

DOKUMENT TECHNICZNY PROJEKTU

Programowanie zaawansowane - P. Płaczek - 2024/2025 ST

1. Tytuł projektu: „Clothers”

2. Tematyka projektu

Projekt polega na stworzeniu internetowego sklepu odzieżowego, który umożliwia użytkownikom przeglądanie produktów, dodawanie ich do koszyka oraz składanie zamówień. W aplikacji jest dostępny system logowania i rejestracji, panel administratora, panel firmy oraz generowanie faktur PDF po złożeniu zamówienia.

3. Autorzy projektu:

Mikołaj Winkiel
Wojciech Strugała

4. Wykorzystane technologie:

- SQL Server Management Studio 20.2.30.0
- MSQL Server 2022
- Projekt napisany w technologii .NET 8.0

5. Instrukcja pierwszego uruchomienia.

- pobranie i instalacja <https://www.microsoft.com/pl-pl/sql-server/sql-server-downloads>

- stworzenie nowej bazy danych w tym programie o nazwie **ClothersDb**

- dla pewności działania wpisać w Konsoli Menedżera Pakietów:

add-migration „nazwa dowolna”

- wpisać do tej samej konsoli kolejną komendę update-database

- po tych akcjach odpalić program

5.5. Opis struktury projektu:

Na podstawie przesłanej struktury projektu, oto opis struktury aplikacji:

Struktura projektu

1. Główne foldery:

- **Properties:**
Przechowuje pliki konfiguracyjne projektu
- **wwwroot:**
Folder przechowujący pliki statyczne aplikacji, takie jak style CSS, skrypty JavaScript, obrazy i inne zasoby dostępne publicznie.

- **Areas:**
Organizacja aplikacji na sekcje/obszary logiczne. Może być używana do zarządzania różnymi modułami projektu.
- **Controllers:**
Zawiera kontrolery obsługujące logikę aplikacji w ramach wzorca MVC (Model-View-Controller). W projekcie znajdują się:
 - **AdminController.cs**
 - **ClothesController.cs**
 - **CompanyController.cs**
 - **OrdersController.cs**
- **Data:**
Przechowuje klasy i konfiguracje związane z bazą danych:
 - **ApplicationDbContext.cs:** Klasa definiująca kontekst bazy danych, zawierająca zestaw DbSetów dla modeli aplikacji.
 - **RoleSeeder.cs:** Klasa odpowiedzialna za inicjalizację ról w systemie.
 - **UserSeeder.cs:** Klasa odpowiedzialna za inicjalizację użytkowników w bazie danych.
- **Enums:**
Zawiera wyliczenia używane w projekcie:
 - **ProductSizes.cs:** Enum definiujący dostępne rozmiary produktów (np. S, M, L, XL).
- **Migrations:**
Folder przechowujący migracje bazy danych wygenerowane przez Entity Framework. Migracje są używane do tworzenia i aktualizacji struktury bazy danych.
- **Models:**
Zawiera klasy reprezentujące dane w aplikacji:
 - **Cart.cs:** Reprezentacja koszyka użytkownika.
 - **CartItem.cs:** Reprezentacja pozycji w koszyku.
 - **Order.cs:** Reprezentacja zamówienia.
 - **OrderItem.cs:** Reprezentacja pozycji w zamówieniu.
 - **Product.cs:** Reprezentacja produktu.
 - **ErrorViewModel.cs:** Model używany do obsługi błędów.
- **Services:**
Zawiera klasy usług, które realizują specyficzną logikę biznesową:
 - **OrderPdfGenerator.cs:** Usługa generująca pliki PDF dla zamówień.

- **ViewModels:**

Zawiera modele widoków używane do przekazywania danych między kontrolerami a widokami:

- **CartItemViewModel.cs:** Model widoku dla pozycji w koszyku.
- **OrderViewModel.cs:** Model widoku dla zamówień.
- **ProductsViewModel.cs:** Model widoku dla listy produktów.

- **Views:**

Zawiera widoki w formacie Razor (.cshtml)

- **Admin:**
 - `_CreateUserPartial.cshtml`: Częściowy widok do tworzenia użytkownika.
 - `_EditUserPartial.cshtml`: Częściowy widok do edycji użytkownika.
 - `AdminPanel.cshtml`: Główny widok panelu administracyjnego.
- **Clothes:**
 - `Cart.cshtml`: Widok koszyka użytkownika.
 - `Create.cshtml`, `Edit.cshtml`, `Delete.cshtml`, `Index.cshtml`: Widoki do zarządzania produktami.
- **Company:**
 - `CompanyPanel.cshtml`: Widok panelu zarządzania firmą.
- **Orders:**
 - `Confirmation.cshtml`: Widok potwierdzenia zamówienia.
 - `MyOrders.cshtml`: Widok listy zamówień użytkownika.
 - `Create.cshtml`: Widok tworzenia zamówienia.

Projekt opiera się na wzorcu MVC:

1. **Modele:** Reprezentują dane w aplikacji (np. produkty, koszyk, zamówienia).
2. **Kontrolery:** Obsługują logikę aplikacji i komunikację między modelami a widokami.
3. **Widoki:** Prezentują dane użytkownikowi.

6. Wylistowane wszystkie modele

Model: Product

Opis pól:

- **Id**

→ Identyfikator produktu

→ Klucz główny, unikalny identyfikator produktu w bazie danych.

- Name

→ Nazwa produktu

→ Pole nie może być puste [Required] oraz nie może mieć więcej niż 100 znaków [StringLength(100)].

→ Błąd walidacji: "Nazwa jest wymagana.", "Nazwa nie może przekraczać 100 znaków."

- Description

→ Opis produktu

→ Pole nie może być puste [Required] oraz nie może mieć więcej niż 1000 znaków [StringLength(1000)].

