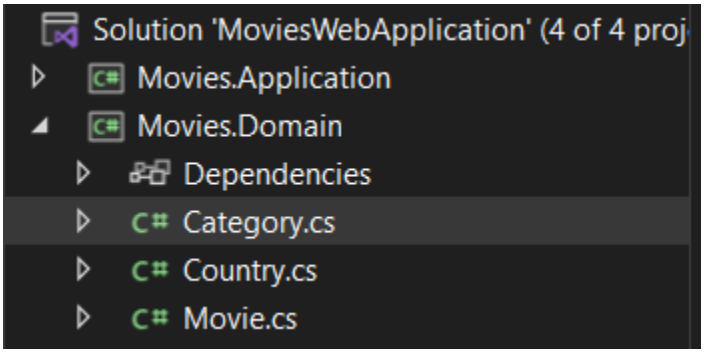
Tworzenie tabel i tabel pośredniczących

Modele (na ich podstawie są tworzone tabele)



Tabele pośredniczące tworzą się automatycznie na podstawie podanych powiązań z innymi tabelami (w tym wypadku są to kolekcje bo mamy relacje wiele:wielu)

```
28 references
public class Movie
{
    [Key]
    4 references
    public Guid MovieId { get; set; }
    4 references
    public string Title { get; set; }
    4 references
    public DateTime ReleaseDate { get; set; }
    4 references
    public string PosterUrl { get; set; }

    //Kolekcje potrzebne do wygenerowania kluczy obcych
    7 references
    public ICollection<Country> Countries { get; set; }
    7 references
    public ICollection<Category> Categories { get; set; }
}
```

Tak wygląda taka tabela która sama się utworzy (zawiera oczywiście odpowiednie ID filmu i gatunku):

	CategoriesCategoryId	MoviesMovieId
	Filter	Filter
1	14141537-1064-4B66-872D-2EF4211...	66076B71-1160-457F-...
2	14141537-1064-4B66-872D-2EF4211...	BFDDC19D-4957-4EEC-...
3	A8FDFD34-05C3-40E3-82FD-695D58...	66076B71-1160-457F-...
4	A8FDFD34-05C3-40E3-82FD-695D58...	B63A817F-7B4D-4EC9-970A-1434E5...
5	D46D4600-36BA-467A-965B-...	B63A817F-7B4D-4EC9-970A-1434E5...
6	D46D4600-36BA-467A-965B-...	BFDDC19D-4957-4EEC-...
7	14141537-1064-4B66-872D-2EF4211...	30DA21CF-...
8	A8FDFD34-05C3-40E3-82FD-695D58...	30DA21CF-...

Tutaj jest także wyjaśnione jak robić relacje innych typów:

https://www.youtube.com/watch?v=9sXXfq0GDYI&ab_channel=ISeeSharp

Migracja

W przypadku dodania nowych tabeli należy dokonać migracji bazy tymi dwoma poleceniami:

dotnet ef migrations add InitialCreate --startup-project MoviesWebApplication --project Movies.Infrastructure

dotnet ef database update --startup-project MoviesWebApplication --project Movies.Infrastructure

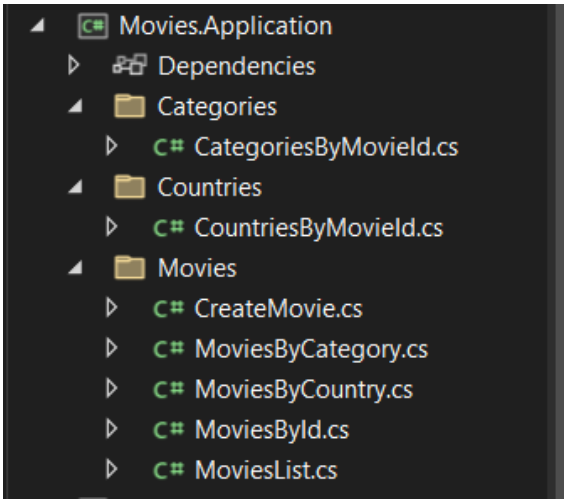
Dodawanie testowych danych

W klasie Seed można dodawać nowe wpisy do tabeli, przed odpaleniem polecam odkomentować czyszczenie tabel żeby stare dane się usunęły.

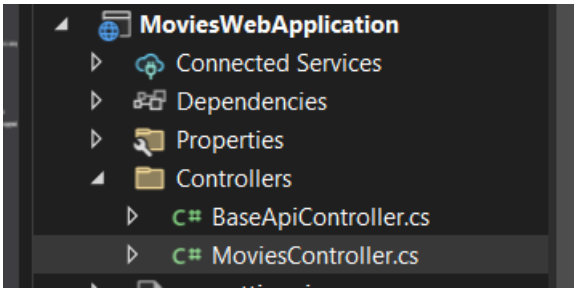
```
public class Seed
{
    1 reference
    public static async Task SeedData(DataContext context)
    {
        //Usuwanie danych w tabelach
        /*context.Movies.RemoveRange(context.Movies);
        context.Countries.RemoveRange(context.Countries);
        context.Categories.RemoveRange(context.Categories);
        await context.SaveChangesAsync();*/


        // jeśli baza ma jakieś rekordy to nic nie rób
        if (context.Movies.Any()) return;

        var usa = new Country
        {
            CountryName = "USA"
        };

        var action = new Category
        {
            CategoryName = "Akcji"
        };
```

