**API Rest com .NET Core 5: Operações essenciais com verbos HTTP**

1° criar um projeto **’API Web do ASP.NET Core’** com .NET Core 5.0;

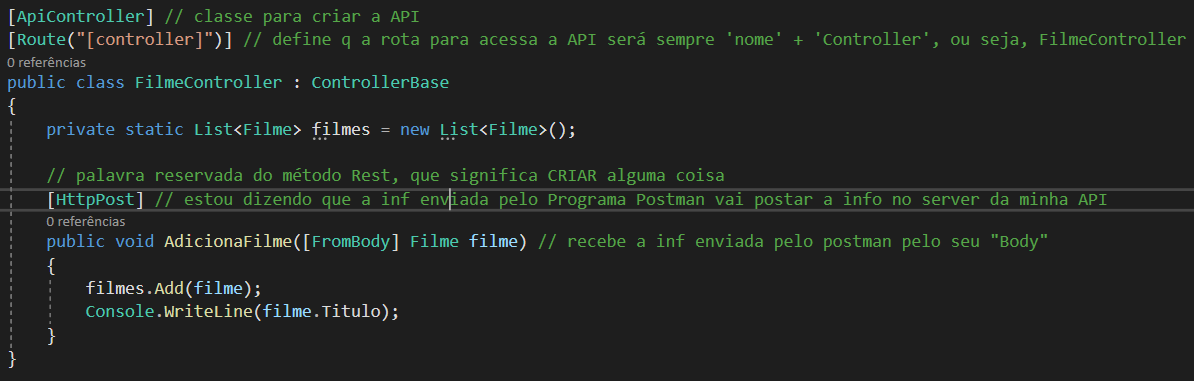
Baixar **Postman;** baixar MySQL

Vamos criar uma API que vai fornecer nomes de filmes a quem realizar a requisição

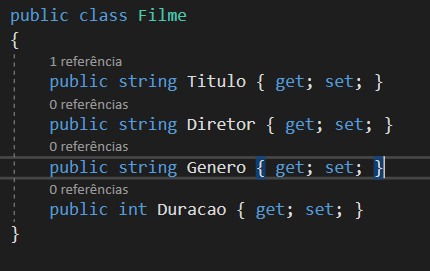
Quando enviamos requisições para a API a fim de criarmos um novo recurso, é padrão explicitarmos um verbo HTTP para tal ação.

Nosso primeiro método (**AdicionaFilme**) vai postar/enviar (**HttpPost**) informações via Postman para o servidor da nossa API (localhost)

1° Criar o controller com suas palavras reservadas

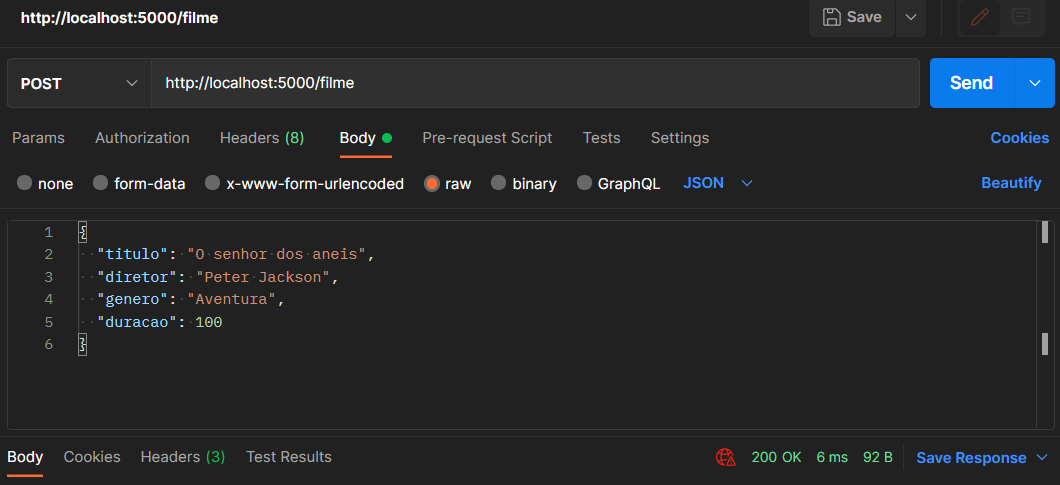
Nosso método adiciona filmes na lista static do controller e mostra no console o título do envio HttpPost feito pelo Postman

2° Criar a class Filme com suas propriedades

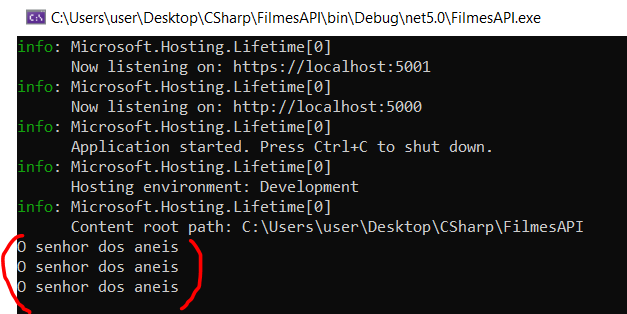


3° configurar o Postman para enviar os dados sobre filmes via **JSON** e pelo seu **BODY**

