

Tytuł projektu: Shopifex - Sklep Internetowy

Opis projektu:

Projekt Shopifex to aplikacja e-commerce umożliwiająca użytkownikom przeglądanie i zakup produktów online. Kluczowe funkcje sklepu obejmują zarządzanie katalogiem produktów, koszykiem, składanie zamówień. Panel administracyjny pozwala właścicielom na dodawanie, edytowanie i usuwanie produktów, zarządzanie zamówieniami oraz monitorowanie sprzedaży.

Główne funkcjonalności:

- **Rejestracja i logowanie:** Użytkownicy mogą zakładać konta, logować się.
- **Katalog produktów:** Przeglądanie listy produktów, filtrowanie ich na podstawie kategorii. Produkty będą miały zdjęcie.
- **Koszyk:** Umożliwia dodawanie i edytowanie ilości produktów przed złożeniem zamówienia.
- **Zamówienia:** Użytkownik może złożyć zamówienie na wybrany produkt / produkty. Projekt nie będzie jednak zawierał integracji z bramką płatności.
- **Panel administracyjny:** Wyłącznie dla użytkowników posiadających rolę administratora. Umożliwia zarządzanie użytkownikami, produktami, kategoriami, zamówieniami.
- **Statystyki:** Dostępne wyłącznie dla administratorów. Zagregowane dane dotyczące zamówień od zalogowanych użytkowników i gości oraz najpopularniejszych produktów.
- **Zarządzanie koszykami:** Zalogowany użytkownik będzie miał możliwość zapisywania swoich koszyków i przywracania ich.
- **Historia zakupów:** Zalogowany użytkownik będzie mógł przeglądać swoją historię zakupów, wraz z szczegółami – zamówione produkty, cena.

Podział ról zarejestrowanych użytkowników będzie następujący Administrator, Użytkownik.

Autor projektu:

- Maciej Wojtkowiak – pzx 110422

Wykorzystane technologie

1. C#

Aplikacja została stworzona w języku C#, który obsługuje logikę backendową. Umożliwia efektywne zarządzanie operacjami związanymi z danymi, autoryzacją użytkowników oraz interakcjami z bazą danych.

2. JavaScript

JavaScript zapewnia dynamiczność aplikacji. Umożliwia interaktywne funkcje, takie jak walidacja formularzy i aktualizacja treści bez przeładowania strony.

3. HTML

Struktura aplikacji opiera się na HTML, który definiuje układ i elementy interfejsu użytkownika.

4. CSS

CSS służy do stylizacji aplikacji, nadając jej atrakcyjny wygląd.

5. SQLite

Jako baza danych wybrano SQLite, co pozwala na łatwe przechowywanie i zarządzanie danymi aplikacji, takimi jak informacje o produktach, użytkownikach i zamówieniach, w prosty i efektywny sposób.

6. Git

Git jest używany do zarządzania wersjami kodu źródłowego, co umożliwia śledzenie zmian, współpracę z zespołem oraz zachowanie historii projektu.

7. GitHub

Dzięki GitHub aplikacja jest hostowana w chmurze, co ułatwia współpracę, przeglądanie kodu oraz zarządzanie projektem. Platforma ta wspiera również kontrolę jakości i przeglądanie kodu przez innych członków zespołu.

8. Visual Studio

Aplikacja była tworzona w Visual Studio, zintegrowanym środowisku programistycznym, które zapewnia narzędzia do efektywnego debugowania, zarządzania kodem.

Biblioteki i Frameworki

- **ASP.NET Core 8 MVC**

Framework ASP.NET Core 8 MVC jest fundamentem aplikacji, który pozwala na łatwe budowanie aplikacji webowych z wzorcem Model-View-Controller (MVC). Umożliwia to logiczne oddzielenie warstw aplikacji i ułatwia rozwój oraz utrzymanie kodu.

- **Entity Framework**

Entity Framework, jako technologia ORM (Object-Relational Mapping), upraszcza interakcje z bazą danych, pozwalając na zarządzanie danymi za pomocą obiektów C#. Dzięki temu deweloperzy mogą skupić się na logice biznesowej, a nie na szczegółach związanych z SQL.

- **Microsoft Identity**

System Identity zapewnia kompleksowe zarządzanie użytkownikami, umożliwiając rejestrację, logowanie i przypisywanie ról. Dzięki temu użytkownicy mogą korzystać z różnych funkcji aplikacji w zależności od swoich uprawnień.

- **Bootstrap**

Biblioteka CSS domyślnie obecna w Frameworku ASP.NET Core 8 MVC. Zapewnia gotowe „style” dla elementów.

Uruchomienie projektu.

1. Uruchomienie Visual Studio
2. Sklonowanie repozytorium <https://github.com/maciejwojtkowiak/Shopifex>
3. Uruchomienie serwera w Visual Studio
4. Projekt został uruchomiony.
5. By uzyskać dostęp do pełnej funkcjonalności, kliknij „Logowanie” na pasku



nawigacji. Zaloguj się jako administrator –

email: admin@shopifex.com

hasło: Admin@123

lub jakiekolwiek dane obecne w poniższej metodzie i pliku appsettings.json

```
private static async Task CreateAdminUser(UserManager<ApplicationUser> userManager, IConfiguration configuration)
{
    var adminEmail = configuration["AdminSettings:AdminEmail"];
    var adminPassword = configuration["AdminSettings:AdminPassword"];

    var adminUser = await userManager.FindByEmailAsync(adminEmail);
    if (adminUser == null)
    {
        var newAdmin = new ApplicationUser { UserName = adminEmail, Email = adminEmail };
        var createAdmin = await userManager.CreateAsync(newAdmin, adminPassword);
        if (createAdmin.Succeeded)
        {
            await userManager.AddToRoleAsync(newAdmin, Roles.Admin);
        }
    }
}
```

```
{
  "ConnectionStrings": {
    "ShopifexContext": "Data Source=Shopifex.db"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "AdminSettings": {
    "AdminEmail": "admin@shopifex.com",
    "AdminPassword": "Admin@123"
  }
}
```

- Po zalogowaniu się jako admin wymagane jest stworzenie danych testowych jeżeli baza danych (sqlite) jest pusta. Służą do tego „dashboardsy” obecne na pasku nawigacji.

