

.NET Core

Público Alvo

Programadores que têm como objetivo aprimorar seus conhecimentos na Linguagem C# utilizando o Framework .NET Core.

Pré-Requisitos

Conhecimentos de Lógica de Programação, Banco de Dados e a base da Linguagem C#.

Índice

.NET CORE.....	I
PÚBLICO ALVO	I
PRÉ-REQUISITOS	I
ÍNDICE.....	1
COPYRIGHT	3
EQUIPE.....	3
HISTÓRICO DAS EDIÇÕES.....	3
CAPÍTULO 01 – HTML	5
O QUE É O HTML	5
O QUE É HTML 5.....	5
COMO FUNCIONA O HTML 5	5
ELEMENTOS DE SCRIPT.....	7
CRIANDO FORMULÁRIO	8
CAPÍTULO 02 – CSS.....	16
O QUE É CSS	16
ESTILIZANDO NOSSA PÁGINA.....	17
CAPÍTULO 03 – JAVASCRIPT	20
O QUE É	20
CRIANDO A INTERAÇÃO COM O SITE	22
CAPÍTULO 04 – CRIANDO PADRÃO ASP.CORE COM MVC	24
PADRÃO ASP.CORE COM MVC	24
CRIANDO A PRIMEIRA PÁGINA	30
CHAMANDO A VIEW PELO CONTROLLER.....	32
CRIANDO ESTRUTURA DO ENTITY FRAMEWORK CORE	35
COMANDOS DO ENTITY FRAMEWORK CORE.....	38
CRIANDO O CRUD COM O ENTITY FRAMEWORK CORE.....	40

	Anotações

	Anotações

Copyright

As informações contidas neste material se referem ao curso de **ASP.NET Core** e estão sujeitas as alterações sem comunicação prévia, não representando um compromisso por parte do autor em atualização automática de futuras versões.

A **Apex** não será responsável por quaisquer erros ou por danos acidentais ou consequenciais relacionados com o fornecimento, desempenho, ou uso desta apostila ou os exemplos contidos aqui.

Os exemplos de empresas, organizações, produtos, nomes de domínio, endereços de e-mail, logotipos, pessoas, lugares e eventos aqui representados são fictícios. Nenhuma associação a empresas, organizações, produtos, nomes de domínio, endereços de e-mail, logotipos, pessoas, lugares ou eventos reais é intencional ou deve ser inferida.

A reprodução, adaptação, ou tradução deste manual mesmo que parcial, para qualquer finalidade é proibida sem autorização prévia por escrito da **Apex**, exceto as permitidas sob as leis de direito autoral.

Equipe

Conteúdos

- Gustavo Rosauro

Diagramação

- Fernanda Pereira

Revisão

- Fernanda Pereira

Histórico das Edições

Edição	Idioma	Edição
1ª	Português	Julho de 2019

	Anotações

	Anotações



O que é o HTML

HTML é a sigla para **Hypertext Markup Language** (Linguagem de marcação de hipertexto). Ao contrário do que muitos pensam, HTML não é uma linguagem de programação, mas sim um conjunto de tags para marcação de texto. Esse conjunto de tags é interpretado pelos navegadores para a visualização de um documento através da web.

O que é HTML 5

HTML 5 foi o nome dado à quinta versão da especificação do html, mas também é um termo utilizado entre as equipes de desenvolvimento para projetos que utilizam as tecnologias HTML CSS3 e Javascript. Isso se deve porque juntamente com a especificação do HTML 5 foram incluídas novas funcionalidades para manipulação de documentos html através do javascript.

Como funciona o HTML 5

A imagem abaixo apresenta a marcação básica de um documento HTML.

```
1  <!DOCTYPE html>
2  <html lang="en">
3      <head>
4          <title>Olá HTML</title>
5          <meta charset="UTF-8">
6      </head>
7      <body>
8          Este é um documento html
9      </body>
10 </html>
```

	Anotações

<!DOCTYPE html>

Esta marcação indica que o documento a seguir segue a especificação html da versão 5, ao contrário das versões anteriores, o doctype para o html 5 é pequeno.

<html>

Indica a abertura do documento html. Todo o conteúdo do documento deve ficar entre a tag de abertura <html> e a tag de fechamento </html>. A tag html possui o atributo lang que é utilizado para indicar qual o idioma utilizado no documento.

<head>

Indica a abertura do cabeçalho do documento. Normalmente o conteúdo adicionado dentro do cabeçalho é utilizado para fins de passar informações ao browser sobre o conteúdo que virá no documento. As informações do head não são visíveis diretamente no documento, mas são utilizadas pelo browser e por robôs para a indexação do documento. O head deve ser encerrado com sua tag de fechamento </head>.

<title></title>

Utilizada para a demarcação do título do documento. O conteúdo colocado entre a tag title não é renderizado no corpo do documento, mas é utilizado pelo navegador para informar o título da página em sua aba superior.

<meta charset="UTF-8">

Tag de meta dados utilizada para definir qual o tipo de encode foi utilizado pelo documento no momento de sua criação.

<body>

Indica o início do corpo do documento. Todo o conteúdo que será apresentado ao usuário deve estar contido entre as tags <body> e </body>

</html>

Indica o fim do documento.

Tipos de Tags HTML

A especificação HTML nos apresenta várias tags/elementos que são utilizados com os mais variados propósitos, entretanto, podemos agrupar essas tags/elementos em alguns grupos distintos:

Elementos Containers

São tags/elementos utilizados para agrupar outros elementos de forma lógica, estes elementos normalmente delimitam partes do documento e tem como característica a quebra de linha antes e após a sua demarcação.