4° ajustar o caminho de envio dos dados para **http://localhost:5000/filme**

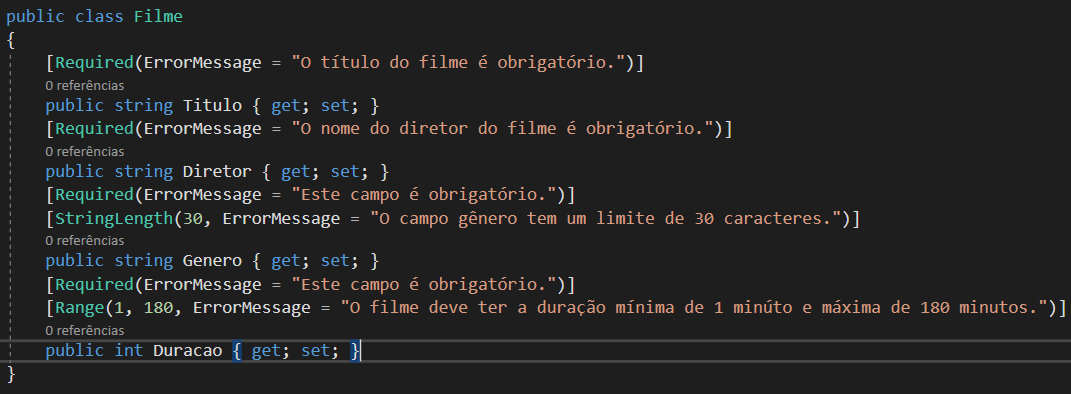


Resultado de 3 envios:



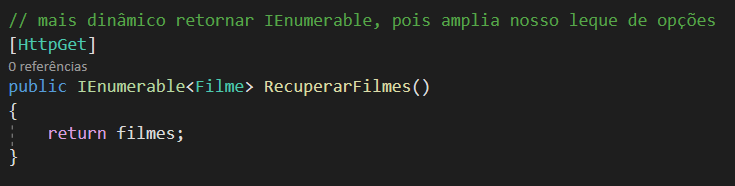
**VALIDANDO INFORMAÇÕES ENVIADAS VIA HTTPPOST PELO POSTMAN**

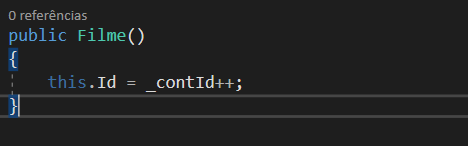
Os campos são autoexplicativos. As ErrorMessage aparecem no próprio console do Postman

