

**UNIWERSYTET RZESZOWSKI**  
**WYDZIAŁ NAUK ŚCISŁYCH I TECHNICZNYCH**  
**INSTYTUT INFORMATYKI**



*Michał Janik*

134915

*Programowanie obiektowe 1*

*Dokumentacja techniczna projektu aplikacji Java  
z wykorzystaniem interfejsu graficznego Swing oraz połączenia  
bazodanowego JDBC pt. "Symulator sklepu spożywczego"*

Praca projektowa

Praca wykonana pod kierunkiem  
mgr inż. Ewa Żesławska

Rzeszów 2025



# **Spis treści**

<b>1. Struktura pracy projektowej z Programowania</b>	
<b>Obiektowego JAVA.....</b>	6
1.1. Streszczenie documentacji w języku polskim .....	6
1.2. Documentation summary in English .....	6
1.3. Opis założeń projektu .....	6
1.4. Opis struktury projektu.....	9
1.4.1. Diagram UML klasy Categories .....	10
1.4.2. Opis klasy Categories oraz jej metod.....	11
1.4.3. Diagram UML klasy Login.....	15
1.4.4. Opis klasy Login oraz jej metod .....	15
1.4.5. Diagram UML klasy Products .....	18
1.4.6. Opis klasy Products oraz jej metod.....	18
1.4.7. Diagram UML klasy Sellers .....	23
1.4.8. Opis klasy Sellers oraz jej metod.....	23
1.4.9. Diagram UML klasy Selling.....	28
1.4.10. Opis klasy Selling oraz jej metod .....	29
1.4.11. Diagram UML klasy Splash.....	33
1.4.12. Opis klasy Splash oraz jej metod .....	33
1.4.13. Diagram UML klasy UpdateAdminPwd .....	35
1.4.14. Opis klasy UpdateAdminPwd oraz jej metod.....	35
1.4.15. Diagram ERD bazy danych .....	37
1.4.16. Struktura bazy danych.....	37
1.5. Harmonogram realizacji projektu.....	40
1.6. Prezentacja warstwy użytkowej projektu .....	40
1.7. Podsumowanie.....	54
1.8. Oświadczenie studenta o samodzielności pracy.....	55
<b>Spis rysunków .....</b>	56
<b>Spis listingów .....</b>	58

# **1. Struktura pracy projektowej z Programowania Obiektowego JAVA**

## **1.1. Streszczenie dokumentacji w języku polskim**

Niniejsza praca projektowa przedstawia szczegółowo opis projektu aplikacji utworzonej przy użyciu języka Java, technologii Swing i JDBC. Omówione zostały wymagania funkcjonalne i niefunkcjonalne projektu, opis struktury projektu, diagramy klas UML łącznie z opisem ich metod, struktura całej bazy danych łącznie z diagramem ERD i tabelami, harmonogram realizacji projektu z użyciem diagramu Gantta oraz prezentacja warstwy użytkowej projektu z perspektywy przewidzianych dla tej aplikacji użytkowników z testowaniem obsługi błędów.

## **1.2. Documentation summary in English**

This project work presents a detailed description of the application project created using Java, Swing and JDBC technologies. It discusses the functional and non-functional requirements of the project, the description of the project structure, UML class diagrams including the description of their methods, the structure of the entire database including the ERD diagram and tables, the project implementation schedule using the Gantt chart and the presentation of the application's user layer from the perspective of the intended users for this application with error handling testing.

## **1.3. Opis założeń projektu**

Celem niniejszego projektu jest zapoznanie studentów z praktycznym połączeniem programu opartego na języku programowania Java z wykorzystaniem biblioteki do tworzenia obiektowych interfejsów graficznych (GUI) Swing oraz interfejsu do połączenia z bazą danych JDBC. Zarówno nieczytelny i nieintuicyjny interfejs użytkownika, jak i awaryjna baza danych może powodować problemy z obsługą systemu, i tym samym, sprzedawaniem produktów. W prawdziwym świecie zdarzają się nieraz sytuacje, że klient chce kupić dostępny towar w sklepie, ale niekoniecznie dostępny w bazie. Do rozwiązania problemu są potrzebne dobrze zaprojektowane bazy danych przechowujące informacje o administratorze, sprzedawcach, a w szczególności o dostępnych produktach i ich kategoriach. Problem został rozwiązyany przy użyciu programowania zorientowanego obiektowo i biblioteki łączącej interfejs graficzny z istniejącą bazą danych. Realizacja projektu przebiegła od zaprojektowania okien aplikacji, ich wyglądu, utworzenia bazy danych oraz zaprogramowania logiki łączącej poszczególne przyciski z poleceniami wysyłającymi lub pobierającymi treści w bazie danych. Wynikiem pracy jest aplikacja desktopowa "Supermarket" z wykorzystaniem biblioteki Swing oraz JDBC.

**Definicja:****Wymagania funkcjonalne**

- Opisują funkcje (czynności, operacje, usługi) wykonywane przez system.
- Często stosowany sposób opisu wymagań – język naturalny.
- Liczba wymagań funkcjonalnych może być bardzo duża; konieczne jest pewnego rodzaju uporządkowanie tych wymagań, które ułatwi pracę nad nimi (złożoność!).
- Opisują, jak funkcja powinna działać.
- Skupiają się na wyniku działania użytkownika.
- Definiują wymagania użytkownika.
- Posiadają funkcje uwzględnione w przypadkach użycia.
- Weryfikują funkcjonalność systemu.

**Wymagania niefunkcjonalne**

- Opisują ograniczenia, przy zachowaniu których system powinien realizować swoje funkcje.
- Opisują, jakie właściwości sprawią, że funkcja będzie działać.
- Skupiają się na uproszczeniu procesu i wykonania wyniku.
- Definiują oczekiwania i doświadczenia użytkownika działania użytkownika.
- Posiadają ograniczenia, które pomogą zredukować czas i koszty rozwoju.
- Weryfikują wydajność systemu.

**Wymagania funkcjonalne w tym projekcie:**

- Rejestracja, logowanie i uwierzytelnianie użytkowników.
- Funkcje administracyjne (np. zmiana hasła, zarządzanie danymi).
- Poziomy autoryzacji (administrator, sprzedawca).
- Generowanie wydruków paragonów.
- Możliwość filtrowania produktów z określonych kategorii u sprzedawcy.
- Pulpit administracyjny do zarządzania profilami sprzedawców, dostępnością produktów i ich kategorii.
- Integracja aplikacji z bazą danych w celu wymiany danych.
- Prosty interfejs, przejrzysta nawigacja.

**Wymagania niefunkcjonalne w tym projekcie:**

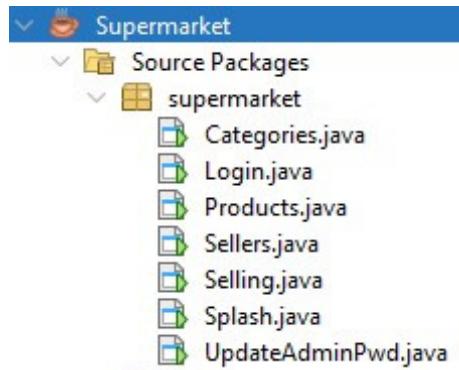
- Pojemność.
- Użyteczność.
- Integralność danych.
- Szybki czas ładowania i zdolność obsługi dużej liczby użytkowników jednocześnie.
- Minimalizacja kosztów serwera i hostingu.
- System zarządzania treścią (CMS) umożliwiający operacje CRUD (dodaj, odczytaj, zmodyfikuj, usuń).

## Rozwinięcie wymagań niefunkcjonalnych:

- Aplikacja IT powinna mieć kolor tła wszystkich ekranów rgb(0,153,153).
- Aplikacja IT powinna mieć krój czcionki Franklin Gothic Demi oraz jej kolor rgb(0,153,102)
- Pulpit zarządzania powinien pojawić się od razu po zalogowaniu administratora.
- Pulpit sprzedawy powinien pojawić się od razu po zalogowaniu sprzedawcy.
- Aplikacja IT powinna być w stanie obsłużyć dużą liczbę sprzedawców, zapewniając płynne działanie.

## 1.4. Opis struktury projektu

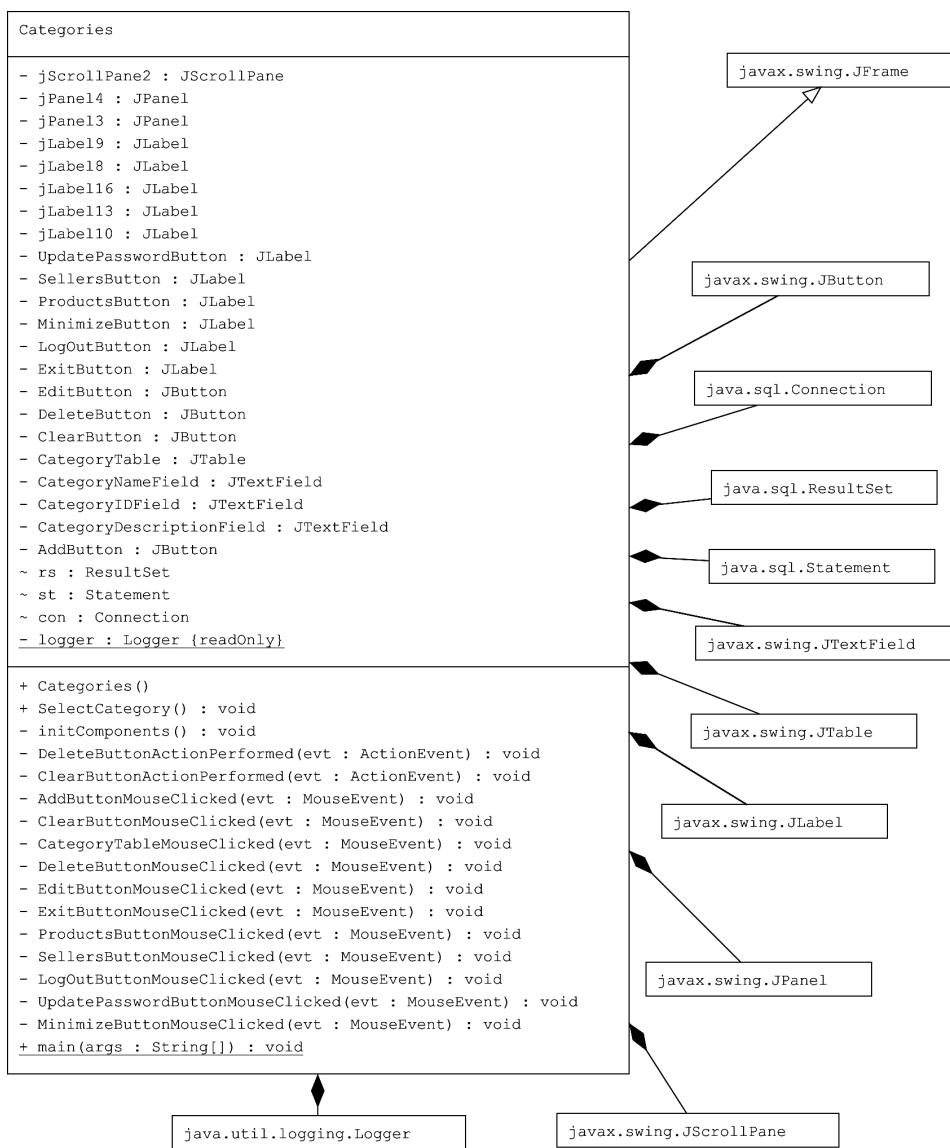
Do utworzenia tego projektu użyto programowania zorientowanego obiektowo, technologii Swing i JDBC. Aplikacja działa bez problemu na systemie Windows. Na strukturę projektu składa się 7 różnych klas okienek, każda z nich dziedziczy po klasie JFrame i opisuje inne okno:



Rys. 1.1. Lista klas projektowanej aplikacji.

### 1.4.1. Diagram UML klasy Categories

Diagram UML klasy Categories prezentuje się następująco:



Rys. 1.2. Diagram UML klasy Categories.

### 1.4.2. Opis klasy Categories oraz jej metod

Klasa Categories składa się z konstruktora oraz innych metod:

- Categories() - konstruktor okna:

**Listing 1.1.** Konstruktor okna Categories().

```

1 public Categories() {
2     initComponents();
3     SelectCategory();
4 }
```

- SelectCategory() - wyświetla wszystkie informacje o kategoriach z bazy danych w tabeli:

**Listing 1.2.** Metoda SelectCategory().

```

1 public void SelectCategory() {
2     try{
3         con = DriverManager.getConnection("jdbc:derby://localhost:1527/Supermarket"
4             , "Employee", "1234");
5         st = con.createStatement();
6         rs = st.executeQuery("SELECT * FROM CATEGORIES ORDER BY ID");
7         CategoryTable.setModel(DbUtils.resultSetToTableModel(rs));
8     } catch (SQLException e) {
9     }
9 }
```

- AddButtonMouseClicked() - podowuje dodanie nowego produktu do bazy danych:

**Listing 1.3.** Metoda AddButtonMouseClicked().

```

1 private void AddButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     if(CategoryIDField.getText().isEmpty() || CategoryNameField.getText().isEmpty()
3         || CategoryDescriptionField.getText().isEmpty()){
4         JOptionPane.showMessageDialog(this, "Missing information");
5     } else {
6         try{
7             con = DriverManager.getConnection("jdbc:derby://localhost:1527/
8             Supermarket", "Employee", "1234");
9             PreparedStatement Add = con.prepareStatement("INSERT INTO CATEGORIES
10             VALUES (?, ?, ?)");
11             Add.setInt(1, Integer.parseInt(CategoryIDField.getText()));
12             Add.setString(2, CategoryNameField.getText());
13             Add.setString(3, CategoryDescriptionField.getText());
14             Add.executeUpdate();
15             JOptionPane.showMessageDialog(this, "Category added successfully");
16             con.close();
17             SelectCategory();
18         } catch(SQLException e) {
19             e.printStackTrace();
20         }
21     }
22 }
```

- ClearButtonMouseClicked() - powoduje wyczyszczenie danych wpisanych w formularz:

**Listing 1.4.** Metoda ClearButtonMouseClicked().

```

1 private void ClearButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     CategoryIDField.setText("");
3     CategoryNameField.setText("");
4     CategoryDescriptionField.setText("");
5 }
```

- CategoryTableMouseClicked() - uzupełnia pola formularza po naciśnięciu przycisku myszy na wyświetlony rekord bazy:

**Listing 1.5.** Metoda CategoryTableMouseClicked().

```

1 private void CategoryTableMouseClicked(java.awt.event.MouseEvent evt) {
2     DefaultTableModel model = (DefaultTableModel)CategoryTable.getModel();
3     int MyIndex = CategoryTable.getSelectedRow();
4     CategoryIDField.setText(model.getValueAt(MyIndex, 0).toString());
5     CategoryNameField.setText(model.getValueAt(MyIndex, 1).toString());
6     CategoryDescriptionField.setText(model.getValueAt(MyIndex, 2).toString());
7 }
```

- DeleteButtonMouseClicked() - powoduje usunięcie istniejącego produktu z bazy danych:

**Listing 1.6.** Metoda DeleteButtonMouseClicked().

```

1 private void DeleteButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     if(CategoryIDField.getText().isEmpty())
3     {
4         JOptionPane.showMessageDialog(this, "Enter the category to be deleted");
5     }
6     else
7     {
8         try{
9             con = DriverManager.getConnection("jdbc:derby://localhost:1527/
Supermarket","Employee","1234");
10            String CId = CategoryIDField.getText();
11            String Query = "DELETE FROM CATEGORIES WHERE ID="+CId;
12            Statement Add = con.createStatement();
13            Add.executeUpdate(Query);
14            SelectCategory();
15            JOptionPane.showMessageDialog(this, "Category deleted successfully");
16        } catch(SQLException e){
17            e.printStackTrace();
18        }
19    }
20 }
```

- EditButtonMouseClicked() - powoduje modyfikację istniejącego produktu w bazie danych:

**Listing 1.7.** Metoda EditButtonMouseClicked().

```

1 private void EditButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     if(CategoryIDField.getText().isEmpty() || CategoryNameField.getText().isEmpty()
3         || CategoryDescriptionField.getText().isEmpty()){
4         JOptionPane.showMessageDialog(this, "Missing information");
5     }
6     else{
7         try{
8             con = DriverManager.getConnection("jdbc:derby://localhost:1527/
9             Supermarket","Employee","1234");
10            String Query = "UPDATE CATEGORIES SET NAME='"+CategoryNameField.getText()
11            ()+"',DESCRIPTION='"+CategoryDescriptionField.getText()+"' WHERE ID="
12            CategoryIDField.getText();
13            Statement Add = con.createStatement();
14            Add.executeUpdate(Query);
15            SelectCategory();
16            JOptionPane.showMessageDialog(this, "Category updated");
17        } catch(SQLException e){
18            e.printStackTrace();
19        }
20    }
21 }
```

- ExitButtonMouseClicked() - powoduje zamknięcie okienka aplikacji i jednoczesne zakończenie jej wykonywania:

**Listing 1.8.** Metoda ExitButtonMouseClicked().

```

1 private void ExitButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     this.dispose();
3 }
```

- ProductsButtonMouseClicked() - powoduje przejście programu do okna produktów:

**Listing 1.9.** Metoda ProductsButtonMouseClicked().

```

1 private void ProductsButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     new Products().setVisible(true);
3     this.dispose();
4 }
```

- SellersButtonMouseClicked() - powoduje przejście programu do okna sprzedawców:

**Listing 1.10.** Metoda SellersButtonMouseClicked().

```

1 private void SellersButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     new Sellers().setVisible(true);
3     this.dispose();
4 }
```

- LogOutButtonMouseClicked() - powoduje przejście programu do okna logowania:

**Listing 1.11.** Metoda LogOutButtonMouseClicked().

```
1 private void LogOutButtonMouseClicked(java.awt.event.MouseEvent evt) {  
2     new Login().setVisible(true);  
3     this.dispose();  
4 }
```

- UpdatePasswordButtonMouseClicked() - powoduje wyświetlenie okna zmiany hasła dla administratora:

**Listing 1.12.** Metoda UpdatePasswordButtonMouseClicked().

```
1 private void UpdatePasswordButtonMouseClicked(java.awt.event.MouseEvent evt) {  
2     new UpdateAdminPwd().setVisible(true);  
3 }
```

- MinimizeButtonMouseClicked() - powoduje minimalizację okna:

