Ulics Daniel-Alexandru

Grupa 30224

Tema 3

Aplicație pentru procesat comenzi

1. Obiectivul temei

Obiectivul principal al acestei teme reprezintă realizarea unei aplicații care interacționează cu o bază de date pentru a procesa comenzile unor clienți.

Obiective secundare:

* Analizarea problemei și identificarea funcționalităților necesare : prin acest obiectiv se dorește o analizare amănunțită a problemei propuse, cât și descoperirea funcționalităților pe care trebuie să le îndeplinească aplicația. Acest obiectiv va fi detaliat în capitolul 2.;
* Proiectarea aplicației pentru procesarea comenzilor : acest obiectiv se referă la găsirea unei soluții adecvate pentru a implementa problema propusă. Acest obiectiv va fi detaliat în capitolul 3. ;
* Implementarea aplicației : la acest pas se va realiza implementarea propriu-zisă a aplicației în mediul de dezvoltare IntelliJ, alături de MySQLWorkbench. Acest obiectiv va fi detaliat în capitolul 4. ;
* Verificarea aplicației realizare : la acest pas se va observa dacă aplicația realizată întrunește cerințele specificate. Acest obiectiv va fi detaliat în capitolul 5. ;

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

Principalele caracteristici funcționale ale aplicației sunt:

* Aplicația trebuie să permită vizualizarea tuturor clienților, produselor și comenzilor existente ;
* Aplicația trebuie să permită adăugarea unui client nou ;
* Aplicația trebuie să permită modificarea unui client ;
* Aplicația trebuie să permită ștergerea unui client ;
* Aplicația trebuie să permită căutarea unui client după un anume atribut ;
* Aplicația trebuie să permită adăugarea unui produs nou ;
* Aplicația trebuie să permită modificarea unui produs ;
* Aplicația trebuie să permită ștergerea unui produs ;
* Aplicația trebuie să permită căutarea unui produs după un anume atribut ;
* Aplicația trebuie să permită plasarea unei noi comenzi și generarea unei facturi aferente ;

Principalele caracteristici non-funcționale ale aplicației sunt:

* Grad redus de dificultate pentru utilizator ;
* Răspuns rapid la solicitările utilizatorului ;
* Păstrarea consistentă a datelor ;

Use case pentru operația de adăugare a unui client/produs :

* Actor principal : utilizatorul
* Scenariu principal în caz de succes :

1. Utilizatorul apasă pe butonul pentru clienți/produse.
2. Utilizatorul introduce datele noului client/produs în interfață.
3. Utilizatorul apasă pe butonul de inserare.
4. Noul client/produs este adăugat, fiind vizibil în tabelul corespunzător.

* Scenarii alternative :
  + Date incorecte :
    - Utilizatorul introduce date invalide într-unul din câmpurile de adăugare.
    - Se va afișa un mesaj în interfață și se revine la pasul numărul 2.

Use case pentru operația de modificare a unui client/produs :

* Actor principal : utilizatorul
* Scenariu principal în caz de succes :

1. Utilizatorul apasă pe butonul pentru clienți/produse.
2. Utilizatorul selectează un client/produs din interfață.
3. Utilizatorul introduce în câmpurile disponibile noile valori.
4. Utilizatorul apasă pe butonul de actualizare.
5. Clientul/produsul este modificat, fiind vizibilă modificarea în tabelul corespunzător.

* Scenarii alternative :
  + Date incorecte :
    - Utilizatorul introduce date invalide într-unul din câmpurile din interfață.
    - Se va afișa un mesaj în interfață și se revine la pasul numărul 2.
  + Neselectarea unui client/produs :
    - Utilizatorul nu selectează un client/produs pentru modificare.
    - Se va afișa un mesaj în interfață și se revine la pasul numărul 2.

Use case pentru operația de ștergere a unui client/produs :

* Actor principal : utilizatorul
* Scenariu principal în caz de succes :

1. Utilizatorul apasă pe butonul de clienți/produse.
2. Utilizatorul selectează un client/produs din interfață.
3. Utilizatorul apasă pe butonul de ștergere.
4. Clientul/produsul este șters, fiind eliminat din tabelul corespunzător.