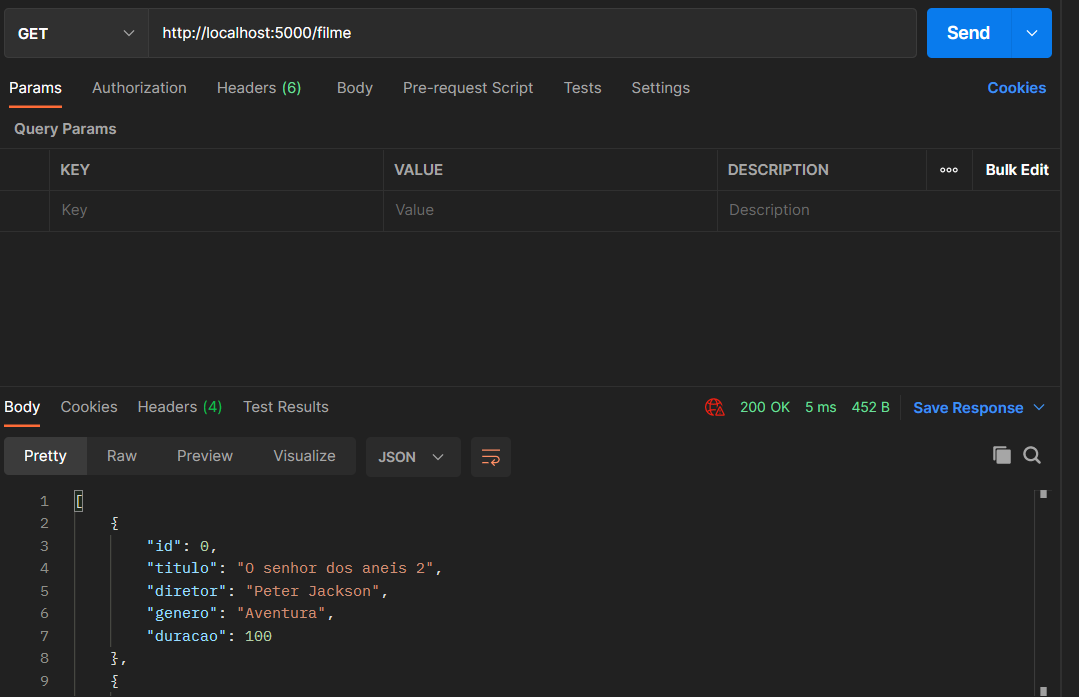


**RETORNANDO TODAS AS INFORMAÇÕES DA API**

1° Criar o método **HttpGet** para capturar os dados da nossa API via Postman

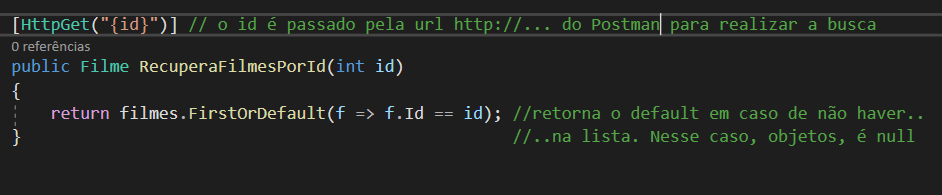


2° Vamos criar um ID manual para os filmes, por enquanto. O fiz na classe Filme

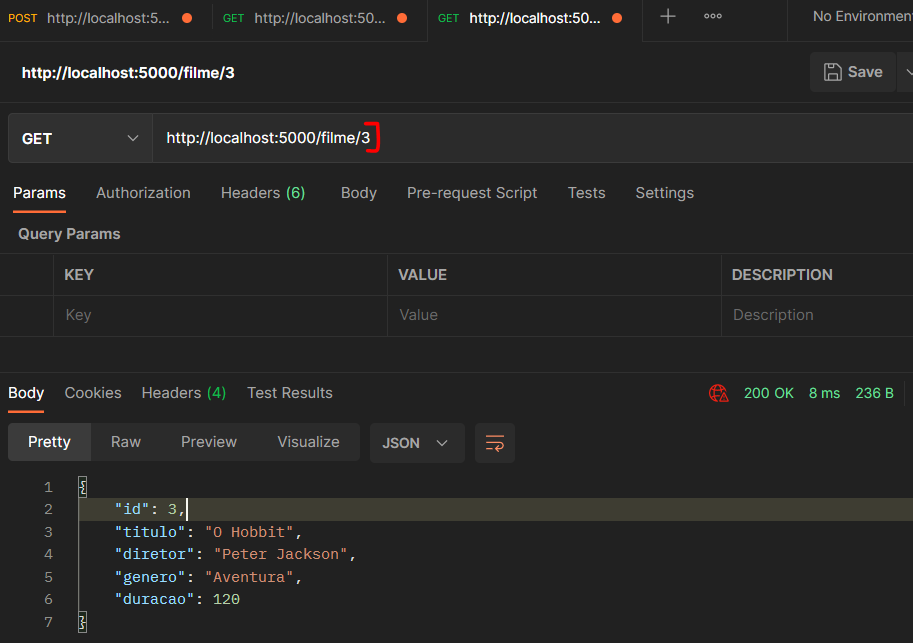
3° criar a requisição GET no Postman. Basta adicionar nova guia e colar o http://...

**RETORNANDO INFORMAÇÕES DA API COM BASE EM ‘Id’ DO FILME**

1° Vamos criar o método que recebe o Id e realizar a busca.

Note que ele recebe o parâmetro **“{id}” em HttpGet.** 

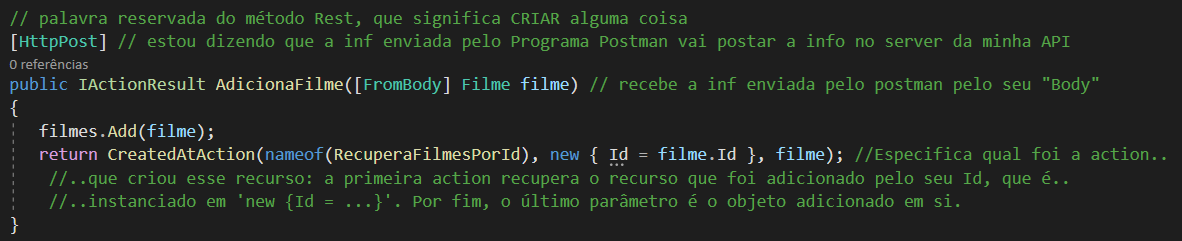
2° Adicionar nova aba GET no Postman e testar a URL com o Id a ser buscado.



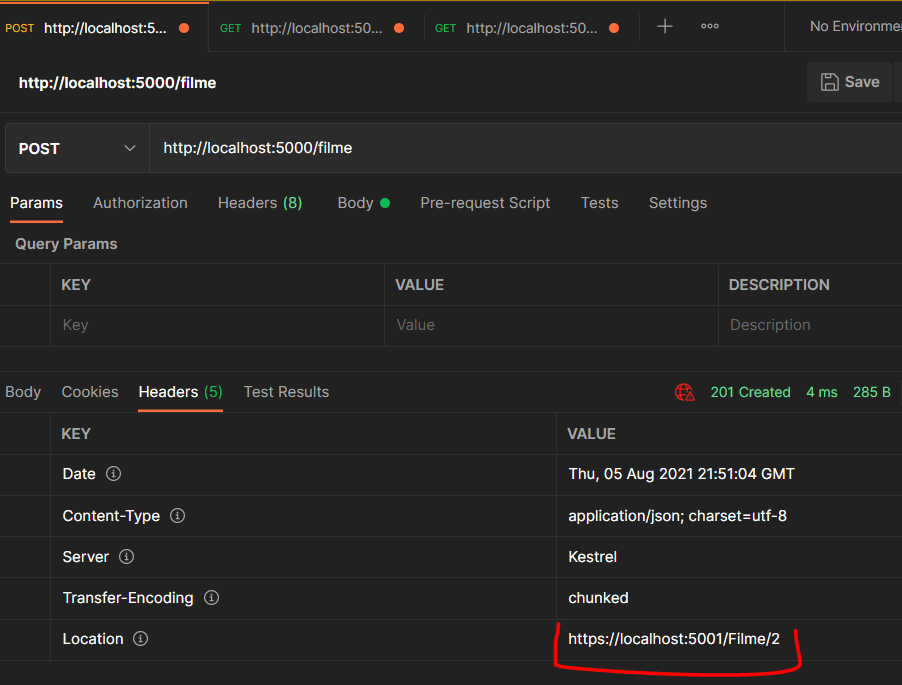
**DEFININDO RETORNO DAS ROTAS/REQUISIÇÕES COM BASE NO PADRÃO REST**

Vamos modificar os returns dos métodos, para que retornem códigos (404, 200, 201, etc) em todas as requisições, além de informações mais sólidas.

