03

Recebendo os dados de um filme

Transcrição

Agora, vamos criar efetivamente as partes de cadastro, leitura, alteração e remoção de filmes, começando pelo cadastro.

Primeiramente, vamos apagar os arquivos `WeatherForecastController.cs` e `WeatherForecast.cs`.

Nosso objetivo será que, ao enviar uma requisição para `/filme`, nossa API faça uma **operação de escrita**. Ou seja, vamos salvar as informações de um filme na nossa aplicação.

Criando um controlador

Para expor um caminho ou *endpoint* para que alguém possa acessar, precisamos criar um controlador. Portanto, no gerenciador de soluções, clicaremos com o botão direito do *mouse* sobre a pasta "Controllers" e selecionaremos "Adicionar > Classe...".

Uma nova janela será aberta. Na parte inferior dela, nomearemos a nova classe de "FilmeController". Vale ressaltar que a palavra "*Controller*" possui duas letras L — usar a grafia incorreta pode causar problemas no projeto.

O resultado será a classe `FilmeController`, inicialmente vazia:

```
namespace FilmesApi.Controllers
{
    public class FilmeController
    {
    }
}
```

COPIAR CÓDIGO

Vamos posicionar o cursor sobre o *namespace*, pressionar "Ctrl + Enter" e selecionar "Converter em namespace com escopo de arquivo". Assim, ganhamos mais espaço e centralizamos nosso conteúdo.

Para que essa classe seja um controlador e esteja hábil a lidar com requisições de usuários, precisamos adicionar alguns elementos. A primeira delas são as **anotações** `[ApiController]` e `[Route]`, antes da definição da classe:

```
namespace FilmesApi.Controllers;

[ApiController]
[Route]
public class FilmeController
{

}
```

COPIAR CÓDIGO

Quando o usuário enviar uma requisição para `/filme`, queremos atingir este controlador. Portanto, indicaremos que rota é para o nome do controlador. Como aprendemos anteriormente, podemos fazer essa indicação com colchetes:

```
namespace FilmesApi.Controllers;

[ApiController]
[Route("[controller]")]
public class FilmeController
{

}
```

COPIAR CÓDIGO

Atualmente, as anotações estão sublinhadas em vermelho como uma indicação de erro, pois faltam algumas **importações**. Basta posicionar o cursor sobre `[ApiController]`, pressionar "Alt + Enter" e selecionar "using Microsoft.ApsNetCore.Mvc":

```
using Microsoft.AspNetCore.Mvc;
```

```
namespace FilmesApi.Controllers;

[ApiController]
[Route("[controller]")]
public class FilmeController
{

}
```

COPIAR CÓDIGO

Além disso, é necessário que a classe `FilmeController` seja uma **extensão do** `ControllerBase`:

```
using Microsoft.AspNetCore.Mvc;

namespace FilmesApi.Controllers;

[ApiController]
[Route("[controller]")]
public class FilmeController : ControllerBase
{

}
```

COPIAR CÓDIGO

A seguir, vamos desenvolver um método chamado `AdicionaFilme()` para cadastrar um filme no sistema. A princípio, não vamos nos preocupar com o tipo de retorno, apenas usaremos o `void`:

```
using Microsoft.AspNetCore.Mvc;

namespace FilmesApi.Controllers;

[ApiController]
[Route("[controller]")]
public class FilmeController : ControllerBase
{
    public void AdicionaFilme()
```

```
    {

    }
}
```

COPIAR CÓDIGO

Ao receber uma requisição para `/filme`, receberemos por parâmetro as informações do filme que será cadastrado. Esse parâmetro será um objeto do tipo `Filme`, que chamaremos de `filme`:

```
// código anterior omitido

public class FilmeController : ControllerBase
{
    public void AdicionaFilme(Filme filme)
    {

    }
}
```

COPIAR CÓDIGO

Em seguida, adicionaremos esse filme a uma lista chamada `filmes`:

```
// ...

public class FilmeController : ControllerBase
{
    public void AdicionaFilme(Filme filme)
    {
        filmes.Add(filme);
    }
}
```

COPIAR CÓDIGO

No entanto, há dois pontos importantes. Nós precisamos:

1. criar essa lista de filmes.
2. criar a classe `Filme`, que representa um filme.