Shopifex Wyloguj Historia zakupów Zapisane koszyki Admin - Produkty Admin - Statystyki Admin - Użytkownicy Admin - Zamówienia Admin - Kategorie Koszyk

Lista produktów

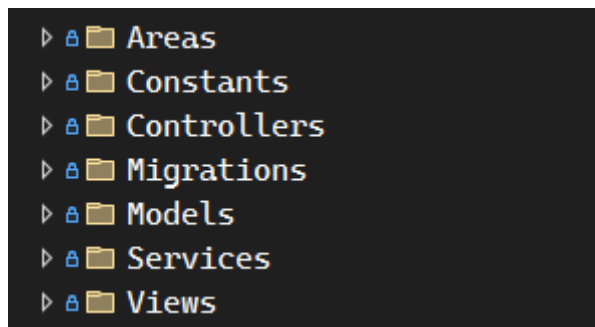
Filtruj według kategorii:

Wszystkie kategorie

Filtruj

- Następnie w celu przetestowania pełnej funkcjonalności wymagane jest wylogowanie się z konta. Sprawdzenie funkcjonalności gościa. Następnie rejestracja nowego użytkownika i sprawdzenie jego możliwości. Stworzony użytkownik nie będzie miał roli **Admina**.

Struktura projektu.



Projekt jest podzielony na foldery. Każdy z nich odpowiada za przechowywanie pewnych części aplikacji. Zastosowany wzorzec to MVC.

Areas

- Pliki .cshtml
- Przechowuje logikę logowania/rejestracji z biblioteki Identity.
- Przechowuje widoki Administratora. Kontrolery administratora mapują do Areas.
- Powodem zastosowania Areas dla administratora jest zastosowanie tych samych nazw kontrolerów. Mapowanie ścieżek za pomocą Areas w takim przypadku jest bardziej dogodne.

Constants

- Przechowuje stałe zmienne – role / wartości „enum”.

Controllers

- Przechowuje wszystkie kontrolery aplikacji. Jej logikę.

Migrations

- Folder Automatycznie utworzony po przez stworzenie migracji za użyciem Entity Framework.

Models

- Przechowuje „kontekst” bazy danych używany do interakcji z bazą danych za pomocą Entity Framework.
- Przechowuje wszystkie modele wykorzystywane jako tabele w bazie danych. Do tego jest część modeli wykorzystywana jako „modele widoku” używane tylko w komunikacji między widokiem a kontrolerem, nie bezpośrednio z bazą danych.

Services

- Folder przechowuje klasy odpowiadające modelom w bazie danych. Zawarte w tych klasach metody są wykorzystywane w wielu miejscach. Celem „serwisów” jest usunięcie duplikacji kodu.

Views

- Przechowuje widoki dla gości, użytkowników, adminów. Pliki .cshtml

Modele

Bazodanowe

Product

Model reprezentujący produkty dostępne w systemie e-commerce. Przechowuje informacje o każdym produkcie, takie jak nazwa, opis, cena, kategoria, do której produkt należy, oraz dane obrazu, które mogą być wykorzystywane do wyświetlania produktu w interfejsie użytkownika. Każdy produkt jest przypisany do konkretnej kategorii.

- Id (int) – Identyfikator produktu.
- Name (string) – Nazwa produktu
- Description (string) – Opis produktu.
- Price (decimal?) – Cena produktu.
- ImageData (string) – Dane obrazu produktu (base64)
- CategoryId (int) – Id kategorii, do której należy produkt.
- Category (Category) – Powiązany obiekt kategorii.

Walidacja dla Product:

- Name: Wymagana (Required), maksymalna długość 100 znaków.
- Description: Maksymalna długość 500 znaków.
- Price: Wymagana, minimalna wartość 0.01.
- ImageData: Maksymalna długość 500 znaków.
- CategoryId: Wymagane.

Category

Model reprezentujący kategorie, do których można przypisać produkty. Służy do grupowania produktów, co ułatwia ich przeglądanie i filtrowanie w interfejsie użytkownika. Każda kategoria może zawierać wiele produktów.

- Id (int) – Identyfikator kategorii.
- Name (string) – Nazwa kategorii. Wymagana, maksymalna długość 100 znaków.
- Products (List<Product>) – Lista produktów przypisanych do tej kategorii.

Walidacja dla Category:

- **Name:** Wymagana, maksymalna długość 100 znaków.

CartItem

Model reprezentujący pozycje w koszyku zakupowym. Każda pozycja odnosi się do konkretnego produktu i zawiera informację o liczbie sztuk danego produktu w koszyku. CartItem jest powiązany z modelem Cart, który przechowuje wszystkie pozycje w koszyku dla danego użytkownika lub zamówienia.

- Id (int) – Identyfikator pozycji koszyka.
- ProductId (int) – Id produktu w koszyku.
- Quantity (int) – Ilość produktu.
- Product (Product) – Powiązany obiekt produktu.

Walidacja dla CartItem:

- ProductId: Wymagane.
- Quantity – co najmniej 1, maksymalnie – zakres typu int

Cart

Model reprezentujący koszyk zakupowy użytkownika lub gościa. Zawiera listę pozycji (CartItem), które użytkownik dodał do swojego koszyka przed złożeniem zamówienia. Koszyk może być przypisany do użytkownika zarejestrowanego (za pomocą UserId) lub może funkcjonować bez użytkownika w przypadku gościa. Niezapisany koszyk jest przechowywany w sesji.

- Id (int) – Identyfikator koszyka.
- CartItems (List<CartItem>) – Lista pozycji w koszyku.
- UserId (string) – Id użytkownika powiązanego z koszykiem (opcjonalnie).

Walidacja dla Cart:

- Brak dodatkowych walidacji dla Cart.

Order

Model odpowiadający za przechowywanie informacji o zamówieniu złożonym przez użytkownika lub gościa. Zawiera szczegóły takie jak imię i nazwisko, adres dostawy, numer telefonu, email oraz powiązany koszyk z produktami zamówionymi przez klienta. Dodatkowo, zamówienie może być powiązane z użytkownikiem zarejestrowanym w systemie.

- Id (int) – Identyfikator zamówienia.
- FullName (string) – Imię i nazwisko zamawiającego.
- Address (string) – Adres dostawy
- Phone (string) – Numer telefonu zamawiającego.
- Email (string) – Email zamawiającego.
- CreatedAt (DateTime) – Data utworzenia zamówienia.
- Cart (Cart) – Koszyk powiązany z zamówieniem.
- UserId (string) – Id użytkownika składającego zamówienie (opcjonalnie).
- ApplicationUser (ApplicationUser) – Powiązany użytkownik (opcjonalnie).

Walidacja dla Order:

- FullName: Wymagane, maksymalna długość 200 znaków.
- Address: Wymagane, maksymalna długość 300 znaków.
- Phone: Wymagane, poprawny format numeru telefonu.
- Email: Wymagane, poprawny format adresu e-mail.

ApplicationUser

Klasa dziedzicząca po **IdentityUser (z biblioteki Microsoft Identity)**, używana do przechowywania danych dotyczących zarejestrowanych użytkowników w aplikacji. Obecnie nie posiada dodatkowych pól poza tymi dziedziczonymi z IdentityUser.