**Listing 1.13.** Metoda MinimizeButtonMouseClicked().

```
1 private void MinimizeButtonMouseClicked(java.awt.event.MouseEvent evt) {  
2     this.setState(Frame.ICONIFIED);  
3 }
```

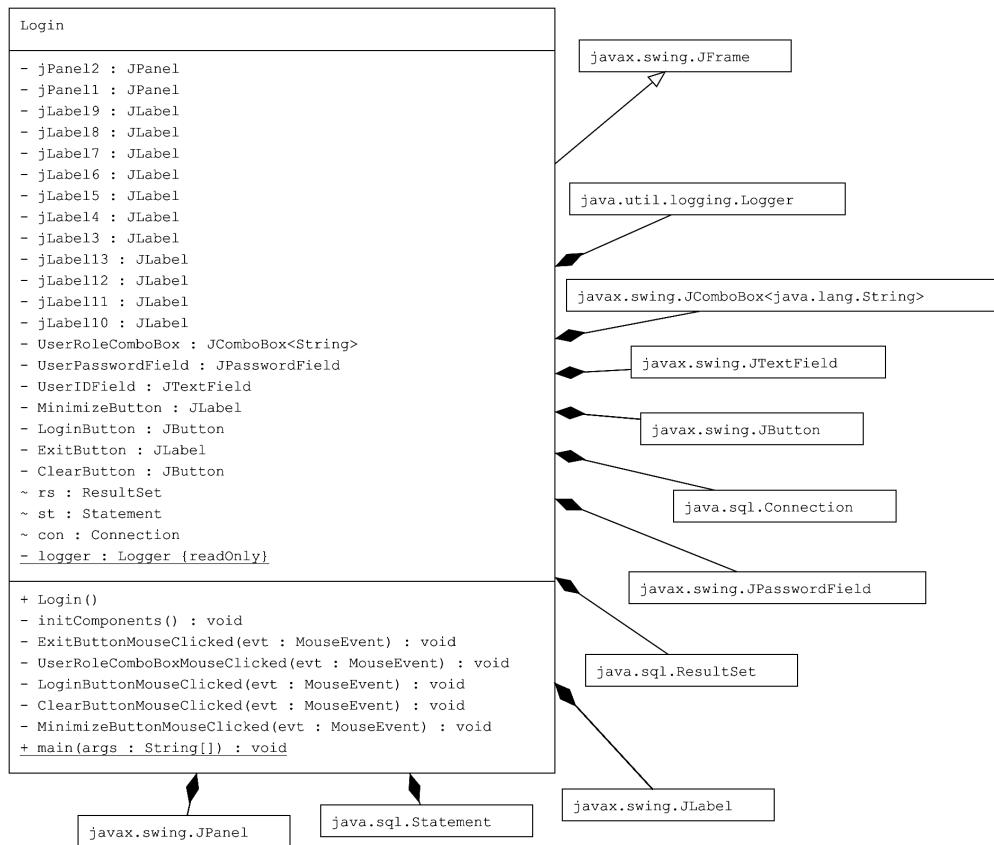
- main() - zawiera metodę tworzącą nowy obiekt okna:

**Listing 1.14.** Metoda uruchomieniowa okna main().

```
1 public static void main(String args[]) {  
2     java.awt.EventQueue.invokeLater(() -> new Categories().setVisible(true));  
3 }
```

### 1.4.3. Diagram UML klasy Login

Diagram UML klasy Login prezentuje się następująco:



Rys. 1.3. Diagram UML klasy Login.

### 1.4.4. Opis klasy Login oraz jej metod

Klasa Login składa się z konstruktora oraz innych metod:

- `Login()` - konstruktor okna:

**Listing 1.15.** Konstruktor okna `Login()`.

```

1 public Login() {
2     initComponents();
3 }
  
```

- `ClearButtonMouseClicked()` - powoduje wyczyszczenie danych wpisanych w formularz:

**Listing 1.16.** Metoda `ClearButtonMouseClicked()`.

```

1 private void ClearButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     UserIDField.setText("");
3     UserPasswordField.setText("");
4 }
  
```

- LoginButtonMouseClicked() - powoduje zalogowanie się użytkownika:

**Listing 1.17.** Metoda LoginButtonMouseClicked().

```

1 private void LoginButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     if(UserIDField.getText().isEmpty() || UserPasswordField.getText().isEmpty()){
3         JOptionPane.showMessageDialog(this, "Missing information");
4     }
5     else
6     {
7         if(UserRoleComboBox.getSelectedItem().toString().equals("Seller")){
8             String Query = "SELECT * FROM SELLERS WHERE NAME='"+UserIDField.getText()
9             +"'" AND PASSWORD='"+UserPasswordField.getText()+"'";
10            try{
11                con = DriverManager.getConnection("jdbc:derby://localhost:1527/
12                Supermarket","Employee","1234");
13                st = con.createStatement();
14                rs = st.executeQuery(Query);
15                if(rs.next()){
16                    new Selling().setVisible(true);
17                    this.dispose();
18                }
19                else
20                {
21                    JOptionPane.showMessageDialog(this, "Wrong seller ID or
22                    password");
23                }
24            } catch(SQLException e){
25                e.printStackTrace();
26            }
27        } else {
28            String Query = "SELECT * FROM ADMIN WHERE NAME='"+UserIDField.getText()
29             +"'" AND PASSWORD='"+UserPasswordField.getText()+"'";
30            try{
31                con = DriverManager.getConnection("jdbc:derby://localhost:1527/
32                Supermarket","Employee","1234");
33                st = con.createStatement();
34                rs = st.executeQuery(Query);
35                if(rs.next()){
36                    new Products().setVisible(true);
37                    this.dispose();
38                }
39            } catch(SQLException e){
40                e.printStackTrace();
41            }
42        }
43    }

```

- ExitButtonMouseClicked() - powoduje zamknięcie okienka aplikacji i jednoczesne zakończenie jej wykonywania:

**Listing 1.18.** Metoda ExitButtonMouseClicked().

```
1 private void ExitButtonMouseClicked(java.awt.event.MouseEvent evt) {  
2     this.dispose();  
3 }
```

- MinimizeButtonMouseClicked() - powoduje minimalizację okna:

**Listing 1.19.** Metoda MinimizeButtonMouseClicked().

```
1 private void MinimizeButtonMouseClicked(java.awt.event.MouseEvent evt) {  
2     this.setState(Frame.ICONIFIED);  
3 }
```

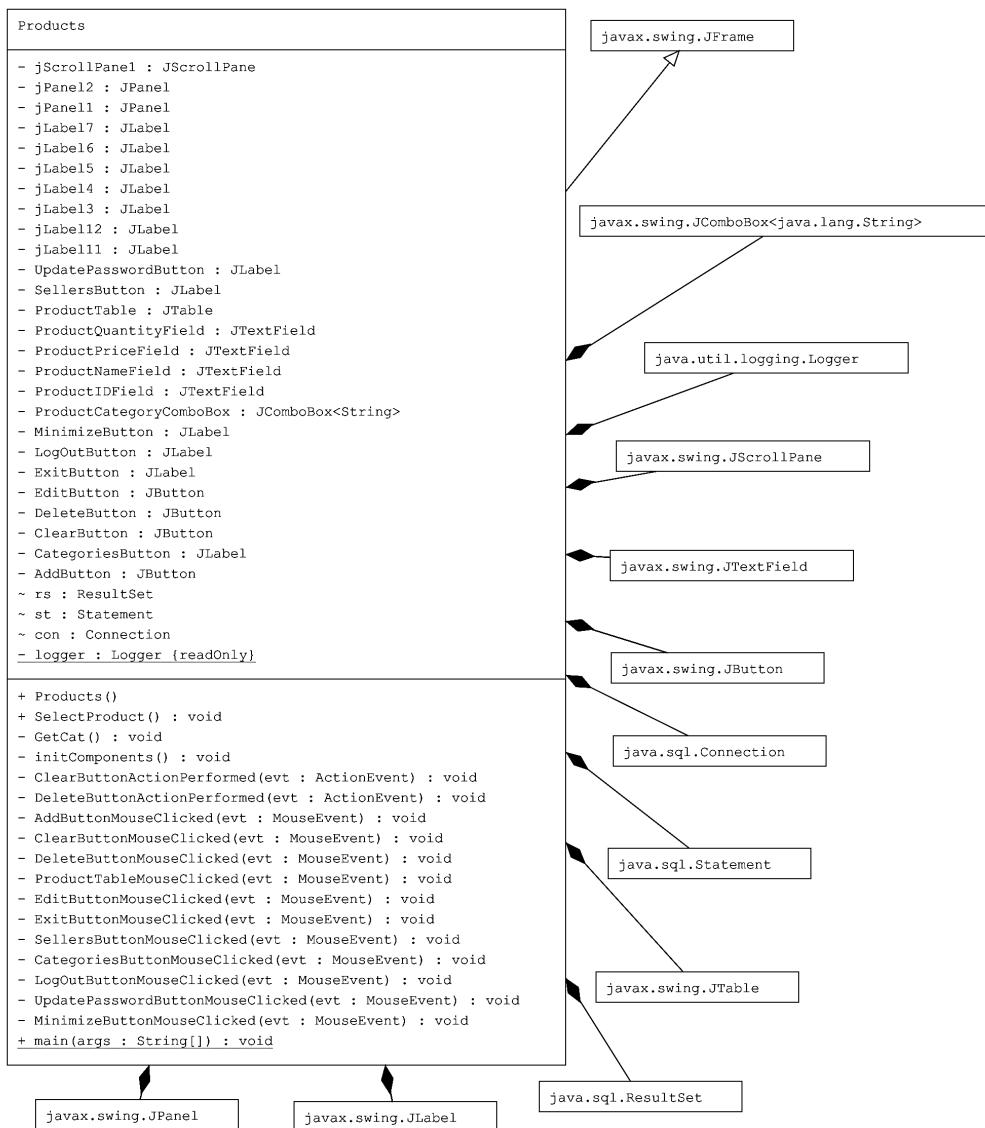
- main() - zawiera metodę tworzącą nowy obiekt okna:

**Listing 1.20.** Metoda uruchomieniowa okna main().

```
1 public static void main(String args[]) {  
2     java.awt.EventQueue.invokeLater(() -> new Login().setVisible(true));  
3 }
```

### 1.4.5. Diagram UML klasy Products

Diagram UML klasy Products prezentuje się następująco:



Rys. 1.4. Diagram UML klasy Products.

### 1.4.6. Opis klasy Products oraz jej metod

Klasa Products składa się z konstruktora oraz innych metod:

- `Products()` - konstruktor okna:

**Listing 1.21.** Konstruktor okna Products().

```

1 public Products() {
2     initComponents();
3     SelectProduct();
4     GetCat();
5 }

```

- AddButtonMouseClicked() - powoduje dodanie danych wpisanych w formularz do bazy danych:

**Listing 1.22.** Metoda AddButtonMouseClicked().

```

1 private void AddButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     if(ProductIDField.getText().isEmpty() || ProductNameField.getText().isEmpty()
3         || ProductPriceField.getText().isEmpty() || ProductQuantityField.getText()
4             .isEmpty()){
5         JOptionPane.showMessageDialog(this, "Missing information");
6     } else {
7         try{
8             con = DriverManager.getConnection("jdbc:derby://localhost:1527/
9                 Supermarket","Employee","1234");
10            PreparedStatement Add = con.prepareStatement("INSERT INTO PRODUCTS
11                VALUES(?, ?, ?, ?, ?)");
12            Add.setInt(1, Integer.parseInt(ProductIDField.getText()));
13            Add.setString(2, ProductNameField.getText());
14            Add.setInt(3, Integer.parseInt(ProductQuantityField.getText()));
15            Add.setDouble(4, Double.parseDouble(ProductPriceField.getText()));
16            Add.setString(5, ProductCategoryComboBox.getSelectedItem().toString());
17            Add.executeUpdate();
18            JOptionPane.showMessageDialog(this, "Product added successfully");
19            con.close();
20            SelectProduct();
21        } catch(SQLException e) {
22            e.printStackTrace();
23        }
24    }
25 }
```

- CategoriesButtonMouseClicked() - powoduje przejście z obecnego okna do okna kategorii:

**Listing 1.23.** Metoda CategoriesButtonMouseClicked().

```

1 private void CategoriesButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     new Categories().setVisible(true);
3     this.dispose();
4 }
```

- ClearButtonMouseClicked() - powoduje wyczyszczenie danych formularza:

**Listing 1.24.** Metoda ClearButtonMouseClicked().

```

1 private void ClearButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     ProductIDField.setText("");
3     ProductNameField.setText("");
4     ProductPriceField.setText("");
5     ProductQuantityField.setText("");
6 }
```

- DeleteButtonMouseClicked() - powoduje usunięcie wybranego rekordu z bazy danych:

**Listing 1.25.** Metoda DeleteButtonMouseClicked().

```

1 private void DeleteButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     if(ProductIDField.getText().isEmpty())
3     {
4         JOptionPane.showMessageDialog(this, "Enter the product to be deleted");
5     }
6     else
7     {
8         try{
9             con = DriverManager.getConnection("jdbc:derby://localhost:1527/
10            Supermarket","Employee","1234");
11             String PrId = ProductIDField.getText();
12             String Query = "DELETE FROM PRODUCTS WHERE ID="+PrId;
13             Statement Add = con.createStatement();
14             Add.executeUpdate(Query);
15             SelectProduct();
16             JOptionPane.showMessageDialog(this, "Product deleted successfully");
17         } catch(SQLException e){
18             e.printStackTrace();
19         }
20     }
21 }
```

- EditButtonMouseClicked() - powoduje modyfikację wybranego rekordu z bazy danych:

**Listing 1.26.** Metoda EditButtonMouseClicked().

```

1 private void EditButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     if(ProductIDField.getText().isEmpty() || ProductNameField.getText().isEmpty()
3     || ProductQuantityField.getText().isEmpty() || ProductPriceField.getText().
4     isEmpty()){
5         JOptionPane.showMessageDialog(this, "Missing information");
6     }
7     else
8     {
9         try{
10             con = DriverManager.getConnection("jdbc:derby://localhost:1527/
11            Supermarket","Employee","1234");
12             String Query = "UPDATE PRODUCTS SET NAME='"+ProductNameField.getText()+
13             "', QUANTITY='"+ProductQuantityField.getText()+"', PRICE='"+ProductPriceField.
14             getText()+"', CATEGORY='"+ProductCategoryComboBox.getSelectedItem().toString()+"'
15             WHERE ID='"+ProductIDField.getText();
16             Statement Add = con.createStatement();
17             Add.executeUpdate(Query);
18             SelectProduct();
19             JOptionPane.showMessageDialog(this, "Product updated");
20         } catch(SQLException e){
21             e.printStackTrace();
22         }
23     }
24 }
```

- GetCat() - powoduje pobranie dostępnych kategorii z bazy danych:

**Listing 1.27.** Metoda GetCat().

```
1 private void GetCat()
2 {
3     try{
4         con = DriverManager.getConnection("jdbc:derby://localhost:1527/Supermarket"
5             , "Employee", "1234");
6         st = con.createStatement();
7         String query = "SELECT * FROM CATEGORIES";
8         rs = st.executeQuery(query);
9         while(rs.next())
10        {
11            String MyCat = rs.getString("NAME");
12            ProductCategoryComboBox.addItem(MyCat);
13        }
14    } catch (SQLException e) {
15        e.printStackTrace();
16    }
17 }
```

- LogOutButtonMouseClicked() - powoduje przejście do ekranu logowania:

**Listing 1.28.** Metoda LogOutButtonMouseClicked().

```
1 private void LogOutButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     new Login().setVisible(true);
3     this.dispose();
4 }
```

- ProductTableMouseClicked() - uzupełnia pola formularza po naciśnięciu przycisku myszy na wyświetlony rekord bazy:

**Listing 1.29.** Metoda ProductTableMouseClicked().

```
1 private void ProductTableMouseClicked(java.awt.event.MouseEvent evt) {
2     DefaultTableModel model = (DefaultTableModel) ProductTable.getModel();
3     int MyIndex = ProductTable.getSelectedRow();
4     ProductIDField.setText(model.getValueAt(MyIndex, 0).toString());
5     ProductNameField.setText(model.getValueAt(MyIndex, 1).toString());
6     ProductQuantityField.setText(model.getValueAt(MyIndex, 2).toString());
7     ProductPriceField.setText(model.getValueAt(MyIndex, 3).toString());
8 }
```

- SelectProduct() - wyświetla wszystkie informacje o produktach z bazy danych w tabeli:

**Listing 1.30.** Metoda SelectProduct().

```

1 public void SelectProduct() {
2     try{
3         con = DriverManager.getConnection("jdbc:derby://localhost:1527/Supermarket"
4             , "Employee", "1234");
5         st = con.createStatement();
6         rs = st.executeQuery("SELECT * FROM PRODUCTS ORDER BY ID");
7         ProductTable.setModel(DbUtils.resultSetToTableModel(rs));
8     } catch (SQLException e) {
9         e.printStackTrace();
10    }
11 }
```

- SellersButtonMouseClicked() - powoduje przejście z obecnego okna do okna sprzedawców:

**Listing 1.31.** Metoda SellersButtonMouseClicked().

```

1 private void SellersButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     new Sellers().setVisible(true);
3     this.dispose();
4 }
```

- UpdatePasswordButtonMouseClicked() - powoduje wyświetlenie okna zmiany hasła dla administratora:

**Listing 1.32.** Metoda UpdatePasswordButtonMouseClicked().

```

1 private void UpdatePasswordButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     new UpdateAdminPwd().setVisible(true);
3 }
```

- ExitButtonMouseClicked() - powoduje zamknięcie okienka aplikacji i jednoczesne zakończenie jej wykonywania:

**Listing 1.33.** Metoda ExitButtonMouseClicked().

```

1 private void ExitButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     dispose();
3 }
```

- MinimizeButtonMouseClicked() - powoduje minimalizację okna:

**Listing 1.34.** Metoda MinimizeButtonMouseClicked().

```

1 private void MinimizeButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     this.setState(Frame.ICONIFIED);
3 }
```

- main() - zawiera metodę tworzącą nowy obiekt okna:

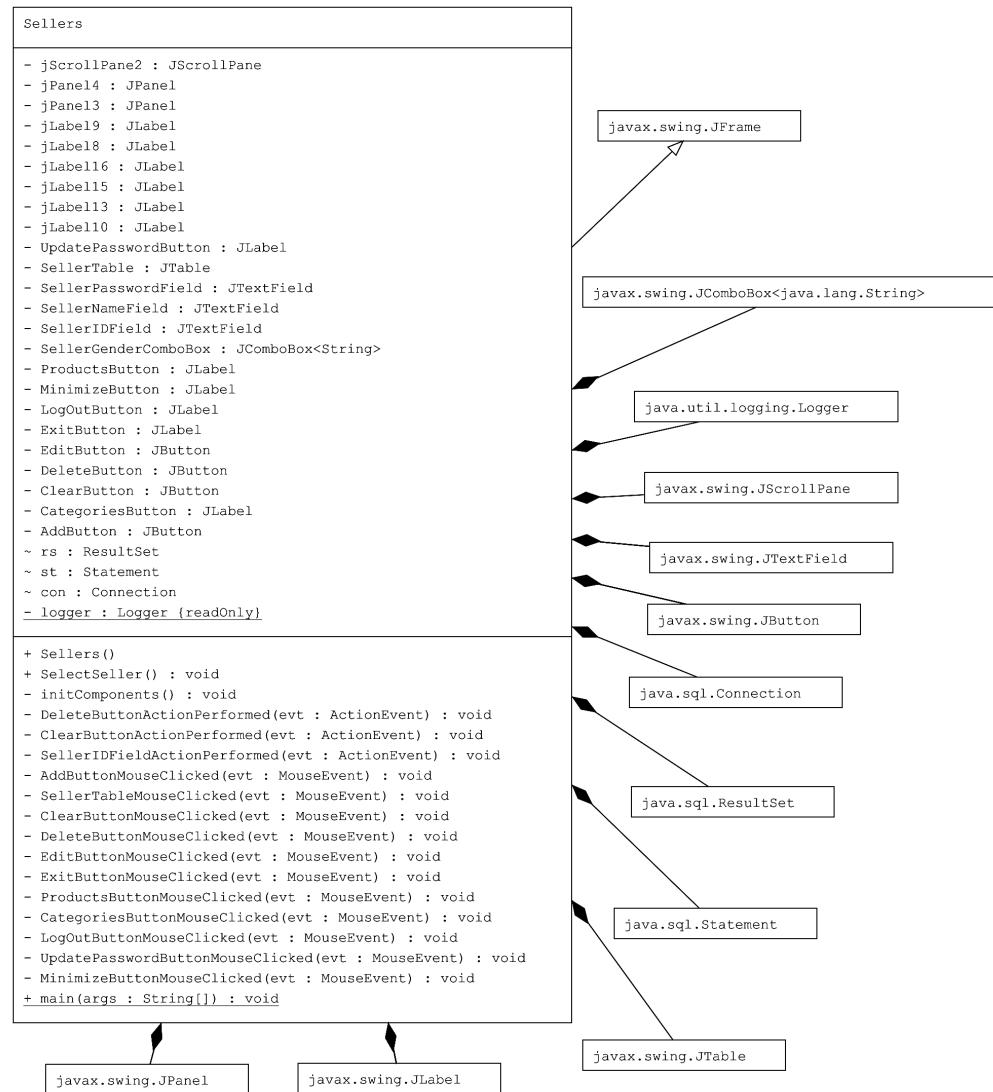
**Listing 1.35.** Metoda uruchomieniowa okna main().

```

1 public static void main(String args[]) {
2     java.awt.EventQueue.invokeLater(() -> new Products().setVisible(true));
3 }
```

### 1.4.7. Diagram UML klasy Sellers

Diagram UML klasy Sellers prezentuje się następująco:



Rys. 1.5. Diagram UML klasy Sellers.

### 1.4.8. Opis klasy Sellers oraz jej metod

Klasa Sellers składa się z konstruktora oraz innych metod:

- `Sellers()` - konstruktor okna:

**Listing 1.36.** Konstruktor okna Sellers().

```

1 public Sellers() {
2     initComponents();
3     SelectSeller();
4 }
  
```

- AddButtonMouseClicked() - powoduje dodanie danych wpisanych w formularz do bazy danych:

**Listing 1.37.** Metoda AddButtonMouseClicked().

```

1 private void AddButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     if(SellerIDField.getText().isEmpty() || SellerNameField.getText().isEmpty() ||
3         SellerPasswordField.getText().isEmpty()){
4         JOptionPane.showMessageDialog(this, "Missing information");
5     } else {
6         try{
7             con = DriverManager.getConnection("jdbc:derby://localhost:1527/
8             Supermarket", "Employee", "1234");
9             PreparedStatement Add = con.prepareStatement("INSERT INTO SELLERS
10             VALUES(?, ?, ?, ?)");
11             Add.setInt(1, Integer.parseInt(SellerIDField.getText()));
12             Add.setString(2, SellerNameField.getText());
13             Add.setString(3, SellerPasswordField.getText());
14             Add.setString(4, SellerGenderComboBox.getSelectedItem().toString());
15             Add.executeUpdate();
16             JOptionPane.showMessageDialog(this, "Seller added successfully");
17             con.close();
18             SelectSeller();
19         } catch(SQLException e) {
20             e.printStackTrace();
21         }
22     }
23 }
```

- CategoriesButtonMouseClicked() - powoduje przejście z obecnego okna do okna kategorii:

**Listing 1.38.** Metoda CategoriesButtonMouseClicked().

```

1 private void CategoriesButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     new Categories().setVisible(true);
3     this.dispose();
4 }
```

- ClearButtonMouseClicked() - powoduje wyczyszczenie danych formularza:

**Listing 1.39.** Metoda ClearButtonMouseClicked().

```

1 private void ClearButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     SellerIDField.setText("");
3     SellerNameField.setText("");
4     SellerPasswordField.setText("");
5 }
```

- DeleteButtonMouseClicked() - powoduje usunięcie wybranego rekordu z bazy danych:

**Listing 1.40.** Metoda DeleteButtonMouseClicked().

```

1 private void DeleteButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     if(SellerIDField.getText().isEmpty())
3     {
4         JOptionPane.showMessageDialog(this, "Enter the seller to be deleted");
5     }
6     else
7     {
8         try{
9             con = DriverManager.getConnection("jdbc:derby://localhost:1527/
10 Supermarket","Employee","1234");
11             String SId = SellerIDField.getText();
12             String Query = "DELETE FROM SELLERS WHERE ID="+SId;
13             Statement Add = con.createStatement();
14             Add.executeUpdate(Query);
15             SelectSeller();
16             JOptionPane.showMessageDialog(this, "Seller deleted successfully");
17         } catch(SQLException e){
18             e.printStackTrace();
19         }
20     }
21 }
```

- EditButtonMouseClicked() - powoduje modyfikację wybranego rekordu z bazy danych:

**Listing 1.41.** Metoda EditButtonMouseClicked().

```

1 private void EditButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     if(SellerIDField.getText().isEmpty() || SellerNameField.getText().isEmpty() ||
3     SellerPasswordField.getText().isEmpty()){
4         JOptionPane.showMessageDialog(this, "Missing information");
5     }
6     else
7     {
8         try{
9             con = DriverManager.getConnection("jdbc:derby://localhost:1527/
10 Supermarket","Employee","1234");
11             String Query = "UPDATE SELLERS SET NAME='"+SellerNameField.getText()+"'
12             ,PASSWORD='"+SellerPasswordField.getText()+"',GENDER='"+SellerGenderComboBox.
13             getSelectedItem().toString()+"' WHERE ID="+SellerIDField.getText();
14             Statement Add = con.createStatement();
15             Add.executeUpdate(Query);
16             SelectSeller();
17             JOptionPane.showMessageDialog(this, "Seller updated");
18         } catch(SQLException e){
19             e.printStackTrace();
20         }
21     }
22 }
```

- LogOutButtonMouseClicked() - powoduje przejście do ekranu logowania:

**Listing 1.42.** Metoda LogOutButtonMouseClicked().

```

1 private void LogOutButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     new Login().setVisible(true);
3     this.dispose();
4 }
```

- ProductsButtonMouseClicked() - powoduje przejście programu do okna produktów:

**Listing 1.43.** Metoda ProductsButtonMouseClicked().

```

1 private void ProductsButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     new Products().setVisible(true);
3     this.dispose();
4 }
```

- SelectSeller() - wyświetla wszystkie informacje o sprzedawcach z bazy danych w tabeli:

**Listing 1.44.** Metoda SelectSeller().

```

1 public void SelectSeller() {
2     try{
3         con = DriverManager.getConnection("jdbc:derby://localhost:1527/Supermarket"
4             , "Employee", "1234");
5         st = con.createStatement();
6         rs = st.executeQuery("SELECT * FROM SELLERS ORDER BY ID");
7         SellerTable.setModel(DbUtils.resultSetToTableModel(rs));
8     } catch (SQLException e) {
9         e.printStackTrace();
10    }
11 }
```

- SellerTableMouseClicked() - uzupełnia pola formularza po naciśnięciu przycisku myszy na wyświetlony rekord bazy:

**Listing 1.45.** Metoda SellerTableMouseClicked().

```

1 private void SellerTableMouseClicked(java.awt.event.MouseEvent evt) {
2     DefaultTableModel model = (DefaultTableModel)SellerTable.getModel();
3     int MyIndex = SellerTable.getSelectedRow();
4     SellerIDField.setText(model.getValueAt(MyIndex, 0).toString());
5     SellerNameField.setText(model.getValueAt(MyIndex, 1).toString());
6     SellerPasswordField.setText(model.getValueAt(MyIndex, 2).toString());
7 }
```

- UpdatePasswordButtonMouseClicked - powoduje wyświetlenie okna zmiany hasła dla administratora:

**Listing 1.46.** Metoda UpdatePasswordButtonMouseClicked().

```

1 private void UpdatePasswordButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     new UpdateAdminPwd().setVisible(true);
3 }
```

- ExitButtonMouseClicked() - powoduje zamknięcie okienka aplikacji i jednoczesne zakończenie jej wykonywania:

**Listing 1.47.** Metoda ExitButtonMouseClicked().

```
1 private void ExitButtonMouseClicked(java.awt.event.MouseEvent evt) {  
2     this.dispose();  
3 }
```

- MinimizeButtonMouseClicked() - powoduje minimalizację okna:

**Listing 1.48.** Metoda MinimizeButtonMouseClicked().

```
1 private void MinimizeButtonMouseClicked(java.awt.event.MouseEvent evt) {  
2     this.setState(Frame.ICONIFIED);  
3 }
```

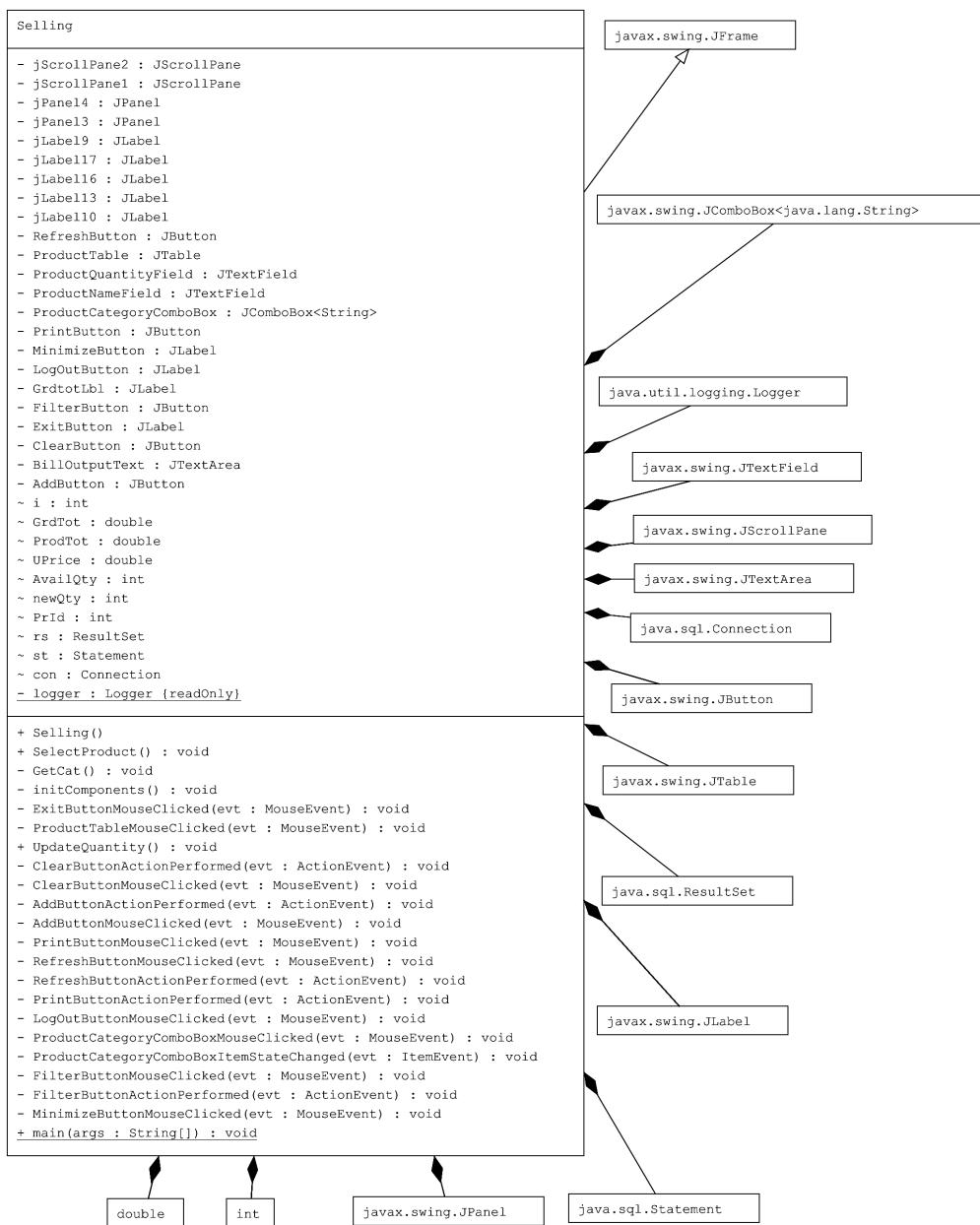
- main() - zawiera metodę tworzącą nowy obiekt okna:

**Listing 1.49.** Metoda uruchomieniowa okna main().

```
1 public static void main(String args[]) {  
2     java.awt.EventQueue.invokeLater(() -> new Sellers().setVisible(true));  
3 }
```

### 1.4.9. Diagram UML klasy Selling

Diagram UML klasy Selling prezentuje się następująco:



Rys. 1.6. Diagram UML klasy Selling.

### 1.4.10. Opis klasy Selling oraz jej metod

Klasa Selling składa się z konstruktora oraz innych metod:

- Selling() - konstruktor okna:

**Listing 1.50.** Konstruktor okna Selling().

```

1 public Selling() {
2     initComponents();
3     SelectProduct();
4     GetCat();
5 }
```

- AddButtonMouseClicked() - powoduje dodanie danych wpisanych w formularz do rachunku:

**Listing 1.51.** Metoda AddButtonMouseClicked().

```

1 private void AddButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     if(ProductQuantityField.getText().isEmpty() || ProductNameField.getText() .
3         isEmpty())
4     {
5         JOptionPane.showMessageDialog(this, "Missing information");
6     }
7     else if(AvailQty <= Integer.parseInt(ProductQuantityField.getText()))
8     {
9         System.out.println(AvailQty+ " ---- "+Integer.valueOf(ProductQuantityField.
10             getText()));
11         JOptionPane.showMessageDialog(this, "Not enough in stock");
12     }
13     else
14     {
15         i++;
16         ProdTot = UPrice * Double.parseDouble(ProductQuantityField.getText());
17         GrdTot = GrdTot + ProdTot;
18
19         if(i == 1)
20         {
21             BillOutputText.setText(BillOutputText.getText()+""
22             "-----YOUR SUPERMARKET-----\nNUM      PRODUCT      PRICE      QUANTITY
23             TOTAL\n"+i+"      "+ProductNameField.getText()+"      "+UPrice+"      "
24             +ProductQuantityField.getText()+"      "+new DecimalFormat("##.##").format(
25             ProdTot)+"\n");
26         }
27         else
28         {
29             BillOutputText.setText(BillOutputText.getText()+i+"      "
30             ProductNameField.getText()+"      "+UPrice+"      "
31             "+ProductQuantityField.
32             getText()+"      "+new DecimalFormat("##.##").format(ProdTot)+"\n");
33         }
34         GrdtotLbl.setText("Grand total: "+new DecimalFormat("##.##").format(GrdTot)
35     );
36         UpdateQuantity();
37     }
38 }
```

- ClearButtonMouseClicked() - powoduje wyczyszczenie danych formularza:

**Listing 1.52.** Metoda ClearButtonMouseClicked().

```

1 private void ClearButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     ProductNameField.setText("");
3     ProductQuantityField.setText("");
4 }
```

- FilterButtonMouseClicked() - powoduje filtrowanie produktów po określonej kategorii:

**Listing 1.53.** Metoda FilterButtonMouseClicked().

```

1 private void FilterButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     try{
3         con = DriverManager.getConnection("jdbc:derby://localhost:1527/Supermarket"
4             , "Employee", "1234");
5         st = con.createStatement();
6         rs = st.executeQuery("SELECT * FROM PRODUCTS WHERE CATEGORY='"+
7             ProductCategoryComboBox.getSelectedItem().toString()+"'");
8         ProductTable.setModel(DbUtils.resultSetToTableModel(rs));
9     } catch (SQLException e) {
10         e.printStackTrace();
11     }
12 }
```

- GetCat() - powoduje pobranie dostępnych kategorii z bazy danych:

**Listing 1.54.** Metoda GetCat().

```

1 private void GetCat()
2 {
3     try{
4         con = DriverManager.getConnection("jdbc:derby://localhost:1527/Supermarket"
5             , "Employee", "1234");
6         st = con.createStatement();
7         String query = "SELECT * FROM CATEGORIES ORDER BY ID";
8         rs = st.executeQuery(query);
9         while(rs.next())
10     {
11         String MyCat = rs.getString("NAME");
12         ProductCategoryComboBox.addItem(MyCat);
13     }
14 } catch (SQLException e) {
15     e.printStackTrace();
16 }
```

- LogOutButtonMouseClicked() - powoduje przejście do ekranu logowania:

**Listing 1.55.** Metoda LogOutButtonMouseClicked().

```

1 private void LogOutButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     new Login().setVisible(true);
3     this.dispose();
4 }
```

- PrintButtonMouseClicked() - umożliwia wydrukowanie paragonu w drukarce lub do pliku PDF:

**Listing 1.56.** Metoda PrintButtonMouseClicked().

```

1 private void PrintButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     try{
3         BillOutputText.print();
4     } catch(PrinterException e){
5         e.printStackTrace();
6     }
7 }
```

