AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Projekt – Bazy danych II

Wiktor Tarsa, Kamil Wnęk

Spis treści

Informacje ogólne

Aplikacja bazodanowa umożliwia komunikację z bazą danych Northwind obsługiwaną poprzez MySql za pomocą linii komend. Możliwości jakie udostępnia aplikacja to dodanie zamówienia, wygenerowanie przykładowych raportów, oraz generowaniu faktur.

System składa się z dwóch elementów:

- Bazy danych Northwind: https://github.com/dalers/mywind
- Aplikacji stanowiącej główny element systemu, napisanej w języku Java przy użyciu Frameworka Hibernate.

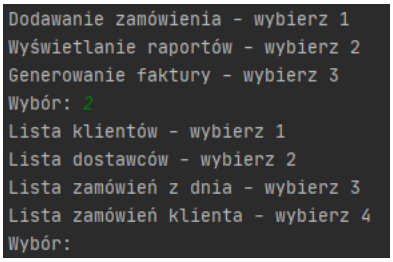
Dostęp do aplikacji możliwy jest przez wirtualną maszynę Javy po uruchomieniu serwera bazy danych z bazą Northwind. Aplikacja posiada minimalistyczne API.

Tryb pracy

Dostęp do pracy nie jest ograniczony żadnym loginem ani hasłem. Wyróżnia się jeden tryb pracy:

1) Tryb użytkownika – umożliwia dodawanie zamówien, generowanie raportów, oraz faktur.

Wygląd interfejsu



Rys. 1. Interfejs aplikacji.

Po uruchomieniu aplikacji wyświetlany jest interfejs linii komend. Użytkownik wybierając opcje cyframi zatwierdza ją kliknięciem klawisza „Enter”. Na rysunku przedstawiony wybór wyświetlania raportów, gdyż zawiera on podmenu z dalszym wyborem analogicznym do poprzedniego.