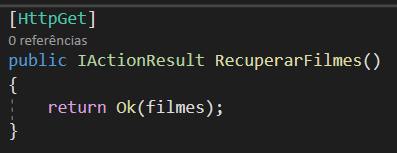
1° Modificar a Action/Método ‘AdicionaFilme’:

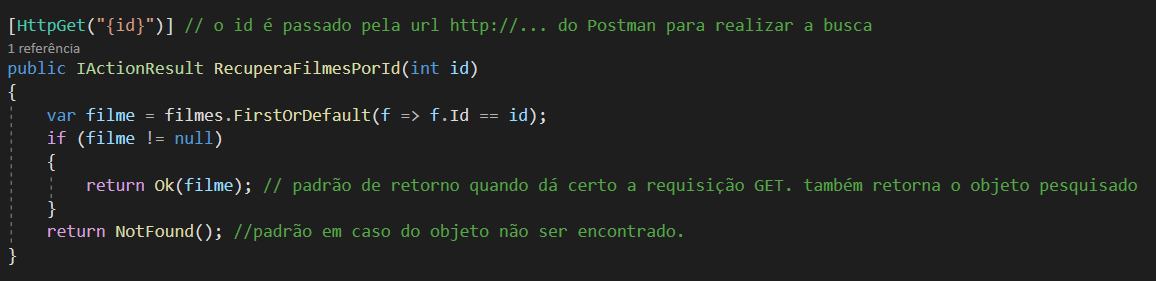
Note que todos os retornos foram mudados para **IActionResult**, por causa dos ‘returns’ específicos da REST API, que exigem isso.

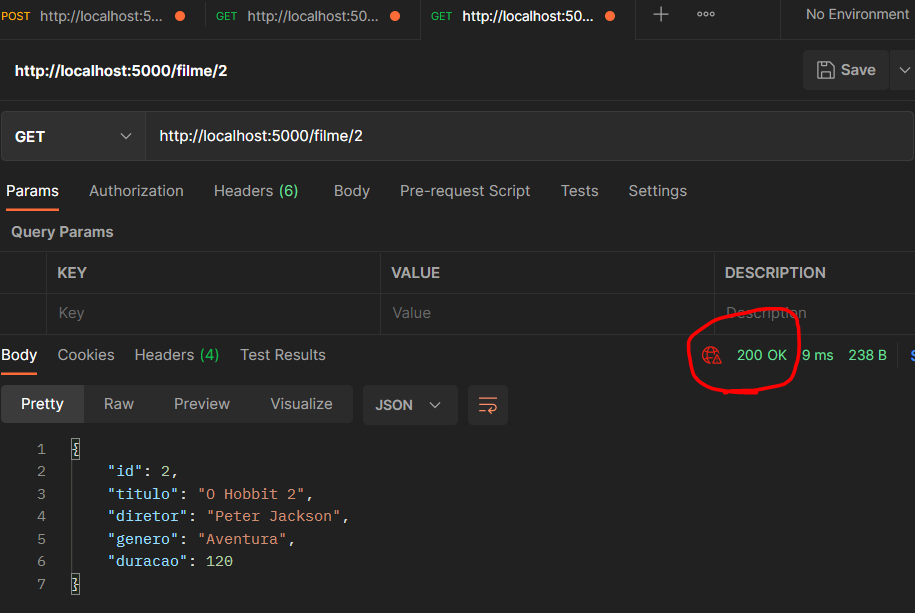
Agora é possível ver o caminho de criação da requisição **POST:**

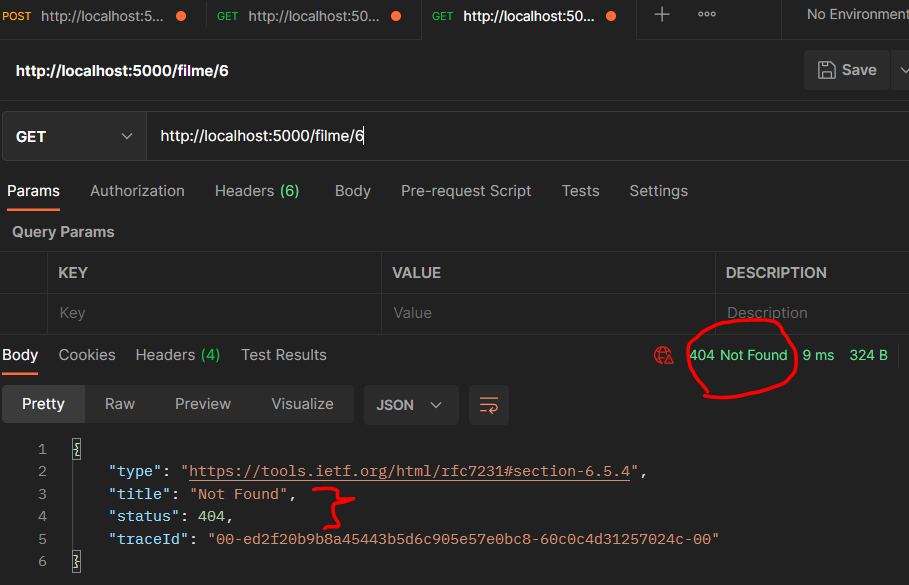


2° Modificar a Action ‘RecuperarFilmes’:



3° Modificar a Action ‘RecuperarFilmePorId’:

Observe os ‘returns’ do método em ação no Postman, em caso de requisição ok e notFound: 



**CONECTANDO A API AO BD MySQL**

Primeiro é necessário baixar os frameworks necessários para rodar a app:

Em ferramentas > gerenciador do pac. Nuget > Gerenciar pacotes do Nuget

Em Procurar: **Microsoft.EntityFrameworkCore** (obrigatório)

