

UNIwersYTET RZESZOWSKI
WYDZIAŁ NAUK ŚCISŁYCH I TECHNICZNYCH
INSTYTUT INFORMATYKI



Bartosz Buczuliński
134894

Informatyka

Symulator sklepu internetowego

Praca projektowa

Praca wykonana pod kierunkiem
mgr inż. Dawid Kosior

Rzeszów 2025

Spis treści

1. Streszczenie w języku polskim i angielskim	6
1.1. Streszczenie w języku polskim.....	6
1.2. Streszczenie w języku angielskim.	6
2. Opis założeń projektu	7
2.1. Założenia projektu	7
2.2. Wymaganie funkcjonalne	7
2.3. Wymaganie нефункционалне	7
3. Opis struktury projektu	8
3.1. Struktura kodu projektu	8
3.1.1. Użytkownicy	8
3.1.2. Produkty	9
3.1.3. Koszyk i Zamówienia	11
3.1.4. Materiały	12
3.2. Baza danych.....	12
3.2.1. Połączenie z bazą danych.....	12
3.2.2. Struktura bazy danych.....	12
3.3. Graficzny interfejs użytkownika.....	13
3.3.1. Struktura GUI.....	13
3.3.2. Katalogi.....	14
3.3.3. Sortowanie	15
4. Harmonogram realizacji projektu	17
4.1. Diagram Ganta pracy	17
4.2. Repozytorium i system kontroli wersji.....	17
5. Prezentacja warstwy użytkowej projektu.....	18
5.1. Logowanie	18
5.2. Panel klienta i gościa	18
5.3. Panel administratora	19
6. Podsumowanie	21
Spis rysunków	22
Spis tabel	23
Spis listingów	24
A. Oświadczenie studenta o samodzielności pracy	25

1. Streszczenie w języku polskim i angielskim

1.1. Streszczenie w języku polskim.

Projekt pt. "Symulator sklepu internetowego" ma na celu stworzenie aplikacji imitującej działanie realnego sklepu internetowego specjalizującego się w sprzedaży książek w różnych formatach: fizycznych, ebooków i audiobooków. Aplikacja została napisana w języku Java z wykorzystaniem technologii takich jak Swing dla interfejsu graficznego oraz MySQL dla zarządzania bazą danych. Głównym celem projektu było odzwierciedlenie pełnego procesu obsługi klienta, od rejestracji i logowania, przez przeglądanie oferty i składanie zamówień, po zarządzanie produktami i użytkownikami przez administratora. Aplikacja uwzględnia również rolę gościa, który może przeglądać ofertę bez konieczności logowania. Projekt spełnia wymagania funkcjonalne i нефункционалне, takie jak wydajność, niezawodność i intuicyjność interfejsu.

1.2. Streszczenie w języku angielskim.

The project titled "Online Store Simulator" aims to create an application that mimics the operation of a real online store specializing in the sale of books in various formats: physical books, ebooks, and audiobooks. The application was developed in Java using technologies such as Swing for the graphical interface and MySQL for database management. The main goal of the project was to reflect the complete customer service process, from registration and login, through browsing the offer and placing orders, to product and user management by the administrator. The application also includes a guest role, allowing users to browse the offer without logging in. The project meets functional and non-functional requirements, such as performance, reliability, and interface intuitiveness.

2. Opis założeń projektu

2.1. Założenia projektu

Projekt to symulator sklepu internetowego "BookHaven" specjalizującego się w sprzedaży książek w różnych formatach: książek fizycznych, ebooków oraz audiobooków. Głównym celem aplikacji jest odzwierciedlenie pełnego, realnego procesu korzystania ze sklepu internetowego, od rejestracji i logowania, po przeglądanie oferty i składanie zamówień po stronie klienta, a także możliwość zarządzania produktami, zamówieniami i użytkownikami po stronie administratora.

2.2. Wymaganie funkcjonalne

Zarządzanie użytkownikami musi obejmować rejestrację nowych użytkowników oraz logowanie na konto. Każdy użytkownik ma przypisaną rolę, która określa jakie działania będzie mógł on podejmować w aplikacji. Administrator ma mieć możliwość usuwania istniejących użytkowników oraz dodawania nowych administratorów, natomiast po stronie klienta dostępna jest opcja aktualizacji danych potrzebnych do wysyłki.

Klient może przeglądać całą ofertę, ma dostęp do filtrowania oraz sortowania produktów w ofercie oraz dodawania ich do koszyka, w którym można przeglądać wybrane produkty, usunąć je z niego, oraz złożyć zamówienie.

Administrator ma mieć narzędzia pozwalające na zarządzanie produktami: dodawanie, edycja i usuwanie.

Proste zarządzanie zamówieniami również musi odbywać się po stronie administratora. Dostępne ma być przeglądanie wszystkich zamówień, gdzie widnieją dane zamawiającego oraz produkty które zamówił, dodatkowo, bezpośrednio w aplikacji, administrator ma możliwość zmiany statusu zamówienia.

Aplikacja obsługuje możliwość zalogowania się jako gość, gdzie nie ma potrzeby logowania, natomiast ma on mieć wówczas dostęp jedynie do przeglądania oferty.

2.3. Wymaganie niefunkcjonalne

Istotna jest wydajność aplikacji, krótki czas ładowania poszczególnych stron aplikacji, a także bezproblemowa obsługa rozbudowanej bazy produktów i użytkowników.