Tryb użytkownika

Dodawanie zamówienia

```
Dodawanie zamówienia - wybierz 1
Wyświetlanie raportów - wybierz 2
Generowanie faktury - wybierz 3
Wybór: 1
Podaj imię: Anna
Podaj nazwisko: Bedecs
id: 1 nazwa: Northwind Traders Chai cena: 13.5 ilość w opakowaniu: 10 boxes x 20 bags
id: 3 nazwa: Northwind Traders Syrup cena: 7.5 ilość w opakowaniu: 12 - 550 ml bottles
id: 4 nazwa: Northwind Traders Cajun Seasoning cena: 16.5 ilość w opakowaniu: 48 - 6 oz jars
id: 5 nazwa: Northwind Traders Olive Oil cena: 16.0125 ilość w opakowaniu: 36 boxes
id: 6 nazwa: Northwind Traders Boysenberry Spread cena: 18.75 ilość w opakowaniu: 12 - 8 oz jars
id: 7 nazwa: Northwind Traders Dried Pears cena: 22.5 ilość w opakowaniu: 12 - 1 lb pkgs.
id: 8 nazwa: Northwind Traders Curry Sauce cena: 30.0 ilość w opakowaniu: 12 - 12 oz jars
id: 14 nazwa: Northwind Traders Walnuts cena: 17.4375 ilość w opakowaniu: 40 - 100 g pkgs.
id: 17 nazwa: Northwind Traders Fruit Cocktail cena: 29.25 ilość w opakowaniu: 15.25 OZ
id: 19 nazwa: Northwind Traders Chocolate Biscuits Mix cena: 6.9 ilość w opakowaniu: 10 boxes x 12 pieces
id: 20 nazwa: Northwind Traders Marmalade cena: 60.75 ilość w opakowaniu: 30 gift boxes
id: 21 nazwa: Northwind Traders Scones cena: 7.5 ilość w opakowaniu: 24 pkgs. x 4 pieces
id: 34 nazwa: Northwind Traders Beer cena: 10.5 ilość w opakowaniu: 24 - 12 oz bottles
id: 40 nazwa: Northwind Traders Crab Meat cena: 13.8 ilość w opakowaniu: 24 - 4 oz tins
id: 41 nazwa: Northwind Traders Clam Chowder cena: 7.2375 ilość w opakowaniu: 12 - 12 oz cans
id: 43 nazwa: Northwind Traders Coffee cena: 34.5 ilość w opakowaniu: 16 - 500 g tins
id: 48 nazwa: Northwind Traders Chocolate cena: 9.5625 ilość w opakowaniu: 10 pkgs
id: 51 nazwa: Northwind Traders Dried Apples cena: 39.75 ilość w opakowaniu: 50 - 300 g pkgs.
id: 52 nazwa: Northwind Traders Long Grain Rice cena: 5.25 ilość w opakowaniu: 16 - 2 kg boxes
id: 56 nazwa: Northwind Traders Gnocchi cena: 28.5 ilość w opakowaniu: 24 - 250 g pkgs.
id: 57 nazwa: Northwind Traders Ravioli cena: 14.625 ilość w opakowaniu: 24 - 250 g pkgs.
id: 65 nazwa: Northwind Traders Hot Pepper Sauce cena: 15.7875 ilość w opakowaniu: 32 - 8 oz bottles
id: 66 nazwa: Northwind Traders Tomato Sauce cena: 12.75 ilość w opakowaniu: 24 - 8 oz jars
id: 72 nazwa: Northwind Traders Mozzarella cena: 26.1 ilość w opakowaniu: 24 - 200 g pkgs.
id: 74 nazwa: Northwind Traders Almonds cena: 7.5 ilość w opakowaniu: 5 kg pkg.
id: 77 nazwa: Northwind Traders Mustard cena: 9.75 ilość w opakowaniu: 12 boxes
id: 80 nazwa: Northwind Traders Dried Plums cena: 3.0 ilość w opakowaniu: 1 lb bag
id: 81 nazwa: Northwind Traders Green Tea cena: 2.0 ilość w opakowaniu: 20 bags per box
id: 82 nazwa: Northwind Traders Granola cena: 2.0 ilość w opakowaniu: null
id: 83 nazwa: Northwind Traders Potato Chips cena: 0.5 ilość w opakowaniu: null
id: 85 nazwa: Northwind Traders Brownie Mix cena: 9.0 ilość w opakowaniu: 3 boxes
id: 86 nazwa: Northwind Traders Cake Mix cena: 10.5 ilość w opakowaniu: 4 boxes
id: 87 nazwa: Northwind Traders Tea cena: 2.0 ilość w opakowaniu: 100 count per box
id: 88 nazwa: Northwind Traders Pears cena: 1.0 ilość w opakowaniu: 15.25 OZ
id: 89 nazwa: Northwind Traders Peaches cena: 1.0 ilość w opakowaniu: 15.25 OZ
id: 90 nazwa: Northwind Traders Pineapple cena: 1.0 ilość w opakowaniu: 15.25 OZ
id: 91 nazwa: Northwind Traders Cherry Pie Filling cena: 1.0 ilość w opakowaniu: 15.25 OZ
id: 92 nazwa: Northwind Traders Green Beans cena: 1.0 ilość w opakowaniu: 14.5 OZ
id: 93 nazwa: Northwind Traders Corn cena: 1.0 ilość w opakowaniu: 14.5 OZ
id: 94 nazwa: Northwind Traders Peas cena: 1.0 ilość w opakowaniu: 14.5 OZ
id: 95 nazwa: Northwind Traders Tuna Fish cena: 0.5 ilość w opakowaniu: 5 oz
id: 96 nazwa: Northwind Traders Smoked Salmon cena: 2.0 ilość w opakowaniu: 5 oz
id: 97 nazwa: Northwind Traders Hot Cereal cena: 3.0 ilość w opakowaniu: null
id: 98 nazwa: Northwind Traders Vegetable Soup cena: 1.0 ilość w opakowaniu: null
id: 99 nazwa: Northwind Traders Chicken Soup cena: 1.0 ilość w opakowaniu: null
Podaj id produktu: 1
Podaj ilość jednostek: 2
Podaj id produktu: 3
Podaj ilość jednostek: 2
Podaj id produktu: 4
Podaj ilość jednostek: 2
Podaj id produktu:
Podaj ilość jednostek:
Do zamówienia zostaną dodane pozycje:
Northwind Traders Chai ilość: 2
Northwind Traders Syrup ilość: 2
Northwind Traders Cajun Seasoning ilość: 2
```

