Euro-Moto S.p. zoo 28.11.2024
Adres:
Telefon:

Euro-Moto

Sklep internetowy z częściami samochodowymi

Omówienie działania projektu:
Euro-Moto to aplikacja webowa umożliwiająca sprzedaż części samochodowych, ze szczególnym naciskiem na tłumiki. Użytkownicy mogą przeglądać katalog produktów, filtrować je według kryteriów takich jak np. rodzaj części, producent. Administratorzy zarządzają produktami i kategoriami produktów za pośrednictwem dedykowanego panelu.

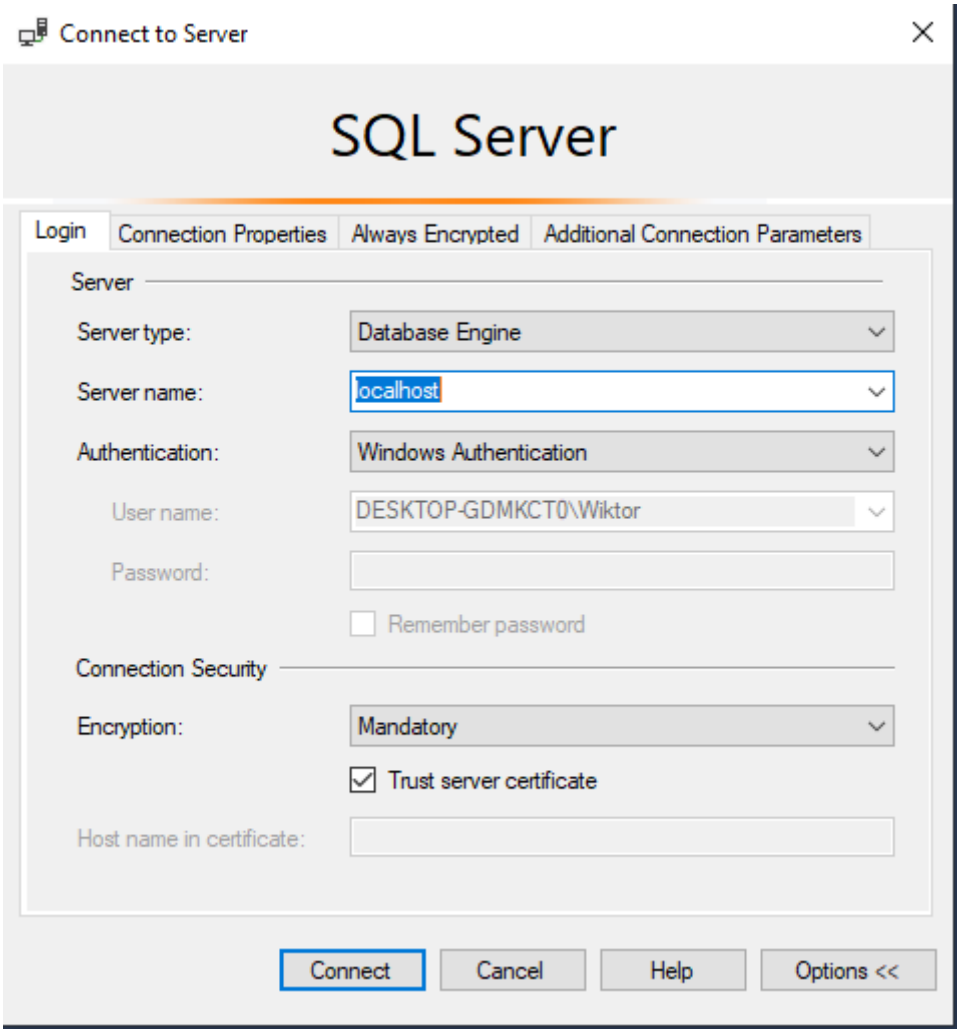
Autor:
Wiktor Matuszak

Specyfikacja technologii:
- C# .NET 8
- ASP.NET
- MS SQL Server
- CSS Bootstrap

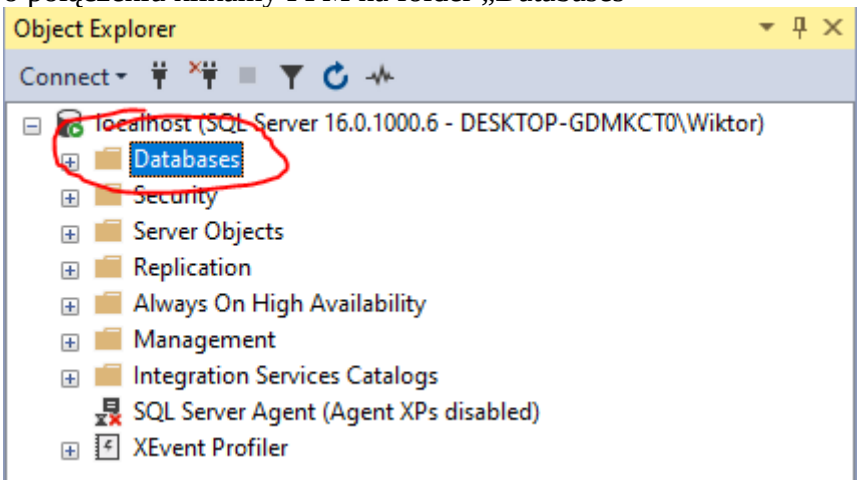
Instrukcja pierwszego uruchomienia:

Do pierwszego uruchomienia potrzebne są:
-Visual Studio 2022
-Microsoft SQL Server 2022
-SQL Server Management Studio 20