Niezawodność projektu powinna być zapewniona przez obsługę wyjątków dla, możliwie wszystkich, funkcjonalności aplikacji oraz informowanie użytkownika o jego błędnych działaniach.

Graficzny interfejs użytkownika musi być prosty i intuicyjny. Poszczególne panele aplikacji powinny być przejrzyste, tak aby klienci, goście i administratorzy nie mieli problemu z nawigowaniem po nich. Oprócz tego interfejs graficzny ma wyglądać estetycznie, każda strona aplikacji nie powinna odbiegać wizualnie od pozostałych dzięki zastosowaniu wszędzie jednolitego białego tła oraz banera w kolorze BA2F33 z nazwą sklepu.

Możliwa ma być również skalowalność aplikacji, obejmująca zwiększającą się bazę produktów, użytkowników i zamówień, a także możliwość późniejszej rozbudowy systemu o dodatkowe funkcjonalności, między innymi dodanie nowych metod płatności lub implementacja przechowywania produktów z koszyka w bazie.

3. Opis struktury projektu

3.1. Struktura kodu projektu

Projekt symulatora sklepu internetowego napisany został w języku obiektowym Java, zatem do jego uruchomienia niezbędne będzie oprogramowanie Java w wersji 11+. Projekt podzielony został na sześć pakietów:

- userPCG,
- productPCG,
- orderPCG,
- shoppingCartPCG,
- GUI,
- figures,
- database.

3.1.1. Użytkownicy

Zarządzanie użytkownikami, klasy i metody im poświęcone znajdują się w pakiecie userPCG. Klasa User zawiera pola informujące o danych użytkownika, konstruktory oraz podstawowe metody, takie jak gettery i settery. Wewnątrz klasy userServices znajdują się kluczowe metody do zarządzania użytkownikami w panelach klienta, jak i administratora, takie jak np. metoda do rejestrowania nowego użytkownika, przedstawiona na Listing 3.1.

Listing 3.1. Metoda do rejestrowania nowych użytkowników

```
1 //Rejestracja nowego użytkownika
2 public void registerUser(User user) throws SQLException {
3     // Walidacja przed rejestracją
4     if (!isValidEmail(user.getEmail())) {
5         throw new IllegalArgumentException("Nieprawidłowy format email");
6     }
7
8     if (usernameExists(user.getUsername())) {
9         throw new IllegalArgumentException("Nazwa użytkownika jest już zajęta");
10    }
11
12    if (emailExists(user.getEmail())) {
13        throw new IllegalArgumentException("Ten email jest już zarejestrowany");
14    }
15
16    String sql = "INSERT INTO users (username, email, password) VALUES (?, ?, ?)";
17
18    try (Connection connection = DBConnection.getConnection();
19         PreparedStatement statement = connection.prepareStatement(sql)) {
20        statement.setString(1, user.getUsername());
```

```

21         statement.setString(2, user.getEmail());
22         statement.setString(3, user.getPassword());
23         statement.executeUpdate();
24     }
25 }

```

Natomiast zarządzaniem aktualnie zalogowanym użytkownikiem zajmuje się klasa `UserSession`. Każdy typ użytkownika ma przypisaną rolę, której odpowiada wartość liczbową, co zaprezentowane jest w Tabeli 3.1.

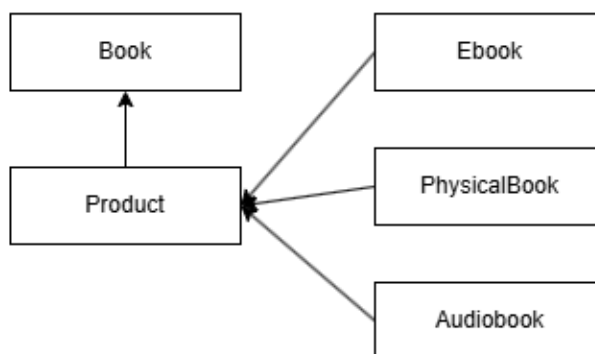
Tabela 3.1. Role użytkowników

rola	wartość liczbową
klient	1
administrator	2
gość	3

Poszczególne typy użytkowników mają dostęp do różnych działań w aplikacji.

3.1.2. Produkty

Wszystkie potrzebne informacje i metody do zarządzania produktami w sklepie zawarte są w pakiecie `productPCG`. Każdy typ sprzedawanego produktu, czyli książki fizyczne, ebooki i audiobooki posiadają własne klasy (`PhysicalBook`, `Ebook`, `Audiobook`), które dziedziczą po klasie abstrakcyjnej `Book`, która to z kolei implementuje interfejs `Product`, co przedstawia Rys. 3.1. Diagram został sporządzony na stronie: <https://www.drawio.com/>.



Rys. 3.1. Diagram dziedziczenia klas produktów.

Pakiet zawiera również enum `BookCategory`, w którym przechowywane są kategorie sprzedawanych książek. Enum przedstawiono na Listing 3.2.

