**1. Opis aplikacji**

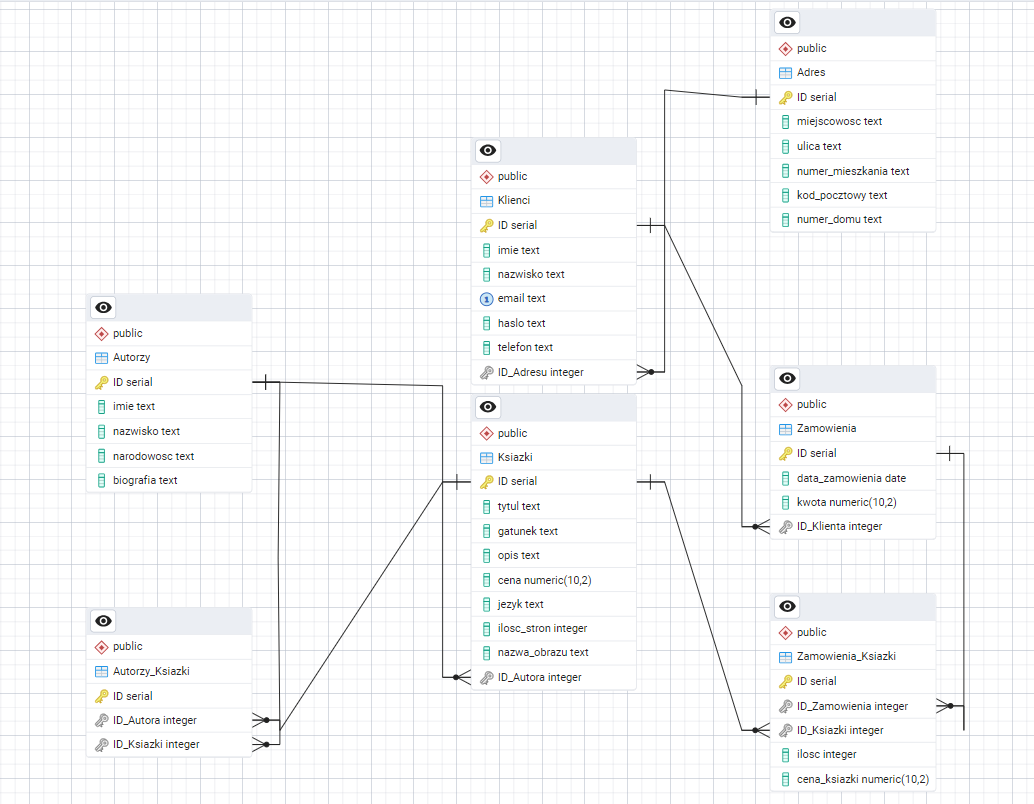
Aplikacja „Księgarnia” to system desktopowy stworzony przy użyciu technologii Windows Presentation Foundation (WPF) i języka C#. Jej głównym celem jest zarządzanie książkami oraz zamówieniami w systemie księgarni.

Do obsługi bazy danych wykorzystano Entity Framework Core, a dane są przechowywane w systemie bazodanowym PostgreSQL. Łańcuch połączenia jest przechowywany w pliku appsettings.json.

**Główne funkcje:**

* Przeglądanie i filtrowanie listy książek
* Zarządzanie pracownikami oraz adresami
* Tworzenie zamówień i przypisywanie do nich książek
* Obsługa kont użytkowników (admin/klient) na podstawie danych z appsettings.json
* Edytowanie książek
* Zapis książek do formatu json, które zostały dodane do koszyka w folderze ‘data’.
* Wczytanie książek do koszyka, przy uruchomieniu aplikacji.

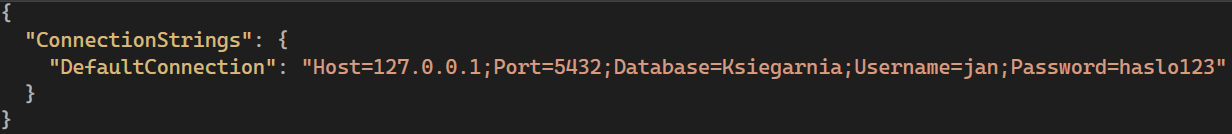
**2. Baza danych**

****

Rysunek 1

Struktura bazy danych została przedstawiona na powyższym diagramie ERD (rysunek 1). Przedstawia on tabele aplikacji oraz relacje między nimi, takie jak powiązania książek z autorami i zamówieniami.

**3. Konfiguracja aplikacji**

****

Rysunek 2

Plik konfiguracyjny appsettings.json zawiera łańcuch połączenia do bazy danych PostgreSQL. Znajduje się on w katalogu głównym projektu. Przykład konfiguracji (rysunek 2).

Dodatkowo aplikacja dynamicznie ustawia kontekst bazy danych w klasie ConnectionStringManager, odczytując dane konfiguracyjne i aktualizując kontekst EF.

Aplikacja obsługuje role użytkowników na podstawie nazwy użytkownika (Username) wyciągniętej z connection stringa (np. postgres, jan, admin\_user).

**4. Mapowanie tabel**

Każda klasa w przestrzeni nazw Ksiegarnia.Entities odpowiada jednej tabeli w bazie danych. Klasy te są dekorowane atrybutami ([Table], [Column]), które służą do mapowania nazw klas i właściwości na odpowiadające im tabele i kolumny w PostgreSQL.