Rys. 2. Dodanie zamówienia dla Anny Bedecs: 2x Traders Syrup.

Generowanie przykładowego raportu

```
Dodawanie zamówienia - wybierz 1
Wyświetlanie raportów - wybierz 2
Generowanie faktury - wybierz 3
Wybór: 2
Lista klientów - wybierz 1
Lista dostawców - wybierz 2
Lista zamówień z dnia - wybierz 3
Lista zamówień klienta - wybierz 4
Wybór: 2
id: 1       company: Supplier A         name: Elizabeth A.          lastname: Andersen          address: null       null
id: 2       company: Supplier B         name: Cornelia          lastname: Weiler            address: null       null
id: 3       company: Supplier C         name: Madeleine         lastname: Kelley            address: null       null
id: 4       company: Supplier D         name: Naoki         lastname: Sato          address: null       null        null
id: 5       company: Supplier E         name: Amaya         lastname: Hernandez-Echevarria          address: null
id: 6       company: Supplier F         name: Satomi            lastname: Hayakawa          address: null       null
id: 7       company: Supplier G         name: Stuart            lastname: Glasson           address: null       null
id: 8       company: Supplier H         name: Bryn Paul         lastname: Dunton            address: null       null
id: 9       company: Supplier I         name: Mikael            lastname: Sandberg          address: null       null
id: 10      company: Supplier J         name: Luis          lastname: Sousa         address: null       null        null

Process finished with exit code 0
```

Rys. 3. Generowanie raportu – lista dostawców.

