

APOSTILA DO CURSO

LINGUAGEM C# e BLAZOR

Abril/2024

Essa apostila foi elaborada exclusivamente para o treinamento na empresa SuperSoft Sistemas realizado no dia 13/04/2024 na cidade de Rio Claro, SP. Boa parte da teoria do seu conteúdo foi obtida do site oficial da Microsoft e pode haver erros de tradução do idioma Inglês para o Português, tanto em palavras quanto em concordância da frase.

LINGUAGEM C# e BLAZOR

Tipos de Dados:

Existem diversos tipos de dados no C#, para entendê-los de uma maneira mais simples, separamos três tipos, o Tipo por Valor, o Tipo por Referência e o Tipo String, que é um tipo especial do .NET.

Tipo por Valor

Tipos por valor tem como características:

- São alocados diretamente na memória Stack (pilha).
- Não precisam ser inicializados com o operador new.
- A variável armazena o valor diretamente.
- A atribuição de uma variável a outra copia o conteúdo, criando efetivamente outra cópia da variável.
- Cada variável tem a definição de seu tamanho mínimo e máximo.

Exemplo de tipos por Valor:

Tipo	Implementação
byte	Inteiro de 8 bits sem sinal (0 a 255).
sbyte	Inteiro de 8 bits com sinal (-127 a 128).
ushort	Inteiro de 16 bits sem sinal (0 a 65 535).
short	Inteiro de 16 bits com sinal (-32 768 a 32 767).
uint	Inteiro de 32 bits sem sinal (0 a 4 294 967 295).
int	Inteiro de 32 bits com sinal (-2 147 483 648 a 2 147 483 647).
ulong	Inteiro de 64 bits sem sinal (0 a 18 446 744 073 709 551 615).
long	Inteiro de 64 bits com sinal (-9 223 372 036 854 775 808 a 9 223 372 036 854 775 807).
double	Ponto flutuante binário IEEE de 8 bytes ($\pm 5.0 \times 10^{-324}$ a $\pm 1.7 \times 10^{308}$), 15 dígitos decimais de precisão.
float	Ponto flutuante binário IEEE de 4 bytes ($\pm 1.5 \times 10^{-45}$ a $\pm 3.4 \times 10^{38}$), 7 dígitos decimais de precisão.
decimal	Ponto flutuante decimal de 128 bits. (1.0×10^{-28} a 7.9×10^{28}), 28 dígitos decimais de precisão.
bool	Pode ter os valores true ou false. Não é compatível com inteiro.
char	Um único caractere Unicode de 16 bits. Não é compatível com inteiro.

Tipo por Referência

Tipos por Referência são tipos de dados que não armazenam valores em si, somente contém um ponteiro que direciona para um espaço específico. Mais de uma variável pode indicar um mesmo espaço de memória.

Todos os tipos por Referência são obrigatoriamente inicializados pela palavra “new”, que é responsável pelo apontamento.

Exemplo: Tipo de dado por referência em C#: Class

Variações

O tipo String, é uma variação de possibilidades do C#. Ele apesar de ser um tipo por Referência, nos permite utilizá-lo como um tipo por valor. Devemos lembrar que tipos de “texto” são cadeias de caracteres, ou seja, podemos entender um String como um vetor de Char.

Variáveis

Uma variável representa um valor numérico ou cadeia de caracteres ou um objeto de uma classe. O valor que armazena a variável poderá ser alterado, mas o nome permanece o mesmo. Uma variável é um tipo de campo que serve para manipular as informações durante o fluxo do programa.

Exemplo: `int velocidadeAgora = 75;`

`bool vontadeDeEstudar = true;`

Constantes

Uma constante é um tipo diferente de campo. Seu funcionamento é semelhante ao da criação de variáveis, porém seu valor após compilado o código, jamais será alterado.

Para definir um valor constante basta adicionar a palavra *const* antes da declaração da variável.

Exemplo: `const int velMaxMargTiete = 90;`

`const double pi = 3.14159265358979323846264338327950;`

Operadores

Aritméticos

Quando pensamos em C# os operadores aritméticos podem ser entendidos basicamente como:

Operadores	Função
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto/Módulo

Adição	Subtração
<pre>int a, b, c; a = 3; b = 4; c = a + b; //Resultado c = 7.</pre>	<pre>int a, b, c; a = 3; b = 4; c = a - b; //Resultado c = -1.</pre>
Multiplicação	Divisão
<pre>int a, b, c; a = 3; b = 4; c = a * b; //Resultado c = 12.</pre>	<pre>int a, b, c; a = 11; b = 2; c = a / b; //Resultado c = 5. - Não pega o resto da divisão (no caso de inteiros).</pre>
Resto/Módulo	
<pre>int a, b, c; a = 11; b = 2; c = a % b; //Resultado c = 1. - Só pega o resto da divisão.</pre>	

Comparação

Operadores de Comparação:

Operador	Significado
==	Igualdade
>	Maior
<	Menor
<=	Menor Igual
>=	Maior igual
!=	Diferente

Exemplo

```
int A = 5; int B = 3;
```

```
A == B // Falso
```

```
A > B // Verdadeiro
```

```
A < B // Falso
```

```
A <= B // Falso
```

```
A >= B // Verdadeiro
```

```
A != B // Verdadeiro
```

Lógicos

Operadores lógicos:

Operador	Significado
&&	E
	OU

Exemplo && (E)

```
int num1 = 1, num2 = 3;
```

```
bool PrimeiroTeste = num1 > 3 && num2 < 10
```

```
//PrimeiroTeste == false (falso)
```

Para que um teste lógico seja verdadeiro perante dois testes, ambos devem ser verdadeiros.

Exemplo || (OU)

```
int num1 = 1, num2 = 3;
```

```
bool SegundoTeste = num1 > 3 || num2 < 10
```

```
//SegundoTeste == true (verdadeiro)
```

Atribuição

Operadores de Atribuição: Operador	Significado	Exemplo
=	Atribuição Simples	string simp = "simples";
+=	Atribuição Aditiva	int a = 3; a += 1; //a = 4
-=	Atribuição Subtrativa	int a = 3; a -= 1; //a = 2
*=	Atribuição Multiplicativa	int a = 3; a *= 4; //a = 12
/=	Atribuição de divisão	int a = 8; a /= 2; //a = 4
%=	Atribuição Modular	int a = 11; a %= 2; //a = 1
++	Adição de 1 unidade	int a = 11; a++; //a=12
--	Subtração de 1 unidade	int a = 11; a--; //a=10

Concatenação

Operador de concatenação
Operador

+

Significado

Concatena caracteres e cadeias de caracteres

Exemplo

```
string algumaVariavel = "olá, meu nome é ";
algumaVariavel = algumaVariavel + "João";
```

// ou utilizando de uma maneira mais simples:
string algumaVariavel = "olá, meu nome é ";

algumaVariavel += "João";

Comandos Condicionais

Estruturas de Decisão

Qualquer linguagem de programação necessita de estruturas de decisão. Isso significa que uma das bases da programação é a comparação. A programação em C# não seria diferente e para isso existem os comandos if (se) e else (senão) e switch (escolha).

Comandos If / Else / Else if

If – Else

A sintaxe dos comandos if e else estão diretamente ligadas ao uso de operadores.

```
if ( expressão lógica )
{
    //Código se expressão lógica for verdadeira
}
else
{
    //Código se falsa
}
```

Exemplo if (bool) {} else {}

```
int a, b;
a = 1; b = 2;
if (a == b)
{
    Console.WriteLine("A e B são iguais");
}
else
{
    Console.WriteLine("A e B são diferentes");
}
```

Lembro-vos que podemos fazer a comparação de qualquer tipo, desde que com o mesmo tipo, String com String, Int com Int e assim por diante.

Else If

Outra possibilidade com o comando if (se) é o encadeamento de decisões. Existem momentos em que nossa opção não é somente verdadeira ou falsa, conforme no exemplo abaixo.

Exemplo if (bool) {} else if () {}

```
Console.WriteLine("Digite o dia da semana em que estamos");
string diaSemana = Console.ReadLine().ToUpper();
if (diaSemana == "SEGUNDA-FEIRA")
{
    Console.WriteLine("Odeio esse dia!");
}
else if (diaSemana == "TERÇA-FEIRA")
{
    Console.WriteLine("Falta muito pro fds?");
}
else if (diaSemana == "QUARTA-FEIRA")
{
    Console.WriteLine("Já estou me arrastando");
}
else if (diaSemana == "QUINTA-FEIRA")
{
    Console.WriteLine("Chega natal e não chega sexta");
}
else if (diaSemana == "SEXTA-FEIRA")
{
    Console.WriteLine("Sextou, bora tomar uma");
}
else if (diaSemana == "SABADO")
{
    Console.WriteLine("Almoçar na casa da vó");
}
else if (diaSemana == "DOMINGO")
{

```



```
        Console.WriteLine("Dormir o dia todo");  
    }  
    else  
    {  
        Console.WriteLine("Você deve preencher somente os dias da semana e por  
extenso, por exemplo: Quarta-Feira");  
    }  
    Console.Read();
```

Estruturas / Laços de Repetição

Comando While / Do While

While

O comando `while` (enquanto) é o mais simples dos laços de repetição quando utilizamos C#. Podemos entendê-lo como um repetidor de informação devido a uma expressão lógica.

Exemplo

```
int n = 1;
while (n <= 10)
{
    Console.WriteLine(String.Format("Valor corrente de n é: {0}", n));
    n++;
}
Console.Read();
```

Do While

A única diferença entre o *While* e o *Do While* é a **primeira execução**. No caso do *While*, somente será executado o comando caso a validação desde a primeira tentativa (execução) seja verdadeira. No *Do While*, o código é executado a primeira vez e posteriormente é validado.

Exemplo

```
int n = 1;
do
{
    Console.WriteLine("Valor de n é: " + n);
    n++;
}
while (n >= 10);
Console.Read();
```

Comando For

O comando For tem as mesmas utilidades de um comando While, porém o incremento da variável cuja expressão lógica está vinculada é feito dentro do próprio comando.

Exemplo

```
int numero;
Console.WriteLine("Até qual número devo escrever de 1 em 1?");
numero = Convert.ToInt32(Console.ReadLine());
for (int i = 0; i < numero; i++)
{
    Console.WriteLine(i+1);
}
Console.Read();
```

Exemplo avançado

No exemplo abaixo vamos relacionar diversos ensinamentos até aqui, colocando em prática e aprendendo como lidar com situações mais complexas. Nesse exemplo iremos relacionar os comandos While, For, comandos de input e output, estruturas de decisão e funções predefinidas.

O nosso objetivo é fazer um programa que leia um número e peça para o computador contar e escrever de 1 até o número digitado. Além disso, o programa deverá perguntar se queremos fazer novamente a contagem, porém com um valor novo. Segue o código abaixo para análise.

```
int numero;
bool satisfeito = false;
while (satisfeito == false)
{
    Console.WriteLine("Até qual número devo escrever de 1 em 1?");
    numero = Convert.ToInt32(Console.ReadLine());
    for (int i = 0; i < numero; i++)
    {
        Console.WriteLine(i + 1);
    }
    Console.WriteLine("Satisfeito? S - para Sim / N - para Não");
```

```
    if (Console.ReadLine().Equals("S"))  
    {  
        satisfeito = true;  
    }  
}  
Console.Read();
```

Comando Foreach

Em lógica de programação vimos que existem variáveis que armazenam informações em forma de vetores e matrizes. O Foreach (para cada um) é o comando mais utilizado para percorrer esse vetor/matriz valor a valor.

Quando ele faz a leitura ele realiza a conversão do vetor de maneira automática a cada passagem, fazendo com que a leitura seja mais demorada em relação ao for.

OBS: A sintaxe de vetores e matrizes serão revistas no próximo módulo.

Exemplo

```
int[] arrayDeInteiros = new int[] { 0, 1, 1, 2, 3, 5, 8, 13 };  
foreach (int inteiro in arrayDeInteiros)  
{  
    Console.WriteLine(inteiro);  
}  
Console.Read();
```

Variáveis Indexadas

Variáveis Indexadas Unidimensionais (Vetores/Arrays)

Array

O C# assim como a maioria das linguagens de programação atuais suporta indexação de valores de mesmo tipo em uma estrutura única, o Vetor ou também chamado de Array.

Vetores em C# são definidos pela classe ou tipo e pelos caracteres "[]", conforme abaixo.

Exemplo

```
int[] arrayDeInteiros = new int[] { 0, 1, 1, 2, 3, 5, 8, 13 };  
string[] arrayDeTextos = new string[] { "Segunda", "Terça", "Quarta" };
```

Dessa maneira conseguimos criar vetores unidimensionais de tamanho fixo. Onde não podemos alterar a quantidade de valores que existem dentro do vetor. Um vetor de 10 posições, pode ter no máximo 10 valores.

Outra maneira de desenvolvermos a mesma ideia é estipular o tamanho do vetor na sua criação e posteriormente adicionar valores a cada posição, conforme o exemplo:

```
int[] arrayDeInteiros = new int[5];  
arrayDeInteiros[0] = 1;  
arrayDeInteiros[1] = 2;  
arrayDeInteiros[2] = 3;  
arrayDeInteiros[3] = 4;  
arrayDeInteiros[4] = 5;  
for (int i = 0; i < arrayDeInteiros.Length; i++)  
{  
    Console.WriteLine(arrayDeInteiros[i]);  
}  
Console.Read();
```

Ou para textos:

```
string[] arrayTextos = new string[5];  
arrayTextos[0] = "Um";  
arrayTextos[1] = "Dois";
```

```
arrayTextos[2] = "Três";
arrayTextos[3] = "Quatro";
arrayTextos[4] = "Cinco";

for (int i = 0; i < arrayTextos.Length; i++)
{
    Console.WriteLine(arrayTextos[i]);
}

Console.Read();
```

Listas - Collections

Em linguagens de programação mais antigas, como o C, existiam momentos em que gostaríamos de aumentar o tamanho de nossos arrays, pois precisávamos de maior quantidade de dados armazenados. Fazíamos isso através da criação de arrays muito maiores do que o necessário, pois não sabíamos o que estaria por vir. No C# existem objetos que funcionam como um array, porém são flexíveis. Esses objetos são derivações do tipo `Collections.Generic`.

Esses objetos contêm as propriedades de vetores, porém podemos inserir valores nele, não sendo necessário fixar seu tamanho.

Para adicionar e remover itens de uma lista (`List`) utilizamos algumas funções pré definidas: conforme os exemplos do objeto `List`.

Exemplo

Add (objeto)

Para adicionarmos a uma lista de inteiros um novo número inteiro utilizamos o método `Add(inteiro)`.

```
List<int> inteiros = new List<int>();
inteiros.Add(1);
inteiros.Add(2);
inteiros.Add(5);
inteiros.Add(7);
for (int i = 0; i < inteiros.Count; ++i)
{
    Console.Write(inteiros[i] + " ");
}

Console.Read();
```

Remove (objeto)

Para remover um objeto específico de sua lista, utilizamos a função `Remove()` passando como parâmetro qual o objeto a ser removido, conforme o exemplo.

Exemplo

```
List<string> listTextos = new List<string>();  
listTextos.Add("Um");  
listTextos.Add("Dois");  
listTextos.Add("Três");  
listTextos.Add("Quatro");  
listTextos.Remove("Dois");  
for (int i = 0; i < listTextos.Count; ++i)  
{  
    Console.Write(listTextos[i] + " ");  
}  
Console.Read();
```

Melhor Guia para estudo da linguagem do C#: <https://docs.microsoft.com/pt-br/dotnet/csharp/>

INTRODUÇÃO AO ASP .NET CORE BLAZOR

Bem-vindo ao Blazor !

Blazor é uma estrutura Web de front-end do .NET que dá suporte à renderização do lado do servidor e à interatividade do cliente em um único modelo de programação:

- Crie interfaces de usuário interativas avançadas usando C#.
- Compartilhe a lógica de aplicativo do lado do cliente e do servidor gravada no .NET.
- Renderize a interface do usuário, como HTML e CSS para suporte amplo de navegadores, incluindo navegadores móveis.
- Crie aplicativos móveis e de área de trabalho híbrida com .NET e Blazor.

Usar o .NET para desenvolvimento web do lado do cliente oferece as seguintes vantagens:

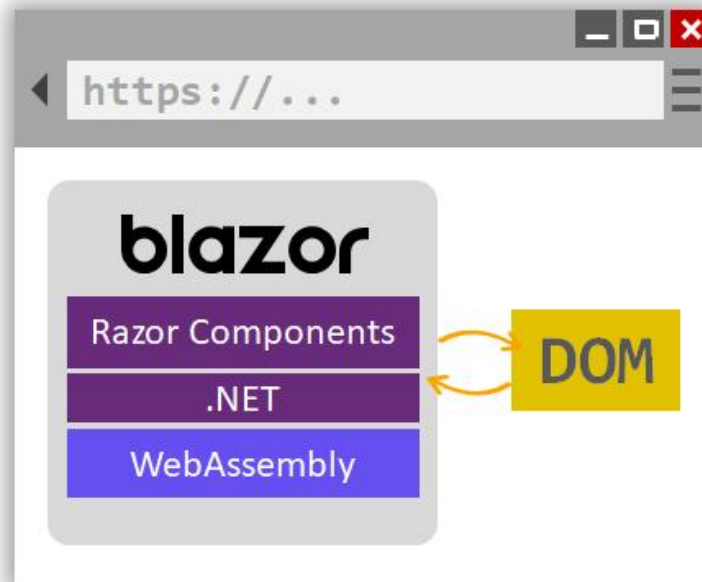
- Escreva código em C#, o que pode melhorar a produtividade no desenvolvimento e manutenção de aplicativos.
- Aproveite o ecossistema .NET existente de [bibliotecas .NET](#).
- Beneficie-se com o desempenho, confiabilidade e segurança do .NET.
- Mantenha-se produtivo no Windows, Linux ou macOS com um ambiente de desenvolvimento, como o Visual Studio ou o Visual Studio Code. Integre-se a plataformas de hospedagem modernas, como o Docker.
- Crie um conjunto comum de linguagens, estruturas e ferramentas que são estáveis, com recursos avançados e fáceis de usar.

Blazor WebAssembly

Blazor WebAssembly é uma [estrutura de Spa \(aplicativo de página única\)](#) para a criação de aplicativos Web do lado do cliente interativos com o .net. Blazor WebAssembly usa padrões abertos da Web sem plugins ou recompilando código em outras linguagens. Blazor WebAssembly funciona em todos os navegadores da Web modernos, incluindo navegadores móveis.

A execução de código .NET dentro de navegadores da Web é possibilitada pelo [WebAssembly](#) (abreviado wasm). O WebAssembly é um formato de código de bytes compacto, otimizado para download rápido e máxima velocidade de execução. O WebAssembly é um padrão aberto da Web compatível com navegadores da Web sem plug-ins.

O código WebAssembly pode acessar a funcionalidade completa do navegador por meio de JavaScript, chamada *interoperabilidade JavaScript*, geralmente abreviada para interoperabilidade *JavaScript* ou *interoperabilidade JS*. O código .NET executado por meio da WebAssembly no navegador é executado na área restrita do JavaScript do navegador com as proteções que a área restrita oferece contra ações mal intencionadas no computador cliente.



Quando um Blazor WebAssembly aplicativo é criado e executado em um navegador:

- Arquivos de código C# Razor e outros arquivos são compilados em assemblies do .NET.
- Os assemblies e o [runtime do .NET](#) são baixados no navegador.
- Blazor WebAssembly inicializa o runtime do .NET e configura o runtime para carregar os assemblies para o aplicativo. O Blazor WebAssembly runtime usa a interop JavaScript para lidar com a manipulação de DOM e chamadas à API do navegador.

O tamanho do aplicativo publicado, seu tamanho *de carga*, é um fator de desempenho crítico para a usabilidade de um aplicativo. Um aplicativo grande leva um tempo relativamente longo para baixar para um navegador, o que afeta a experiência do usuário. Blazor WebAssembly otimiza o tamanho do conteúdo para reduzir os tempos de download:

- O código não utilizado é removido do aplicativo quando ele é publicado pelo IL [\(Intermediate Language\) Trimmer](#).
- As respostas HTTP são compactadas.
- O runtime do .NET e os assemblies são armazenados em cache no navegador.