Poniżej opis przykładowej klasy:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

Rysunek 3

**Opis:**

* Author odpowiada tabeli "Autorzy**"** w bazie danych.
* Właściwości klasy odwzorowują kolumny:
  + Name → imie
  + Surname → nazwisko
  + Nationality → narodowosc
  + Biography → biografia
* Klasa zawiera także właściwość BookAuthors do odwzorowania relacji wiele-do-wielu z książkami.
* Właściwość FullName jest tylko pomocnicza i nie jest mapowana do bazy danych – służy w logice aplikacji.

**Opis klasy DynamicImageSourceExtension**

Klasa DynamicImageSourceExtension jest niestandardowym rozszerzeniem znaczników (ang. *MarkupExtension*) służącym do dynamicznego ładowania obrazów w aplikacji WPF. Dzięki niej możliwe jest podanie tylko nazwy pliku obrazu w XAML, a rozszerzenie samo znajdzie plik i przekształci go w obiekt BitmapImage.

Kluczowe elementy logiki:

* Właściwość FileName: Określa nazwę pliku graficznego do załadowania.
* Konstruktor z parametrem fileName: Umożliwia przekazanie nazwy pliku bezpośrednio w XAML.
* Domyślny konstruktor: Potrzebny do prawidłowego działania w XAML, gdzie najpierw tworzona jest instancja, a dopiero potem przypisywane właściwości.
* Metoda ProvideValue(IServiceProvider): Kluczowa metoda, która:
  + - Buduje pełną ścieżkę do katalogu Images względem katalogu głównego aplikacji,
    - Sprawdza istnienie pliku,
    - W przypadku sukcesu tworzy i zwraca obiekt BitmapImage,
    - W przypadku braku pliku zwraca pusty obraz lub null.

**Opis klasy ShoppingCart (Koszyk zakupowy)**

Klasa ShoppingCart jest statyczną klasą odpowiedzialną za przechowywanie i zarządzanie koszykiem zakupowym w aplikacji księgarni. Przechowuje dane o aktualnie dodanych książkach, umożliwia obliczenie całkowitej kwoty zamówienia, zapis i odczyt koszyka z pliku JSON, a także przełączanie okna do widoku koszyka.

**Zarządzanie zawartością koszyka:**

* **books** – Kolekcja (ObservableCollection<Book>) zawierająca wszystkie aktualnie dodane książki.
* **itemsCounter** – Licznik książek w koszyku.
* **AddBookToCart(Button)** – Aktualizuje wizualnie licznik pozycji w koszyku przypisany do przycisku (np. w UI). W obecnej formie nie dodaje książki, a tylko aktualizuje widoczność

**5. Okna aplikacji**

Okno MainWindow (rysunek 4)pełni funkcję **głównego widoku aplikacji księgarni**, dostępnego po zalogowaniu użytkownika. Zawiera elementy takie jak pasek użytkownika, wyszukiwarkę książek, reklamę graficzną, listę losowych książek oraz wyniki wyszukiwania. Zapewnia podstawową nawigację po aplikacji i prezentację produktów (książek) użytkownikowi.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

**Logika backend:**

* Losowanie książek: przy starcie aplikacji losowane są 3 unikalne książki do RandomBooks.
* Debounce wyszukiwania: po każdej zmianie tekstu w SearchBox, odczekuje się 1 sekundę (timer debounceTimer), zanim uruchomione zostanie zapytanie do bazy.
* Wyszukiwanie książek: filtrowane są książki po tytule i imieniu+nazwisku autora, grupowane i wyświetlane w BooksList.
* Przejście do szczegółów książki: kliknięcie książki w BooksList lub RandomBooks otwiera okno BookDetailsWindow.
* Koszyk: licznik książek w koszyku jest aktualizowany, a kliknięcie przenosi do widoku koszyka.
* Zamykanie i przełączanie okien: większość interakcji powoduje zamknięcie MainWindow i otwarcie innego widoku (np. szczegóły książki, koszyk, panel użytkownika).

Okno BookDetailsWindow służy do wyświetlania szczegółowych informacji na temat wybranej książki, takich jak: tytuł, autor, cena, gatunek, liczba stron, opis, narodowość autora, biografia, liczba dostępnych egzemplarzy oraz okładka. Umożliwia użytkownikowi dodanie książki do koszyka, powrót do wyników wyszukiwania, edycję danych książki oraz przejście do widoku koszyka.