Generowanie faktury

```
Dodawanie zamówienia - wybierz 1
Wyświetlanie raportów - wybierz 2
Generowanie faktury - wybierz 3
Wybór: 3
id: 1       name: Anna          lastname: Bedecs           address: 123 1st Street     Seattle     99999
id: 2       name: Antonio           lastname: Gratacos Solsona          address: 123 2nd Street     Boston      99999
id: 3       name: Thomas            lastname: Axen          address: 123 3rd Street     Los Angelas     99999
id: 4       name: Christina         lastname: Lee           address: 123 4th Street     New York        99999
id: 5       name: Martin            lastname: O'Donnell         address: 123 5th Street     Minneapolis     99999
id: 6       name: Francisco         lastname: Pérez-Olaeta          address: 123 6th Street     Milwaukee       99999
id: 7       name: Ming-Yang         lastname: Xie           address: 123 7th Street     Boise       99999
id: 8       name: Elizabeth         lastname: Andersen          address: 123 8th Street     Portland        99999
id: 9       name: Sven          lastname: Mortensen         address: 123 9th Street     Salt Lake City      99999
id: 10      name: Roland            lastname: Wacker            address: 123 10th Street        Chicago     99999
id: 11      name: Peter         lastname: Krschne           address: 123 11th Street        Miami       99999
id: 12      name: John          lastname: Edwards           address: 123 12th Street        Las Vegas       99999
id: 13      name: Andre         lastname: Ludick            address: 456 13th Street        Memphis     99999
id: 14      name: Carlos            lastname: Grilo         address: 456 14th Street        Denver      99999
id: 15      name: Helena            lastname: Kupkova           address: 456 15th Street        Honolulu        99999
id: 16      name: Daniel            lastname: Goldschmidt           address: 456 16th Street        San Francisco
id: 17      name: Jean Philippe         lastname: Bagel         address: 456 17th Street        Seattle     99999
id: 18      name: Catherine         lastname: Autier Miconi         address: 456 18th Street        Boston      99999
id: 19      name: Alexander         lastname: Eggerer           address: 789 19th Street        Los Angelas     99999
id: 20      name: George            lastname: Li            address: 789 20th Street        New York        99999
id: 21      name: Bernard           lastname: Tham          address: 789 21th Street        Minneapolis     99999
id: 22      name: Luciana           lastname: Ramos         address: 789 22th Street        Milwaukee       99999
id: 23      name: Michael           lastname: Entin         address: 789 23th Street        Portland        99999
id: 24      name: Jonas         lastname: Hasselberg            address: 789 24th Street        Salt Lake City      999
id: 25      name: John          lastname: Rodman            address: 789 25th Street        Chicago     99999
id: 26      name: Run           lastname: Liu           address: 789 26th Street        Miami       99999
id: 27      name: Karen         lastname: Toh           address: 789 27th Street        Las Vegas       99999
id: 28      name: Amritansh         lastname: Raghav            address: 789 28th Street        Memphis     99999
id: 29      name: Soo Jung          lastname: Lee           address: 789 29th Street        Denver      99999
Podaj ID klienta: 1
id: 44      customer: Anna Bedecs       order_date: 2006-03-24 00:00:00.0       ship_name: Anna Bedecs
id: 71      customer: Anna Bedecs       order_date: 2006-05-24 00:00:00.0       ship_name: Anna Bedecs
id: 82      customer: Anna Bedecs       order_date: 2020-05-27 11:19:13.0       ship_name: null
id: 83      customer: Anna Bedecs       order_date: 2020-05-27 11:33:04.0       ship_name: null
Podaj ID zamówienia: 83
product_name: Northwind Traders Chai        category: Beverages     quantity: 2     unit_price: 13.5
product_name: Northwind Traders Syrup       category: Condiments        quantity: 2     unit_price: 7.5
product_name: Northwind Traders Cajun Seasoning     category: Condiments        quantity: 2     unit_price: 16.5

Process finished with exit code 0
```

Rys. 4. Generowanie faktury wykonanego wczesniej zamówienia.

Kod aplikacji

```
@Entity
@Table(name = "customers")
public class Customers {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Integer id;
   private String company;
   private String first_name;
   private String last_name;
   private String email_address;
   private String job_title;
   private String business_phone;
   private String home_phone;
   private String mobile_phone;
   private String fax_number;
   private String address;
   private String city;
   private String state_province;
   private String zip_postal_code;
   private String country_region;
   private String web_page;
   private String notes;

   public Customers(){}

   public Customers(String first_name, String last_name, String address, String city, String zip_postal_code){
         this.first_name = first_name;
         this.last_name = last_name;
         this.address = address;
         this.city = city;
         this.zip_postal_code = zip_postal_code;
   }


   public String getFirstName() { return this.first_name; }
   public String getLastName() { return this.last_name; }
   public String getAddress() { return this.address; }
   public String getCity() { return this.city; }
   public String getPostalCode() { return this.zip_postal_code; }

   /**
    * Zwraca klienta, którego imię i nazwisko jest podane w parametrach.
    * Jeżeli klient o podanych danych nie istnieje zwraca null.
    * @param first_name
    * @param last_name
    * @param session
    * @return
    */
   public static Customers getID(String first_name, String last_name, Session session){
         Customers customer = null;
         Transaction tx = session.beginTransaction();
         String hql = "FROM Customers";
         Query query = session.createQuery(hql);
         List<Customers> customers = query.list();
         for(Customers c: customers){
            if(first_name.equals(c.getFirstName()) && last_name.equals(c.getLastName()))
               customer = c;
         }
         tx.commit();
         return customer;
   }
   /**
    * Wyświetla wszystkich klientów istniejących w tabeli customers.
    * @param session
```

```
    */
    public static void showAllCustomers(Session session) {
        Transaction tx = session.beginTransaction();
        List<Object[]> customers = session.createQuery("SELECT id, first_name, last_name, address, city, zip_postal_code FROM
Customers").getResultList();

        for (Object[] c : customers) {
            System.out.println("id: " + c[0] + "\t\t" + "name: " + c[1] + "\t\t\t" + "lastname: " + c[2] + "\t\t\t" +  "address: " + c[3] + "\t\t" +
c[4] + "\t\t" + c[5]);
        }
        tx.commit();
    }
}
```

