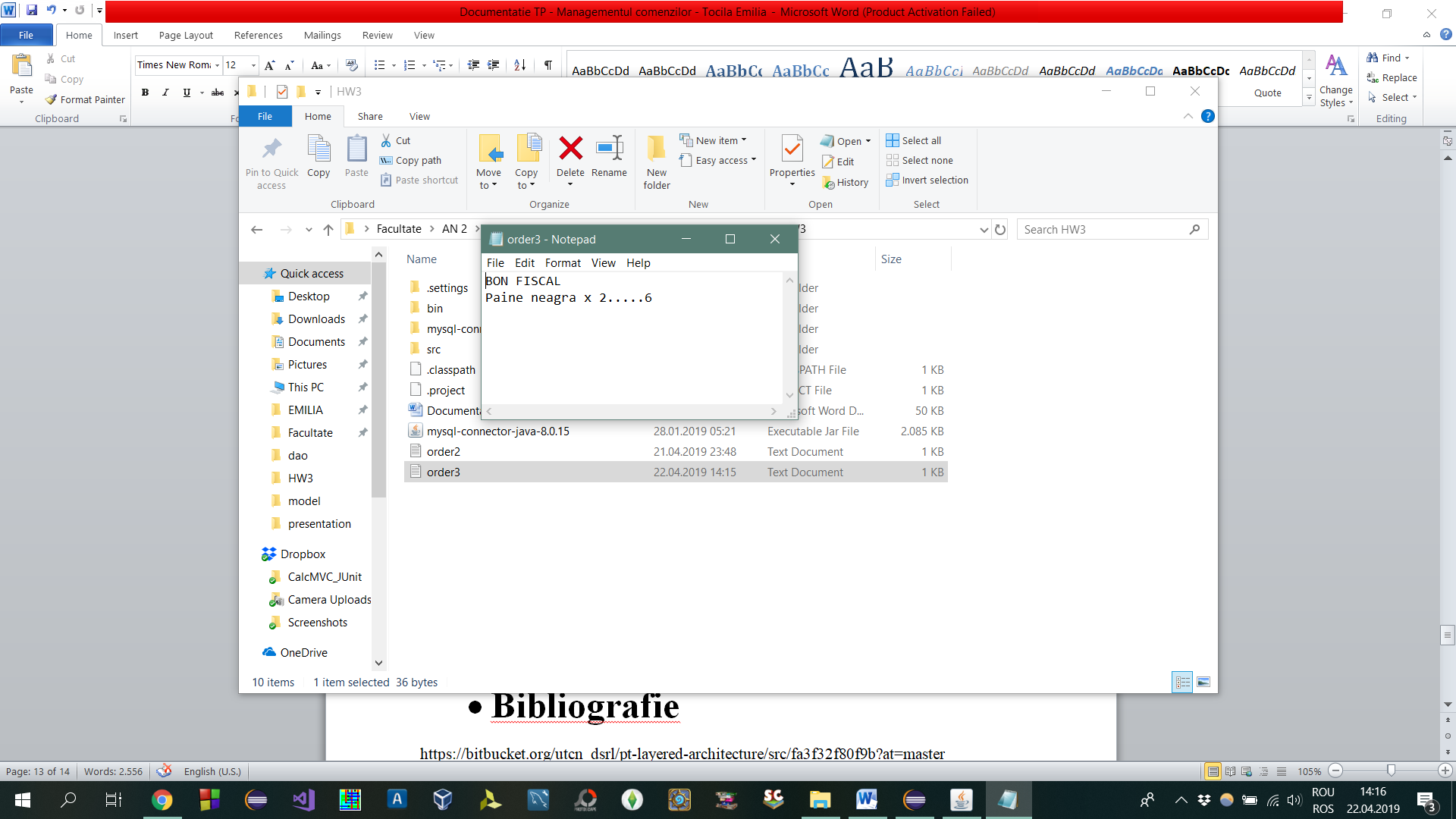












* **Concluzii**

În urma dezvoltării acestei aplicații am învățat foarte multe lucruri: cum se folosește tehnica reflexiei, cum funcționează astfel bazele de date, cum să creez interfețe grafice mai complexe.

* **Bibliografie**

https://bitbucket.org/utcn\_dsrl/pt-layered-architecture/src/fa3f32f80f9b?at=master

https://docs.oracle.com/javase/tutorial/uiswing/components/table.html#simple

https://www.roseindia.net/java/example/java/swing/jtable-display-database-data.shtml

https://www.logicbig.com/tutorials/java-swing/generate-jtable-model-with-annotation.html

https://stackoverflow.com/questions/45896437/how-display-in-jtable-the-data-from-mysql#

https://stackoverflow.com/questions/953972/java-jtable-setting-column-width

https://docs.oracle.com/javase/tutorial/uiswing/components/tabbedpane.html

https://stackoverflow.com/questions/48667656/center-text-in-jtable/48704508

https://stackoverflow.com/questions/11686535/how-to-change-jtable-header-height

https://coderanch.com/t/342466/java/JTable-model-changed-update-view

https://stackoverflow.com/questions/29345792/java-jtable-getting-the-data-of-the-selected-row

https://docs.oracle.com/javase/7/docs/api/javax/swing/JTable.html

https://stackoverflow.com/questions/6555040/multiple-input-in-joptionpane-showinputdialog/6555051

https://coderanch.com/t/655711/java/Field-JTable-editable

https://stackoverflow.com/questions/33739941/read-cell-value-in-jtable-while-it-is-being-edited

https://stackoverflow.com/questions/35414482/get-column-value-from-jtable-after-table-is-been-edited

https://stackoverflow.com/questions/2885173/how-do-i-create-a-file-and-write-to-it-in-java