	Anotações

Exemplos:

```
<p></p>
```

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

Elementos in-line

São tags/elementos utilizados normalmente para a formatação do texto, aplicando ao documento certo grau de estilização. São denominados in-line por não apresentarem quebra de linha no documento, não alterando seu direcionamento.

Exemplos:

```
<strong></strong>
```

```
<span></span>
```

Elementos de imagem e multimídia

São elementos utilizados para a inclusão de imagem, som e vídeo dentro de um documento html.

Exemplos:

```

```

```
<audio src="http://developer.mozilla.org/@api/deki/files/2926/=AudioTest_(1).ogg" autoplay>
```

O seu navegador não suporta o elemento `<code>audio</code>`.

```
</audio>
```

Elementos de Script

São utilizados para a inclusão de scripts ao documento para posterior processamento do navegador.

Exemplos:

```
<script></script>
```

```
<canvas></canvas>
```

Elementos de Meta Dados

São utilizados para adicionarem informações referente ao documento e são utilizados normalmente pelo navegador e por robôs de indexação de páginas.

Exemplo:

```
<meta charset="UTF-8">
```

	Anotações

Criando Formulário

Elemento Form

A tag **<form>** é utilizada para demarcar o conteúdo de um formulário. Este elemento possui dois atributos importantes que são:

- action: indica qual é a ação a ser processada no servidor.
- method: indica a forma de envio dos dados. Os valores válidos para estes atributos são “get” ou “post”.

Exemplo:

```
<!DOCTYPE html>

<html lang="pt">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Exemplo tag form</title>

</head>

<body>

<form action="salvar" method="post">

</form>

</body>

</html>
```

Elemento Fieldset

A tag **<fieldset>** é um elemento utilizado para demarcar um agrupamento de dados no formulário.

Elemento Legend

A tag **<legend>** é o elemento utilizado para demarcar o título de um fieldset.

Elemento Label

A tag **<label>** é um elemento utilizado para demarcar o marcador de um campo do formulário.

Exemplo:

	Anotações

```

<!DOCTYPE html>

<html lang="pt">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Exemplo tag fieldset</title>

</head>

<body>

<form action="salvar" method="post">

<fieldset>

<legend>Dados Pessoais</legend>

<label>Nome</label>

<input type="text">

</fieldset>

</form>

</body>

</html>

```

Elemento input

A tag **<input>** é utilizado para apresentar um campo de entrada de dados de um formulário. A tag input é extremamente versátil podendo renderizar vários tipos de entrada de dados apenas alterando o valor do atributo type. As renderizações mais comuns são:

Input type:text

Apresenta um campo para entrada de dados do tipo texto.

Input type:email

Apresenta um campo para entrada de dados do tipo email.

Input type:checkbox

Apresenta um campo para entrada de dados do tipo checkbox (caixa de seleção).

Input type:radio

	Anotações

Apresenta um campo para entrada de dados do tipo radio(caixa de seleção).

Input type:number

Apresenta um campo para entrada de dados do tipo numérico.

Input type:password

Apresenta um campo para entrada de dados para senhas.

Input type:file

Apresenta um campo para seleção de arquivos para envio ao servidor.

Input type:reset

Apresenta um botão para retornar o conteúdo dos campos do formulário ao seus valores padrão.

Input type:submit

Apresenta um botão para enviar os dados formulário.

Input type:tel

Apresenta um campo para informar um número de telefone. Quebras de linha são removidas automaticamente do texto informado.

Input type:button

Apresenta um botão sem comportamento padrão.

Input type:hidden

Apresenta um controle não visível utilizado para enviar informações para o servidor.

Além do atributo type, o elemento input apresenta os seguintes atributos também muito utilizados:

name: define um nome para o campo. Este nome é utilizado pelo servidor para ler o valor do campo no formulário.

value: utilizado para definir um valor para o campo

placeholder: utilizado para apresentar um exemplo de texto a ser inserido no elemento.

required: utilizado para indicar se o valor do campo é obrigatório

readonly: utilizado para indicar se o valor do campo é somente para leitura, sem que o usuário possa editar esse valor.

	Anotações

Exemplo:

```
<!DOCTYPE html>

<html lang="pt">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Exemplo tag input</title>

</head>

<body>

<form action="salvar" method="post">

<fieldset>

<legend>Dados Pessoais</legend>

<input type="hidden" name="codigo"><br>

<label>Nome</label><br>

<input type="text"><br>

<label>Email</label><br>

<input type="email"><br>

<label>Telefone</label><br>

<input type="tel"><br>

<label>Idade</label><br>

<input type="number"><br>

<label>Senha</label><br>

<input type="password" required><br>

</fieldset>

<fieldset>

<legend>Benefícios</legend>

<label> <input type="checkbox">Unimed</label><br>
```

	Anotações

```

<label><input type="checkbox">Uniodonto</label><br>
</fieldset>
<fieldset>
<legend>Alimentação</legend>
<label> <input type="radio" name="alimentacao">Cartão Refeição</label><br>
<label><input type="radio" name="alimentacao">Cartão Alimentação</label><br>
</fieldset>
</form>
</body>
</html>

```

Elemento button

A tag **<button>** é utilizada para criar um botão clicável dentro do formulário.