Obraz zawierający tekst, zrzut ekranu, Strona internetowa, oprogramowanie

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

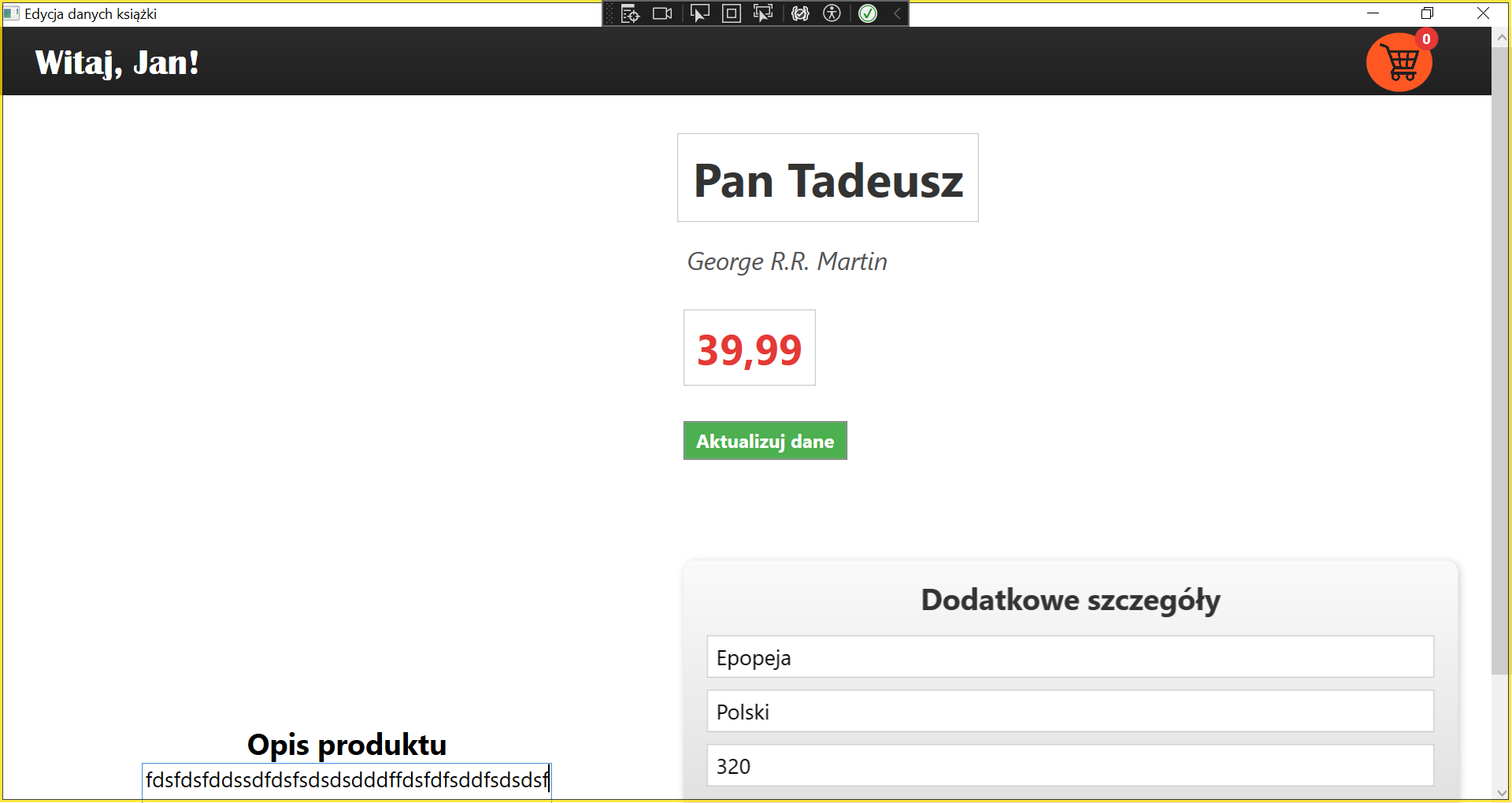
**Logika backend:**

* Wyświetlanie danych książki: po otwarciu okna, dane książki (tytuł, autor, opis, liczba stron, cena, tagi, biografia autora itd.) są wypełniane na podstawie przekazanego obiektu Book.
* Liczba egzemplarzy: na podstawie tytułu książki, z listy GroupedBookViewModel wyszukiwana jest liczba dostępnych egzemplarzy i wyświetlana użytkownikowi.
* Okładka książki: jeśli podana jest ścieżka do obrazu, jest on wczytywany i ustawiany jako źródło w kontrolce BookCover.
* Dodawanie do koszyka: kliknięcie przycisku „Dodaj do koszyka” zwiększa licznik książek w koszyku, dodaje książkę do globalnej listy ShoppingCart.books i aktualizuje przycisk z liczbą pozycji.
* Przejście do koszyka: kliknięcie w licznik koszyka otwiera widok koszyka (ShoppingCart.ChangeWindowToCart()) i zamyka bieżące okno.
* Panel użytkownika: kliknięcie w przycisk użytkownika powoduje otwarcie panelu użytkownika z poziomu MainWindow, z zamknięciem aktualnego widoku.

Powrót do wyszukiwania: przycisk „Powrót” zamyka bieżące okno i otwiera ponownie MainWindow.

Edycja książki: kliknięcie przycisku „Edytuj” otwiera okno BookEditWindow z przekazaną książką i zamyka bieżące okno.

