

HTML Helper e Tag Helper

Os HTML Helpers, são métodos que possibilitam a renderização de controles HTML nas views, para renderizar algum elemento HTML ou um comportamento desejável nessa view sendo possível também a implementação de HTML Helpers personalizados.

A função de um HTML Helper é encapsular um código HTML. Por exemplo, para adicionar um link, podemos usar o método `ActionLink` do objeto HTML, ao invés de usar a tag `<a>` do HTML diretamente.

Além dos HTML Helper o .Net Core nos oferece a opção de usar os Tag Helpers.

Segundo a Microsoft os Tag Helpers permitem que o código do lado do servidor participe da criação e renderização de elementos HTML em arquivos Razor, existem muitos Tag Helpers integrados para tarefas comuns - como criar formulários, links, carregar recursos e muito mais.

As Tag Helpers permitem que o código do lado do servidor participe na criação e renderização de elementos HTML em arquivos Razor. Elas são um novo recurso semelhante aos HTML Helpers, que nos ajudam a renderizar o HTML.

A marcação é muito mais limpa e fácil de ler, editar e manter do que a abordagem de auxiliares de HTML. O código C# é reduzido ao mínimo que o servidor precisa saber.

Razor

As Razor Pages é um um novo recurso da ASP.NET Core MVC que torna a codificação de cenários focados em páginas mais fácil e mais produtiva.

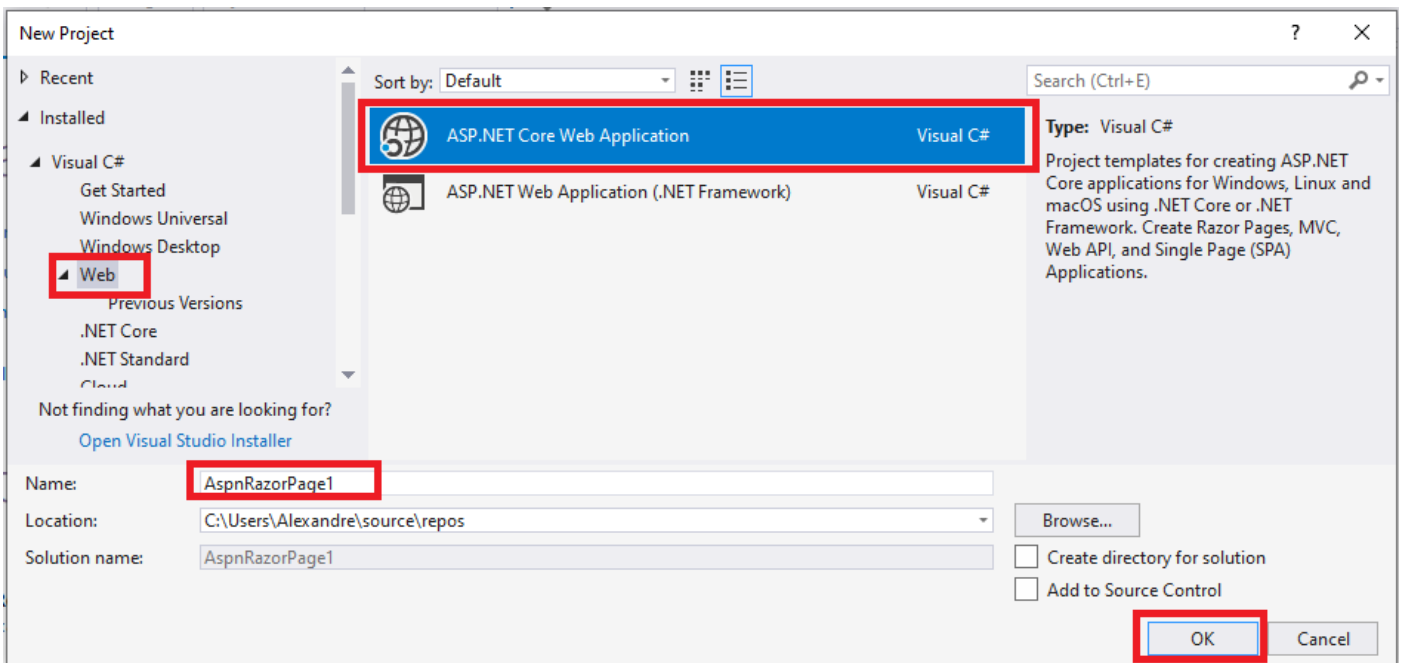
Uma das vantagens das Razor Pages é que sua configuração é bem direta e simplificada. Basta criar um novo projeto vazio, adicionar a pasta Pages, criar uma página, e daí você apenas escreve código e a marcação dentro de seu arquivo .cshtml.

Criando o projeto no VS 2017

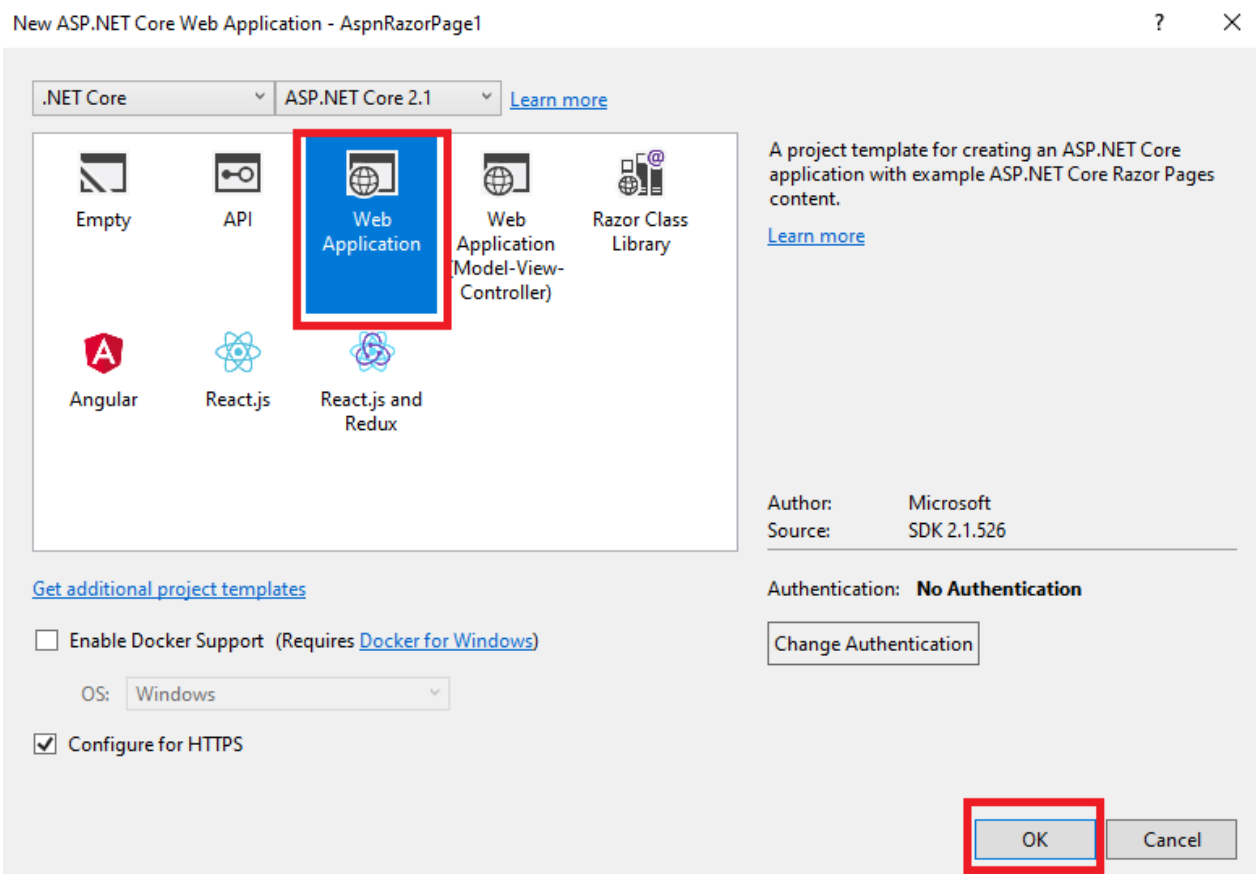
Abra o **VS 2017 Community** e crie um novo projeto ASP .NET Core usando o template **Empty**.