Modele Widoku (View-Models)

Modele widoku są potrzebne do przekazania danych nieprzechowywanych w bazie danych takich jak np. Statystyki, które są wynikiem operacji na danych wyciągniętych z bazy danych. Oraz do ograniczenia przekazywania danych klientowi jak i ułatwieniu komunikacji między frontendem czy backendem. W przypadku tego projektu takie wykorzystanie jest obce dla ApplicationUser. Model ApplicationUser nie jest wysyłany klientowi by zmniejszyć zakres przekazywanych danych osobowych.

Statistics

Model widoku używany do wyświetlania statystyk zamówień i sprzedaży w systemie. Służy do prezentowania danych takich jak liczba zamówień, łączna wartość zamówień, średnia wartość zamówienia oraz najlepiej sprzedające się produkty.

- TotalOrders (int) – łączna liczba zamówień.
- OrdersByGuests (int) – Liczba zamówień złożonych przez gości.
- OrdersByUsers (int) – Liczba zamówień złożonych przez zarejestrowanych użytkowników.
- TotalOrderPriceByGuests (decimal) – łączna wartość zamówień złożonych przez gości.
- AverageOrderPriceByGuests (decimal) – Średnia wartość zamówień złożonych przez gości.
- TotalOrderPriceByUsers (decimal) – łączna wartość zamówień złożonych przez użytkowników.
- AverageOrderPriceByUsers (decimal) – Średnia wartość zamówień złożonych przez użytkowników.
- TotalOrderPrice (decimal) – łączna wartość wszystkich zamówień.
- AverageOrderPrice (decimal) – Średnia wartość zamówień.
- TopProducts (List<TopProduct>) – Lista najlepiej sprzedających się produktów.

TopProduct

Model widoku przedstawiający informacje o najlepiej sprzedających się produktach w systemie. Wykorzystywany w statystykach.

- **ProductName (string)** – Nazwa najlepiej sprzedającego się produktu.
- **QuantitySold (int)** – Ilość sprzedanych sztuk produktu.

UserViewModel

Model widoku wykorzystywany do przekazania danych do widoku „Index” w panelu administracyjnym użytkowników. Ogranicza dane wystawione przez backend.

- **Email (string)** – Adres e-mail użytkownika.
- **Id (string)** – Identyfikator użytkownika
- **Role (string)** – Rola przypisana użytkownikowi.

CreateUserViewModel

Model widoku używany do tworzenia nowych użytkowników. Przechowuje dane takie jak adres e-mail, hasło oraz przypisaną rolę użytkownika. Jest wykorzystywany w procesie rejestracji użytkownika przez administratora systemu.

- **Email (string)** – Adres e-mail użytkownika.
- **Password (string)** – Hasło użytkownika.
- **Role (string)** – Rola przypisana użytkownikowi.

Walidacja dla CreateUserViewModel:

- **Email:** Wymagane, poprawny format adresu e-mail. Email nie może być zajęty – walidacja w kontrolerze.
- **Password:** Wymagane. Minimum 6 znaków, maksimum 100 znaki.
- **Role:** Wymagane.

EditUserViewModel:

Model widoku używany do edycji danych użytkownika, takich jak adres e-mail oraz przypisana rola. Jest wykorzystywany w panelu administratora do zarządzania użytkownikami.

- Email (string) – Adres e-mail użytkownika.
- Role (string) – Rola użytkownika.

Walidacja dla EditUserViewModel:

- Email: Wymagane, poprawny format adresu e-mail. Email nie może być zajęty – walidacja w kontrolerze.
- Role: Wymagane.

OrderCreateViewModel

Model reprezentujący dane wymagane do utworzenia nowego zamówienia. Zawiera informacje o zamawiającym, adresie dostawy, statusie zamówienia oraz listę produktów wybranych przez użytkownika wraz z ich ilością.

- FullName (string) – Imię i nazwisko osoby zamawiającej. Wymagane.
- Address (string) – Adres dostawy zamówienia. Wymagane.
- Phone (string) – Numer telefonu osoby zamawiającej. Wymagane, walidowane jako poprawny numer telefonu.
- Email (string) – Adres e-mail osoby zamawiającej. Wymagane, walidowane jako poprawny adres e-mail.
- Status (OrderStatusEnum) – Status zamówienia, umożliwia śledzenie postępu zamówienia.
- Products (List<ProductSelectionViewModel>) – Lista wybranych produktów, wraz z ich ilościami. Każdy produkt zawiera informacje o wyborze i liczbie sztuk.

Walidacja dla OrderCreateViewModel:

- FullName: Pole wymagane.
- Address: Pole wymagane.
- Phone: Pole wymagane, musi być poprawnym numerem telefonu.
- Email: Pole wymagane, musi być poprawnym adresem e-mail.

- **Products:** Pole wymagane, musi zawierać przynajmniej jeden wybrany produkt. Walidacja jest przeprowadzana w kontrolerze, nie za pomocą atrybutu w modelu.
- **Status** – wymagana poprawna wartość typu wyliczeniowego „InProgress” (1) albo „Completed” (2). Walidacja niestandardowa umieszczona w tym samym pliku co model.

ProductSelectionViewModel

Model reprezentujący pojedynczy produkt w zamówieniu, wybrany przez użytkownika. Zawiera informacje o identyfikatorze produktu, nazwie, statusie wyboru oraz ilości.

- **ProductId (int)** – Identyfikator produktu. Używany do powiązania z konkretnym produktem.
- **ProductName (string)** – Nazwa produktu. Wyświetlana użytkownikowi w formularzu zamówienia.
- **IsSelected (bool)** – Flaga określająca, czy produkt został wybrany przez użytkownika do zamówienia.
- **Quantity (int)** – Ilość wybranego produktu. Pole wymagane, minimalna wartość to 1 (jeśli produkt jest wybrany).

Walidacja dla ProductSelectionViewModel:

- **Quantity:** Pole wymagane, minimalna wartość to 1, ale tylko wtedy, gdy produkt jest zaznaczony jako wybrany (**IsSelected** = true). Ta walidacja jest przeprowadzana w kontrolerze.

ErrorViewModel

Model ErrorViewModel jest domyślnym modelem dostarczonym przez szablony ASP.NET do obsługi błędów w aplikacji. Reprezentuje informacje o błędach, które mogą wystąpić podczas działania aplikacji, oraz umożliwia ich wyświetlanie użytkownikowi w przyjazny sposób.

- **RequestId (string?)** – Identyfikator żądania, który może pomóc w śledzeniu konkretnego błędu. Wartość ta jest opcjonalna.
- **ShowRequestId (bool)** – Właściwość logiczna zwracająca true, jeśli RequestId nie jest puste. Używana, aby określić, czy identyfikator żądania powinien zostać wyświetlony na stronie błędu.

Kontrolery

Kontrolery administratorów

CategoryController

1. Index

Metoda HTTP: GET

Parametry: Brak

Opis działania: Wyświetla listę wszystkich kategorii z bazy danych.