* Scenarii alternative :
  + Neselectarea unui client/produs :
    - Utilizatorul nu selectează un client/produs pentru modificare.
    - Se va afișa un mesaj în interfață și se revine la pasul numărul 2.

Use case pentru operația de căutare a unui client/produs :

* Actor principal : utilizatorul
* Scenariu principal în caz de succes :

1. Utilizatorul apasă pe butonul pentru clienți/produse.
2. Utilizatorul apasă pe butonul de găsire.
3. Utilizatorul introduce id-ul clientului/produsului căutat.
4. Utilizatorul apasă pe butonul de confirmare.
5. Se va afișa în tabel clientul/produsul căutat.

* Scenarii alternative :
  + Introducerea unui id greșit :
    - Utilizatorul introduce la pasul numărul 3 un id greșit.
    - Se va afișa un mesaj în interfață și se revine la pasul numărul 3.

Use case pentru operația de plasare a unei noi comenzi :

* Actor principal : utilizatorul
* Scenariu principal în caz de succes :

1. Utilizatorul apasă pe butonul pentru comenzi.
2. Utilizatorul apasă pe butonul pentru crearea unei comenzi noi.
3. Utilizatorul selectează un client și un produs.
4. Utilizatorul introduce cantitatea dorită.
5. Utilizatorul apasă pe butonul de confirmare.
6. Comanda este plasată, fiind vizibilă în tabelul corespunzător, fiind generată și o factură.

* Scenarii alternative :
  + Neselectarea unui client și/sau produs :
    - Utilizatorul nu selectează un client și un produs pentru comandă.
    - Se va afișa un mesaj în interfață și se revine la pasul numărul 3.
  + Introducerea unui cantități greșite :
    - Utilizatorul introduce o cantitate greșită pentru comandă.
    - Se va afișa un mesaj în interfață și se revine la pasul numărul 4.
  + Nu există stoc suficient :
    - Cantitatea introdusă depășește stocul produsului.
    - Se va afișa un mesaj în interfață și se revine la pasul numărul 4.

1. Proiectare

Aplicația utilizează o arhitectură de tip layered, în care clasele sunt împărțite în următoarele pachete : model, dataAccesLayer, businessLayer și presentationLayer.