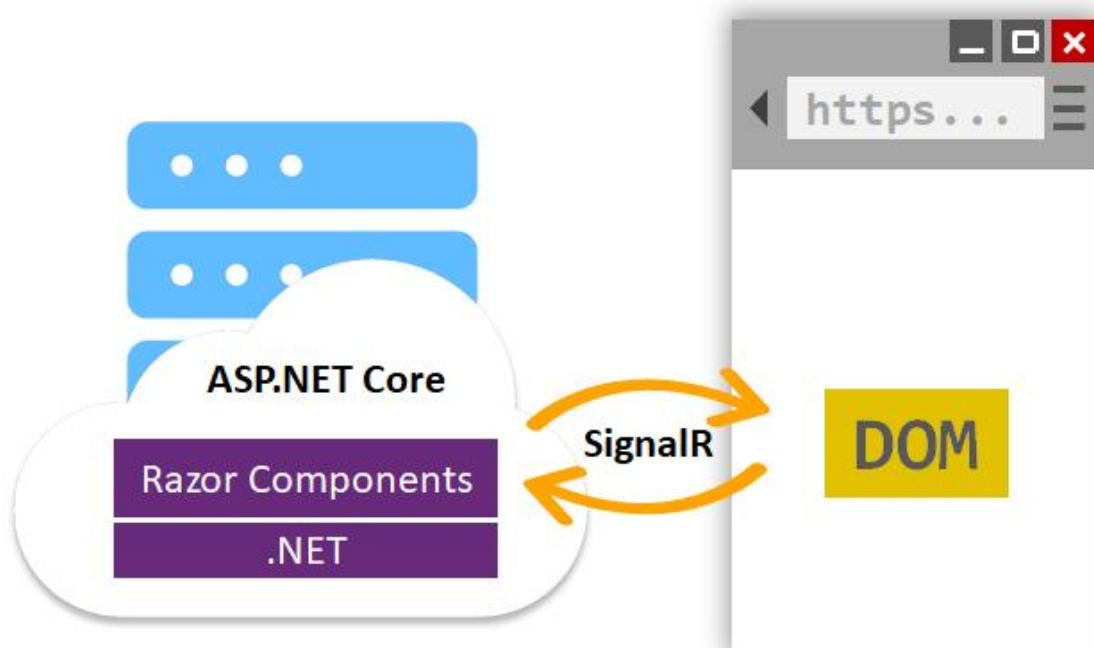
Blazor Server

Blazor desacopla a lógica de renderização de componentes de como as atualizações da interface do usuário são aplicadas. *Blazor Server* fornece suporte para hospedar Razor componentes no servidor em um ASP.NET Core. As atualizações da interface do usuário são tratadas em uma [SignalR](#) conexão.

O runtime permanece no servidor e lida com:

- Executar o código C# do aplicativo.
- Enviar eventos de interface do usuário do navegador para o servidor.
- Aplicar atualizações de interface do usuário ao componente renderizado que é enviado de volta pelo servidor.

A conexão usada pelo Blazor Server para se comunicar com o navegador também é usada para lidar com chamadas de interop JavaScript.



Interoperabilidade do JavaScript

Para aplicativos que exigem bibliotecas JavaScript e acesso a APIs do navegador de terceiros, os componentes interoperam com o JavaScript. Os componentes são capazes de usar qualquer biblioteca ou API que o JavaScript possa usar. O código C# [pode chamar no código JavaScript](#) e o código JavaScript pode chamar no código [C#](#).

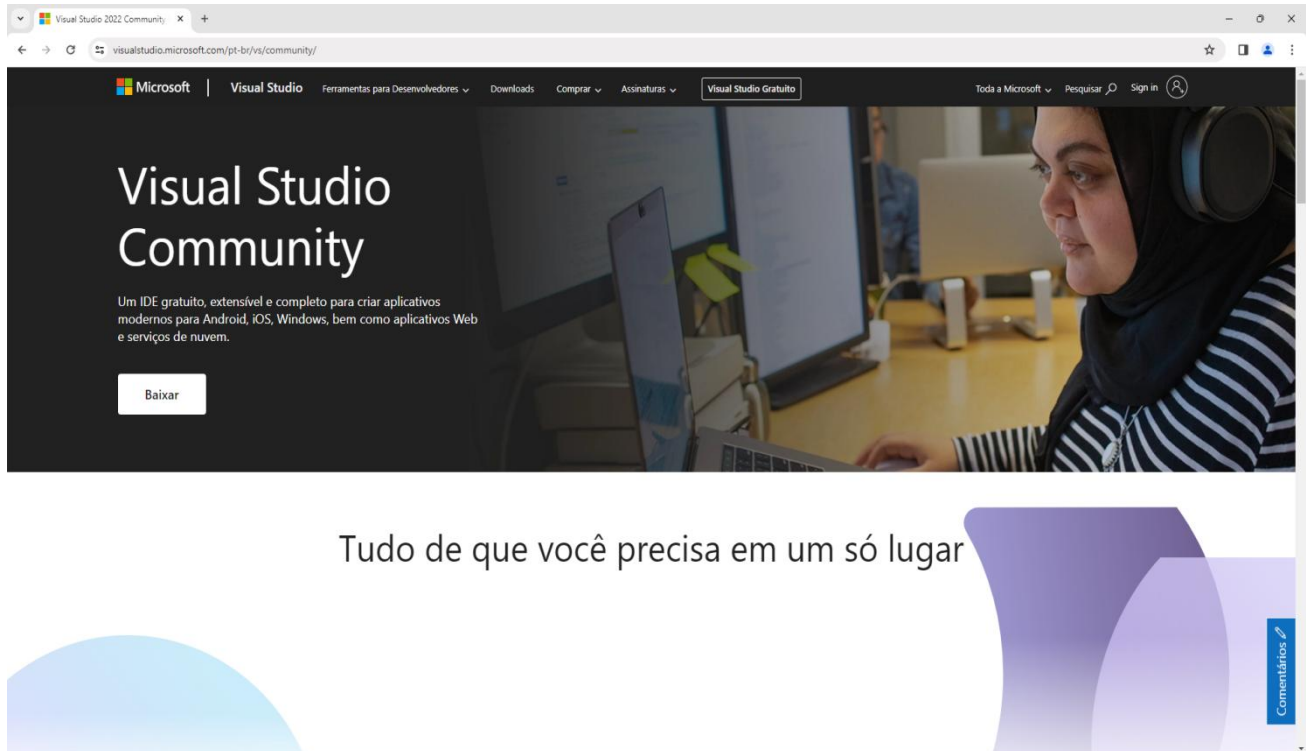
Compartilhamento de código e o .NET Standard

Blazor implementa o [.net Standard](#), que permite que os projetos Blazor referenciem bibliotecas que estão em conformidade com as especificações .net Standard. O .NET Standard é uma especificação formal das APIs do .NET que são comuns entre as implementações do .NET. As bibliotecas de classe .NET Standard podem ser compartilhadas entre diferentes plataformas .NET, como Blazor, .NET Framework, .NET Core, Xamarin, mono e Unity.

Fonte de dados: <https://docs.microsoft.com/pt-br/aspnet/core/blazor/?view=aspnetcore-5.0>

Preparando o ambiente: Instalações

Acessar o endereço <https://visualstudio.microsoft.com/pt-br/vs/community/> e baixar o Visual Studio Community. Essa versão já vem com o SDK do Dot Net 8.



Caso já possua o Visual Studio instalado em seu computador, porém não tenha instalado o SDK do Dot Net 8, faça o próximo passo abaixo.

Acessar o endereço <https://dotnet.microsoft.com/pt-br/download/dotnet/8.0> e baixar o SDK do Dot Net 8 versão Windows x64 (ou correspondente ao seu Sistema Operacional).

ⓘ Não tem certeza do que baixar? [Consulte os downloads recomendados para a versão mais recente do .NET.](#)

[Notas de versão](#) **Data de lançamento mais recente** 9 de janeiro de 2024

SDK 8.0.101

Suporte para o Visual Studio
Visual Studio 2022 (v17.8)

Runtimes incluidos

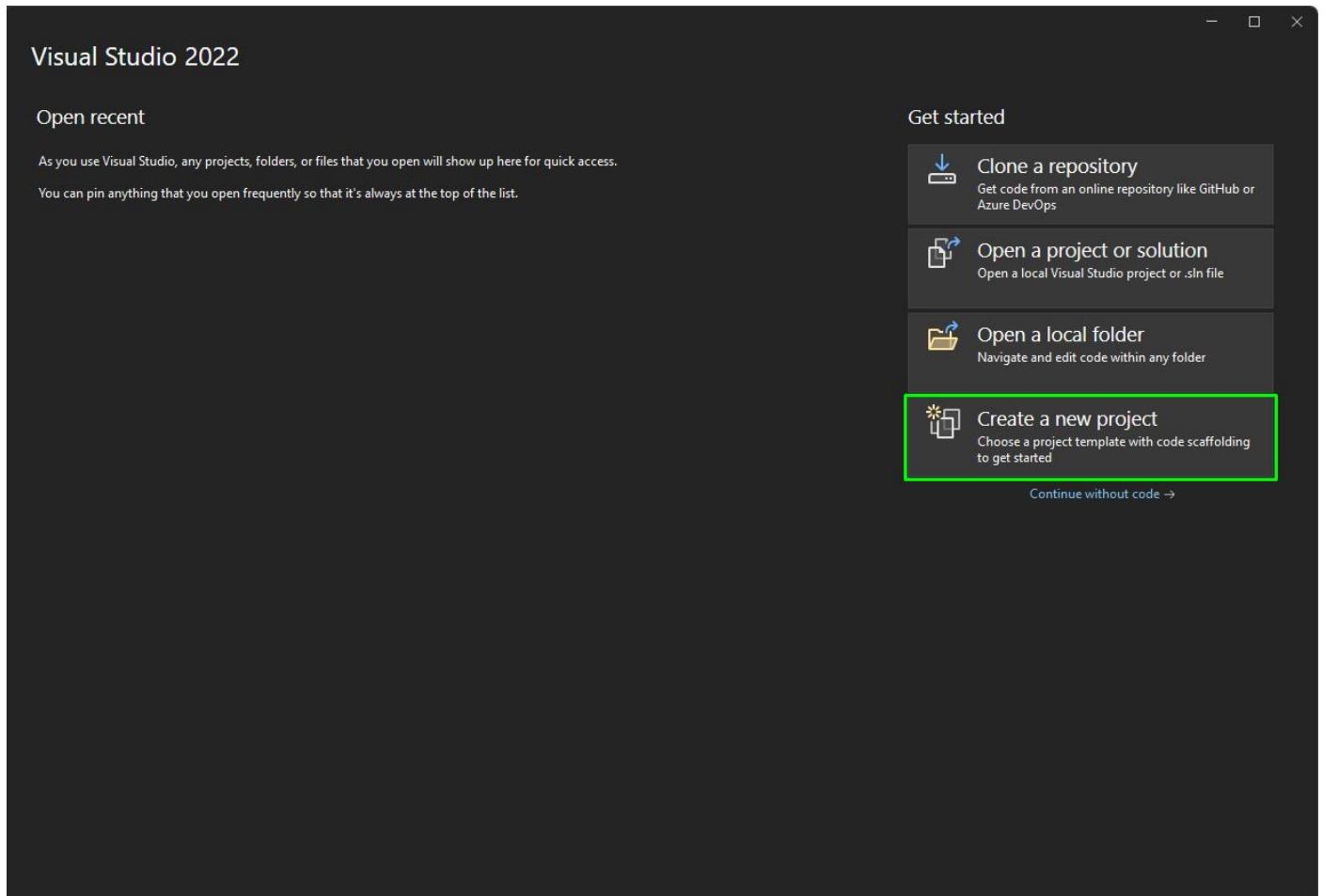
Runtime do ASP.NET Core 8.0.1

Sistema Operacional	Instaladores	Binários
Linux	Instruções do gerenciador de pacotes	Arm32 Arm32 Alpine Arm64 Arm64 Alpine x64 x64 Alpine
macOS		Arm64 x64
Windows	Hosting Bundle x64 x86 Instruções winget	Arm64 x64 x86

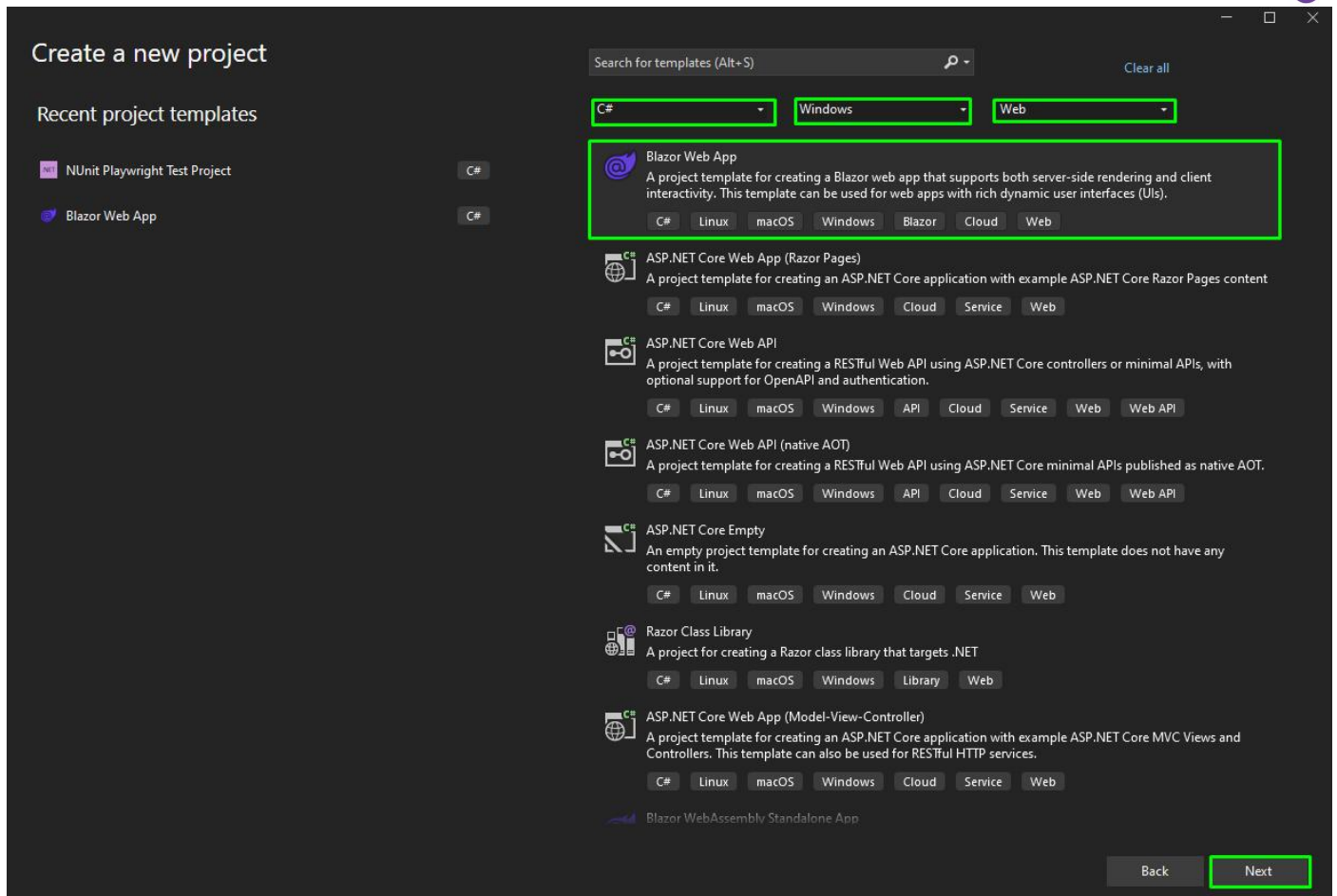
Feedback

Meu primeiro Projeto

Abra o Visual Studio Community e clique em Create a new Project (ou Criar um Novo Projeto, caso tenha optado pelo idioma Português BR)



A IDE do Visual Studio trabalha com diversas linguagens e diversos tipos de projetos. Para o nosso exemplo, vamos utilizar a linguagem C#, na plataforma Windows, projeto Web e a opção Blazor Server App, conforme figura abaixo.



Clique em Next e a próxima tela é referente ao nome do Projeto.

Configure your new project

Blazor Web App C# Linux macOS Windows Blazor Cloud Web

Project name
BlazorCRUD

Location
C:\BlazorProjects ...

Solution name ⓘ
BlazorCRUD

☐ Place solution and project in the same directory


Project will be created in "C:\BlazorProjects\BlazorCRUD\BlazorCRUD\"


Back Next


Clique em Next e configure conforme a imagem a baixo e então clique em Create para criarmos nosso primeiro projeto Blazor Server App.


Additional information


Blazor Web App C# Linux macOS Windows Blazor Cloud Web


Framework 
.NET 8.0 (Long Term Support)


Authentication type 
None

☒ Configure for HTTPS 

Interactive render mode 
Server

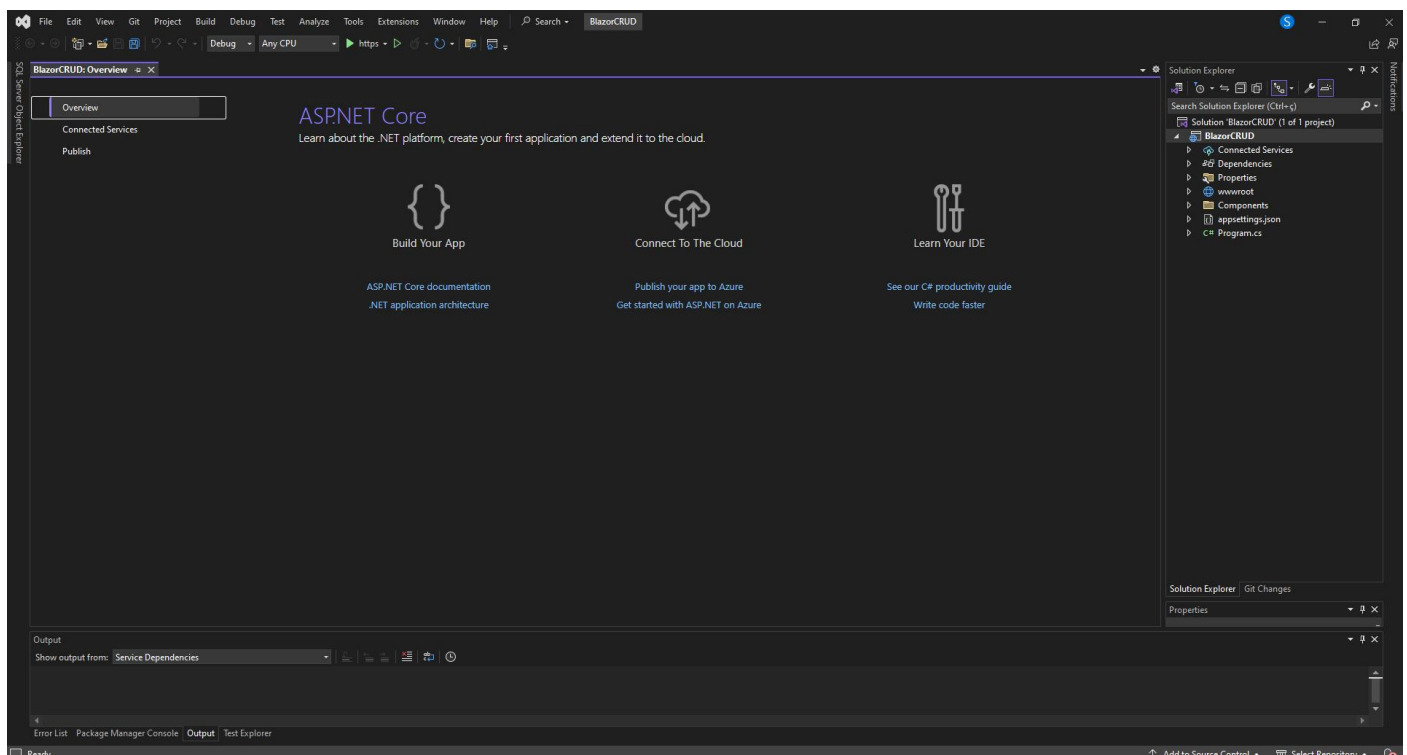
Interactivity location 
Global

☒ Include sample pages 

☐ Do not use top-level statements 

Back Create

Tela com o carregamento do projeto:



Entendendo o Aplicativo Web Blazor e sua estrutura

O modelo de projeto Blazor Web App fornece um único ponto de partida para usar componentes Razor para construir qualquer estilo de UI da web, renderizado no lado do servidor e renderizado no lado do cliente. Ele combina os pontos fortes dos modelos de hospedagem Blazor Server e Blazor WebAssembly existentes com renderização no lado do servidor, renderização de streaming, navegação aprimorada e manipulação de formulários e a capacidade de adicionar interatividade usando Blazor Server ou Blazor WebAssembly por componente.

Se a renderização do lado do cliente (CSR) e a renderização interativa do lado do servidor (SSR interativo) forem selecionadas na criação do aplicativo, o modelo de projeto usará o modo de renderização automática interativa. O modo de renderização automática inicialmente usa SSR interativo enquanto o pacote de aplicativos .NET e o tempo de execução são baixados para o navegador. Após a ativação do tempo de execução do .NET WebAssembly, a renderização muda para CSR.

Por padrão, o modelo Blazor Web App permite a renderização estática e interativa do lado do servidor usando um único projeto. Se você também ativar a renderização interativa do WebAssembly, o projeto incluirá um projeto cliente adicional (.Client) para seus componentes baseados em WebAssembly. A saída construída do projeto cliente é baixada para o navegador e executada no cliente. Quaisquer componentes que usam os modos de renderização Interactive WebAssembly ou Interactive Auto devem ser criados a partir do projeto do cliente.