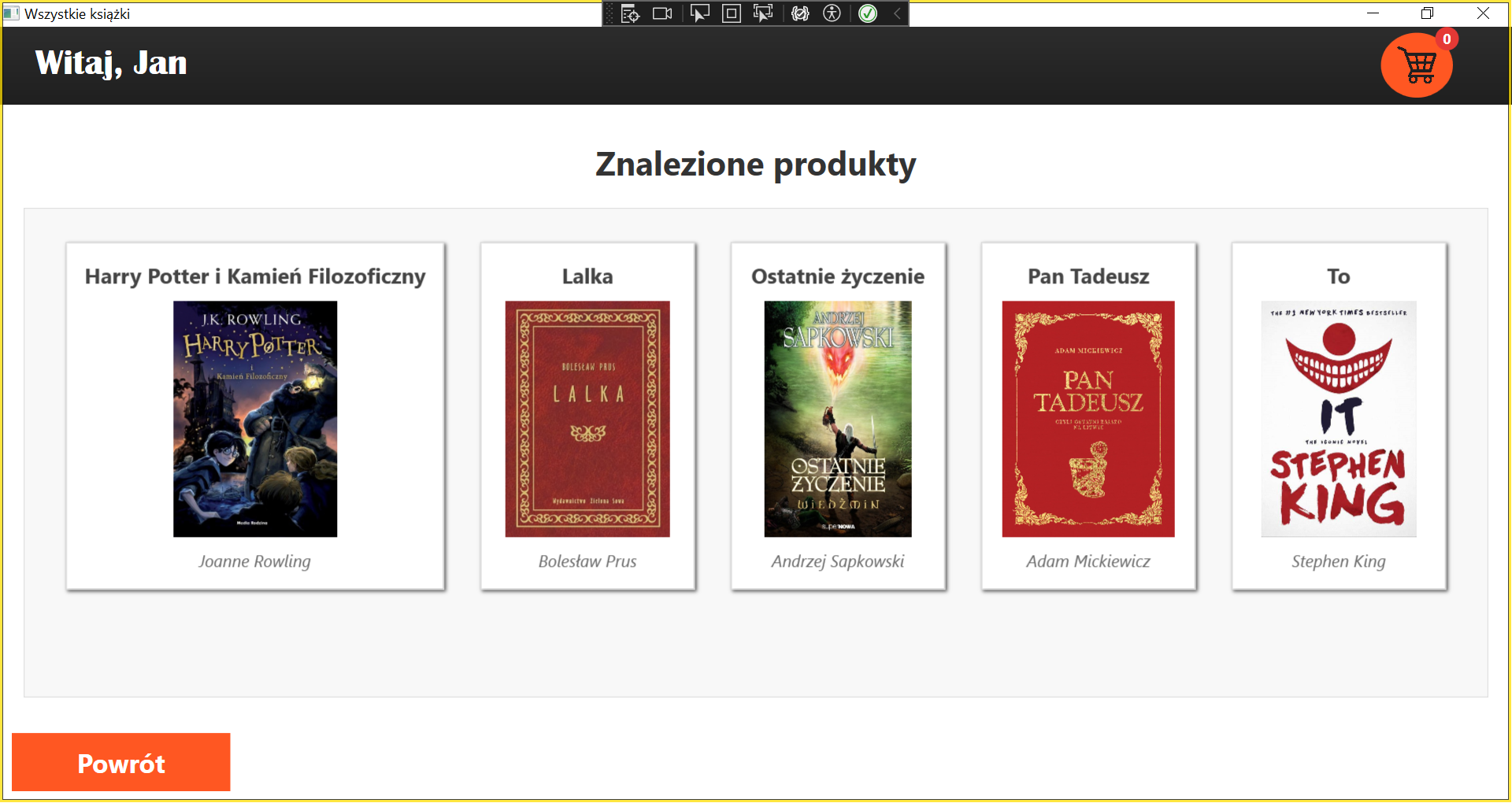
Okno BookEditWindow służy do edycji danych wybranej książki, takich jak tytuł, autor, cena, gatunek, język, liczba stron, opis oraz okładka. Pozwala na aktualizację informacji oraz nawigację do panelu użytkownika, widoku koszyka oraz powrót do wyników wyszukiwania.



**Logika backend:**

* Wczytywanie danych książki: po otwarciu okna pola edycyjne (tytuł, opis, cena, język, liczba stron, gatunek) są wypełniane wartościami z przekazanego obiektu Book.
* Aktualizacja danych: kliknięcie przycisku „Aktualizuj dane” zapisuje zmiany w obiekcie Book po sprawdzeniu poprawności danych (np. parsowanie ceny i liczby stron). W przypadku błędów wyświetlane jest odpowiednie powiadomienie.
* Nawigacja do panelu użytkownika: kliknięcie przycisku użytkownika otwiera panel użytkownika (UserPanel) i zamyka bieżące okno.
* Nawigacja do koszyka: kliknięcie licznika pozycji w koszyku otwiera widok koszyka i zamyka aktualne okno.
* Powrót do wyszukiwania: kliknięcie przycisku „Powrót” zamyka aktualne okno i otwiera ponownie MainWindow.

Okno AllBooksFound służy do wyświetlania wszystkich znalezionych książek na podstawie wyszukiwania lub filtrowania. Umożliwia użytkownikowi przeglądanie dostępnych pozycji w formie listy z okładkami i podstawowymi informacjami oraz nawigację do szczegółów wybranej książki, panelu użytkownika, widoku koszyka oraz powrót do wyszukiwania.



**Logika backend:**

* Inicjalizacja i wyświetlanie książek:  
  Po utworzeniu okna, lista FoundBooks jest wypełniana danymi książek przekazanymi z zewnętrznego kontrolera (np. z listy wyników wyszukiwania w ListView). Licznik pozycji w koszyku (itemsInCartCounter) jest aktualizowany na podstawie globalnej liczby książek w koszyku (ShoppingCart.itemsCounter). Nazwa użytkownika jest aktualizowana za pomocą funkcji ConnectionStringManager.ChangeUserNameNotification, która modyfikuje tekst na przycisku użytkownika.
* Nawigacja do koszyka:  
  Kliknięcie przycisku licznika pozycji w koszyku powoduje wywołanie metody ShoppingCart.ChangeWindowToCart(), która przełącza widok na koszyk (okno koszyka).
* Wyświetlanie szczegółów wybranej książki:  
  Zmiana zaznaczenia w kontrolce FoundBooks wywołuje metodę FoundBooks\_SelectionChanged. Ta metoda wywołuje w MainWindow funkcję SelectedBook(FoundBooks), która prawdopodobnie otwiera szczegóły wybranej książki lub inicjuje dalszą akcję związaną z wyborem.
* Powrót do widoku wyszukiwania:  
  Kliknięcie przycisku „Powrót” powoduje otwarcie nowego okna MainWindow oraz zamknięcie bieżącego okna, pozwalając użytkownikowi wrócić do ekranu głównego wyszukiwania.