Klasa Customers

```java
@Entity
@Table(name = "employees")
public class Employees {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String company;
    private String last_name;
    private String first_name;
    private String email_address;
    private String job_title;
    private String business_phone;
    private String home_phone;
    private String mobile_phone;
    private String fax_number;
    private String address;
    private String city;
    private String state_province;
    private String zip_postal_code;
    private String country_region;
    private String web_page;
    private String notes;

    public Employees(){}

    /**
     * Zwraca losowego pracownika z tabeli Employees.
     * @param session
     * @return
     */
    public static Employees getEmployee(Session session){
          String hql = "FROM Employees";
         Query query = session.createQuery(hql);
          List<Employees> employees = query.list();
          Random generator = new Random();
          return employees.get(generator.nextInt(employees.size()));
      }
}
```

Klasa Employees

```
@Entity
@Table(name = "invoices")
public class Invoices {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private int order_id;
    @Temporal(TemporalType.TIMESTAMP)
    private Date invoice_date;
    @Temporal(TemporalType.TIMESTAMP)
    private Date due_date;
    private int tax;
    private int shipping;
    private int amount_due;

    public Invoices(){}
}
```

Klasa Invoices.

```java
@Entity
@Table(name = "orders")
public class Orders {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @Column(insertable = false, updatable = false)
    private int employee_id;
    @Column(insertable = false, updatable = false)
    private int customer_id;
    @Temporal(TemporalType.TIMESTAMP)
    private Date order_date;
    @Temporal(TemporalType.TIMESTAMP)
    private Date shipped_date;
    private String ship_name;
    private String ship_address;
    private String ship_city;
    private String ship_state_province;
    private String ship_zip_postal_code;
    private String ship_country_region;
    private double shipping_fee;
    private double taxes;
    private String payment_type;
    @Temporal(TemporalType.TIMESTAMP)
    private Date paid_date;
    private String notes;
    private double tax_rate;
    private int status_id;

    @ManyToOne
    private Employees employee;

    @ManyToOne
    private Customers customer;

    public Orders(){}
    public Orders(Customers customer, Date order_date, Employees employee){
        this.order_date = order_date;
        this.employee = employee;
        this.customer = customer;
        this.ship_address = customer.getAddress();
        this.ship_city = customer.getCity();
        this.ship_zip_postal_code = customer.getPostalCode();
    }
    public int getOrderID(){
        return this.id;
    }

    /**
     * Wyświetla zamówienia, które zostały złożone w podanym dniu.
     * @param year
     * @param month
     * @param day
     * @param session
     */
    public static void showOrders(int year, int month, int day, Session session){
        Transaction tx = session.beginTransaction();
        List<Objects[]> orders = session.createQuery("SELECT o.id, c.first_name, c.last_name, o.order_date, o.ship_name FROM Orders o
JOIN Customers c on o.customer_id = c.id").getResultList();

        for(Object[] o: orders){
            Calendar calendar = Calendar.getInstance();
            calendar.setTime((Date)o[3]);
            if(year == calendar.get(Calendar.YEAR) && (month-1) == calendar.get(Calendar.MONTH) && day ==
calendar.get(Calendar.DAY_OF_MONTH))
                System.out.println("id: " + o[0] + "\t\t" + "customer: " + o[1] + " " + o[2] + "\t\t" +  "order_date: " + o[3] + "\t\t" +
"ship_name: " + o[4]);
        }
        tx.commit();
    }
```

```
    /**
     * Wyświetla wszystkie zamówienia osoby o podanym id.
     * @param customer_id
     * @param session
     */
    public static void showOrders(int customer_id, Session session){
        Transaction tx = session.beginTransaction();
        List<Object[]> orders = session.createQuery("SELECT o.id, c.id, c.first_name, c.last_name, o.order_date, o.ship_name FROM
Orders o JOIN Customers c ON o.customer_id = c.id").getResultList();

        for(Object[] o: orders){
            if((Integer)o[1] == customer_id){
                System.out.println("id: " + o[0] + "\t\t" + "customer: " + o[2] + " " + o[3] + "\t\t" + "order_date: " + o[4] + "\t\t" +
"ship_name: " + o[5]);
            }
        }
        tx.commit();
    }

}
```