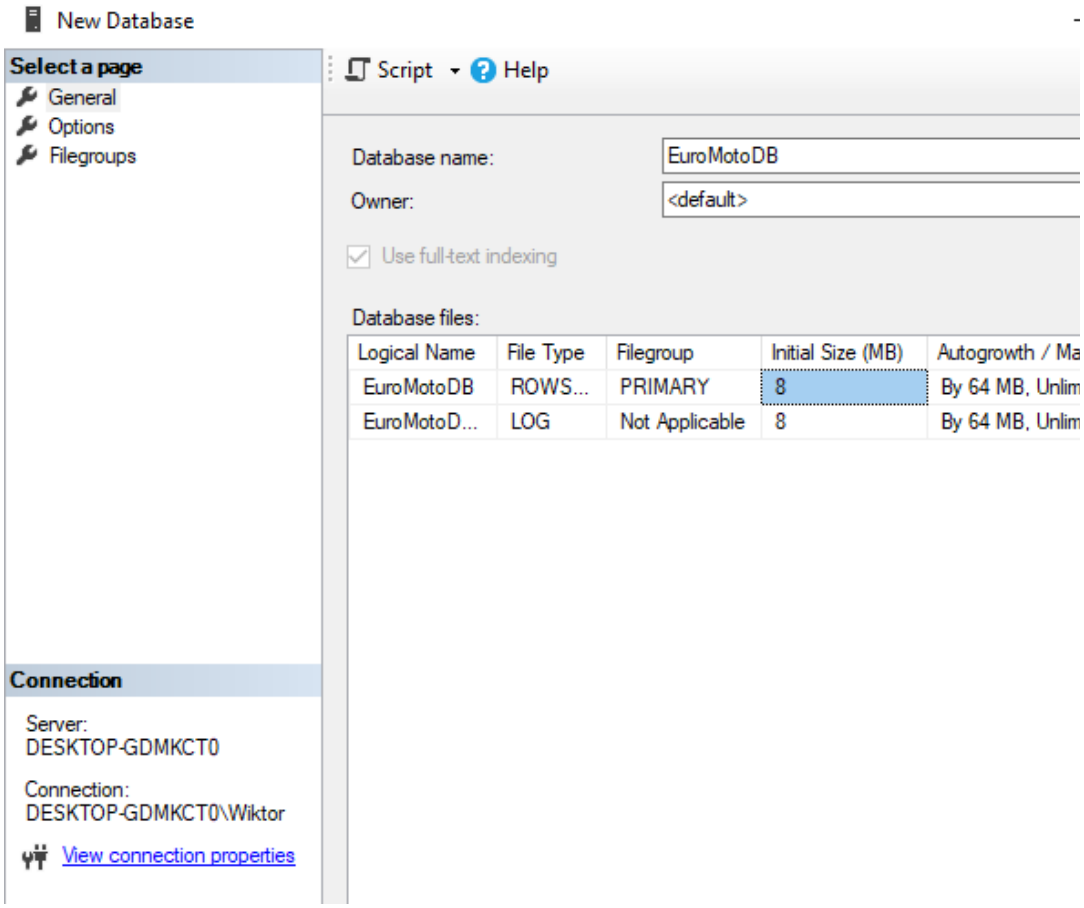
1. Utworzenie bazy danych:
-uruchamiamy SQL Server Management Studio 20
-łączymy się do serwera localhost, ustawienia tak jak na poniższym zrzucie ekranu



-po połączeniu klikamy PPM na folder „Databases”



-następnie klikamy „New Database”



-nazwa bazy danych to „EuroMotoDB”.
-Tworzymy bazę danych.

2. Uruchamiamy Visual Studio 2022:
   -Wybieramy „Otwórz projekt lub rozwiązanie”
   -Wskazujemy miejsce w którym zapisał się projekt
   -Wybieramy rozwiązanie i otwieramy
   -Po otwarciu z paska górnego wybieramy „Narzędzia”>”Menedżer pakietów NuGet”>”Konsola menedżera pakietów”
   -pamiętaj zmienić projekt domyślny na: Euro.DataAccess
   -w konsoli wpisujemy „add-migration przykladowa_nazwa_migracji” i wciskamy enter
   -po pomyślnym wykonaniu migracji w konsoli wpisujemy „update-database”
   -po pomyślnym updacie możemy uruchomić projekt.
3. Konta użytkowników:
   -automatycznie nie zostały dodane, lecz można samodzielnie zarejestrować konto klienta jak i administratora.
   -po rejestracji administratora można dodać/usunąć/edytować produkt lub kategorię w zakładce zarządzaj, klient nie ma do niej dostępu.
   -zdjęcia poglądowe produktów nie zostały dodane, można je też dodać bezpośrednio z dysku.

Struktura projektu:

A) Kontrolery odpowiedzialne za logikę aplikacji:

- ProductController:

Atrybuty:
-[Area("Admin")]: Określa, że kontroler należy do obszaru administracyjnego --(Admin).
-[Authorize(Roles = SD.Role_Admin)]: Ogranicza dostęp do kontrolera tylko dla użytkowników z rolą administratora (SD.Role_Admin).

Wstrzykiwanie zależności:
-_unitOfWork: Repozytorium do zarządzania operacjami na danych w sposób jednolity.
-_webHostEnvironment: Służy do uzyskiwania ścieżki do katalogu wwwroot.

Metody kontrolera:
Index:
-Zwraca widok główny listy produktów.
-Pobiera wszystkie produkty z repozytorium, uwzględniając ich kategorie -(includeProperties: "Category").
-Dane są przekazywane do widoku w formie listy List<Product>.

Upsert (get):
-Obsługuje zarówno tworzenie nowego produktu, jak i edycję istniejącego.
-Parametry:

- id:
    - null lub 0: Tworzenie nowego produktu.
    - Inna wartość: Pobieranie istniejącego produktu do edycji.