- ProductTableMouseClicked() - uzupełnia pola formularza po naciśnięciu przycisku myszy na wyświetlony rekord bazy:

**Listing 1.57.** Metoda ProductTableMouseClicked().

```

1 private void ProductTableMouseClicked(java.awt.event.MouseEvent evt) {
2     DefaultTableModel model = (DefaultTableModel)ProductTable.getModel();
3     int MyIndex = ProductTable.getSelectedRow();
4     PrId = Integer.parseInt(model.getValueAt(MyIndex, 0).toString());
5     ProductNameField.setText(model.getValueAt(MyIndex, 1).toString());
6     AvailQty = Integer.parseInt(model.getValueAt(MyIndex, 2).toString());
7     newQty = AvailQty - Integer.parseInt(ProductQuantityField.getText());
8     UPrice = Double.parseDouble(model.getValueAt(MyIndex, 3).toString());
9 }
```

- RefreshButtonMouseClicked() - odświeża listę dostępnych produktów na liście:

**Listing 1.58.** Metoda RefreshButtonMouseClicked().

```

1 private void RefreshButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     SelectProduct();
3 }
```

- SelectProduct() - wyświetla wszystkie informacje o produktach z bazy danych w tabeli:

**Listing 1.59.** Metoda SelectProduct().

```

1 public void SelectProduct () {
2     try{
3         con = DriverManager.getConnection("jdbc:derby://localhost:1527/Supermarket"
4             , "Employee", "1234");
5         st = con.createStatement();
6         rs = st.executeQuery("SELECT * FROM PRODUCTS ORDER BY ID");
7         ProductTable.setModel(DbUtils.resultSetToTableModel(rs));
8     } catch (SQLException e) {
9         e.printStackTrace();
10    }
11 }
```

- UpdateQuantity() - aktualizuje ilość produktów w bazie po dodaniu ich do rachunku:

**Listing 1.60.** Metoda UpdateQuantity().

```

1 public void UpdateQuantity() {
2     try{
3         con = DriverManager.getConnection("jdbc:derby://localhost:1527/Supermarket"
4             , "Employee", "1234");
5         String Query = "UPDATE PRODUCTS SET QUANTITY="+newQty+" WHERE ID="+PrId;
6         Statement Add = con.createStatement();
7         Add.executeUpdate(Query);
8         SelectProduct();
9     } catch(SQLException e){
10        e.printStackTrace();
11    }
12 }
```

- ExitButtonMouseClicked() - powoduje zamknięcie okienka aplikacji i jednoczesne zakończenie jej wykonywania:

**Listing 1.61.** Metoda ExitButtonMouseClicked().

```

1 private void ExitButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     this.dispose();
3 }
```

- MinimizeButtonMouseClicked() - powoduje minimalizację okna:

**Listing 1.62.** Metoda MinimizeButtonMouseClicked().

```

1 private void MinimizeButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     this.setState(Frame.ICONIFIED);
3 }
```

- main() - zawiera metodę tworzącą nowy obiekt okna:

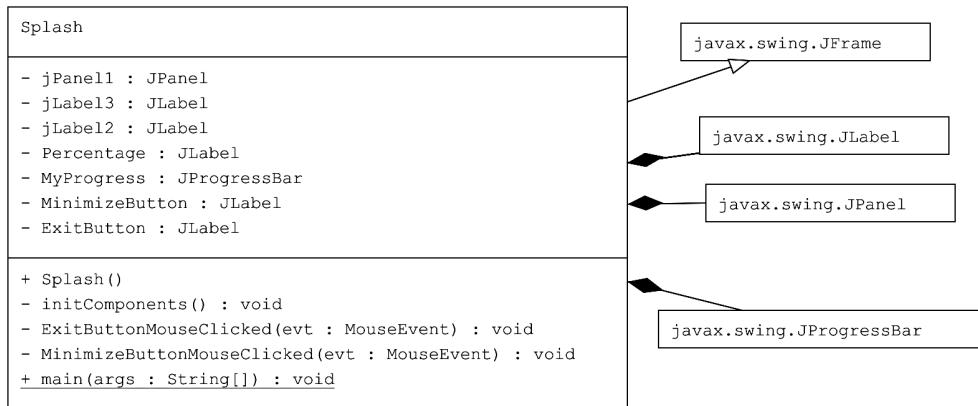
**Listing 1.63.** Metoda uruchomieniowa okna main().

```

1 public static void main(String args[]){ 
2     java.awt.EventQueue.invokeLater(() -> new Selling().setVisible(true));
3 }
```

### 1.4.11. Diagram UML klasy Splash

Diagram UML klasy Splash prezentuje się następująco:



Rys. 1.7. Diagram UML klasy Splash.

### 1.4.12. Opis klasy Splash oraz jej metod

Klasa Splash składa się z konstruktora oraz innych metod:

- Splash() - konstruktor okna:

**Listing 1.64.** Konstruktor okna Splash().

```

1 public Splash() {
2     initComponents();
3 }
  
```

- ExitButtonMouseClicked() - powoduje zamknięcie okienka aplikacji i jednoczesne zakończenie jej wykonywania, bez uruchamiania innych okienek:

**Listing 1.65.** Metoda ExitButtonMouseClicked().

```

1 private void ExitButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     System.exit(0);
3 }
  
```

- MinimizeButtonMouseClicked() - powoduje minimalizację okna:

**Listing 1.66.** Metoda MinimizeButtonMouseClicked().

```

1 private void MinimizeButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     this.setState(Frame.ICONIFIED);
3 }
  
```

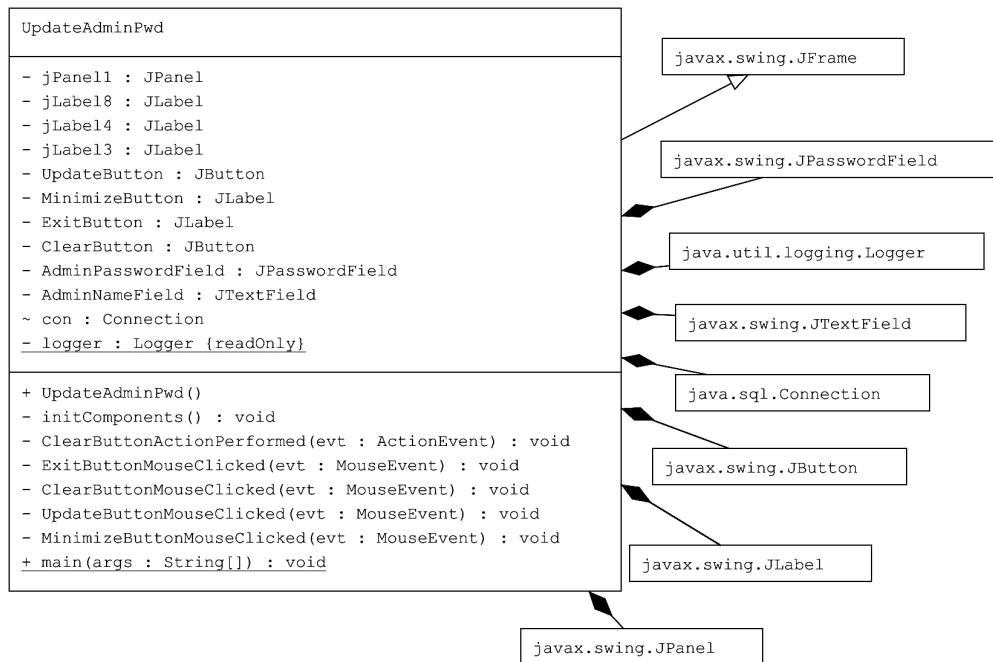
- main() - zawiera metodę tworzącą nowy obiekt okna:

**Listing 1.67.** Metoda uruchomieniowa okna main().

```
1 public static void main(String args[]) {
2     Splash MySplash = new Splash();
3     MySplash.setVisible(true);
4     try{
5         for(int i = 0; i <= 100; i++)
6         {
7             Thread.sleep(40);
8             MySplash.MyProgress.setValue(i);
9             MySplash.Percentage.setText(Integer.toString(i)+"%");
10        }
11    } catch(InterruptedException e) {
12    }
13    new Login().setVisible(true);
14    MySplash.dispose();
15 }
```

### 1.4.13. Diagram UML klasy UpdateAdminPwd

Diagram UML klasy UpdateAdminPwd prezentuje się następująco:



Rys. 1.8. Diagram UML klasy UpdateAdminPwd.

### 1.4.14. Opis klasy UpdateAdminPwd oraz jej metod

Klasa `UpdateAdminPwd` składa się z konstruktora oraz innych metod:

- `UpdateAdminPwd()` - konstruktor okna:

**Listing 1.68.** Konstruktor okna `UpdateAdminPwd()`.

```

1 public UpdateAdminPwd() {
2     initComponents();
3 }
  
```

- `ClearButtonMouseClicked()` - powoduje wyczyszczenie danych formularza:

**Listing 1.69.** Metoda `ClearButtonMouseClicked()`.

```

1 private void ClearButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     AdminNameField.setText("");
3     AdminPasswordField.setText("");
4 }
  
```

- UpdateButtonMouseClicked() - powoduje zmianę loginu i hasła administratora:

**Listing 1.70.** Metoda UpdateButtonMouseClicked().

```

1 private void UpdateButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     if(AdminNameField.getText().isEmpty() || AdminPasswordField.getText().isEmpty())
3     {
4         JOptionPane.showMessageDialog(this, "Missing information");
5     }
6     else
7     {
8         try{
9             con = DriverManager.getConnection("jdbc:derby://localhost:1527/
10            Supermarket","Employee","1234");
11             String Query = "UPDATE ADMIN SET NAME='"+AdminNameField.getText()+"',
12             PASSWORD='"+AdminPasswordField.getText()+"' WHERE ID="+1;
13             Statement Add = con.createStatement();
14             Add.executeUpdate(Query);
15             JOptionPane.showMessageDialog(this, "Password successfully updated");
16         } catch(SQLException e){
17             e.printStackTrace();
18         }
19     }
20 }
```

- ExitButtonMouseClicked() - powoduje zamknięcie okienka aplikacji i jednoczesne zakończenie jej wykonywania:

**Listing 1.71.** Metoda ExitButtonMouseClicked().

```

1 private void ExitButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     this.dispose();
3 }
```

- MinimizeButtonMouseClicked() - powoduje minimalizację okna:

**Listing 1.72.** Metoda MinimizeButtonMouseClicked().

```

1 private void MinimizeButtonMouseClicked(java.awt.event.MouseEvent evt) {
2     this.setState(Frame.ICONIFIED);
3 }
```

- main() - zawiera metodę tworzącą nowy obiekt okna:

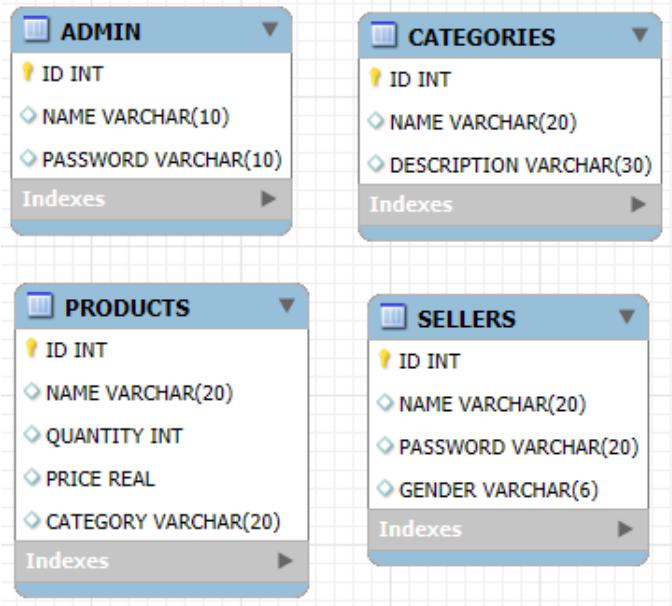
**Listing 1.73.** Metoda uruchomieniowa okna main().

```

1 public static void main(String args[]) {
2     java.awt.EventQueue.invokeLater(() -> new UpdateAdminPwd().setVisible(true));
3 }
```

### 1.4.15. Diagram ERD bazy danych

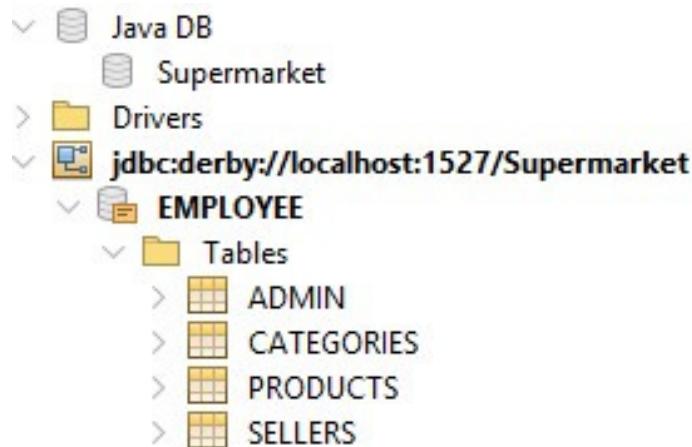
Diagram związków encji (ERD, Entity Relationship Diagram) bazy danych Supermarket składa się z czterech tabel, przedstawia się następująco:



Rys. 1.9. Diagram ERD bazy danych Supermarket.

### 1.4.16. Struktura bazy danych

Baza danych działa domyślnie na porcie 1527. Domyślnie zalogowany użytkownik nazywa się "Employee" i posiada hasło "1234". Jej struktura przedstawia się następująco:



Rys. 1.10. Baza danych Supermarket i jej struktura tabel.

Baza danych składa się z czterech tabel. Przykładowe dane testowe zostały do nich pomyślnie dodane:

- ADMIN:

#	ID	NAME	PASSWORD
1	1	Admin	admin

Rys. 1.11. Tabela ADMIN.

- CATEGORIES:

#	ID	NAME	DESCRIPTION
1	1	Beverage	Drink for your desire
2	2	Baking	For baking and pastry
3	3	Bread	Your feeding primary
4	4	Personal care	Perfect products for your care
5	5	Dairy	Nice dairy to enjoy
6	6	Meat	Fresh and tasty meat

Rys. 1.12. Tabela CATEGORIES.

- PRODUCTS:

#	ID	NAME	QUANTITY	PRICE	CATEGORY
1	1	Pepsi bottle	100	12,59	Beverage
2	2	Sausage	189	35,99	Meat
3	3	Ice tea	64	14,99	Beverage
4	4	Coca-Cola can	103	13,99	Beverage
5	5	Strawberry yoghurt	174	10,99	Dairy

Rys. 1.13. Tabela PRODUCTS.

- SELLERS:

#	ID	NAME	PASSWORD	GENDER
1	1	Angela	1642	Female
2	2	Antonio	7189	Male
3	3	Martha	1837	Female
4	4	James	6435	Male

Rys. 1.14. Tabela SELLERS.

Każda z tych tabel ma odpowiednio dobrane typy danych do przechowywania informacji:

Table name: ADMIN							
Key	Index	Null	Unique	Column name	Data type	Size	
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	ID	INTEGER	0	
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NAME	VARCHAR	10	
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	PASSWORD	VARCHAR	10	

Rys. 1.15. Typy danych tabeli ADMIN.

Table name: CATEGORIES							
Key	Index	Null	Unique	Column name	Data type	Size	
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	ID	INTEGER	0	
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NAME	VARCHAR	20	
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	DESCRIPTION	VARCHAR	30	

Rys. 1.16. Typy danych tabeli CATEGORIES.

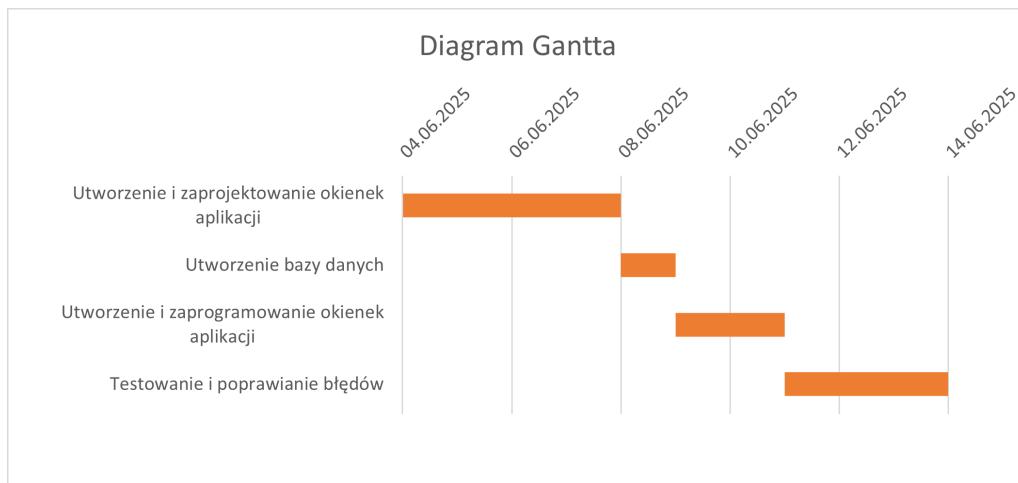
Table name: PRODUCTS							
Key	Index	Null	Unique	Column name	Data type	Size	
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	ID	INTEGER	0	
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NAME	VARCHAR	20	
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	QUANTITY	INTEGER	0	
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	PRICE	REAL	0	
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	CATEGORY	VARCHAR	20	

Rys. 1.17. Typy danych tabeli PRODUCTS.

Table name: SELLERS							
Key	Index	Null	Unique	Column name	Data type	Size	
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	ID	INTEGER	0	
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NAME	VARCHAR	20	
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	PASSWORD	VARCHAR	20	
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	GENDER	VARCHAR	6	

Rys. 1.18. Typy danych tabeli SELLERS.

## 1.5. Harmonogram realizacji projektu



Rys. 1.19. Diagram Gantta realizacji projektu.