- Create New Project;
- **Visual C# > Web > ASP .NET Core > Web Application;**

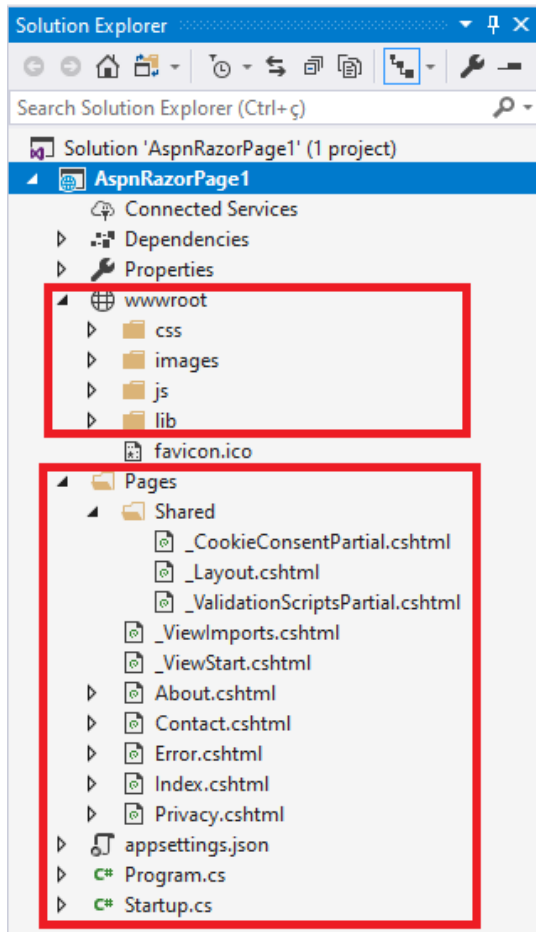
- Informe o nome **AspnRazorPage1**



- Selecione o template **Web Application**, marque ASP .NET Core e .NET Core



Confirme as opções clicando em OK para criar o projeto. Ao final teremos o projeto criado com a seguinte estrutura:



Observe que a estrutura do projeto é diferente da estrutura do projeto MVC.

No projeto Razor Pages você não tem as pastas Models, Views, Controllers, Services, Data.

Temos apenas as pastas **wwwroot** onde ficam os arquivos estáticos nas subpastas: **css, images, js e lib**

Temos também a pasta Pages contendo as páginas da aplicação e os arquivos:

`_Layout.cshtml`, `_ViewImports.cshtml` e `_ViewStart.cshtml` usados para configurar a aplicação.

Além disso temos os arquivos: `appsettings.json` - arquivo de configuração contendo a string de conexão para o banco de dados

`Program.cs` - arquivo de entrada da

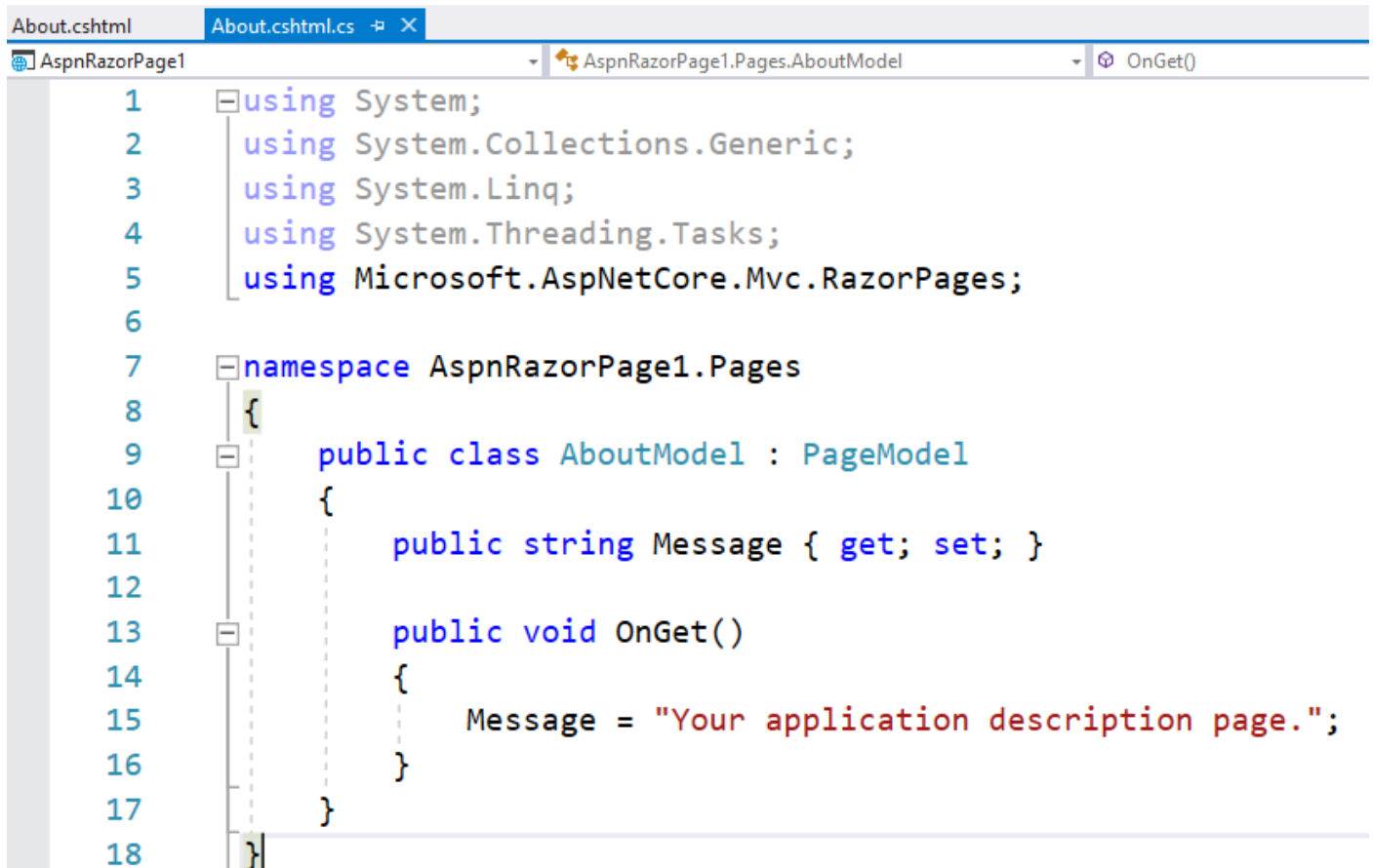
aplicação.

`Startup.cs` - arquivo de inicialização da aplicação.

Nota: Se você criar o projeto Razor Pages com autenticação será criada a pasta Controllers com o controlador AccountController e uma pasta Account dentro da pasta Pages.

Vamos abrir o arquivo **About.cshtml** e o seu arquivo code-behind **About.cshtml.cs** e visualizar a estrutura destes arquivos:

```
About.cshtml  About.cshtml.cs
1  @page
2  @model AboutModel
3  @{
4      ViewData["Title"] = "About";
5  }
6  <h2>@ViewData["Title"]</h2>
7  <h3>@Model.Message</h3>
8
9  <p>Use this area to provide additional information.</p>
```