Exemplo:

```

<!DOCTYPE html>
<html lang="pt">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Exemplo tag button</title>
</head>
<body>
<form action="salvar" method="post">
<fieldset>
<legend>Botoes</legend>
<button type="button" onclick="alert('Bem Vindo!')">Click</button>
<button type="submit" >Enviar</button>
</fieldset>

```

	Anotações

```
</form>

</body>

</html>
```

Elemento select

A tag **<select>** é utilizada para criar um elemento contendo um menu de opções. Este elemento normalmente é utilizado em conjunto com as tags **<option>** e **<optgroup>**.

Exemplo:

```
<!DOCTYPE html>

<html lang="pt">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Exemplo tag input</title>

</head>

<body>

<form action="salvar" method="post">

<fieldset>

<legend>Cidades</legend>

<select>

<optgroup label="Santa Catarina">

<option value="Blumenau" label="Blumenau"></option>

<option value="Florianópolis" label="Florianópolis"></option>

<option value="Criciúma" label="Criciúma"></option>

<option value="Lages" label="Lages"></option>

</optgroup>

<optgroup label="Rio Grande do Sul">

<option value="Porto Alegre" label="Porto Alegre"></option>

<option value="Canoas" label="Canoas"></option>
```

	Anotações

```

<option value="Guaíba" label="Guaíba"></option>

<option value="Santana do Livramento" label="Santana do Livramento"></option>

</optgroup>

</select>

</fieldset>

</form>

</body>

</html>

```

Elemento *textarea*

A tag **<textarea>** é utilizada para apresentar uma caixa de texto com tamanho grande.

Exemplo:

```

<!DOCTYPE html>

<html lang="pt">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Exemplo tag textarea</title>

</head>

<body>

<form action="salvar" method="post">

<fieldset>

<legend>Informações</legend>

<label >Comentários</label><br>

<textarea cols="40" rows="10"></textarea>

</fieldset>

</form>

</body>

```

	Anotações

</html>

Elemento progress

A tag **<progress>** é utilizada para apresentar uma barra de progresso.

Exemplo:

```
<!DOCTYPE html>

<html lang="pt">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Exemplo tag progress</title>

</head>

<body>

<form action="salvar" method="post">

<fieldset>

<legend>Informações</legend>

<label >andamento</label><br>

<progress value="70" max="100">70%</progress>

</fieldset>

</form>

</body>

</html>
```

Para informações mais aprofundadas sobre o HTML 5, consulte apostila disponível em sua área do aluno.

	Anotações



O que é CSS

Folhas de estilo em cascata ou no inglês “*Cascading Style sheet*” (CSS), é uma linguagem de marcação utilizada para a estilização de documentos html, e tem como objetivo separar a estilização de apresentação do documento da informação contida no próprio documento.

Com o uso do CSS podemos adicionar fontes, cores, margens, bordas entre outros, a elementos contidos dentro de documentos web.

Regra CSS

Regra CSS é a unidade básica utilizada para a estilização de um documento através do CSS. Uma regra CSS é composta por um seletor e uma ou mais declarações de propriedades/valores.

Exemplo:

```
seletor {  
  propriedade:valor;  
}
```

Onde:

Seletor: É o alvo da regra css, o componente que se deseja estilizar;

Propriedade: É o atributo/característica do componente que desejamos configurar;

Valor: É a qualificação do atributo;

Exemplo:

```
h1{  
  color: red;  
  Font-family: “Times New Roman”;  
}
```

Comentários de regras CSS

Em folhas de estilo em cascata podemos adicionar comentários para melhor organização e manutenção.

	Anotações

Para adicionarmos um comentário devemos utilizar os caracteres ‘/*’ para iniciar o bloco de comentários e em seguida os caracteres ‘*/’ para fechar o bloco de comentários.

Exemplo:

```
/* Este é um comentário válido em uma folha de estilos */
```

Estilizando nossa Página

Existem 4 formas básicas de vincularmos estilos a um documento html:

Estilos inline

Estilos inline são vinculados através da propriedade style pertencente a todos os componentes html. Os valores atribuídos à propriedade style serão utilizados para a estilização do componente.

Exemplo:

```
/* documento html */  
<h2 style="color:blue; background-color:#ccc"> Meu título</h2>
```

Estilos incorporados

Outra forma de adicionarmos estilos a um documento html é configurando-os dentro da tag <style></style>. Esta tag deve ser definida dentro do componente head.

Exemplo:

```
/* documento html */  
<!DOCTYPE html>  
<html>  
<head>  
  <style>  
    body{  
      margin: 0 0 0 0;  
    }  
    h2{  
      color:blue;  
      background-color:#ccc;  
    }  
  </style>  
</head>  
</html>
```

	Anotações

```

        .container{
            margin-left: 10px;
        }
    </style>
</head>
<body>
</body>
</html>

```

Folhas de estilo ligadas

Folhas de estilo ligadas são criadas dentro de arquivos com a extensão ‘.css’ e posteriormente ligadas ao documento onde serão utilizadas.

Exemplo:

```

/* documento meu-estilo.css */
body{
    margin: 0 0 0 0;
}
h2{
    color:blue;
    background-color:#ccc;
}

```

```

.container{
    margin-left: 10px;
}

```

```

/* documento html */
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="meu-estilo.css">
</head>
<body>
</body>
</html>

```

	Anotações

Folhas de estilo importadas

Folhas de estilo importadas são folhas de estilo criadas dentro de um arquivo com a extensão “.css” e posteriormente importados para o documento através da propriedade import dentro da tag style.

Exemplo:

```
/* documento meu-estilo.css */

body{
  margin: 0 0 0 0;
}

h2{
  color:blue;
  background-color:#ccc;
}

.container{
  margin-left: 10px;
}

/* documento html */

<!DOCTYPE html>
<html>
<head>
  <style>
    @import url('meu-estilo.css');
  </style>
</head>
<body>
</body>
</html>
```

Para informações mais aprofundadas sobre o CSS, consulte apostila disponível em sua área do aluno.

