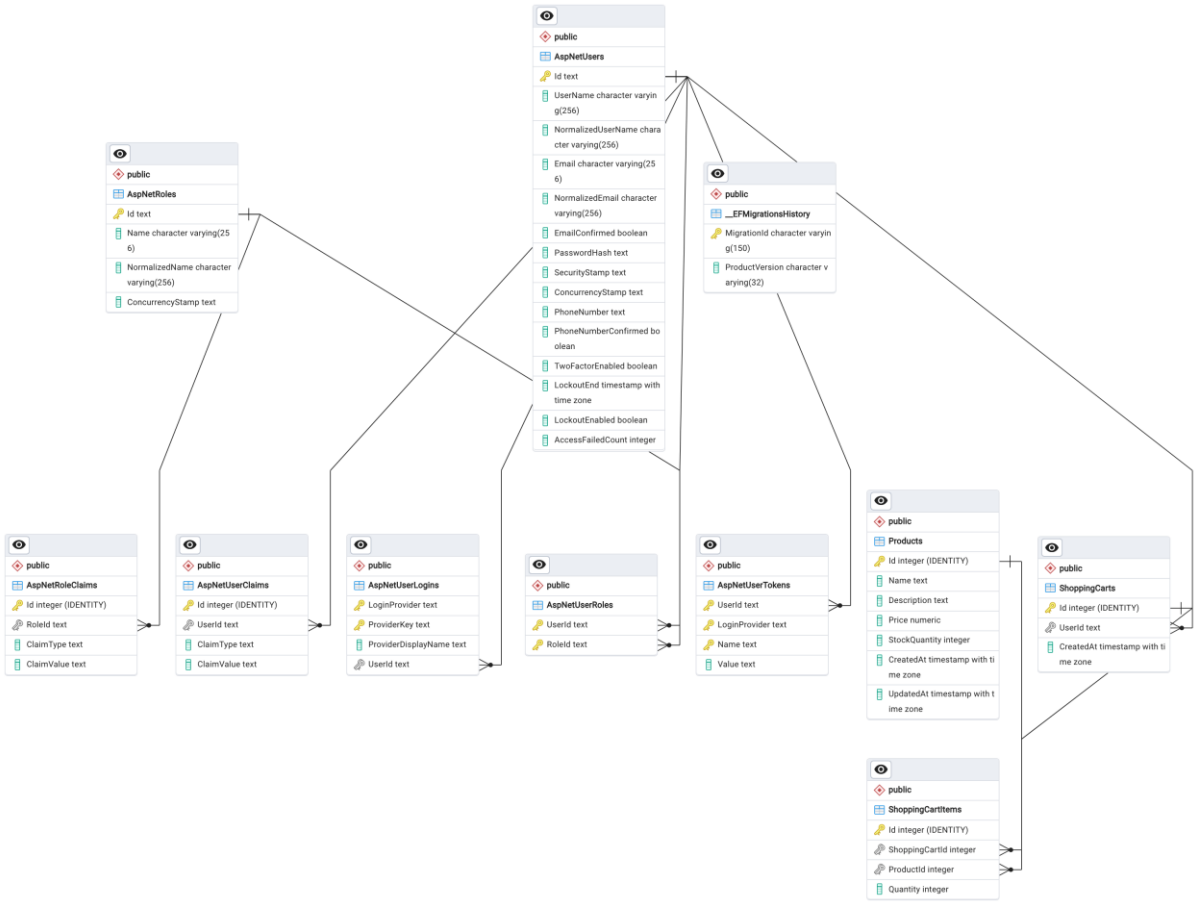
Sklep - Platforma e-commerce

1. Opis ogólny aplikacji

Aplikacja jest platformą e-commerce, która umożliwia użytkownikom logowanie się, zarządzanie koszykiem oraz przeglądanie i zarządzanie produktami. Dzięki zastosowaniu **ASP.NET Identity**, użytkownicy mają możliwość logowania się, rejestracji i zarządzania sesjami. Dodatkowo, aplikacja oferuje możliwość zarządzania danymi przez role administratorów (CRUD dla produktów) oraz operacje na koszykach, z wykorzystaniem technologii Entity Framework.

2. Opis bazy danych

1. Baza danych zawiera tabele zarządzające użytkownikami oraz procesem zakupowym. Kluczowe tabele to:
2. **AspNetUsers**: Przechowuje dane użytkowników, w tym dane kontaktowe, hasła i statusy weryfikacji.
3. **AspNetRoles**: Zawiera definicje ról użytkowników.
4. **AspNetRoleClaims**: Zawiera informacje o roszczeniach przypisanych do ról.
5. **AspNetUserClaims**: Przechowuje roszczenia przypisane bezpośrednio do użytkowników.
6. **AspNetUserLogins**: Dane logowania z zewnętrznymi dostawcami (np. Google, Facebook).
7. **AspNetUserRoles**: Powiązania użytkowników z rolami.
8. **AspNetUserTokens**: Tokeny używane do autentykacji dwuskładnikowej.
9. **Products**: Dane o produktach, w tym nazwa, opis, cena i ilość w magazynie.
10. **ShoppingCarts**: Zawiera informacje o koszykach użytkowników.
11. **ShoppingCartItems**: Przechowuje szczegóły dotyczące przedmiotów w koszykach.
12. **EFMigrationsHistory**: Historia migracji Entity Framework.