Dodawanie endpointów

Jako iż został wdrożony mediator to jest to trochę rozdzielone. Tutaj na dole tworzę dla każdej tabeli osobny folder i klase pod odpowiedni endpoint (aktualnie wszystko to w sumie gety, jest tylko jeden put dla testów). Można to dodawać w dowolny sposób ale fajnie jakby była zachowana konwencja typu nazwa folderu to zwracany obiekt (bądź lista obiektów) tego typu czyli w Movies pakujemy wszystko co zwraca obiekty filmów, a w countries wszystko co zwraca obiekty krajów. Podobnie z dodawaniem usuwaniem itd.



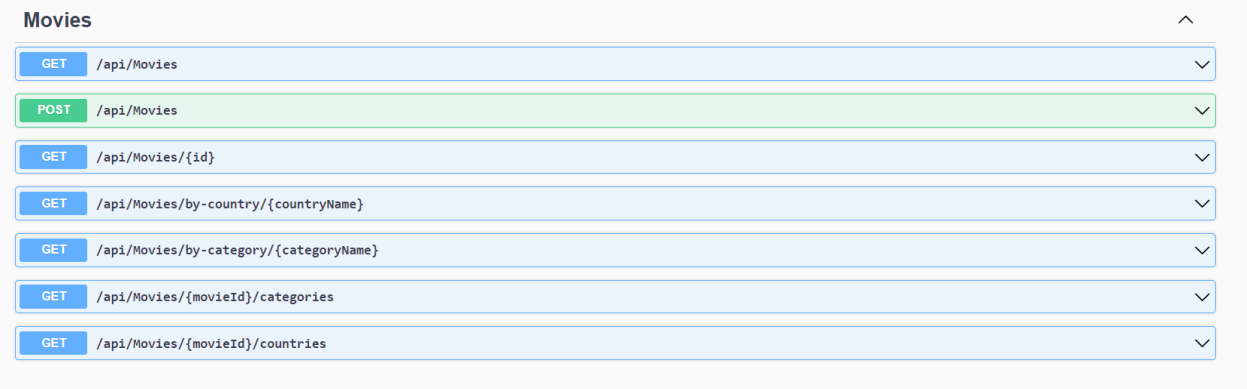
Po dodaniu tam klasy która obsłuży polecenie należy to jeszcze podpiąć w tym miejscu:



W nawiasie podaje się trasę (np. „by-country/{countryName}”), której potem będzie używal react do komunikacji z API (w sumie to można dodawać dowolne nazwy, byle było wiadomo o co chodzi).

```
//Zwracanie filmów na podstawie nazwy kraju
[HttpGet("by-country/{countryName}")]
0 references
public async Task<ActionResult<List<Movie>>> GetMoviesByCountry(string countryName)
{
    var query = new MoviesByCountry.Query { CountryName = countryName };
    var movies = await Mediator.Send(query);
    if (movies == null || !movies.Any())
    {
        return NotFound($"Nie znaleziono filmów dla podanego kraju '{countryName}'.");
    }
    return Ok(movies);
}
//Zwracanie filmów na podstawie kategorii
[HttpGet("by-category/{categoryName}")]
0 references
public async Task<ActionResult<List<Movie>>> GetMoviesByCategory(string categoryName)
{
    var query = new MoviesByCategory.Query { CategoryName = categoryName };
```

W swagerze jest to też wyświetlane więc raczej nie będzie problemu z zlokalizowaniem tego.



Dodatkowe informacje

Jako iż są relacje pomiędzy tabelami to można zwracać rozszerzoną wersje zapytania (zrobiłem cos takiego dla zapytania o zwrócenie filmu o podanym ID). Trzeba wtedy dodać Include przy zapytaniu, ale to raczej nie będziemy używać bo lepiej zwrócić 3 łatwe zapytania jak 1 skomplikowane. (NA RAZIE JEST TO USUNIĘTE ALE MOŻNA SOBIE SPRAWDZIC JAK TO WYGLADA DODAJAC TE DWIE LINIJKI).

```
1 reference
public class Handler : IRequestHandler<Query, Movie>
{
    private readonly DataContext _context;
    0 references
    public Handler(DataContext context)
    {
        _context = context;
    }

    //Dla testów zwracam również z informacjami o gatunkach i krajach produkcji
    0 references
    public async Task<Movie> Handle(Query request, CancellationToken cancellationToken)
    {
        // Pobierz film z bazy na podstawie ID
        return await _context.Movies
            .Include(m => m.Countries)
            .Include(m => m.Categories)
            .FirstOrDefaultAsync(m => m.MovieId == request.Id);
    }
}
```

Dodawanie jest zaimplementowane w postaci gdzie będzie trzeba podać id gatunków i id krajów (można wiele po przecinku). Przy okazji jest zaimplementowane sprawdzanie czy wgl podane ID istnieją.