Listing 3.2. Enum kategorii książek

```

1 public enum BookCategory {
2     BIOGRAPHY("Biografia"),
3     FANTASY("Fantastyka"),

```

```

4     SCIENCE_FICTION("Science Fiction"),
5     HISTORICAL("Historyczne"),
6     CRIME("Kryminały"),
7     CHILDREN("Dla dzieci"),
8     LANGUAGE_LEARNING("Nauka języków"),
9     SCIENTIFIC("Naukowe"),
10    POPULAR_SCIENCE("Popularnonaukowe"),
11    CONTEMPORARY("Obyczajowe"),
12    LITERATURE("Literatura piękna"),
13    OTHER("Inne");
14
15    private final String polishName;
16
17    BookCategory(String polishName) {
18        this.polishName = polishName;
19    }
20
21    @Override
22    public String toString() {
23        return polishName;
24    }
25 }

```

Kluczowe metody do zarządzania produktami znajdują się w klasie `ProductServices`. Zawiera między innymi metody do dodawania, usuwania, pobierania, a także metody aktualizowania danych podstawowych produktu (`Book`) oraz pochodne metody do aktualizowania danych specyficznych dla konkretnych typów produktów (`PhysicalBook`, `Ebook`, `Audiobook`), których przykłady podano w Listing 3.3 i Listing 3.4.

Listing 3.3. Metoda aktualizująca podstawowe dane produktu

```

1     public static boolean updateBaseProduct(Book book) throws SQLException {
2         String sql = "UPDATE products SET title = ?, author = ?, price = ?, category_id
3         = ? WHERE id = ?";
4
5         try (Connection conn = DBConnection.getConnection();
6             PreparedStatement stmt = conn.prepareStatement(sql)) {
7
8             stmt.setString(1, book.getTitle());
9             stmt.setString(2, book.getAuthor());
10            stmt.setDouble(3, book.getPrice());
11            stmt.setInt(4, getCategoryId(book.getCategory()));
12            stmt.setString(5, book.getId());
13
14            int rowsAffected = stmt.executeUpdate();
15            return rowsAffected > 0;
16        }
17    }

```

Listing 3.4. Metoda aktualizująca specyficzne dane dla Ebooka

```

1     public static boolean updateEbook(Ebook ebook) throws SQLException {
2         String sql = "UPDATE ebooks SET file_format = ?, file_size_mb = ?, download_link
3         = ? WHERE product_id = ?";
4
5         try (Connection conn = DBConnection.getConnection();
6             PreparedStatement stmt = conn.prepareStatement(sql)) {

```



```
6
7         stmt.setString(1, ebook.getFileFormat());
8         stmt.setDouble(2, ebook.getFileSizeMB());
9         stmt.setString(3, ebook.getDownloadLink());
10        stmt.setString(4, ebook.getId());
11
12        int rowsAffected = stmt.executeUpdate();
13        return rowsAffected > 0;
14    }
15 }
```

Rozbudowany unikalny identyfikator produktu jest generowany automatycznie dzięki użyciu metody `randomUUID()` ze standardowej klasy `UUID`.

Wewnątrz klasy `ProductServices` umieszczona została dodatkowo klasa wewnętrzna `ProductInfo`, która zawiera konkretne pola ułatwiające przechowywanie informacji o produktach, np. w katalogach i koszyku. Zaprezentowana została ona w Listing 3.5.

Listing 3.5. Wewnętrzna klasa `ProductInfo`

```
1  public static class ProductInfo {
2      private String id;
3      private String title;
4      private String author;
5      private String productType;
6      private String categoryName;
7      private double price;
8
9      public ProductInfo(String id, String title, String author, String productType,
10                          String categoryName, double price) {
11          this.id = id;
12          this.title = title;
13          this.author = author;
14          this.productType = productType;
15          this.categoryName = categoryName;
16          this.price = price;
17      }
18
19      public String getId() { return id; }
20      public String getTitle() { return title; }
21      public String getAuthor() { return author; }
22      public String getProductType() { return productType; }
23      public String getCategoryName() { return categoryName; }
24      public double getPrice() { return price; }
25 }
```

3.1.3. Koszyk i Zamówienia

Obsługa zamówień odbywa się przy pomocy klasy `OrderServices` w pakiecie `orderPCG`. Oprócz metody do tworzenia nowego zamówienia i aktualizacji statusu zamówienia, dostępne są również metody tworzące listy wszystkich zamówień wykorzystujące wewnętrzne klasy `OrderInfo`, z polami dotyczącymi danych samego zamówienia, a także `OrderItemInfo`, z polami produktów w zamówieniu.

Koszyk z produktami działający przy pomocy klasy `ShoppingCart` zawiera listę produktów, która później trafia do zamówienia, a także metody do dodawania i usuwania produktów z niego.

3.1.4. Materiały

Jedynym zewnętrznym materiałem wykorzystywanym w aplikacji jest `book.png`, przechowywany w `figures`, materiał ten służy jako logo, wyświetlane w banerze na każdej stronie sklepu. Logo to, przedstawione na Rys. 3.2, pobrane zostało z <https://icons8.com/icons/set/book>.



Rys. 3.2. obrazek książki wykorzystany w aplikacji.

3.2. Baza danych

W symulatorze sklepu internetowego, dane przechowywane i zarządzane są za pomocą relacyjnej bazy danych MySQL.