→ Błąd walidacji: "Opis jest wymagany.", "Opis nie może przekraczać 1000 znaków."

- Price

→ Cena produktu

→ Pole nie może być puste [Required].

→ Cena musi być większa od 0 [Range(0.01, double.MaxValue)].

→ Precyzja pola: 18 miejsc całkowitych, 2 miejsca dziesiętne [Precision(18, 2)].

→ Błąd walidacji: "Cena jest wymagana.", "Cena musi być większa od 0."

- Sizes

→ Rozmiar produktu

→ Pole nie może być puste [Required].

→ Przechowywane jako string w bazie danych z maksymalną długością 20 znaków [Column(TypeName = "nvarchar(20)")].

→ Pole jest typu wyliczeniowego 'ProductSizes', definiowanego w przestrzeni nazw 'Clothers.Enums'.

→ Błąd walidacji: "Rozmiar jest wymagany."

- Quantity

- Ilość w magazynie
- Pole nie może być puste [Required] oraz nie może być ujemne [Range(0, int.MaxValue)].
- Błąd walidacji: "Ilość jest wymagana.", "Ilość nie może być ujemna."

- Image

- Zdjęcie produktu
- Pole opcjonalne, przechowuje dane binarne obrazu w formacie byte[].

- IsApproved

- Status zatwierdzenia produktu
- Wskazuje, czy produkt został zatwierdzony przez administratora. Pole typu bool.

- UserId

- Identyfikator użytkownika
- Pole nie może być puste [Required].
- Określa powiązanie produktu z użytkownikiem, do którego należy.

- User

- Relacja z użytkownikiem
- Klucz obcy odnoszący się do tabeli 'AspNetUsers' przy użyciu 'UserId'.
- Powiązanie z modelem 'IdentityUser' wbudowanym w ASP.NET Identity.

Po co jest?:

Model ten służy do zapisywania produktów w bazie danych. Co w późniejszym etapie daje takie możliwości jak, dodawanie nowych produktów, edycja produktów, usuwanie produktów jak i wczytywanie tych produktów z bazy danych żeby je np wyświetlić.

Model: CartItem

Opis pól:

- **Id**
 - Identyfikator elementu koszyka
 - Klucz główny, unikalny identyfikator elementu w koszyku.
- **CartId**
 - Identyfikator koszyka
 - Pole wymagane [Required], przechowuje identyfikator koszyka, do którego należy dany element.
 - Jest kluczem obcym wskazującym na tabelę Cart [ForeignKey(nameof(CartId))].
- **Cart**
 - Relacja z koszykiem
 - Powiązanie z modelem Cart, reprezentującym koszyk, do którego należy dany element.
- **ProductId**
 - Identyfikator produktu
 - Pole wymagane [Required], przechowuje identyfikator produktu, który został dodany do koszyka.
 - Jest kluczem obcym wskazującym na tabelę Product [ForeignKey(nameof(ProductId))].
- **Product**
 - Relacja z produktem
 - Powiązanie z modelem Product, reprezentującym produkt dodany do koszyka.
- **Quantity**
 - Ilość produktu
 - Określa liczbę sztuk danego produktu w koszyku.
 - Pole musi mieć wartość przynajmniej 1 [Range(1, int.MaxValue)].
 - Błąd walidacji: "Ilość musi być przynajmniej 1."
 - Wyświetlana nazwa: "Ilość" [Display(Name = "Ilość")].

Po co to jest?:

Model CartItem służy do reprezentowania pojedynczego elementu w koszyku. Umożliwia przechowywanie informacji o produkcie dodanym do koszyka oraz jego ilości.

Model: Cart

Opis pól:

- **Id**
 - Identyfikator koszyka
 - Klucz główny, unikalny identyfikator koszyka w bazie danych.

- **UserId**
 - Identyfikator użytkownika
 - Pole wymagane [Required], przechowuje identyfikator użytkownika, do którego należy koszyk.
 - Jest kluczem obcym wskazującym na tabelę AspNetUsers [ForeignKey(nameof(UserId))].
 - **User**
 - Relacja z użytkownikiem
 - Powiązanie z modelem IdentityUser, reprezentującym właściciela koszyka.
 - **Items**
 - Lista elementów koszyka
 - Kolekcja elementów powiązanych z koszykiem.
 - Reprezentowana przez kolekcję ICollection<CartItem> (domyślnie inicjalizowana jako pusta lista).
-

Zastosowanie modelu Cart

Model Cart służy do reprezentowania koszyka użytkownika w aplikacji. Jest to kontener przechowujący informacje o wszystkich produktach, które użytkownik dodał do koszyka.

Model: OrderItem

Opis pól:

- **Id**
 - Identyfikator elementu zamówienia
 - Klucz główny, unikalny identyfikator elementu zamówienia.
- **OrderId**
 - Identyfikator zamówienia

→ Pole wymagane [Required], przechowuje identyfikator zamówienia, do którego należy dany element.

→ Jest kluczem obcym wskazującym na tabelę Order
[ForeignKey(nameof(OrderId))].

- **Order**

→ Relacja z zamówieniem

→ Powiązanie z modelem Order, reprezentującym zamówienie, do którego należy element.

- **ProductId**

→ Identyfikator produktu

→ Pole wymagane [Required], przechowuje identyfikator produktu powiązanego z zamówieniem.

→ Jest kluczem obcym wskazującym na tabelę Product
[ForeignKey(nameof(ProductId))].

- **Product**

→ Relacja z produktem

→ Powiązanie z modelem Product, reprezentującym produkt będący częścią zamówienia.

- **Quantity**

→ Ilość produktu

→ Pole wymagane [Required], określa liczbę sztuk danego produktu w zamówieniu.

