Analiza kodu ProductsController

1. Wprowadzenie

ProductsController to kontroler API w ASP.NET Core, który implementuje operacje CRUD (Create, Read, Update, Delete) dla produktów. Wykorzystuje Entity Framework Core do komunikacji z bazą danych i korzysta z mechanizmu Dependency Injection.

2. Importowanie przestrzeni nazw

Na początku pliku importowane są niezbędne przestrzenie nazw:

```
using Core.Entities;

using Infrastructure.Data;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;
```

- Core.Entities – zawiera definicję encji Product.

- Infrastructure.Data – zawiera StoreContext, czyli klasę kontekstu bazy danych.

- Microsoft.AspNetCore.Mvc – dostarcza funkcjonalności do obsługi API w ASP.NET Core.

- Microsoft.EntityFrameworkCore – umożliwia operacje na bazie danych przy użyciu Entity Framework Core.

3. Definicja kontrolera

```
[ApiController]

[Route("api/[controller]")]

public class ProductsController : ControllerBase
```

- [ApiController] – oznacza klasę jako kontroler API, co włącza dodatkowe funkcje, takie jak automatyczna walidacja modeli.

- [Route("api/[controller]")] – definiuje ścieżkę URL dla tego kontrolera jako api/products.

- ControllerBase – oznacza, że jest to kontroler API (nie obsługuje widoków, w przeciwieństwie do Controller).

4. Wstrzykiwanie kontekstu bazy danych

```
private readonly StoreContext context;

public ProductsController(StoreContext context)

{

   this.context = context;

}
```

- Dependency Injection pozwala na przekazanie StoreContext do kontrolera.

- Dzięki temu kontroler może komunikować się z bazą danych bez konieczności jej ręcznego inicjalizowania.

5. Operacje CRUD

5.1 Pobranie wszystkich produktów

```
[HttpGet]

public async Task<ActionResult<IEnumerable<Product>>> GetProducts()

{

   return await context.Products.ToListAsync();

}
```

- Metoda zwraca listę wszystkich produktów w bazie danych w sposób asynchroniczny.

- [HttpGet] oznacza, że metoda odpowiada na żądania HTTP GET.

- ToListAsync() pobiera dane z bazy w sposób asynchroniczny, poprawiając wydajność aplikacji.

5.2 Pobranie pojedynczego produktu

```
[HttpGet("{id:int}")]
public async Task<ActionResult<Product>> GetProduct(int id)
{
   var product = await context.Products.FindAsync(id);
   if (product == null) return NotFound();
   return product;
}
```

- Pobiera produkt na podstawie jego id.
- Jeśli produkt nie istnieje, zwracany jest kod błędu 404 (Not Found).

6. Podsumowanie

- ProductsController implementuje pełny CRUD dla produktów w bazie danych.
- Wykorzystuje Entity Framework Core do interakcji z bazą.
- Używa Dependency Injection do uzyskania dostępu do kontekstu bazy danych (StoreContext).
- Wszystkie metody działają asynchronicznie, co zwiększa wydajność aplikacji.
- Walidacja odbywa się poprzez sprawdzanie, czy produkt istnieje, a także przez atrybut [ApiController], który automatycznie obsługuje błędy modelu.

Kontroler ten jest solidnym fundamentem do obsługi operacji CRUD i może być łatwo rozszerzony o dodatkowe funkcjonalności, takie jak filtrowanie, paginacja czy obsługa relacji między encjami.