Blazor Server:

- Pasta Components:
 - Pasta Layout: contém os seguintes componentes de layout e folhas de estilo
 - Componente MainLayout (): o componente de layout MainLayout.razor do aplicativo .
 - MainLayout.razor.css: folha de estilo do layout principal do aplicativo.
 - NavMenu componente (NavMenu.razor): Implementa a navegação na barra lateral. Inclui o NavLink componente (NavLink), que renderiza links de navegação para outros componentes do Razor. O componente NavLink indica ao usuário qual componente está sendo exibido no momento.

- NavMenu.razor.css: folha de estilo do menu de navegação do aplicativo.
 - Pasta Pages: contém os componentes Razor roteáveis do lado do servidor do aplicativo (.razor). A rota para cada página é especificada usando a @pagediretiva. O modelo inclui o seguinte:
 - Componente Counter (Counter.razor): Implementa a página Contador .
 - Componente Error (Error.razor): implementa a página de erro.
 - Componente Home (Home.razor): Implementa a página inicial .
 - Componente Weather (Weather.razor): Implementa a página de previsão do tempo .
 - _Imports.razor: inclui diretivas comuns do Razor para incluir nos componentes do aplicativo renderizado pelo servidor (.razor), como @usingdiretivas para namespaces.
 - Componente App (App.razor): o componente raiz do aplicativo com <head>marcação HTML, o Routescomponente e a <script>tag Blazor. O componente raiz é o primeiro componente que o aplicativo carrega.
 - Componente Routes (Routes.razor): configura o roteamento usando o componente Roteador. Para componentes interativos do lado do cliente, o componente Router intercepta a navegação do navegador e renderiza a página que corresponde ao endereço solicitado.
- Pasta Properties: contém a configuração do ambiente de desenvolvimento no arquivo launchSettings.json.
 - Pasta wwwroot: a pasta raiz da Web do projeto do servidor que contém os ativos estáticos públicos do aplicativo.
 - Arquivo Program.cs: o ponto de entrada do projeto do servidor que configura o host do aplicativo Web ASP.NET Core e contém a lógica de inicialização do aplicativo, incluindo registros de serviço, configuração, registro em log e pipeline de processamento de solicitação.

- Arquivo appsettings.json: Arquivos de configurações do aplicativo (appsettings.Development.json, appsettings.json): fornecem definições de configuração para o projeto do servidor.

Blazor WebAssembly (.Client):

- Pasta Pages: contém os componentes Razor roteáveis do lado do cliente do aplicativo (.razor). A rota para cada página é especificada usando a @pagediretiva. O modelo inclui o componente Counter (Counter.razor) que implementa a página Contador .
- Pasta wwwroot: a pasta raiz da Web do projeto do lado do cliente que contém os ativos estáticos públicos do aplicativo, incluindo arquivos de configurações do aplicativo (appsettings.Development.json, appsettings.json) que fornecem definições de configuração para o projeto do lado do cliente.
- _Imports.razor: inclui diretivas comuns do Razor para incluir nos componentes do aplicativo renderizados pelo WebAssembly (.razor), como @usingdiretivas para namespaces.
- Arquivo Program.cs: o ponto de entrada do projeto do lado do cliente que configura o host WebAssembly e contém a lógica de inicialização do projeto, incluindo registros de serviço, configuração, registro em log e pipeline de processamento de solicitação.

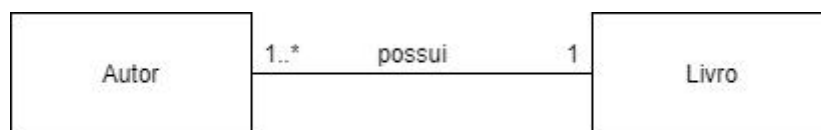
Criando um CRUD do zero

Vamos iniciar o nosso projeto e o tema será Autores e Livros. Para esse projeto vamos adotar a abordagem Code First e o nosso Banco de Dados será gerado automaticamente através do Entity Framework (mais a frente teremos um tópico abordando esse assunto).

Para simplificar o projeto e a técnica de relacionamento das entidades, vamos adotar a seguinte situação:

“Um autor poderá ter 1 ou mais livros e um livro, no nosso caso, deverá ter apenas 1 autor.”

A representação do Diagrama Entidade Relacionamento do Banco de Dados seria assim:

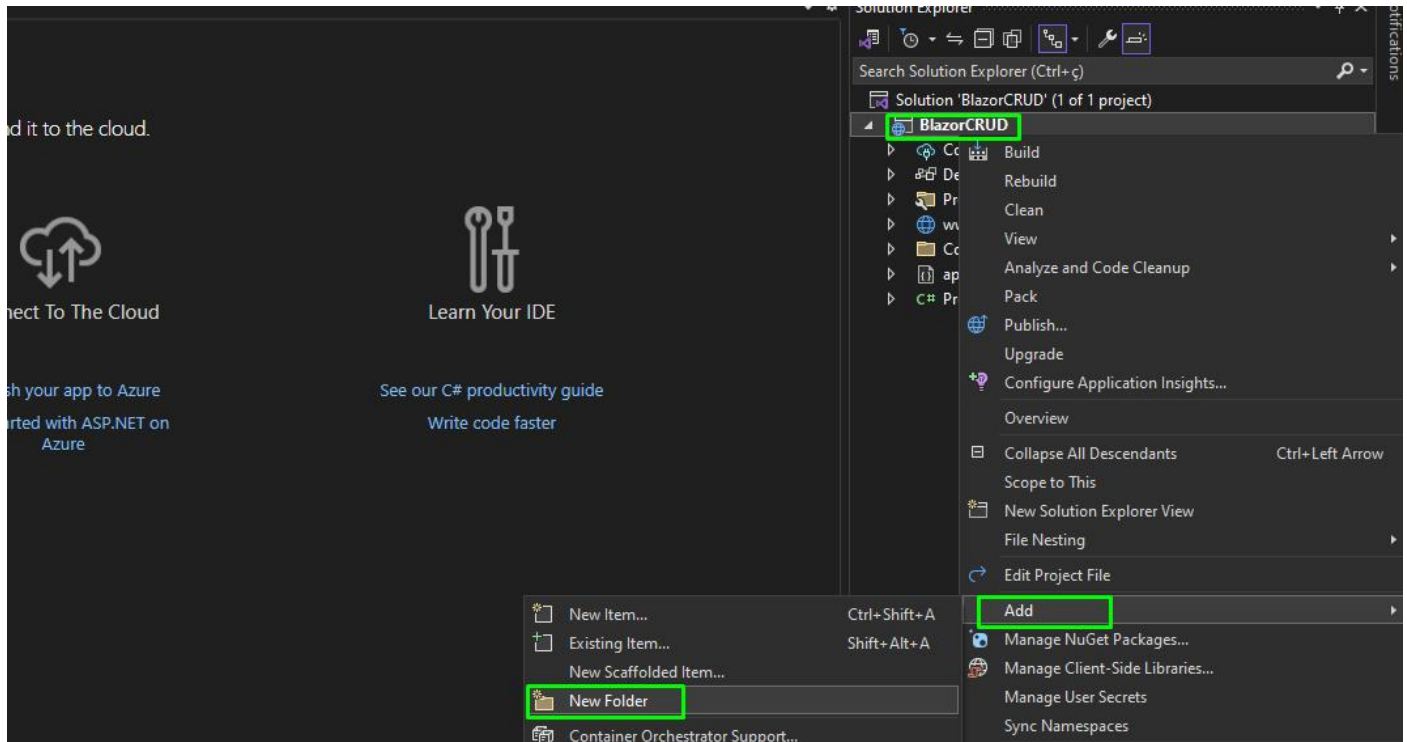


Com isso, ao cadastrar um Livro, já devemos informar quem é o autor dele. No caso do cadastro do Autor, não iremos relacionar os livros que o mesmo possui, pois nesse caso, na listagem dos dados, podemos mostrar juntos todos os livros e o autor de cada um.

Vamos iniciar a codificação:

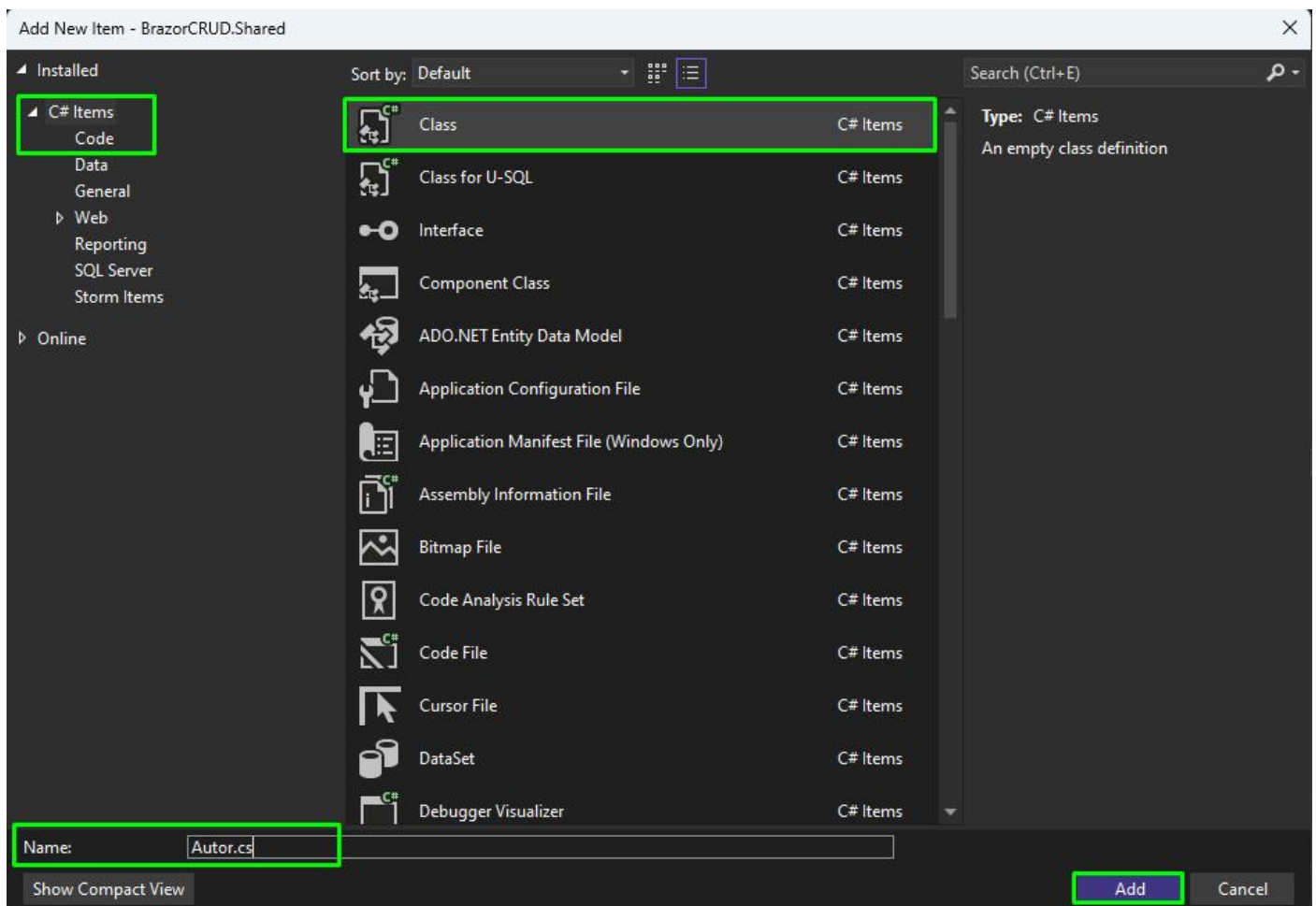
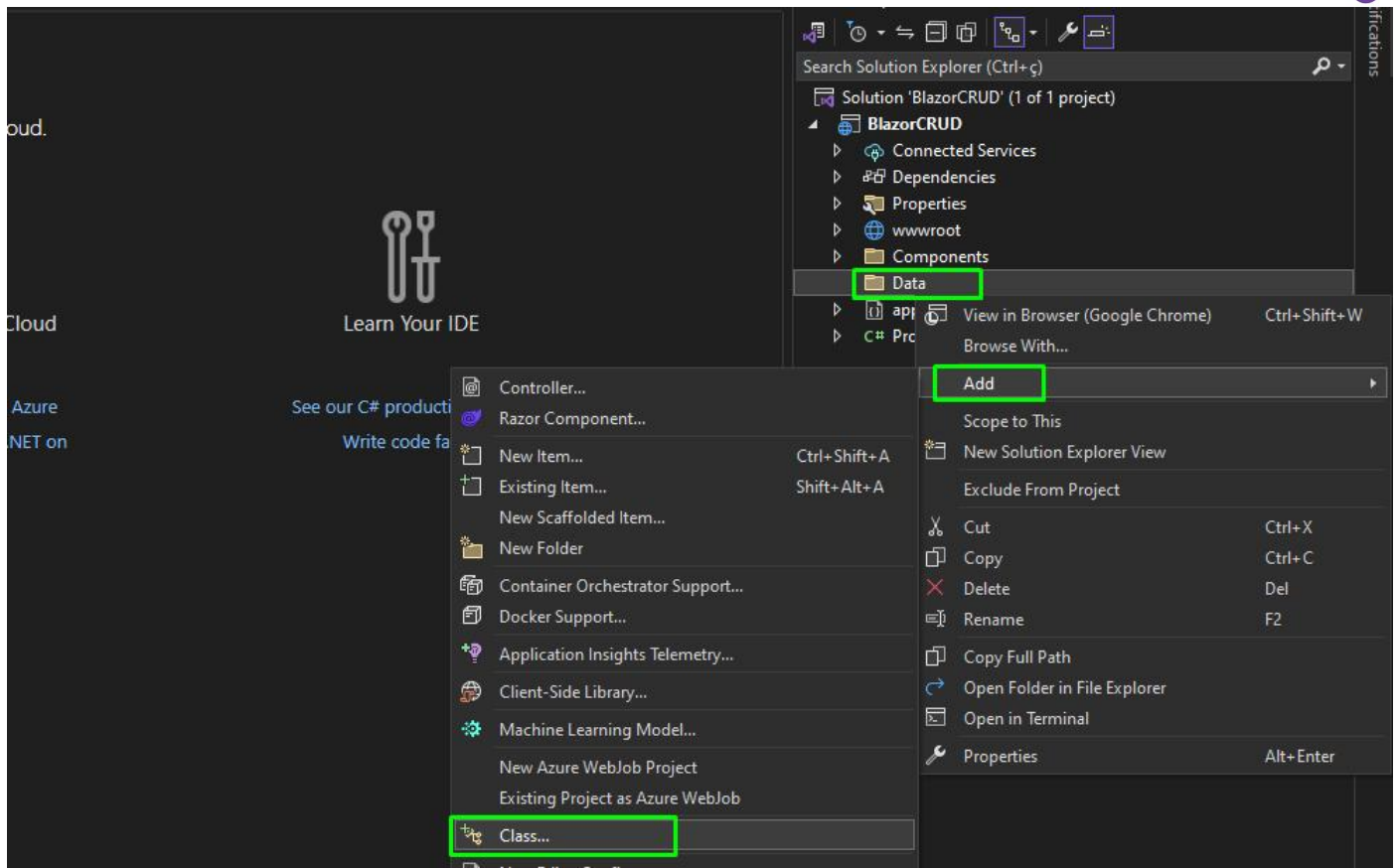
Em nosso projeto BlazorCRUD:

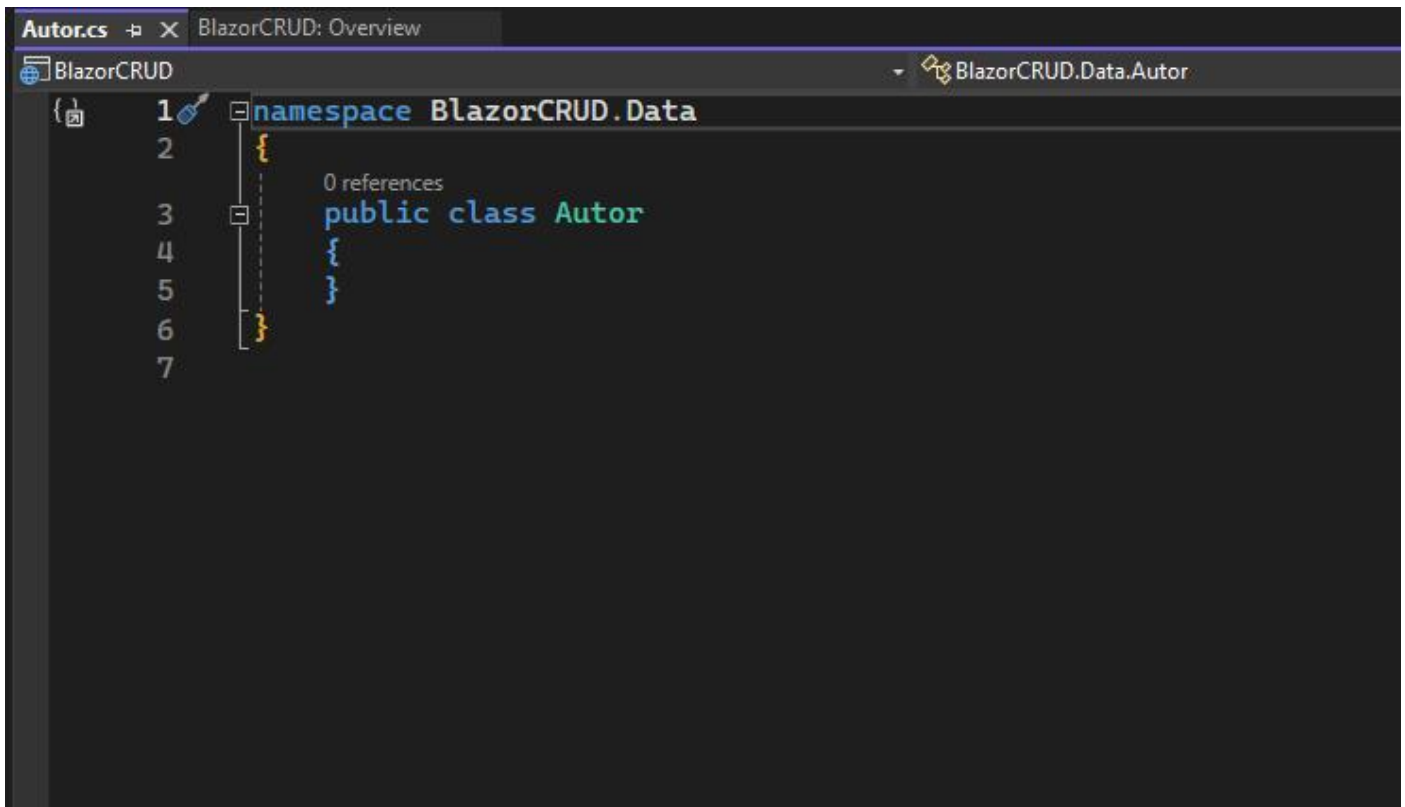
Clique com o botão direito em cima do projeto e escolha a opção Add -> New Folder e o coloque o nome desta pasta de "Data".



Agora Vamos iniciar com a criação da classe Autor.

Clique com o botão direito em cima da pasta "Data" e escolha a opção Add -> Class.





Vamos para os seguintes atributos da Classe Autor

```
using System.ComponentModel.DataAnnotations.Schema;

namespace BlazorCRUD.Data
{
    public class Autor
    {
        public int Id { get; set; }

        public string NomeCompleto { get; set; }

        public string Nacionalidade { get; set; }
    }
}
```

Próxima classe: Livro


```
using System.ComponentModel.DataAnnotations.Schema;

namespace BlazorCRUD.Data
{
    public class Livro
    {
        public int Id { get; set; }

        public string Nome { get; set; }

        public int AnoPublicacao { get; set; }

        [ForeignKey("Autor")]
        public int AutorId { get; set; }
    }
}
```

Observação: na classe Livro é criado uma relação com o Autor. Isso ocorre quando coloco um atributo dentro da classe e faço uma Annotation informando que o field é chave para outra classe. Isso será compreendido como um relacionamento. Para entendermos melhor como essa relação irá nos beneficiar na hora da geração do Banco de Dados, vamos entender melhor o que é Entity Framework e Mapeamento Objeto Relacional.

Entity Framework Core

O EF (Entity Framework) Core é uma versão leve, extensível, de [software livre](#) e multiplataforma da popular tecnologia de acesso a dados do Entity Framework.

EF Core pode servir como um mapeado relacional de objeto (O/RM), que:

- Permite que os desenvolvedores do .NET trabalhem com um banco de dados usando objetos .NET.
- Elimina a necessidade de maior parte do código de acesso a dados que normalmente precisa ser gravado.