Anotações	



O que é

Javascript é uma linguagem interpretada baseada em Objetos e funções com tipagem dinâmica. JavaScript foi originalmente desenvolvido por [Brendan Eich](#), da [Netscape](#) sob o nome de *Mocha*, posteriormente teve seu nome mudado para *LiveScript* e por fim *JavaScript*.

LiveScript foi o nome oficial da linguagem quando foi lançada pela primeira vez na versão beta do navegador Netscape 2.0 em setembro de 1995, mas teve seu nome mudado em um anúncio conjunto com a [Sun Microsystems](#) em dezembro de 1995 quando foi implementado no navegador Netscape versão 2.0B3. (<https://pt.wikipedia.org/wiki/JavaScript>)

Javascript e a especificação ECMAScript

O JavaScript é padronizado pela [Ecma International](#) — a associação Europeia para a padronização de sistemas de comunicação e informação (antigamente ECMA era um acrônimo para European Computer Manufacturers Association) para entregar uma linguagem de programação padronizada, internacional baseada em Javascript.

Esta versão padronizada de Javascript, chamada ECMAScript, comporta-se da mesma forma em todas as aplicações que suportam o padrão. As empresas podem usar a linguagem de padrão aberto para desenvolver a sua implementação de Javascript. O padrão ECMAScript é documentado na especificação ECMA-262.

Como incorporar Javascript nas páginas html

Em Podemos incorporar javascript a uma página html basicamente de duas formas:

- Criar o Script diretamente na página html.

Exemplo:

	Anotações

```

1  <!doctype html>
2  <html>
3  <head></head>
4  <body>
5  <script>
6  document.write("<h1>Bem Vindo ao Javascript</h1>");
7  </script>
8  </body>
9  </html>

```

- Criar um arquivo de script separado da página e referenciá-lo posteriormente na página em que desejamos utilizá-lo. Primeiro precisamos criar um arquivo com a extensão .js.

Exemplo:

```

1  //app.js
2  document.write("<h1>Bem Vindo ao Javascript</h1>");

```

Em seguida importamos o arquivo javascript informando sua localização na propriedade src (source) da tag javascript.

```

1  <!doctype html>
2  <html>
3  <head></head>
4  <body>
5  <script src="app.js"></script>
6  </body>
7  </html>

```

	Anotações

Criando a interação com o Site

O que são eventos

Eventos são funções executadas pelo browser em decorrência de ações realizadas pelo usuário ou pelo próprio browser.

Principais eventos

Click - Disparado quando o mouse é pressionado sobre um elemento

Keydown - Disparado quando a tecla é pressionada para baixo.

Keyup - Disparado quando a tecla é liberada.

Keypress - Disparado após a tecla ser pressionada e liberada.

Focus - Disparado quando um elemento recebe o foco.

Blur - Disparado quando um elemento perde o foco.

MouseOver - Disparado quando o mouse está sobre o elemento

Input - Disparado quando um input, textarea ou select é alterado.

Load - Disparado quando o elemento é carregado.

Como se registrar para um evento

Para que possamos ser notificados sobre um determinado evento, devemos nos registrar no elemento passando uma função de notificação. Esse registro é feito através da função `addEventListener`.

Exemplo:

```
let listener= document.querySelector('button').addEventListener(  
    'click',function(evt){  
    Console.log('Click no botão')  
})
```

	Anotações

Como deixar de ouvir um evento

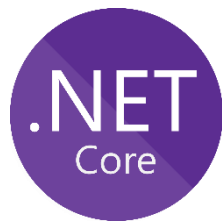
Para deixar de ouvir um evento, basta chamar a função `removeEventListener`, passando como parâmetros o nome do evento e o listener que se ligava ao evento.

Exemplo:

```
document.querySelector('button').removeEventListener('click',listener);
```

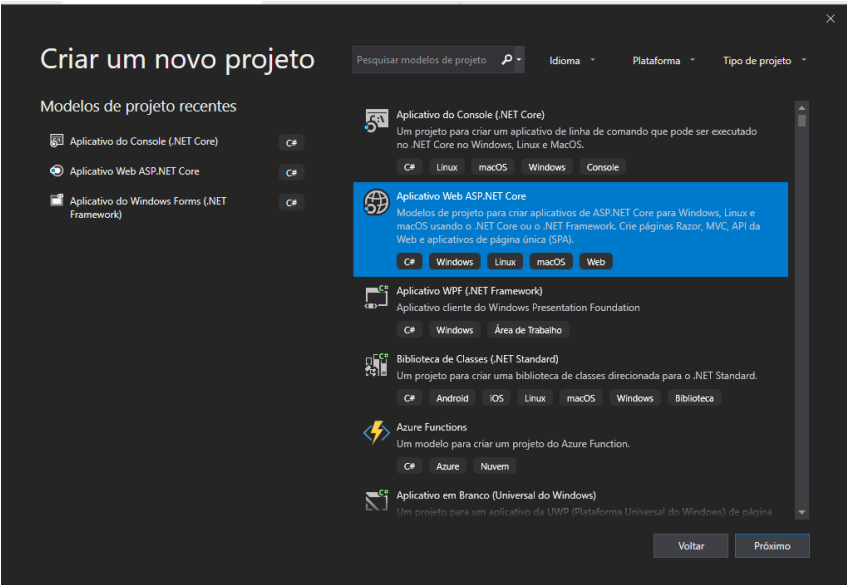
Para informações mais aprofundadas sobre o Javascript, consulte apostila disponível em sua área do aluno.