-Widok:

- Tworzony jest obiekt ProductVM (model widoku), który zawiera:
- CategoryList: Lista kategorii dla rozwijanej listy (SelectList).
- Product: Produkt do utworzenia lub edycji.

Upsert (post):

-Obsługuje zapis nowego lub edytowanego produktu.

-Logika:

- Waliduje model.
- Obsługuje przesyłanie plików (np. obrazów produktów):
    - Przechowuje plik w folderze images/product w katalogu wwwroot.
    - Usuwa stary plik obrazu, jeśli istnieje.
    - Generuje unikalną nazwę dla nowego pliku.
- Zapisuje dane do repozytorium:
    - Dodaje nowy produkt, jeśli Product.Id == 0.
    - Aktualizuje istniejący produkt, jeśli Product.Id != 0.
- Po pomyślnym zapisie przekierowuje do widoku Index.
- Jeśli model jest niepoprawny, ponownie renderuje widok Upsert z danymi.

Sekcja API:

GetAll (get):

- Zwraca wszystkie produkty w formacie JSON.

- Dane zawierają informacje o produktach wraz z powiązanymi kategoriami.

Delete:

Usuwa produkt o podanym `id`.

-**Logika**:

- Pobiera produkt z repozytorium.
- Jeśli produkt istnieje:
  - Usuwa powiązany plik obrazu z serwera.
  - Usuwa rekord z bazy danych.
- Zwraca wynik operacji w formacie JSON:
  - success: true, jeśli usunięcie się powiodło.
  - success: false w przypadku błędu.

-CategoryController:

Atrybuty:
[Area("Admin")]: Kontroler jest częścią obszaru administracyjnego (Admin)
[Authorize(Roles = SD.Role_Admin)]: Dostęp mają tylko użytkownicy z rolą administratora (SD.Role_Admin).

Wstrzykiwanie zależności:
-_unitOfWork: Repozytorium do zarządzania danymi, zapewniające operacje CRUD na kategoriach.

Metody kontrolera:
Index:
-Wyświetla listę wszystkich kategorii.
-Pobiera dane z repozytorium metodą GetAll() i przekazuje je do widoku w formie listy List<Category>.
Create (get):
-Zwraca widok umożliwiający utworzenie nowej kategorii.
Create (post):
-Obsługuje tworzenie nowej kategorii.
-Logika:

- Sprawdza, czy nazwa kategorii (Name) nie jest identyczna z kolejnością wyświetlania (DisplayOrder).
  - Jeśli są takie same, dodaje błąd walidacji do modelu (ModelState.AddModelError).
- Jeśli model jest poprawny:
  - Dodaje nową kategorię za pomocą metody Add() w repozytorium.
  - Zapisuje zmiany (Save()) i przekierowuje do widoku Index.
- W przypadku błędów walidacji renderuje ponownie widok z wprowadzonymi danymi.

Edit (get):
-Wyświetla widok edycji kategorii.

-Parametry:

- id: Identyfikator kategorii do edycji.

-Logika:

- Sprawdza, czy id jest null lub 0 – w takim przypadku zwraca błąd 404 (NotFound).
- Pobiera kategorię z repozytorium metodą Get() według identyfikatora.
- Jeśli kategoria nie istnieje, zwraca błąd 404.
- Jeśli kategoria istnieje, przekazuje ją do widoku.

Edit (post):

-Obsługuje zapis zmian w edytowanej kategorii

-Logika:

- Jeśli model jest poprawny:
- Aktualizuje kategorię za pomocą metody Update() w repozytorium.
- Zapisuje zmiany i przekierowuje do widoku Index.
- W przypadku błędów walidacji renderuje widok z wprowadzonymi danymi.

Delete (get):

-Wyświetla widok potwierdzenia usunięcia kategorii.

-Parametry:

- id: Identyfikator kategorii do usunięcia.

-Logika:

- Sprawdza, czy id jest null lub 0 – w takim przypadku zwraca błąd 404.
- Pobiera kategorię z repozytorium metodą Get() według identyfikatora.
- Jeśli kategoria nie istnieje, zwraca błąd 404.
- Jeśli kategoria istnieje, przekazuje ją do widoku.

Delete (post):

-Obsługuje usuwanie kategorii.

-Parametry:

- id: Identyfikator kategorii do usunięcia.

-Logika:

- Pobiera kategorię z repozytorium metodą Get().
- Jeśli kategoria nie istnieje, zwraca błąd 404.
- Jeśli kategoria istnieje:
    - Usuwa ją z repozytorium metodą Remove().
- Zapisuje zmiany i przekierowuje do widoku Index.

HomeController:

Atrybuty:

[Area("Customer")]: Określa, że kontroler należy do obszaru Customer.

Wstrzykiwanie zależności:

_logger: Służy do logowania zdarzeń i błędów w aplikacji.
_unitOfWork: Repozytorium do zarządzania danymi w aplikacji, umożliwiające operacje CRUD.

Metody:

Index:

Odpowiada za wyświetlenie głównej strony aplikacji dla klientów.

Logika:

- Pobiera listę wszystkich produktów z repozytorium metodą GetAll().
- Uwzględnia właściwość Category dzięki includeProperties: "Category", co pozwala na pobranie powiązanych danych o kategoriach.
- Przekazuje dane do widoku w postaci kolekcji IEnumerable<Product>.

Details:

Wyświetla szczegóły wybranego produktu.

Parametry:

- productId: Identyfikator produktu, którego szczegóły mają zostać wyświetlone.

Logika:

- Pobiera dane produktu z repozytorium metodą Get() na podstawie identyfikatora (productId).
- Uwzględnia właściwość Category, aby pobrać dane o powiązanej kategorii.
- Przekazuje dane produktu do widoku.