Zwracane dane: Widok z listą kategorii (List<Category>).

Widok:

Shopifex Wyloguj Historia zakupów Zapisane koszyki Admin - Produkty Admin - Statystyki Admin - Użytkownicy Admin - Zamówienia Admin - Kategorie Koszyk	
Lista kategorii	
Dodaj kategorię	
Nazwa	Akcje
a	Edytuj Szczegóły Usuń

2. Create (GET)

Metoda HTTP: GET

Parametry: Brak

Opis działania: Wyświetla formularz do utworzenia nowej kategorii.

Zwracane dane: Widok z pustym modelem kategorii (Category).

Widok:

Shopifex Wyloguj Historia zakupów Zapisane koszyki Admin - Produkty Admin - Statystyki Admin - Użytkownicy Admin - Zamówienia Admin - Kategorie Koszyk	
Dodaj kategorię	
Nazwa kategorii	
<input type="text"/>	
Dodaj	

3. Create (POST)

Metoda HTTP: POST

Parametry: Category category – obiekt kategorii przekazany z formularza.

Opis działania: Tworzy nową kategorię na podstawie danych przesłanych przez użytkownika. Po pomyślnym dodaniu, przekierowuje do akcji Index. W przypadku błędów walidacji zwraca ponownie formularz.

Zwracane dane: Przekierowanie do Index po sukcesie, lub formularz z błędami walidacji w przypadku niepowodzenia.

4. Edit (GET)

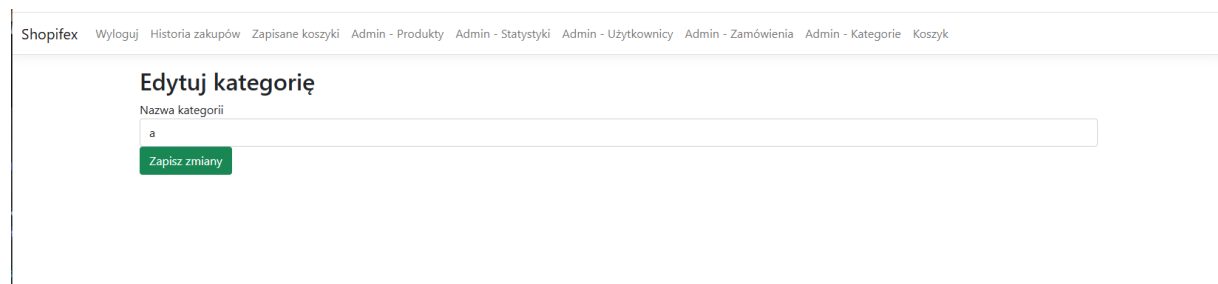
Metoda HTTP: GET

Parametry: int id – identyfikator kategorii do edycji.

Opis działania: Wyświetla formularz do edycji kategorii na podstawie podanego ID. Jeśli kategoria nie istnieje, zwraca błąd 404.

Zwracane dane: Widok z modelem edytowanej kategorii (Category).

Widok:



Shopifex Wyloguj Historia zakupów Zapisane koszyki Admin - Produkty Admin - Statystyki Admin - Użytkownicy Admin - Zamówienia Admin - Kategorie Koszyk

Edytuj kategorię

Nazwa kategorii

Zapisz zmiany

5. Edit (POST)

Metoda HTTP: POST

Parametry: Category category – zaktualizowany obiekt kategorii.

Opis działania: Aktualizuje kategorię w bazie danych. Po pomyślnym zapisaniu zmian, przekierowuje do akcji Index. W przypadku błędów walidacji zwraca ponownie formularz edycji.

Zwracane dane: Przekierowanie do Index po sukcesie, lub formularz z błędami walidacji w przypadku niepowodzenia.

6. Delete (GET)

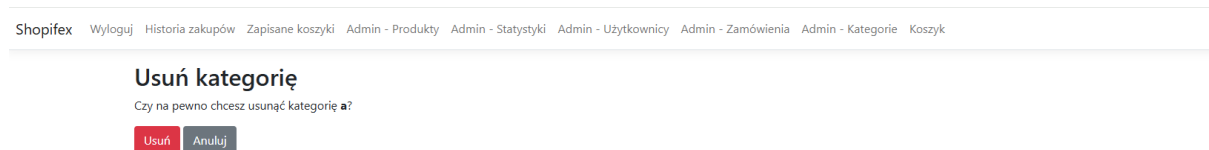
Metoda HTTP: GET

Parametry: int id – identyfikator kategorii do usunięcia.

Opis działania: Wyświetla widok potwierdzenia usunięcia kategorii. Jeśli kategoria nie istnieje, zwraca błąd 404.

Zwracane dane: Widok z modelem kategorii do usunięcia (Category).

Widok:



7. DeleteConfirmed (POST)

Metoda HTTP: POST

Parametry: int id – identyfikator kategorii do usunięcia.

Opis działania: Usuwa kategorię, jeśli nie jest przypisana do żadnych produktów. Jeśli kategoria ma przypisane produkty, zwraca komunikat o błędzie i nie usuwa kategorii.

Zwracane dane: Przekierowanie do Index po sukcesie. W przypadku kategorii z przypisanymi produktami – przekierowanie do Index z komunikatem o błędzie.

8. Details

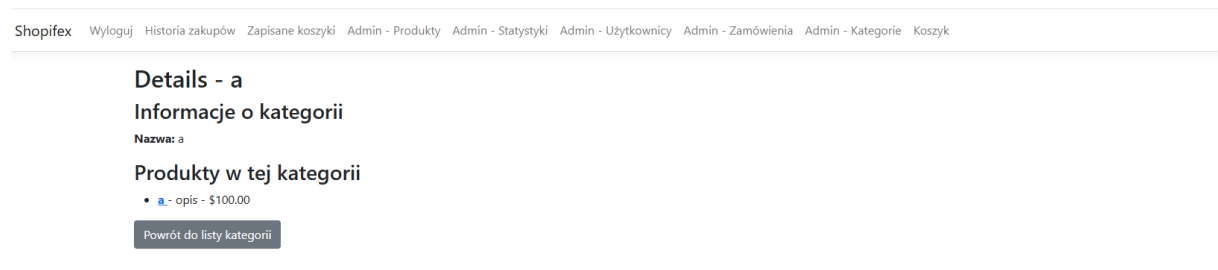
Metoda HTTP: GET

Parametry: int id – identyfikator kategorii do wyświetlenia szczegółów.

Opis działania: Wyświetla szczegóły kategorii oraz listę produktów przypisanych do tej kategorii. Jeśli kategoria nie istnieje, zwraca błąd 404.

Zwracane dane: Widok ze szczegółami kategorii (Category).