	Anotações



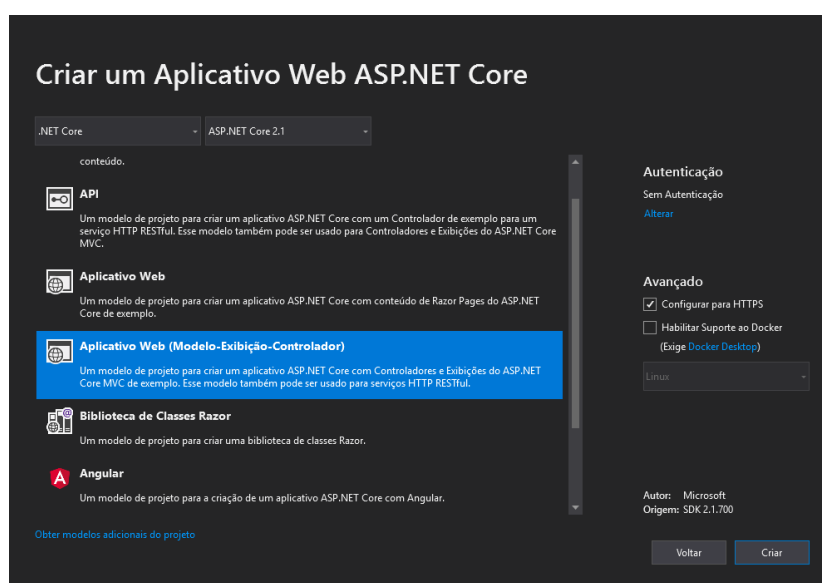
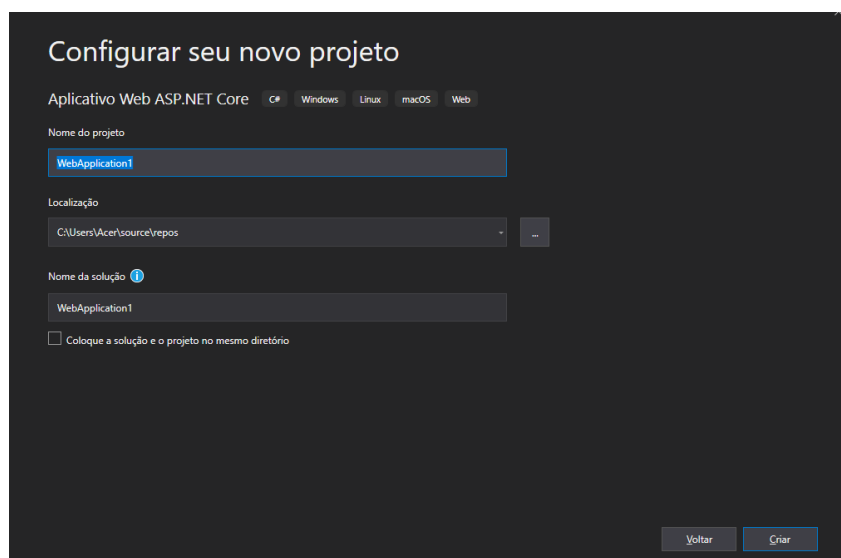
Padrão ASP.Core com MVC

Para iniciarmos, vamos escolher a opção Aplicativo Web ASP.CORE. Dentro dele existem alguns padrões de desenvolvimento, são eles o SPA Single Page Application e o MVC Model View Controller.



Vamos escolher o aplicativo Web Model View Controller e em seguida vamos clicar em próximo e dar nome à nossa aplicação, conforme imagem abaixo:

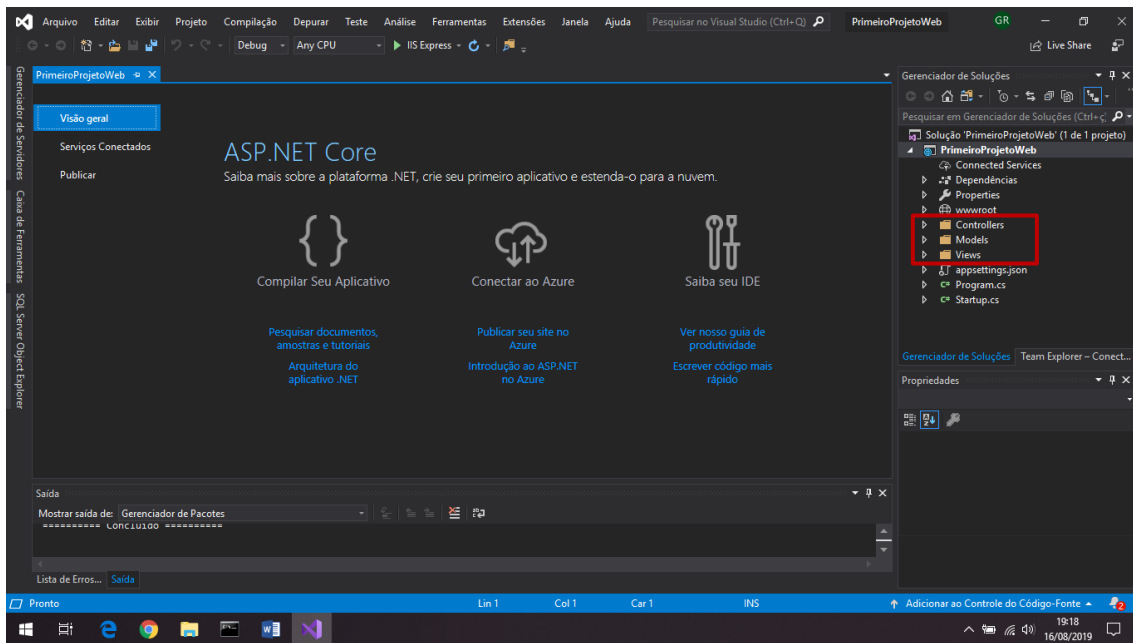
	Anotações



Após, vamos mandar ele criar o nosso projeto com o nosso tema escolhido.