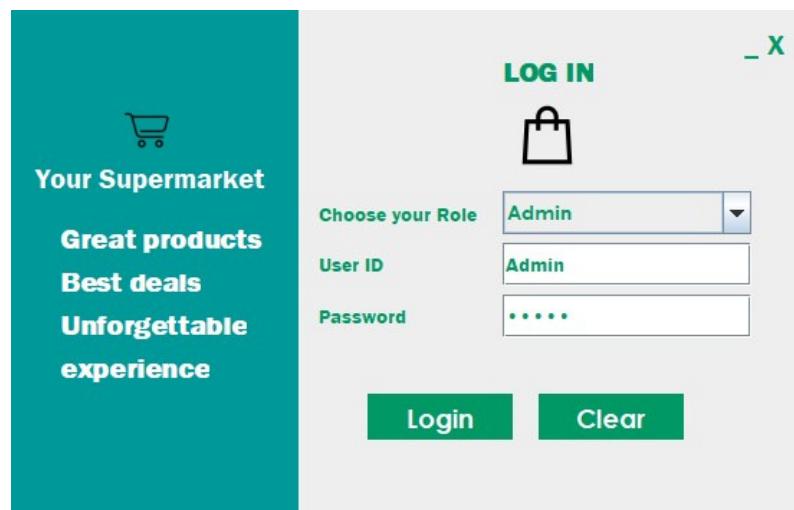
Diagram przedstawia wykonywane czynności przy tworzeniu programu. Najdłużej czasu zajęło projektowanie i programowanie okienek aplikacji, a najmniej - utworzenie bazy danych. Największe trudności przy programowaniu wystąpiły przy realizacji okienka sprzedaży i zliczaniu produktów. Poza tym potrzebny był specjalny plik do połączenia interfejsu graficznego do bazy danych (rs2xml.jar) i zainstalowanie specjalnej biblioteki (Java DB Driver). Projekt został umieszczony w repozytorium GitHub o adresie URL: <https://github.com/MichalJanikUR/Java-Swing-Supermarket>

## 1.6. Prezentacja warstwy użytkowej projektu

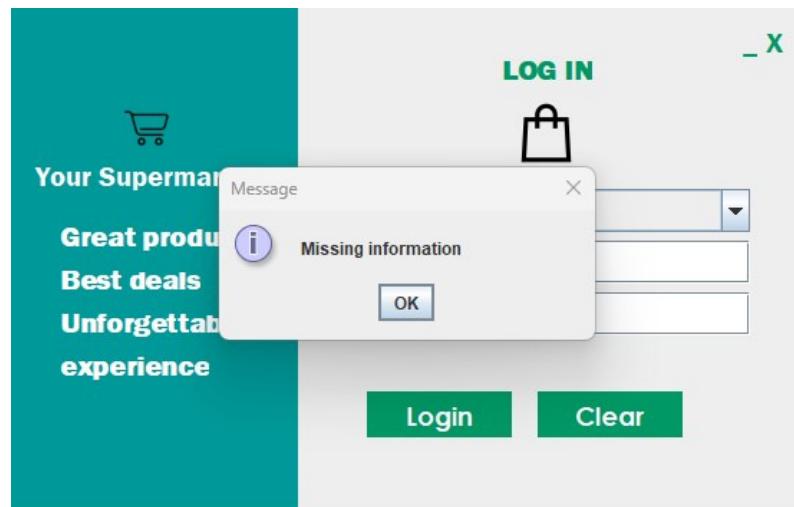
Aplikacja służy do zarządzania produktami i ich kategoriami w sklepie, a także do zarządzania ich sprzedawcami i sprzedażą produktów. Warstwa użytkowa projektu składa się z jednego okna uruchamiającego (Splash) oraz sześciu innych okienek funkcjonalnych:



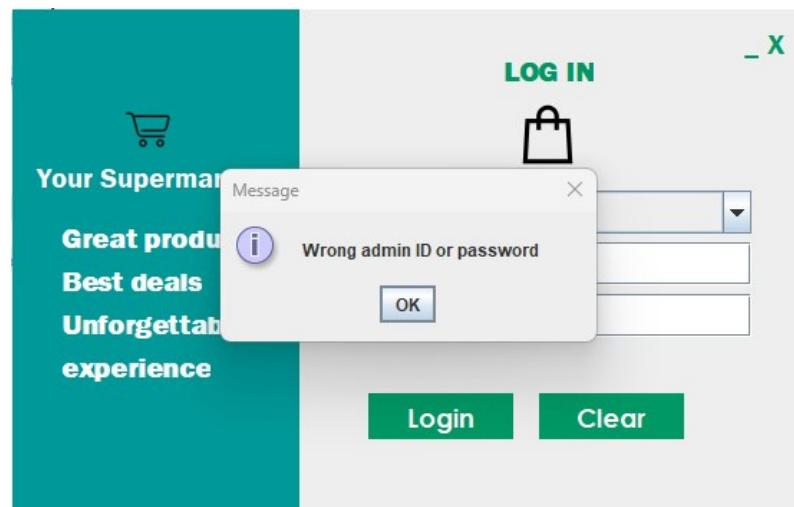
Rys. 1.20. Okienko załadowywania programu.



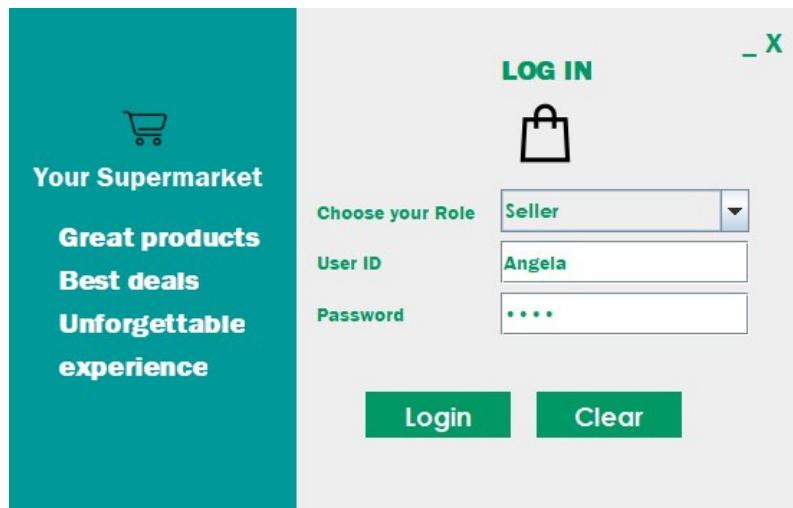
Rys. 1.21. Panel logowania - administrator.



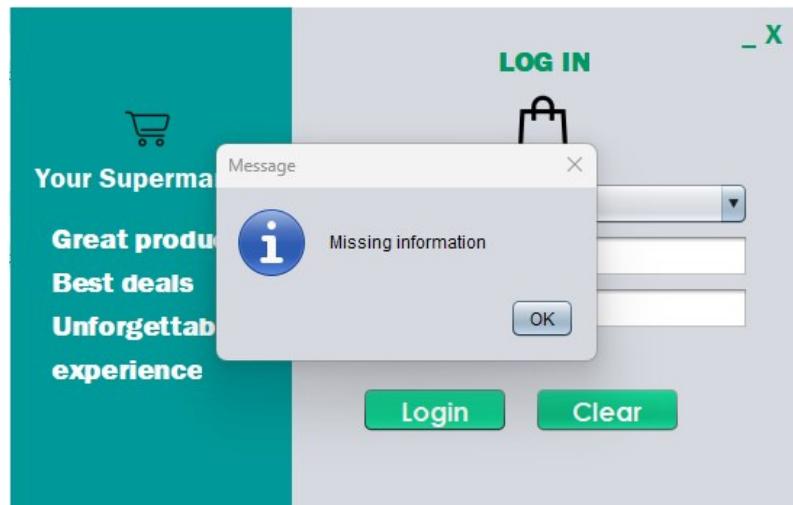
Rys. 1.22. Panel logowania - administrator. Brak danych.



Rys. 1.23. Panel logowania - administrator. Błędne dane logowania.



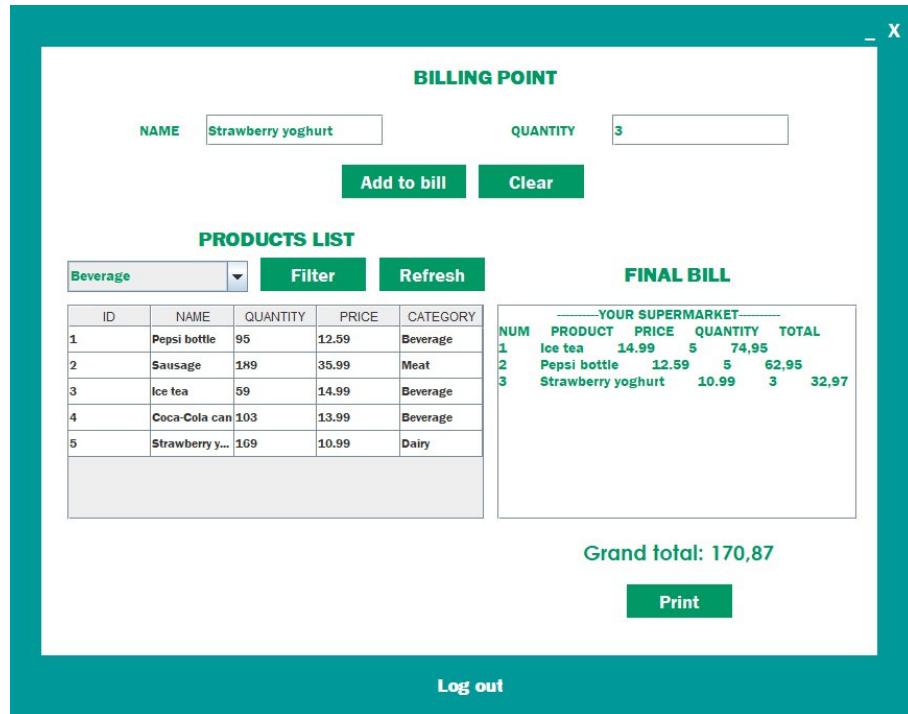
Rys. 1.24. Panel logowania - sprzedawca.



Rys. 1.25. Panel logowania - sprzedawca. Brak danych.



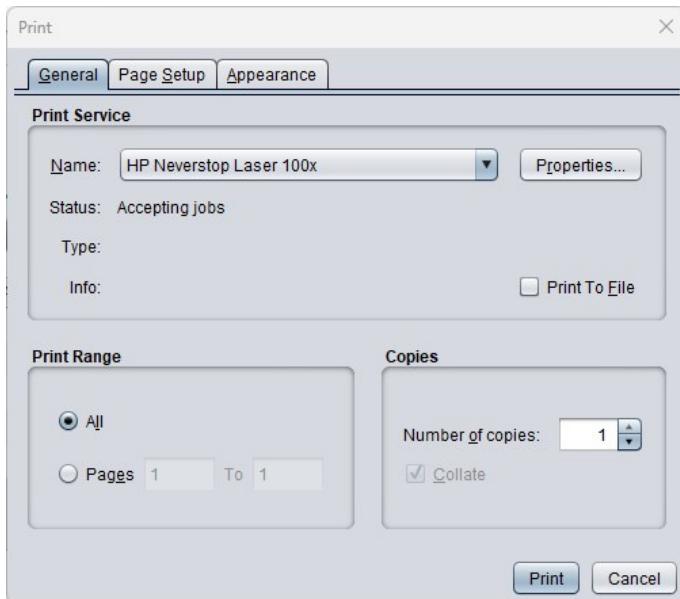
Rys. 1.26. Panel logowania - sprzedawca. Błędne dane logowania.



Rys. 1.27. Panel sprzedaży produktów.

PRODUCTS LIST				
Meat		Filter	Refresh	
ID	NAME	QUANTITY	PRICE	CATEGORY
2	Sausage	189	35.99	Meat

Rys. 1.28. Filtrowanie produktów po kategorii.



Rys. 1.29. Możliwość wydruku wystawionego rachunku.

**MANAGE PRODUCTS**

PRODUCT ID	3	QUANTITY	59
NAME	Ice tea	PRICE	14.99
CATEGORY	Beverage		

Add Edit Delete Clear

**PRODUCTS LIST**

ID	NAME	QUANTITY	PRICE	CATEGORY
1	Pepsi bottle	95	12.59	Beverage
2	Sausage	189	35.99	Meat
3	Ice tea	59	14.99	Beverage
4	Coca-Cola can	103	13.99	Beverage
5	Strawberry yoghurt	169	10.99	Dairy

**SELLERS**    **CATEGORIES**    [Update password](#)    [Log out](#)

Rys. 1.30. Panel zarządzania produktami.

**MANAGE PRODUCTS**

PRODUCT ID	3	QUANTITY	59
NAME	Ice tea	PRICE	14.99
CATEGORY	Beverage		

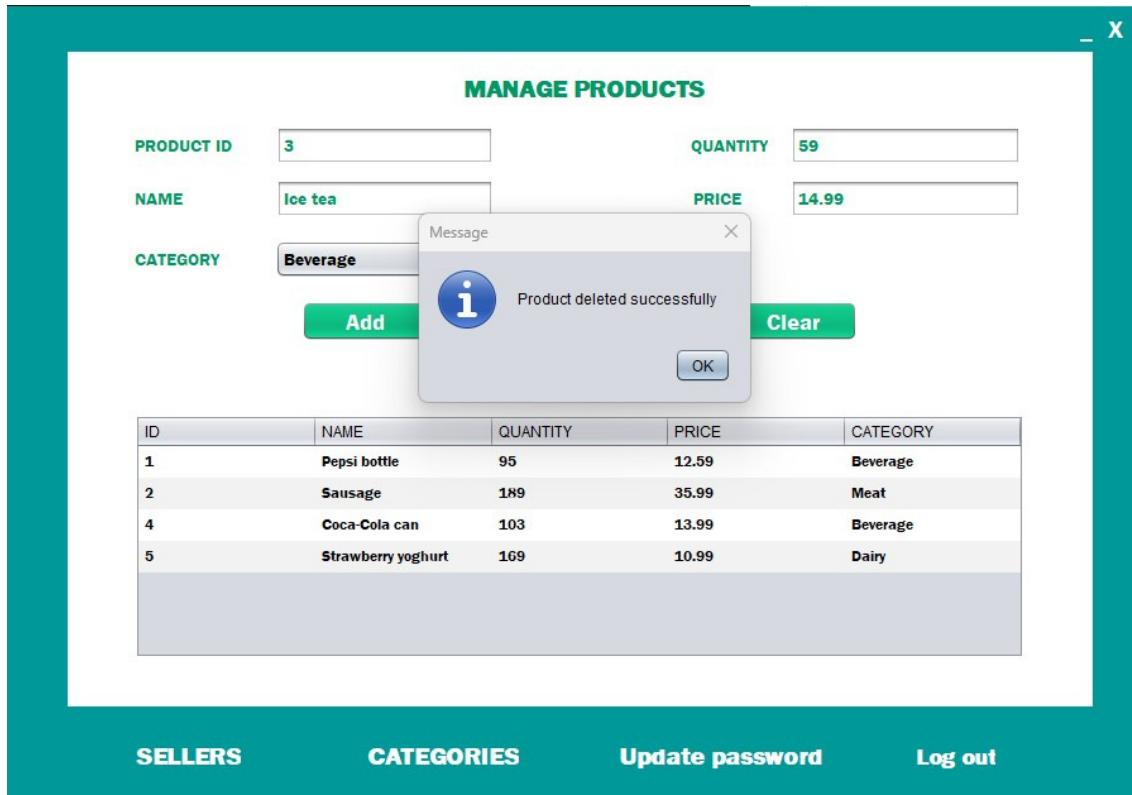
Add Clear

**Message**

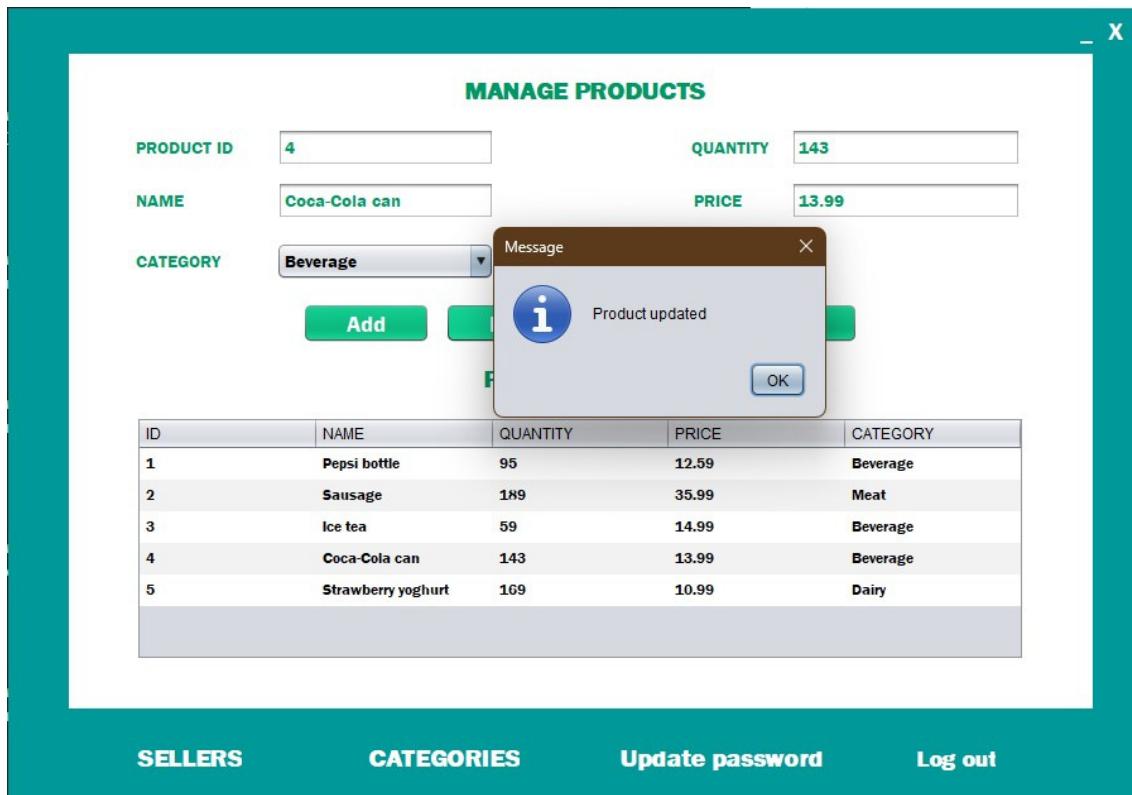
Product added successfully

**SELLERS**    **CATEGORIES**    [Update password](#)    [Log out](#)

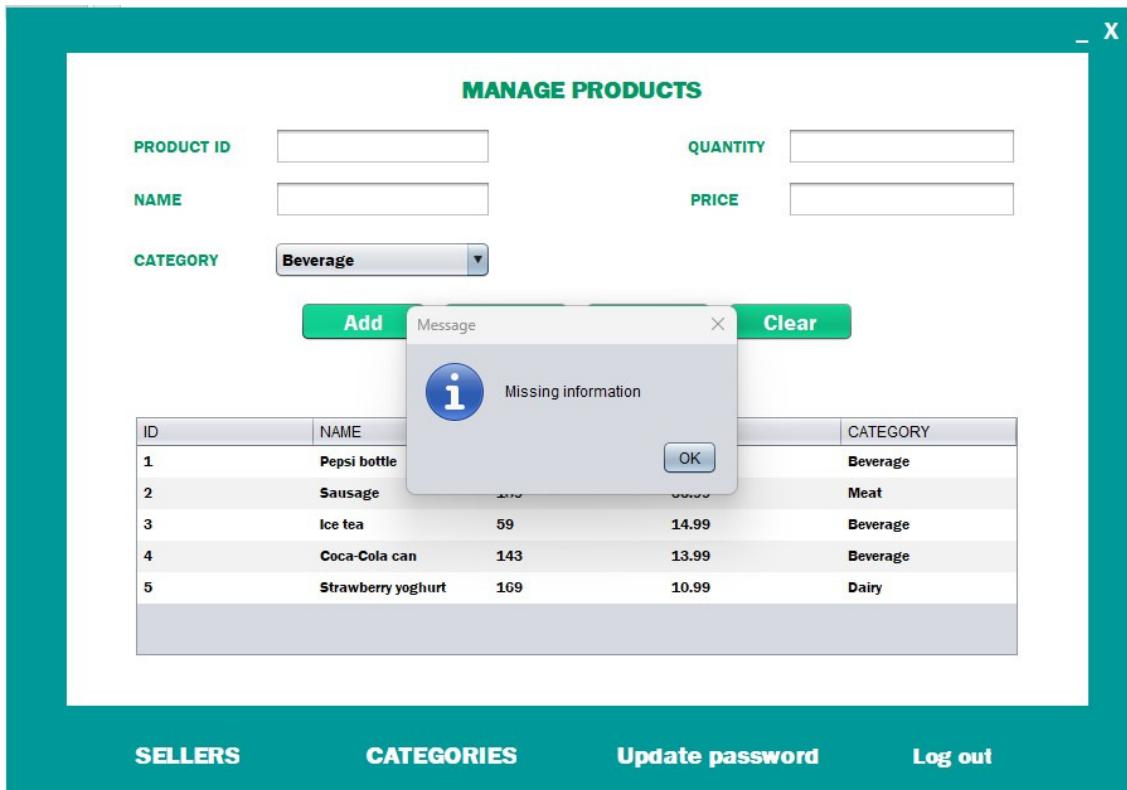
Rys. 1.31. Dodawanie produktu.