Widok:



OrderController

1. Index

Metoda HTTP: GET

Parametry: Brak

Opis działania: Wyświetla listę wszystkich zamówień w bazie danych. Pobiera wszystkie zamówienia z kontekstu bazy danych i przekazuje je do widoku.

Zwracane dane: Widok z listą zamówień (List<Order>).

Widok:

Lista Zamówień				
Nr Zamówienia	Klient	Email	Data Utworzenia	Akcje
1	a	oscypek567@gmail.com	10/26/2024 11:21 AM	Szczegóły Edytuj Usuń

2. Details

Metoda HTTP: GET

Parametry: int id – identyfikator zamówienia do wyświetlenia szczegółów.

Opis działania: Pobiera zamówienie na podstawie podanego ID oraz dołącza powiązany koszyk i jego pozycje. Jeśli zamówienie nie istnieje, zwraca błąd 404.

Zwracane dane: Widok ze szczegółami zamówienia (Order), w tym informacje o koszyku i pozycjach zamówienia.

Widok:

[Shopifex](#) [Wyloguj](#) [Historia zakupów](#) [Zapisane koszyki](#) [Admin - Produkty](#) [Admin - Statystyki](#) [Admin - Użytkownicy](#) [Admin - Zamówienia](#) [Admin - Kategorie](#) [Koszyk](#)

Szczegóły Zamówienia #1

Imię i nazwisko: a

Adres: a

Telefon: 669-63132

Email: oscypek567@gmail.co

Data zamówienia: 10/26/2024 11:59 AM

Produkty

Produkt	Ilość	Cena jednostkowa	Łączna cena
a	1	\$100.0	\$100.0

[Powrót](#)

3. Edit (GET)

Metoda HTTP: GET

Parametry: int id – identyfikator zamówienia do edycji.

Opis działania: Wyświetla formularz do edycji zamówienia na podstawie podanego ID. Jeśli zamówienie nie istnieje, zwraca błąd 404.

Zwracane dane: Widok z modelem edytowanego zamówienia (Order).

Widok:

Shopifex Wyloguj Historia zakupów Zapisane koszyki Admin - Produkty Admin - Statystyki Admin - Użytkownicy Admin - Zamówienia Admin - Kategorie Koszyk

Edytuj Zamówienie #1

Imię i nazwisko

Adres

Numer telefonu

Email

Status

4. Edit (POST)

Metoda HTTP: POST

Parametry: Order order – zaktualizowany obiekt zamówienia.

Opis działania: Aktualizuje zamówienie w bazie danych. Po pomyślnym zapisaniu zmian, przekierowuje do akcji Index. W przypadku błędów walidacji zwraca ponownie formularz edycji.

Zwracane dane: Przekierowanie do Index po sukcesie, lub formularz z błędami walidacji w przypadku niepowodzenia.

5. Delete (GET)

Metoda HTTP: GET

Parametry: int id – identyfikator zamówienia do usunięcia.

Opis działania: Wyświetla widok potwierdzenia usunięcia zamówienia. Jeśli zamówienie nie istnieje, zwraca błąd 404.

Zwracane dane: Widok z modelem zamówienia do usunięcia (Order).

6. DeleteConfirmed (POST)

Metoda HTTP: POST

Parametry: int id – identyfikator zamówienia do usunięcia.

Opis działania: Usuwa zamówienie z bazy danych na podstawie podanego ID. Jeśli zamówienie nie istnieje, zwraca błąd 404. Po pomyślnym usunięciu przekierowuje do akcji Index.

Zwracane dane: Przekierowanie do Index po sukcesie.

7. Create

Metoda HTTP: GET

Parametry: Brak

Opis działania: Metoda Create wyświetla formularz do utworzenia nowego zamówienia. Pobiera listę dostępnych produktów z serwisu `_productService` i przekazuje ją do widoku poprzez `ViewData["Products"]`. Inicjalizuje nowy obiekt Order z pustym koszykiem (Cart) i przekazuje go jako model do widoku, umożliwiając użytkownikowi dodanie produktów i ich ilości do zamówienia.

Zwracane dane: Widok formularza tworzenia nowego zamówienia z listą dostępnych produktów.

Widok:

Shopifex Wyloguj Historia zakupów Zapisane koszyki Admin - Produkty Admin - Statystyki Admin - Użytkownicy Admin - Zamówienia Admin - Kategorie Koszyk

Utwórz nowe zamówienie

Imię i nazwisko

Adres

Numer telefonu

Adres e-mail

Status

W trakcie

Produkty

Produkt	Ilość
<input type="checkbox"/> film	<div>0</div>
<input type="checkbox"/> ks	<div>0</div>

Utwórz zamówienie

Anuluj

8. Create (POST)

Metoda HTTP: POST

Parametry:

OrderCreateViewModel model – model widoku z danymi wymaganymi do stworzenia zamówienia,

Opis działania: Metoda Create obsługuje tworzenie nowego zamówienia na podstawie danych z formularza.

- Najpierw wybiera produkty, które użytkownik zaznaczył (IsSelected), oraz sprawdza, czy co najmniej jeden produkt jest wybrany i ma ustawioną ilość większą niż 0.
- Jeśli żaden produkt nie został wybrany lub ilość dla wybranego produktu jest równa lub mniejsza od 0, dodaje odpowiednie komunikaty błędów walidacyjnych.
- Gdy model przejdzie walidację, tworzy Cart, który zawiera wybrane produkty i ich ilości.
- Następnie tworzy nowe zamówienie Order, przypisując do niego dane kontaktowe użytkownika oraz utworzony koszyk (Cart).
- Na końcu zapisuje zamówienie w bazie danych za pomocą _orderService i przekierowuje użytkownika do widoku listy zamówień (Index).

Zwracane dane:

W przypadku powodzenia: Przekierowanie do widoku Index z listą zamówień.

W przypadku błędów walidacyjnych: Ponownie wyświetla formularz z informacjami o błędach.

StatisticsController

Metody kontrolera

Index

Metoda HTTP: GET

Parametry: Brak

Opis działania:

Metoda zbiera dane o wszystkich zamówieniach, w tym całkowitą liczbę zamówień, zamówienia złożone przez gości, zamówienia złożone przez zarejestrowanych użytkowników, oraz analizuje wartość zamówień. Oblicza całkowitą wartość zamówień złożonych przez gości i użytkowników, a także średnią wartość tych zamówień. Zbiera pięć najlepiej sprzedających się produktów.