Agora que ele carregou podemos ver os padrões na lateral direita com as pastas Model, View e Controller.

	Anotações



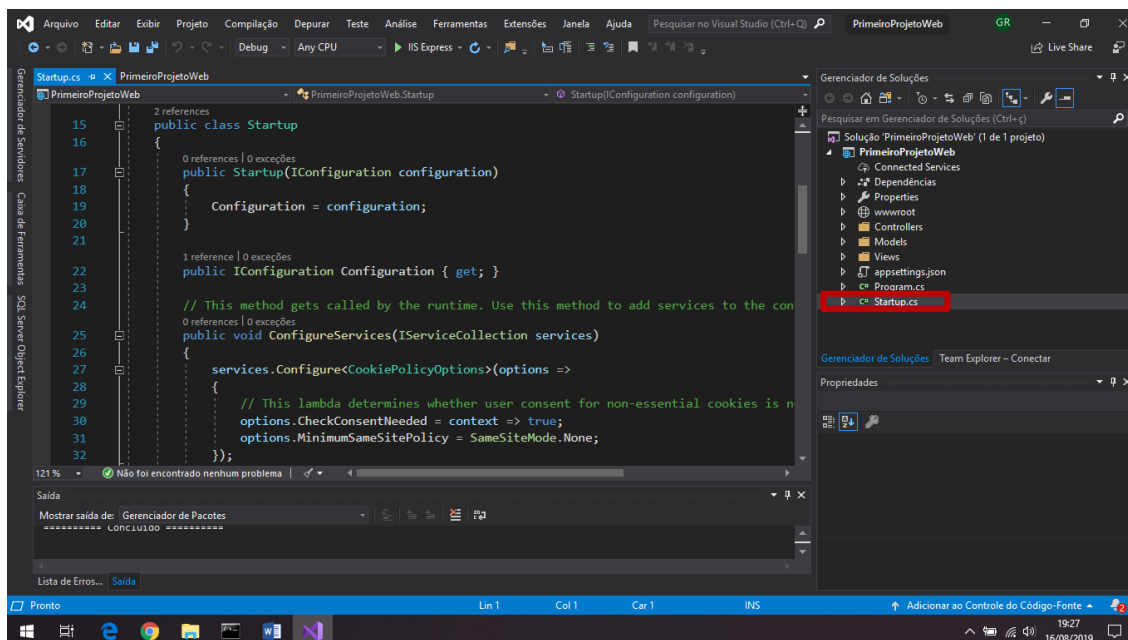
Essas três pastas seguem o padrão que iremos utilizar ao longo deste treinamento.

Model: que serão as nossas classes como modelos de dados;

Controllers: onde serão feitas as nossas regras de negócios e manipulação desses dados;

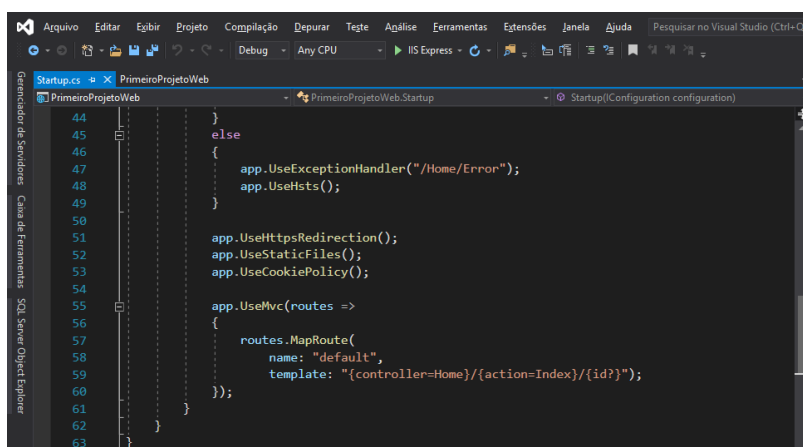
Views: que são as visualizações da nosso sistema, ou como já aprendemos até esse momento, serão as nossas páginas html, que é o local onde será feita a interação com o usuário.

	Anotações



Agora veja a imagem acima. Essa classe destacada em vermelho sempre irá rodar quando formos iniciar nosso projeto. Ela é responsável por gerenciar a configuração de inicialização do nosso projeto.