**Microsoft.EntityFrameworkCore.Tools** (obrigatório)

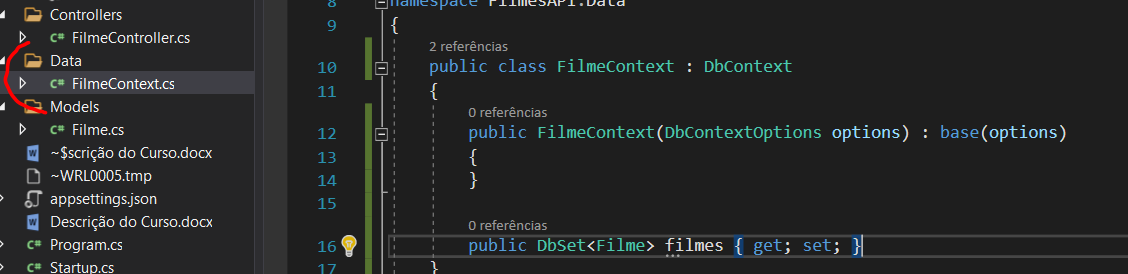
**MySql.EntityFrameworkCore** (para permitir conexão com o MySQL)

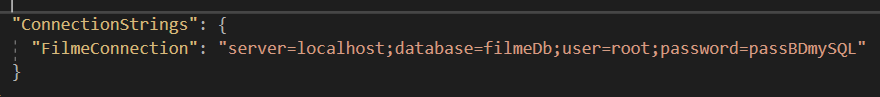
Instalar um por um. No caso de SQL, seria necessário instalar o **Microsoft.EntityFrameworkCore.SqlServer**

Vamos conectar agora:

1° Criar uma pasta no projeto (nome pode ser Data), criar a class **Context,** a qual vai estabelecer a conexão com **DbContext**

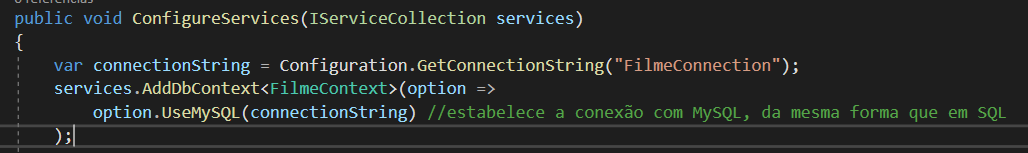
Construtor obrigatório. Basta usar Ctrl + . na classe e gerar automaticamente.



2° Criar a **StringConnection** em **appsettings.json:**

O atributo que recebe a string de conexão pode ter o nome que quisermos. O padrão e o mais simples é sempre: **local do BD; Nome do BD;** **Nome do USER; Senha do BD.**

3° Criar o Service que vai realizar a conexão na classe **Startup.cs** e realizar a conexão via código:

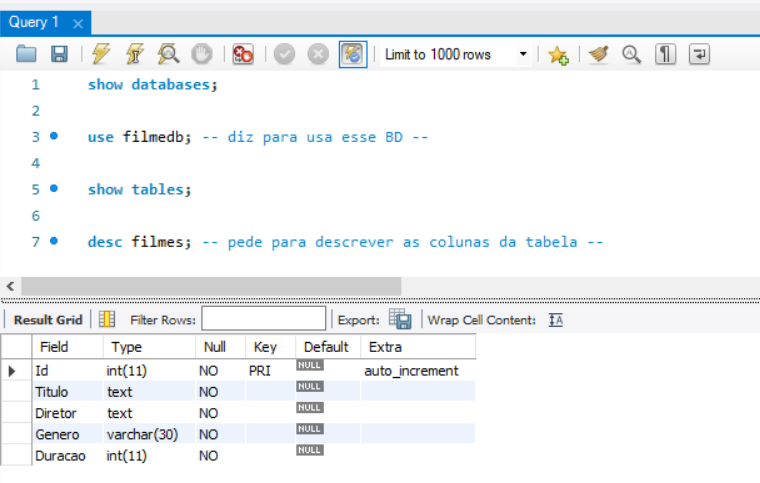
com SQL server seria “**UseSqlServer(ConnectionString)**”

**GERANDO A 1ª MIGRATION**

1° No console do gerenciador de pacotes NuGet: **Add-Migration ‘DarNomeAEla’** >

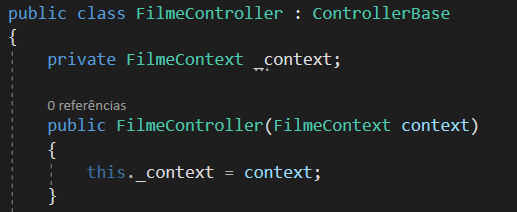
checar se está tudo correto > **Update-Database**

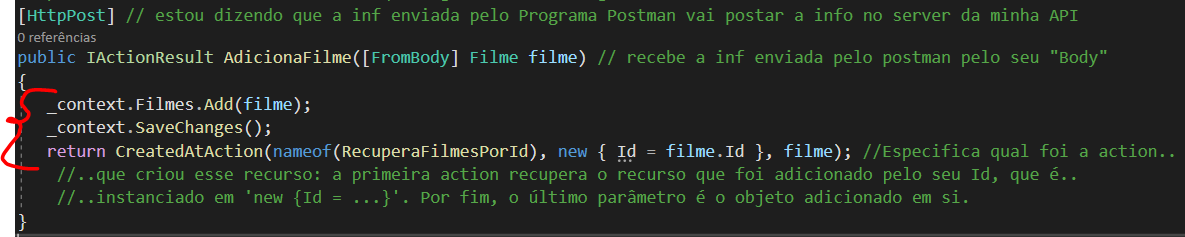
2° Entrar no MySQL e checar se o BD e suas tabelas foram criadas:

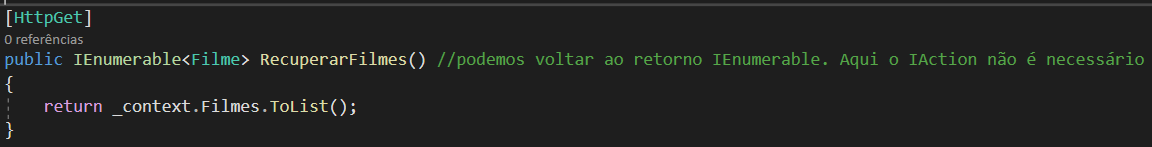


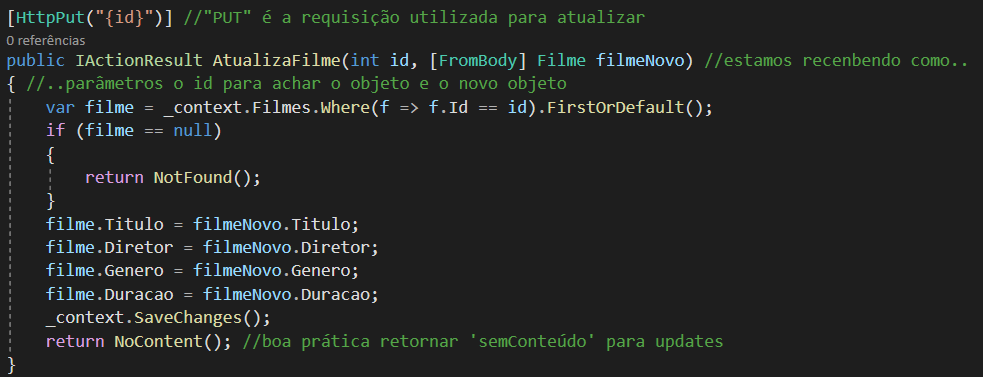
**ESCREVENDO E LENDO DADOS NO BD**

1° Vamos acessar a tabela de filmes pela instanciação da classe **FilmeContext** em **FilmeController**



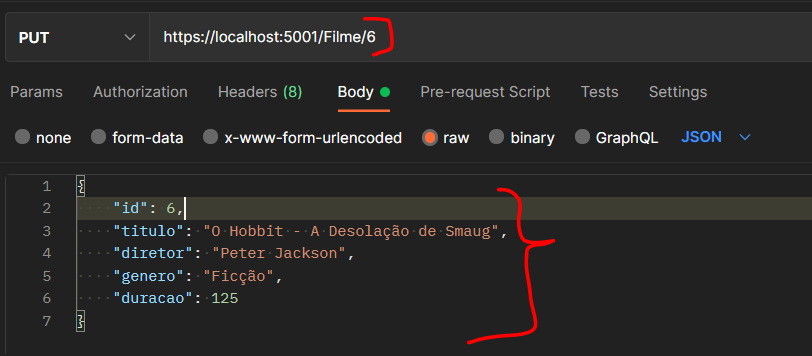
2° Agora devemos realizar as substituições da **List<Filme>** que estava sendo usada nos métodos/actions pelo campo de acesso ao BD.

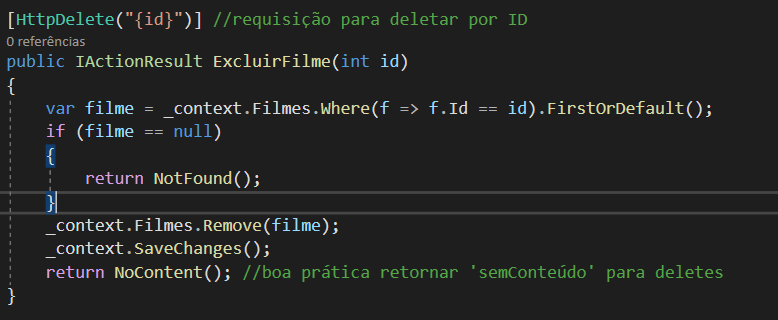


**ATUALIZANDO E DELETANDO DADOS NO BD**

Veja no Postman como a requisição deve ser feita:

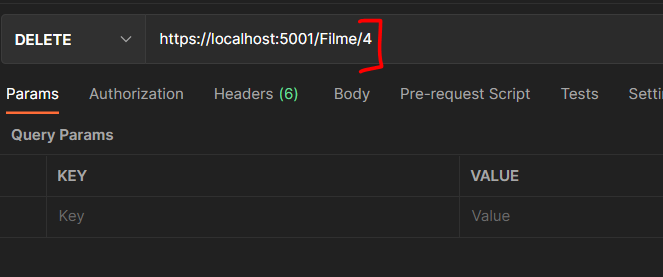
Colocamos o ID do objeto a ser mudado na URL e os novos dados.





Veja como enviar a requisição DELETE pelo Postman:

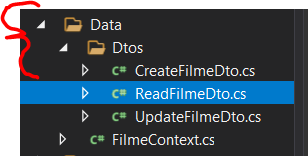
Colocar o ID do objeto a ser deletado na URL.