Zwracane dane:

Zwraca widok z modelem **Statistics**

Widok:

[Shopifex](#) [Wyloguj](#) [Historia zakupów](#) [Zapisane koszyki](#) [Admin - Produkty](#) [Admin - Statystyki](#) [Admin - Użytkownicy](#) [Admin - Zamówienia](#) [Admin - Kategorie](#) [Koszyk](#)

Statystyki Zamówień

	Liczba zamówień	Łączna cena	Średnia cena
Goście	0	\$0.00	\$0.00
Użytkownicy	1	\$200.00	\$200.00
Łącznie	1	\$200.00	\$200.00

Top 5 Najpopularniejszych Produktów

Produkt	Liczba sprzedanych sztuk
Książka	2

UserController

Opis kontrolera

Kontroler UserController jest odpowiedzialny za zarządzanie użytkownikami w systemie. Tylko administratorzy mogą uzyskiwać dostęp do tego kontrolera. Umożliwia on tworzenie, edytowanie, usuwanie oraz przeglądanie szczegółowych informacji o użytkownikach.

Metody kontrolera

Index

Metoda HTTP: GET

Parametry: Brak

Opis działania: Zbiera listę wszystkich użytkowników z systemu i wyświetla ją na stronie.

Zwracane dane: Widok z listą użytkowników. List<UserViewModel>

Widok:

Shopifex Wyloguj Historia zakupów Zapisane koszyki Admin - Produkty Admin - Statystyki Admin - Użytkownicy Admin - Zamówienia Admin - Kategorie Koszyk		
Użytkownicy		
Dodaj nowego użytkownika		
Email	Rola	Akcje
admin@shopifex.com	admin	Nie możesz edytować ani usunąć konta na którym jesteś zalogowany
oscypek567@gmail.com	user	Edytuj Usuń

Create

Metoda HTTP: GET

Parametry: Brak

Opis działania: Wyświetla formularz do tworzenia nowego użytkownika.

Zwracane dane: Widok z formularzem do tworzenia użytkownika.

Widok:

Shopifex Wyloguj Historia zakupów Zapisane koszyki Admin - Produkty Admin - Statystyki Admin - Użytkownicy Admin - Zamówienia Admin - Kategorie Koszyk

Dodaj nowego użytkownika

Email

Password

Role

Create (POST)

Metoda HTTP: POST

Parametry: CreateUserViewModel model

Opis działania: Przetwarza formularz tworzenia nowego użytkownika. Sprawdza, czy adres e-mail jest już zajęty. Jeśli nie, tworzy nowego użytkownika i przypisuje mu rolę, jeśli taka została określona.

Zwracane dane:

W przypadku sukcesu: Przekierowanie do akcji Index.

W przypadku błędów: Powrót do formularza z komunikatami o błędach.

Edit

Metoda HTTP: GET

Parametry: string id

Opis działania: Wyświetla formularz edycji dla użytkownika o podanym identyfikatorze. Nie jest możliwym wyświetlenie tego widoku dla edycji obecnie zalogowanego administratora.

Zwracane dane: Widok z formularzem edycji użytkownika. Model EditUserViewModel

Widok:

Shopifex Wyloguj Historia zakupów Zapisane koszyki Admin - Produkty Admin - Statystyki Admin - Użytkownicy Admin - Zamówienia Admin - Kategorie Koszyk

Edytuj użytkownika

Email

admin@shopifex.com

Role

admin

Zapisz zmiany

Anuluj

Edit (POST)

Metoda HTTP: POST

Parametry: string id, EditUserViewModel model

Opis działania: Przetwarza formularz edycji użytkownika. Sprawdza, czy adres e-mail jest już zajęty przez innego użytkownika. Aktualizuje dane użytkownika i jego rolę. Nie jest możliwym edycja aktualnie zalogowanego administratora.

Zwracane dane:

W przypadku sukcesu: Przekierowanie do akcji Index.

W przypadku błędów: Powrót do formularza z komunikatami o błędach.

Delete

Metoda HTTP: GET

Parametry: string id

Opis działania: Wyświetla formularz potwierdzenia usunięcia użytkownika.

Zwracane dane: Email i ID użytkownika zwracane w ViewData oraz widok usunięcia.

Widok:

Shopifex Wyloguj Historia zakupów Zapisane koszyki Admin - Produkty Admin - Statystyki Admin - Użytkownicy Admin - Zamówienia Admin - Kategorie Koszyk

Usuń użytkownika

Czy na pewno chcesz usunąć tego użytkownika?

oscypek567@gmail.com

Delete (POST)

Metoda HTTP: POST

Parametry: string id

Opis działania: Przetwarza formularz potwierdzenia usunięcia użytkownika. Usuwa użytkownika z systemu.

Zwracane dane:

W przypadku sukcesu: Przekierowanie do akcji Index.

W przypadku błędów: Powrót do formularza z komunikatami o błędach.

Kontrolery nie-administratorskie

Kontrolery nie-administratorskie są używane dla pozostałych użytkowników – zalogowanych lub nie zalogowanych. Nie mapują widoków do folderu Areas. Widoki tych kontrolerów są w folderze Views.

CartController

Index

Metoda HTTP: GET

Opis działania: Wyświetla aktualny koszyk użytkownika. Metoda korzysta z CartService, aby pobrać koszyk z sesji przy użyciu metody GetCartFromSession(). Jeśli koszyk nie istnieje w sesji, zostaje stworzony nowy koszyk z przypisanym UserId. Następnie koszyk jest przekazywany do widoku, gdzie użytkownik może przeglądać i edytować zawartość.

Zwracane dane: Widok z aktualnymi pozycjami w koszyku.

Widok:

Koszyk

Produkt	Ilość	Cena	Akcja
Książka	<div>- 1 +</div>	\$100.00	<div>Usuń</div>

Łączna cena: 100.0 zł

[Przejdź do zamówienia](#)

[Zapisz koszyk](#)

AddToCart

Metoda HTTP: POST

Parametry: int productId, int quantity

Opis działania: Dodaje produkt do koszyka na podstawie jego ID oraz ilości. Serwis ProductService pobiera produkt z bazy danych, a CartService dodaje produkt do koszyka za pomocą metody AddToCart(). Produkt zostaje następnie zapisany w sesji przy użyciu metody SaveCartToSession(). Ilość produktu musi być co najmniej: 1. Inaczej użytkownik jest przekierowany do listy produktów i wyświetlany jest błąd na stronie listy produktów.

Zwracane dane: Przekierowanie do akcji Index (widoku koszyka), wraz z modelem koszyka „Cart”.

RemoveFromCart

Metoda HTTP: POST

Parametry: int productId

Opis działania: Usuwa produkt z koszyka na podstawie jego ID. Serwis CartService pobiera aktualny koszyk z sesji, usuwa wybrany produkt za pomocą metody RemoveFromCart(), a następnie zapisuje zmiany w sesji.

Zwracane dane: Przekierowanie do akcji Index.

SavedCarts

Metoda HTTP: GET

Wymaga autoryzacji: Tak

Opis działania: Pobiera zapisane koszyki użytkownika z bazy danych przy użyciu metody GetUserSavedCarts(). Serwis CartService wyszukuje koszyki zapisane przez

użytkownika na podstawie jego UserId oraz na podstawie atrybutu „IsSavedByUser”, który rozróżnia koszyki zapisane przez użytkownika i przez system (podczas tworzenia zamówienia). Koszyki te są przekazywane do widoku (IEnumerable<Cart>), który wyświetla listę zapisanych koszyków.