Privacy:

Wyświetla stronę polityki prywatności aplikacji.
Nie zawiera dodatkowej logiki, jedynie zwraca odpowiedni widok.

Error:

Obsługuje wyświetlanie strony błędu.

Logika:

- Tworzy obiekt ErrorViewModel, który zawiera identyfikator żądania (RequestId).
- Pobiera identyfikator z bieżącego kontekstu żądania (HttpContext.TraceIdentifier) lub z obiektu Activity.

- Przekazuje model błędu do widoku Error.

B)Definicje modeli danych:

ApplicationUser:

Dziedziczenie

- IdentityUser: Podstawowy model użytkownika dostarczany przez ASP.NET Core Identity, zawierający takie pola jak:
  - Id: Unikalny identyfikator użytkownika.
  - UserName: Nazwa użytkownika.
  - Email: Adres e-mail użytkownika.
  - PasswordHash: Hash hasła użytkownika.
  - I inne właściwości związane z zarządzaniem użytkownikami, np. blokowanie konta, weryfikacja e-mail.

Dodatkowe właściwości:

- **Name**:
  - Typ: string?
  - Atrybut: [Required] - Pole jest wymagane.
  - Opis: Przechowuje pełne imię i nazwisko użytkownika.
- **City**:
  - Typ: string?
  - Opis: Przechowuje nazwę miasta użytkownika.
- **State**:
  - Typ: string?
  - Opis: Przechowuje nazwę stanu lub województwa użytkownika.
- **PostalCode**:
  - Typ: string?
  - Opis: Kod pocztowy użytkownika.
- **StreetAddress**:
  - Typ: string?
  - Opis: Przechowuje adres ulicy użytkownika.

Category:

Właściwości:

- **Id**:
  - Typ: int
  - Atrybut: [Key] - Oznacza, że jest to klucz główny tabeli w bazie danych.
  - Opis: Unikalny identyfikator kategorii.
- **Name**:

        - Typ: string
        - Atrybuty:
            - [Required]: Pole jest wymagane.
            - [MaxLength(30)]: Maksymalna długość nazwy to 30 znaków.
            - [DisplayName("Nazwa Kategorii")]: Określa nazwę wyświetlaną w formularzach i interfejsach użytkownika jako "Nazwa Kategorii".
        - Opis: Przechowuje nazwę kategorii.
    - **DisplayOrder**:
        - Typ: int
        - Atrybuty:
            - [DisplayName("#")]: Nazwa wyświetlana w interfejsach użytkownika to "#".
            - [Range(1, 100, ErrorMessage = "Podaj # między 1-100")]:
                - Wartość musi mieścić się w przedziale od 1 do 100.
                - W przypadku naruszenia reguły walidacji wyświetlany jest komunikat: "Podaj # między 1-100".
        - Opis: Określa kolejność wyświetlania kategorii.

Przeznaczenie modelu:

Model Category jest używany do przechowywania danych o kategoriach produktów w bazie danych oraz do zarządzania nimi w aplikacji.

Walidacja

Model zawiera wbudowane mechanizmy walidacji, które zapewniają integralność danych:

- Wartość Name jest wymagana i nie może przekraczać 30 znaków.
- DisplayOrder musi mieścić się w zakresie 1–100, co zapobiega błędom podczas ustalania kolejności wyświetlania.

Zastosowania w aplikacji

- **Zarządzanie kategoriami**:
    - Może być używany w panelu administracyjnym do tworzenia, edytowania, wyświetlania i usuwania kategorii.
- **Organizacja produktów**:
    - Kategorie są kluczowe dla grupowania produktów, co ułatwia użytkownikom poruszanie się po ofercie.
- **Walidacja danych w formularzach**:
    - Dzięki atrybutom walidacji, aplikacja zapewnia poprawność wprowadzanych danych (np. wymagana nazwa, odpowiednia wartość dla DisplayOrder).
- **Wyświetlanie danych**:

- Atrybuty DisplayName poprawiają czytelność i dostosowują nazwy pól do języka użytkownika (np. "Nazwa Kategorii").

ErrorViewModel:

Właściwości

- RequestId
  - Typ: string? (nullable)
  - Opis:
    - Przechowuje identyfikator żądania (Request ID), który jest unikalny dla każdego zapytania HTTP.
    - Może być używany do śledzenia i diagnozowania problemów w aplikacji.
- ShowRequestId
  - Typ: bool
  - Opis:
    - Wartość wyliczana (ang. **computed property**) na podstawie tego, czy RequestId jest pusty lub null.
    - Zwraca true, jeśli RequestId ma wartość; w przeciwnym razie false.
    - Używana w widokach do określenia, czy identyfikator żądania powinien być wyświetlany użytkownikowi.

Product:

Właściwości

- **Id**:
  - Typ: int
  - Atrybut: [Key]
  - Opis: Unikalny identyfikator produktu.
- **Name**:
  - Typ: string
  - Atrybut: [Required]
  - Opis: Nazwa produktu. Pole jest obowiązkowe.
- **Description**:
  - Typ: string
  - Atrybut: [Required]
  - Opis: Szczegółowy opis produktu. Pole jest obowiązkowe.
- **Producent**:
  - Typ: string
  - Atrybut: [Required]
  - Opis: Nazwa producenta produktu. Pole jest obowiązkowe.
- **SerialNumber**:

        - Typ: string
        - Atrybut: [Required]
        - Opis: Numer seryjny produktu. Pole jest obowiązkowe.
- **ListPrice**:
    - Typ: double
    - Atrybuty:
        - [Required]: Pole jest obowiązkowe.
        - [Display(Name = "Cena katalogowa")]: Określa nazwę wyświetlaną w interfejsie użytkownika jako "Cena katalogowa".
    - Opis: Cena katalogowa produktu.
- **CategoryId**:
    - Typ: int
    - Opis: Klucz obcy łączący produkt z odpowiednią kategorią.
- **Category**:
    - Typ: Category
    - Atrybuty:
        - [ForeignKey("CategoryId")]: Określa, że ta właściwość jest kluczem obcym powiązanym z tabelą Category.
        - [ValidateNever]: Wyłącza walidację modelu dla tej właściwości.
    - Opis: Obiekt reprezentujący kategorię, do której należy produkt.
- **ImageURL**:
    - Typ: string
    - Atrybut: [ValidateNever]
    - Opis: Adres URL obrazu produktu. Nie jest poddawany walidacji modelu.