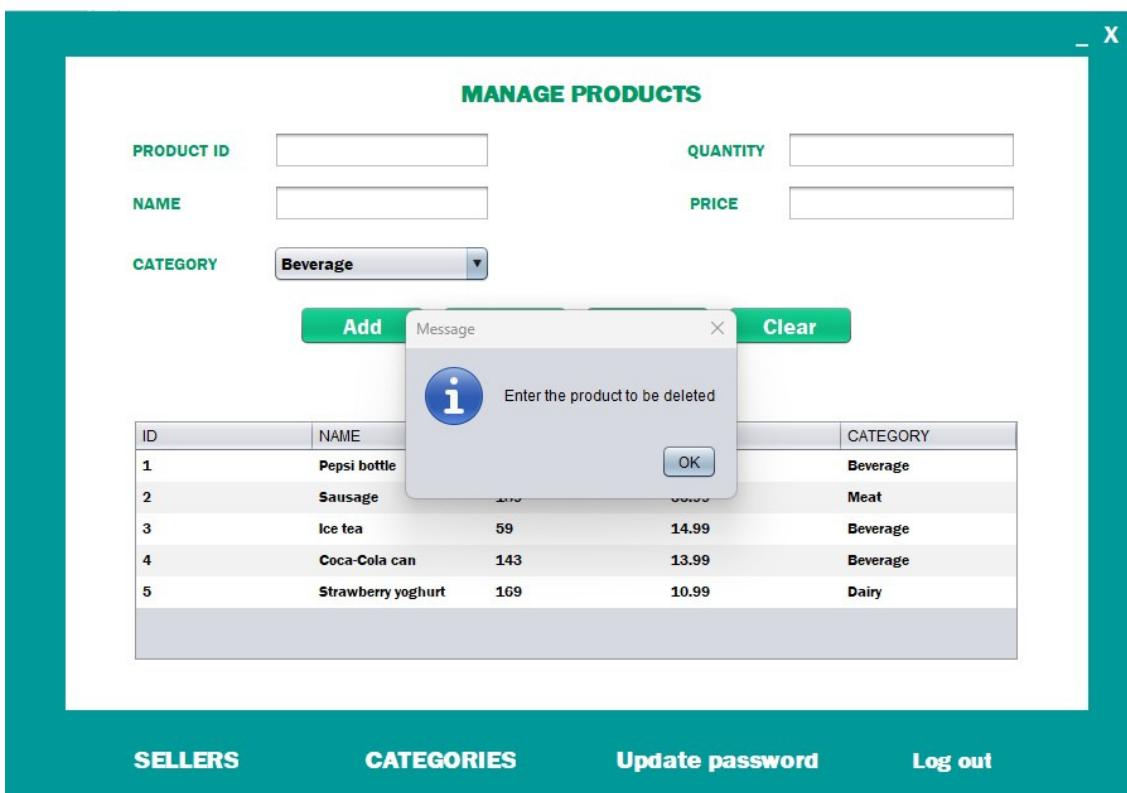
Rys. 1.32. Usuwanie produktu.



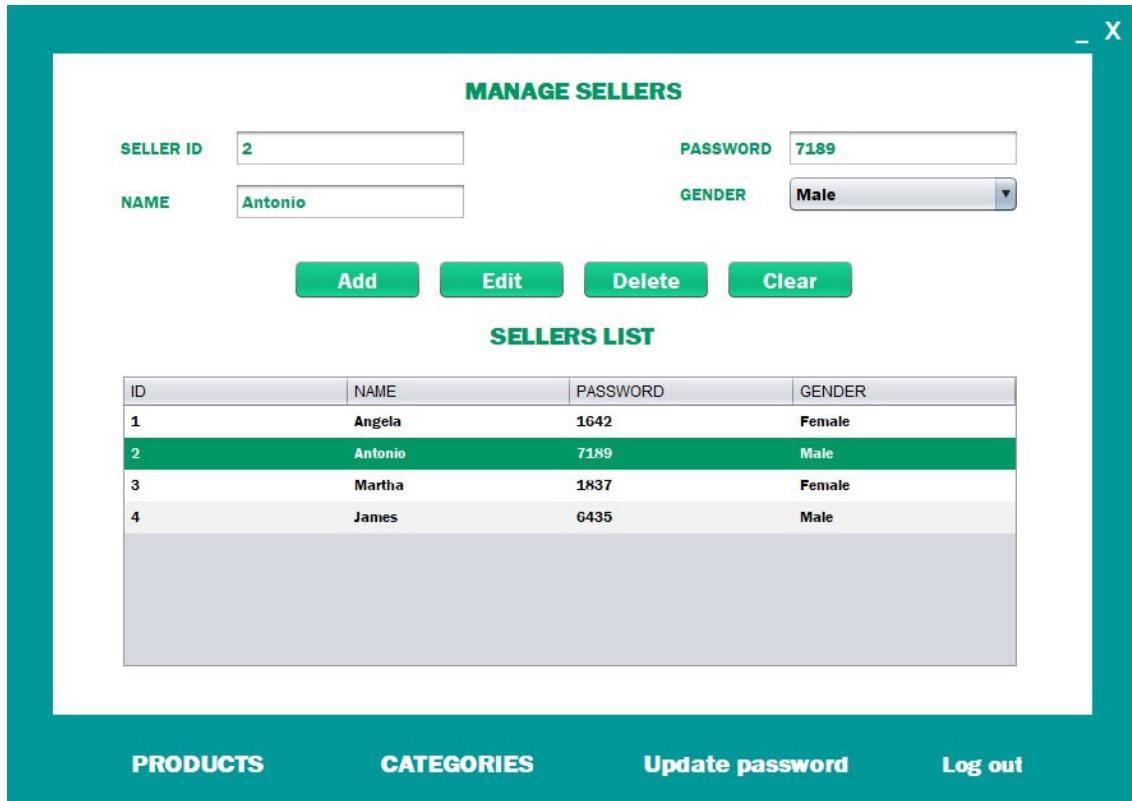
Rys. 1.33. Modyfikacja produktu.



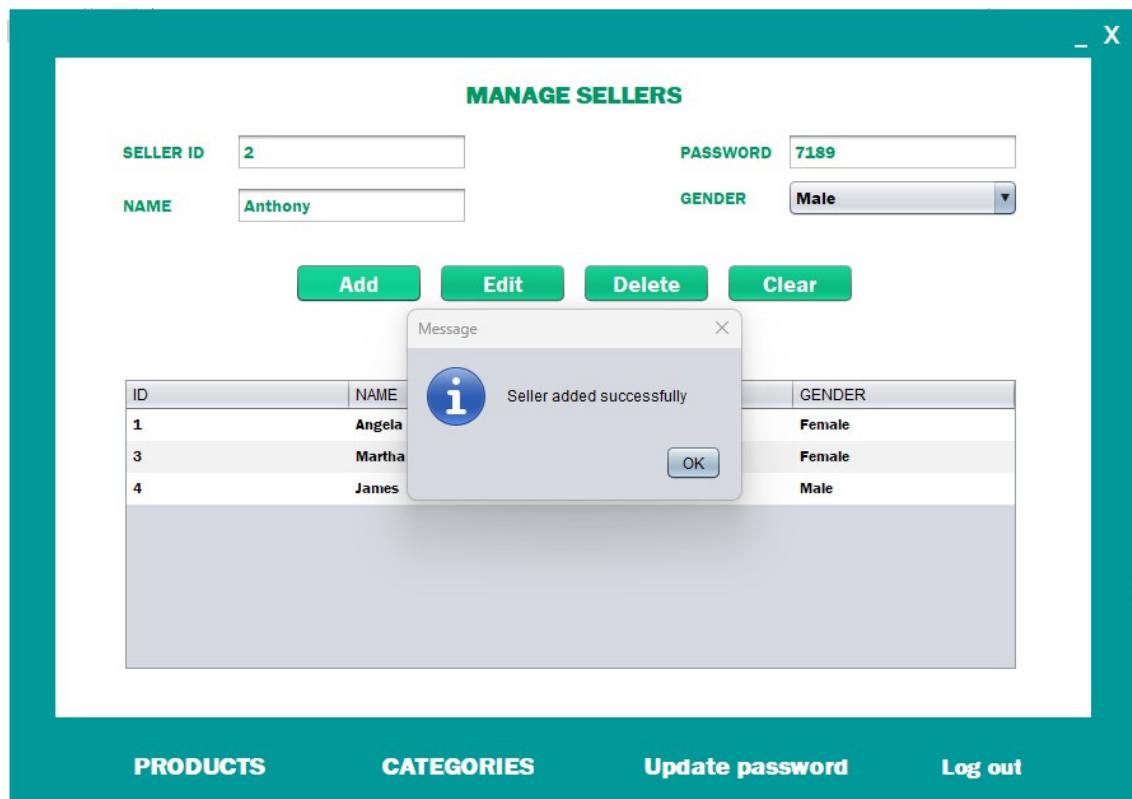
Rys. 1.34. Panel zarządzania produktami. Brak danych - brak możliwości zaktualizowania informacji o nieistniejącym produkcie.



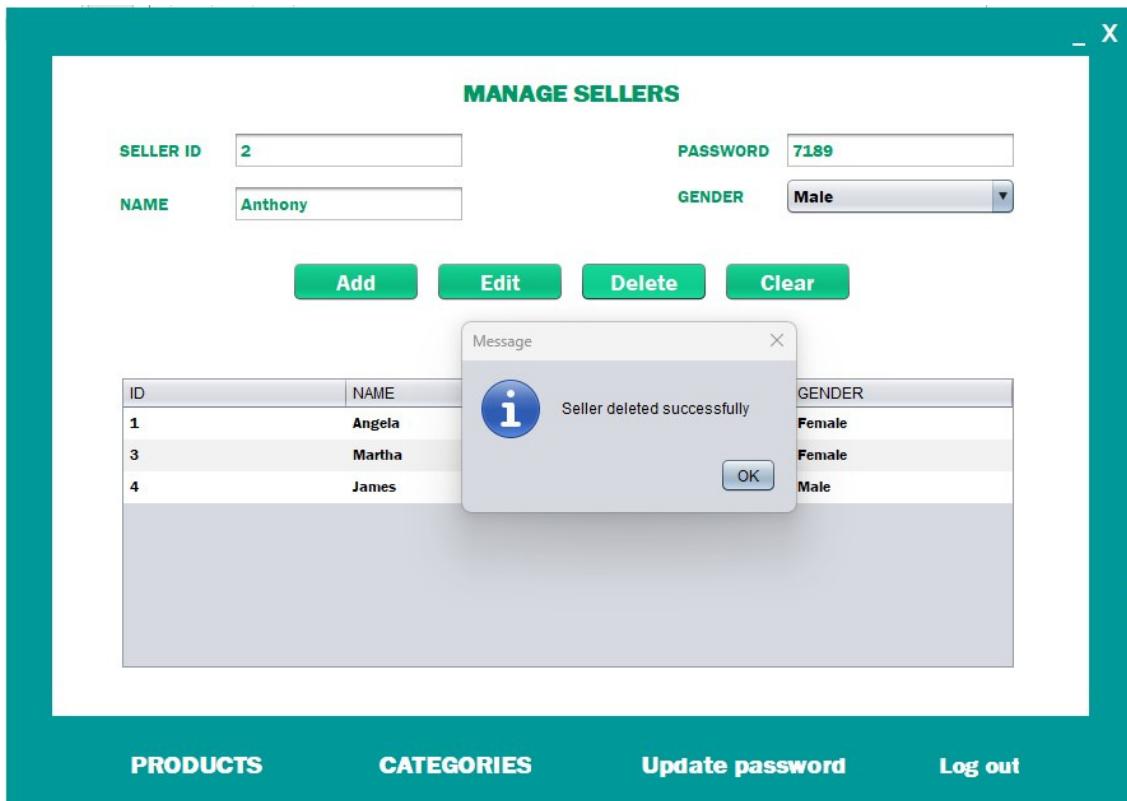
Rys. 1.35. Panel zarządzania produktami. Brak danych - brak możliwości usunięcia nieistniejącego produktu.



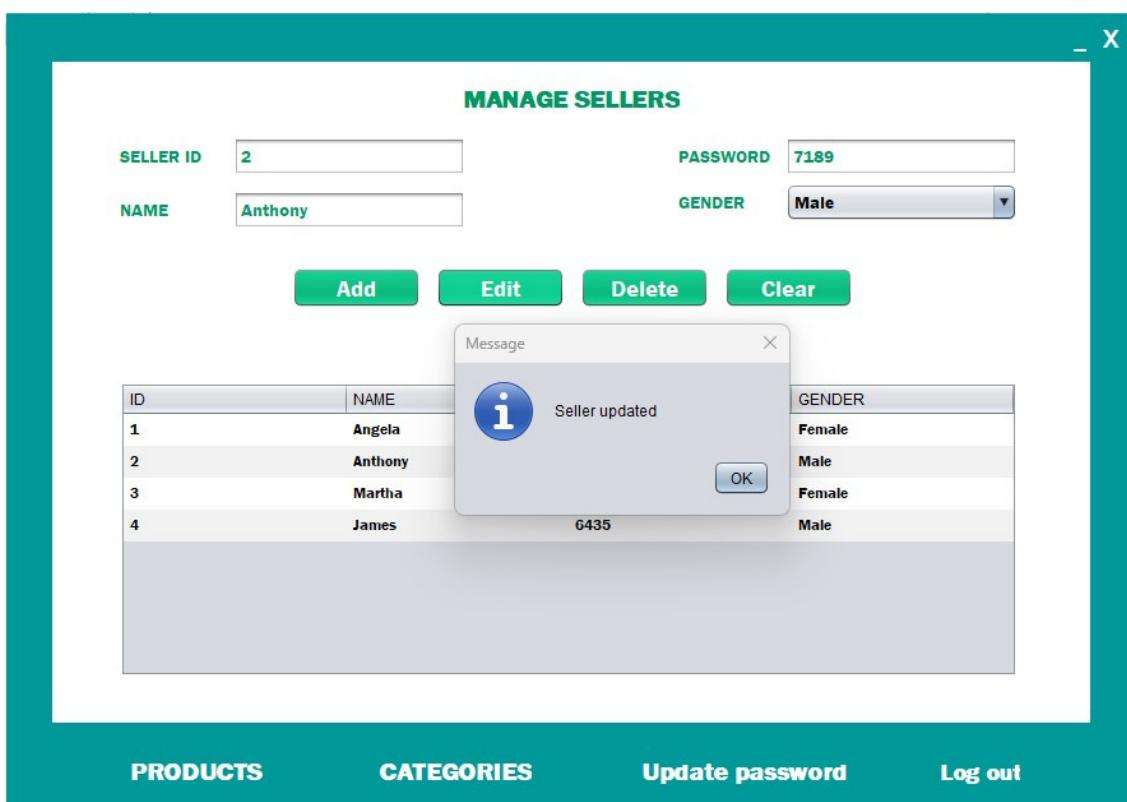
Rys. 1.36. Panel zarządzania sprzedawcami.