3.2.1. Połączenie z bazą danych

Połączenie z bazą danych odbywa się za pomocą interfejsu API, JDBC. Baza danych testowo przechowywana jest na serwerze pakietu XAMPP, natomiast połączenie z serwerem przeprowadzone zostało poprzez implementację do projektu `Connector/J`, dostępnego na <https://dev.mysql.com/downloads/connector/j/>.

Celem łatwego łączenia się z bazą danych utworzona została klasa `DBConnection`, przedstawiona w Listing 3.6.

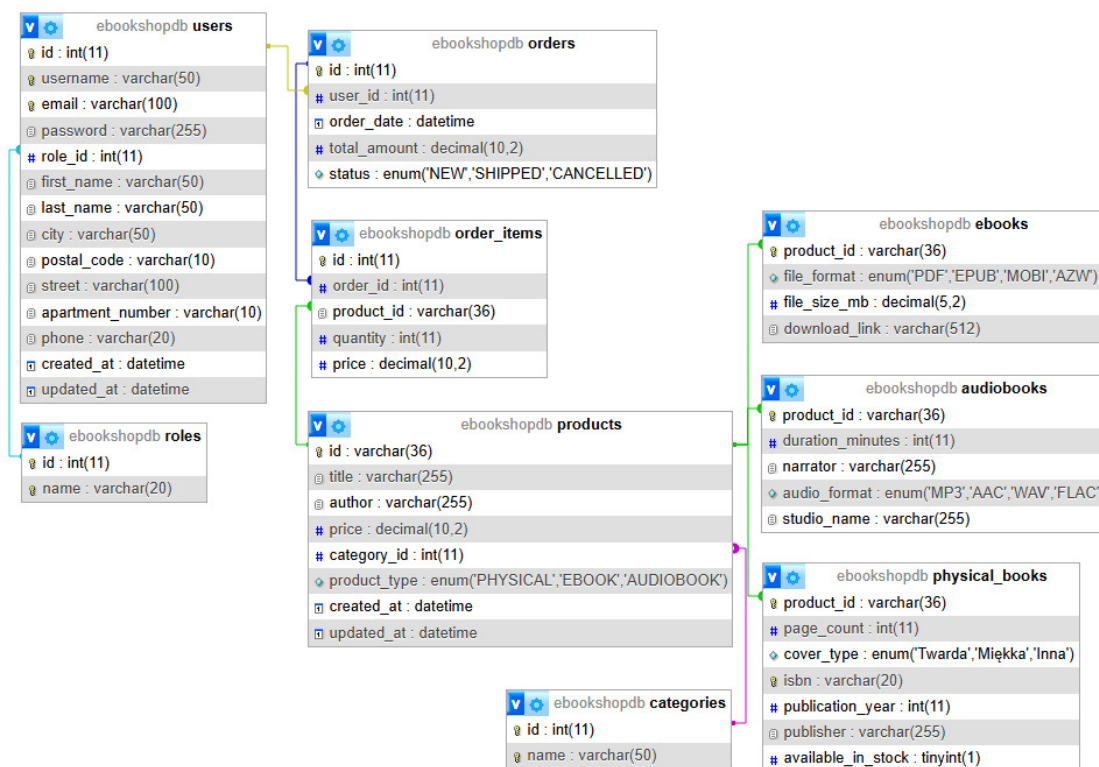
Listing 3.6. Klasa łączenia się z bazą danych

```
1 package database;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
6
7 public class DBConnection {
8     private static final String URL = "jdbc:mysql://localhost:3306/ebookshopdb";
9     private static final String USER = "root";
10    private static final String PASSWORD = "";
11
12    //Nawiązywanie połączenia z bazą danych
13    public static Connection getConnection() throws SQLException {
14        return DriverManager.getConnection(URL, USER, PASSWORD);
15    }
16 }
```

3.2.2. Struktura bazy danych

Struktura bazy, zaprojektowana została w taki sposób, aby umożliwić łatwe zarządzanie informacjami o użytkownikach, produktach oraz zamówieniach. Baza danych składa się z tabel głównych `users`,

products i orders, a także tabel physical book, ebooks, audiobooks z danymi specyficznymi dla konkretnych typów książek, tabeli order items, zawierającej dane o produktach z zamówień oraz tabel pomocniczych categories i roles. Diagram ERD przedstawiono na Rys. 3.3.