Przeznaczenie modelu

Model Product jest używany do przechowywania danych o produktach, ich kategoriach oraz powiązanych obrazach.

Walidacja

Model zawiera wbudowane mechanizmy walidacji:

- Pola Name, Description, Producent, SerialNumber, oraz ListPrice są wymagane.
- CategoryId jest kluczem obcym, co wymusza poprawność powiązania z kategorią.
- ImageURL oraz Category są wyłączone z walidacji, co pozwala uprościć operacje na modelu.

Zastosowania w aplikacji

- **Zarządzanie produktami**:
    - Tworzenie, edytowanie, usuwanie oraz wyświetlanie produktów w panelu administracyjnym.
- **Organizacja produktów**:

        - Powiązanie produktów z kategoriami ułatwia użytkownikom wyszukiwanie i przeglądanie oferty.
    - **Prezentacja w katalogach**:
        - Dane takie jak ImageURL, ListPrice, i Description są kluczowe dla prezentacji produktów w widoku katalogu.
    - **Operacje biznesowe**:
        - Numery seryjne (SerialNumber) mogą być używane w systemach śledzenia lub do obsługi gwarancji.

ProductVM:

Właściwości

- **Product**:
    - Typ: Product
    - Opis: Reprezentuje dane o pojedynczym produkcie, takie jak nazwa, opis, cena, producent, kategoria, itp.
    - Używane do przechowywania i manipulacji szczegółami produktu.
- **CategoryList**:
    - Typ: IEnumerable<SelectListItem>
    - Atrybut: [ValidateNever]
        - Wyłącza walidację tej właściwości w modelu.
    - Opis: Lista kategorii w formacie odpowiednim dla użycia w elementach interfejsu użytkownika, takich jak rozwijane listy (dropdowns). Każdy element SelectListItem zawiera:
        - Text – Nazwę kategorii.
        - Value – Identyfikator kategorii.

Przeznaczenie modelu

- Model widoku ProductVM służy do obsługi scenariuszy związanych z tworzeniem i edytowaniem produktów:
    - Wyświetlanie szczegółów produktu.
    - Przypisywanie produktu do określonej kategorii.
    - Prezentacja dostępnych kategorii w rozwijanej liście.

Zastosowanie w aplikacji

- **Widoki w kontrolerach**:
    - Używany w metodach kontrolera, takich jak Upsert w kontrolerze ProductController, aby dostarczyć dane do widoku i umożliwić operacje CRUD na produktach.
- **Formularze użytkownika**:
    - CategoryList jest używana do renderowania listy kategorii w rozwijanym polu, co pozwala użytkownikowi przypisać produkt do odpowiedniej kategorii.

C)Widoki aplikacji:
Sekcja Administratora:

Create:

Przeznaczenie widoku

- Widok służy do dodawania nowej kategorii w aplikacji.
- Umożliwia użytkownikowi wprowadzenie nazwy kategorii oraz numeru określającego kolejność wyświetlania.

Interakcje użytkownika

- **Walidacja**:
    - Pola są walidowane zarówno po stronie klienta (dzięki skryptom), jak i na serwerze (ASP.NET ModelState).
    - Komunikaty błędów są wyświetlane bezpośrednio pod polami.
- **Przesyłanie danych**:
    - Po naciśnięciu "Dodaj" dane są przesyłane do akcji Create kontrolera Category.
- **Nawigacja**:
    - Użytkownik może powrócić do listy kategorii za pomocą przycisku "Wróć".

Delete:

Przeznaczenie widoku

- Widok pozwala użytkownikowi zobaczyć szczegóły kategorii, którą zamierza usunąć.
- Główne akcje to potwierdzenie usunięcia lub anulowanie operacji i powrót do listy kategorii.

Interakcje użytkownika

- **Przeglądanie szczegółów kategorii**:
    - Użytkownik widzi nazwę i kolejność wyświetlania kategorii.
    - Nie może zmieniać danych dzięki użyciu disabled.
- **Usuwanie kategorii**:
    - Kliknięcie "Usuń" przesyła formularz do kontrolera, gdzie jest obsługiwana logika usuwania.
- **Anulowanie operacji**:
    - Użytkownik może anulować operację, klikając "Wróć".

Edit:

Przeznaczenie widoku

- Widok pozwala na edytowanie istniejącej kategorii w aplikacji, umożliwiając użytkownikowi zmianę jej nazwy oraz kolejności wyświetlania.

- Przycisk "Aktualizuj" zapisuje zmiany, a "Wróć" pozwala powrócić do poprzedniej strony.

Interakcje użytkownika

- **Edytowanie kategorii**:
  - Użytkownik może zmienić nazwę i kolejność wyświetlania kategorii.
- **Aktualizowanie**:
  - Kliknięcie "Aktualizuj" przesyła zmienione dane na serwer w celu zapisania.
- **Anulowanie operacji**:
  - Użytkownik może kliknąć "Wróć", aby powrócić do widoku listy kategorii, bez wprowadzania zmian.