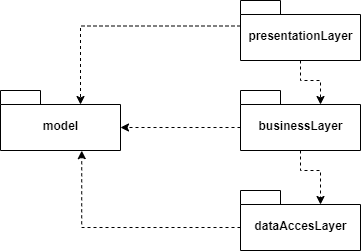
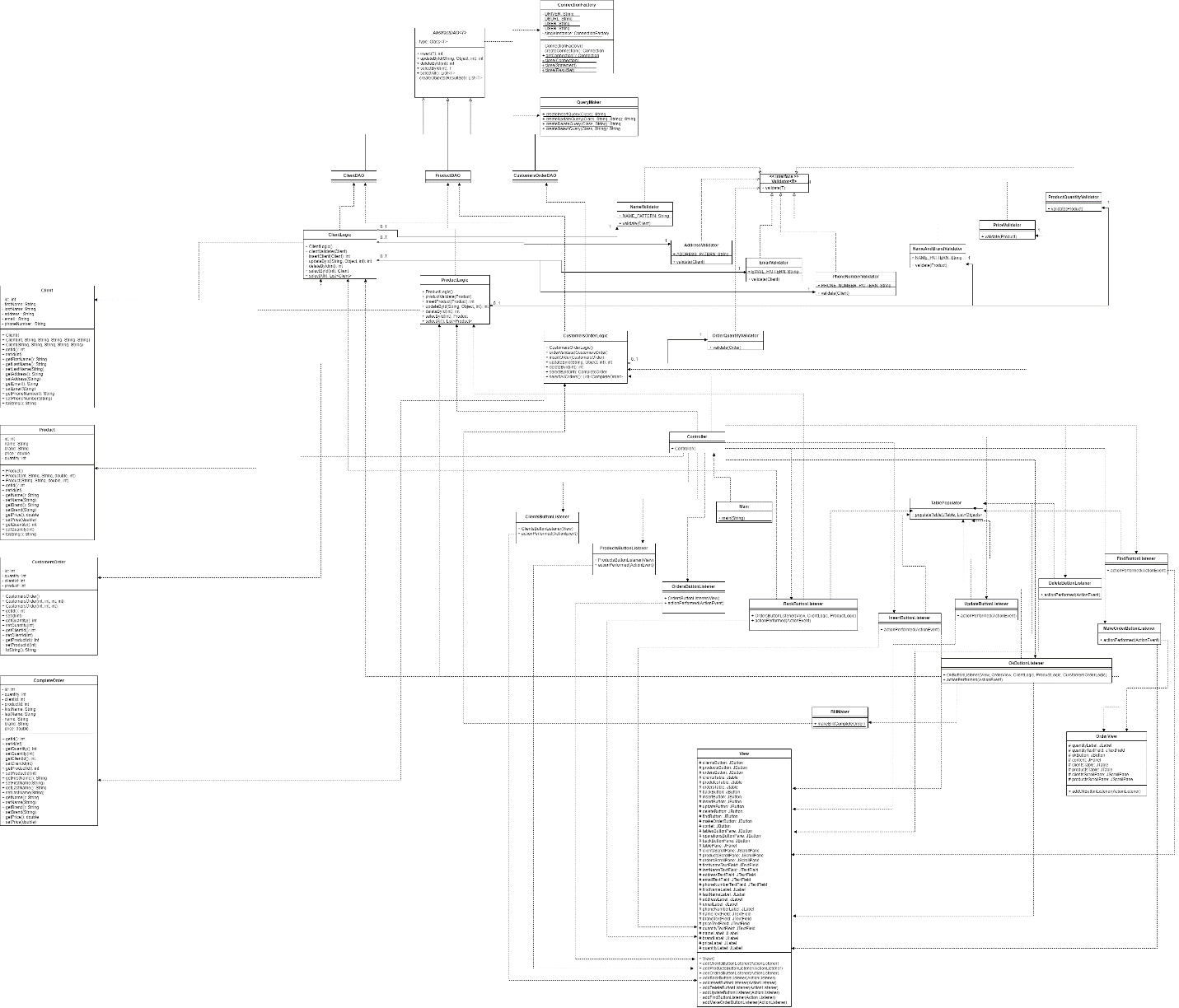


Diagrama de clase a sistemului :



1. Implementare

Clasa Client este utilizată pentru a reprezenta un obiect din tabelul cu clienți.

Câmpuri :

* id – identificator unic
* firstName
* lastName
* address
* email
* phoneNumber

Metode :

* getId
* setId
* getFirstName
* setFirstName
* getLastName
* setLirstName
* getAddress
* setAddress
* getEmail
* setEmail
* getPhoneNumber
* setPhoneNumber
* toString

Clasa Product este utilizată pentru a reprezenta un obiect din tabela cu produse.

Câmpuri :

* id
* name
* brand
* price
* quantity

Metode :

* getId
* setId
* getName
* setName
* getBrand
* setBrand
* getPrice
* setPrice
* getQuantity
* setQuantity
* toString

Clasa CustomersOrder este utilizată pentru a reprezenta un obiect din tabela de comenzi.

Câmpuri :

* id
* quantity
* clientId – identificatorul clientului care a realizat comanda
* productId – identificatorul produsului pentru care s-a realizat comanda

Metode :

* getId
* setId
* getQuantity
* setQuantity
* getClientId
* setClientId
* getProductId
* setProductId
* toString

Clasa CompleteOrder este utilizată pentru a reprezenta o comandă, alături de clientul care a comanda-to și produsul pe care l-a comandat.

Câmpuri :

* id
* quantity
* clientId
* productId
* firstName
* lastName
* name
* brand
* price

Metode :

* getId
* setId
* getQuantity
* setQuantity
* getClientId
* setClientId
* getProductId
* setProductId
* getFirstName
* setFirstName
* getLastName
* setLastName
* getName
* setName
* getBrand
* setBrand
* getPrice
* setPrice

Clasa QueryMaker este utilizată pentru a crea string-urile necesare folosite in query-uri.