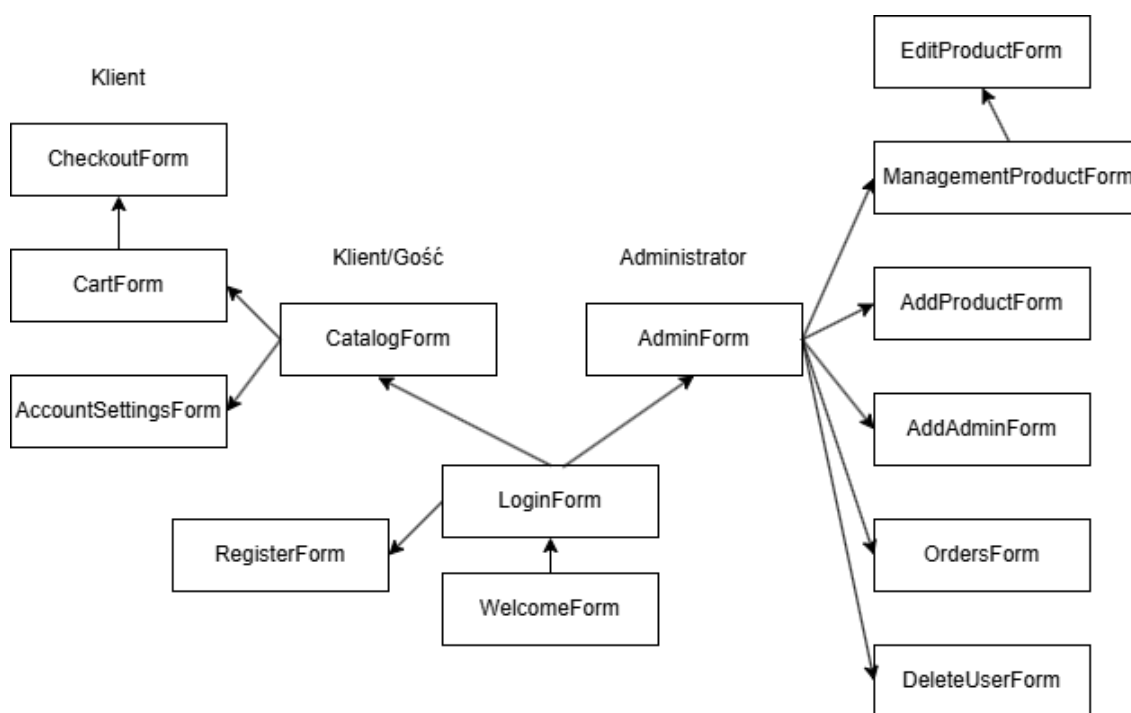
Rys. 3.3. Diagram ERD bazy danych.

3.3. Graficzny interfejs użytkownika

3.3.1. Struktura GUI

Graficzny interfejs użytkownika został stworzony w taki sposób, aby był przejrzysty, prosty w obsłudze, a także estetyczny wizualnie. Do tworzenia GUI wykorzystana została biblioteka SWING, a do samego projektowania, rozszerzenie SWING UI DESIGNER.

Wszystkie elementy interfejsu graficznego przechowywane są w pakiecie GUI, podzielone są one na okna dostępne dla klientów oraz gości, a także okna administratora, dostępne w zależności od tego na konto, z jaką rolą, zaloguje się użytkownik. Struktura GUI zaprezentowana została na diagramie na Rys. 3.4. Diagram utworzony został przy użyciu <https://www.drawio.com/>.



Rys. 3.4. Diagram kolejnych okien aplikacji.

3.3.2. Katalogi

W sklepie internetowym często występują katalogi, takie jak np. katalog produktów i katalog zamówień. Jest to bardzo ważny element aplikacji, zatem konieczne jest aby były one czytelne oraz proste w korzystaniu z nich.

W projekcie wykorzystano metodę tworzenia katalogów, za pomocą dodawania do `JScrollPane` co-raz to kolejnych paneli, zawierających dane na temat produktu (w przypadku katalogu produktów), a także przyciski funkcyjne do np. dodawania przedmiotów do koszyka lub usuwania użytkowników (w przypadku panelu do usuwania użytkowników przez administratora). Wewnętrzną klasę paneli produktów prezentuje Listing 3.7.

Listing 3.7. Klasa paneli z produktami

```

1  private class ProductItemPanel extends JPanel {
2      public ProductItemPanel(ProductInfo product) {
3          setLayout(new BorderLayout(10, 10));
4          setBorder(BorderFactory.createCompoundBorder(
5              BorderFactory.createEmptyBorder(5, 5, 5, 5),
6              BorderFactory.createEtchedBorder()
7          ));
8          setMaximumSize(new Dimension(Integer.MAX_VALUE, 100));
9
10         // Panel informacji
11         JPanel infoPanel = new JPanel(new GridLayout(0, 2, 5, 5));
12         infoPanel.add(new JLabel("Tytuł:"));
13         infoPanel.add(new JLabel(product.getTitle()));
14         infoPanel.add(new JLabel("Autor:"));
15         infoPanel.add(new JLabel(product.getAuthor()));
16         infoPanel.add(new JLabel("Kategoria:"));
17         infoPanel.add(new JLabel(BookCategory.valueOf(product.getCategoryName()).
18             toString()));
  
```

```

18         infoPanel.add(new JLabel("Cena:"));
19         infoPanel.add(new JLabel(String.format("%.2f zł", product.getPrice())));
20
21         // Przyciski "Dodaj do koszyka"
22         JButton addToCartButton = new JButton("Dodaj do koszyka");
23         addToCartButton.addActionListener(new ActionListener() {
24             @Override
25             public void actionPerformed(ActionEvent e) {
26                 if (roleId == 3) {
27                     JOptionPane.showMessageDialog(null,
28                         "Gość ma dostęp jedynie do przeglądania oferty.",
29                         "Zalogowano jako gość",
30                         JOptionPane.INFORMATION_MESSAGE);
31                 }
32                 else {
33                     ShoppingCart.getInstance().addItem(product);
34                     JOptionPane.showMessageDialog(null,
35                         "Produkt dodany do koszyka: " + product.getTitle(),
36                         "Sukces",
37                         JOptionPane.INFORMATION_MESSAGE);
38                 }
39             }
40         });
41
42         add(infoPanel, BorderLayout.CENTER);
43         add(addToCartButton, BorderLayout.EAST);
44     }
45 }

```

3.3.3. Sortowanie

W tym przypadku katalogu produktów istotna jest również możliwość sortowania i filtrowania listy produktów. W tym celu wykorzystane zostały standardowe pakiety i klasy Javy takie jak np. `Collectors`. Metoda `filterProducts` wykorzystuje strumień `stream` do filtrowania produktów na podstawie wybranego typu i kategorii, a następnie zbiera wyniki z powrotem do listy za pomocą Listing 3.8. Sortowanie natomiast odbywa się za pomocą metody `sort`, wywoływanej dla `mutableList`. Rys. 4.1 przedstawia implementację tych funkcjonalności.