Index (Category):

Lista kategorii:

- Użytkownicy mogą zobaczyć wszystkie dostępne kategorie w formie tabeli. Każda kategoria zawiera numer porządkowy i nazwę.

Dodawanie kategorii:

- Użytkownicy mogą dodać nową kategorię klikając przycisk "Dodaj", który przenosi ich do formularza tworzenia nowej kategorii.

Edytowanie kategorii:

- Przycisk "Edytuj" pozwala użytkownikowi na przejście do formularza edytowania wybranej kategorii.

Usuwanie kategorii:

- Przycisk "Usuń" prowadzi użytkownika do formularza usuwania wybranej kategorii. Po potwierdzeniu kategoria zostanie usunięta.

Upsert:

Widok służy do dodawania lub edytowania produktu w aplikacji. Zawiera formularz, w którym użytkownicy mogą wprowadzać informacje o produkcie, takie jak jego nazwa, opis, numer seryjny, producent, cena, kategoria oraz zdjęcie. W zależności od tego, czy użytkownik edytuje istniejący produkt, czy dodaje nowy, widok dostosowuje przyciski oraz dane formularza.

Elementy formularza:

- **Id i ImageURL**: Ukryte pola (hidden) dla Product.Id oraz Product.ImageURL. Umożliwiają przesyłanie tych danych w przypadku edycji.
- **Nazwa produktu**:
  - Pole tekstowe z asp-for="Product.Name", które odpowiada za nazwę produktu. Zawiera również walidację.
- **Opis produktu**:

        - Pole tekstowe (textarea) dla opisu produktu. Jest obsługiwane przez bibliotekę TinyMCE do formatowania tekstu.
    - **Numer seryjny, Producent, Cena**:
        - Pola do wprowadzenia numeru seryjnego, producenta i ceny.
    - **Kategoria produktu**:
        - Dropdown (select), z którego użytkownik może wybrać kategorię produktu. Lista kategorii jest przekazywana z widoku modelu (Model.CategoryList).
    - **Obrazek produktu**:
        - Pole umożliwiające przesłanie pliku obrazu. Wybrany obrazek jest wyświetlany w formularzu.
    - **Przyciski**:
        - W zależności od tego, czy edytujesz istniejący produkt, formularz wyświetli przycisk "Aktualizuj" lub "Dodaj".

    Widok **Dodaj/Edytuj Produkt** pozwala użytkownikowi na:

    - Dodanie nowego produktu lub edytowanie istniejącego.
- Wprowadzenie szczegółowych informacji o produkcie, takich jak: nazwa, opis, numer seryjny, producent, cena, kategoria oraz zdjęcie.
- Użycie edytora TinyMCE do wprowadzania opisów.
- Wyświetlenie podglądu obrazu produktu.

Index (Product):

Widok służy do wyświetlania listy produktów, w tym ich szczegółów, takich jak nazwa, producent, numer seryjny, cena oraz kategoria. Dodatkowo umożliwia dodanie nowego produktu lub edycję istniejącego. Widok zawiera także strukturę tabeli do wyświetlania danych.

Tabela, która ma wyświetlać produkty w kilku kolumnach:

- **Nazwa**: Nazwa produktu.
- **Producent**: Producent produktu.
- **Numer seryjny**: Numer seryjny produktu.
- **Cena**: Cena produktu.
- **Kategoria**: Kategoria produktu.
- **Ostatnia kolumna**: Miejsce na ikony akcji (np. edycja, usunięcie).

Widok Lista Produktów:

- Wyświetla tabelę z produktami, przedstawiając podstawowe informacje o każdym z nich.
- Umożliwia dodawanie nowych produktów lub edytowanie istniejących przez przycisk "Dodaj".
- Zawiera dynamiczną tabelę (potencjalnie kontrolowaną przez JavaScript), która może zawierać więcej interaktywnych funkcji, takich jak edycja lub usuwanie produktów.

Sekcja użytkownika:

Details:

Widok ten przedstawia szczegóły konkretnego produktu, wyświetlając jego dane, takie jak nazwa, producent, numer seryjny, cena, kategoria, opis, oraz obrazek. Użytkownik ma również możliwość dodania produktu do koszyka (choć przycisk jest na razie wyłączony).

Index:

Widok ten jest odpowiedzialny za wyświetlanie listy produktów w formie kart. Każdy produkt jest przedstawiony z obrazkiem, nazwą, producentem, ceną oraz przyciskiem umożliwiającym przejście do szczegółów produktu.

Privacy:

Widok ten jest odpowiedzialny za wyświetlanie informacji na temat polityki prywatności.

Sekcja Identity:

To wbudowane widoki które zostały użyte do wprowadzenia rejestracji oraz logowania użytkownika

Udostępnione widoki:

_Layout:

Ten plik layoutu służy jako szablon, który definiuje strukturalne elementy strony takie jak nagłówek, stopka oraz główny obszar treści. Dzięki temu każda strona korzystająca z tego układu będzie miała jednolitą strukturę. Dodatkowo obsługuje dynamiczne wstawianie treści i skryptów, umożliwiając łatwe rozszerzanie i modyfikowanie aplikacji.

_LoginPartial:

W tym fragmencie kodu używam ASP.NET Core Identity do zarządzania procesem logowania i rejestracji użytkowników. Działa on w ramach nawigacji (część menu) w aplikacji, wyświetlając różne opcje w zależności od stanu logowania użytkownika.

_Error:

Ten kod warunkowo wyświetla różne opcje w menu nawigacyjnym w zależności od tego, czy użytkownik jest zalogowany. Jeśli tak, pokazuje opcje zarządzania kontem i wylogowania; jeśli nie, umożliwia rejestrację lub logowanie. Dzięki temu użytkownik może łatwo przejść do odpowiednich sekcji w zależności od swojego stanu zalogowania.