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5 using Microsoft.AspNetCore.Mvc.RazorPages;
6
7 namespace AspnRazorPage1.Pages
8 {
9     public class AboutModel : PageModel
10     {
11         public string Message { get; set; }
12
13         public void OnGet()
14         {
15             Message = "Your application description page.";
16         }
17     }
18 }
```

Note que no arquivo About.cshtml, no topo da página temos a diretiva **@page** que informa ao Razor que este arquivo .cshtml representa uma Razor Page.

A diretiva **@page** torna o arquivo uma Action MVC, o que significa que ele pode tratar requisições diretamente, sem passar pelo controlador. Esta diretiva afeta o comportamento de outras construções Razor.

A diretiva **@model AboutModel** informa ao ASP .NET Core para vincular esta página com uma instância de **AboutModel**.

Por convenção o arquivo da classe **PageModel** tem o mesmo nome que o arquivo da página Razor com a extensão .cs.

Neste caso a Razor Page About.cshtml contém a diretiva **@model AboutModel** e o arquivo About.cshtml.cs contém a classe **AboutModel** que herda da classe **PageModel**.

Observe que no arquivo About.cshtml.cs a classe **AboutModel** possui o método **OnGet()** que é o método padrão para operação GET.

O Model Binding, que já conhecemos do MVC, também funciona com a Razor Pages. Assim como os métodos Actions dos Controllers MVC, temos na Razor Pages os Handlers.

O model binding da ASP.NET Core mapeia dados de requisições HTTP para parâmetros dos métodos Action dos controladores.

Usamos os **Handlers** ou manipuladores como métodos para lidar com solicitações HTTP (GET, POST, PUT, DELETE ...). Podemos ter os seguintes métodos:

OnGet / OnGetAsync

OnPost / OnPostAsync

OnDelete / OnDeleteAsync

Estes métodos serão automaticamente utilizados pela ASP.NET Core com base no tipo de solicitação HTTP.

Desta forma assim que você abrir a página About.cshtml o método OnGet() será executado. Da mesma forma em uma página contendo o método OnPost() este método será executado ao submeter a página.

Podemos ainda mover estes métodos para a página Razor About.cshtml usando a diretiva @functions :