Klasa Orders.

```java
@Entity
@Table(name = "products")
public class Products {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String supplier_ids;
    private String product_code;
    private String product_name;
    private String description;
    private double standard_cost;
    private double list_price;
    private int reorder_level;
    private String quantity_per_unit;
    private int discontinued;
    private String category;

    public Products(){}

    public String getProductName(){
        return this.product_name;
    }

    public double getStandardCost() { return this.standard_cost; }

    public int getProductID() { return this.id; }

    public String getQuantityPerUnit() { return this.quantity_per_unit; }

    /**
     * Wypisuje id, nazwę, cenę oraz informację na temat ilości jednostek w opakowaniu dla każdego produktu
     * znajdującego się w tabeli Products.
     * @param session
     */
    public static void listAllProducts(Session session){
        Transaction tx = session.beginTransaction();
        String hql = "FROM Products";
        Query query = session.createQuery(hql);
        List<Products> allProducts = query.list();
        for(Products product: allProducts){
            System.out.println("id: " + product.getProductID() +
                    " nazwa: " + product.getProductName() +
                    " cena: " + product.getStandardCost() +
                    " ilość w opakowaniu: " + product.getQuantityPerUnit());
        }
        tx.commit();
    }

    /**
     * Zwraca produkt o podanym id. Jeżeli tabela nie zawiera produktu o podanym id zwraca null.
     * @param session
     * @param id
     * @return
     */
    public static Products getProduct(Session session, int id){
        Products product = null;
        Transaction tx = session.beginTransaction();
        String hql = "FROM Products";
        Query query = session.createQuery(hql);
        List<Products> allProducts = query.list();
        for(Products p: allProducts){
            if(p.id == id) product = p;
        }
        tx.commit();
        return product;
    }

}
```

Klasa Products.

```
@Entity
@Table(name = "suppliers")
public class Suppliers {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String company;
    private String last_name;
    private String first_name;
    private String email_address;
    private String job_title;
    private String business_phone;
    private String home_phone;
    private String mobile_phone;
    private String fax_number;
    private String address;
    private String city;
    private String state_province;
    private String zip_postal_code;
    private String country_region;
    private String web_page;
    private String notes;

    public Suppliers(){}

    /**
     * Wyświetla dostawców istniejących w tabeli suppliers.
     * @param session
     */
    public static void showAllSuppliers(Session session) {
        Transaction tx = session.beginTransaction();
        List<Object[]> suppliers = session.createQuery("SELECT id, company, first_name, last_name, address, city, zip_postal_code FROM
Suppliers").getResultList();

        for (Object[] s : suppliers) {
            System.out.println(
                    "id: " + s[0] + "\t\t" +  "company: " + s[1] + "\t\t\t" +"name: " + s[2] + "\t\t\t" + "lastname: " + s[3] + "\t\t\t" + "address: "
+ s[4] + "\t\t" + s[5] + "\t\t" + s[6]
            );
        }
        tx.commit();
    }

}
```

