**Analiza kodu ProductConfiguration.cs**

1. Wprowadzenie

Plik `ProductConfiguration.cs` definiuje konfigurację encji `Product` w bazie danych przy użyciu \*\*Entity Framework Core\*\*. Umożliwia dostosowanie właściwości encji do wymagań bazy danych, np. poprzez określenie typu danych czy oznaczenie wymaganych pól.

2. Importowanie przestrzeni nazw

using Core.Entities;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

- \*\*Core.Entities\*\* – zawiera definicję encji `Product`.

- \*\*Microsoft.EntityFrameworkCore\*\* – zapewnia funkcjonalność ORM (Object-Relational Mapping) dla pracy z bazą danych.

- \*\*Microsoft.EntityFrameworkCore.Metadata.Builders\*\* – umożliwia konfigurację encji poprzez `EntityTypeBuilder`.

3. Definicja klasy ProductConfiguration

namespace Infrastructure.Config;

- \*\*namespace Infrastructure.Config\*\* – przestrzeń nazw, w której przechowywana jest konfiguracja encji.

public class ProductConfiguration : IEntityTypeConfiguration<Product>

- \*\*ProductConfiguration\*\* – klasa implementuje `IEntityTypeConfiguration<Product>`, co oznacza, że definiuje sposób mapowania encji `Product` do tabeli bazy danych.

4. Metoda Configure

public void Configure(EntityTypeBuilder<Product> builder)

- \*\*Configure(EntityTypeBuilder<Product> builder)\*\* – metoda konfiguruje encję `Product`, określając szczegóły mapowania pól w bazie danych.

Konfiguracja właściwości:

builder.Property(x => x.Price).HasColumnType("decimal(18,2)");

- \*\*Property(x => x.Price)\*\* – definiuje konfigurację dla pola `Price`.

- \*\*HasColumnType("decimal(18,2)")\*\* – określa typ kolumny jako `decimal` z precyzją `18` i `2` miejscami po przecinku.

builder.Property(x => x.Name).IsRequired();

- \*\*Property(x => x.Name)\*\* – określa konfigurację dla pola `Name`.

- \*\*IsRequired()\*\* – oznacza, że wartość tej właściwości musi być zawsze ustawiona (nie może być `null`).

5. Zastosowanie klasy ProductConfiguration

Klasa `ProductConfiguration` jest używana w kontekście bazy danych (`StoreContext`) w celu automatycznego konfigurowania encji `Product`. W `StoreContext` należy dodać:

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.ApplyConfiguration(new ProductConfiguration());

base.OnModelCreating(modelBuilder);

}

- \*\*ApplyConfiguration(new ProductConfiguration())\*\* – rejestruje konfigurację `ProductConfiguration` w `StoreContext`.

- \*\*OnModelCreating\*\* – metoda w `DbContext`, w której definiowane są konfiguracje encji.

6. Podsumowanie

- `ProductConfiguration` definiuje konfigurację encji `Product`.

- Określa typ kolumny `Price` jako `decimal(18,2)`.

- Wymusza, aby pole `Name` było wymagane.

- Stosuje się ją w `DbContext`, aby zapewnić jednolitą konfigurację bazy danych.

Jest to dobre podejście do zarządzania mapowaniem encji na bazę danych w \*\*Entity Framework Core\*\*, zapewniające większą kontrolę nad schematem tabeli.