O EF Core é compatível com vários mecanismos de banco de dados.

O modelo

Com o EF Core, o acesso a dados é executado usando um modelo. Um modelo é feito de classes de entidade e um objeto de contexto que representa uma sessão com o banco de dados. O objeto de contexto permite consultar e salvar dados.

O EF dá suporte às seguintes abordagens de desenvolvimento de modelo:

- Gere um modelo de um banco de dados existente.
- Codificar manualmente um modelo para corresponder ao banco de dados.
- Depois que um modelo for criado, use [Migrações do EF](#) para criar um banco de dados com base no modelo. As migrações permitem a evolução do banco de dados conforme o modelo muda.

Consultas

Instâncias de suas classes de entidade são recuperadas do banco de dados usando LINQ.

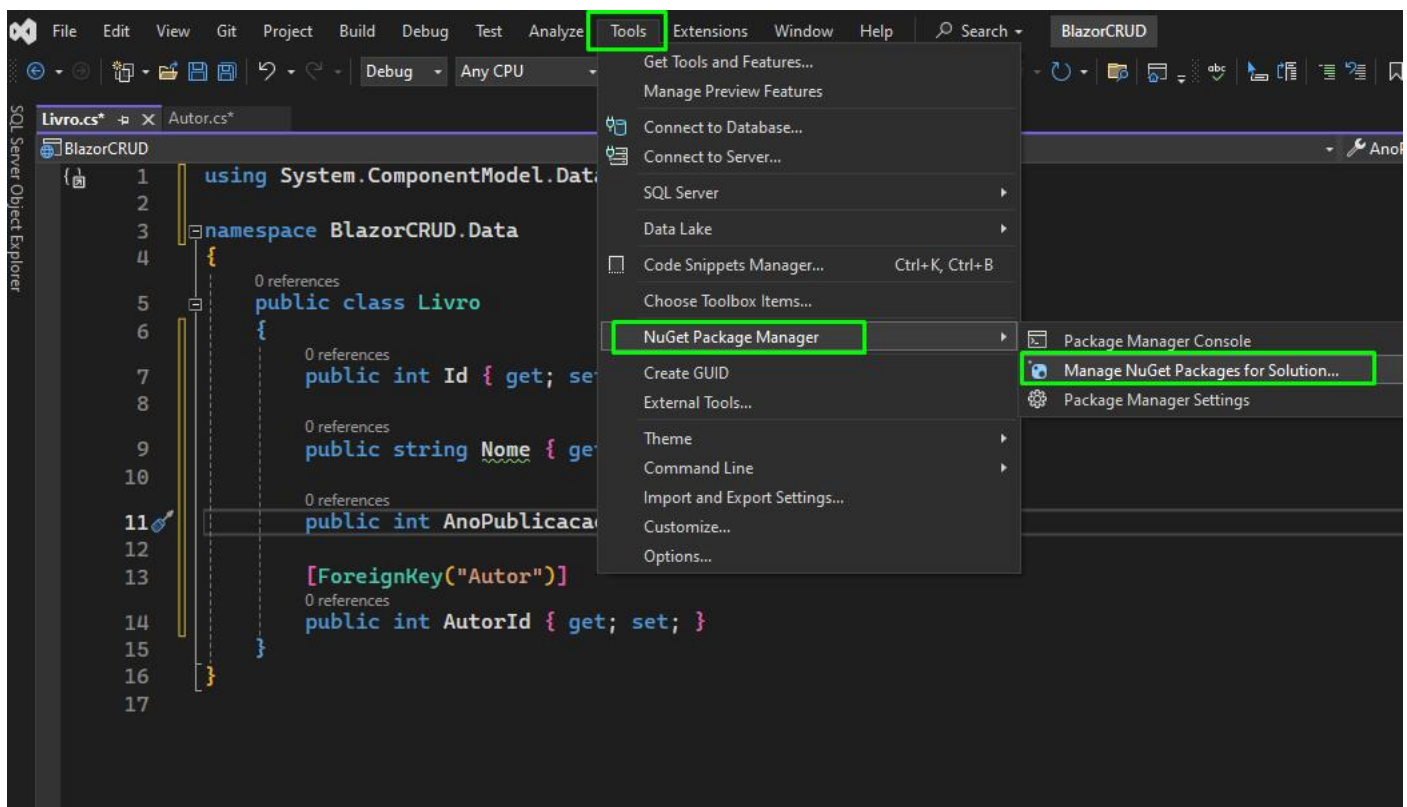
Salvando dados

Dados são criados, excluídos e modificados no banco de dados usando as instâncias de suas classes de entidade. Consulte [Salvar dados](#) para saber mais.

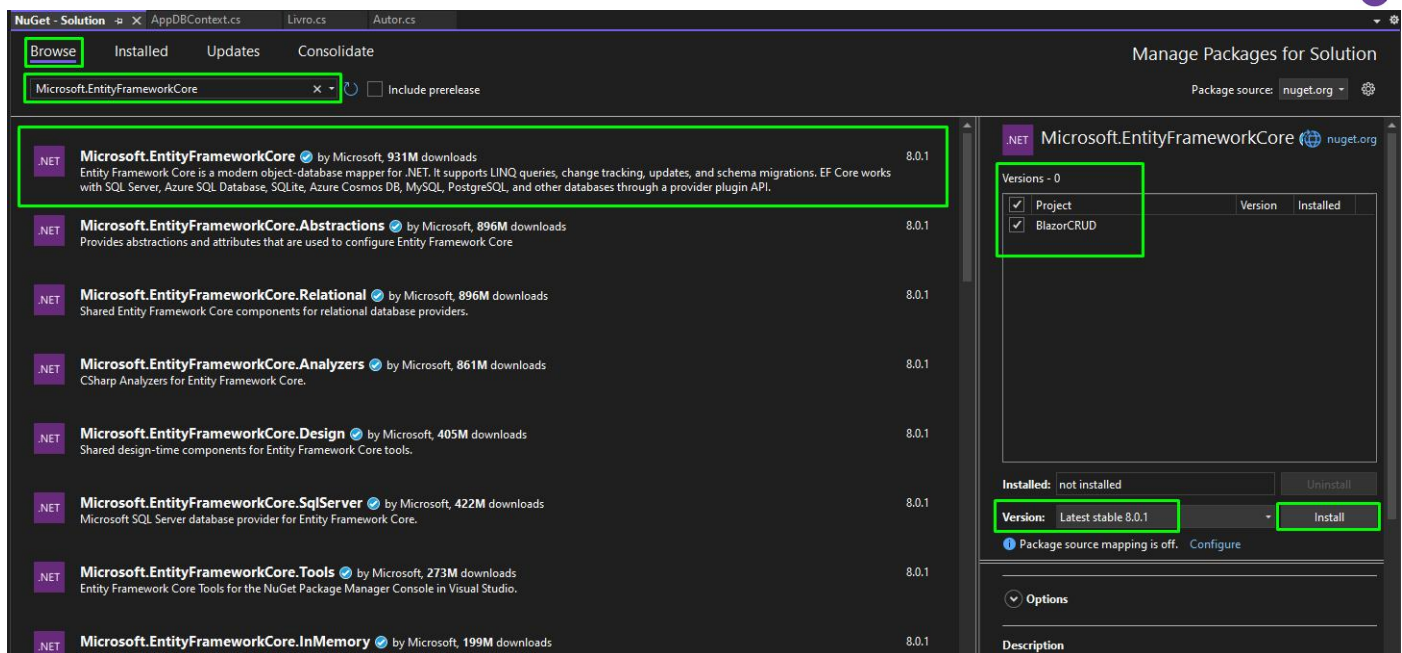
Continuando com o projeto Blazor CRUD

Para o funcionamento do Entity Framework é necessário a criação de uma classe que tenha como herança a DbContext. Para criar essa classe vamos antes instalar os pacotes/dependências necessárias para o funcionamento correto desse recurso. Caso queira maiores detalhes, acesse o link <https://docs.microsoft.com/pt-br/nuget/what-is-nuget> para entender mais sobre.

Acessar o Menu Tools -> NuGet Package Manager -> Manage NuGet Packages for Solution...

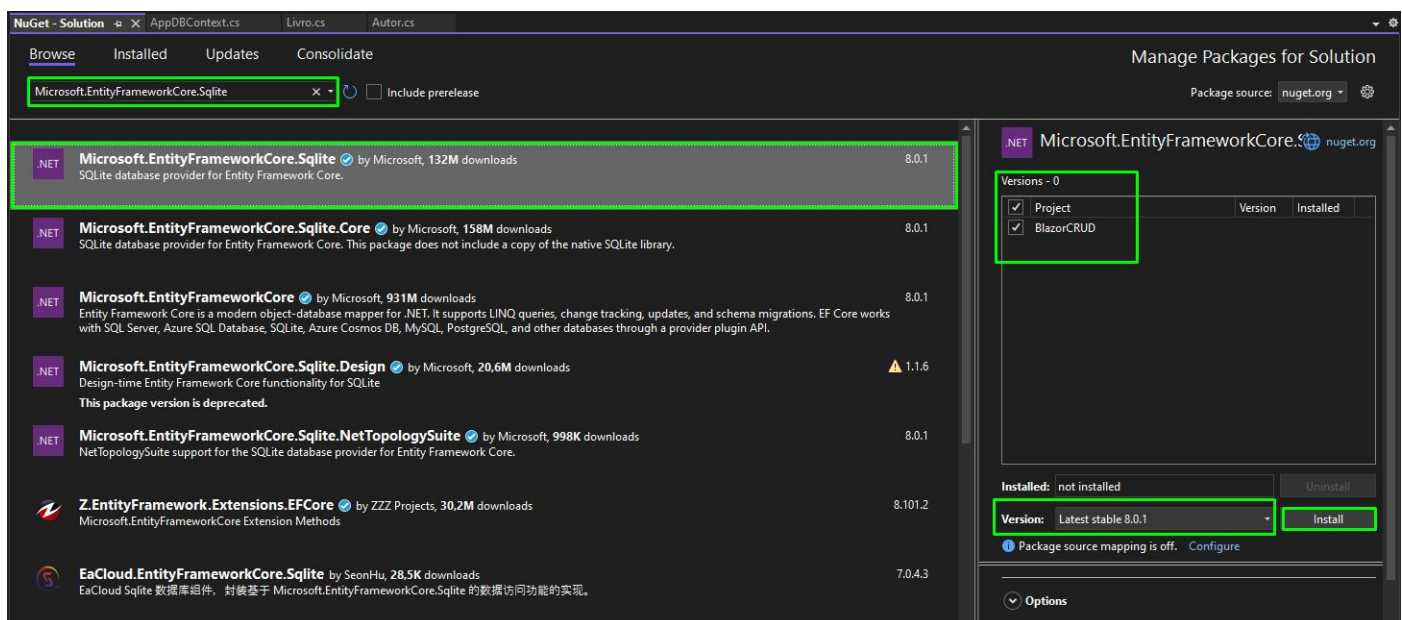


Será aberta uma nova guia. Clique em Browse e pesquise pelo pacote Microsoft.EntityFrameworkCore



Atenção: marque as opções acima e instale a mesma versão do .Net Core SDK feito no início deste documento para não ter problemas de incompatibilidade.

Volte na pesquisa e agora pesquise pelo pacote Microsoft.EntityFrameworkCore.Sqlite



Atenção: marque as opções acima e instale a mesma versão do .Net Core SDK feito no início deste documento para não ter problemas de incompatibilidade.

Próximo pacote: Microsoft.EntityFrameworkCore.Tools

The screenshot shows the NuGet Package Manager interface in Visual Studio. The 'Browse' tab is selected, and the search filter is set to 'Microsoft.EntityFrameworkCore.Tools'. The package list on the left shows several packages, with 'Microsoft.EntityFrameworkCore.Tools' (version 8.0.1) highlighted. The details pane on the right shows the package information, including the version '8.0.1' and the 'Install' button. The 'Project' and 'BlazorCRUD' checkboxes are also visible in the details pane.

Atenção: marque as opções acima e instale a mesma versão do .Net Core SDK feito no início deste documento para não ter problemas de incompatibilidade.

Na pasta Data, crie uma nova classe chamada “AppDbContext”. Abaixo segue o código fonte completo do arquivo .cs

```
using Microsoft.EntityFrameworkCore;

namespace BlazorCRUD.Data
{
    public class AppDbContext : DbContext
    {
        public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)
        {
        }
        public DbSet<Autor> Autores { get; set; }
        public DbSet<Livro> Livros { get; set; }
    }
}
```

Abra o arquivo Program.cs e vamos fazer o vínculo dessa classe AppDbContext no projeto. Coloque o fonte abaixo antes do comando “var app = builder.Build();”

```
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlite($"Data Source=blazorCrud.db"));
```

```
Program.cs* x AppDbContext.cs* Livro.cs Autor.cs
BlazorCRUD
1 using BlazorCRUD.Client.Pages;
2 using BlazorCRUD.Components;
3 using BlazorCRUD.Data;
4 using Microsoft.EntityFrameworkCore;
5
6 var builder = WebApplication.CreateBuilder(args);
7
8 // Add services to the container.
9 builder.Services.AddRazorComponents()
10 .AddInteractiveWebAssemblyComponents();
11
12 builder.Services.AddDbContext<AppDbContext>(options => options.UseSqlite($"Data Source=blazorCrud.db"));
13
14 var app = builder.Build();
15
16 // Configure the HTTP request pipeline.
17 if (app.Environment.IsDevelopment())
18 {
19     app.UseWebAssemblyDebugging();
20 }
21 else
22 {
23     app.UseExceptionHandler("/Error", createScopeForErrors: true);
24     // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts
25     app.UseHsts();
26 }
27
28 app.UseHttpsRedirection();
29
30 app.UseStaticFiles();
31 app.UseAntiforgery();
32
```


Visão geral das migrações

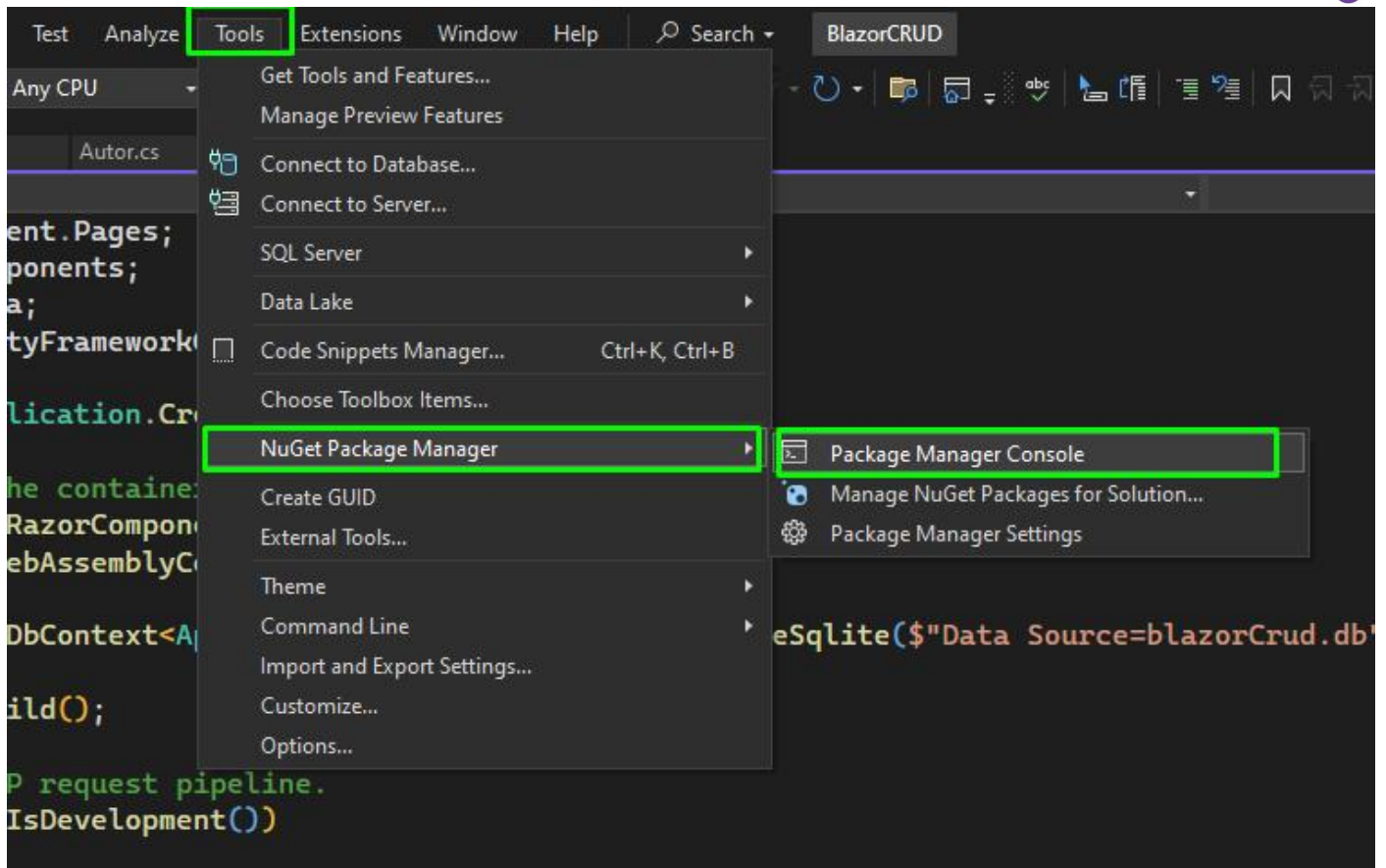
Em projetos do mundo real, os modelos de dados mudam conforme os recursos são implementados: novas entidades ou propriedades são adicionadas e removidas, e os esquemas de banco de dados precisam ser alterados de acordo para serem mantidos em sincronia com o aplicativo. O recurso de migrações no EF Core oferece uma maneira de atualizar de forma incremental o esquema de banco de dados para mantê-lo em sincronia com o modelo de dados do aplicativo, preservando os dados existentes no banco de dados.

Em um nível alto, as migrações funcionam da seguinte maneira:

- Quando uma alteração de modelo de dados é introduzida, o desenvolvedor usa as ferramentas do EF Core para adicionar uma migração correspondente que descreve as atualizações necessárias para manter o esquema de banco de dados em sincronia. O EF Core compara o modelo atual com um instantâneo do modelo antigo para determinar as diferenças e gerar os arquivos de origem da migração; os arquivos podem ser acompanhados no controle do código-fonte do projeto, como qualquer outro arquivo de origem.
- Depois que uma nova migração é gerada, é possível aplicá-la a um banco de dados de várias maneiras. O EF Core registra todas as migrações aplicadas em uma tabela de histórico especial, permitindo que ela saiba quais migrações foram ou não aplicadas.

Voltando ao projeto, agora iremos usar os recursos do mapeado relacional de objeto (O/RM) com o Entity Framework para gerar automaticamente nosso Banco de Dados.

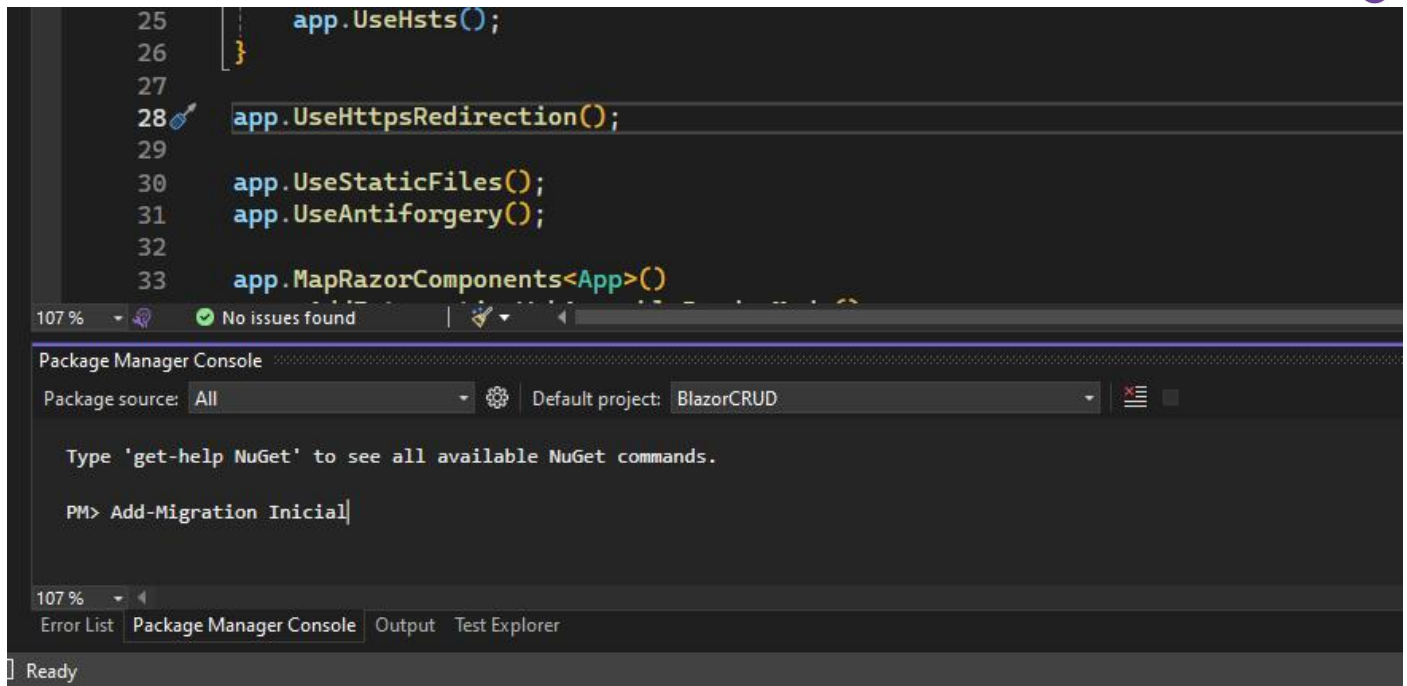
Na IDE do Visual Studio, clique no Menu Tools -> NuGet Package Manager -> Package Manager Console



Será destacada na IDE uma nova janela abaixo.