Metode :

* createInsertQuery - metoda este utilizată pentru a crea un string pentru un query de inserare.

StringBuilder stringBuilder=new StringBuilder();

stringBuilder.append("INSERT INTO ");

stringBuilder.append(type.getSimpleName());

stringBuilder.append("(");

StringBuilder fields=new StringBuilder();

Field[] fieldsList=type.getDeclaredFields();

for(int i=1;i<fieldsList.length;i++)

{

fields.append(fieldsList[i].getName());

if((i+1)!=fieldsList.length)

fields.append(",");

}

stringBuilder.append(String.valueOf(fields));

stringBuilder.append(") VALUES (");

fields=new StringBuilder();

for(int i=1;i< fieldsList.length;i++)

{

fields.append("?");

if((i+1)!=fieldsList.length)

fields.append(",");

}

stringBuilder.append(fields);

stringBuilder.append(")");

return String.valueOf(stringBuilder);

* createUpdateQuery - metodă utilizată pentru a crea un query pentru actualizare.

StringBuilder stringBuilder=new StringBuilder();

stringBuilder.append("UPDATE ");

stringBuilder.append(type.getSimpleName());

stringBuilder.append(" SET ");

stringBuilder.append(updateField);

stringBuilder.append("=? ");

stringBuilder.append("WHERE ");

stringBuilder.append(filterField);

stringBuilder.append("=?");

return String.valueOf(stringBuilder);

* createDeleteQuery - metodă utilizată pentru a crea string-ul pentru un query de stergere.

StringBuilder stringBuilder=new StringBuilder();

stringBuilder.append("DELETE FROM ");

stringBuilder.append(type.getSimpleName());

stringBuilder.append(" WHERE ");

stringBuilder.append(field);

stringBuilder.append("=?");

return String.valueOf(stringBuilder);

* createSelectQuery - metodă utilizată pentru a crea string-ul pentru un query de selectare.

StringBuilder stringBuilder=new StringBuilder();

stringBuilder.append("SELECT \* FROM ");

stringBuilder.append(type.getSimpleName());

stringBuilder.append(" WHERE ");

stringBuilder.append(field);

stringBuilder.append("=?");

return String.valueOf(stringBuilder);

Clasa AbstractDAO este utilizată pentru a realiza operațiile necesare asupra bazei de date.

Câmpuri :

* type – reprezintă clasa pentru care se realizează operațiile respective

Metode :

* insert
* updateById
* deleteById
* selectById
* selectAll
* createObjects

Clasa ClientDAO este utilizată pentru a manipula clienți din baza de date.

Clasa ProductDAO este utilizată pentru a manipula produsele din baza de date.

Clasa CustomersOrderDAO este utilizată pentru a manipula comenzile din baza de date.

Clasa AddressValidator este utilizată pentru a valida adresa unui client.

Interfata Validator este utilizată pentru a crea mai multi validat-ori pentru date.

Metode :

* validate

Câmpuri :

* ADDRESS\_PATTERN

Metode :

* validate - metoda verifică dacă în adresă există alte caractere decât cifre, litere, punct sau linie.

Pattern pattern=Pattern.compile(ADDRESS\_PATTERN);

if(pattern.matcher(object.getAddress()).matches())

{

throw new IllegalArgumentException("Address is not valid!");

}

Clasa EmailValidator este utilizată pentru a valida email-ul unui client.

Câmpuri :

* EMAIL\_PATTERN

Metode :

* validate

Clasa NameAndBrandValidator este utilizată pentru a valida numele și firma unui produs.

Câmpuri :

* NAME\_PATTERN

Metode :

* validate - metoda verifică dacă produsul are în denumire sau în denumirea firmei altceva decât litere sau cifre.