**Opis okna ShoopingCartWindow (Koszyk)**

Okno to prezentuje zawartość koszyka zakupowego — wyświetla książki, które użytkownik dodał do koszyka, pozwala usuwać je, pokazuje podsumowanie cenowe oraz umożliwia przejście do podsumowania zamówienia lub powrót do wyszukiwania.

Obraz zawierający tekst, zrzut ekranu, Strona internetowa, Reklama internetowa

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

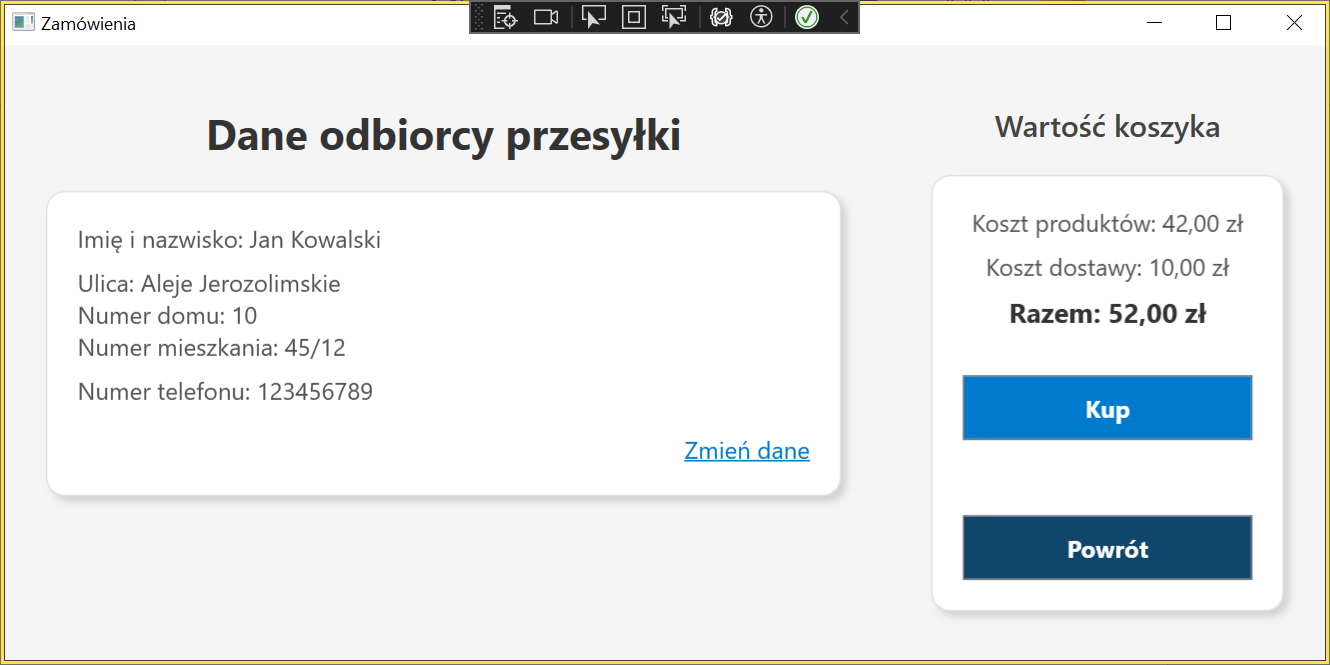
**Logika backend:**

* Inicjalizacja i wyświetlanie książek w koszyku:  
  Po utworzeniu okna ShoopingCartWindow kontrolka BooksInShoopingCart jest wypełniana listą książek znajdujących się aktualnie w koszyku (ShoppingCart.books). Następnie wywoływana jest metoda AllBooksPrice(), która oblicza i wyświetla całkowitą wartość zamówienia w elemencie TotalPrice.
* Usuwanie książki z koszyka:  
  Kliknięcie przycisku „Usuń” przy danej książce wywołuje metodę RemoveButton\_Click. Metoda ta usuwa wybraną książkę z globalnej listy ShoppingCart.books, dekrementuje licznik ShoppingCart.itemsCounter oraz aktualizuje wyświetlaną sumę zamówienia wywołując AllBooksPrice().
* Obliczanie i aktualizacja sumy zamówienia:  
  Metoda AllBooksPrice() wywołuje funkcję ShoppingCart.TotalPrice(), która zwraca sumę cen wszystkich książek w koszyku. Wynik jest formatowany i przypisywany do kontrolki TotalPrice.Text w formacie: "Kwota końcowa: XX zł".
* Przejście do podsumowania zamówienia:  
  Kliknięcie przycisku ShoppingSummary wywołuje metodę ShoppingSummary\_Click. Jeśli koszyk jest pusty (ShoppingCart.itemsCounter == 0), metoda nic nie robi. W przeciwnym wypadku otwiera nowe okno DeliveryAndPayment i zamyka aktualne okno koszyka.
* Powrót do wyszukiwania książek:  
  Kliknięcie przycisku BackToSearch wywołuje metodę BackToSearch\_Click, która tworzy i wyświetla nowe okno główne (MainWindow), a następnie zamyka okno koszyka.