Nessa nova janela, vamos digitar o comando Add-Migration Inicial

*Inicial é apenas um nome. Serve para controle das migrações, que devem ser feitas toda vez que uma classe é criada ou uma existente sofre modificação.



```
25     app.UseHsts();
26 }
27
28 app.UseHttpsRedirection();
29
30 app.UseStaticFiles();
31 app.UseAntiforgery();
32
33 app.MapRazorComponents<App>()
```

107 % No issues found

Package Manager Console

Package source: All Default project: BlazorCRUD

Type 'get-help NuGet' to see all available NuGet commands.

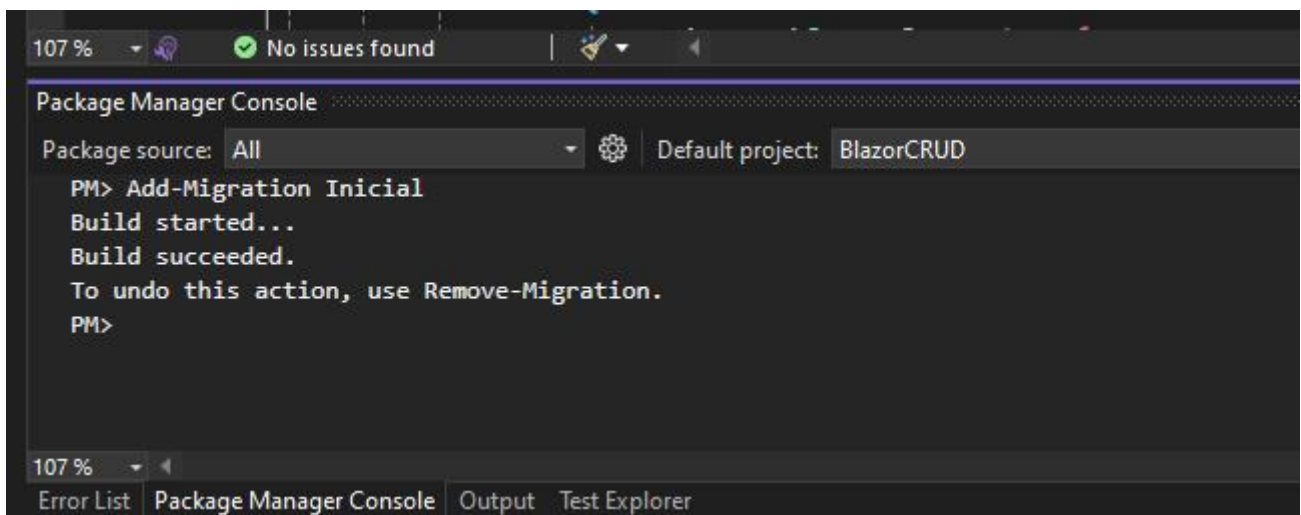
PM> Add-Migration Inicial

107 %

Error List Package Manager Console Output Test Explorer

Ready

Aperte Enter para confirmar o comando.



```
107 % No issues found
```

Package Manager Console

Package source: All Default project: BlazorCRUD

```
PM> Add-Migration Inicial
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM>
```

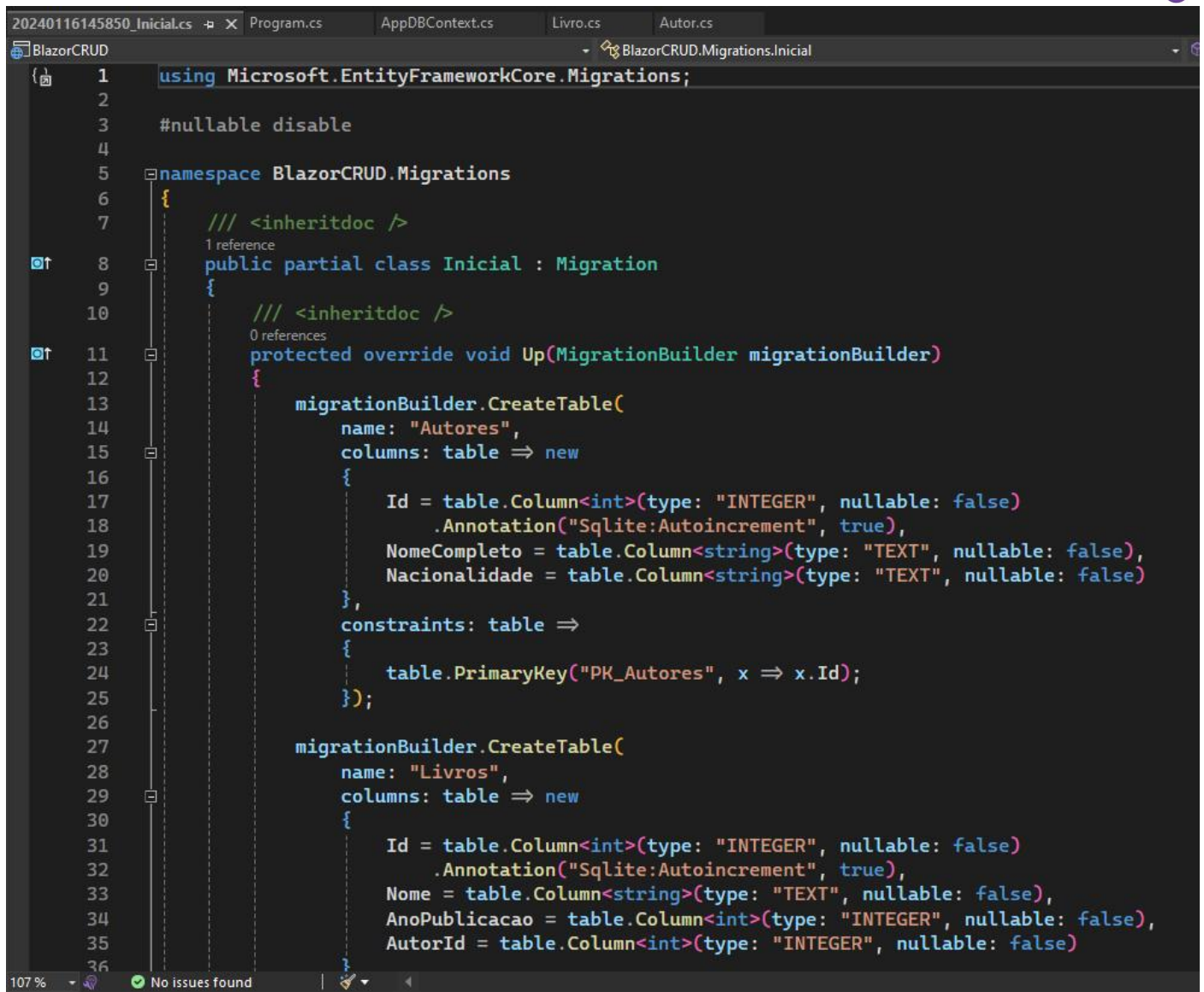
107 %

Error List Package Manager Console Output Test Explorer

Compreendendo o que foi feito:

O Comando add-migration irá ler a classe `AppDbContext` e buscar todos os modelos existentes no projeto. Automaticamente será feita a análise relacional desses modelos e os mesmos serão transformados em tabela.

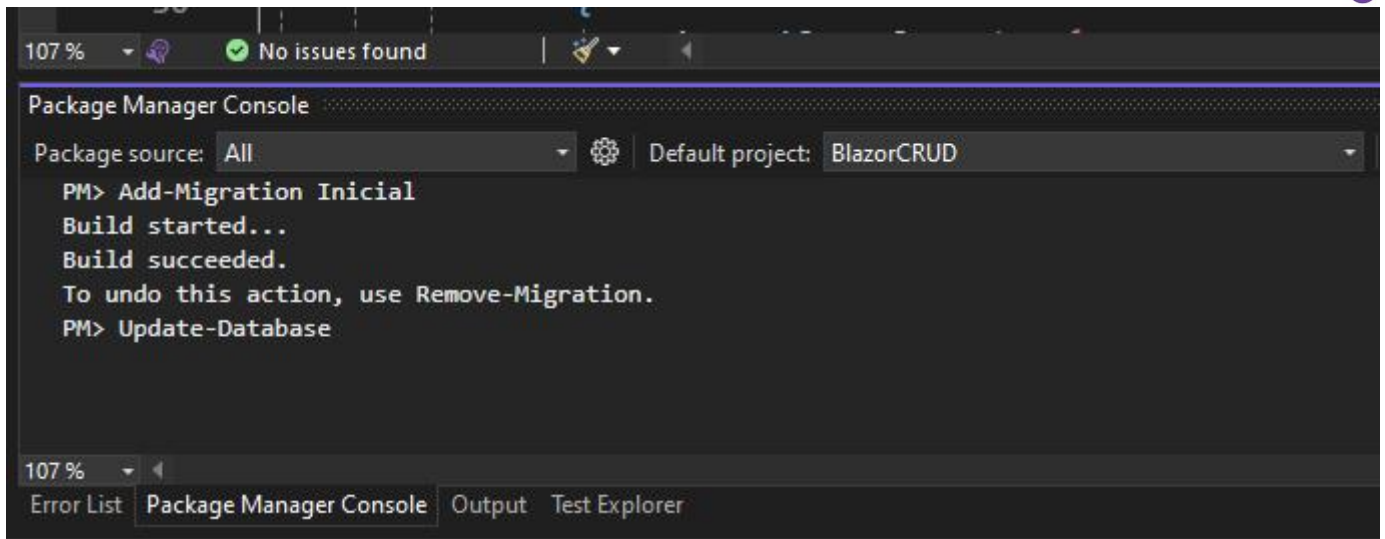
Repare que, após confirmar o comando com o enter, a IDE está mostrando um novo arquivo criado. Nele é descrito, em código C#, o que será feito em relação a criação das tabelas.



```
1 using Microsoft.EntityFrameworkCore.Migrations;
2
3 #nullable disable
4
5 namespace BlazorCRUD.Migrations
6 {
7     /// <inheritdoc />
8     public partial class Inicial : Migration
9     {
10         /// <inheritdoc />
11         protected override void Up(MigrationBuilder migrationBuilder)
12         {
13             migrationBuilder.CreateTable(
14                 name: "Autores",
15                 columns: table => new
16                 {
17                     Id = table.Column<int>(type: "INTEGER", nullable: false)
18                         .Annotation("Sqlite:Autoincrement", true),
19                     NomeCompleto = table.Column<string>(type: "TEXT", nullable: false),
20                     Nacionalidade = table.Column<string>(type: "TEXT", nullable: false)
21                 },
22                 constraints: table =>
23                 {
24                     table.PrimaryKey("PK_Autores", x => x.Id);
25                 });
26
27             migrationBuilder.CreateTable(
28                 name: "Livros",
29                 columns: table => new
30                 {
31                     Id = table.Column<int>(type: "INTEGER", nullable: false)
32                         .Annotation("Sqlite:Autoincrement", true),
33                     Nome = table.Column<string>(type: "TEXT", nullable: false),
34                     AnoPublicacao = table.Column<int>(type: "INTEGER", nullable: false),
35                     AutorId = table.Column<int>(type: "INTEGER", nullable: false)
36                 });
37         }
38     }
39 }
```

Esse código pode ser analisado e até ser modificado antes do próximo comando que irá aceitar e realizar os comandos para criar o Banco de Dados, as Tabelas, as Chaves Primárias, os Índices e todos os outros recursos.

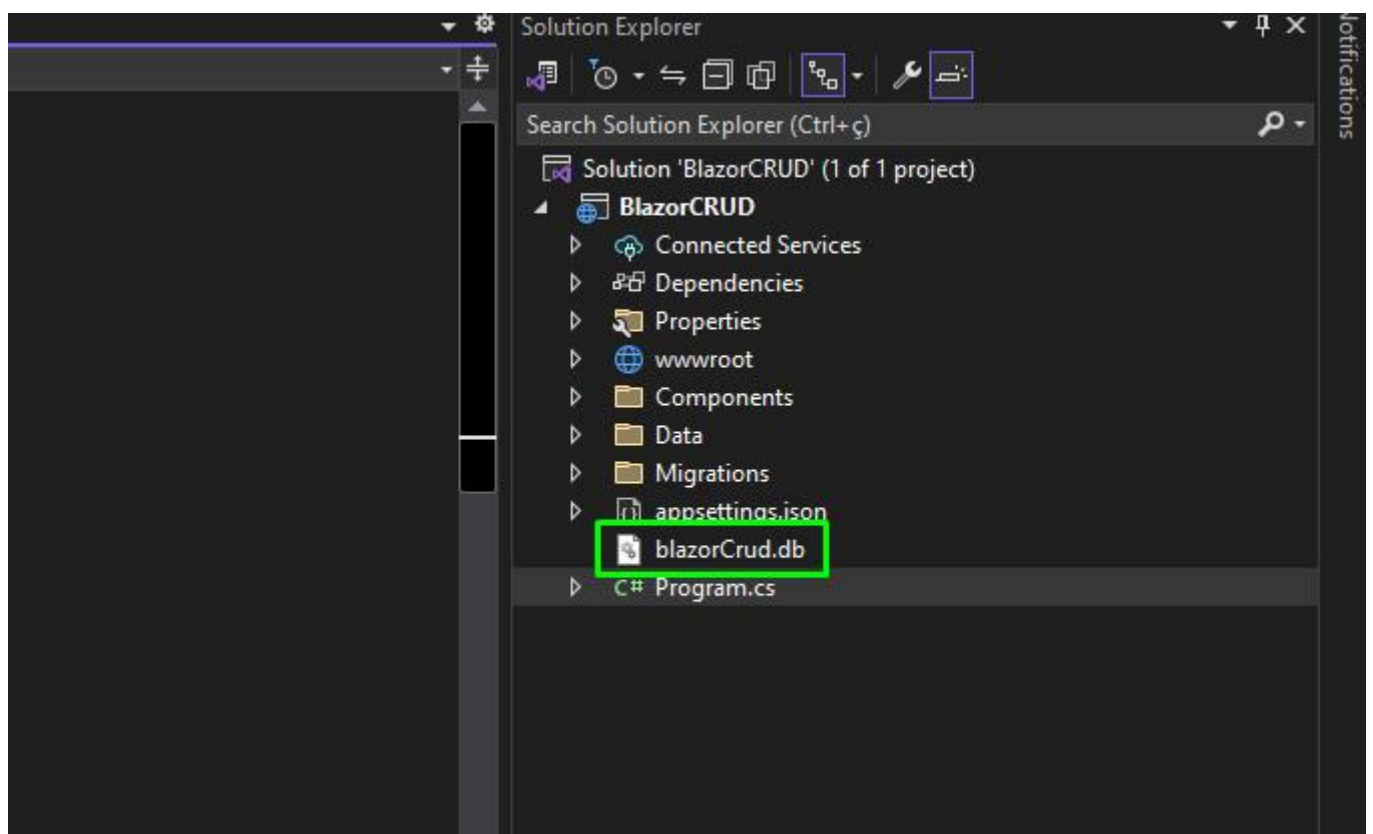
Novamente na janela do Package Manage Console, digite o comando Update-Database



Clique em Enter para confirmar o comando

```
PM> Update-Database
Build started...
Build succeeded.
Done.
PM>
```

Agora temos o arquivo .db do Sqlite criado em nosso projeto.



Observação: esse tutorial é didático, por isso estamos usando o Banco de Dados Sqlite, onde não há necessidade de instalação de um SGBD (Sistema Gerenciador de Banco de Dados). Para projetos internos ou comerciais, procure usar um Banco de Dados de alta performance para obter os melhores resultados.

O próximo passo é criar uma nova classe para fazer a manipulação dos dados no Banco. Novamente na pasta Data, crie uma nova classe chamada AutorService

```
using System.Collections.Generic;
using System.Linq;

namespace BlazorCRUD.Data
{
    public class AutorService
    {
        private readonly AppDBContext _db;

        public AutorService(AppDBContext db)
        {
            _db = db;
        }

        // Operações CRUD (Create, Read, Update e Delete)

        // Obter todos os autores
        public List<Autor> GetAutor()
        {
            var autorList = _db.Autores.ToList();

            return autorList;
        }

        // Inserir um Autor
        public string Create(Autor objAutor)
        {
            _db.Autores.Add(objAutor);
            _db.SaveChanges();
            return "Autor salvo com sucesso";
        }

        // Obter um Autor pelo Id
        public Autor GetAutorById(int id)
        {
            Autor objAutor = _db.Autores.FirstOrDefault(a => a.Id == id);
            return objAutor;
        }
    }
}
```

```
// Atualizar um Autor
public string UpdateAutor(Autor objAutor)
{
    _db.Autores.Update(objAutor);
    _db.SaveChanges();
    return "Autor atualizado com sucesso";
}

// Remover um Autor
public string DeleteAutor(Autor objAutor)
{
    _db.Autores.Remove(objAutor);
    _db.SaveChanges();
    return "Autor removido com sucesso";
}
}
```

Crie mais uma classe chamada LivroService

```
using System.Collections.Generic;
using System.Linq;

namespace BlazorCRUD.Data
{
    public class LivroService
    {
        private readonly AppDBContext _db;

        public LivroService(AppDBContext db)
        {
            _db = db;
        }

        // Operações CRUD (Create, Read, Update e Delete)

        // Obter todos os livros
        public List<Livro> GetLivro()
        {
            var livroList = _db.Livros.ToList();
        }
    }
}
```

```
        return livroList;
    }

    // Inserir um Livro
    public string Create (Livro objLivro)
    {
        _db.Livros.Add(objLivro);
        _db.SaveChanges();
        return "Livro salvo com sucesso";
    }

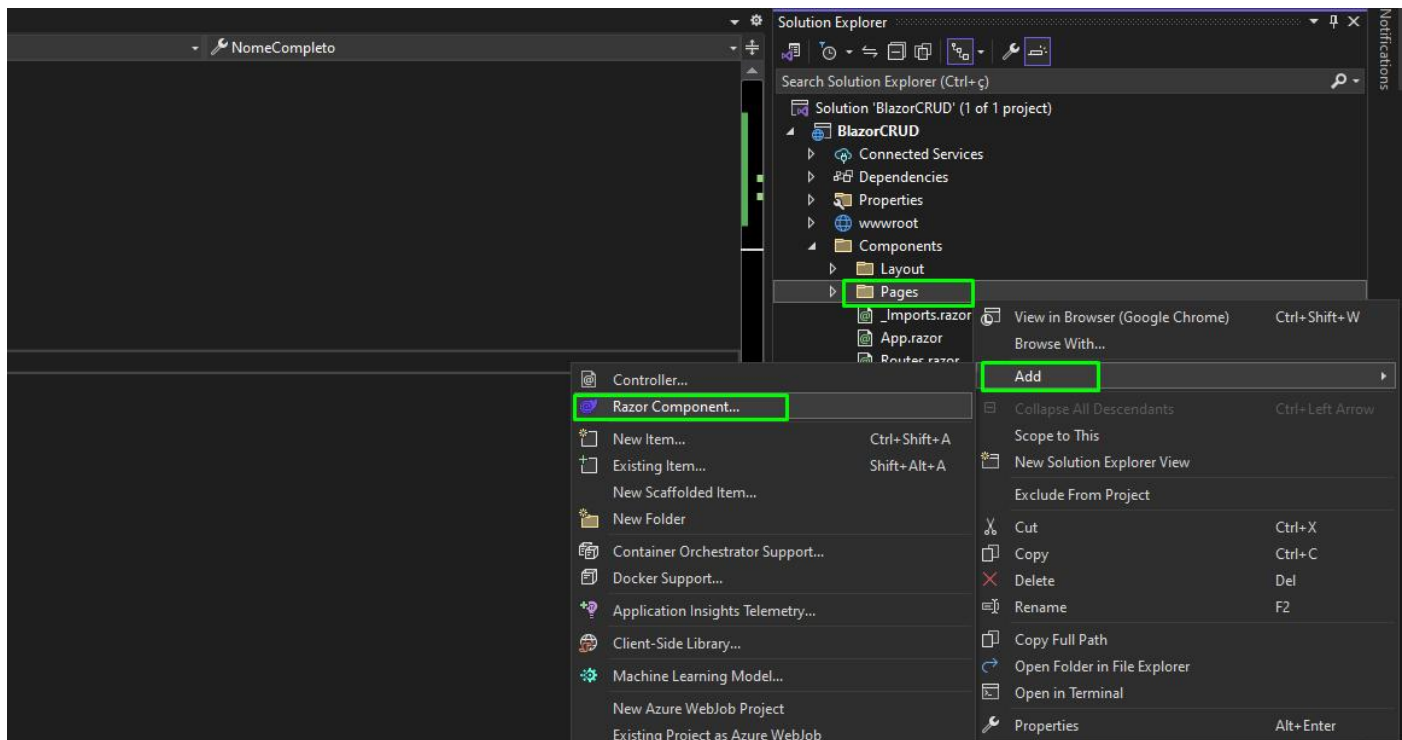
    // Obter um Livro pelo Id
    public Livro GetLivroById(int id)
    {
        Livro objLivro = _db.Livros.FirstOrDefault(l => l.Id == id);
        return objLivro;
    }

    // Atualizar um Livro
    public string UpdateLivro(Livro objLivro)
    {
        _db.Livros.Update(objLivro);
        _db.SaveChanges();
        return "Livro atualizado com sucesso";
    }

    // Remover um Livro
    public string Deletelivro(Livro objLivro)
    {
        _db.Livros.Remove(objLivro);
        _db.SaveChanges();
        return "Livro removido com sucesso";
    }
}
}
```