→ Wartość musi wynosić przynajmniej 1 [Range(1, int.MaxValue)].

→ Błąd walidacji: "Ilość musi być przynajmniej 1."

- **UnitPrice**

→ Cena jednostkowa produktu

→ Pole wymagane [Required], przechowuje cenę jednostkową produktu w ramach zamówienia.

→ Wartość musi być większa od 0 [Range(0.01, double.MaxValue)].

→ Precyzja pola ustawiona na 18 miejsc całkowitych i 2 miejsca dziesiętne
[Precision(18, 2)].

→ Błąd walidacji: "Cena jednostkowa musi być większa od 0."

- **TotalPrice**

→ Całkowita cena elementu zamówienia

→ Pole obliczeniowe (bez zapisu do bazy danych), które automatycznie wylicza wartość jako UnitPrice * Quantity.

→ Precyzja pola ustawiona na 18 miejsc całkowitych i 2 miejsca dziesiętne
[Precision(18, 2)].

Zastosowanie modelu OrderItem

Model OrderItem reprezentuje **pojedynczy element w zamówieniu**, umożliwiając przechowywanie informacji o produktach zamówionych przez użytkownika, ich ilości, cenie jednostkowej oraz całkowitym koszcie.

Model: Order

Opis pól:

- **Id**
 - Identyfikator zamówienia
 - Klucz główny, unikalny identyfikator zamówienia.
- **FirstName**
 - Imię
 - Pole wymagane [Required], przechowuje imię osoby składającej zamówienie.
 - Walidacje:
 - Nie może zawierać cyfr [RegularExpression(@"^[^0-9]*\$")].
 - Maksymalna długość: 50 znaków [StringLength(50)].

- Błąd walidacji: "Imię nie może zawierać cyfr." lub "Imię nie może przekraczać 50 znaków."
- **LastName**
 - Nazwisko
 - Pole wymagane [Required], przechowuje nazwisko osoby składającej zamówienie.
 - Walidacje:
 - Nie może zawierać cyfr [RegularExpression(@"^[^0-9]*\$")].
 - Maksymalna długość: 50 znaków [StringLength(50)].
 - Błąd walidacji: "Nazwisko nie może zawierać cyfr." lub "Nazwisko nie może przekraczać 50 znaków."
- **DeliveryAddress**
 - Adres dostawy
 - Pole wymagane [Required], przechowuje adres, na który ma zostać dostarczone zamówienie.
 - Maksymalna długość: 200 znaków [StringLength(200)].
 - Błąd walidacji: "Adres dostawy nie może przekraczać 200 znaków."
- **PaymentMethod**
 - Sposób płatności
 - Pole wymagane [Required], przechowuje informacje o wybranej metodzie płatności.
 - Maksymalna długość: 50 znaków [StringLength(50)].
 - Błąd walidacji: "Sposób płatności nie może przekraczać 50 znaków."
- **DeliveryMethod**
 - Sposób dostawy
 - Pole wymagane [Required], przechowuje informacje o wybranej metodzie dostawy.
 - Maksymalna długość: 50 znaków [StringLength(50)].
 - Błąd walidacji: "Sposób dostawy nie może przekraczać 50 znaków."
- **OrderDate**
 - Data złożenia zamówienia
 - Automatycznie ustawiana na bieżącą datę i godzinę w momencie tworzenia zamówienia.
 - Typ: DateTime.
- **UserId**
 - Identyfikator użytkownika
 - Pole wymagane [Required], przechowuje identyfikator użytkownika składającego zamówienie.
 - Może być użyte do identyfikacji relacji z kontem użytkownika.
- **OrderItems**
 - Lista elementów zamówienia
 - Kolekcja elementów zamówienia, przechowująca szczegóły dotyczące zamówionych produktów.

- Musi zawierać co najmniej jeden element [MinLength(1)].
- Błąd walidacji: "Zamówienie musi zawierać przynajmniej jeden przedmiot."
- Typ: ICollection<OrderItem> (domyślnie inicjalizowana jako pusta lista).

Zastosowanie modelu Order

Model Order reprezentuje zamówienie złożone przez użytkownika w aplikacji. Służy do przechowywania szczegółowych informacji o zamówieniu, w tym danych klienta, metod dostawy i płatności, a także listy zamówionych produktów.

Nasz projekt Clothiers zawiera 4 kontrolery; AdminController, ClothesController, CompanyController, OrdersController.

Kontroller ClothesController

Index

GET

Wyświetla listę zatwierdzonych produktów dostępnych w magazynie, z opcją wyszukiwania.

- Parametry:
 - searchString: Opcjonalny ciąg znaków do wyszukiwania produktów po nazwie.


- Funkcjonalność:

1. Sprawdza, czy ciąg wyszukiwania nie przekracza 50 znaków. Jeśli tak, obcina go i wyświetla komunikat.

2. Pobiera tylko zatwierdzone produkty (IsApproved = true), które mają dodatnią ilość (Quantity > 0).
3. Jeśli podano ciąg wyszukiwania, stosuje filtrację (EF.Functions.Like) na nazwach produktów.
4. Zwraca widok z listą produktów.

Produkty


Szukaj produktów... Szukaj



Majtkisfdgds
Majtki męski, slipkisrfdgdfgr
Cena: \$29.00
Ilość: 622
Rozmiar: M
[Zaloguj się, aby dodać do koszyka](#)



koszulka
wq3fewefwef
Cena: \$9.00
Ilość: 68
Rozmiar: XS
[Zaloguj się, aby dodać do koszyka](#)



123123123
123123213
Cena: \$123,213.00
Ilość: 123123123
Rozmiar: M
[Zaloguj się, aby dodać do koszyka](#)

5.