**Opis okna DeliveryAndPayment (Zamówienia)**  
Okno to służy do wyświetlania danych odbiorcy przesyłki oraz podsumowania wartości koszyka wraz z kosztem dostawy. Umożliwia użytkownikowi zmianę danych adresowych, finalizację zakupu (utworzenie zamówienia i zapis do bazy) oraz powrót do wyszukiwania książek.

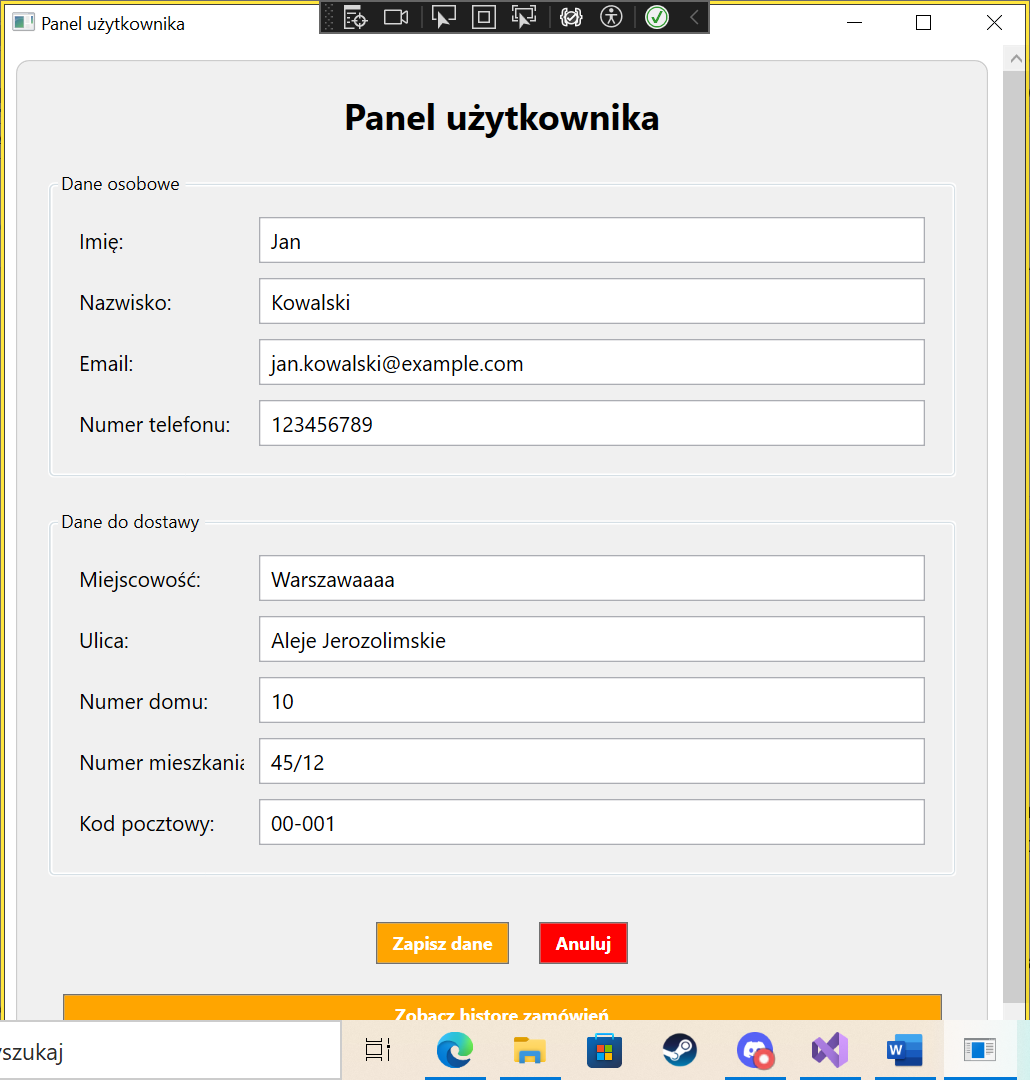
**Logika backend:**

* Inicjalizacja i wczytanie danych odbiorcy przesyłki:  
  W konstruktorze okna DeliveryAndPayment() pobierane są dane klienta o stałym customerId = 1 wraz z powiązanym adresem z bazy danych (BookstoreContex.context.Customers.Include(c => c.Address)).  
  Na podstawie stanu koszyka (ShoppingCart.TotalPrice()) wyliczana jest całkowita kwota zamówienia, która wyświetlana jest w kontrolkach txtPurchaseAmount (koszt produktów) oraz txtAllCost (razem z dostawą, 10 zł).  
  Jeżeli dane klienta lub adres są puste, pokazuje się komunikat txtNoData. W przeciwnym wypadku odpowiednie pola tekstowe są uzupełniane danymi klienta i adresu.