Com nossos modelos e serviços criados, podemos agora partir para o Front da aplicação. Vamos criar nossas páginas de interação com o usuário. No Solution Explorer Expanda a pasta Components, dentro dela temos a pasta "Pages", clique com

o botão direito nesta pasta "Pages" selecione Add → Razor Component e digite o nome AutorPage.razor



Abaixo segue o código fonte da página AutorPage.razor

```
@page "/autores"

@using BlazorCRUD.Data

@inject AutorService _objAutorService

<NavLink class="nav-link" href="AddAutor">
    <span class="oi oi-plus" aria-hidden="true"></span>Adicionar
</NavLink>

<h1>Informações dos Autores</h1>

@if (objAutores == null)
{
    <p><em>Carregando...</em></p>
}
else
{
    <table class="table">
```

```
<thead>
  <tr>
    <th>Id</th>
    <th>Nome</th>
    <th>Nacionalidade</th>
  </tr>
</thead>
<tbody>
  @foreach (var autor in objAutores)
  {
    <tr>
      <td>@autor.Id</td>
      <td>@autor.NomeCompleto</td>
      <td>@autor.Nacionalidade</td>
      <td>
        <a class="nav-link" href="EditAutor/@autor.Id">
          <span class="oi oi-pencil" aria-hidden="true"></span> Editar
        </a>
        <a class="nav-link" href="DeleteAutor/@autor.Id">
          <span class="oi oi-trash" aria-hidden="true"></span> Deletar
        </a>
      </td>
    </tr>
  }
</tbody>
</table>
}

@code {
    List<Autor> objAutores;

    protected override async Task OnInitializedAsync()
    {
        objAutores = await Task.Run(() => _objAutorService.GetAutor());
    }
}
```

Novamente, na pasta Pages, clique em Add – Razor Component e digite o nome AddAutorPage.razor

```
@page "/AddAutor"
```

```
@using BlazorCRUD.Data
```

```
@inject AutorService _objAutorService
```

```
@inject NavigationManager _objNavigationManager
```

```
<h2>Adicionar Autor</h2>
```

```
<hr />
```

```
<form>
```

```
    <div class="row">
```

```
        <div class="col-md-8">
```

```
            <div class="form-group">
```

```
                <label for="NomeCompleto" class="control-label">Nome</label>
```

```
                <input form="NomeCompleto" class="form-control"
```

```
@bind="@objAutor.NomeCompleto" />
```

```
            </div>
```

```
        <div class="form-group">
```

```
            <label for="Nacionalidade" class="control-label"></label>
```

```
            <select @bind="@objAutor.Nacionalidade" class="form-control">
```

```
                <option value="">--Selecione uma Nacionalidade--</option>
```

```
                <option value="Brasileiro">Brasileiro</option>
```

```
                <option value="Alemã">Alemã</option>
```

```
                <option value="Britânico">Britânico</option>
```

```
                <option value="Norte-Americano">Norte-Americano</option>
```

```
                <option value="Russo">Russo</option>
```

```
                <option value="Outra">Outra</option>
```

```
            </select>
```

```
        </div>
```

```
    </div>
```

```
</div>
```

```
    <div class="row">
```

```
        <div class="col-md-4">
```

```
            <div class="form-group">
```

```
                <input type="button" class="btn btn-primary" @onclick="@CreateAutor"
```

```
value="Salvar" />
```

```
                <input type="button" class="btn btn-primary" @onclick="@Cancel"
```

```
value="Cancelar" />
```

```
</div>
</div>
</div>
</form>

@code {
    Autor objAutor = new Autor();

    protected void CreateAutor()
    {
        _objAuthService.Create(objAutor);
        _objNavigationManager.NavigateTo("autores");
    }

    void Cancel()
    {
        _objNavigationManager.NavigateTo("autores");
    }
}
```

Novamente, na pasta Pages, clique em Add – Razor Component e digite o nome EditAutorPage.razor

```
@page "/EditAutor/{CurrentID}"

@using BlazorCRUD.Data

@inject AuthService _objAuthService
@inject NavigationManager _objNavigationManager

<h2>Editar Autor</h2>
<hr />

<form>
    <div class="row">
        <div class="col-md-8">
            <div class="form-group">
                <input form="Id" disabled class="form-control" @bind="@objAutor.Id" />
            </div>
        </div>
    </div>
</form>
```

```

<div class="form-group">
  <label for="NomeCompleto" class="control-label">Nome</label>
  <input form="NomeCompleto" class="form-control"
@bind="@objAutor.NomeCompleto" />
</div>

<div class="form-group">
  <label for="Nacionalidade" class="control-label"></label>
  <select @bind="@objAutor.Nacionalidade" class="form-control">
    <option value="">--Selecione uma Nacionalidade--</option>
    <option value="Brasileiro">Brasileiro</option>
    <option value="Alemã">Alemã</option>
    <option value="Britânico">Britânico</option>
    <option value="Norte-Americano">Norte-Americano</option>
    <option value="Russo">Russo</option>
    <option value="Outra">Outra</option>
  </select>
</div>
</div>
<div class="row">
  <div class="col-md-4">
    <div class="form-group">
      <input type="button" class="btn btn-primary" @onclick="@UpdateAutor"
value="Atualizar" />
      <input type="button" class="btn btn-primary" @onclick="@Cancel"
value="Cancelar" />
    </div>
  </div>
</div>
</form>

@code {
  [Parameter] public string CurrentId { get; set; }

  Autor objAutor = new Autor();

  protected override async Task OnInitializedAsync()
  {

```

```
objAutor = await Task.Run(() =>
_objAutorService.GetAutorById(Convert.ToInt32(CurrentId) ));
}

protected void UpdateAutor()
{
    _objAutorService.UpdateAutor(objAutor);
    _objNavigationManager.NavigateTo("autores");
}

void Cancel()
{
    _objNavigationManager.NavigateTo("autores");
}
}
```

Novamente, na pasta Pages, clique em Add – Razor Component e digite o nome DeleteAutorPage.razor

```
@page "/DeleteAutor/{CurrentID}"

@using BlazorCRUD.Data

@inject AutorService _objAutorService
@inject NavigationManager _objNavigationManager

<h2>Apagar Autor</h2>
<hr />
<h3>Tem certeza que deseja apagar esse registro?</h3>

<div class="row">
    <div class="col-md-4">
        <div class="form-group">
            <label>Id do Autor: </label>
            <label>@objAutor.Id</label>
        </div>
        <div class="form-group">
```

```
<label>Nome: </label>
<label>@objAutor.NomeCompleto</label>
</div>
</div>
</div>
<div class="row">
  <div class="col-md-4">
    <div class="form-group">
      <input type="button" class="btn btn-danger" @onclick="@DeleteAutor"
value="Apagar" />
      <input type="button" class="btn btn-primary" @onclick="@Cancel"
value="Cancelar" />
    </div>
  </div>
</div>
```

```
@code {
  [Parameter] public string CurrentId { get; set; }

  Autor objAutor = new Autor();

  protected override async Task OnInitializedAsync()
  {
    objAutor = await Task.Run(() =>
      _objAutorService.GetAutorById(Convert.ToInt32(CurrentId)));
  }

  protected void DeleteAutor()
  {
    _objAutorService.DeleteAutor(objAutor);
    _objNavigationManager.NavigateTo("autores");
  }

  void Cancel()
  {
    _objNavigationManager.NavigateTo("autores");
  }
}
```



```

        <td>
            <a class="nav-link" href="EditLivro/@livro.Id">
                <span class="oi oi-pencil" aria-hidden="true"></span>Editar
            </a>
            <a class="nav-link" href="DeleteLivro/@livro.Id">
                <span class="oi oi-trash" aria-hidden="true"></span>Deletar
            </a>
        </td>
    </tr>
}
</tbody>
</table>
}

```

```

@code {
    List<Livro> objLivros;

    protected override async Task OnInitializedAsync()
    {
        objLivros = await Task.Run(() => _objLivroService.GetLivro());
    }
}

```

Novamente, na pasta Pages, clique em Add – Razor Component e digite o nome AddLivroPage.razor

```

@page "/AddLivro"

@using BlazorCRUD.Data

@inject AutorService _objAutorService
@inject LivroService _objLivroService
@inject NavigationManager _objNavigationManager

<h2>Adicionar Livro</h2>
<hr />

@if (objAutoresList == null)
{
    <p><em>Carregando...</em></p>
}

```

```
}  
else  
{  
    <form>  
        <div class="row">  
            <div class="col-md-8">  
                <div class="form-group">  
                    <label for="Nome" class="control-label">Nome</label>  
                    <input form="Nome" class="form-control" @bind="@objLivro.Nome" />  
                </div>  
  
                <div class="form-group">  
                    <label for="AnoPublicacao" class="control-label">Ano Publicação</label>  
                    <input form="AnoPublicacao" class="form-control"  
@bind="@objLivro.AnoPublicacao" />  
                </div>  
  
                <div class="form-group">  
                    <label for="AutorId" class="control-label">Autor</label>  
                    <select required @bind="@objLivro.AutorId" class="form-control">  
                        @foreach (var autor in objAutoresList)  
                        {  
                            <option value="@autor.Id">@autor.NomeCompleto</option>  
                        }  
                    </select>  
                </div>  
            </div>  
            <div class="row">  
                <div class="col-md-4">  
                    <div class="form-group">  
                        <input type="button" class="btn btn-primary" @onclick="@CreateLivro"  
value="Salvar" />  
                        <input type="button" class="btn btn-primary" @onclick="@Cancel"  
value="Cancelar" />  
                    </div>  
                </div>  
            </div>  
        </form>  
    }  
}
```

```
@code {  
    Livro objLivro = new Livro();  
    List<Autor> objAutoresList;  
  
    protected override async Task OnInitializedAsync()  
    {  
        objAutoresList = await Task.Run(() => _objAutorService.GetAutor());  
    }  
  
    protected void CreateLivro()  
    {  
        _objLivroService.Create(objLivro);  
        _objNavigationManager.NavigateTo("livros");  
    }  
  
    void Cancel()  
    {  
        _objNavigationManager.NavigateTo("livros");  
    }  
}
```

Observações: Na questão do Livro, onde eu preciso indicar o Autor, foi criado uma lista de autores trazida do Banco de Dados e disponibilizados no componente html Select para o usuário poder escolher qual autor é o correspondente. Com esse recurso evitamos o mesmo de precisar digitar o ID ou o Nome do Autor.

Novamente, na pasta Pages, clique em Add – Razor Component e digite o nome EditLivroPage.razor

```
@page "/EditLivro/{CurrentId}"  
  
@using BlazorCRUD.Data  
  
@inject AutorService _objAutorService  
@inject LivroService _objLivroService  
@inject NavigationManager _objNavigationManager  
  
<h2>Atualizar Livro</h2>  
<hr />
```

```
@if (objAutoresList == null)
{
    <p><em>Carregando...</em></p>
}
else
{
    <form>
        <div class="row">
            <div class="col-md-8">
                <div class="form-group">
                    <input form="Id" disabled class="form-control" @bind="@objLivro.Id" />
                </div>

                <div class="form-group">
                    <label for="Nome" class="control-label">Nome</label>
                    <input form="Nome" class="form-control" @bind="@objLivro.Nome" />
                </div>

                <div class="form-group">
                    <label for="AnoPublicacao" class="control-label">Ano Publicação</label>
                    <input form="AnoPublicacao" class="form-control"
@bind="@objLivro.AnoPublicacao" />
                </div>

                <div class="form-group">
                    <label for="AutorId" class="control-label">Autor</label>
                    <select required @bind="@objLivro.AutorId" class="form-control">
                        @foreach (var autor in objAutoresList)
                        {
                            <option value="@autor.Id">@autor.NomeCompleto</option>
                        }
                    </select>
                </div>
            </div>
            <div class="col-md-4">
                <div class="form-group">
                    <input type="button" class="btn btn-primary" @onclick="@UpdateLivro"
value="Atualizar" />
                </div>
            </div>
        </div>
    </form>
}
```

```
        <input type="button" class="btn btn-primary" @onclick="@Cancel"
value="Cancelar" />
    </div>
</div>
</div>
</form>
}
```

```
@code {
    [Parameter] public string CurrentId { get; set; }

    Livro objLivro = new Livro();
    List<Autor> objAutoresList;

    protected override async Task OnInitializedAsync()
    {
        objLivro = await Task.Run(() =>
        _objLivroService.GetLivroById(Convert.ToInt32(CurrentId)));
        objAutoresList = await Task.Run(() => _objAutorService.GetAutor());
    }

    protected void UpdateLivro()
    {
        _objLivroService.UpdateLivro(objLivro);
        _objNavigationManager.NavigateTo("livros");
    }

    void Cancel()
    {
        _objNavigationManager.NavigateTo("livros");
    }
}
```

Novamente, na pasta Pages, clique em Add – Razor Component e digite o nome DeleteLivroPage.razor

```
@page "/DeleteLivro/{CurrentID}"
```

```
@using BlazorCRUD.Data
```

```
@inject LivroService _objLivroService
@inject NavigationManager _objNavigationManager

<h2>Apagar Livro</h2>
<hr />

<h3>Tem certeza que deseja apagar esse registro?</h3>

<div class="row">
  <div class="col-md-8">
    <div class="form-group">
      <label>Id do Livro: </label>
      <label>@objLivro.Id</label>
    </div>
    <div class="form-group">
      <label>Nome: </label>
      <label>@objLivro.Nome</label>
    </div>
    <div class="form-group">
      <label>Ano publicação: </label>
      <label>@objLivro.AnoPublicacao</label>
    </div>
    <div class="form-group">
      <label>Nome: </label>
      <label>@objLivro.AutorId</label>
    </div>
  </div>
</div>
<div class="row">
  <div class="col-md-4">
    <div class="form-group">
      <input type="button" class="btn btn-danger" @onclick="@DeleteLivro"
value="Apagar" />
      <input type="button" class="btn btn-primary" @onclick="@Cancel"
value="Cancelar" />
    </div>
  </div>
</div>

@code {
```

```
[Parameter] public string CurrentId { get; set; }

Livro objLivro = new();

protected override async Task OnInitializedAsync()
{
    objLivro = await Task.Run(() =>
_objLivroService.GetLivroById(Convert.ToInt32(CurrentId)));
}

protected void DeleteLivro()
{
    _objLivroService.DeleteLivro(objLivro);
    _objNavigationManager.NavigateTo("livros");
}

void Cancel()
{
    _objNavigationManager.NavigateTo("livros");
}
}
```

O próximo passo é dizer ao Blazor que ele deve iniciar nossas novas classes de serviços que criamos anteriormente. Para isso, localize no Solution Explorer o arquivo Program.cs (é para ser o último arquivo), abra-o para edição e em cima do comando que adicionamos anteriormente adicione as seguintes linhas:

```
builder.Services.AddScoped<AutorService>();
builder.Services.AddScoped<LivroService>();
```

```

1  using BlazorCRUD.Components;
2  using BlazorCRUD.Data;
3  using Microsoft.EntityFrameworkCore;
4
5  var builder = WebApplication.CreateBuilder(args);
6
7  // Add services to the container.
8  builder.Services.AddRazorComponents()
9      .AddInteractiveServerComponents();
10
11  builder.Services.AddScoped<AutorService>();
12  builder.Services.AddScoped<LivroService>();
13
14  builder.Services.AddDbContext<AppDbContext>(options => options.UseSqlite($"Data Source=blazorCrud.db"));
15
16  var app = builder.Build();
17
18  // Configure the HTTP request pipeline.
19  if (!app.Environment.IsDevelopment())
20  {
21      app.UseExceptionHandler("/Error", createScopeForErrors: true);
22      // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts
23      app.UseHsts();
24  }
25
26  app.UseHttpsRedirection();

```

E o último passo antes de testarmos nossa Web Application é criar a nova chamada dos Menus para Autor e Livro. No Solution Explorer, localize a pasta Layout ela está dentro da pasta Components e então abra o arquivo NavMenu.razor

Dentro do contêiner <div> class="nav-scrollable" adicione as duas novas chamadas:

```

<div class="nav-scrollable" onclick="document.querySelector('.navbar-
toggler').click()">
  <nav class="flex-column">
    <div class="nav-item px-3">
      <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
        <span class="bi bi-house-door-fill-nav-menu" aria-hidden="true"></span>
Home
      </NavLink>
    </div>

    <div class="nav-item px-3">
      <NavLink class="nav-link" href="counter">
        <span class="bi bi-plus-square-fill-nav-menu" aria-hidden="true"></span>
Counter
      </NavLink>
    </div>

    <div class="nav-item px-3">
      <NavLink class="nav-link" href="weather">