3. Kontroler: AuthController

Cel: Obsługuje logowanie, wylogowanie i sprawdzanie uprawnień użytkownika.

- **CheckAuth()**:
  - **Opis**: Sprawdza, czy użytkownik jest zalogowany.
  - **Metoda HTTP**: GET
  - **Ścieżka**: /api/auth/check-auth
  - **Statusy odpowiedzi**:
    - 200 OK: Użytkownik jest zalogowany.
    - 401 Unauthorized: Użytkownik nie jest zalogowany.
- **Logout()**:
  - **Opis**: Wylogowuje użytkownika.
  - **Metoda HTTP**: POST
  - **Ścieżka**: /api/auth/logout
  - **Statusy odpowiedzi**:
    - 200 OK: Użytkownik został wylogowany.
    - 401 Unauthorized: Brak danych do wylogowania.
- **CheckRole()**:
  - **Opis**: Sprawdza, czy użytkownik ma rolę "Admin".

    - **Metoda HTTP**: GET
    - **Ścieżka**: /api/auth/check-role
    - **Statusy odpowiedzi**:
        - 200 OK: Informacja o roli użytkownika.

4. Kontroler: CartController

Cel: Obsługuje operacje związane z koszykiem użytkownika (dodawanie, usuwanie, aktualizacja, obliczanie ceny).

- **GetCartItems()**:
    - **Opis**: Zwraca elementy znajdujące się w koszyku zalogowanego użytkownika.
    - **Metoda HTTP**: GET
    - **Ścieżka**: /api/cart
    - **Statusy odpowiedzi**:
        - 200 OK: Lista przedmiotów w koszyku.
        - 401 Unauthorized: Użytkownik nie jest zalogowany.
- **AddToCart()**:
    - **Opis**: Dodaje produkt do koszyka.
    - **Metoda HTTP**: POST
    - **Ścieżka**: /api/cart/add
    - **Statusy odpowiedzi**:
        - 200 OK: Produkt został dodany do koszyka.
        - 401 Unauthorized: Użytkownik nie jest zalogowany.
- **UpdateCartItem()**:
    - **Opis**: Aktualizuje ilość produktu w koszyku.
    - **Metoda HTTP**: PUT
    - **Ścieżka**: /api/cart/update/{cartItemId}
    - **Statusy odpowiedzi**:
        - 200 OK: Element w koszyku został zaktualizowany.
        - 401 Unauthorized: Użytkownik nie jest zalogowany.
- **RemoveFromCart()**:
    - **Opis**: Usuwa produkt z koszyka.
    - **Metoda HTTP**: DELETE
    - **Ścieżka**: /api/cart/delete/{cartItemId}
    - **Statusy odpowiedzi**:
        - 200 OK: Produkt został usunięty z koszyka.
        - 401 Unauthorized: Użytkownik nie jest zalogowany.
- **CalculateTotalCartPrice()**:

    - **Opis**: Oblicza łączną cenę wszystkich produktów w koszyku.
    - **Metoda HTTP**: GET
    - **Ścieżka**: /api/cart/totalPrice/{cartId}
    - **Statusy odpowiedzi**:
        - 200 OK: Łączna cena koszyka.

5. Kontroler: ProductController

Cel: Obsługuje operacje związane z produktami (przeglądanie, tworzenie, aktualizacja, usuwanie).

- **GetAllProducts()**:
    - **Opis**: Zwraca listę wszystkich dostępnych produktów.
    - **Metoda HTTP**: GET
    - **Ścieżka**: /api/product
    - **Statusy odpowiedzi**:
        - 200 OK: Lista produktów.
- **GetProductById()**:
    - **Opis**: Zwraca szczegóły produktu po jego ID.
    - **Metoda HTTP**: GET
    - **Ścieżka**: /api/product/{id}
    - **Statusy odpowiedzi**:
        - 200 OK: Produkt znaleziony.
        - 404 Not Found: Produkt o danym ID nie istnieje.
- **CreateProduct()**:
    - **Opis**: Tworzy nowy produkt (dostępne tylko dla administratorów).
    - **Metoda HTTP**: POST
    - **Ścieżka**: /api/product
    - **Statusy odpowiedzi**:
        - 201 Created: Produkt został stworzony.
- **UpdateProduct()**:
    - **Opis**: Aktualizuje dane produktu.
    - **Metoda HTTP**: PUT
    - **Ścieżka**: /api/product/{id}
    - **Statusy odpowiedzi**:
        - 200 OK: Produkt zaktualizowany.
        - 404 Not Found: Produkt o danym ID nie istnieje.
- **DeleteProduct()**:
    - **Opis**: Usuwa produkt.
    - **Metoda HTTP**: DELETE

- **Ścieżka**: /api/product/{id}
- **Statusy odpowiedzi**:
  - 204 No Content: Produkt został usunięty.
  - 404 Not Found: Produkt o danym ID nie istnieje.