Listing 3.8. Metody filtrowania oraz sortowania

```

1     //Filtrowanie produktów w panelu
2     private List<ProductInfo> filterProducts(List<ProductInfo> products) {
3         String selectedType = (String) productTypeCombo.getSelectedItem();
4         String selectedCategory = (String) categoryCombo.getSelectedItem();
5
6         return products.stream()
7             .filter(p -> selectedType.equals("Wszystkie typy") ||
8                 (selectedType.equals("Książki fizyczne") && p.getProductType().
9                     equals("PHYSICAL")) ||
10                    (selectedType.equals("E-booki") && p.getProductType().equals("
11                        EBOOK")) ||
12                    (selectedType.equals("Audiobooki") && p.getProductType().equals("
13                        AUDIOBOOK")))
14             .filter(p -> selectedCategory.equals("Wszystkie kategorie") ||
15                 BookCategory.valueOf(p.getCategoryName()).toString().equals(
16                     selectedCategory))

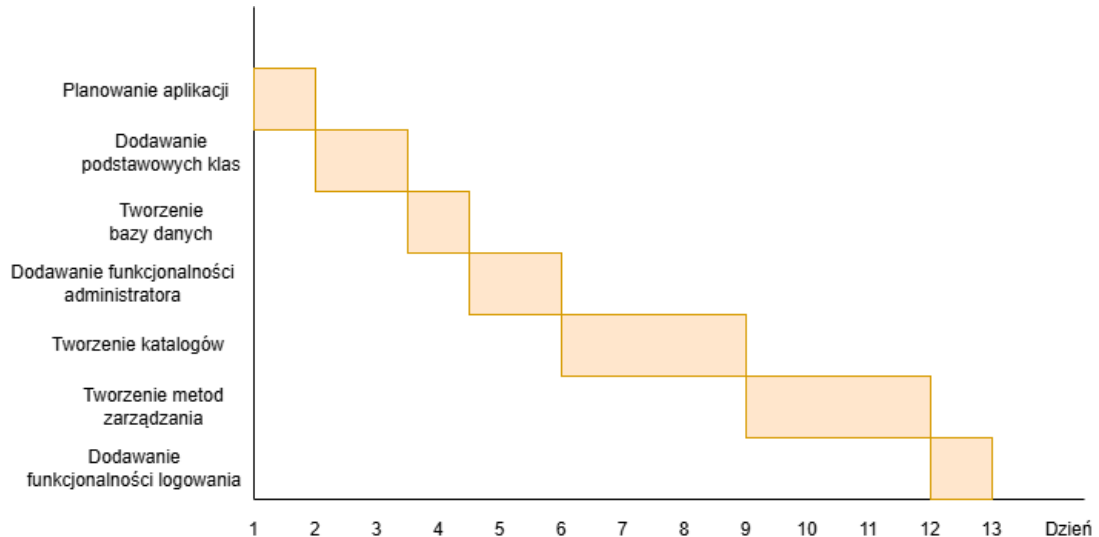
```

```
13         .collect(Collectors.toList());
14     }
15
16     //Sortowanie listy
17     private void sortProducts(List<ProductInfo> products) {
18         String sortBy = (String) sortByCombo.getSelectedItem();
19
20         List<ProductInfo> mutableList = new ArrayList<>(products);
21
22         mutableList.sort((p1, p2) -> {
23             switch (sortBy) {
24                 case "Cena (rosnąco)":
25                     return Double.compare(p1.getPrice(), p2.getPrice());
26                 case "Tytuł (od A do Z)":
27                     return p1.getTitle().compareToIgnoreCase(p2.getTitle());
28                 case "Autor (od A do Z)":
29                     return p1.getAuthor().compareToIgnoreCase(p2.getAuthor());
30                 default:
31                     return 0;
32             }
33         });
34         products.clear();
35         products.addAll(mutableList);
36     }
```

4. Harmonogram realizacji projektu

4.1. Diagram Ganta pracy

Na Rys. 4.1 przedstawiono diagram, zawierający informacje o kolejności podejmowanych działań oraz ich długości pracy. Diagram utworzony został przy użyciu <https://www.drawio.com/>.



Rys. 4.1. Diagram Ganta pracy.

Praca nad projektem została rozpoczęta 31 maja, a zakończyły się 12 czerwca, trwały one 13 dni, po 5 godzin dziennie, a etapy pracy zostały rozłożone tak, aby nie nakładały się na siebie i pozwalały na skupienie się na dodawaniu konkretnych funkcjonalności.

Najbardziej wymagającą i czasochłonną częścią prac okazuje się być zrealizowanie funkcjonalnych, przystępnych i estetycznych wizualnie katalogów produktów, zamówień i użytkowników.