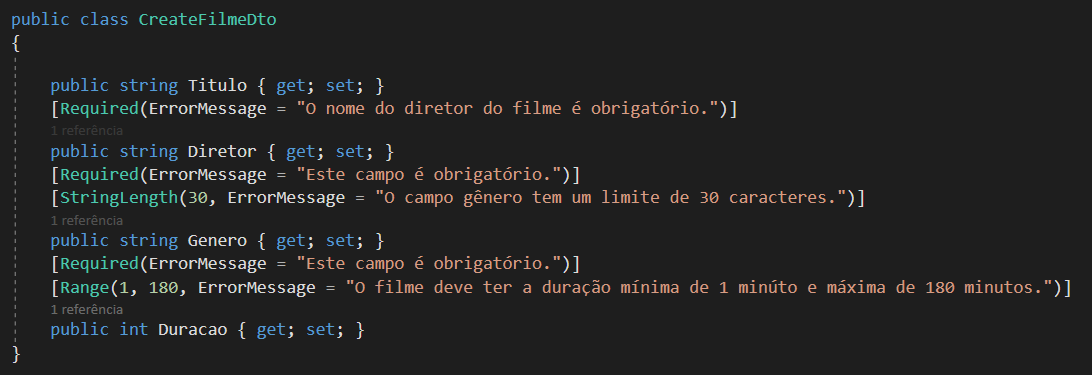
**MELHORANDO O CÓDIGO COM DTOs**

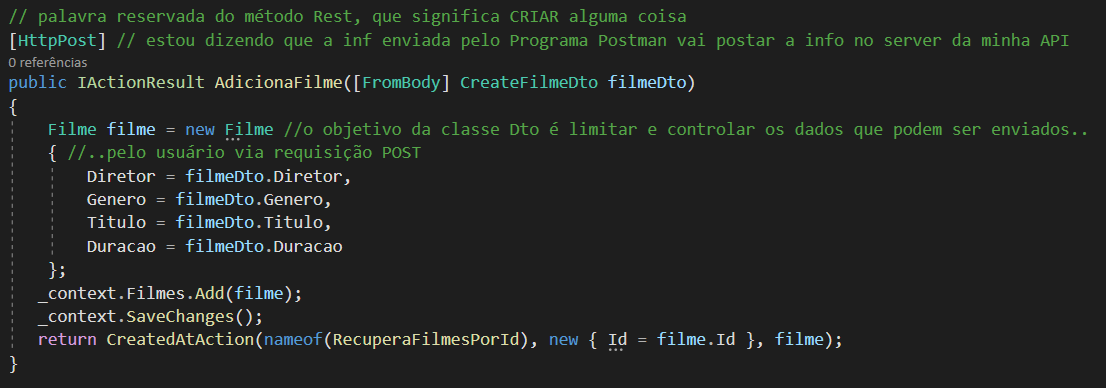
Vamos usar **Data Transfer Object (DTO)**, que é um padrão de projeto que visa criar classes que vão instanciar os objetos que transferem dados entre as partes do nosso projeto.

1° Criar o diretório **Dtos** dentro de **Data.** Cada tipo de operação vai iniciar o nome da classe DTO, como criar objetos no BD: **“Create”+Nome da Classe+”Dto”**



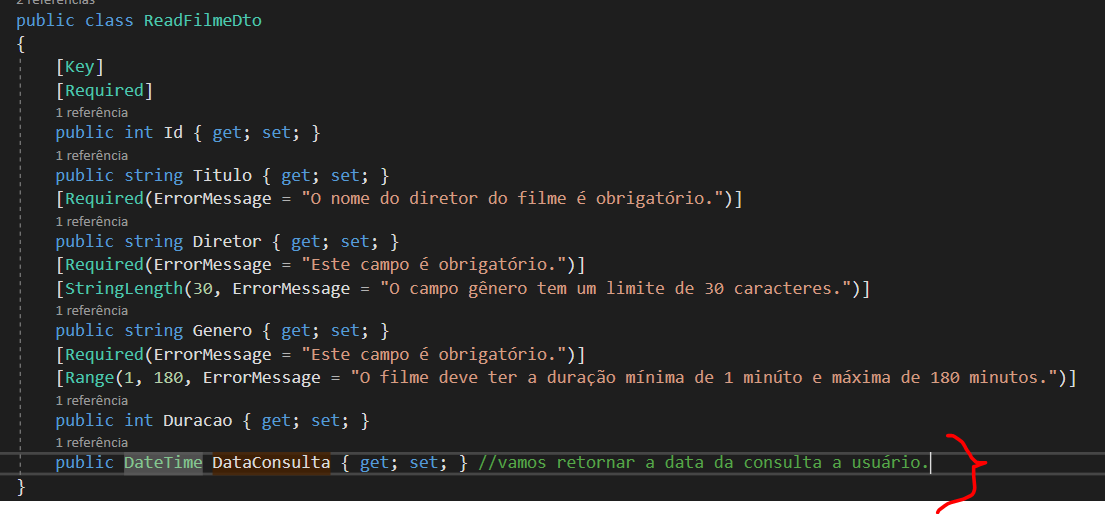
2° na classe DTO, passar apenas os atributos que poderão ser acessados:

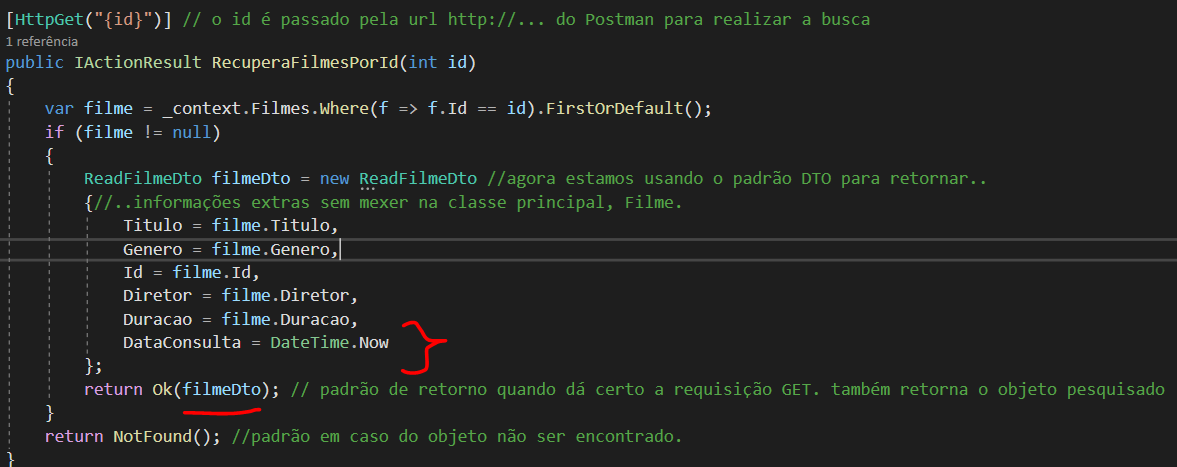
Note que tiramos a prop ID, pois nessa requisição POST ele não deve ser acessado pelo usuário.

