01

Criando um projeto .NET 6

Transcrição

Caso você esteja usando Linux e Visual Studio Code, mais adiante o instrutor explicará como criar um projeto com esse editor de texto também.

Criando um projeto com Visual Studio 2022

Vamos abrir o Visual Studio 2022 para criar nosso projeto. À direita da tela inicial, na seção "Introdução", clicaremos na opção "Criar um projeto".

Na barra de pesquisa (cujo atalho é "Alt + S"), vamos buscar por "api" e selecionar o modelo "API Web do ASP.NET Core" com C#. Depois, pressionaremos o botão "Próximo" no canto direito inferior.

Na próxima tela, definiremos o nome do projeto e o nome da solução. Ambos se chamarão "FilmesApi". Além disso, você pode escolher o diretório de criação do projeto, conforme sua preferência. Em seguida, clicaremos no botão "Próximo".

O passo seguinte é escolher o *framework*. Vamos utilizar o **.NET 6.0 (Suporte de longo prazo)** — ou seja, o LTS. Em seguida, clicaremos em "Criar" no canto inferior direito da tela.

O .NET 7 ainda não está na versão LTS. Não é recomendado ter projetos (seja em produção ou prontos para produção) com *frameworks* sem a versão LTS, pois perdemos o suporte tanto da comunidade quanto da própria empresa que desenvolveu a ferramenta.

Arquivos iniciais

O Visual Studio criará uma base para o nosso projeto, com alguns arquivos iniciais que exploraremos a seguir. Nós vamos acessá-los usando o gerenciador de soluções, à direita da IDE.

> Caso o gerenciador não esteja aparecendo na sua tela, basta usar o atalho "Ctrl + Alt + L" ou clicar em "Exibir > Gerenciador de Solução" no menu superior do Visual Studio.

No gerenciador de soluções, temos:

- a pasta "Properties"
- a pasta "Controllers"
- o arquivo `appsettings.json`
- o arquivo `appsettings.Development.json`
- o arquivo `Program.cs`
- o arquivo `WeatherForecast.cs`

O arquivo `WeatherForecast.cs` é um modelo para exemplificar uma API base do .NET. Por enquanto, vamos mantê-lo no projeto, pois executaremos a aplicação para entender como interagimos com a API.

O `Program.cs` é onde nossa aplicação é iniciada e onde temos as definições dos serviços que utilizaremos nela, bem como as dependências necessárias, entre outras informações.

Caso você já tenha usado versões anteriores do .NET, provavelmente reparou que esse arquivo é bem minimalista — não há definição de classe nem do escopo do *namespace*. A partir do .NET 6, não precisamos mais fazer essas definições. Basta criar o `builder`, gerar a aplicação e executá-la, em linhas enfileiras. Ao longo do curso, modificaremos bastante o arquivo `Program.cs` e não precisaremos nos preocupar com questões de indentação. Entre outras vantagens, o código ficará mais legível.

Caso o .NET 6 seja sua primeira experiência com a criação de APIs, saiba que não é necessário ter conhecimentos de versões anteriores.

Nos arquivos `appsettings.json` e `appsettings.Development.json`, definimos informações que serão carregadas durante a execução da nossa aplicação, como o endereço do banco de dados ou as credenciais usadas em determinado serviço.

Na pasta "Controllers", temos os **controladores**. Como o nome sugere, trata-se de classes responsáveis pelo controle. Elas recebem e respondem às requisições dos usuários que interagem com a API. Por ora, temos apenas o arquivo `WeatherForecastController.cs`. O uso do sufixo "Controller" no nome é uma convenção para indicar o papel da classe. Perceba que o próprio .NET adotou essa convenção ao gerar o arquivo `WeatherForecastController.cs`.

Da linha 5 a 7 desse arquivo, vale notar o uso da extensão `: ControllerBase` e das anotações `[ApiController]` e `[Route]`, importantes para o funcionamento do controlador .NET. Vamos aprender sobre elas mais adiante:

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

COPIAR CÓDIGO

Ainda nesse arquivo, temos definições de atributos estáticos, como o `Summaries` (linha 9), `_logger` (linha 14) e o construtor (linha 16). Ao final, temos um método chamado `Get()` que não recebe parâmetro nenhum e cujo retorno é um enumerável de `WeatherForecast`.