Create (GET)

GET

Wyświetla formularz do tworzenia nowego produktu.

Nazwa

Opis

Cena

Rozmiar

-- Wybierz rozmiar --

Ilość

Prześlij zdjęcie

Wybierz plik

Nie wybrano pliku

Stwórz

Anuluj

Create (POST)

POST

Dodaje nowy produkt do bazy danych.

- Parametry:
 - ProductsViewModel productVm: Model widoku zawierający dane produktu.
 - IFormFile Image: Opcjonalny obraz produktu.
 - Funkcjonalność:
 1. Waliduje dane wejściowe.
 2. Tworzy nowy obiekt Product z danymi z formularza.
 3. Jeśli podano obraz:
 - Zapisuje go w formacie binarnym do właściwości Image.
 4. Oznacza produkt jako niezatwierdzony (IsApproved = false).
 5. Zapisuje produkt w bazie danych.
-

Edit (GET)

GET

Wyświetla formularz do edycji istniejącego produktu.

- Parametry:
 - id: Identyfikator produktu.
- Funkcjonalność:
 1. Pobiera produkt z bazy danych na podstawie id.
 2. Jeśli produkt nie istnieje, zwraca błąd 404.
 3. Tworzy model widoku ProductsViewModel i wypełnia go danymi produktu.

4. Zwraca widok edycji.

Edit Product

Nazwa

Majtkisfdgds

Opis

Majtki męski, slipkisrfdgdfgr

Cena

29,00

Rozmiar

M

Ilość

622

Product Image



Wybierz plik

Nie wybrano pliku

Save Changes

Cancel

Edit (POST)

POST

Aktualizuje dane istniejącego produktu w bazie danych.

- Parametry:
 - int id: Identyfikator produktu.
 - ProductsViewModel productVm: Model widoku z zaktualizowanymi danymi.
 - IFormFile Image: Opcjonalny obraz produktu.
- Funkcjonalność:

1. Sprawdza, czy identyfikatory produktu są zgodne.
2. Pobiera produkt z bazy danych.
3. Aktualizuje jego dane na podstawie modelu widoku.
4. Jeśli podano nowy obraz, zapisuje go w formacie binarnym.
5. Zapisuje zmiany w bazie danych.

Delete (GET)

GET

Wyświetla szczegóły produktu przed usunięciem.

- Parametry:

- id: Identyfikator produktu.
- Funkcjonalność:
 1. Pobiera produkt z bazy danych.
 2. Jeśli produkt nie istnieje, zwraca błąd 404.
 3. Zwraca widok z danymi produktu.

Usuń ilość produktu

Nazwa	Majtkisfdgds
Opis	Majtki męski, slipkisrfdgdfgr
Cena	\$29.00
Rozmiar	M
Ilość dostępna	622

Czy na pewno chcesz zmniejszyć ilość tego produktu? Podaj ilość, którą chcesz usunąć.

Ilość do usunięcia

Usuń ilość

Anuluj

Delete (POST)

POST

Zmniejsza ilość produktu w magazynie lub usuwa go całkowicie.


- Parametry:
 - int id: Identyfikator produktu.
 - int quantityToRemove: Ilość do usunięcia.
- Funkcjonalność:
 1. Pobiera produkt z bazy danych.
 2. Jeśli produkt nie istnieje, zwraca błąd.
 3. Jeśli quantityToRemove jest większe od ilości dostępnej, wyświetla komunikat o błędzie.
 4. Zmniejsza ilość produktu lub usuwa go z bazy danych.
 5. Zapisuje zmiany.

Cart

GET

Wyświetla zawartość koszyka użytkownika.

- Funkcjonalność:
 1. Pobiera identyfikator użytkownika.
 2. Ładuje koszyk użytkownika wraz z jego produktami.
 3. Tworzy model widoku CartViewModel zawierający szczegóły produktów w koszyku.
 4. Zwraca widok koszyka.

Koszyk				
Produkt	Cena	Ilość	Razem	Akcje
 Majtkisfdgds	\$29.00	5	\$145.00	Usuń
			Razem: \$145.00	
Zrealizuj Zamówienie				

AddToCart

POST

Dodaje produkt do koszyka użytkownika.

- Parametry:
 - int id: Identyfikator produktu.
 - int quantity: Ilość do dodania.
- Funkcjonalność:
 1. Pobiera produkt na podstawie id.
 2. Sprawdza, czy ilość jest prawidłowa i dostępna w magazynie.
 3. Pobiera lub tworzy koszyk użytkownika.
 4. Dodaje produkt do koszyka lub zwiększa jego ilość.
 5. Zapisuje zmiany w bazie danych.

RemoveFromCart

POST

Usuwa produkt z koszyka użytkownika.

- Parametry:

- int id: Identyfikator produktu.
- Funkcjonalność:
 1. Pobiera koszyk użytkownika.
 2. Usuwa wskazany produkt z koszyka.
 3. Zapisuje zmiany w bazie danych.

Checkout

POST

Finalizuje zamówienie, aktualizuje magazyn i czyści koszyk użytkownika.

- Funkcjonalność:
 1. Pobiera koszyk użytkownika wraz z produktami.
 2. Aktualizuje ilości produktów w magazynie.
 3. Usuwa koszyk użytkownika.
 4. Zapisuje zmiany w bazie danych.

Kontroler CompanyController

Kontroler CompanyController jest odpowiedzialny za zarządzanie funkcjami związanymi z kontem firmowym użytkownika (np. przeglądanie ofert i zamówień). Korzysta z ról Company oraz Admin, które są przypisane do autoryzowanych użytkowników. Pracuje z bazą danych przy pomocy ApplicationDbContext i zarządza użytkownikami za pomocą UserManager.