```



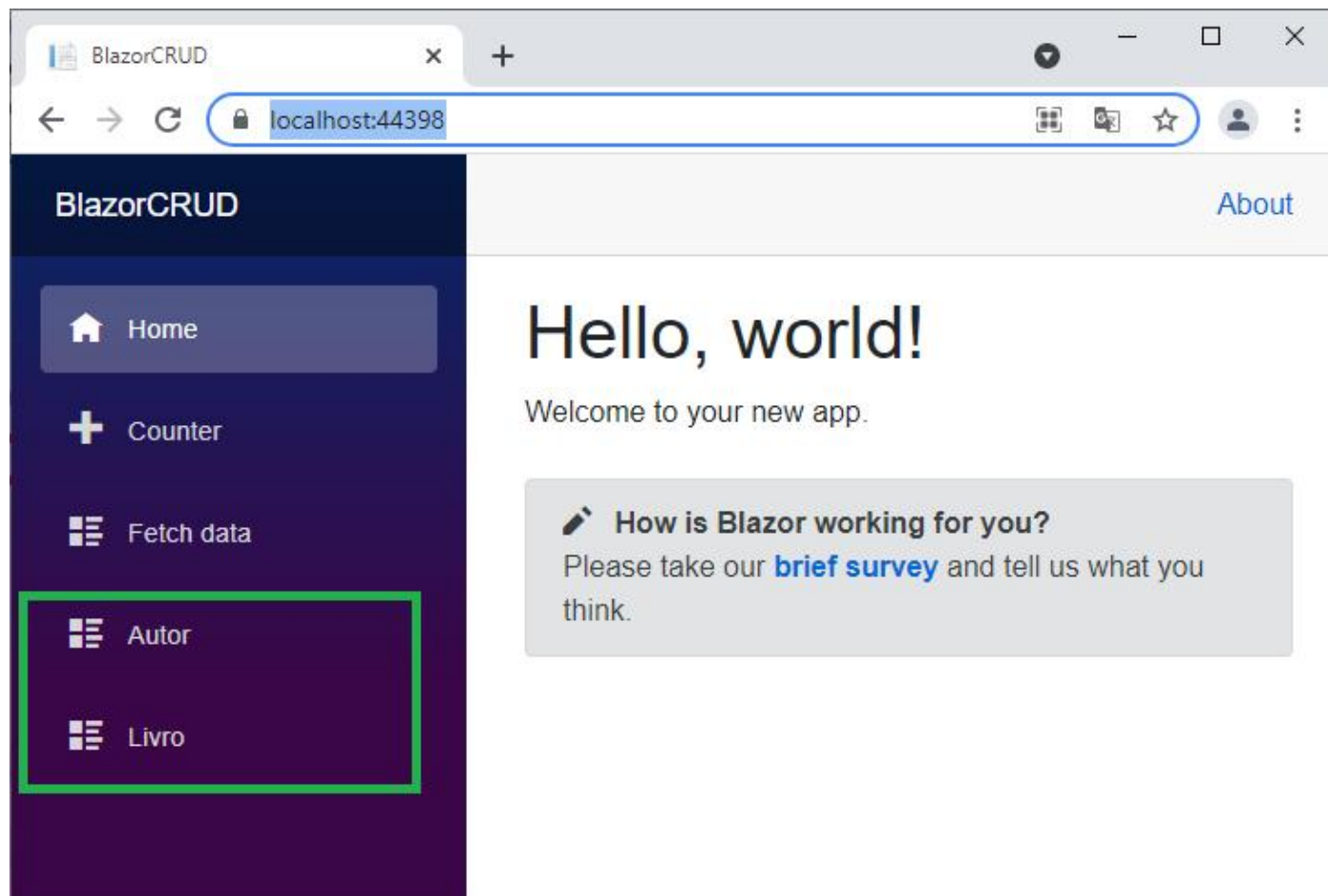
```
        <span class="bi bi-list-nested-nav-menu" aria-hidden="true"></span>
Weather
    </NavLink>
</div>

<div class="nav-item px-3">
    <NavLink class="nav-link" href="autores">
        <span class="bi bi-list-nested-nav-menu" aria-hidden="true"></span> Autor
    </NavLink>
</div>

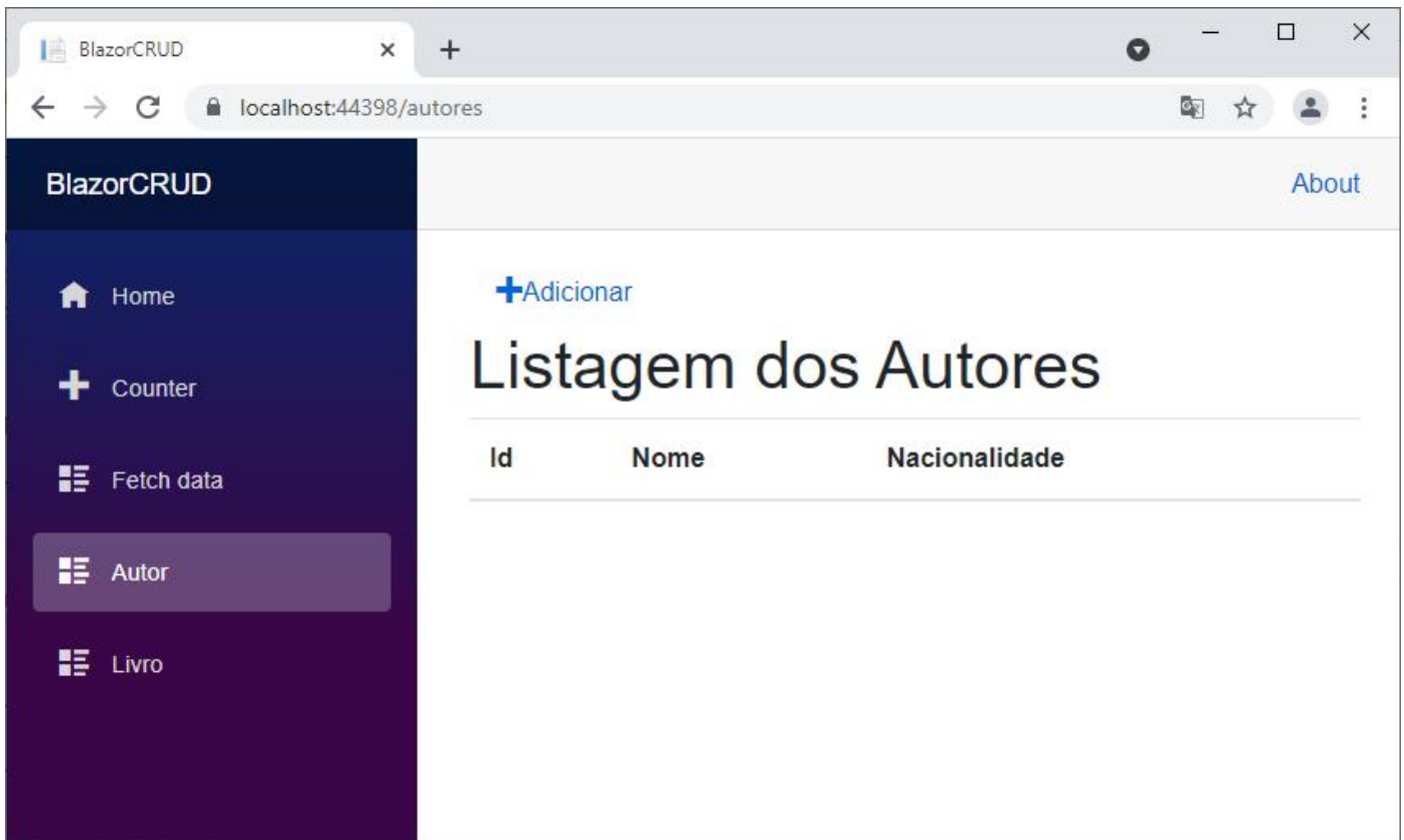
<div class="nav-item px-3">
    <NavLink class="nav-link" href="livros">
        <span class="bi bi-list-nested-nav-menu" aria-hidden="true"></span> Livro
    </NavLink>
</div>
</nav>
</div>
```

Nosso BlazorCRUD rodando...

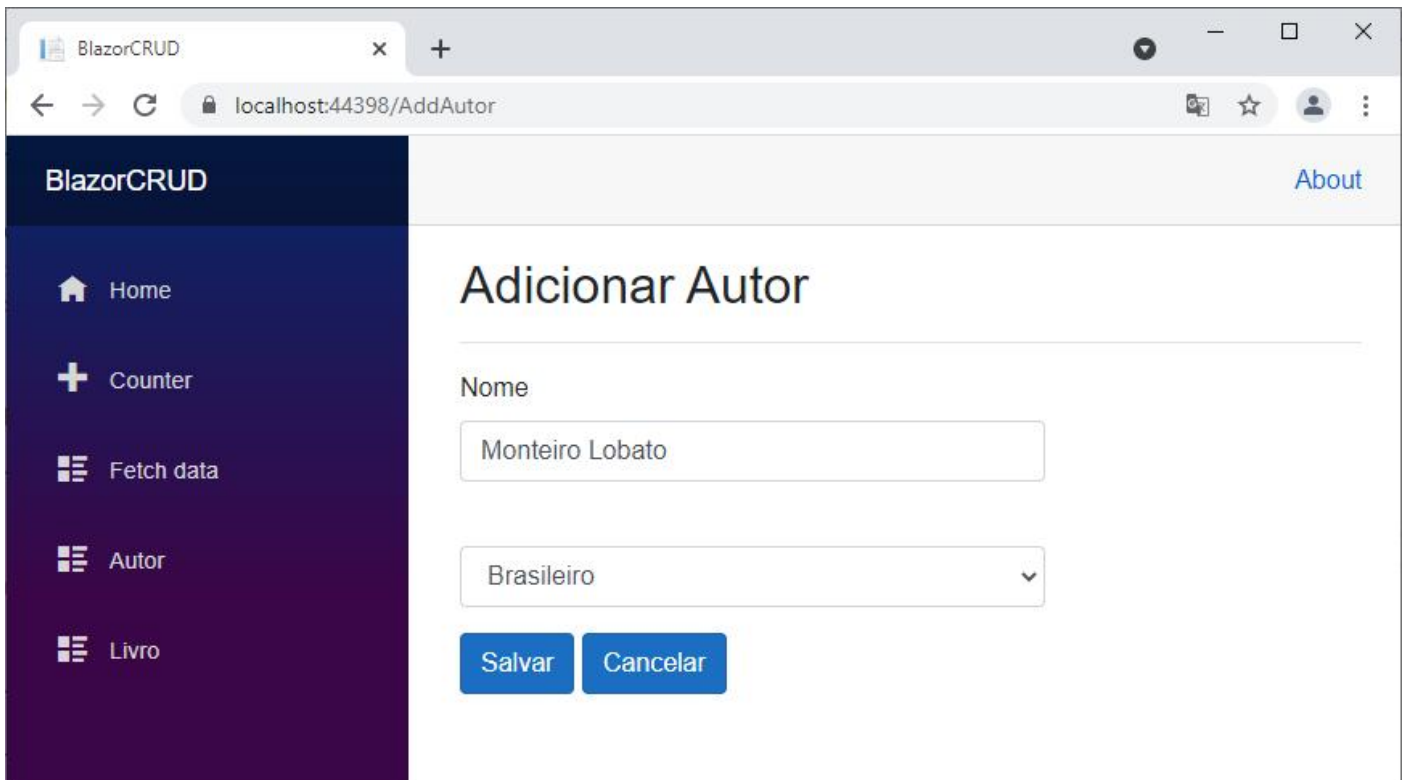
Novas chamadas do Menu



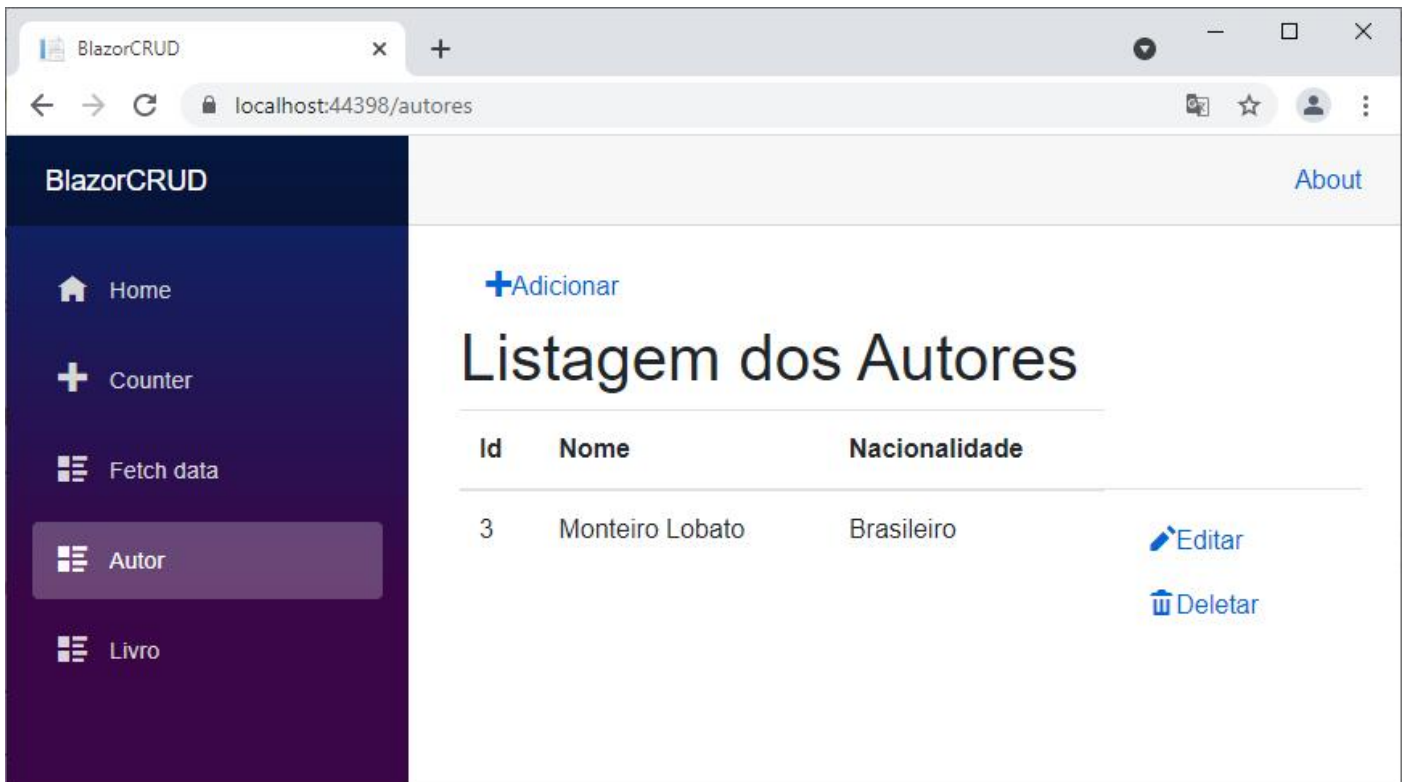
Lista de Autores (Banco de dados vazio):



Cadastrar um novo Autor:



Listagem de Autores (Banco de Dados com registro):



BlazorCRUD

About

+Adicionar

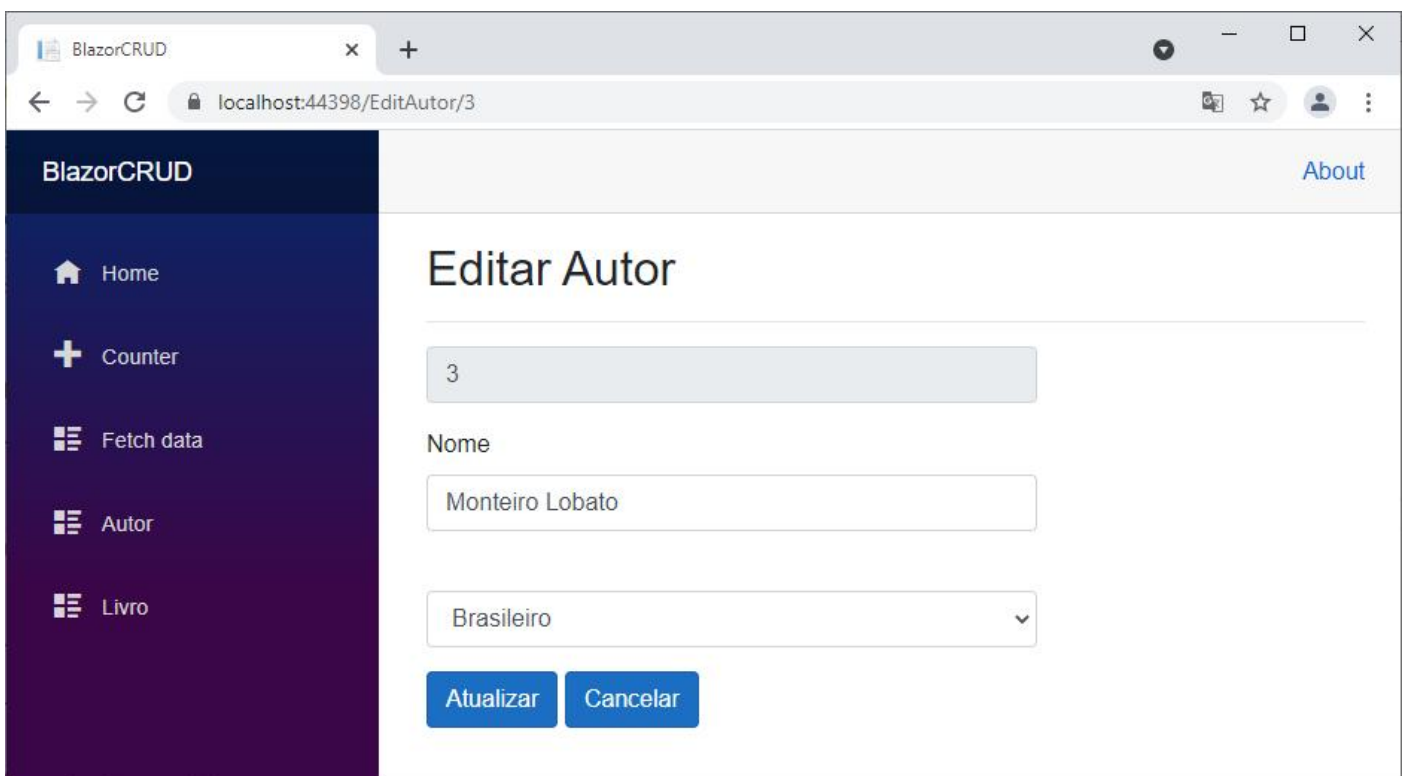
Listagem dos Autores

Id	Nome	Nacionalidade
3	Monteiro Lobato	Brasileiro

Editar

Deletar

Editar um Autor:



BlazorCRUD

About

Editar Autor

3

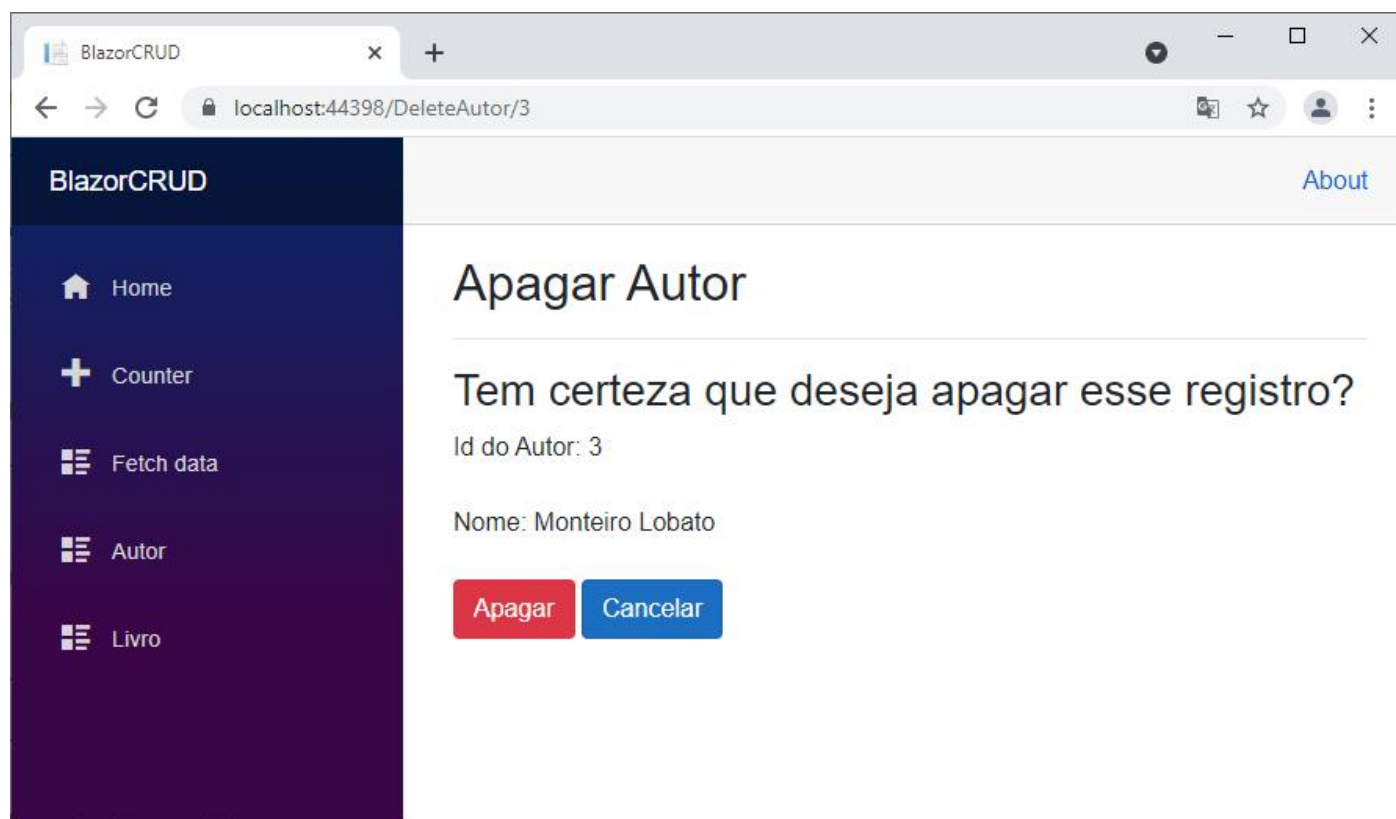
Nome

Monteiro Lobato

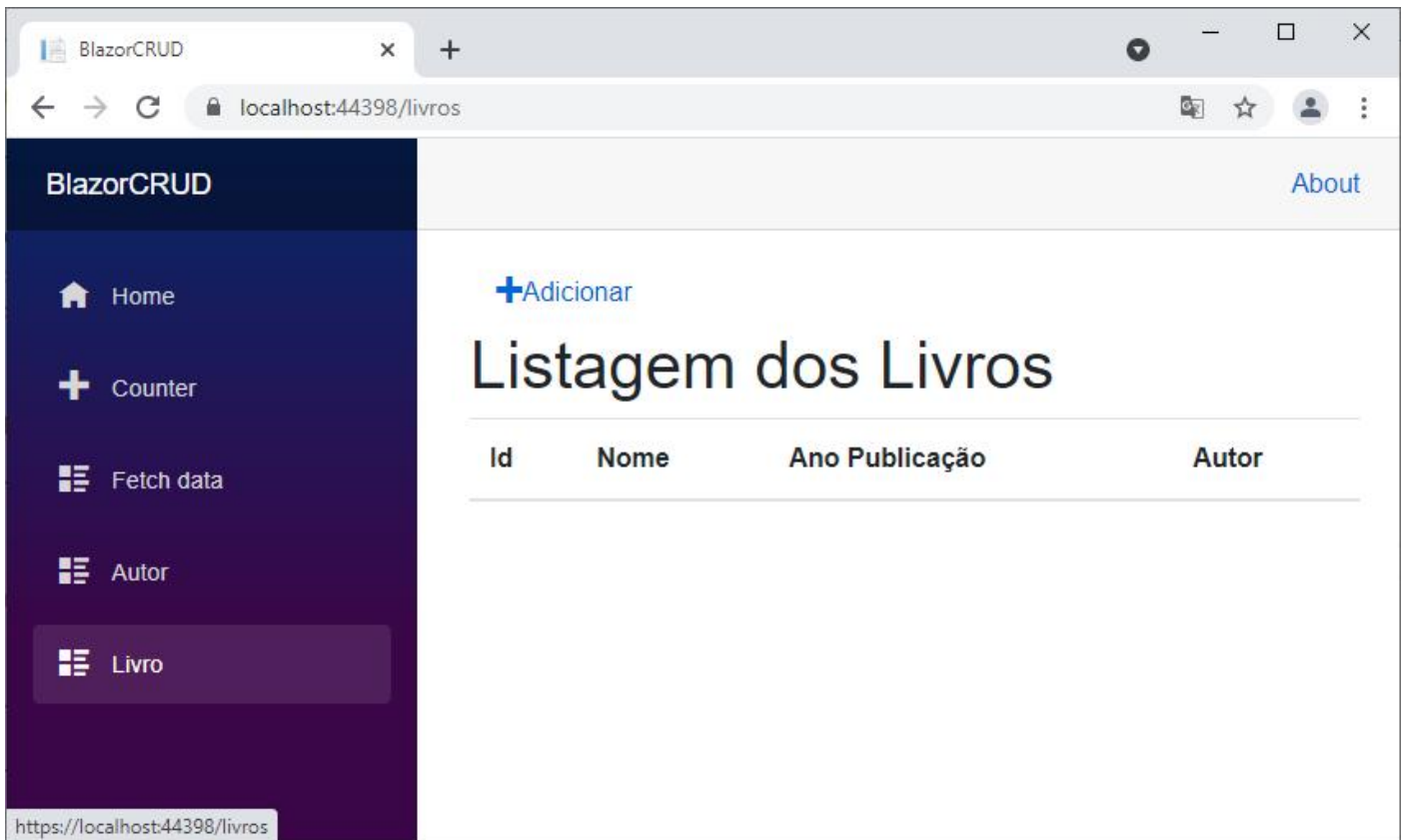
Brasileiro

Atualizar Cancelar

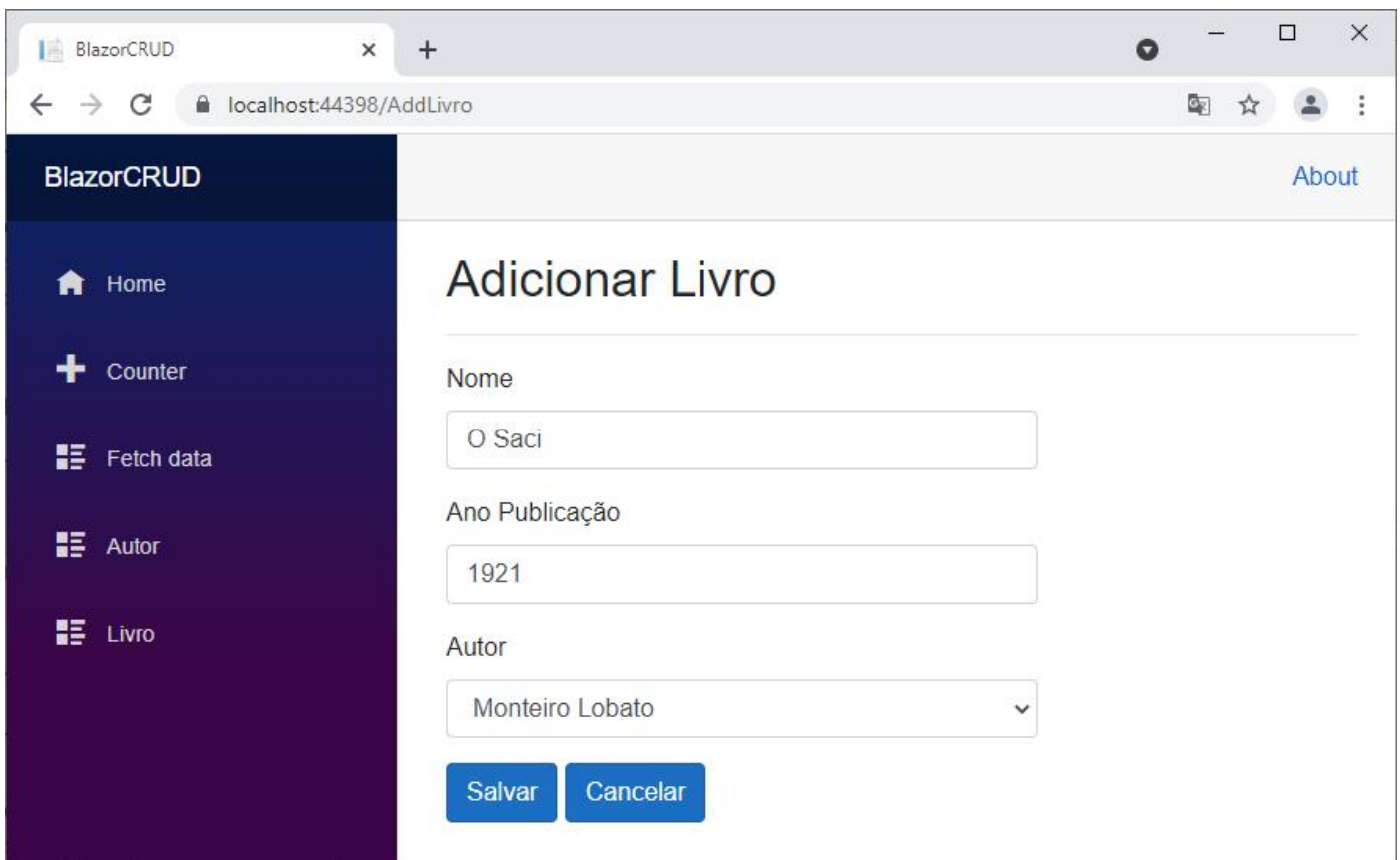
Apagar um Autor:



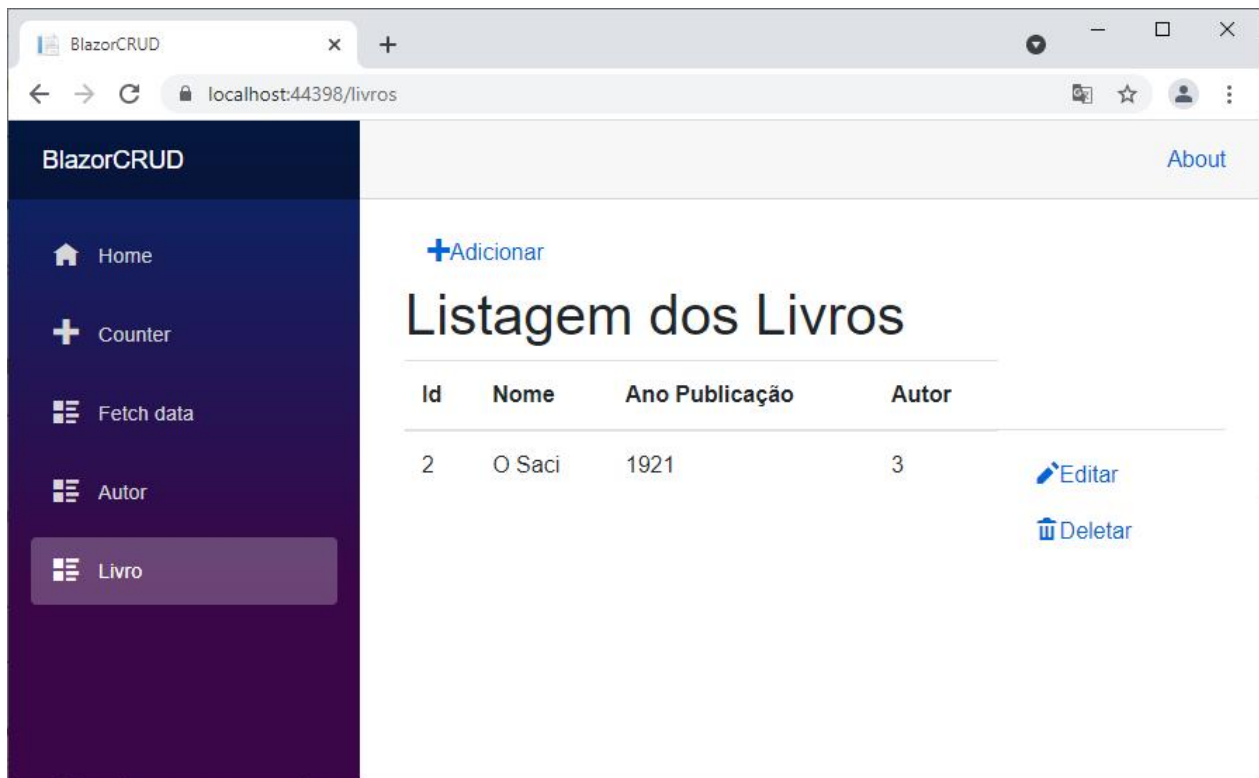
Lista de Livros (Banco de dados vazio):



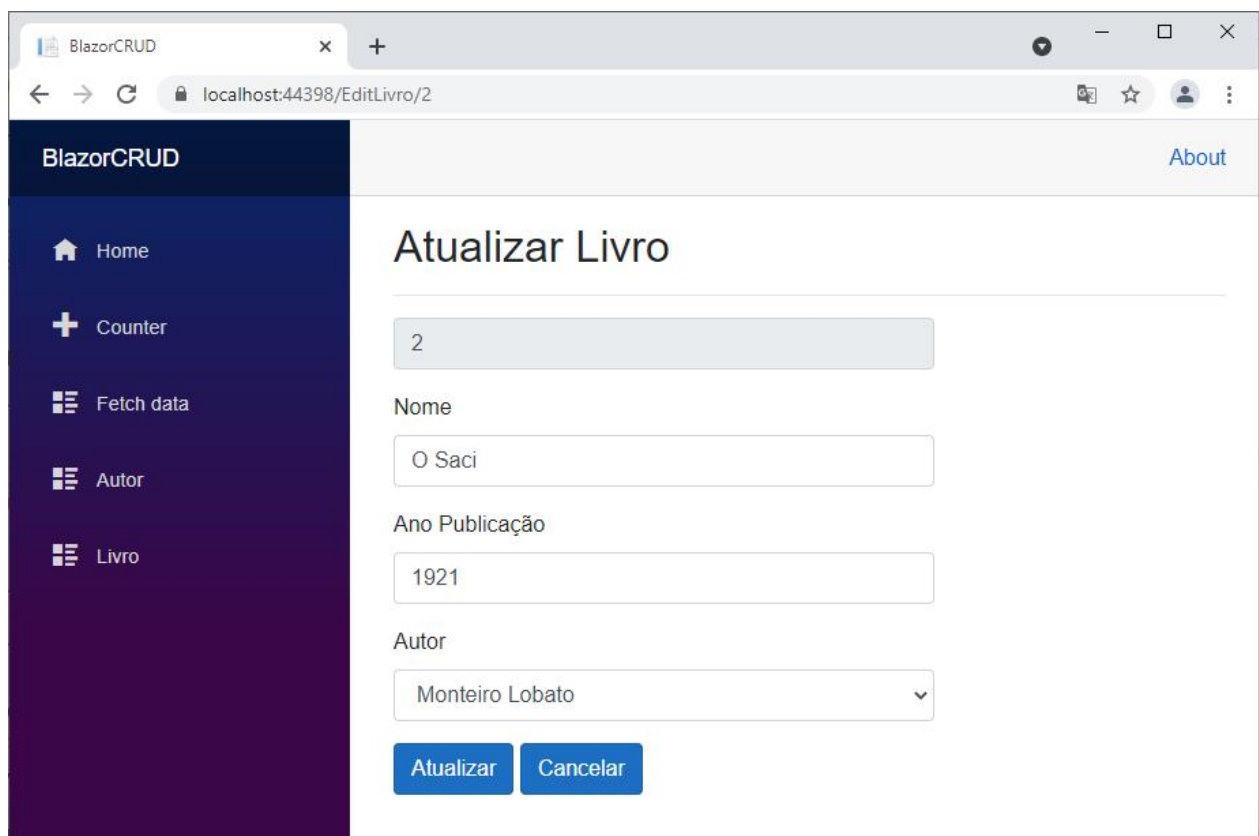
Adicionar um Livro:



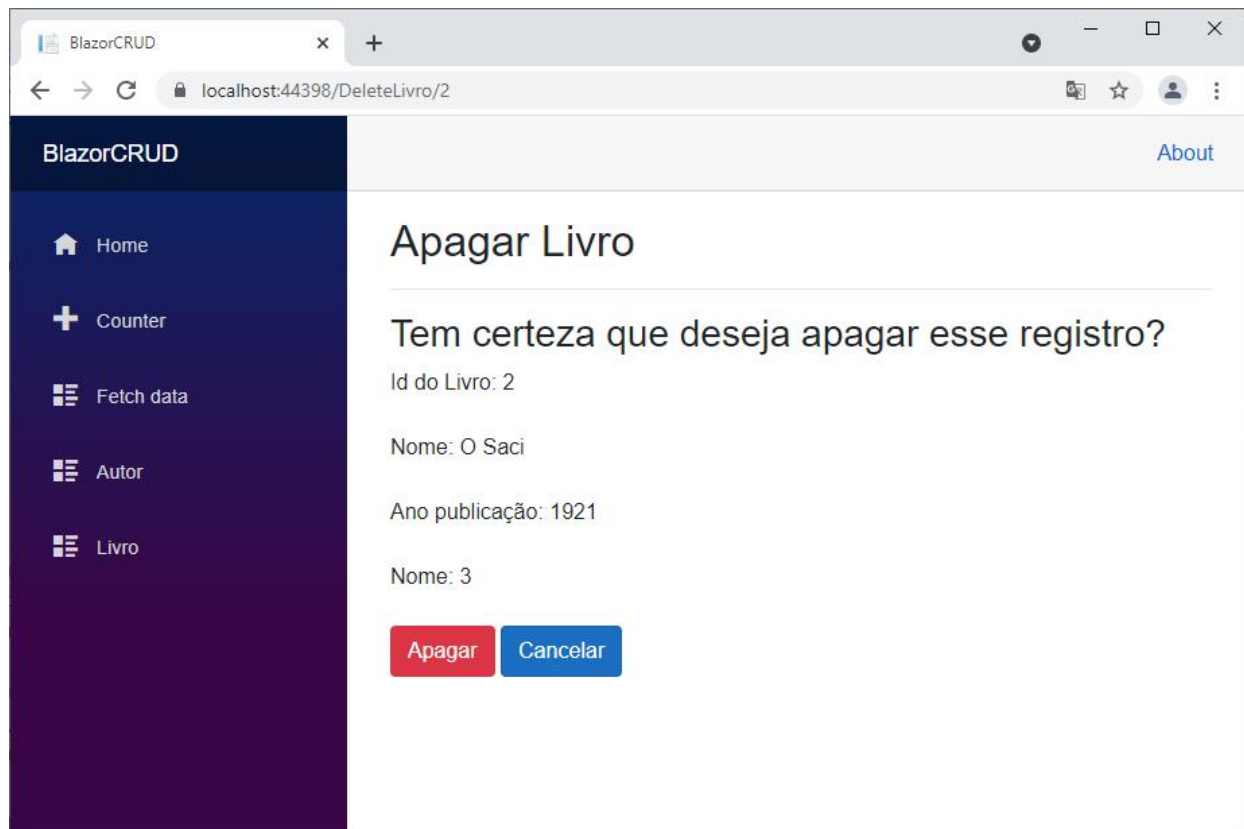
Lista de Livros (Banco de dados preenchido):



Atualizar um Livro:



Apagar um Livro:



Observações importantes: reparem que, na listagem dos Livros é mostrado na coluna Autor o Id dele e não o Nome do Autor. Isso acontece também quando você clica em Apagar Livro. Vamos realizar alguns ajustes em todos esses pontos para melhorar nossa apresentação da informação em tela.

Volte na Classe Livro.cs e inclua um novo atributo:

```
public class Livro
{
    public int Id { get; set; }

    public string Nome { get; set; }

    public int AnoPublicacao { get; set; }

    [ForeignKey("Autor")]
    public int AutorId { get; set; }

    public virtual Autor Autor { get; set; }
}
```


Com esse novo atributo, vamos criar um vínculo virtual para facilitar a busca o objeto completo do Autor e assim ter acesso a todos os seus atributos.

No arquivo LivrosService.cs realize o ajuste abaixo no método GetLivro()

```
// Obter todos os livros
public List<Livro> GetLivro()
{
    var livroList = _db.Livros.Include(l => l.Autor).ToList();
    return livroList;
}
```

Observação: adicione a using Microsoft.EntityFrameworkCore para usar o Include

```
// Obter todos os livros
1 reference
public List<Livro> GetLivro()
{
    var livroList = _db.Livros.Include(l => l.Autor).ToList();

    return livroList;
}

// Inserir um Livro
1 reference
public string Create(Livro objLivro)
{
    db.Livros.Add(objLivro);
}
```

using Microsoft.EntityFrameworkCore;

CS1061 'DbSet<Livro>' does not contain a definition for 'Include' and no accessible extension method 'Include' accepting a first argument of type 'DbSet<Livro>' could be found (are you missing a using directive or...)

Lines 2 to 3

```
using BlazorCRUD.Model;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
```

Preview changes

No arquivo LivroPage.razor, no trecho abaixo, faça o ajuste conforme o modelo:

```
@foreach (var livro in objLivros)
{
    <tr>
        <td>@livro.Id</td>
        <td>@livro.Nome</td>
        <td>@livro.AnoPublicacao</td>
        <td>@livro.Autor.NomeCompleto </td>
        <td>
            <a class="nav-link" href="EditLivro/@livro.Id">
                <span class="oi oi-pencil" aria-hidden="true"></span> Editar
            </a>
        </td>
    </tr>
}
```

```
        <a class="nav-link" href="DeleteLivro/@livro.Id">
            <span class="oi oi-trash" aria-hidden="true"></span>Deletar
        </a>
    </td>
</tr>
}
```

No arquivo DeleteLivroPage.razor, faça os seguintes ajustes:

```
@page "/DeleteLivro/{CurrentID}"

@using BlazorCRUD.Data

@inject LivroService _objLivroService
@inject AutorService _objAutorService
@inject NavigationManager _objNavigationManager

<h2>Apagar Livro</h2>
<hr />

@if (objLivro.Autor == null)
{
    <p><em>Carregando...</em></p>
}
else
{
    <h3>Tem certeza que deseja apagar esse registro?</h3>

    <div class="row">
        <div class="col-md-8">
            <div class="form-group">
                <label>Id do Livro: </label>
                <label>@objLivro.Id</label>
            </div>
            <div class="form-group">
                <label>Nome: </label>
                <label>@objLivro.Nome</label>
            </div>
            <div class="form-group">
                <label>Ano publicação: </label>
```

```

        <label>@objLivro.AnoPublicacao</label>
    </div>
    <div class="form-group">
        <label>Nome: </label>
        <label>@objLivro.Autor.NomeCompleto</label>
    </div>
</div>
</div>
<div class="row">
    <div class="col-md-4">
        <div class="form-group">
            <input type="button" class="btn btn-danger" @onclick="@DeleteLivro"
value="Apagar" />
            <input type="button" class="btn btn-primary" @onclick="@Cancel"
value="Cancelar" />
        </div>
    </div>
</div>
}

```

```

@code {
    [Parameter] public string CurrentId { get; set; }

    Livro objLivro = new();

    protected override async Task OnInitializedAsync()
    {
        objLivro = await Task.Run(() =>
        _objLivroService.GetLivroById(Convert.ToInt32(CurrentId)));
        objLivro.Autor = await Task.Run(() =>
        _objAutorService.GetAutorById(objLivro.AutorId));
    }

    protected void DeleteLivro()
    {
        _objLivroService.DeleteLivro(objLivro);
        _objNavigationManager.NavigateTo("livros");
    }

    void Cancel()
    {

```

```
    _objNavigationManager.NavigateTo("livros");  
}  
}
```

Rode novamente a aplicação e veja os resultados.