```
About.cshtml ➤ X About.cshtml.cs
1  @page
2  @model AboutModel
3  @{
4      ViewData["Title"] = "About";
5  }
6  <h2>@ViewData["Title"]</h2>
7  <h3>@Model.Message</h3>
8
9  <p>Use this area to provide additional information.</p>
10
11  @functions{
12      public string Message { get; set; }
13      public void OnGet()
14      {
15          Message = "Your application description page.";
16      }
17  }
```

Neste caso não precisaríamos do arquivo code-behind About.cshtml.cs.

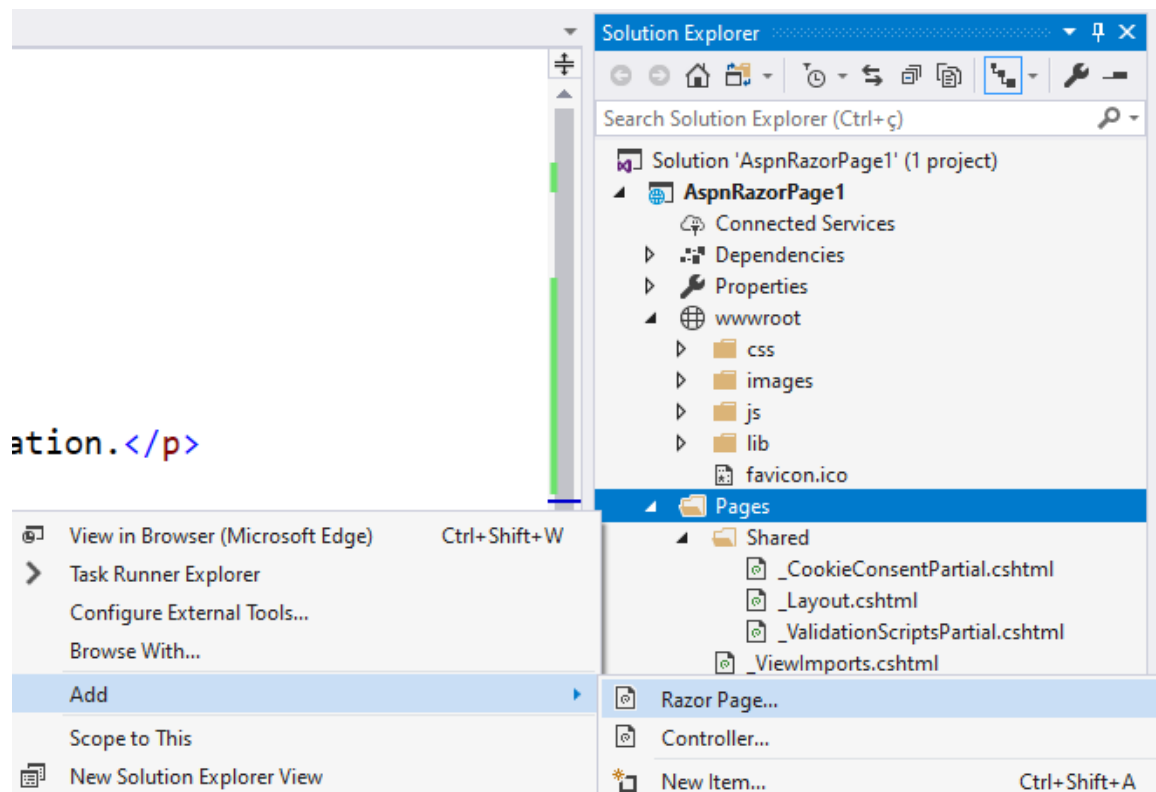
As associações de caminhos de URL para páginas são determinadas pela localização da página no sistema de arquivos.

A tabela a seguir mostra um caminho da Página Razor e a URL correspondente:

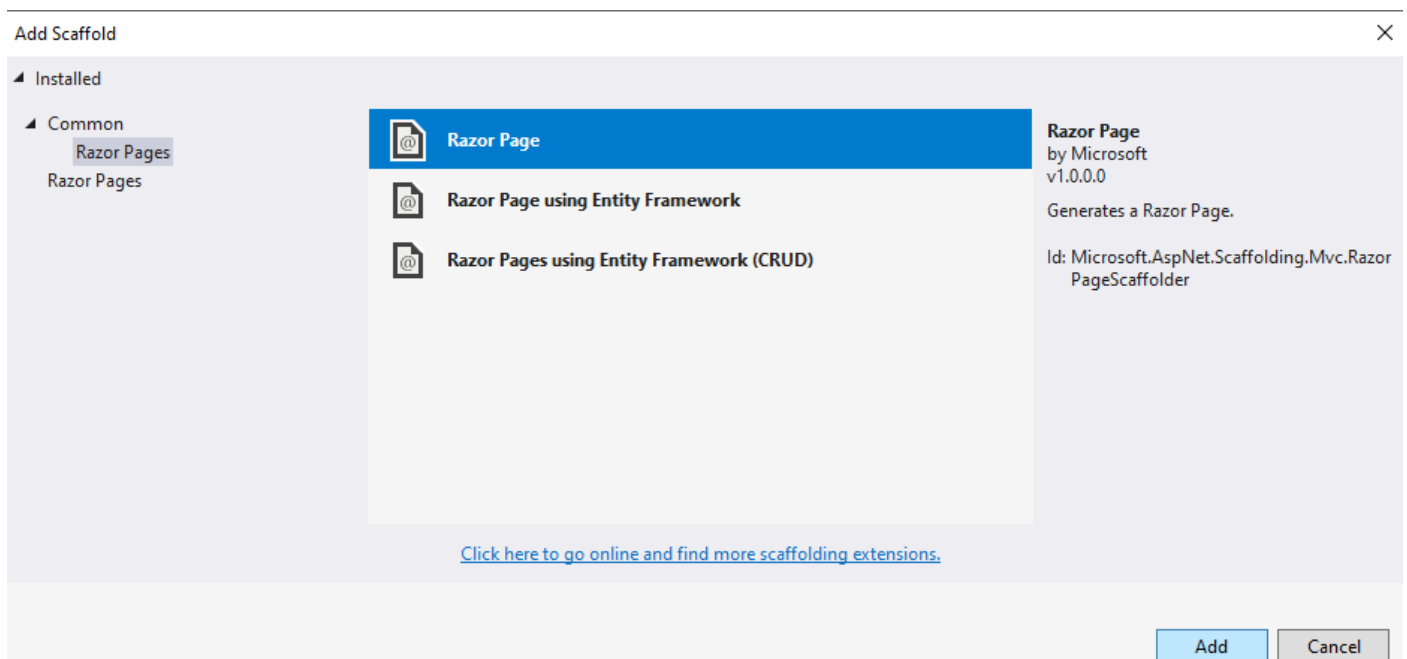
Nome do arquivo e caminho	URL correspondente
/Pages/Index.cshtml	/ ou /Index
/Pages/Contact.cshtml	/Contact
/Pages/Store/Contact.cshtml	/Store/Contact
/Pages/Store/Index.cshtml	/Store ou /Store/Index

Para incluir novas Razor Page em nosso projeto podemos usar o **Scaffolding** do Visual Studio clicando com o botão direito do mouse sobre a pasta **Pages** > **Add** > selecionar a opção **Razor Page**:

Scaffolding é uma estrutura de geração de código para aplicativos ASP.NET Web.



Selecione Razor Page e clique em Add



Dê o nome de Create, verifique as opções e clique em Add;

Add Razor Page

Razor Page name:

Options:

- ☒ Generate PageModel class
- ☐ Create as a partial view
- ☒ Reference script libraries
- ☒ Use a layout page:

...

(Leave empty if it is set in a Razor _viewstart file)

Para concluir vamos executar a nossa aplicação, onde iremos obter o seguinte resultado:

Home page - AspRazorPage1 x

https://localhost:44318

Use this space to summarize your privacy and cookie use policy. [Learn More](#) [Accept](#)