4.2. Repozytorium i system kontroli wersji

Całość projektu wraz z wyeksportowaną bazą danych zostały przeniesione do repozytorium GitHub https://github.com/Buczul/Symulator_sklepu_internetowego_repo.

5. Prezentacja warstwy użytkowej projektu

5.1. Logowanie

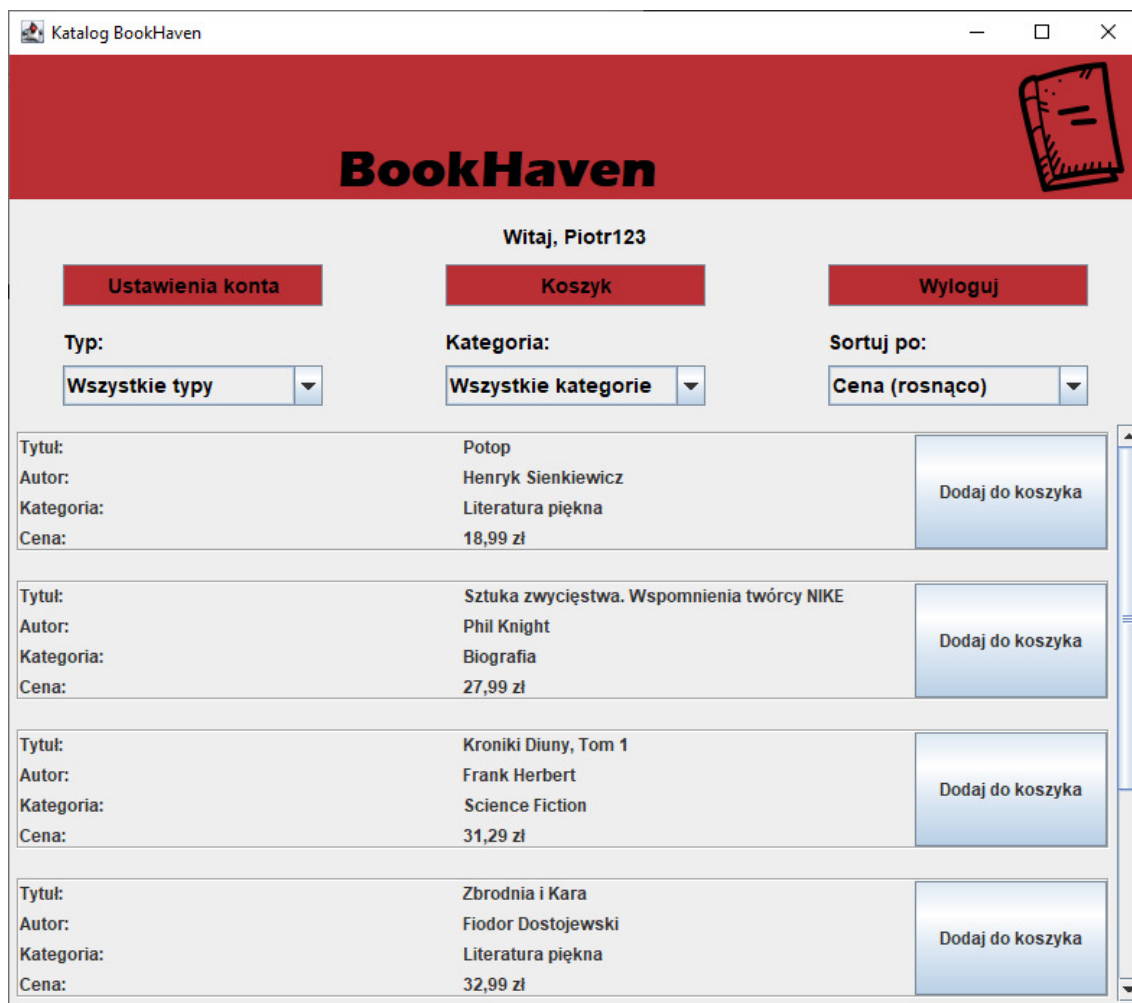
Po uruchomieniu aplikacji pojawia się okno ładowania, które szybko znika, a otwarty zostaje panel logowania, przedstawiony na Rys. 5.1. Dostępne jest tutaj logowanie na konto, rejestracja nowego użytkownika, a także logowanie jako gość, należy jednak pamiętać że gość ma jedynie dostęp do przeglądania oferty.

The image shows a window titled "Logowanie" (Login) with a red header bar. The header bar contains the "BookHaven" logo in white text and a small icon of a book. Below the header, the word "Logowanie" is centered in bold. Underneath, the text "Wprowadź dane logowania:" is followed by "Nazwa użytkownika lub email:". Below this is a white text input field. Then, the text "Hasło:" is followed by another white text input field. Below the password field is a red button labeled "Zaloguj". Below the "Zaloguj" button is the text "Wprowadz login/email i hasło." in red. Below this text are three more red buttons stacked vertically: "Zarejestruj", "Kontynuuj jako gość", and "Wyjdź". The window has standard OS window controls (minimize, maximize, close) in the top right corner.

Rys. 5.1. Panel logowania.