> Apesar deste curso ser voltado para quem está começando a aprender sobre APIs com .NET, é importante que você já tenha uma base de conhecimento sobre C#. Por exemplo, não vamos explicar o que é uma lista, um enumerável ou como implementar uma interface. O foco do treinamento é entender a criação de APIs e conhecer os padrões a serem seguidos.

Sabemos que o *controller* é responsável por receber e responder requisições dos usuários, porém como ele sabe que deve chamar esse método `Get()`? Em breve, vamos executar nossa aplicação para entender como esse processo funciona.

Antes disso, vamos explorar o arquivo `lauchSettings.json`, na pasta "Properties". A partir da linha 11, temos alguns `profiles`, perfis para a nossa aplicação. O que nos interessa é o trecho referente a "FilmesApi". Inclusive, vamos apagar o trecho do "IIS Express". O arquivo ficará assim:

```
{
  "$schema": "https://json.schemastore.org/launchsettings.json",
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:23847",
      "sslPort": 44335
    }
  },
  "profiles": {
    "FilmesApi": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "launchUrl": "swagger",
      "applicationUrl": "https://localhost:7106;http://localhost:5106
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

COPIAR CÓDIGO

Para salvar as alterações, pressionaremos "Ctrl + S".

Na linha 17 desse arquivo, temos a `applicationUrl`. Com protocolo HTTPS, usamos o *localhost*, na porta 7106. Com protocolo HTTP, a porta 5106. Na linha 15, temos a

configuração `lauchBrowser: true`, de modo que o navegador será aberto automaticamente, quando a aplicação for iniciada.

Executando a aplicação

A seguir, vamos executar a aplicação. No menu superior da IDE, temos o símbolo de uma seta para a direita com preenchimento verde claro. À sua direita, há uma pequena seta branca para baixo. Clicaremos nela e selecionaremos "FilmesApi". Não queremos WSL.

Em seguida, vamos **iniciar sem depurar**, clicando no símbolo de seta para a direita com o contorno verde claro, sem preenchimento. Alternativamente, podemos pressionar "Ctrl + F5". A aplicação será aberta no navegador.

Estamos trabalhando com uma API e o foco não é *front-end*, contudo note que nossa aplicação tem uma tela bonita, bem estruturada e interativa, com links clicáveis. Esse resultado é obtido com o **Swagger**. Estudaremos mais sobre ele no final deste curso.

Abaixo do título "WeatherForecast", temos a seção "GET /WeatherForecast", indicando que podemos acessar este caminho. Vamos clicar nela para expandi-la. Em seguida, clicaremos no botão "Try it out" (Testar), no canto superior direito da área expandida e, depois, em "Execute".

Mais abaixo, no tópico "Server Response", recebemos um *response body* com *status code* 200 e informações de data e temperatura. Examinando esse retorno, reparamos que ele nada mais é que um enumerável convertido em *array*, como conferimos no método `Get()` do nosso controlador.

Analisando o tópico "Curl", sabemos que a rota utilizada foi `https://localhost:7106/WeatherForecast`. No arquivo `lauchSettings.json`, verificamos há pouco que o caminho em que nossa aplicação está sendo executada é `https://localhost:7106`. Mas como o controlador sabe que, com `/WeatherForecast`, ele deve executar o método `Get()`?

Anteriormente, reparamos que o nome da classe `WeatherForecastController` segue a convenção de usar o sufixo "Controller". Na anotação `[Route("[controller]")]` na

linha 6, os colchetes indicam que devemos selecionar o prefixo do nome da classe e colocá-lo no lugar de `[controller]`. Ou seja, o controlador interpretará a rota como `[Route("WeatherForecast")]`.

A vantagem de usar a anotação `[Route("[controller]")]` em vez de `[Route("WeatherForecast")]` é que, se eventualmente alterarmos o nome da classe, não será necessário adaptar a anotação também. Assim, tudo fica mais prático e fácil.

Além disso, imediatamente acima do método `Get()`, temos a seguinte anotação:

```
[HttpGet(Name = "GetWeatherForecast")]
```

COPIAR CÓDIGO

O `HttpGet` é uma das operações que podem ser executadas em uma API. Na nossa aplicação, foi exatamente essa ação que executamos ao expandir a seção "GET /WeatherForecast" e pressionar o botão "Execute".

Dessa forma, o controlador sabe que, ao acessar a rota `/WeatherForecast` com o verbo GET, ele deve executar toda a lógica do método `Get()`!

Não se preocupe se esse processo não ficou muito claro ainda. Ao longo do curso, vamos recriar essa lógica com nossas próprias rotas e aprender mais a fundo como tudo funciona. Por enquanto, a ideia era apenas visualizar esse escopo inicial.

Namespace com escopo de arquivo

No arquivo `WeatherForecastController.cs`, note que todo o código está indentado para que a classe e as anotações fiquem dentro de um *namespace*, em um bloco de chaves. O .NET 6 oferece um recurso para aprimorar essa estrutura. Posicionando o cursor sobre o nome do *namespace* (linha 3), pressionaremos "Alt + Enter" e selecionaremos "Converter em namespace com escopo de arquivo". Assim, o código ficará indentado mais à esquerda e o bloco de chaves do *namespace* é removido. O projeto torna-se mais centralizado e legível dessa forma.

Criando um projeto com Visual Studio Code

Até agora, usamos apenas o Visual Studio 2022. Caso você tenha optado pelo **Visual Studio Code**, há duas extensões que recomendamos instalar para desenvolver projetos .NET. No painel à esquerda do VS Code, basta acessar a aba "Extensions" e pesquisar as seguintes extensões:

- C# (da Microsoft)
- C# Snippets (de Jorge Serrano)

Em seguida, vamos criar um projeto via linha de comando. No menu superior do VS Code, selecionaremos "Terminal > New Terminal". Alternativamente, podemos usar o atalho "Ctrl + Shift + '".

Um novo terminal será aberto na parte inferior da tela. Vamos criar uma pasta chamada "projetolinux", com o seguinte comando:

```
mkdir projetolinux
```

COPIAR CÓDIGO

Em seguida, vamos acessar a pasta:

```
cd projetolinux
```

COPIAR CÓDIGO

E abrir uma nova instância do Visual Studio Code nela:

```
code .
```

COPIAR CÓDIGO

Na nova instância, abriremos o terminal novamente, com "Ctrl + Shift + '". Para criar um projeto .NET, rodaremos o seguinte comando:

```
dotnet new webapi --name FilmesApi
```

COPIAR CÓDIGO

Assim, criaremos um modelo de Web API com o nome "FilmesApi". No painel à esquerda, temos a mesma estrutura de pastas que teríamos no Visual Studio 2022!

Para executar o projeto, basta rodar o seguinte comando no terminal:

```
dotnet run --project FilmesApi/FilmesApi.csproj
```

COPIAR CÓDIGO

Após a compilação, serão mostradas várias informações no terminal, como o endereço de execução da aplicação. No caso, no *localhost*, na porta 7174 com protocolo HTTPS, ou na porta 5177 com HTTP. Basta dar um "Ctrl + Clique" sobre o endereço para acessá-lo no navegador.

Inicialmente, teremos um erro, pois é preciso acessar o endereço com `/swagger` ao final. Portanto, usaremos o seguinte endereço:

```
https://localhost:7174/swagger/index.html
```

COPIAR CÓDIGO

O foco dessa aula foi apenas criar o projeto, conferir algumas particularidade do .NET 6 e entender como interagir com uma API. Caso tenha ficado alguma dúvida, não se preocupe! Na próxima aula, vamos recriar toda essa lógica do início ao fim com nossos próprios controladores e modelos. Pensaremos como apagar os elementos de *weather forecast* e começar a migrar para o nosso conceito de filmes.