* Edycja danych odbiorcy:  
  Kliknięcie na tekst „Zmień dane” wywołuje metodę EditShippingData, która otwiera okno UserPanel (prawdopodobnie do edycji danych klienta) i zamyka okno aktualne.
* Realizacja zakupu (kliknięcie "Kup"):  
  Metoda buyButton\_Click najpierw sprawdza, czy koszyk zawiera jakieś produkty (ShoppingCart.itemsCounter > 0).  
  Następnie tworzy nowy rekord zamówienia (Orders) z datą, sumą zamówienia i identyfikatorem klienta.  
  Po zapisaniu zamówienia do bazy pobiera z bazy książki, które są w koszyku (po ID), a następnie tworzy powiązania między zamówieniem a książkami (OrdersBooks), z ilością i ceną.  
  Książki te są usuwane z bazy (symbolizując ich sprzedaż).  
  Po zatwierdzeniu zmian czyści koszyk oraz resetuje licznik pozycji, wyświetla komunikat o powodzeniu i przechodzi do okna głównego MainWindow.
* Powrót do wyszukiwania książek:  
  Kliknięcie przycisku backToSearch powoduje otwarcie okna głównego MainWindow i zamknięcie obecnego okna.

**Opis okna UserPanel (Panel użytkownika)**  
Okno to umożliwia użytkownikowi przeglądanie i edycję danych osobowych oraz adresowych zapisanych w systemie. Użytkownik może zapisać zmiany, anulować edycję, przejść do historii zamówień oraz — w razie potrzeby — przełączyć się na inne konto użytkownika (np. administratora) poprzez odświeżenie połączenia z bazą danych.