5.2. Panel klienta i gościa

Po zalogowaniu na konto użytkownika otworzy się okno z sortowalnym i filtrowalnym katalogiem produktów dostępnych w aplikacji. Zostało to przedstawione na Rys. 5.2. Dostępne są tutaj opcje zmiany ustawień konta (zmiana danych do późniejszej wysyłki), dodawanie produktów do koszyka, a także wejście do koszyka, wraz z możliwością złożenia zamówienia przez klienta.



Katalog BookHaven

BookHaven

Witaj, Piotr123

Ustawienia konta **Koszyk** **Wyloguj**

Typ: Wszystkie typy Kategoria: Wszystkie kategorie Sortuj po: Cena (rosnąco)

Tytuł:	Potop	Dodaj do koszyka
Autor:	Henryk Sienkiewicz	
Kategoria:	Literatura piękna	
Cena:	18,99 zł	
Tytuł:	Sztuka zwycięstwa. Wspomnienia twórcy NIKE	Dodaj do koszyka
Autor:	Phil Knight	
Kategoria:	Biografia	
Cena:	27,99 zł	
Tytuł:	Kroniki Diuny, Tom 1	Dodaj do koszyka
Autor:	Frank Herbert	
Kategoria:	Science Fiction	
Cena:	31,29 zł	
Tytuł:	Zbrodnia i Kara	Dodaj do koszyka
Autor:	Fiodor Dostojewski	
Kategoria:	Literatura piękna	
Cena:	32,99 zł	

Rys. 5.2. Panel klienta i gościa.

5.3. Panel administratora

Po zalogowaniu na konto administratora (konto testowe o nazwie użytkownika "admin" oraz hasło "admin"), wyświetlony zostaje panel z możliwymi do podjęcia, przez niego, działaniami. Panel ten przedstawiony jest na Rys. 5.3. Administrator ma możliwość usuwania użytkowników z bazy, dodawania nowych administratorów oraz nowych produktów do bazy. Zarządzanie produktami daje administratorowi narzędzia do usuwania i edycji istniejących produktów. Administrator może również zarządzać zamówieniami (przeglądać wszystkie zamówienia oraz zmieniać ich status).



Rys. 5.3. Panel administratora.

6. Podsumowanie

Projekt "Symulator sklepu internetowego" został zrealizowany zgodnie z założeniami i wymaganiami określonymi na początku prac. Aplikacja umożliwia użytkownikom pełne korzystanie z funkcjonalności sklepu internetowego, w tym rejestrację, logowanie, przeglądanie oferty, dodawanie produktów do koszyka oraz składanie zamówień. Administratorzy mają możliwość zarządzania użytkownikami, produktami i zamówieniami, co zapewnia kompleksową obsługę systemu.

Interfejs graficzny został zaprojektowany w sposób przejrzysty i intuicyjny, co ułatwia nawigację zarówno klientom, jak i administratorom. Wykorzystanie bazy danych MySQL pozwoliło na efektywne przechowywanie i zarządzanie danymi, a zastosowanie podejścia obiektowego Javy oraz korzystanie z biblioteki SWING zapewniło wysoką wydajność i skalowalność aplikacji.

Projekt spełnił wszystkie założone cele, a także uwzględnił wymagania нефункционалне, takie jak szybkość działania, niezawodność i estetyka interfejsu. Dalszy rozwój aplikacji może obejmować rozszerzenie funkcjonalności, np. dodanie nowych metod płatności czy tworzenie nowych kategorii produktów.

Podsumowując, projekt stanowi kompleksowe rozwiązanie symulujące działanie sklepu internetowego, które może służyć jako podstawa do dalszych prac rozwojowych.

Spis rysunków

3.1	Diagram dziedziczenia klas produktów.	9
3.2	obrazek książki wykorzystany w aplikacji.	12
3.3	Diagram ERD bazy danych.	13
3.4	Diagram kolejnych okien aplikacji.	14
4.1	Diagram Ganta pracy.	17
5.1	Panel logowania.	18
5.2	Panel klienta i gościa.	19
5.3	Panel administratora.	20

Spis tabel

3.1	Role użytkowników	9
-----	-----------------------------	---

Spis listingów

3.1	Metoda do rejestrowania nowych użytkowników	8
3.2	Enum kategorii książek	9
3.3	Metoda aktualizująca podstawowe dane produktu	10
3.4	Metoda aktualizująca specyficzne dane dla Ebooka	10
3.5	Wewnętrzna klasa ProductInfo	11
3.6	Klasa łączenia się z bazą danych	12
3.7	Klasa paneli z produktami	14
3.8	Metody filtrowania oraz sortowania	15

A. 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

..... Bartosz Buczułiński
Imię (imiona) i nazwisko studenta

Wydział Nauk Ścisłych i Technicznych

..... Informatyka
Nazwa kierunku

..... 134894
Numer albumu

1. Oświadczam, że moja praca projektowa pt.: Symulator sklepu internetowego
 - 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 ~~zgode~~ ~~nie wyrażam zgody~~** na udostępnienie mojej pracy projektowej do celów naukowo-badawczych z poszanowaniem przepisów ustawy o prawie autorskim i prawach pokrewnych.

Rzeszów, 15.06.2025 r.
(miejscowość, data)

Bartosz Buczułiński
(czytelny podpis studenta)

* Uwzględniając merytoryczny wkład prowadzącego przedmiot

** – niepotrzebne skreślić