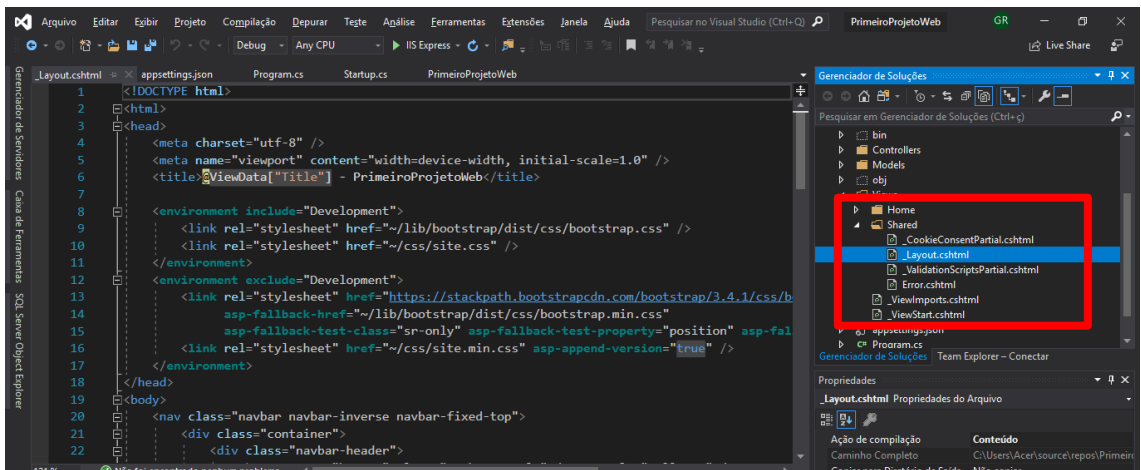
Dentro dela teremos um método de configurações de rotas. Quando carregar o nosso projeto ele irá chamar o nosso controller e a ação dentro dele para carregar o nosso primeiro método, que ficará responsável por mostrar a view, conforme imagem abaixo:



Dentro da nossa pasta Views existe uma pasta chamado Shared, esta pasta possui o layout padrão do nosso projeto.

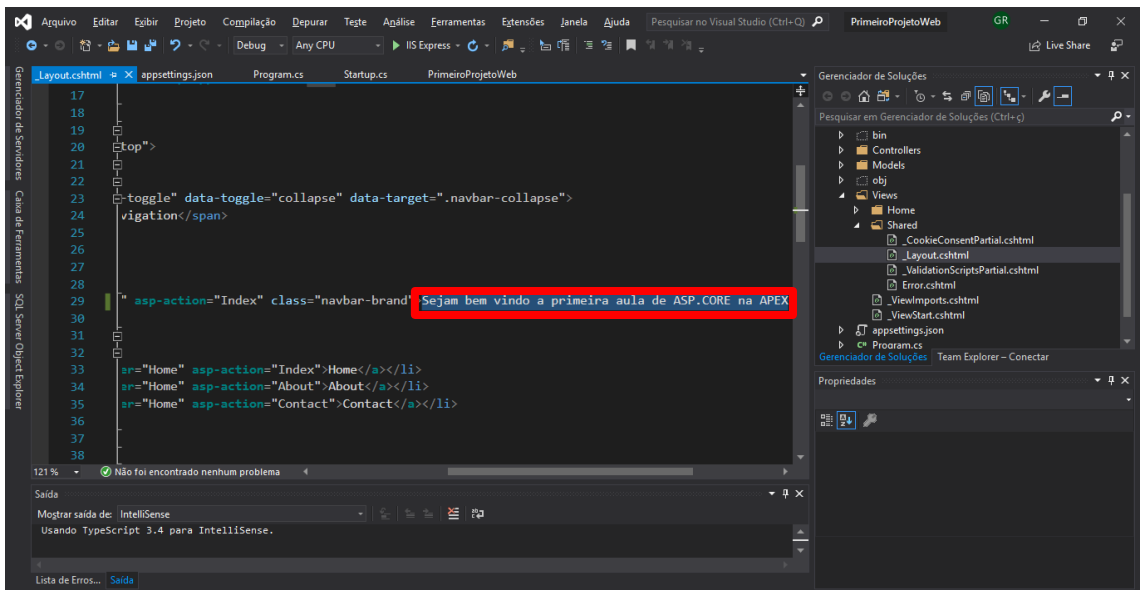
Esse layout funciona como uma Máster Page, onde tudo que nós criarmos será exibido dentro desse formato.

	Anotações



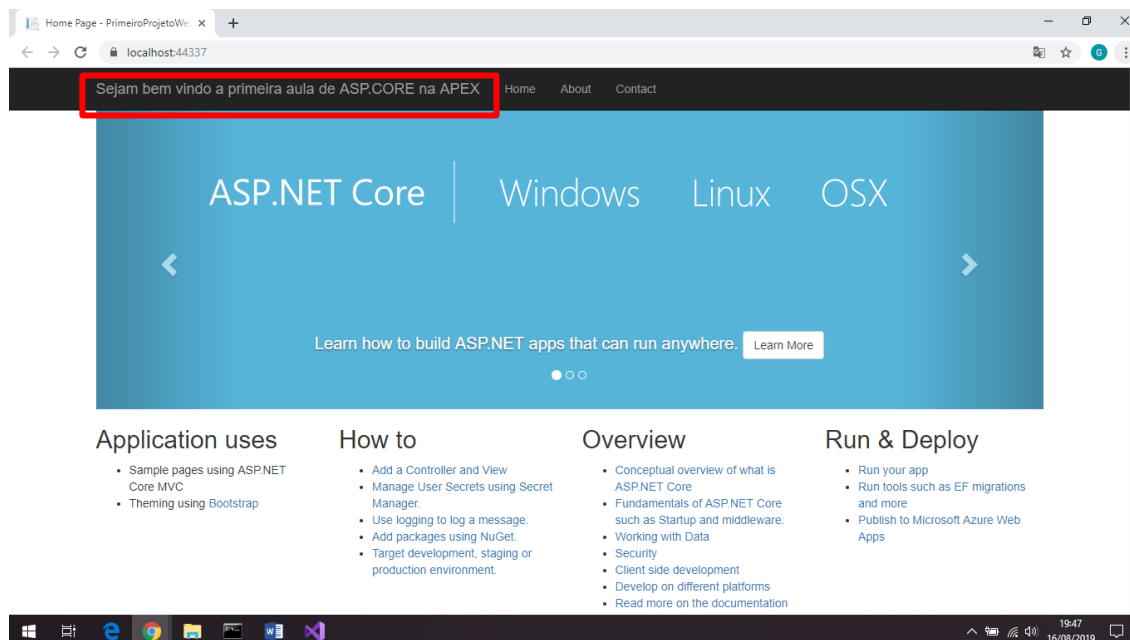
Então, como já temos um tema padrão, vamos fazer uma leve alteração e ver como ficará quando recarregarmos o nosso projeto.