Vamos começar pelo primeiro item, que é mais simples. Antes do método `AdicionaFilme()`, criaremos uma lista estática e privada, chamada `filmes`:

```
// ...

public class FilmeController : ControllerBase
{

    private static List<Filme> filmes = new List<Filme>();

    public void AdicionaFilme(Filme filme)
    {
        filmes.Add(filme);
    }
}
```

COPIAR CÓDIGO

Na sequência, criaremos nossa classe.

Criando um modelo

A fim de manter a organização do nosso projeto, vamos inserir uma pasta que conterá nossos modelos, que mapearemos do mundo real para o mundo .NET. No gerenciador de soluções, clicaremos com o botão direito sobre "FilmesApi", selecionaremos "Adicionar > Nova Pasta" e a chamaremos de "Models".

Depois, clicaremos com o botão direito sobre a pasta "Models", selecionaremos "Adicionar > Classe...". Na parte inferior da nova janela, nomearemos a nova classe de "Filme". O resultado será a classe `Filme`, inicialmente vazia:

```
namespace FilmesApi.Models
{
    public class Filme
    {
    }
}
```

COPIAR CÓDIGO

Vamos posicionar o cursor sobre o *namespace*, pressionar "Alt + Enter" e selecionar "Converter em namespace com escopo de arquivo".

Agora, pensaremos nas propriedades de um filme. Existem inúmeras informações relevantes de um filme — neste curso, trabalharemos com:

- título
- gênero
- tempo de duração

Para adicionar uma propriedade, podemos digitar "prop" e pressionar a tecla "Tab" duas vezes para gerar um modelo e adaptá-lo. Nosso modelo `Filme`, com as três propriedades, ficará assim:

```
namespace FilmesApi.Models;

public class Filme
{
    public string Titulo { get; set; }
    public string Genero { get; set; }
    public int Duracao { get; set; }
}
```

COPIAR CÓDIGO

Após salvar o arquivo, vamos voltar ao `FilmeController.cs` e fazer a importação do *namespace* em questão. Posicionando o cursor sobre `Filme` (sublinhado em vermelho), pressionaremos "Alt + Enter" e selecionaremos "using FilmeApi.Models":

```
using FilmesApi.Models;
using Microsoft.AspNetCore.Mvc;

namespace FilmesApi.Controllers;

[ApiController]
[Route("[controller]")]
public class FilmeController : ControllerBase
{
```

```
    private static List<Filme> filmes = new List<Filme>();

    public void AdicionaFilme(Filme filme)
    {
        filmes.Add(filme);
    }
}
```

COPIAR CÓDIGO

A princípio, tudo está funcionando! O Visual Studio não está mostrando mais nenhuma indicação de erro.

Anteriormente, quando fizemos uma requisição GET para `WeatherForecast`, estávamos recuperando e **lendo** uma informação do nosso sistema. Agora, nosso objetivo é **escrever** dados, criando algo dentro do sistema. Portanto, em lugar do verbo GET, usaremos o **verbo POST**.

Se em `WeatherController` usamos a anotação `[HttpGet]`, dessa vez utilizaremos o `[HttpPost]`:

```
// ...

public class FilmeController : ControllerBase
{

    private static List<Filme> filmes = new List<Filme>();

    [HttpPost]
    public void AdicionaFilme(Filme filme)
    {
        filmes.Add(filme);
    }
}
```

COPIAR CÓDIGO

Dessa maneira, sempre que fizermos uma operação do tipo POST para o controlador de prefixo "Filme", cadastraremos o filme recebido por parâmetro. Apesar de recebê-lo por parâmetro, as informações são enviadas através do corpo da requisição e, para explicitar esse fato, usaremos a anotação `[FromBody]`:

```
// ...

public class FilmeController : ControllerBase
{

    private static List<Filme> filmes = new List<Filme>();

    [HttpPost]
    public void AdicionaFilme([FromBody] Filme filme)
    {
        filmes.Add(filme);
    }
}
```

COPIAR CÓDIGO

Sendo assim, definimos que o filme virá pelo corpo da requisição e conterá informações de título, gênero e tempo de duração. Com o método `AdicionaFilmes()`, adicionaremos esse elemento à lista `filmes`. Por ora, não estamos nos preocupando em como esse dado será armazenado, vamos simplesmente colocá-lo em uma lista. Mais adiante, podemos melhorar elaborar esses processos.