3° Trocar o tipo do objeto no respectivo método/Action:

4° Criar os acessos DTO restantes:

Vamos criar um acesso para retornar mais dados do que a classe tem de propriedades:

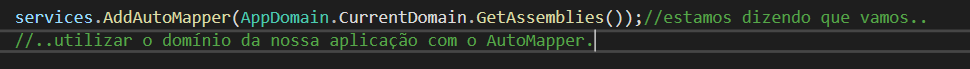
Note a eficiência dessse padrão de projeto.

Veja como fica a mudança no método:

**MELHORANDO O CÓDIGO COM AUTOMAPPER**

É um pacote que é utilizado para auxiliar na **conversão de tipos dados**, evitando a conversão manual de propriedade por propriedade, o que aumenta o número de linhas do código.

1° Baixar o framework **AutoMapper.Extensions.Microsoft.DependencyInjection** pelo NuGet.

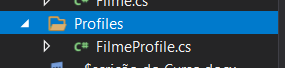
2° Adicionar o service na classe Startup:

3° Configurar quais as classes/tipos de objetos que deverão ser implicitamente convertidos pelo AutoMapper. São os chamados Perfis/Profiles.

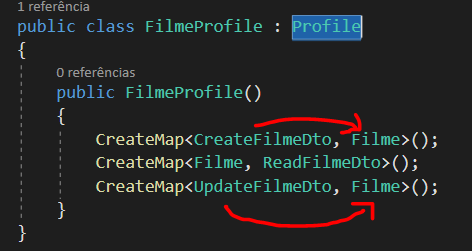
4° Criar o diretório Profiles no projeto, onde serão armazenados os tipos de conversões entre classes.

As conversões serão entre objetos de tipo Filme, somente.

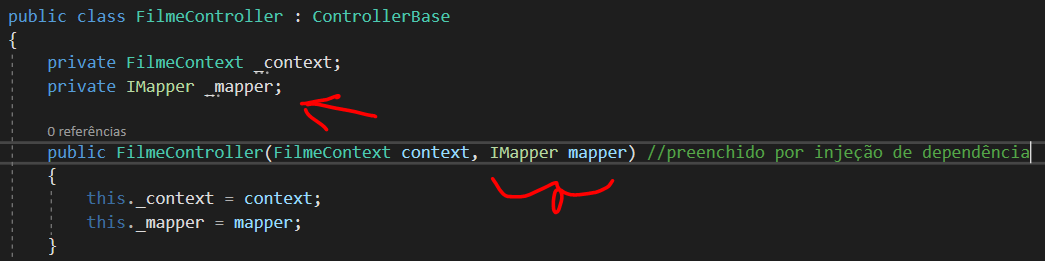
5° Criar a classe Nome+”Profile”.



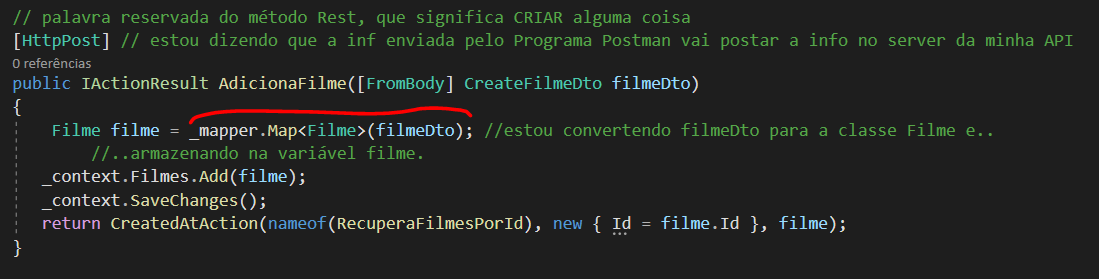
Note que devemos adicionar quais tipos serão convertidos para quais:

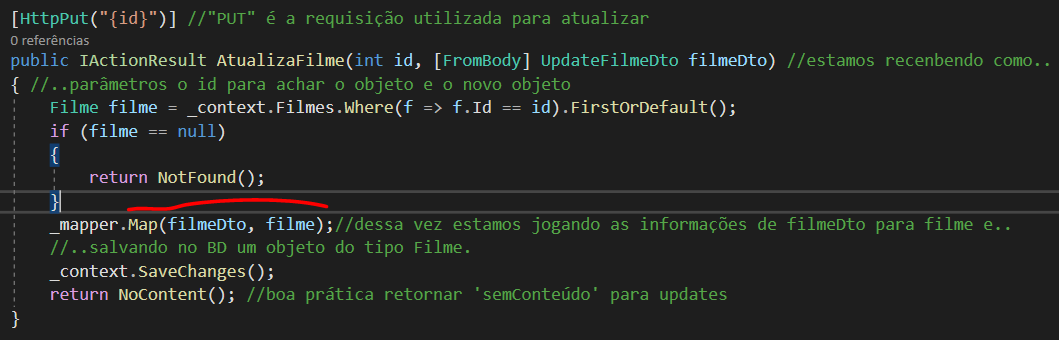


Agora vamos usar a nova classe criada, instanciando uma propriedade do tipo **IMapper** na classe que usará as conversões:



Hora de realizar as conversões. Usaremos dois tipos:

1° Converter de um tipo para o outro:

2° Jogar os dados de uma variável de um tipo para uma variável de outro.

Pronto. Está tudo mais legível e com boas práticas de desenvolvimento e funcionando perfeitamente.

**FIM DO CURSO**