Klasa Suppliers.

```java
@Entity
@Table(name = "order_details")
public class OrderDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @Column(insertable = false, updatable = false)
    private int order_id;
    @Column(insertable = false, updatable = false)
    private int product_id;
    private int quantity;
    private double unit_price;
    private double discount;
    private int status_id;
    @Temporal(TemporalType.TIMESTAMP)
    private Date date_allocated;

    @ManyToOne
    private Orders order;

    @ManyToOne
    private Products product;

    public OrderDetails(){}


    public OrderDetails(Products product, Orders order, int quantity){
        this.product = product;
        this.order = order;
        this.quantity = quantity;
        this.unit_price = product.getStandardCost();
    }

    public int getProductID(){
        return this.product_id;
    }

    public int getOrderID(){
        return this.order_id;
    }
}
```

Klasa OrderDetails

```java
public class OrderHandler {

    private Session session;

    public OrderHandler(Session session) {
        this.session = session;
    }

    /**
     * Realizuje zamówienie. Zbiera dane dotyczące klienta oraz zamówienia(id produktów oraz ich ilości).
     * Zapisuje zamówienie w tabeli Orders. Zapisuje w tabeli OrderDetails szczegóły dotyczące zamawianych pozycji.
     */
    public void submitOrder(){
        Customers customer = customerID();

        Products.listAllProducts(session);
        String product_id = readString("Podaj id produktu");
        String quantity = readString("Podaj ilość jednostek");
        ArrayList<Products> products = new ArrayList<>();
        ArrayList<Integer> quantities = new ArrayList<>();
        while(!(product_id.equals("")) &&!(quantity.equals(""))){
            products.add(Products.getProduct(session, Integer.parseInt(product_id)));
            quantities.add(Integer.parseInt(quantity));
            product_id = readString("Podaj id produktu");
            quantity = readString("Podaj ilość jednostek");
        }

        System.out.println("Do zamówienia zostaną dodane pozycje:");
        for(int i = 0; i < products.size(); i++){
            System.out.println(products.get(i).getProductName() + " ilość: " + quantities.get(i));
        }

        Transaction tx = session.beginTransaction();
        Orders order = new Orders(customer, new Date(), Employees.getEmployee(session));
        session.save(order);
        for(int i = 0; i < products.size(); i++){
            OrderDetails od = new OrderDetails(products.get(i), order, quantities.get(i));
            session.save(od);
        }
        tx.commit();
    }

    /**
     * Prosi użytkownika o podanie danych klienta. Jeśli klient widnieje w bazie danych - zwraca go.
     * Jeśli klient nie został wcześniej dodany do bazy prosi o podanie dodatkowych informacji po czym zapisuje klienta w bazie.
     * Następnie zwraca wcześniej utworzonego klienta.
     * @return
     */
    public Customers customerID(){
        String first_name = readString("Podaj imię");
        String last_name = readString("Podaj nazwisko");
        Customers c = Customers.getID(first_name, last_name, session);
        if(c == null){
            System.out.println("Nie znaleziono klienta");
            String address = readString("Podaj ulicę");
            String city = readString("Podaj miasto");
            String zip_postal_code = readString("Podaj kod pocztowy");
            Transaction tx = this.session.beginTransaction();
            Customers customer = new Customers(first_name, last_name, address, city, zip_postal_code);
            session.save(customer);
            tx.commit();
            c = Customers.getID(first_name, last_name, session);
        }
        return c;
    }
```

```
    /**
     * Wypisuje komunikat i pobiera odpowiedź od użytkownika.
     * @param message
     * @return
     */
    public static String readString(String message){
        System.out.print(message + ": ");
        Scanner scanner = new Scanner(System.in);
        return scanner.nextLine();
      }
}
```

