02

# Retornando filmes da API

## Transcrição

Já conseguimos inserir dados no nosso sistema, realizando a validação. Agora, vamos partir para as operações de busca para conferir os filmes cadastrados!

Como não estamos usando um banco de dados, vamos reiniciar a nossa aplicação para limpar os dados em memória. Uma vez reiniciada, nossa lista de filmes estará vazia novamente.

No Postman, enviaremos duas requisições POST para `https://localhost:7106/filme` com os dados de filmes, para popular nossa lista. Primeiro, o "Avatar":

```
{
    "Titulo" : "Avatar",
    "Genero" : "Ação",
    "Duracao" : 162
}
```

COPIAR CÓDIGO

Depois, "Star Wars":

```
{
    "Titulo" : "Star Wars",
    "Genero" : "Aventura",
    "Duracao" : 100
}
```

COPIAR CÓDIGO

Abrindo o console da aplicação, saberemos que as inserções foram feitas, pois temos as seguintes mensagens impressas:

```
Avatar

162

Star Wars

100
```

## Recuperando filmes

Nós já criamos o método `AdicionaFilme()` que, com o verbo HTTP POST executa a operação de inserção de recurso no sistema. Agora, para realizar uma **operação de leitura**, vamos desenvolver outro método e usar o verbo HTTP GET. No controlador, abaixo de `AdicionaFilme()`, vamos incluir o método `RecuperaFilmes()`. Inicialmente, deixaremos o tipo de retorno como `void`:

```
// ...

public void RecuperaFilmes()
{

}
```

COPIAR CÓDIGO

Vamos retornar a lista `filmes`:

```
// ...

public void RecuperaFilmes()
```

```
{
        return filmes;
}
```

COPIAR CÓDIGO

Em seguida, posicionaremos o cursor sobre a palavra `return`, pressionaremos "Alt + Enter" e selecionar "Corrigir tipo de retorno". Dessa forma, o `void` na assinatura do método será substituído por `List<Filme>`:

```
// ...

public List<Filme> RecuperaFilmes()
{
        return filmes;
}
```

COPIAR CÓDIGO

Na sequência, precisamos definir o verbo HTTP utilizado. Para uma operação de leitura, o verbo mais semântico é o GET:

```
// ...

[HttpGet]
public List<Filme> RecuperaFilmes()
{
        return filmes;
}
```

COPIAR CÓDIGO

Vale lembrar que os verbos HTTP são convenções. Como estamos seguindo o padrão arquitetural REST, é comum usar esse padrão para facilitar a interpretação do código. Ao se deparar com o `[HttpPost]`, uma pessoa rapidamente identifica que o método `AdicionaFilme()` é responsável pela inserção de recursos no sistema, por exemplo.

## Testando

Vamos salvar essas alterações e reiniciar nossa aplicação. No Postman, enviaremos duas requisições POST novamente, para popular nossa lista de filmes, com os seguintes dados:

```
{
    "Titulo" : "Avatar",
    "Genero" : "Ação",
    "Duracao" : 162
}
```

COPIAR CÓDIGO

```
{
    "Titulo" : "Star Wars",
    "Genero" : "Aventura",
    "Duracao" : 100
}
```

COPIAR CÓDIGO

À direita da aba atual do Postman, clicaremos no ícone de "+" para criar outra aba. Nela, selecionaremos o verbo GET e digitaremos a seguinte URL:

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

COPIAR CÓDIGO

Ao pressionar o botão "Send", obtemos o seguinte resultado:

```
Status: 200 OK
```

```
[
    {
```

```
        "titulo": "Avatar",
        "genero": "Ação",
        "duracao": 162
    },
    {
        "titulo": "Star Wars",
        "genero": "Aventura",
        "duracao": 100
    }
]
```

COPIAR CÓDIGO

Conseguimos recuperar os filmes cadastrados! A seguir, vamos voltar à aba da requisição POST e tentar inserir um recurso com informações inválidas. Por exemplo, com o tempo de duração igual a -1:

```
{
    "Titulo" : "Star Wars",
    "Genero" : "Aventura",
    "Duracao" : -1
}
```

COPIAR CÓDIGO

Ao enviar, receberemos um erro no painel inferior:

Status: 400 Bad Request

```
{
    "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
    "title": "One or more validation errors occurred.",
    "status": 400,
    "traceId": "00-9b23f53c18b421f70c3aa9ac33f390e0-aca586ca9aecf4b0
    "errors": {
```

```
        "Duracao": [
            "A duração deve ter entre 70 e 600 minutos"
        ]
    }
}
```

COPIAR CÓDIGO

Uma vez que houve falha na validação, esperamos que esse último filme não tenha sido inserido na nossa lista. Vamos voltar à aba da requisição GET e pressionar o botão "Send" novamente para conferir a nossa listagem. O resultado será o seguinte:

Status: 200 OK

```
[
    {
        "titulo": "Avatar",
        "genero": "Ação",
        "duracao": 162
    },
    {
        "titulo": "Star Wars",
        "genero": "Aventura",
        "duracao": 100
    }
]
```

COPIAR CÓDIGO

Continuamos apenas com o filme "Avatar" e "Star Wars", não consta o filme cujos dados não passaram pela validação. Nosso sistema está funcionando perfeitamente!

## Considerações sobre a classe `List<T>`

Por fim, comentaremos alguns pontos que envolvem **conceitos de polimorfismo**. No método `RecuperaFilmes()`, estamos retornando uma lista de filmes. Vamos fazer um "Ctrl + Clique" sobre `List<>` para explorar como a classe `List<T>` funciona.

A classe `List<T>` faz a extensão e implementação de algumas classes e interfaces, como `ICollection<T>`, `IEnumerable<T>` e `IEnumerable`.

Voltando ao método `RecuperaFilmes()`, em vez de definir o retorno como uma lista de filmes (`List<Filme>`), vamos usar um **enumerável** de filmes (`IEnumerable<Filme>`). Posteriormente, se a implementação da nossa lista for alterada e deixar de utilizar a classe `List<>` por outra classe que implemente `IEnumerable`, não precisaremos trocar a assinatura do nosso método:

```
// ...

[HttpGet]
public IEnumerable<Filme> RecuperaFilmes()
{
        return filmes;
}
```

COPIAR CÓDIGO

Ou seja, estamos deixando nosso código mais abstrato possível. Quanto menos dependermos de classes concretas, melhor para o nosso código.

Assim, desenvolvemos mais uma operação no nosso sistema. Agora, os usuários conseguem verificar a lista de filmes cadastrados. A seguir, exploraremos como retornar filmes com critérios mais específicos.