Pattern pattern=Pattern.compile(NAME\_PATTERN);

if(pattern.matcher(object.getName()).matches() || object.getName().equals(""))

{

throw new IllegalArgumentException("Name is not valid!");

}

if(pattern.matcher(object.getBrand()).matches() || object.getBrand().equals(""))

{

throw new IllegalArgumentException("Brand is not valid!");

}

Clasa NameValidator este utilizată pentru a valida numele și prenumele unui client.

Câmpuri :

* NAME\_PATTERN

Metode :

* validate

Clasa OrderQuantityValidator este utilizată pentru a valida cantitea unei comenzi.

Metode :

* validate

Clasa PhoneNumberValidator este utilizată pentru a valida numărul de telefon al unui client.

Metode :

* validate - metoda verifică corectitudinea numărului de telefon al unui client, acesta fiind necesar de a conține numai cifre, de a începe cu 07 și de a avea o lungime de 10 caractere.

Pattern pattern=Pattern.compile(PHONE\_NUMBER\_PATTERN);

if(!pattern.matcher(object.getPhoneNumber()).matches())

{

throw new IllegalArgumentException("Phone number is not valid!");

}

Clasa PriceValidator este utilizată pentru a valida prețul unui produs.

Metode :

* validate

Clasa ProductQuantityValidator este utilizată pentru a valida cantitatea unui produs din stoc.

Metode :

* validate

Clasa ProductLogic este clasa utilizată pentru a implementa logica pentru obiecte de tipul Product.

Câmpuri :

* validators
* productDAO

Metode :

* productValidate
* insertProduct
* updateById - Metoda este utilizată pentru a actualiza un produs, verificându-se inainte dacă modificarile nu ar incălca corectitudinea datelor.

Product product=productDAO.selectById(id);

try {

Field classField=product.getClass().getDeclaredField(field);

classField.setAccessible(true);

classField.set(product,value);

this.productValidate(product);

} catch (IllegalAccessException | NoSuchFieldException e) {

e.printStackTrace();

}

int result=productDAO.updateById(field,value,id);

if(result==0)

throw new IllegalArgumentException("Product with id:"+id+" can't be updated!");

return result;

* deleteById
* selectById
* selectAll

Clasa ClientLogic este utilizată pentru a implementa logica pentru obiecte de tip Client.

Câmpuri :

* validators
* clientDAO

Metode :

* clientValidate
* insertClient
* updateById - metoda este utilizată pentru a actualiza un client, verificându-se înainte dacă modificările nu ar încălca corectitudinea datelor.

Client client=clientDAO.selectById(id);

try {

Field classField=client.getClass().getDeclaredField(field);

classField.setAccessible(true);

classField.set(client,value);

this.clientValidate(client);

} catch (IllegalAccessException | NoSuchFieldException e) {

e.printStackTrace();

}

int result=clientDAO.updateById(field,value,id);

if(result==0)

throw new IllegalArgumentException("Client with id:"+id+" can't be updated!");

return result;

* deleteById
* selectById
* selectAll

Clasa CustomersOrderLogic este utilizată pentru a implementa logica pentru obiecte de tip CustomersOrder.

Câmpuri :

* validator
* clientDAO
* productDAO
* customersDAO

Metode :

* insertOrder - metoda insearează o comandă nouă dacă stocul de produse este suficiet.

this.orderValidate(order);

if(order.getQuantity()>productDAO.selectById(order.getProductId()).getQuantity())

throw new IllegalArgumentException("Not enough stock!");

int result=customersOrderDAO.insert(order);

if(result==0)

throw new IllegalArgumentException("Order with id:"+order.getId()+" can't be inserted!");

return result;

* updateById
* deleteById
* selectById - metoda care returnează o comandă complete pentru un anume id specificat. Aceasta va selecta un customersOrder, și în cazul în care acesta exista, va crea un obiect de tip CompleteOrder, care va conține și informațiile despre client si produs.

CustomersOrder customersOrder=customersOrderDAO.selectById(id);

if(customersOrder==null)

throw new NoSuchElementException("Order with id:"+id+" can't be found!");

CompleteOrder completeOrder=new CompleteOrder(customersOrder,clientDAO.selectById(customersOrder.getClientId()),productDAO.selectById(customersOrder.getProductId()));

return completeOrder;

* selectAllOrders - metoda selectează toate comenziile complete, fiind selectate toate obiectele de tipul CustomersOrder, fiind apoi selectați clientul si produsul.