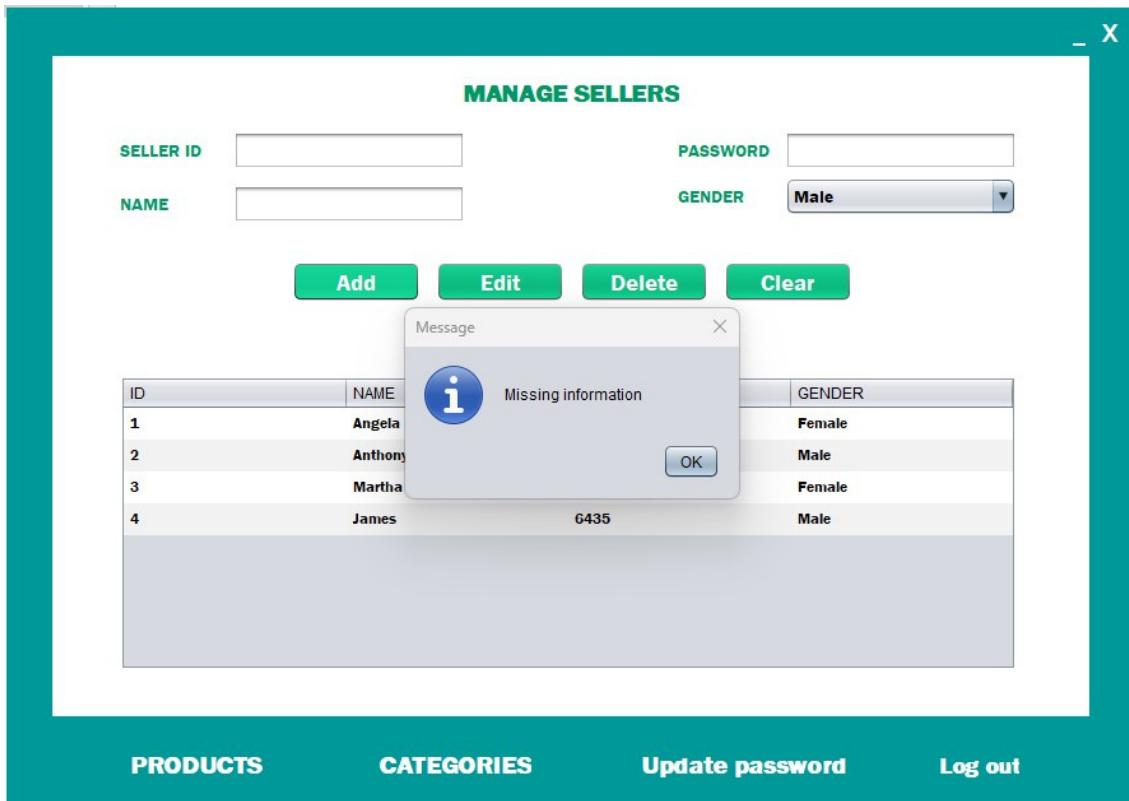
Rys. 1.37. Dodawanie sprzedawcy.



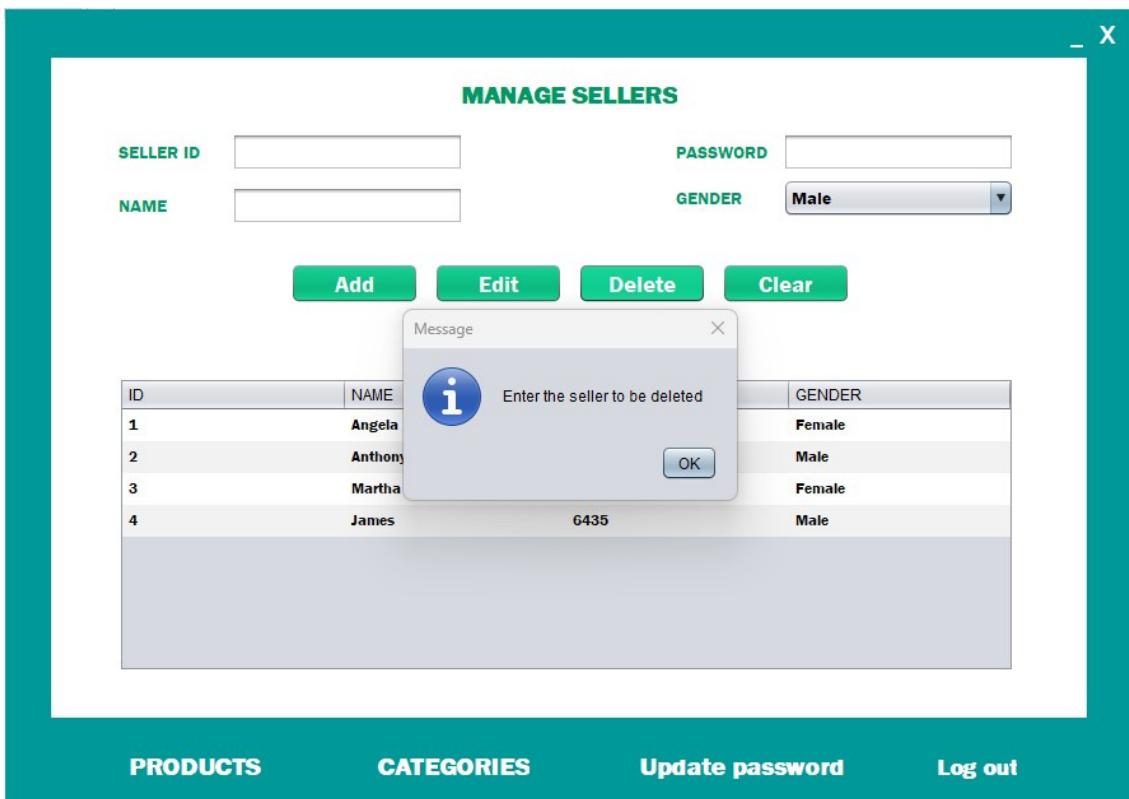
Rys. 1.38. Usuwanie sprzedawcy.



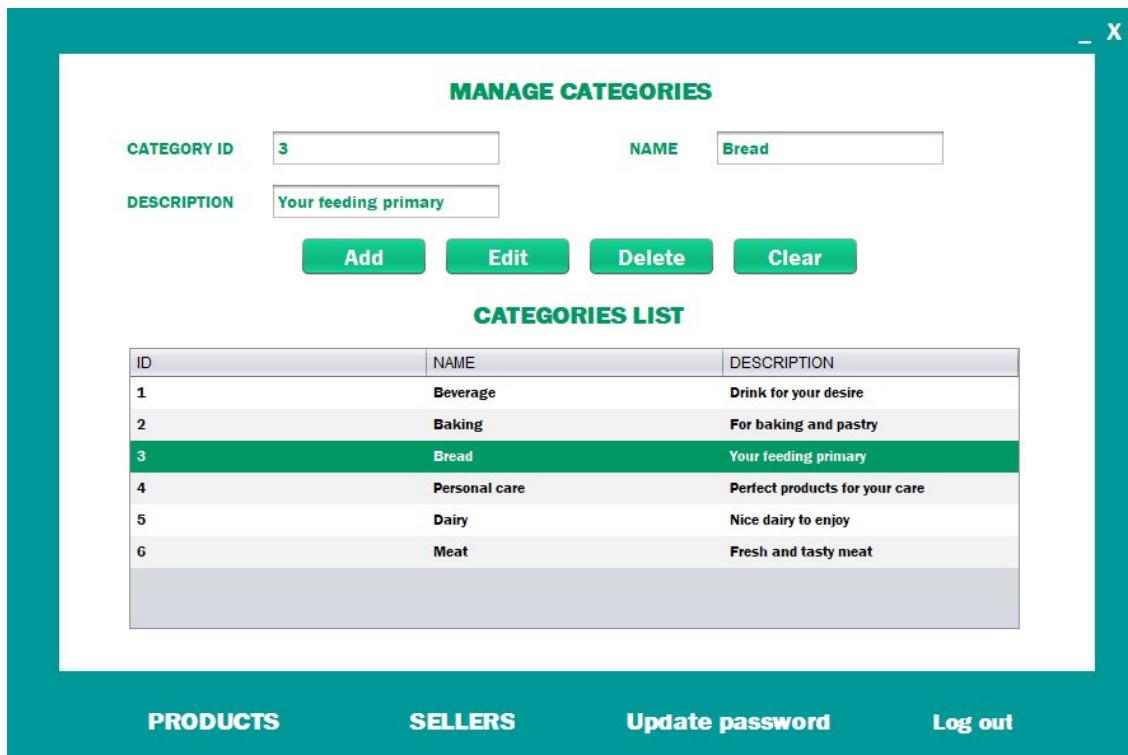
Rys. 1.39. Modyfikacja sprzedawcy.



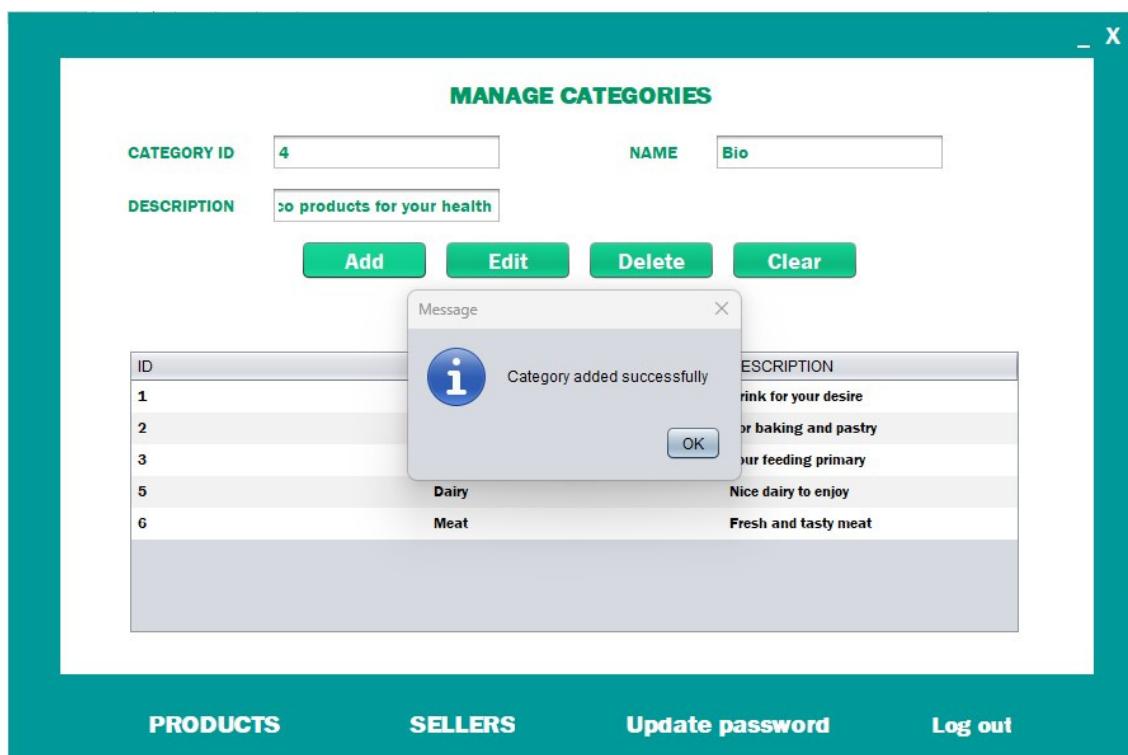
Rys. 1.40. Panel zarządzania sprzedawcami. Brak danych - brak możliwości zaktualizowania informacji o nieistniejącym sprzedawcy.



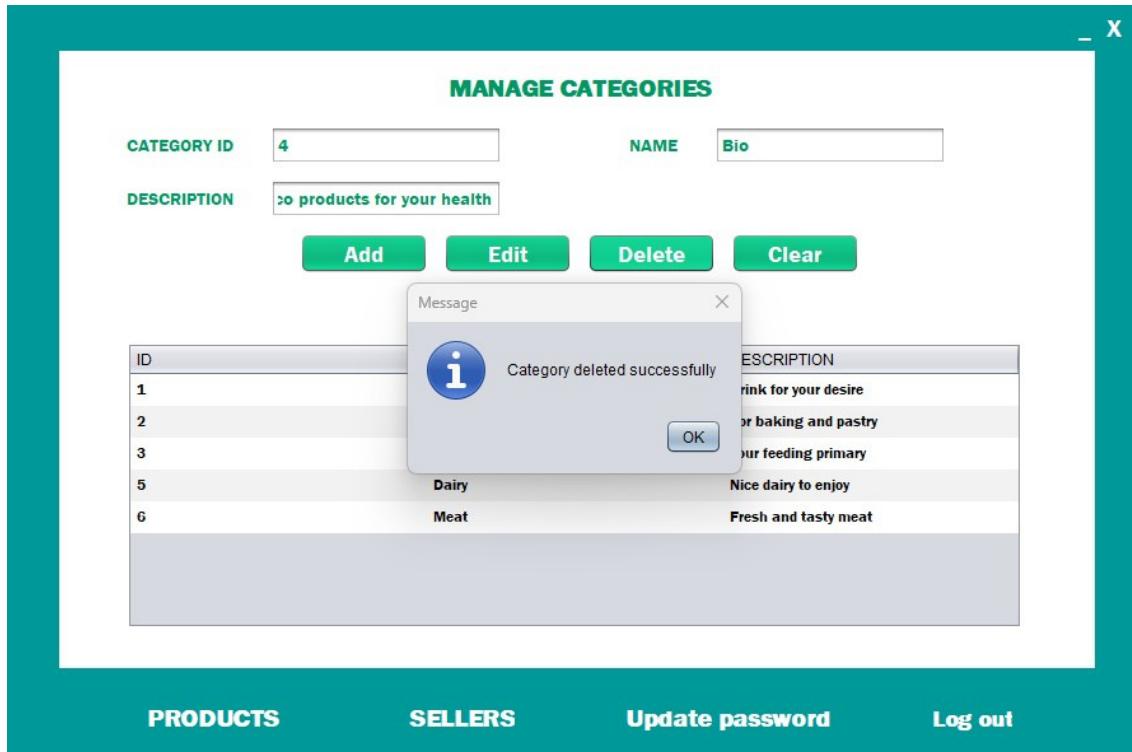
Rys. 1.41. Panel zarządzania sprzedawcami. Brak danych - brak możliwości usunięcia nieistniejącego sprzedawcy.



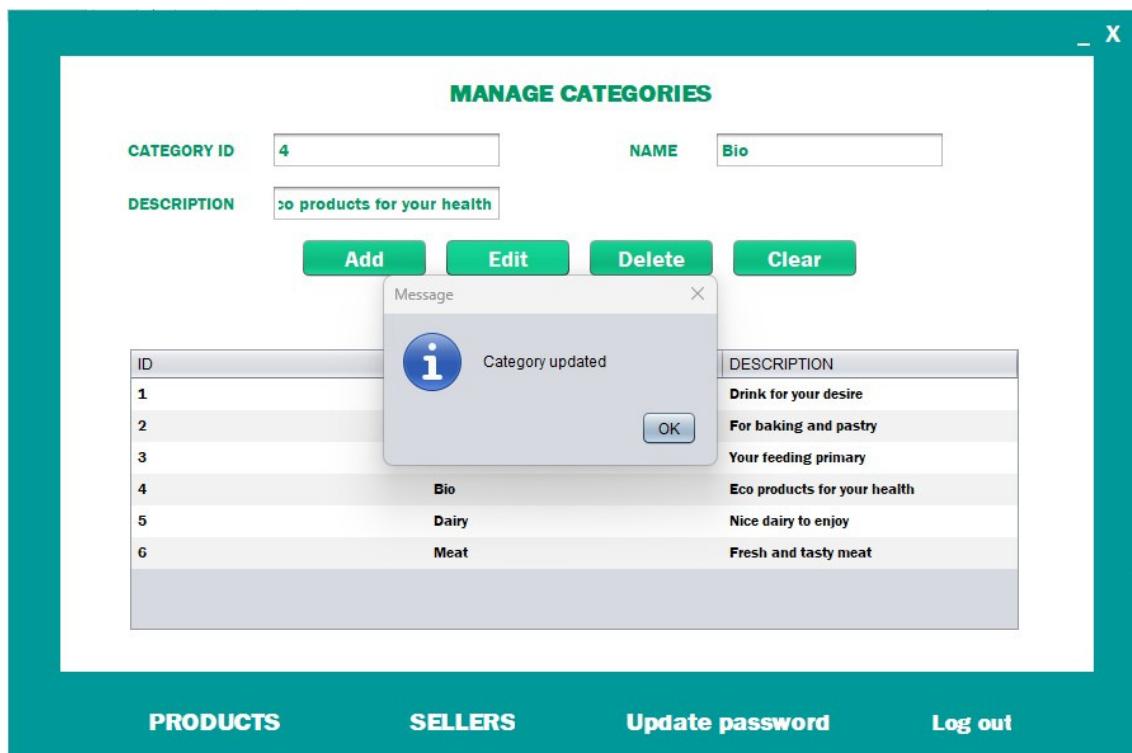
Rys. 1.42. Panel zarządzania kategoriami.



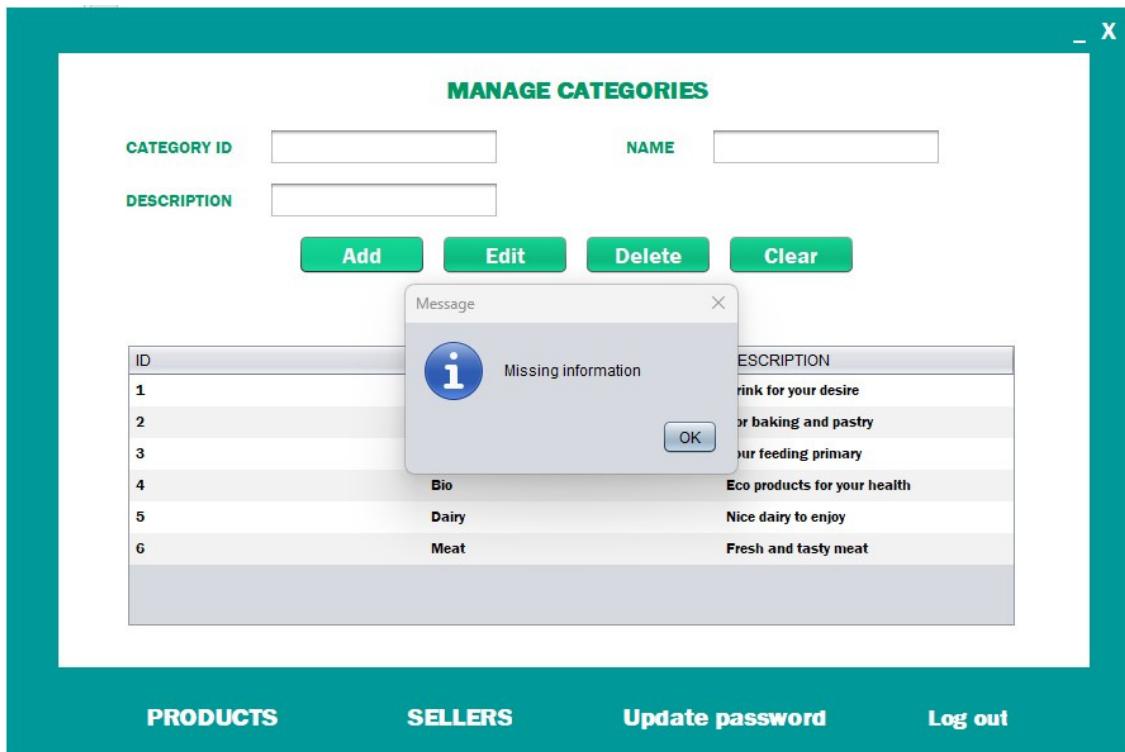
Rys. 1.43. Dodawanie kategorii.