1. CompanyPanel

GET

Metoda ta wyświetla panel firmowy użytkownika, w którym widoczne są wszystkie produkty danej firmy (tj. oferty, które zostały przez niego dodane).

- **Parametry:**

- Brak otrzymywanych parametrów

- **Funkcjonalność:**

1. **Pobranie identyfikatora użytkownika:**

- Za pomocą UserManager.GetUserId(User) pobierany jest userId bieżącego użytkownika.
- Jeśli użytkownik nie jest zalogowany lub identyfikator użytkownika nie jest dostępny, zwróci odpowiedź Unauthorized().

2. **Pobranie produktów:**

- Wykonuje zapytanie do bazy danych przy użyciu Entity Framework (_context.Products) w celu uzyskania produktów przypisanych do danego użytkownika (UserId).
- Wyniki są ładowane asynchronicznie za pomocą ToListAsync().

3. **Zwrócenie widoku:**

- Jeżeli użytkownik ma jakieś produkty, zwraca je w widoku CompanyPanel.
- Jeśli nie, będzie wyświetlany pusty widok.

Panel Firmy

[Dodaj Nową Ofertę](#)[Moje Zamówienia](#)

Majtkisfdgds

Majtki męski, slipkisrfdgdfgr

Cena: \$29.00

Ilość: 622

Rozmiar: M

[Edytuj](#)[Usuń](#)

Majtki

Majtki męski, slipki

Cena: \$9.00

Ilość: 0

Rozmiar: XS

[Edytuj](#)[Usuń](#)

123123123

123123213

Cena: \$123,213.00

Ilość: 123123123

Rozmiar: M

[Edytuj](#)[Usuń](#)

2. MyOrders

GET

Metoda ta umożliwia wyświetlenie zamówień związanych z produktami danej firmy. Użytkownik może zobaczyć wszystkie zamówienia, w których jego produkty zostały zakupione.

- **Parametry:**

- Brak otrzymywanych parametrów

- **Funkcjonalność:**

1. **Pobranie identyfikatora użytkownika:**

- Używa UserManager.GetUserId(User), aby pobrać identyfikator użytkownika.
- Jeżeli identyfikator jest pusty, metoda zwróci Unauthorized().

2. **Pobranie zamówień:**

- Wykonuje zapytanie do bazy danych (_context.Orders), aby pobrać zamówienia, które zawierają produkty danej firmy. Używa metody Include() do załadowania powiązanych OrderItems oraz Product.
- Filtrowanie następuje po tym, czy w zamówieniu znajduje się produkt przypisany do tego użytkownika.
- Zamówienia są ładowane asynchronicznie za pomocą ToListAsync().

3. **Walidacja zamówień:**

- Jeśli brak zamówień (lub nie znaleziono żadnych powiązanych z użytkownikiem), zwróci widok NoOrders.

4. Zwrócenie widoku:

- Jeśli zamówienia są dostępne, metoda zwraca widok MyOrders z listą zamówionych produktów.
- Jeśli nie ma zamówień, użytkownik zobaczy widok o braku zamówień.

Moje Zamówienia

ID Zamówienia	Data Zamówienia	Klient	Adres Dostawy	Sposób Płatności	Sposób Dostawy	Produkty
12	19-01-2025 15:40	John Doe	Kwiatowa 5	CreditCard	Express	• Majtkisfdgds - Ilość: 10 - Cena: \$29.00 - Razem: \$290.00

Kontroller AdminPanel

Opis kontrolera AdminController

Kontroler AdminController obsługuje funkcje związane z administracją aplikacji, które mogą być wykonywane tylko przez użytkowników posiadających rolę Admin. Zarządza on produktami, użytkownikami, a także pozwala na tworzenie i edytowanie użytkowników.

1. AdminPanel

GET

Metoda ta ładuje dane do panelu administratora, gdzie administrator może przeglądać wszystkie produkty oraz użytkowników.

- **Parametry:**
 - Brak otrzymywanych parametrów
- **Funkcjonalność:**

1. Pobranie produktów i użytkowników:

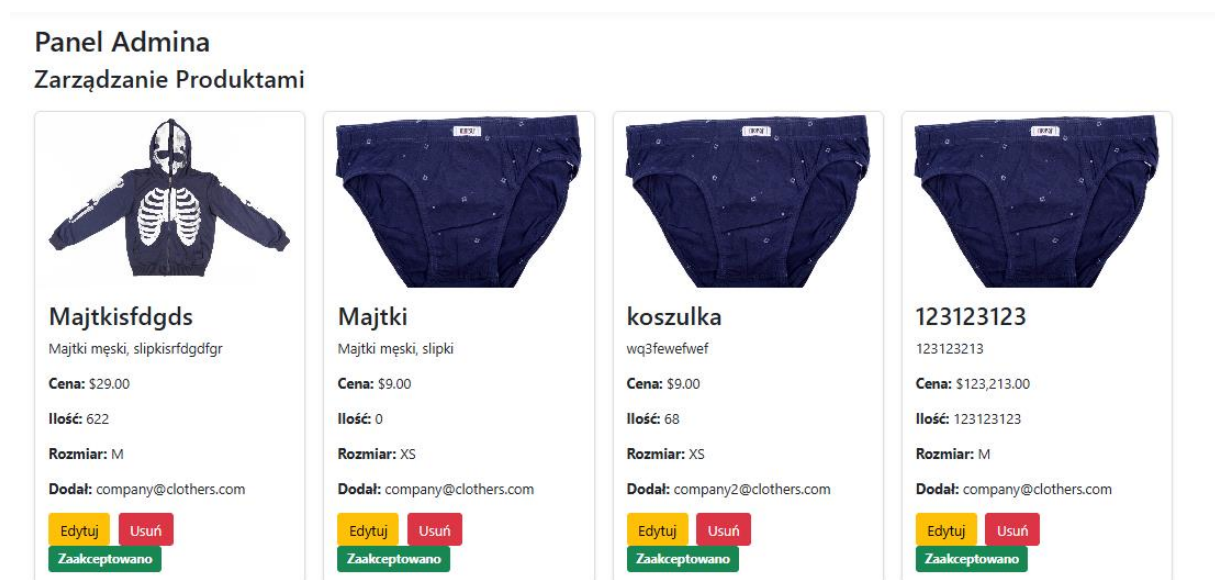
- Produkty są pobierane z bazy danych razem z powiązonym użytkownikiem (właścicielem produktu) za pomocą Include(p => p.User).
- Użytkownicy są pobierani z UserManager przy pomocy ToListAsync().