List<CompleteOrder> completeOrders=new ArrayList<>();

List<CustomersOrder> customersOrders=customersOrderDAO.selectAll();

for(CustomersOrder order: customersOrders)

{

completeOrders.add(new CompleteOrder(order,clientDAO.selectById(order.getClientId()),productDAO.selectById(order.getProductId())));

}

return completeOrders;

Clasa View este utilizată pentru a implementa fereastra principală. La început vor fi cele trei butoane pentru selectarea interfețelor pentru clienți, produse și comenzi, după care aceastea vor fi eliminate, fiind adăugate elementele specifice fiecărei categorii.

Câmpuri :

* clientsButton
* productsButton
* ordersButton
* clientsTable
* produtcsTable
* ordersTable
* backButton
* insertButton
* insertButton
* updateButton
* deleteButton
* findButton
* makeOrderButton
* contet
* tablesButtonPane
* operationsButtonPane
* backButtonPane
* tablePane
* clientsScrollPane
* productsScrollPane
* ordersScrollPane
* firstNameTextField
* lastNameTextField
* addressTextField
* emailTextField
* phoneNumberTextField
* firstNameLabel
* lastNameLabel
* addressLabel
* emailLabel
* phoneNumberLabel
* nameTextField
* brandTextField
* priceTextField
* quantityTextField
* nameLabel
* brandLabel
* priceLabel
* quantityLabel

Metode :

* addClientsButtonListener
* addProductsButtonListener
* addOrdersButtonListener
* addBackButtonListener
* addInsertButtonListener
* addDeleteButtonListener
* addUpdateButtonListener
* addFindButtonListener
* addMakeOrderButtonListener

Clasa OrderView este utilizată pentru a implementa fereastra în care utilizatorul alege comanda, produsul și cantitatea pentru noua comandă.

Câmpuri :

* quantityLabel
* quantityTextField
* okButton
* content
* clientsTable
* productsTable
* clientsScrollPane
* productsScrollPane
* view

Metode :

* addOkButtonListener

Clasa Controller este utilizată pentru a lega clasele din model cu interfața grafică.

Câmpuri :

* view
* orderView
* clientLogic
* productLogic
* customersOrderLogic

Clasa ClientsButtonListener este utilizată pentru a accesa interfața corespunzătoare clienților.

Câmpuri :

* view

Metode :

* actionPerformed - când butonul este apăsat se vor elimina componentele din scenă, urmând a fi adăugat tabelul cu clienți, butoanele cu operațiile posibile, cat și etichetele și câmpurile pentru adăugarea datelor.

view.getContentPane().removeAll();

view.operationsButtonPane.removeAll();

view.operationsButtonPane.add(view.insertButton);

view.operationsButtonPane.add(view.updateButton);

view.operationsButtonPane.add(view.deleteButton);

view.operationsButtonPane.add(view.findButton);

view.getContentPane().add(view.operationsButtonPane, BorderLayout.PAGE\_START);

view.tablePane.removeAll();

view.tablePane.add(view.firstNameLabel);

view.tablePane.add(view.firstNameTextField);

view.tablePane.add(view.lastNameLabel);

view.tablePane.add(view.lastNameTextField);

view.tablePane.add(view.addressLabel);

view.tablePane.add(view.addressTextField);

view.tablePane.add(view.emailLabel);

view.tablePane.add(view.emailTextField);

view.tablePane.add(view.phoneNumberLabel);

view.tablePane.add(view.phoneNumberTextField);

view.tablePane.add(view.clientsScrollPane);

view.getContentPane().add(view.tablePane,BorderLayout.CENTER);

view.getContentPane().add(view.backButtonPane,BorderLayout.PAGE\_END);

view.repaint();

view.revalidate();

Clasa ProductsButtonListener este utilizată pentru a oferi acces la interfața corespunzatoare produselor.