ASP.NET Core | Windows Linux OSX

Learn how to build ASP.NET apps that can run anywhere.

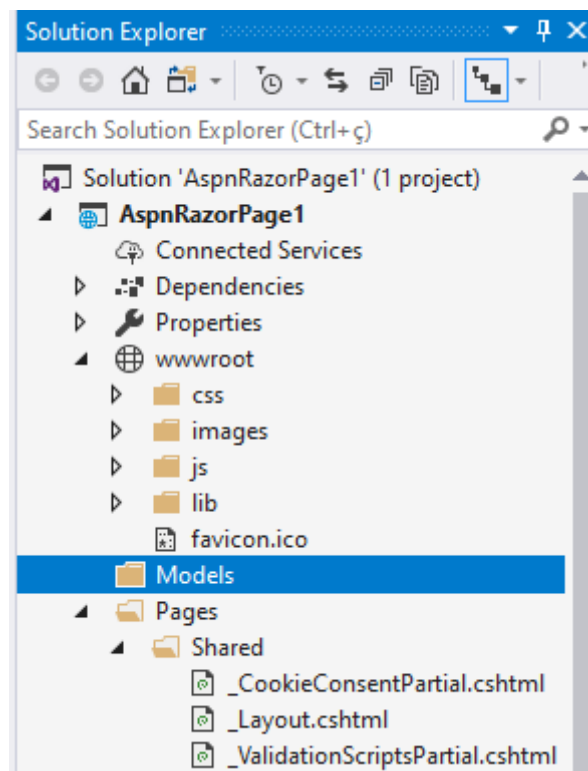
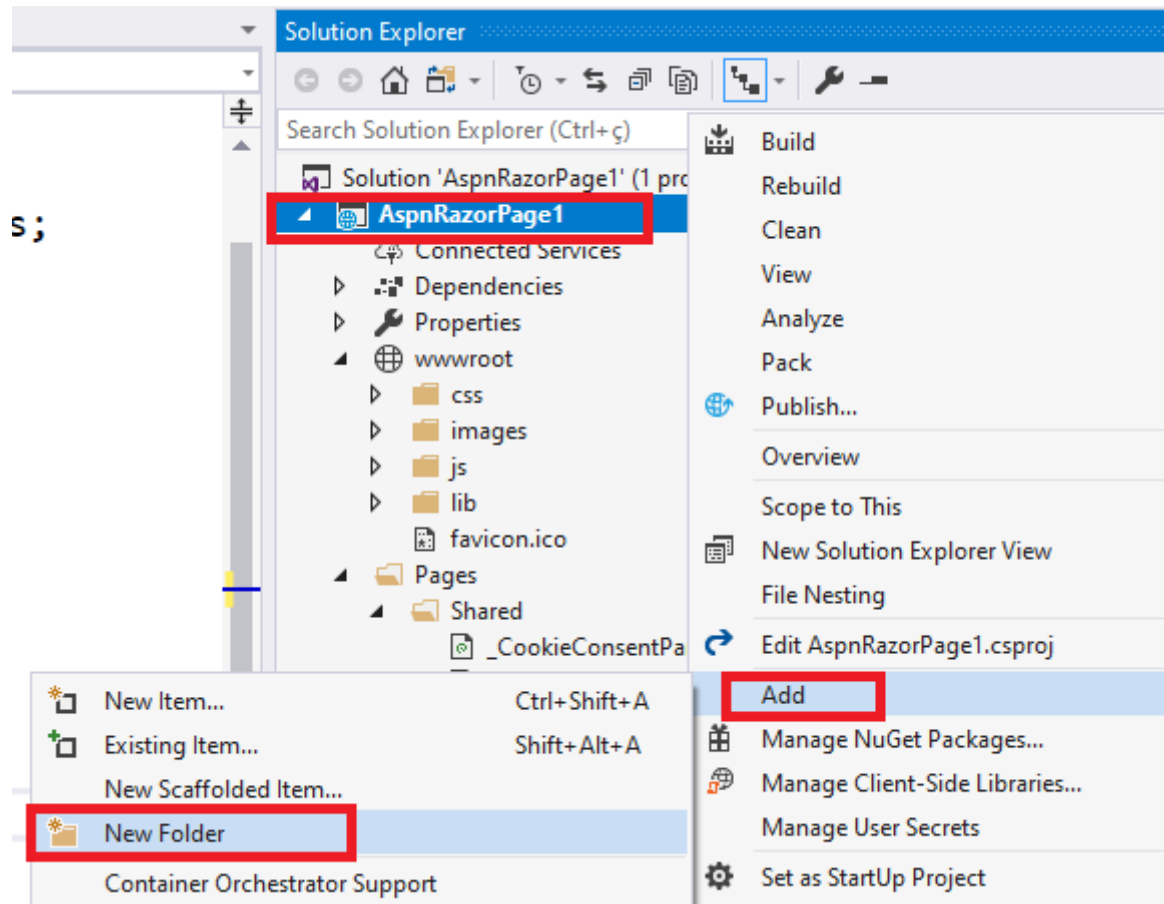
[Learn More](#)

- Application uses
 - Sample pages using ASP.NET Core Razor Pages
 - Theming using Bootstrap
- How to
 - Working with Razor Pages.
 - Manage User Secrets using Secret Manager.
 - Use logging to log a message.
 - Add packages using NuGet.
 - Target development, staging or production environment.
- Overview
 - Conceptual overview of what is ASP.NET Core
 - Fundamentals of ASP.NET Core such as Startup and middleware.
 - Working with Data
 - Security
 - Client side development
 - Develop on different platforms
 - Read more on the documentation site
- Run & Deploy
 - Run your app
 - Run tools such as EF migrations and more
 - Publish to Microsoft Azure App Service

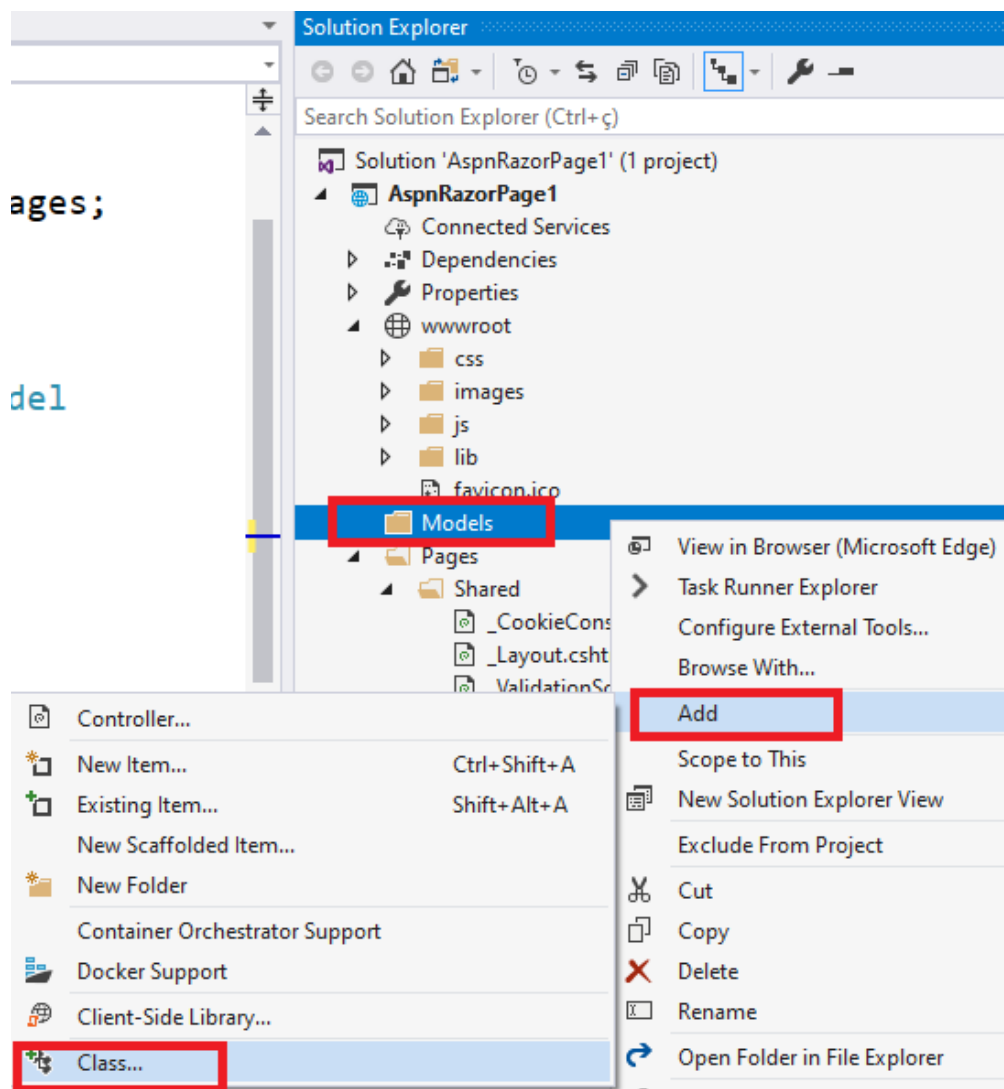
© 2021 - AspRazorPage1

Ajustando o projeto para gerenciar informações de clientes

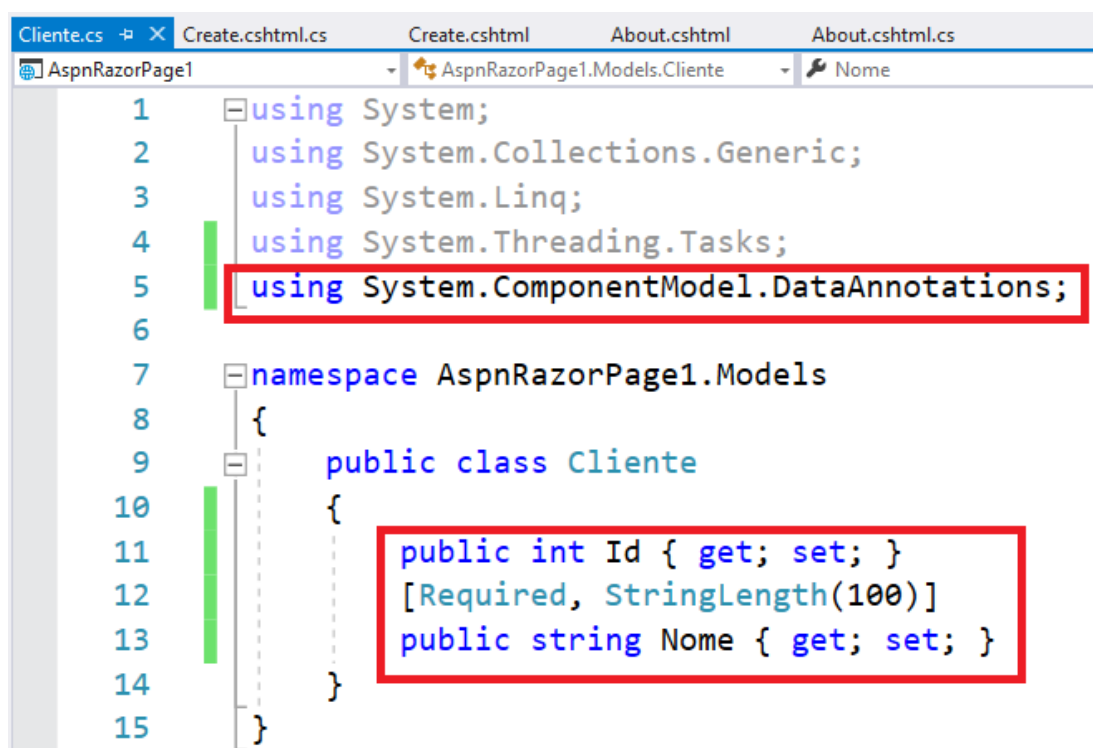
Abra o projeto AspnetRazorPage1 criado na primeira parte do artigo e inclua uma pasta **Models** no projeto. (Project-> New Folder)



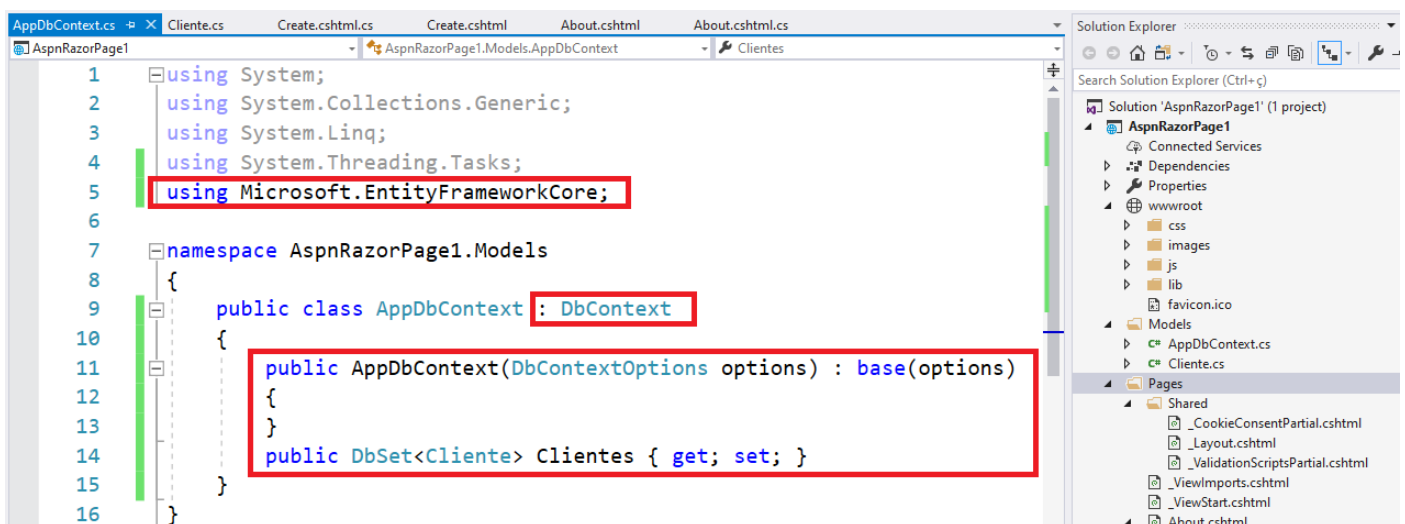
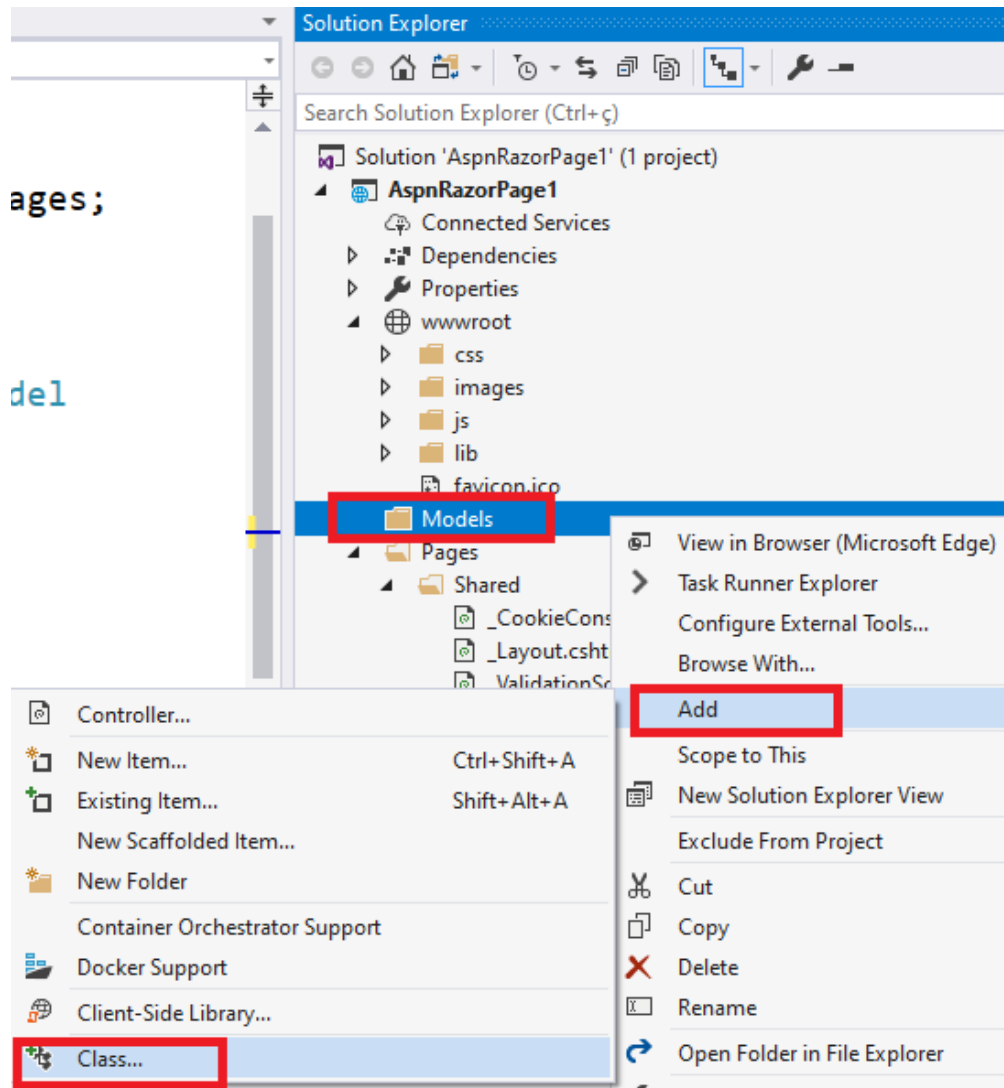
Crie a classe Cliente que vai representar um cliente do nosso modelo de domínio com o seguinte código:



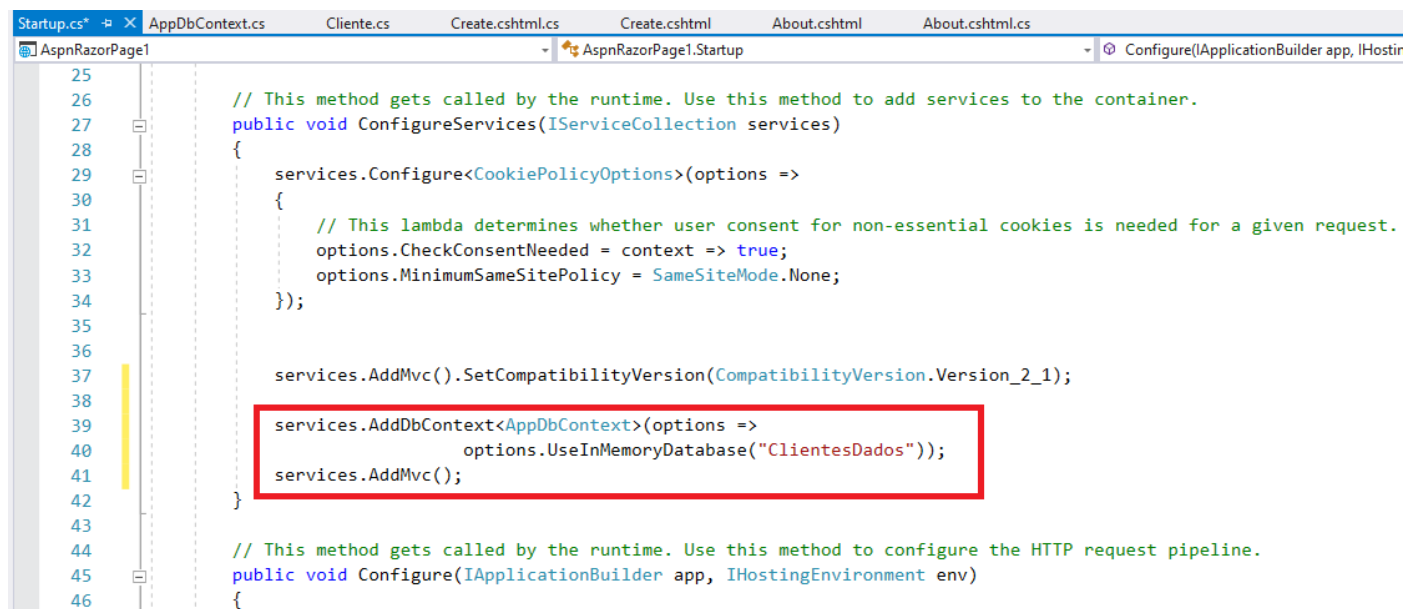
Adicione a biblioteca e o código para cadastro do cliente.



A seguir vamos criar a classe de contexto chamada **AppDbContext** que herda de **DbContext** na pasta **Models** com o código abaixo:

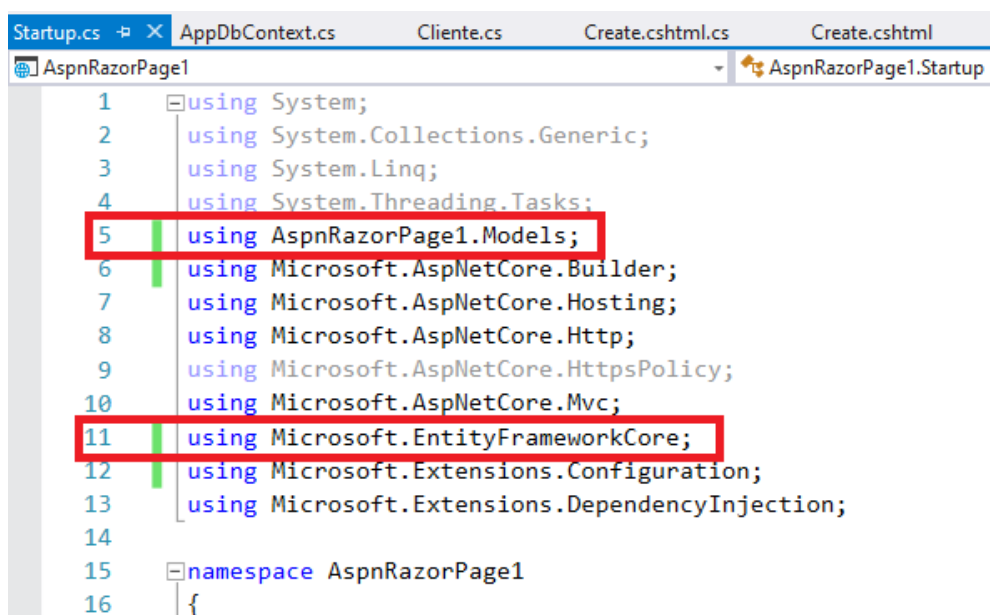


Agora vamos configurar o método **ConfigureServices** da classe **Startup.cs** definindo a inicialização do **DbContext**:



```
25
26 // This method gets called by the runtime. Use this method to add services to the container.
27 public void ConfigureServices(IServiceCollection services)
28 {
29     services.Configure<CookiePolicyOptions>(options =>
30     {
31         // This lambda determines whether user consent for non-essential cookies is needed for a given request.
32         options.CheckConsentNeeded = context => true;
33         options.MinimumSameSitePolicy = SameSiteMode.None;
34     });
35
36     services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
37
38     services.AddDbContext<AppDbContext>(options =>
39         options.UseInMemoryDatabase("ClientesDados"));
40     services.AddMvc();
41 }
42
43 // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
44 public void Configure(IApplicationBuilder app, IHostingEnvironment env)
45 {
46
```

E adicionaremos a classe Models e o EntityFramework



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5 using AspNetCore.Builder;
6 using AspNetCore.Hosting;
7 using AspNetCore.Http;
8 using AspNetCore.HttpsPolicy;
9 using AspNetCore.Mvc;
10 using Microsoft.EntityFrameworkCore;
11 using Microsoft.Extensions.Configuration;
12 using Microsoft.Extensions.DependencyInjection;
13
14 namespace AspNetCore
15 {
16
```

Entity Framework

O Entity Framework é um ORM (Object-Relational Mappers ou Mapeamento objeto-relacional em português), criado pela Microsoft, que permite o mapeamento de tabelas como objetos.