D)Kontekst bazy danych

Struktura tabel:

1. **Tabele**: Aplikacja tworzy trzy główne tabele w bazie danych:
    1. Categories: Przechowuje kategorie produktów, np. tłumiki, uchwyty, uszczelki.
    2. Products: Zawiera informacje o produktach, takie jak nazwa, producent, cena, numer seryjny, opis i przypisana kategoria.
    3. ApplicationUsers: Przechowuje użytkowników aplikacji (dzięki dziedziczeniu po IdentityDbContext).

Klasy modeli:

- **Category**: Model reprezentujący kategorię produktu. Posiada właściwości takie jak Id, Name, DisplayOrder.
- **Product**: Model reprezentujący produkt, z właściwościami takimi jak Id, Name, Description, Producer, ListPrice, CategoryId i inne.
- **ApplicationUser**: Model reprezentujący użytkowników aplikacji (dziedziczy po IdentityUser).

Ten kontekst bazy danych jest skonfigurowany do zarządzania produktami i kategoriami w aplikacji. Dzięki metodzie OnModelCreating oraz HasData, dane początkowe (seeding) są wstawiane do bazy przy jej tworzeniu. Aplikacja korzysta z mechanizmu ASP.NET Identity, co pozwala na zarządzanie użytkownikami. Struktura bazy danych jest odpowiednia dla aplikacji zajmującej się sprzedażą produktów, gdzie istnieje potrzeba zarządzania produktami, kategoriami i użytkownikami.

Migracje:

W folderze Migrations zapisane są pliki migracyjne.

Repozytorium:

Folder Repository zawiera:

CategoryRepository:

ApplicationDbContext:

To główny kontekst bazy danych, który jest przekazywany do repozytoriów. Obejmuje on dostęp do wszystkich tabel w bazie danych, w tym tabeli Categories.

Repozytorium:

Klasa **CategoryRepository** jest częścią wzorca repozytorium, który umożliwia łatwiejszą organizację i testowanie kodu, a także abstrakcję od szczegółów bazy danych.

Aktualizacja:

Metoda **Update** aktualizuje kategorię w bazie danych, a zapisywanie zmian jest wykonywane później, gdy wywołana zostanie metoda **Save()**.

ProductRepository:

ProductRepository to repozytorium odpowiadające za operacje na tabeli produktów w bazie danych. Klasa ta implementuje metodę **Update**, która aktualizuje dane produktu w bazie na podstawie nowego obiektu produktu. Repozytorium to dziedziczy po ogólnym repozytorium **Repository<Product>**, co daje mu dostęp do podstawowych operacji CRUD, takich jak dodawanie, usuwanie czy pobieranie produktów.

Repository:

Repository<T> to ogólna klasa repozytorium, która realizuje podstawowe operacje na encjach w bazie danych. Dzięki użyciu **DbSet<T>** wykonuje operacje na tabelach, które odpowiadają klasom **T** w kontekście **ApplicationDbContext**. Klasa ta jest parametryzowana, więc może obsługiwać różne typy encji w aplikacji.

- **Metody CRUD**: Metody takie jak **Add**, **Remove**, **Get**, **GetAll** oferują wszystkie podstawowe operacje na obiektach w bazie danych.
- **Właściwości nawigacyjne**: Możliwość dołączania relacji (np. Include) do zapytań SQL.
- **Ogólność i elastyczność**: Klasa jest zaprojektowana do pracy z dowolnym typem klasy, który jest klasą encji (np. Product, Category, Order).

UnitOfWork:

Wzorzec **Unit of Work** w tym przypadku pomaga zarządzać transakcjami i operacjami na repozytoriach w jednym miejscu. Klasa **UnitOfWork** koordynuje dostęp do repozytoriów dla różnych encji (takich jak **Category** i **Product**) i zapewnia, że wszystkie zmiany są zapisywane w bazie danych za pomocą jednej metody **Save()**. Ten wzorzec ułatwia zarządzanie danymi, poprawia organizację kodu i zapewnia większą spójność transakcji w aplikacji.

Zalety i zastosowanie wzorca Unit of Work:

- **Transakcje**: Wzorzec **Unit of Work** pomaga zarządzać operacjami w ramach jednej transakcji. Dzięki temu, gdy w jednym miejscu kodu wykonujesz wiele operacji na różnych repozytoriach (np. na produktach i kategoriach), możesz upewnić się, że zmiany zostaną zapisane w bazie danych jednocześnie.
- **Optymalizacja**: Dzięki zarządzaniu transakcjami w jednym miejscu, minimalizujesz liczbę połączeń z bazą danych, co może poprawić wydajność aplikacji.
- **Spójność**: Wzorzec ten zapewnia, że operacje wykonane na różnych repozytoriach nie będą miały miejsca w różnych kontekstach transakcyjnych, co oznacza, że nie dojdzie do niespójności danych.

Jak działa w aplikacji:

- **UnitOfWork** umożliwia dostęp do różnych repozytoriów za pomocą właściwości **Category** i **Product**.
- Kiedy użytkownik wykonuje operacje na kategoriach lub produktach (np. dodaje nowy produkt), te operacje są wykonywane za pomocą odpowiednich repozytoriów, a wszystkie zmiany są zapisane w bazie danych po wywołaniu **Save()**.
- Zamiast wywoływać **SaveChanges()** w każdym repozytorium osobno, wywołanie metody **Save()** na jednostce pracy zapewnia, że wszystkie zmiany są zapisane razem, co zwiększa spójność aplikacji.