Câmpuri :

* view

Metode :

* actionPerformed - când butonul este apăsat, toate componentele sunt eliminate, fiind adăugate tabelul cu produsele, butoanele cu operațiile posibile, cat și etichetele și câmpurile pentru adăugarea datelor.

view.getContentPane().removeAll();

view.operationsButtonPane.removeAll();

view.operationsButtonPane.add(view.insertButton);

view.operationsButtonPane.add(view.updateButton);

view.operationsButtonPane.add(view.deleteButton);

view.operationsButtonPane.add(view.findButton);

view.getContentPane().add(view.operationsButtonPane, BorderLayout.PAGE\_START);

view.tablePane.removeAll();

view.tablePane.add(view.nameLabel);

view.tablePane.add(view.nameTextField);

view.tablePane.add(view.brandLabel);

view.tablePane.add(view.brandTextField);

view.tablePane.add(view.priceLabel);

view.tablePane.add(view.priceTextField);

view.tablePane.add(view.quantityLabel);

view.tablePane.add(view.quantityTextField);

view.tablePane.add(view.productsScrollPane);

view.getContentPane().add(view.tablePane,BorderLayout.CENTER);

view.getContentPane().add(view.backButtonPane,BorderLayout.PAGE\_END);

view.repaint();

view.revalidate();

Clasa OrdersActionListener este utilizată pentru a accesa interfața corespunzătoare comenziilor.

Câmpuri :

* view

Metode :

* actionPerformed - când butonul este apăsat, toate componentele sunt eliminate, fiind adăugate tabelul cu comenzile și butonul de creare al comenzilor.

view.getContentPane().removeAll();

view.operationsButtonPane.removeAll();

view.operationsButtonPane.add(view.makeOrderButton);

view.getContentPane().add(view.operationsButtonPane, BorderLayout.PAGE\_START);

view.tablePane.removeAll();

view.tablePane.add(view.ordersScrollPane);

view.getContentPane().add(view.tablePane,BorderLayout.CENTER);

view.getContentPane().add(view.backButtonPane,BorderLayout.PAGE\_END);

view.repaint();

view.revalidate();

Clasa BackButtonListener este utilizată pentru a implementa funcționalitatea butonului de back din interfață.

Câmpuri :

* view
* clientLogic
* productLogic

Metode :

* actionPerfomed - când butonul este apăsat, toate tabelele vor fi actualizate, iar din fereastră se vor elemina toate elementele, fiind adăugate butoanele pentru selectarea categoriei dorite.

((DefaultTableModel)view.clientsTable.getModel()).setColumnCount(0);

((DefaultTableModel) view.clientsTable.getModel()).setRowCount(0);

TablePopulator.populateTable(view.clientsTable, clientLogic.selectAll().parallelStream().map(x -> (Object) x).collect(Collectors.toList()));

((DefaultTableModel)view.productsTable.getModel()).setColumnCount(0);

((DefaultTableModel) view.productsTable.getModel()).setRowCount(0);

TablePopulator.populateTable(view.productsTable, productLogic.selectAll().parallelStream().map(x -> (Object) x).collect(Collectors.toList()));

view.getContentPane().removeAll();

view.getContentPane().add(view.tablesButtonPane, BorderLayout.PAGE\_START);

view.repaint();

view.revalidate();

Clasa OkButtonListener este utilizată pentru a implementa funcționalitatea butonului de ok.

Câmpuri :

* orderView
* view
* clientLogic
* productLogic
* customersOrderLogic

Metode :

* actionPerformed - când butonul este apăsat se verifică dacă au fost selectați un client și un produs, iar cantitatea de produse este introdusă, urmând ca apoi o comandă sa fie adăugataă, tabelul de comenzi sa fie reactualizat și o factură sa fie generate. În caz de eroare se afișează un mesaj.

try {

if(orderView.clientsTable.getSelectedRow()==-1)

throw new IllegalArgumentException("No client selected!");

int clientId = (Integer) orderView.clientsTable.getValueAt(orderView.clientsTable.getSelectedRow(), 0);

if(orderView.productsTable.getSelectedRow()==-1)

throw new IllegalArgumentException("No product selected!");

int productId = (Integer) orderView.productsTable.getValueAt(orderView.productsTable.getSelectedRow(), 0);

if(orderView.quantityTextField.getText().equals(""))

throw new IllegalArgumentException("No quantity entered!");

customersOrderLogic.insertOrder(new CustomersOrder(Integer.parseInt(orderView.quantityTextField.getText()), clientId, productId));