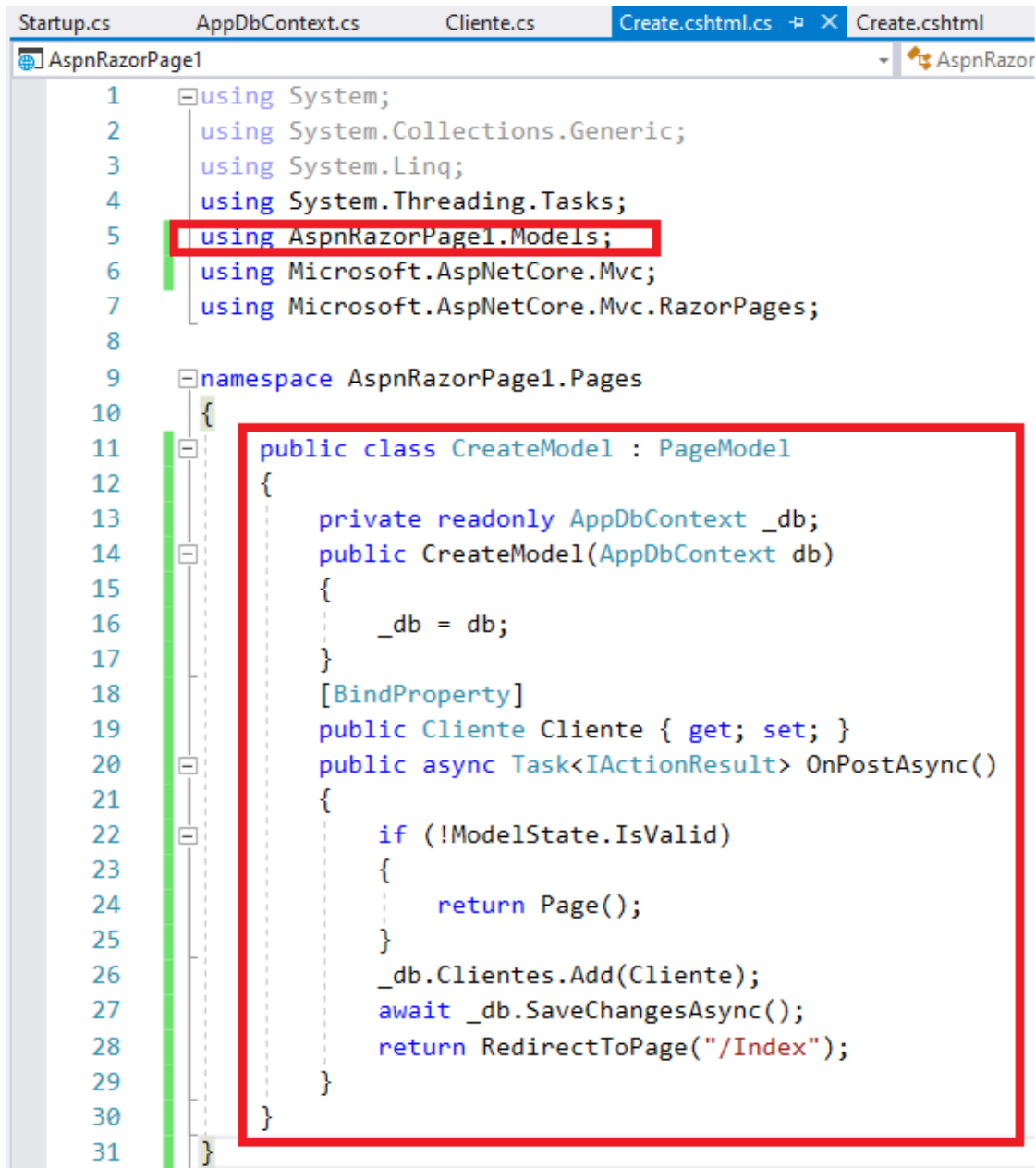
De maneira mais detalhada, um ORM é responsável por mapear objetos em registros (classes em tabelas) e permitir a recuperação e manutenção dos dados relacionais, seguindo o paradigma orientado a objetos.

Segundo a Microsoft, no EF Core o acesso a dados é executado usando um modelo.

Um modelo é composto de classes de entidade e um objeto de contexto que representa uma sessão com o banco de dados.

O objeto **Context** permite consultar e salvar dados.

No arquivo Create.cshtml.cs adicionaremos a classe Models e o código para registrar os dados.



```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using AspnRazorPage1.Models;
6  using Microsoft.AspNetCore.Mvc;
7  using Microsoft.AspNetCore.Mvc.RazorPages;
8
9  namespace AspnRazorPage1.Pages
10 {
11     public class CreateModel : PageModel
12     {
13         private readonly AppDbContext _db;
14         public CreateModel(AppDbContext db)
15         {
16             _db = db;
17         }
18         [BindProperty]
19         public Cliente Cliente { get; set; }
20         public async Task<IActionResult> OnPostAsync()
21         {
22             if (!ModelState.IsValid)
23             {
24                 return Page();
25             }
26             _db.Clientes.Add(Cliente);
27             await _db.SaveChangesAsync();
28             return RedirectToPage("/Index");
29         }
30     }
31 }
```

Por convenção, a classe **PageModel** é chamada **<PageName>Model**, no exemplo: **<Create>Model**, e está no mesmo namespace da página.

A classe **PageModel** permite a separação da lógica de uma página de sua apresentação. Ela define manipuladores de página para requisições enviadas para a página e os dados usados para renderizar a página. *(Esta separação permite gerir dependências de páginas através da injeção de dependência e testar as páginas.)* - "O padrão de injeção de dependências visa remover dependências desnecessárias entre as classes".

A página possui um método **OnPostAsync**, que é executado em requisições POST (*quando um usuário publica o formulário*). Você pode adicionar métodos de manipulador para qualquer verbo HTTP.

O sufixo de nomeação **Async** é opcional, mas geralmente é usado por convenção para funções assíncronas. O código **OnPostAsync** é semelhante ao que você normalmente escreveria em um controlador. O código anterior é típico das Razor Pages. A maioria dos recursos do MVC, como o **Model Binding**, a **validação** e os **action results**, são compartilhados.

No método **OnPostAsync**: temos a verificação para validação de erros com as seguintes ações:

1 - Se não houver erros, salva os dados e faz o redirecionamento

2 - Se houver erros, exibe a página novamente com as mensagens de validação. (*A validação do lado do cliente é idêntica às feitas nas aplicações ASP.NET Core MVC. Em muitos casos, erros de validação seriam detectados no cliente.*)

Quando os dados são inseridos com sucesso, o método do manipulador **OnPostAsync** chama o método helper **redirectToPage** para retornar uma instância de **RedirectToPageResult**.

Aqui, **RedirectToPage** é um novo action result, semelhante ao **RedirectToAction** ou **RedirectToRoute**, mas personalizado para páginas. No código estamos redirecionando para a página **Index** da raiz (/Index).

Quando o formulário enviado tiver erros de validação (*que são passados para o servidor*), o método **OnPostAsync** chama o método Page Helper e **Page** retorna uma instância do **PageResult**. **PageResult** é o tipo de retorno padrão para um método de manipulador. Um método de manipulador que retorna void renderiza a página.