Como **validação**, vamos inserir alguns `Console.WriteLine()` para exibir o título e a duração do filme:

```
// ...

public class FilmeController : ControllerBase
{

    private static List<Filme> filmes = new List<Filme>();

    [HttpPost]
    public void AdicionaFilme([FromBody] Filme filme)
    {
        filmes.Add(filme);
        Console.WriteLine(filme.Titulo);
        Console.WriteLine(filme.Duracao);
```

```
    }
}
```

COPIAR CÓDIGO

Em seguida, vamos executar nossa aplicação. No menu superior do Visual Studio, basta pressionar o ícone de *play* com borda verde, ou usar o atalho "Ctrl + F5". Como usamos a configuração `launchBrowser: true` no arquivo `launchSettings.json`, a aplicação será executada no navegador, novamente com o Swagger.

Por enquanto, não nos interessa abrir o *browser*, então vamos alterar essa configuração e alterar seu valor para `false`:

```
// código anterior omitido

"profiles": {
    "FilmesApi": {
        "commandName": "Project",
        "dotnetRunMessages": true,
        "launchBrowser": false,
        "launchUrl": "swagger",
        "applicationUrl": "https://localhost:7106;http://localhost:510
        "environmentVariables": {
            "ASPNETCORE_ENVIRONMENT": "Development"
        }
    }
}
```

COPIAR CÓDIGO

A partir de agora, quando executarmos a aplicação, apenas um console será aberto e utilizaremos o Postman para interagir com a nossa API.

Postman

Na interface do Postman, temos um menu superior, um painel na lateral esquerda e uma grande área central denominada *workbench*. No canto superior esquerdo dessa área, clicaremos no símbolo de "+" para criar uma nova requisição.

Uma nova aba será criada, chamada "Untitled Request". À esquerda do método GET, vamos digitar a seguinte URL:

```
https://localhost:7106/filme
```

COPIAR CÓDIGO

No menu seguinte, selecionaremos a aba "Body". Depois, vamos marcar "raw" e trocar a opção "Text" para "JSON". Dessa forma, enviaremos um texto no corpo da requisição no formato JSON.

Como estamos usando o **padrão arquitetural REST**, é comum trafegar dados através de **JSON — JavaScript Object Notation**. Assim, facilitamos a maneira como trafegamos a informação e sabemos como ela será recebida ou enviada entre os diferentes consumidores e servidores do nosso sistema.

Na sequência, temos a área onde criaremos o nosso JSON. Colocaremos o título, o gênero e o tempo de duração:

```
{
    "Titulo" : "Alura Filmes",
    "Genero" : "Aventura",
    "Duracao" : 180
}
```

COPIAR CÓDIGO

Em seguida, vamos clicar no botão azul "Send", à direita da URL que definimos. No painel inferior do Postman, o resultado será o seguinte:

```
Status: 405 Method Not Allowed
```

Ou seja, "método não permitido". Enviamos um GET para `https://localhost:7106/filme`, mas nosso controlador está preparado para lidar com um POST. Então, à esquerda da URL que acabamos de digitar, vamos substituir o verbo GET por POST e clicar no botão "Send" novamente.

Agora, no painel inferior do Postman, temos o seguinte resultado:

```
Status: 200 OK
```

Acessando o console que foi aberto quando iniciamos a aplicação, notaremos que foram impressas as seguintes linhas:

```
Alura Filmes

180
```

Essas mensagens comprovam que passamos pelo método `AdicionaFilme()` e inserimos um filme na nossa lista! Mais adiante, descobriremos como ler a nossa lista de filmes.

Validação de dados

Há pouco, enviamos os seguintes dados de um filme, via Postman:

```
{
    "Titulo" : "Alura Filmes",
    "Genero" : "Aventura",
    "Duracao" : 180
}
```

COPIAR CÓDIGO

No entanto, o usuário poderia escrever informações erradas, com caracteres inválidos, por exemplo:

```
{
    "Titulo" : "Alura Filmes----------!@#!@#!@#",
    "Genero" : "Televisão",
    "Duracao" : 180000
}
```

COPIAR CÓDIGO

É necessário validar a entrada do usuário, verificando o que pode ou não ser enviado ao sistema. Na próxima aula, estudaremos como validar informações, por exemplo, quais