Klasa OrderHanlder służąca do tworzenia zamówien

```
public class InvoiceHandler {

    private Session session;

    public InvoiceHandler(Session session) {
        this.session = session;
    }

    /**
     * Wyświetla produkty zawarte na jednym zamówieniu. (Faktura)
     */
    public void createInvoice(){
        Customers.showAllCustomers(session);
        int client_id = Integer.parseInt(OrderHandler.readString("Podaj ID klienta"));

        Orders.showOrders(client_id, session);
        int order_id = Integer.parseInt(OrderHandler.readString("Podaj ID zamówienia"));

        Transaction tx = session.beginTransaction();
        List<Object[]> invoices = session.createQuery("SELECT o.order_id, p.product_name, p.category, o.quantity, o.unit_price FROM
OrderDetails o JOIN Products p ON o.product_id = p.id").getResultList();

        int no_results = 0;
        for(Object[] i: invoices){
            if((Integer)i[0] == order_id){
                System.out.println("product_name: " + i[1] + "\t\t" +"category: " + i[2] + "\t\t" +"quantity: " + i[3] + "\t\t" +"unit_price: " +
i[4]);
            }
        }
        tx.commit();
    }
}
```

Klasa InvoiceHandler służąca do tworzenia faktur.

```
public class Main {
   private static final SessionFactory ourSessionFactory;

   static {
      try {
         Configuration configuration = new Configuration();
         configuration.configure();

         ourSessionFactory = configuration.buildSessionFactory();
      } catch (Throwable ex) {
         throw new ExceptionInInitializerError(ex);
      }
   }

   public static Session getSession() throws HibernateException {
      return ourSessionFactory.openSession();
   }

   public static void main(final String[] args) throws Exception {
      final Session session = getSession();
      try {
         String choice = OrderHandler.readString("Dodawanie zamówienia - wybierz 1\n" +"Wyświetlanie raportów - wybierz 2\n" +
                  "Generowanie faktury - wybierz 3\n" +"Wybór");
         if (Integer.parseInt(choice) == 1) {
            OrderHandler handler = new OrderHandler(session);
            handler.submitOrder();
         }
         else if (Integer.parseInt(choice) == 2) {
            String report_choice = OrderHandler.readString("Lista klientów - wybierz 1\n" +"Lista dostawców - wybierz 2\n" + "Lista
zamówień z dnia - wybierz 3\n" +"Lista zamówień klienta - wybierz 4\n" +"Wybór");
            if (Integer.parseInt(report_choice) == 1) {
               Customers.showAllCustomers(session);
            }
            else if(Integer.parseInt(report_choice) == 2){
               Suppliers.showAllSuppliers(session);
            }
            else if(Integer.parseInt(report_choice) == 3){
               String s_year = OrderHandler.readString("Podaj rok");
               String s_month = OrderHandler.readString("Podaj miesiąc");
               String s_day = OrderHandler.readString("Podaj dzień");
               int year = Integer.parseInt(s_year);
               int month = Integer.parseInt(s_month);
               int day = Integer.parseInt(s_day);
               Orders.showOrders(year, month, day, session);
            }
            else if (Integer.parseInt(report_choice) == 4){
               Customers.showAllCustomers(session);
               String customer_id = OrderHandler.readString("Podaj id klienta");
               int cid = Integer.parseInt(customer_id);
               Orders.showOrders(cid, session);
            }
            else{
               System.out.println("Nieprawidłowy wybór!");
            }
         }
         else if (Integer.parseInt(choice) == 3) {
            InvoiceHandler handler = new InvoiceHandler(session);
            handler.createInvoice();
         }
         else {
            System.out.println("Nieprawidłowy wybór!");
         }

      } finally {
         session.close();
      }
   }
}
```

Klasa Main odpowiadająca za logikę interfejsu.