Zwracane dane: Widok z listą zapisanych koszyków. Jeżeli koszyk nie istnieje lub nie został utworzony przez zalogowanego użytkownika zwraca błąd 404.

Widok:

Shopifex Wyloguj Historia zakupów Zapisane koszyki Admin - Produkty Admin - Statystyki Admin - Użytkownicy Admin - Zamówienia Admin - Kategorie Koszyk

Zapisane koszyki			
Nr Koszyka	Łączna Cena	Ilość Produktów	Akcje
2	100.0 zł	1	Pokaż Usuń

SaveCart

Metoda HTTP: POST

Wymaga autoryzacji: Tak

Opis działania: Zapisuje aktualny koszyk użytkownika w bazie danych jako zapisany koszyk. CartService pobiera koszyk z sesji, oznacza go jako zapisany przy użyciu właściwości IsSavedByUser i zapisuje w bazie danych za pomocą metody AddToDatabase(). Produkty w koszyku są również dodawane do kontekstu bazy danych, by zapobiec powórnemu zapisaniu tych samych produktów do bazy danych przez Entity Framework.

Zwracane dane: Przekierowanie do widoku zapisanych koszyków.

Details

Metoda HTTP: GET

Parametry: int id

Wymaga autoryzacji: Tak

Opis działania: Wyświetla szczegóły zapisanego koszyka na podstawie jego ID. CartService pobiera koszyk z bazy danych przy użyciu metody GetSavedCartById(). Kontroler sprawdza, czy koszyk należy do zalogowanego użytkownika poprzez

porównanie UserId koszyka z aktualnym użytkownikiem. Jeśli koszyk nie istnieje, zwracany jest błąd 404, a jeśli koszyk nie należy do użytkownika – błąd 403.

Zwracane dane: Widok z informacjami o zapisie koszyka, lub błąd 404 jeśli koszyk nie istnieje, lub błąd 403 jeśli koszyk nie należy do użytkownika.

Widok:

shopifex Wyloguj Historia zakupów Zapisane koszyki Admin - Produkty Admin - Statystyki Admin - Użytkownicy Admin - Zamówienia Admin - Kategorie Koszyk

Szczegóły koszyka nr 1

Produkt	Ilość	Cena jednostkowa	Łączna cena
a	1	\$100.00	\$100.00

Łączna cena: 100.0 \$

[Powrót do zapisanych koszyków](#)

[Usuń](#)

[Załaduj](#)

DeleteSavedCart

Metoda HTTP: POST

Parametry: int id

Wymaga autoryzacji: Tak

Opis działania: Usuwa zapisany koszyk z bazy danych. CartService pobiera koszyk na podstawie ID i usuwa go z bazy danych za pomocą metody DeleteSavedCart(). Po usunięciu koszyka użytkownik jest przekierowany do widoku listy zapisanych koszyków. Próba usunięcia koszyka, który nie istnieje, lub usunięcie koszyka przez użytkownika, który nie jest jego właścicielem zwróci błąd 404.

Zwracane dane: Przekierowanie do widoku zapisanych koszyków.

RestoreSavedCart

Metoda HTTP: POST

Parametry: int id

Wymaga autoryzacji: Tak

Opis działania: Przywraca zapisany koszyk jako aktualny koszyk użytkownika. CartService przy użyciu metody ChangeCart() zmienia aktywny koszyk użytkownika, przypisując mu produkty z zapisanego koszyka. Zmieniony koszyk jest następnie zapisywany w sesji.

Zwracane dane: Przekierowanie do akcji Index (widoku koszyka), wraz z danymi z koszyka, który został przywrócony.

UpdateQuantity

Metoda HTTP: POST

Parametry: int productId, string action

Opis działania: Aktualizuje ilość produktu w koszyku. Metoda pobiera koszyk z sesji przy użyciu CartService. Następnie szuka produktu na podstawie productId i odpowiednio zwiększa lub zmniejsza ilość produktu. Jeśli ilość produktu zostanie zmniejszona do 1, produkt zostanie usunięty z koszyka. Po modyfikacji koszyk jest zapisywany w sesji za pomocą metody SaveCartToSession().

Zwracane dane: Przekierowanie do akcji Index. Model „Cart” z zaktualizowaną ilością produktu.

OrderController

Index (GET)

Metoda HTTP: GET

Opis działania: Wyświetla widok formularza zamówienia. Kontroler pobiera aktualny koszyk użytkownika przy użyciu serwisu CartService (metoda GetCart()). Jeśli koszyk jest pusty lub nie istnieje, użytkownik zostaje przekierowany do strony głównej. Jeśli koszyk zawiera produkty, zostaje utworzony nowy obiekt zamówienia, który zostaje przekazany do widoku.

Zwracane dane: Widok formularza zamówienia. Wraz z modelem „Order”, jeżeli koszyk użytkownika istnieje – czyli wtedy kiedy użytkownik wybrał już produkt.

Widok:

[Shopifex](#) [Wyloguj](#) [Historia zakupów](#) [Zapisane koszyki](#) [Koszyk](#)

Koszyk

Produkt	Ilość	Cena
Książka	1	\$100.00

Łączna cena: \$100.0

Formularz zamówienia

Imię i nazwisko

Adres

Numer telefonu

Email

Zamów

Index (POST)

Metoda HTTP: POST

Opis działania: Przesyła dane zamówienia, gdy użytkownik złoży zamówienie. Serwis CartService dodaje koszyk do bazy danych, a zamówienie zostaje oznaczone jako InProgress. Jeśli model zamówienia jest niepoprawny (sprawdzone przez ModelState), użytkownik zostaje zwrócony do formularza zamówienia. Jeśli zamówienie jest poprawne, użytkownik zostaje przekierowany do widoku potwierdzenia zamówienia.

Dane zamówienia są zapisywane w bazie za pomocą serwisu OrderService (metoda AddOrder()). Jeżeli wszystko się powiedzie to czyści koszyk w sesji.

Zwracane dane: Widok potwierdzenia zamówienia. Jeżeli walidacja się nie powiedzie to zwraca widok zamówienia wraz z wyświetlonymi błędami.

OrderHistory

Metoda HTTP: GET

Wymaga autoryzacji: Tak

Opis działania: Wyświetla historię zamówień zalogowanego użytkownika. Serwis OrderService pobiera zamówienia użytkownika na podstawie jego UserId (metoda GetUserOrders()), które następnie zostają przekazane do widoku.

Zwracane dane: Widok historii zamówień użytkownika. Od najmłodszych do najstarszych. Model IEnumerable<Order>.