2. Przygotowanie modelu:

- Tworzy obiekt AdminPanelViewModel, który zawiera listy produktów i użytkowników.

3. Zwrócenie widoku:

- Zwraca widok AdminPanel z załadowanym modelem. W przypadku błędu w ładowaniu danych, metoda loguje wyjątek i zwraca kod statusu 500.



2. Accept

POST

Metoda ta zatwierdza (akceptuje) ofertę produktu, ustawiając pole IsApproved na true.

- **Parametry:**
 - id - identyfikator produktu, który ma zostać zaakceptowany.
- **Funkcjonalność:**

1. Wyszukiwanie produktu:

- Używa metody FindAsync(id) do pobrania produktu o podanym identyfikatorze.

2. Aktualizacja produktu:

- Jeśli produkt istnieje, pole IsApproved jest ustawiane na true i zapisane w bazie danych.

3. Zwrócenie przekierowania:

- Po zapisaniu zmian, użytkownik jest przekierowywany z powrotem do panelu administratora (AdminPanel).

3. DeleteUser

POST

Metoda ta usuwa użytkownika z systemu na podstawie jego identyfikatora.

- **Parametry:**
 - id - identyfikator użytkownika, który ma zostać usunięty.
- **Funkcjonalność:**

1. Wyszukiwanie użytkownika:

- Używa FindByIdAsync(id) do znalezienia użytkownika o podanym identyfikatorze.

2. Usunięcie użytkownika:

- Używa metody DeleteAsync(user) z UserManager do usunięcia użytkownika.

3. Zwrócenie odpowiedzi:

- Jeśli usunięcie użytkownika zakończy się sukcesem, użytkownik zostaje przekierowany do panelu administratora. W przeciwnym przypadku zwróci błąd.

Zarządzanie Użytkownikami

[Dodaj Użytkownika](#)

Email	Role	Akcje
company@clothers.com	Company	Edytuj Usuń
company23@clothers.com	SiteUser	Edytuj Usuń
company2@clothers.com	Company	Edytuj Usuń
user@clothers.com	SiteUser	Edytuj Usuń
admin@clothers.com	Admin	Edytuj Usuń
user33@clothers.com	SiteUser	Edytuj Usuń

4. CreateUser

GET

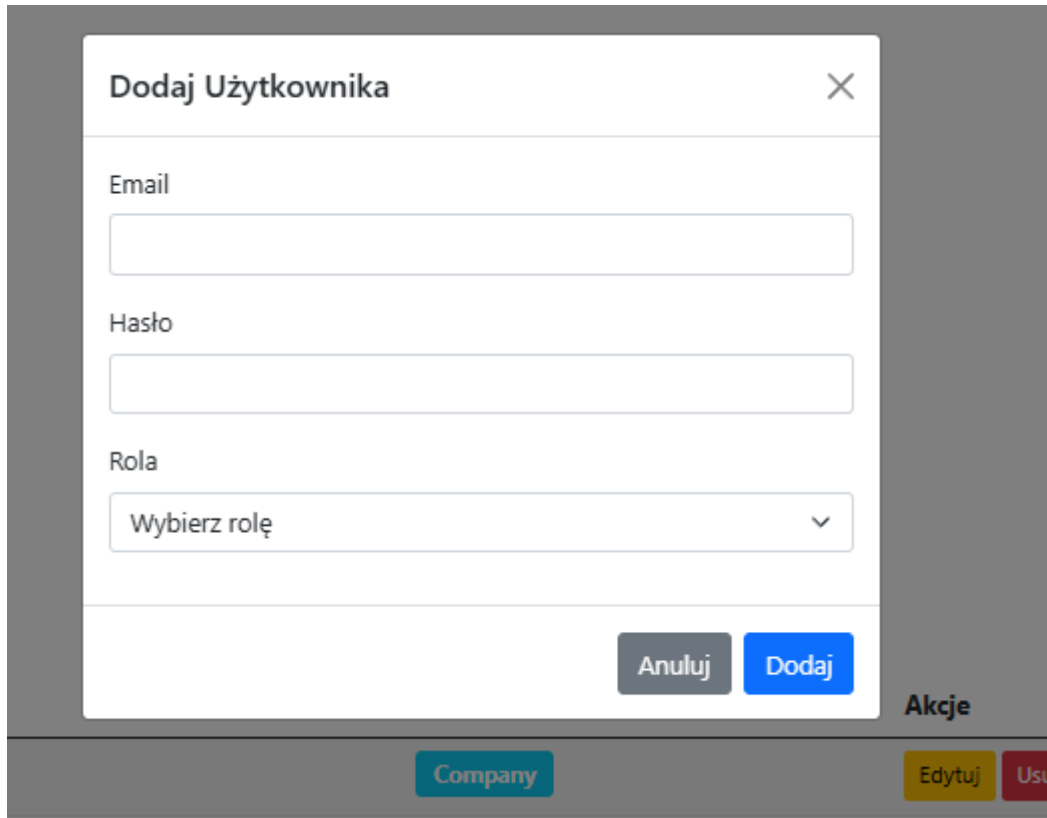
Metoda ta wyświetla formularz do tworzenia nowego użytkownika w systemie.