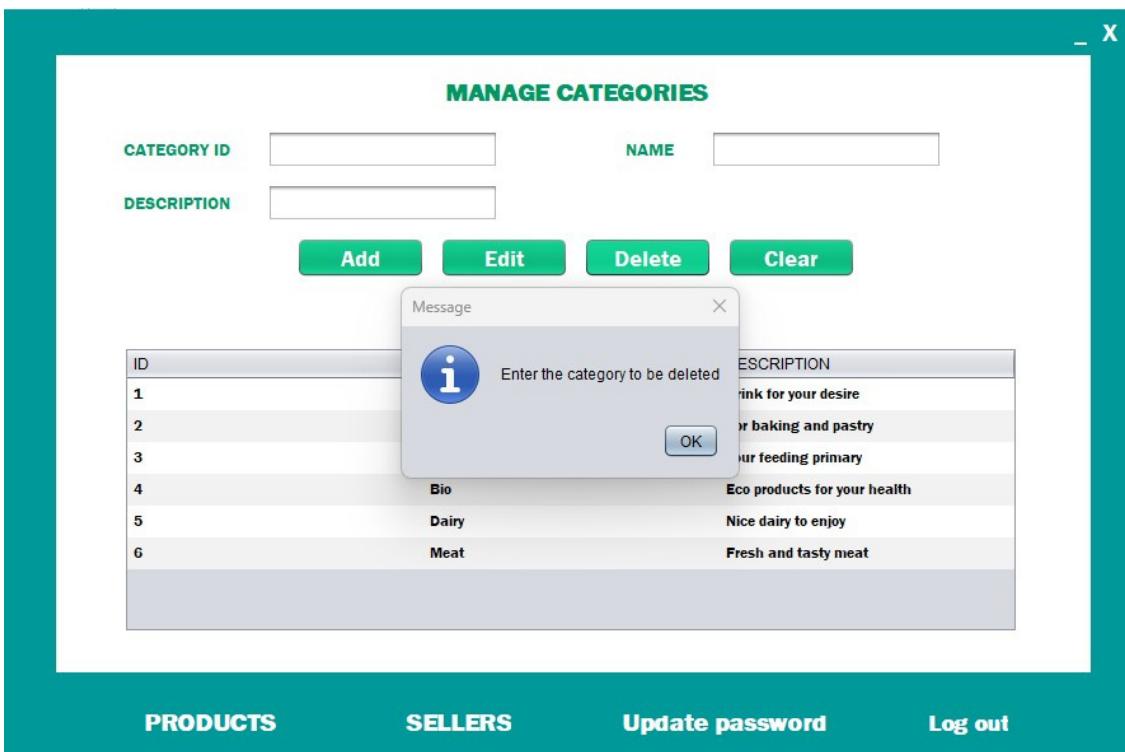
Rys. 1.44. Usuwanie kategorii.



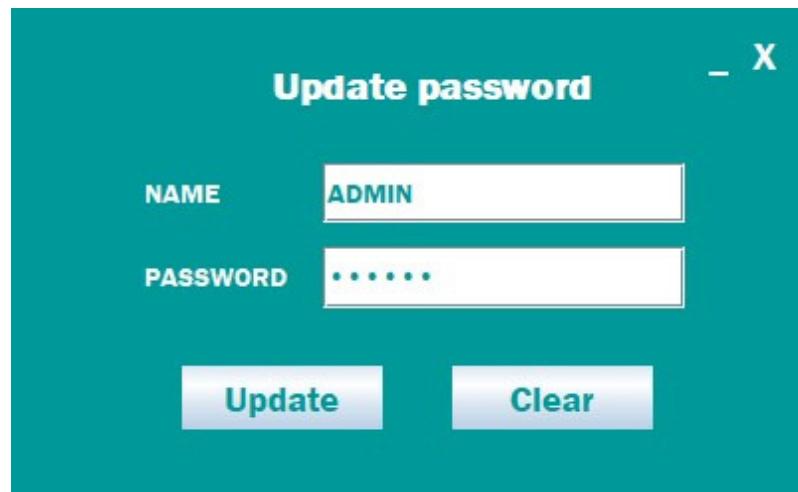
Rys. 1.45. Modyfikacja kategorii.



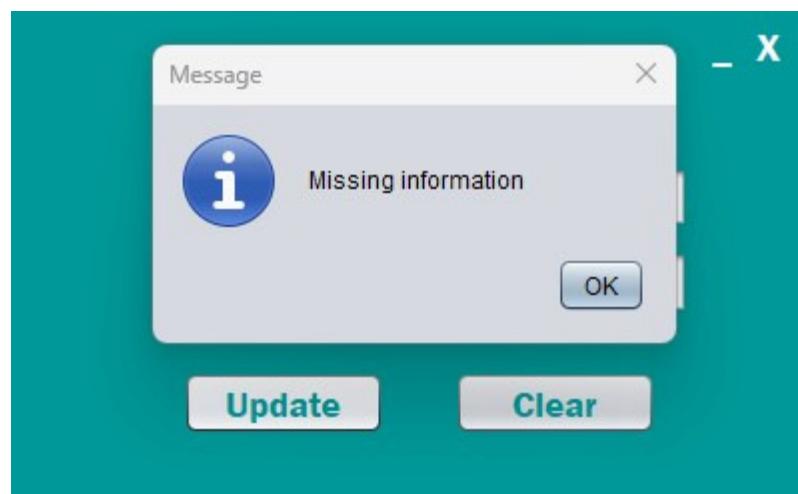
Rys. 1.46. Panel zarządzania kategoriami. Brak danych - brak możliwości zaktualizowania informacji o nieistniejącej kategorii.



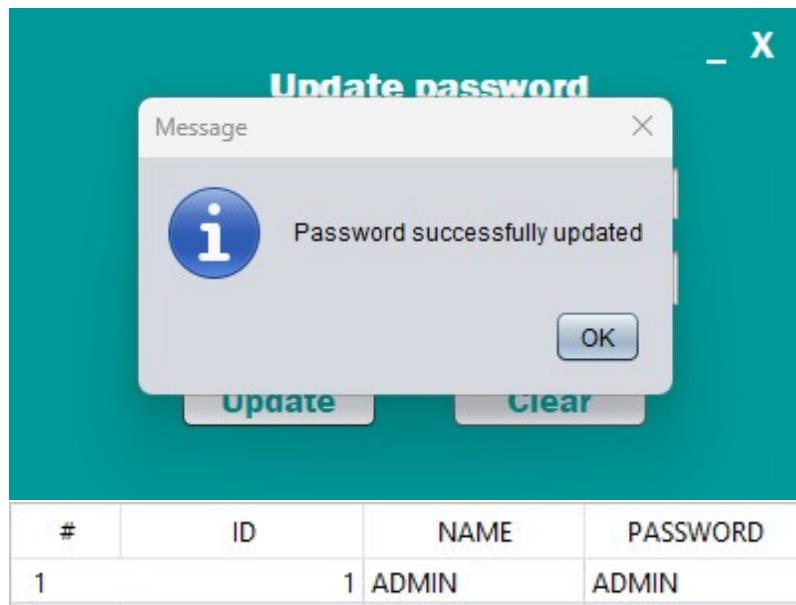
Rys. 1.47. Panel zarządzania kategoriami. Brak danych - brak możliwości usunięcia nieistniejącej kategorii.



Rys. 1.48. Panel zmiany hasła dla administratora.



Rys. 1.49. Panel zmiany hasła dla administratora. Brak danych.



Rys. 1.50. Panel zmiany hasła dla administratora. Pomyślna zmiana hasła.

## 1.7. Podsumowanie

Projekt został stworzony przy użyciu programowania zorientowanego obiektowo w języku programowania Java, interfejsu graficznego (GUI) Swing oraz połączenia między interfejsem graficznym a bazą danych z użyciem JDBC. Program spełnia podstawowe założenia, takie jak prosty i przyjazny dla użytkownika interfejs, pokazanie działania na bardzo prostym przykładzie. Przyszłościowe prace projektu będą obejmować m.in. optymalizację programu pod względem szybkości działania, łączenia ze zdalnymi bazami danych, doskonalenie jeszcze bardziej przyjaznego dla użytkownika interfejsu graficznego oraz szyfrowanie danych.

## 1.8. Oświadczenie studenta o samodzielności pracy

Załącznik nr 2 do Zarządzenia nr 228/2021 Rektora Uniwersytetu Rzeszowskiego z dnia 1 grudnia 2021 roku w sprawie ustalenia procedury antyplagiatowej w Uniwersytecie Rzeszowskim

**OŚWIADCZENIE STUDENTA O SAMODZIELNOŚCI PRACY**

..... Michał Janik .....

Imię (imiona) i nazwisko studenta

Wydział Nauk Ścisłych i Technicznych

..... Programowanie obiektowe 1 .....

Nazwa kierunku

..... 134915 .....

Numer albumu

1. Oświadczam, że moja praca projektowa pt.: Przygotowanie dokumentacji do projektu aplikacji Java z wykorzystaniem interfejsu graficznego Swing oraz połączenia bazodanowego JDBC pt. "Symulator sklepu spożywczego" w systemie  $\text{\LaTeX}$

1) została przygotowana przeze mnie samodzielnie\*,  
2) nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2021 r., poz. 1062) oraz dóbr osobistych chronionych prawem cywilnym,  
3) nie zawiera danych i informacji, które uzyskałem/am w sposób niedozwolony,  
4) nie była podstawą otrzymania oceny z innego przedmiotu na uczelni wyższej ani mnie, ani innej osobie.

2. Jednocześnie wyrażam zgodę/~~nie wyrażam zgody~~\*\* na udostępnienie mojej pracy projektowej do celów naukowo–badawczych z poszanowaniem przepisów ustawy o prawie autorskim i prawach pokrewnych.

Rzeszów, 15.06.2025r  
(miejscowość, data)

Michał Janik  
(czytelny podpis studenta)

\* Uwzględniając merytoryczny wkład prowadzącego przedmiot  
\*\* – niepotrzebne skreślić

**Rys. 1.51.** Oświadczenie studenta o samodzielności pracy, wydrukowane i ręcznie podpisane.

# Spis rysunków

1.1	Lista klas projektowanej aplikacji.	9
1.2	Diagram UML klasy Categories.	10
1.3	Diagram UML klasy Login.	15
1.4	Diagram UML klasy Products.	18
1.5	Diagram UML klasy Sellers.	23
1.6	Diagram UML klasy Selling.	28
1.7	Diagram UML klasy Splash.	33
1.8	Diagram UML klasy UpdateAdminPwd.	35
1.9	Diagram ERD bazy danych Supermarket.	37
1.10	Baza danych Supermarket i jej struktura tabel.	37
1.11	Tabela ADMIN.	38
1.12	Tabela CATEGORIES.	38
1.13	Tabela PRODUCTS.	38
1.14	Tabela SELLERS.	38
1.15	Typy danych tabeli ADMIN.	39
1.16	Typy danych tabeli CATEGORIES.	39
1.17	Typy danych tabeli PRODUCTS.	39
1.18	Typy danych tabeli SELLERS.	39
1.19	Diagram Gantta realizacji projektu.	40
1.20	Okienko załadowywania programu.	40
1.21	Panel logowania - administrator.	41
1.22	Panel logowania - administrator. Brak danych.	41
1.23	Panel logowania - administrator. Błędne dane logowania.	41
1.24	Panel logowania - sprzedawca.	42
1.25	Panel logowania - sprzedawca. Brak danych.	42
1.26	Panel logowania - sprzedawca. Błędne dane logowania.	42
1.27	Panel sprzedaży produktów.	43
1.28	Filtrowanie produktów po kategorii.	43
1.29	Możliwość wydruku wystawionego rachunku.	43
1.30	Panel zarządzania produktami.	44
1.31	Dodawanie produktu.	44
1.32	Usuwanie produktu.	45
1.33	Modyfikacja produktu.	45

1.34 Panel zarządzania produktami. Brak danych - brak możliwości zaktualizowania informacji o nieistniejącym produkcie. . . . .	46
1.35 Panel zarządzania produktami. Brak danych - brak możliwości usunięcia nieistniejącego produktu. . . . .	46
1.36 Panel zarządzania sprzedawcami. . . . .	47
1.37 Dodawanie sprzedawcy. . . . .	47
1.38 Usuwanie sprzedawcy. . . . .	48
1.39 Modyfikacja sprzedawcy. . . . .	48
1.40 Panel zarządzania sprzedawcami. Brak danych - brak możliwości zaktualizowania informacji o nieistniejącym sprzedawcy. . . . .	49
1.41 Panel zarządzania sprzedawcami. Brak danych - brak możliwości usunięcia nieistniejącego sprzedawcy. . . . .	49
1.42 Panel zarządzania kategoriami. . . . .	50
1.43 Dodawanie kategorii. . . . .	50
1.44 Usuwanie kategorii. . . . .	51
1.45 Modyfikacja kategorii. . . . .	51
1.46 Panel zarządzania kategoriami. Brak danych - brak możliwości zaktualizowania informacji o nieistniejącej kategorii. . . . .	52
1.47 Panel zarządzania kategoriami. Brak danych - brak możliwości usunięcia nieistniejącej kategorii. . . . .	52
1.48 Panel zmiany hasła dla administratora. . . . .	53
1.49 Panel zmiany hasła dla administratora. Brak danych. . . . .	53
1.50 Panel zmiany hasła dla administratora. Pomyślna zmiana hasła. . . . .	54
1.51 Oświadczenie studenta o samodzielności pracy, wydrukowane i ręcznie podpisane. . . . .	55

# Spis listingów

1.1 Konstruktor okna Categories()	11
1.2 Metoda SelectCategory()	11
1.3 Metoda AddButtonMouseClicked()	11
1.4 Metoda ClearButtonMouseClicked()	12
1.5 Metoda CategoryTableMouseClicked()	12
1.6 Metoda DeleteButtonMouseClicked()	12
1.7 Metoda EditButtonMouseClicked()	13
1.8 Metoda ExitButtonMouseClicked()	13
1.9 Metoda ProductsButtonMouseClicked()	13
1.10 Metoda SellersButtonMouseClicked()	13
1.11 Metoda LogOutButtonMouseClicked()	14
1.12 Metoda UpdatePasswordButtonMouseClicked()	14
1.13 Metoda MinimizeButtonMouseClicked()	14
1.14 Metoda uruchomieniowa okna main()	14
1.15 Konstruktor okna Login()	15
1.16 Metoda ClearButtonMouseClicked()	15
1.17 Metoda LoginButtonMouseClicked()	16
1.18 Metoda ExitButtonMouseClicked()	17
1.19 Metoda MinimizeButtonMouseClicked()	17
1.20 Metoda uruchomieniowa okna main()	17
1.21 Konstruktor okna Products()	18
1.22 Metoda AddButtonMouseClicked()	19
1.23 Metoda CategoriesButtonMouseClicked()	19
1.24 Metoda ClearButtonMouseClicked()	19
1.25 Metoda DeleteButtonMouseClicked()	20
1.26 Metoda EditButtonMouseClicked()	20
1.27 Metoda GetCat()	21
1.28 Metoda LogOutButtonMouseClicked()	21
1.29 Metoda ProductTableMouseClicked()	21
1.30 Metoda SelectProduct()	22
1.31 Metoda SellersButtonMouseClicked()	22
1.32 Metoda UpdatePasswordButtonMouseClicked()	22
1.33 Metoda ExitButtonMouseClicked()	22
1.34 Metoda MinimizeButtonMouseClicked()	22
1.35 Metoda uruchomieniowa okna main()	22
1.36 Konstruktor okna Sellers()	23
1.37 Metoda AddButtonMouseClicked()	24
1.38 Metoda CategoriesButtonMouseClicked()	24
1.39 Metoda ClearButtonMouseClicked()	24
1.40 Metoda DeleteButtonMouseClicked()	25
1.41 Metoda EditButtonMouseClicked()	25

1.42 Metoda LogOutButtonMouseClicked(). . . . .	26
1.43 Metoda ProductsButtonMouseClicked(). . . . .	26
1.44 Metoda SelectSeller(). . . . .	26
1.45 Metoda SellerTableMouseClicked(). . . . .	26
1.46 Metoda UpdatePasswordButtonMouseClicked(). . . . .	26
1.47 Metoda ExitButtonMouseClicked(). . . . .	27
1.48 Metoda MinimizeButtonMouseClicked(). . . . .	27
1.49 Metoda uruchomieniowa okna main(). . . . .	27
1.50 Konstruktor okna Selling(). . . . .	29
1.51 Metoda AddButtonMouseClicked(). . . . .	29
1.52 Metoda ClearButtonMouseClicked(). . . . .	30
1.53 Metoda FilterButtonMouseClicked(). . . . .	30
1.54 Metoda GetCat(). . . . .	30
1.55 Metoda LogOutButtonMouseClicked(). . . . .	30
1.56 Metoda PrintButtonMouseClicked(). . . . .	31
1.57 Metoda ProductTableMouseClicked(). . . . .	31
1.58 Metoda RefreshButtonMouseClicked(). . . . .	31
1.59 Metoda SelectProduct(). . . . .	31
1.60 Metoda UpdateQuantity(). . . . .	32
1.61 Metoda ExitButtonMouseClicked(). . . . .	32
1.62 Metoda MinimizeButtonMouseClicked(). . . . .	32
1.63 Metoda uruchomieniowa okna main(). . . . .	32
1.64 Konstruktor okna Splash(). . . . .	33
1.65 Metoda ExitButtonMouseClicked(). . . . .	33
1.66 Metoda MinimizeButtonMouseClicked(). . . . .	33
1.67 Metoda uruchomieniowa okna main(). . . . .	34
1.68 Konstruktor okna UpdateAdminPwd(). . . . .	35
1.69 Metoda ClearButtonMouseClicked(). . . . .	35
1.70 Metoda UpdateButtonMouseClicked(). . . . .	36
1.71 Metoda ExitButtonMouseClicked(). . . . .	36
1.72 Metoda MinimizeButtonMouseClicked(). . . . .	36
1.73 Metoda uruchomieniowa okna main(). . . . .	36