((DefaultTableModel) view.ordersTable.getModel()).setColumnCount(0);

((DefaultTableModel) view.ordersTable.getModel()).setRowCount(0);

TablePopulator.populateTable(view.ordersTable, customersOrderLogic.selectAllOrders().parallelStream().map(x -> (Object) x).collect(Collectors.toList()));

productLogic.updateById("quantity",productLogic.selectById(productId).getQuantity()-Integer.parseInt(orderView.quantityTextField.getText()),productId);

((DefaultTableModel)view.productsTable.getModel()).setColumnCount(0);

((DefaultTableModel) view.productsTable.getModel()).setRowCount(0);

TablePopulator.populateTable(view.productsTable, productLogic.selectAll().parallelStream().map(x -> (Object) x).collect(Collectors.toList()));

((DefaultTableModel)orderView.productsTable.getModel()).setColumnCount(0);

((DefaultTableModel) orderView.productsTable.getModel()).setRowCount(0);

TablePopulator.populateTable(orderView.productsTable, productLogic.selectAll().parallelStream().map(x -> (Object) x).collect(Collectors.toList()));

orderView.quantityTextField.setText("");

orderView.setVisible(false);

BillMaker.makeBill(customersOrderLogic.selectById((Integer)view.ordersTable.getValueAt(view.ordersTable.getRowCount()-1,0)));

} catch (IllegalArgumentException exception) {

JOptionPane.showMessageDialog(orderView,exception.getMessage());

}

}

Clasa InsertButtonListener este utilizată pentru a implementa funcționalitate butonului de insert.

Metode :

* actionPerformed

Clasa DeleteButtonListener este utilizată pentru a implementa funcționalitateabutonului de delete.

Metode :

* actionPerformed

Clasa UpdateButtonListener este utilizată pentru a implementa funcționalitatea butonului de update.

Metode :

* actionPerformed

Clasa FindButtonListener este utilizată pentru a implementa funcționalitatea butonului de find.

Metode :

* actionPerformed

Clasa MakeOrderButtonListener este utilizată pentru a implementa funcționalitatea butonului de make order.

Metode :

* actionPerformed

Clasa Main este utilizată pentru a rula programul.

Metode :

* main

1. Rezultate

Aplicația a fost testată conform cerințelor specificate și aceasta îndeplinește cerințele specificate în tema de laborator.

1. Concluzii

În urma acestei teme s-au desprins două cunoștințe noi importante, și anume, lucrul cu baze de date și realizarea aplicațiilor folosind arhitectura layered. Ca și dezvoltări ulterioare, se pot introduce noi tipuri de interfețe și se poate implementa noi mecanisme, precum căutarea folosind diverse criterii sau comandarea mai multor produse într-o singură comandă.

1. Bibliografie
2. Cioara, Tudor. ASSIGMENT\_3\_SUPPORT\_PRESENTATION. April 2021, <https://teams.microsoft.com/l/file/615BFD47-ECF2-4D39-BF9F-CDC591B3048D?tenantId=a6eb79fa-c4a9-4cce-818d-b85274d15305&fileType=pdf&objectUrl=https%3A%2F%2Fdidatec.sharepoint.com%2Fsites%2FTP274%2FClass%20Materials%2FLaborator%2FTema%203%2FASSIGNMENT_3_SUPPORT_PRESENTATION.pdf&baseUrl=https%3A%2F%2Fdidatec.sharepoint.com%2Fsites%2FTP274&serviceName=teams&threadId=19:4fed5c2e262a4057b1dabca3437d3a19@thread.tacv2&groupId=adfab837-689b-4a06-946c-77ee180b149a>. PowerPoint Presentation.
3. MySQL Workbench Manual. Data Export and Import Wizard. May 2021, <https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html>