- **Parametry:**
 - Brak otrzymywanych parametrów

- **Funkcjonalność:**

1. **Zwrócenie części widoku:**

- Zwraca fragment widoku `_CreateUserPartial`, który umożliwia wprowadzenie danych nowego użytkownika.



5. **CreateUser (POST)**

POST

Metoda ta obsługuje proces tworzenia nowego użytkownika. Sprawdza, czy użytkownik o podanym adresie e-mail już istnieje, a następnie tworzy użytkownika i przypisuje go do odpowiedniej roli.

- **Parametry:**
 - model - model zawierający dane do stworzenia nowego użytkownika.

- **Funkcjonalność:**

1. **Sprawdzenie unikalności e-maila:**

- Przed utworzeniem nowego użytkownika sprawdzany jest warunek, czy użytkownik o podanym adresie e-mail już istnieje.

2. **Tworzenie użytkownika:**

- Jeśli e-mail jest unikalny, tworzy nowego użytkownika za pomocą `CreateAsync()`.

3. **Przypisanie roli:**

- Jeżeli użytkownik ma przypisaną rolę, jest ona dodawana za pomocą AddToRoleAsync().

4. Zwrócenie odpowiedzi:

- Po pomyślnym utworzeniu użytkownika użytkownik jest przekierowywany do panelu administratora.

6. EditUser (GET)

GET

Metoda ta wyświetla formularz do edytowania danych użytkownika, w tym jego adresu e-mail i przypisanych ról.

- **Parametry:**
 - id - identyfikator użytkownika, którego dane mają zostać edytowane.
- **Funkcjonalność:**
 1. Pobranie użytkownika i jego ról:
 - Wyszukuje użytkownika za pomocą FindByIdAsync(id) oraz pobiera jego przypisane role.
 2. **Zwrócenie widoku:**
 - Przekazuje dane do widoku _EditUserPartial w celu edycji danych użytkownika.

Role	Akcje
Company	Edytuj Usun
SiteUser	Edytuj Usun

7. EditUser (POST)

POST

Metoda ta aktualizuje dane użytkownika, w tym e-mail oraz role.

- **Parametry:**
 - model - model zawierający dane do edycji użytkownika.
- **Funkcjonalność:**

1. Sprawdzenie unikalności e-maila:

- Sprawdzany jest warunek, czy adres e-mail jest już używany przez innego użytkownika.

2. Aktualizacja danych użytkownika:

- Jeśli walidacja przejdzie pomyślnie, adres e-mail użytkownika jest aktualizowany.

3. Przypisanie ról:

- Użytkownikowi przypisywane są odpowiednie role (po usunięciu poprzednich).

4. Zapisanie zmian:

- Zaktualizowane dane są zapisywane za pomocą UpdateAsync(). Jeśli operacja jest udana, użytkownik jest przekierowywany do panelu administratora.

Kontroller OrdersController

1. Create()

- HTTP Methods: GET

- **Parametry:**

- Brak

- **Funkcjonalność:**

1. Pobiera koszyk użytkownika wraz z przedmiotami i powiązanymi produktami z bazy danych.
2. Sprawdza, czy koszyk zawiera produkty. Jeśli koszyk jest pusty, użytkownik zostaje przekierowany do strony koszyka z odpowiednim komunikatem.
3. Przygotowuje pusty model OrderViewModel.
 - **Zwracane dane:** Widok Create z modelem zamówienia.

2. Create(OrderViewModel model)

- HTTP Methods: POST

- **Parametry:**

- model (*OrderViewModel*): Dane wprowadzone przez użytkownika w formularzu zamówienia.

- **Funkcjonalność:**

1. Sprawdza poprawność modelu.
2. Pobiera koszyk użytkownika i jego produkty.
3. Tworzy zamówienie na podstawie danych użytkownika oraz produktów z koszyka.
4. Aktualizuje stan magazynowy produktów.
5. Usuwa produkty z koszyka.
6. Zapisuje zamówienie w bazie danych.
 - **Zwracane dane:**
 - W przypadku sukcesu: Przekierowanie do akcji Confirmation z orderId.
 - W przypadku błędów: Widok Create z błędami walidacji.

Złóż Zamówienie

Imię
Mikołaj

Nazwisko
Kowalski

Adres Dostawy
Kwiatowa 5

Sposób Płatności
Karta Kredytowa

Sposób Dostawy
Odbiór Osobisty

Złóż Zamówienie

3. Confirmation(int orderId)

- HTTP Methods: **GET**
 - Parametry:
 - orderId (*int*): Identyfikator zamówienia.
 - Funkcjonalność:
1. Pobiera zamówienie użytkownika z bazy danych, uwzględniając powiązane produkty.
 2. Sprawdza, czy zamówienie należy do zalogowanego użytkownika.
 - Zwracane dane: Widok Confirmation z modelem zamówienia.

4. MyOrders()

- HTTP Methods: **GET**
 - Parametry:
 - Brak
 - Funkcjonalność:
1. Pobiera zamówienia zawierające produkty należące do zalogowanego użytkownika o roli Company.
 2. Uwzględnia powiązane produkty i dane zamówień.
 - Zwracane dane: Widok MyOrders z listą zamówień.

Potwierdzenie Zamówienia

Dziękujemy za złożenie zamówienia, Mikołaj Kowalski!