Interfejs repozytorium:

Zawiera:

Irepository:

Interfejs **IRepository<T>** jest ogólnym repozytorium, które definiuje podstawowe operacje na encjach, takich jak dodawanie, usuwanie, pobieranie oraz pobieranie wszystkich obiektów z bazy danych. Interfejs ten jest zaprojektowany w sposób, który umożliwia łatwe rozszerzenie o różne typy encji, na przykład **Category**, **Product**, czy inne modele, które można zmapować do tabel w bazie danych.

Opis metod interfejsu:

GetAll(string? includeProperties = null):

- **Opis**: Zwraca wszystkie rekordy (instancje klasy **T**) z bazy danych.
- **Parametry**:
  - **includeProperties** (opcjonalny): Łańcuch zawierający nazwę właściwości nawigacyjnej, która ma zostać dołączona do wyników zapytania (np. Category w przypadku produktu, jeśli produkt ma powiązaną kategorię). Wartość null oznacza brak dołączonych właściwości.
- **Zwracany wynik**: Kolekcja obiektów typu **T**.

Get(Expression<Func<T, bool>> filter, string? includeProperties = null):

- **Opis**: Zwraca pojedynczy obiekt typu **T**, który spełnia określony warunek (filtr).
- **Parametry**:
  - **filter**: Wyrażenie lambda (**Expression<Func<T, bool>>**) służące do filtrowania danych (np. x => x.Id == 1).
  - **includeProperties** (opcjonalny): Jak w poprzedniej metodzie, umożliwia dołączenie powiązanych właściwości.
- **Zwracany wynik**: Pojedynczy obiekt typu **T**, który spełnia warunki filtru. Jeśli żaden obiekt nie pasuje, zwracane jest null.

Add(T entity):

- **Opis**: Dodaje nowy obiekt typu **T** do repozytorium (czyli do bazy danych).
- **Parametry**:
  - **entity**: Obiekt, który ma zostać dodany do repozytorium.
- **Zwracany wynik**: Brak (operacja wykonywana na bazie danych, obiekt zostaje dodany).

Remove(T entity):

- **Opis**: Usuwa określony obiekt typu **T** z repozytorium (czyli z bazy danych).
- **Parametry**:
  - **entity**: Obiekt, który ma zostać usunięty z repozytorium.
- **Zwracany wynik**: Brak (obiekt jest usuwany z bazy danych).

RemoveRange(IEnumerable<T> entity):

- **Opis**: Usuwa wiele obiektów typu **T** z repozytorium (czyli z bazy danych).
- **Parametry**:
  - **entity**: Kolekcja obiektów, które mają zostać usunięte.
- **Zwracany wynik**: Brak (obiekty są usuwane z bazy danych).

IunitOfWork:

Interfejs **IUnitOfWork** stanowi wzorzec projektowy, który pomaga w zarządzaniu transakcjami w aplikacji. Zamiast wykonywać operacje na poszczególnych repozytoriach osobno, **Unit of Work** pozwala na zarządzanie wszystkimi operacjami w ramach jednej jednostki roboczej (transakcji). W kontekście tego interfejsu, jest on

odpowiedzialny za zarządzanie repozytoriami, takimi jak Category i Product, i zapewnia mechanizm zapisywania zmian do bazy danych.

Opis metod i właściwości interfejsu:

ICategoryRepository Category { get; }:

- **Opis**: Właściwość, która zapewnia dostęp do repozytorium kategorii. Dzięki temu możemy wykonywać operacje na danych kategorii w aplikacji.
- **Zwracany wynik**: Obiekt typu ICategoryRepository, który zawiera metody do operacji na tabeli kategorii w bazie danych.

IProductRepository Product { get; }:

- **Opis**: Właściwość, która zapewnia dostęp do repozytorium produktów. Dzięki temu możemy wykonywać operacje na danych produktów w aplikacji.
- **Zwracany wynik**: Obiekt typu IProductRepository, który zawiera metody do operacji na tabeli produktów w bazie danych.

void Save():

- **Opis**: Metoda, która zapisuje wszystkie zmiany dokonane w repozytoriach (np. dodanie, usunięcie, aktualizacja danych) do bazy danych w ramach jednej transakcji.
- **Zwracany wynik**: Brak. Po wywołaniu tej metody wszystkie zmiany w repozytoriach zostaną zapisane do bazy danych.

IcategoryRepository oraz IProductRepository:

IRepository<T> to ogólny interfejs repozytorium, który dostarcza podstawowe operacje na encjach, takie jak:

- **Add(T entity)**: Dodaje nowy obiekt do repozytorium.
- **Get(Expression<Func<T, bool>> filter, string? includeProperties = null)**: Pobiera obiekt na podstawie podanego filtru.
- **GetAll(string? includeProperties = null)**: Pobiera wszystkie obiekty z repozytorium.
- **Remove(T entity)**: Usuwa obiekt z repozytorium.
- **RemoveRange(IEnumerable<T> entity)**: Usuwa wiele obiektów z repozytorium.

Dzięki dziedziczeniu po tym interfejsie, **ICategoryRepository/** IProductRepository zyskuje te wszystkie operacje, które są ogólne dla każdej encji, a dodatkowo może definiować metody specyficzne tylko dla **Category/Product**, takie jak **Update**.

E)Statyczne zasoby: CSS – bootstrap, JS.

System użytkowników:

-Role:

-Administrator: zarządzanie produktami i kategoriami

-Klient: przeglądanie produktów, edycja profilu

-Uprawnienia:

-Gość: przeglądanie produktów, możliwość rejestracji

-Zalogowany użytkownik: przeglądanie produktów, edycja profilu

Charakterystyka ciekawych funkcjonalności:

- możliwość dodania zdjęć do produktów
- w sekcji dodawania produktów zaimplementowany został edytor tekstu tinyMCE który pozwala formatować tekst przy wprowadzaniu opisu produktu
- w profilu użytkownika można pobrać dane konta które zostały wprowadzone