Widok:

[Shopifex](#) [Wyloguj](#) [Historia zakupów](#) [Zapisane koszyki](#) [Koszyk](#)

Historia zamówień

Data	Cena	
10/23/2024 9:37:05 PM	200.0	Szczegóły
10/25/2024 9:40:36 PM	100.0	Szczegóły

Details

Metoda HTTP: GET

Parametry: int id

Wymaga autoryzacji: Tak

Opis działania: Wyświetla szczegóły zamówienia na podstawie jego ID. Serwis OrderService pobiera wszystkie zamówienia użytkownika (metoda GetUserOrders()), a następnie kontroler wyszukuje zamówienie o podanym ID. Jeśli zamówienie nie istnieje lub nie należy do zalogowanego użytkownika, zostaje zwrócony błąd 404.

Zwracane dane: Widok szczegółów zamówienia (model Order). Lub błąd 404.

Widok:

Shopifex Wyloguj Historia zakupów Zapisane koszyki Koszyk

Szczegóły zamówienia nr 1

Dane zamówienia

Imię i nazwisko: a

Adres: gostynska12

Telefon: 669064371

Email: admin@shopifex.com

Data zamówienia: 23-10-2024 21:37

Produkty w zamówieniu

Produkt	Ilość	Cena jednostkowa	Łączna cena
Książka	2	\$100.00	200.0\$

Łączna cena zamówienia: 200.0\$

[Powrót do zamówień](#)

OrderConfirmation

Metoda HTTP: GET

Opis działania: Wyświetla widok potwierdzenia zamówienia. Metoda sprawdza, czy dane o zamówieniu są dostępne w TempData. Jeśli tak, wyświetla widok potwierdzenia zamówienia, a następnie czyści koszyk użytkownika przy użyciu serwisu CartService (metoda ClearCart()). Jeśli brak danych o zamówieniu, użytkownik zostaje przekierowany na stronę główną (Przypadek, wejścia użytkownika bezpośrednio przez adres URL).

Zwracane dane: Widok potwierdzenia zamówienia. Model „Order”.

Widok:

Shopifex Wyloguj Historia zakupów Zapisane koszyki Koszyk

Potwierdzenie zamówienia

Twoje dane:

- **Pełne imię i nazwisko:** a
- **Adres:** a
- **Telefon:** 669-63132
- **Email:** oscypek567@gmail.com

Twoje zamówienie zostało pomyślnie złożone. Dziękujemy za zakupy!

[Powrót do strony głównej](#)

ProductController

Index

Metoda HTTP: GET

Parametry: int? categoryId (opcjonalne ID kategorii)

Opis działania: Wyświetla listę wszystkich produktów z możliwością filtrowania po kategorii. Kontroler pobiera wszystkie produkty przy użyciu serwisu ProductService (metoda GetAllProducts()). Jeśli parametr categoryId zostanie podany, produkty są filtrowane na podstawie przypisanej kategorii. Następnie dane kategorii są przekazywane do widoku przy użyciu ViewData jako lista wyboru (SelectList), co pozwala użytkownikowi na wybór kategorii produktów.

Zwracane dane: Widok zawierający listę produktów, z opcjonalnym filtrem kategorii. Model IEnumerable<Products> i „SelectList” na podstawie kategorii stworzonych w bazie danych.

Widok

Shopifex Wyloguj Historia zakupów Zapisane koszyki Admin - Produkty Admin - Statystyki Admin - Użytkownicy Admin - Zamówienia Admin - Kategorie Koszyk

Lista produktów

Filtruj według kategorii:

Wszystkie kategorie

Filtruj



Ksiazka

ksiazka

Cena: \$100.00

Ilość: 1

Dodaj do koszyka

System użytkowników

System użytkowników w aplikacji Shopifex opiera się na integracji z Microsoft Identity. Aplikacja obsługuje różne typy użytkowników, takie jak Administrator, Zarejestrowany Użytkownik oraz Gość, z jasno określonymi uprawnieniami i dostępem do funkcjonalności systemu.

1. **Administrator (Admin):** Administrator ma najwyższy poziom uprawnień, umożliwiając zarządzenie produktami, kategoriami, użytkownikami i zamówieniami. Może dodawać, edytować i usuwać produkty oraz kategorie, zarządzać kontami użytkowników, przeglądać zamówienia i statystyki sprzedaży, co pozwala na efektywne zarządzanie sklepem.
2. **Zarejestrowany Użytkownik (User):** Zarejestrowany użytkownik posiada konto, co umożliwia korzystanie z dodatkowych funkcji. Może składać zamówienia, przeglądać historię zakupów, zarządzać swoimi danymi oraz zapisywać koszyk zakupów.
3. **Gość (Guest):** Gość może przeglądać produkty i składać zamówienia bez rejestracji. Jego możliwości są ograniczone, ponieważ nie ma dostępu do historii zamówień ani możliwości zapisywania koszyków na przyszłość.

Nadawanie ról:

Każdy użytkownik rejestrujący się na stronie otrzymuje rolę „user”.

Rola Admina jest nadawane przez innych administratorów w „dashboardzie” użytkowników. Pierwszy admin, jest tworzony w kodzie.

Modele globalne:

- Product
- Category
- User

Modele powiązane z użytkownikiem.

- Cart
- Order

Krótką charakterystyka najciekawszych funkcjonalności

1. Statystyki sprzedaży i zamówień

Funkcjonalność statystyk w aplikacji Shopifex dostarcza administratorowi kluczowe informacje na temat sprzedaży i aktywności użytkowników.

Łączna liczba zamówień: Wyświetlana jest całkowita liczba zamówień, co pozwala administratorowi śledzić ogólny poziom aktywności w sklepie.

Zamówienia składane przez gości i użytkowników: System oddzielnie gromadzi dane o zamówieniach składanych przez zarejestrowanych użytkowników i gości, co pomaga ocenić, która grupa klientów jest bardziej aktywna.

Łączna i średnia wartość zamówień: Administrator może monitorować zarówno łączną wartość wszystkich zamówień, jak i średnią wartość zamówienia – zarówno dla gości, jak i zarejestrowanych użytkowników.

Najlepiej sprzedające się produkty: Lista najczęściej kupowanych produktów pozwala administratorowi analizować trendy zakupowe.

2. Zapisywanie i przywracanie koszyka

Funkcjonalność zapisywania i przywracania koszyka umożliwia zarejestrowanym użytkownikom kontynuowanie zakupów bez utraty wcześniej dodanych produktów, nawet po opuszczeniu sklepu. Koszyk jest przypisany do konta użytkownika i zapisany w bazie danych, co pozwala na jego automatyczne przywrócenie przy kolejnych logowaniach. To ułatwia dokończenie zakupów, zwiększa wygodę użytkownika, a także zapobiega porzucaniu koszyków.