Numer zamówienia: 13

Data zamówienia: 19-01-2025 16:04

Szczegóły Zamówienia

Nazwa Produktu	Ilość	Cena Jednostkowa	Razem
Majtkisfdgds	5	\$29.00	\$145.00

Suma Totalna: \$145.00

Pobierz Potwierdzenie PDF

5. **DownloadPdf**(int orderId)

- HTTP Methods: **GET**
- **Parametry:**
 - orderId (*int*): Identyfikator zamówienia.

- **Funkcjonalność:**

1. Pobiera zamówienie użytkownika z bazy danych, uwzględniając produkty.
2. Sprawdza, czy zamówienie należy do zalogowanego użytkownika.
3. Generuje plik PDF ze szczegółami zamówienia za pomocą usługi OrderPdfGenerator.
 - **Zwracane dane:** Plik PDF (application/pdf) z nazwą w formacie Order_{orderId}.pdf.

Potwierdzenie Zamówienia #13

Data Zamówienia: 19-01-2025 16:04

Klient: Mikołaj Kowalski

Adres Dostawy: Kwiatowa 5

Sposób Płatności: CreditCard

Sposób Dostawy: Pickup

Produkty:

Nazwa Produktu	Ilość	Cena Jednostkowa	Razem
Majtkisfdgds	5	\$29.00	\$145.00

Suma Totalna: \$145.00

Nasz projekt zawiera 3 role; **SiteUser**, **Company**, **Admin**

Jak nadać rolę użytkownikowi?

Można to zrobić na 2 sposoby:

- 1) Wybór użytkownika przy rejestracji

Zarejestruj się

Stwórz nowe konto na naszej stronie

☒ Zarejestruj się jako użytkownik☐ Zarejestruj się jako Firma

- 2) W admin panelu, admin może stworzyć nowego użytkownika i przypisać mu jakąkolwiek rolę z dostępnych albo edytować już istniejącego użytkownika i zmienić mu rolę.

Edytuj Użytkownika

Email

company@clothers.com

Role

Admin

UżytkownikStrony

Firma

AnulujZapisz

Role	Akcje
Company	EdytujUsuń
SiteUser	EdytujUsuń
Company	EdytujUsuń
SiteUser	EdytujUsuń
Admin	EdytujUsuń
SiteUser	EdytujUsuń

Jakie możliwości mają użytkownicy?

1)**Gość** -> niezalogowany użytkownik może:

- przeglądać produkty, które zostały dodane i zaakceptowane przez administratora strony
- korzystać z paska wyszukiwania ubrań
- utworzyć konto na stronie albo się na już istniejące konto zalogować

2)**SiteUser** -> normalny zalogowany użytkownik może:

- to samo co Gość

- dodać ubranie do koszyka
- usunąć rzecz z koszyka
- złożyć zamówienie
- po złożeniu zamówienia pobrać potwierdzenie zamówienia w formie PDF-a

3)**Company** -> użytkownik zarejestrowany jako firma może:

- wystawiać nowe oferty ubrań, edytować oraz je usuwać
- posiada **2** specjalne panele stworzone właśnie dla tej strony:
 - Moje Oferty** – w tym widoku firma może sprawdzić tylko i wyłącznie te oferty, które ta firma wystawiła
 - Moje zamówienia** – w tym widoku firma może sprawdzić zamówienia, które zrobił użytkownik (jeśli użytkownik złoży zamówienie składające się z dwóch produktów od innych firm, te zamówienia zostaną rozdzielone pomiędzy dwie zakładki Moje zamówienia dla każdej z firm wystawiających te produkty)

4)**Admin** -> użytkownik zarejestrowany jako administrator może:

- wystawiać nowe oferty ubrań, edytować oraz je usuwać
- posiada **1** specjalny panel stworzony właśnie dla tej strony
 - Panel administratora** – w tym widoku admin musi nacisnąć przycisk Zaakceptuj, żeby oferta pojawiła się na stronie głównej oraz w tym panelu może dodawać, edytować oraz usuwać użytkowników.

Informacje powiązane z użytkownikiem:

- Koszyk użytkownika (Cart i CartItem).
- Dane osobowe i adresowe (Order).
- Historia zamówień (OrderItems).
- Produkty dodane przez użytkownika (jeśli użytkownik jest właścicielem produktów).

Informacje Globalne:

- Wszystkie produkty dostępne w systemie (widoczne publicznie po zatwierdzeniu).
- Role użytkowników i ich uprawnienia.
- Metody płatności i dostawy.

Najciekawsze funkcjonalności

1) Generowanie PDF-a, jako potwierdzenia zamówienia

ClothersStrona GłównaKoszyk

Hello user@clothers.com!Logout

Potwierdzenie Zamówienia

Dziękujemy za złożenie zamówienia, John Doe!

Numer zamówienia: 12

Data zamówienia: 19-01-2025 15:40

Szczegóły Zamówienia

Nazwa Produktu	Ilość	Cena Jednostkowa	Razem
koszulka	5	\$9.00	\$45.00
Majtkisfdgds	10	\$29.00	\$290.00

Suma Totalna: \$335.00

Pobierz Potwierdzenie PDF

Order_12.pdf

1 / 175%+

Potwierdzenie Zamówienia #12

Data Zamówienia: 19-01-2025 15:40

Klient: John Doe

Adres Dostawy: Kwiatowa 5

Sposób Płatności: CreditCard

Sposób Dostawy: Express

Produkty:

Nazwa Produktu	Ilość	Cena Jednostkowa	Razem
koszulka	5	\$9.00	\$45.00
Majtkisfdgds	10	\$29.00	\$290.00

Suma Totalna: \$335.00

Dziękujemy za zakupy w Clothers!

Aktywuj system Windows

Przejdź do ustawień aby aktywować system Windows

2) Wyszukiwarka ubrań po nazwie

Produkty

Szukaj produktów...

Szukaj