- **GetProductsByPriceRange()**:
  - **Opis**: Zwraca produkty w określonym przedziale cenowym, wykorzystując subzapytanie SQL.
  - **Metoda HTTP**: GET
  - **Ścieżka**: /api/product/filter/{minPrice}/{maxPrice}
  - **Statusy odpowiedzi**:
    - 200 OK: Lista produktów w przedziale cenowym.

6.Operacje na bazie danych

1. *Zastosowanie podzapytania SQL*

W metodzie GetProductsByPriceRangeUsingSubquery użyto podzapytań SQL, aby dynamicznie określić minimalny i maksymalny zakres cen produktów. Metoda wykonuje zapytanie, które filtruje produkty według ceny, mieszczącej się w przedziale zdefiniowanym przez dwa podzapytania:

1. Pierwsze podzapytanie oblicza minimalną cenę produktów większą lub równą minPrice.
2. Drugie podzapytanie oblicza maksymalną cenę produktów mniejszą lub równą maxPrice.

```
SELECT *
FROM public."Products" p
WHERE p."Price" BETWEEN
    (SELECT MIN("Price") FROM public."Products" WHERE "Price" >= @p0)
    AND
    (SELECT MAX("Price") FROM public."Products" WHERE "Price" <= @p1);
```

2. *Funkcja update_timestamp*

Tworzy funkcję, która ustawia bieżący czas w kolumnie UpdatedAt za każdym razem, gdy rekord w tabeli Products jest aktualizowany.

```
CREATE OR REPLACE FUNCTION update_timestamp()
RETURNS TRIGGER AS $$
BEGIN
    NEW."UpdatedAt" = NOW();
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

3. *Wyzwalacz update_product_timestamp*

Tworzy wyzwalacz, który uruchamia funkcję update_timestamp przed każdą aktualizacją rekordu w tabeli Products, zapewniając automatyczną aktualizację znacznika czasu.

```
CREATE TRIGGER update_product_timestamp
BEFORE UPDATE ON "Products"
FOR EACH ROW
EXECUTE FUNCTION update_timestamp();
```

4. *Procedura assign_user_role*

Tworzy procedurę, która przypisuje rolę użytkownikowi w tabeli AspNetUserRoles na podstawie jego UserId i nazwy roli (np. "User").

```
CREATE OR REPLACE PROCEDURE assign_user_role(user_id text, role_name text)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO public."AspNetUserRoles" ("UserId", "RoleId")
    SELECT user_id, r."Id"
    FROM public."AspNetRoles" r
    WHERE r."Name" = role_name;
END;
$$;
```

5. *Funkcja trigger_assign_user_role*

Tworzy funkcję, która automatycznie wywołuje procedurę assign_user_role, aby przypisać rolę 'User' nowo utworzonemu użytkownikowi po jego dodaniu do tabeli AspNetUsers.

```
CREATE OR REPLACE FUNCTION trigger_assign_user_role()
RETURNS TRIGGER AS $$
BEGIN
    CALL assign_user_role(NEW."Id", 'User');
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

6. *Wyzwalacz set_user_role*

Tworzy wyzwalacz, który uruchamia funkcję trigger_assign_user_role po dodaniu nowego użytkownika do tabeli AspNetUsers, przypisując mu rolę 'User'.

```
CREATE TRIGGER set_user_role
AFTER INSERT ON public."AspNetUsers"
FOR EACH ROW
EXECUTE FUNCTION trigger_assign_user_role();
```

7. *Funkcja calculate_total_price*

Tworzy funkcję, która oblicza całkowitą cenę produktów w koszyku, sumując ceny produktów pomnożone przez ich ilość w tabeli ShoppingCartItems.

```
CREATE OR REPLACE FUNCTION calculate_total_price(cart_id UUID)
RETURNS DECIMAL AS $$
DECLARE
    total_price DECIMAL := 0;
BEGIN
    SELECT SUM(COALESCE(p."Price", 0) * COALESCE(ci."Quantity", 0)) INTO total_price
    FROM public."ShoppingCartItems" ci
    JOIN public."Products" p ON ci."ProductId" = p."Id"
    WHERE ci."ShoppingCartId" = cart_id;

    RETURN COALESCE(total_price, 0);
END;
$$ LANGUAGE plpgsql;
```

7.Technologie:

- **Frontend:** Angular 18
- **Backend:** .NET 8
- **Baza danych:** PostgreSQL, Entity Framework

- **Autentykacja i zarządzanie użytkownikami:** ASP.NET Identity