Observe que a propriedade Cliente utiliza o atributo **[BindProperty]** para ativar o model binding.

Agora abra o arquivo **Create.cshtml** e inclua o código abaixo:

```
Startup.cs  AppDbContext.cs  Cliente.cs  Create.cshtml.cs  Create.cshtml  About.cshtml
1  @page
2  @model AspnRazorPage1.Pages.CreateModel
3  @{
4      ViewData["Title"] = "Create";
5  }
6
7  <html>
8  <body>
9      <p>
10         Informe seu nome
11     </p>
12     <div asp-validation-summary="All"></div>
13     <form method="POST">
14         <div>Nome: <input asp-for="Cliente.Nome" /></div>
15         <input type="submit" />
16     </form>
17 </body>
18 </html>
```

Por padrão as Razor Pages, vinculam as propriedades somente com verbos que não são do tipo GET. Essa vinculação de propriedades pode reduzir a quantidade de código que você precisa escrever, pois ela reduz o código usando a mesma propriedade para renderizar campos de formulário (**<input asp-for = "Cliente.Nome" />**) e aceita a entrada de dados.

Vamos agora alterar o código da página **Index.cshtml** existente na pasta **Pages** conforme abaixo:

```
Index.cshtml  Startup.cs  AppDbContext.cs  Cliente.cs  Create.cshtml.cs  Create.cshtml  About.cshtml  About.cshtml.cs
1  @page
2  @model IndexModel
3  @{
4      ViewData["Title"] = "Home page";
5  }
6
7  <h2>Contatos</h2>
8  <form method="post">
9      <table class="table">
10         <thead>
11             <tr>
12                 <th>ID</th>
13                 <th>Nome</th>
14             </tr>
15         </thead>
16         <tbody>
17             @foreach (var contato in Model.Cientes)
18             {
19                 <tr>
20                     <td>@contato.Id</td>
21                     <td>@contato.Nome</td>
22                     <td>
23                         <a asp-page="./Edit" asp-route-id="@contato.Id">Editar</a>
24                         <button type="submit" asp-page-handler="delete" asp-route-id="@contato.Id">Deletar</button>
25                     </td>
26                 </tr>
27             }
28         </tbody>
29     </table>
30     <a asp-page="./Create">Criar Contato</a>
31 </form>
```

Nesta view **Index.cshtml** definimos o **PageModel** como **IndexModel** e a propriedade **Cientes** que será definida no code-behind onde estamos exibindo o Id e Nome do Cliente.

Para editar e/ou deletar usamos o **asp-route-id** para passar o Id do cliente para a página.

A seguir inclua o código abaixo no arquivo code-behind **Index.cshtml.cs** :

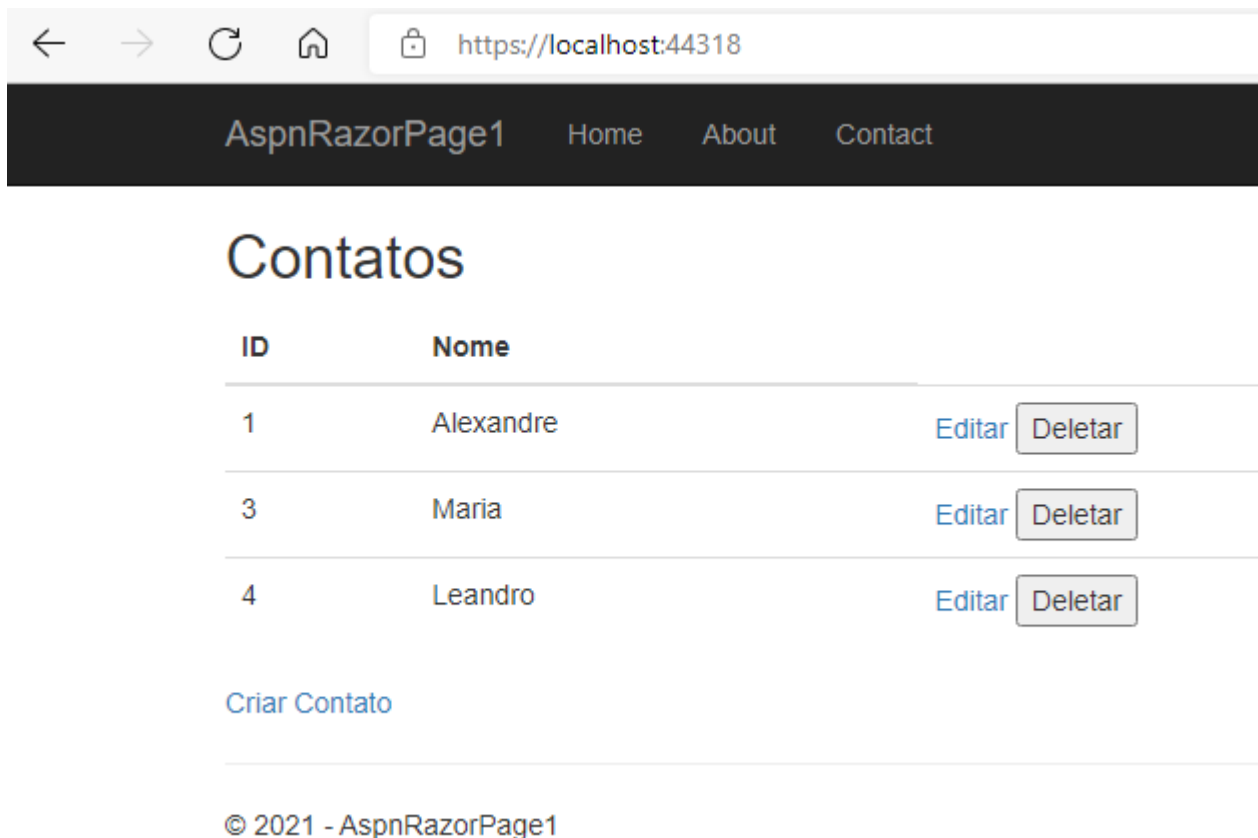
```
Index.cshtml.cs  X Index.cshtml  Startup.cs  AppDbContext.cs  Cliente.cs  Create.cshtml.cs  C
AspnRazorPage1  AspnRazorPage1.Pages.IndexModel

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using AspnRazorPage1.Models;
6  using Microsoft.AspNetCore.Mvc;
7  using Microsoft.AspNetCore.Mvc.RazorPages;
8  using Microsoft.EntityFrameworkCore;
9
10 namespace AspnRazorPage1.Pages
11 {
12     public class IndexModel : PageModel
13     {
14         private readonly AppDbContext _db;
15         public IndexModel(AppDbContext db)
16         {
17             _db = db;
18         }
19         public IList<Cliente> Clientes { get; private set; }
20         public async Task OnGetAsync()
21         {
22             Clientes = await _db.Clientes.AsNoTracking().ToListAsync();
23         }
24         public async Task<IActionResult> OnPostDeleteAsync(int id)
25         {
26             var contact = await _db.Clientes.FindAsync(id);
27             if (contact != null)
28             {
29                 _db.Clientes.Remove(contact);
30                 await _db.SaveChangesAsync();
31             }
32             return RedirectToPage();
33         }
34     }
35 }
```

No arquivo code-behind injetamos uma instância do nosso contexto no construtor e definimos os métodos `OnGetAsync()` para exibir os clientes e `OnPostDeleteAsync()` que vai localizar o cliente e deletar suas informações.

Executando o nosso projeto neste momento teremos o seguinte resultado:

Exibição da view Index.cshtml



Referências:

ASP .NET Core - Apresentando Razor Pages (macoratti.net)

http://www.macoratti.net/18/02/aspcore_rzpg1.htm

ASP .NET Core - Apresentando Razor Pages - II (macoratti.net)

http://www.macoratti.net/18/02/aspcore_rzpg2.htm

ASP.NET Core - Como funciona o Model Binding (macoratti.net)

http://www.macoratti.net/19/06/aspc_fmodbind1.htm

ASP .NET - Usando Tag Helpers em formulários (macoratti.net)

http://www.macoratti.net/17/04/aspn_taghlp1.htm