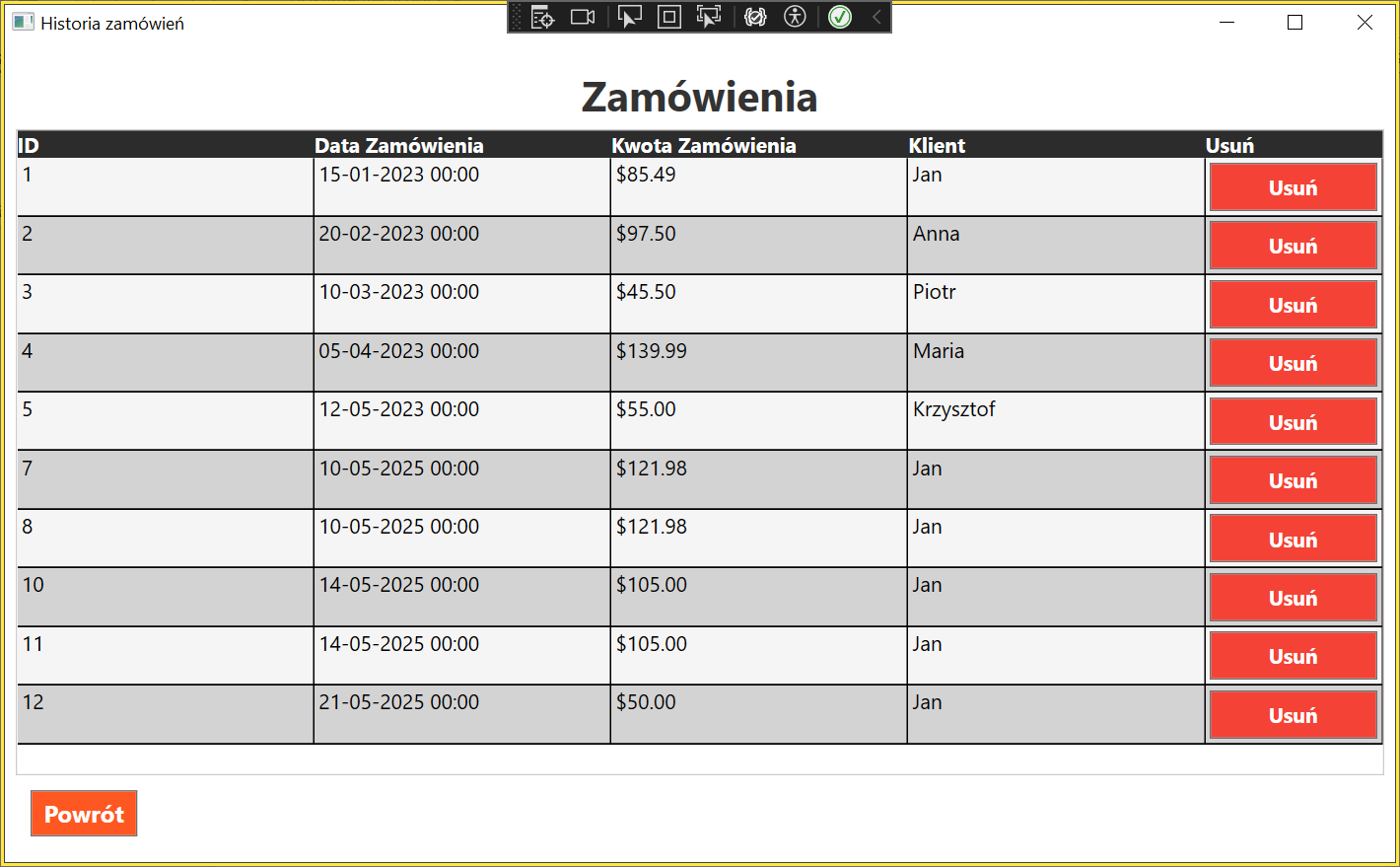


**Logika backend:**

* Wczytanie danych użytkownika:  
  Przy otwarciu okna metoda LoadUserData pobiera z bazy danych informacje o użytkowniku (Customer) oraz jego adresie (Address), a następnie wyświetla je w odpowiednich polach tekstowych.
* Zapisanie zmian danych:  
  Po kliknięciu przycisku Zapisz dane, metoda SaveData\_Click aktualizuje dane osobowe i adresowe klienta na podstawie zawartości formularza i zapisuje je do bazy danych. Następnie okno się zamyka i otwierane jest okno główne (MainWindow).
* Anulowanie edycji:  
  Kliknięcie przycisku Anuluj zamyka okno i otwiera MainWindow bez zapisywania zmian.
* Przegląd historii zamówień:  
  Przycisk Zobacz historię zamówień otwiera nowe okno OrdersWindow z historią zakupów użytkownika.
* Przełączenie użytkownika (odświeżenie połączenia):  
  Przycisk Przełącz użytkownika uruchamia metodę changeUserToAdmin\_Click, która próbuje przeładować kontekst bazy danych poprzez ConnectionStringManager.ReloadDatabaseContext(). W przypadku błędu przywracana jest domyślna konfiguracja bazy z pliku appsettings.json.

**Opis okna OrdersWindow (Historia zamówień)**

Okno to służy do przeglądania historii zamówień zapisanych w systemie. Wyświetla listę zamówień w formie tabeli, zawierającej informacje takie jak data zamówienia, kwota, dane klienta oraz przycisk umożliwiający usunięcie konkretnego zamówienia. Użytkownik może również wrócić do głównego okna aplikacji.



**Logika backend:**

* Wczytanie zamówień:  
  Przy otwarciu okna uruchamiana jest metoda LoadData, która pobiera z bazy danych wszystkie zamówienia (Orders) wraz z powiązanymi klientami (Customer) i książkami (OrderBooks.Book). Wynikowe dane są przypisywane do źródła danych kontrolki DataGrid.
* Usunięcie zamówienia:  
  Kliknięcie przycisku Usuń w tabeli powoduje wywołanie metody DeleteOrder\_Click. Metoda:
  + - Pobiera ID wybranego zamówienia z Tag przycisku,
    - Pyta użytkownika o potwierdzenie usunięcia,
    - Jeśli użytkownik potwierdzi, usuwa zamówienie (wraz z powiązaniami OrderBooks) z bazy danych,
    - Po usunięciu odświeża widok zamówień (LoadData).
* Powrót do wyszukiwania książek:  
  Przycisk Powrót zamyka bieżące okno (OrdersWindow) i otwiera główne okno aplikacji (MainWindow).

**6. Wczytywanie okładek książek**

**Okładki książek w aplikacji nie są przechowywane lokalnie w projekcie, lecz są hostowane zewnętrznie na stronie** [**imgbb.com**](https://imgbb.com)**, na koncie autora projektu:**

[**https://sebastian-mikos.imgbb.com/**](https://sebastian-mikos.imgbb.com/)

**Zamiast ładowania obrazów z folderu Images, aplikacja pobiera je dynamicznie na podstawie linków URL przypisanych do książek w bazie danych. Linki te wskazują bezpośrednio na obrazy hostowane w chmurze.**

**Dzięki temu:**

* **rozmiar aplikacji jest mniejszy,**
* **możliwa jest szybka podmiana obrazów bez rekompilacji projektu,**
* **aplikacja może być łatwo rozszerzona o więcej książek bez potrzeby dodawania nowych plików graficznych lokalnie.**

**Przykład linku do okładki:**

**https://i.ibb.co/5XTKXKPz/pan-tadeusz.jpg**

7. **Struktura projektu Ksiegarnia**

**Główne foldery:**

Entities/

* Zawiera klasy reprezentujące encje bazy danych, np. Book, Order, Customer. Klasy te są używane przez Entity Framework do mapowania danych.

Windows/

zawiera wszystkie okna aplikacji WPF, takie jak:

* Główne okno (MainWindow.xaml)
* Panel użytkownika (UserPanel.xaml)
* Koszyk zakupowy (ShoopingCartWindow.xaml)
* Zarządzanie zamówieniami itd.

Images/

* Zawiera obrazy wykorzystywane w aplikacji. Pliki graficzne są ładowane dynamicznie za pomocą klasy DynamicImageSourceExtension.

Migrations/

* Folder wygenerowany przez Entity Framework, zawierający migracje do zarządzania strukturą bazy.

database\_backup/

* Pomocniczy folder z plikami kopii zapasowej bazy danych lub zrzutami danych.

**Kluczowe pliki:**

* App.xaml – Konfiguracja globalnych zasobów aplikacji, np. style (Styles.xaml) i startowe okno.
* BookstoreContext.cs – Kontekst bazy danych (DbContext) używany przez Entity Framework do operacji CRUD.
* ConnectionStringManager.cs – Klasa pomocnicza do zarządzania łańcuchem połączenia z bazą danych (np. z appsettings.json).
* DynamicImageSourceExtension.cs – Własne rozszerzenie XAML umożliwiające dynamiczne wczytywanie obrazów z folderu Images na podstawie nazwy pliku.
* GroupedBookViewModel.cs – Model widoku wspierający grupowanie książek, np. według autora lub kategorii.
* ShoppingCart.cs – Statyczna klasa do zarządzania koszykiem zakupowym, jego zawartością i zapisem/odczytem do/z pliku cart.json.

**Inne pliki:**

* appsettings.json – Plik konfiguracyjny zawierający np. połączenie do bazy danych.
* Styles.xaml – Zbiór globalnych stylów i szablonów kontrolek dla aplikacji (np. kolory, fonty, przyciski).