Vamos dentro do nosso navbar-header e vamos alterar de primeiro projeto web para Sejam bem vindos à primeira aula de ASP.CORE na Apex.



Depois de realizada essa alteração, basta clicar no play e esperar abrir no nosso browser.

	Anotações



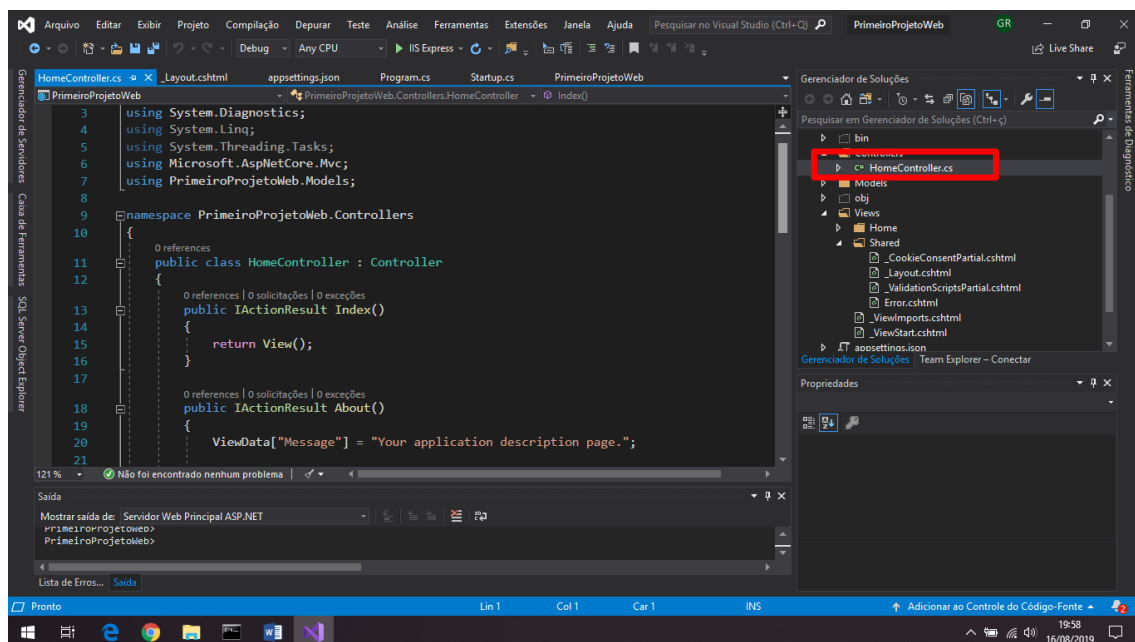
Dentro da Master Page você vai encontrar um cara chamdo `@RenderBody()`. Este cara irá renderizar as nossas páginas, seguindo esse padrão de layout.

	Anotações

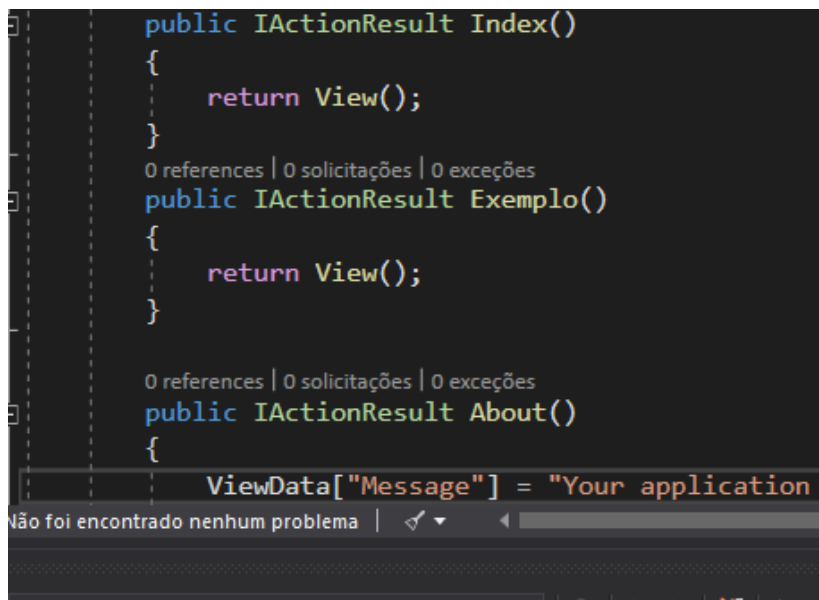
Criando a Primeira Página

Agora que entendemos um pouco melhor o conceito das páginas do nosso tema MVC, vamos lá nosso startup, onde estão as nossas rotas, para ver o controller e o método que ele está chamando.

Visto isso podemos acessar o nosso controller dentro da pasta controllers, cujo o nome é HomeController, conforme mostra a imagem abaixo.

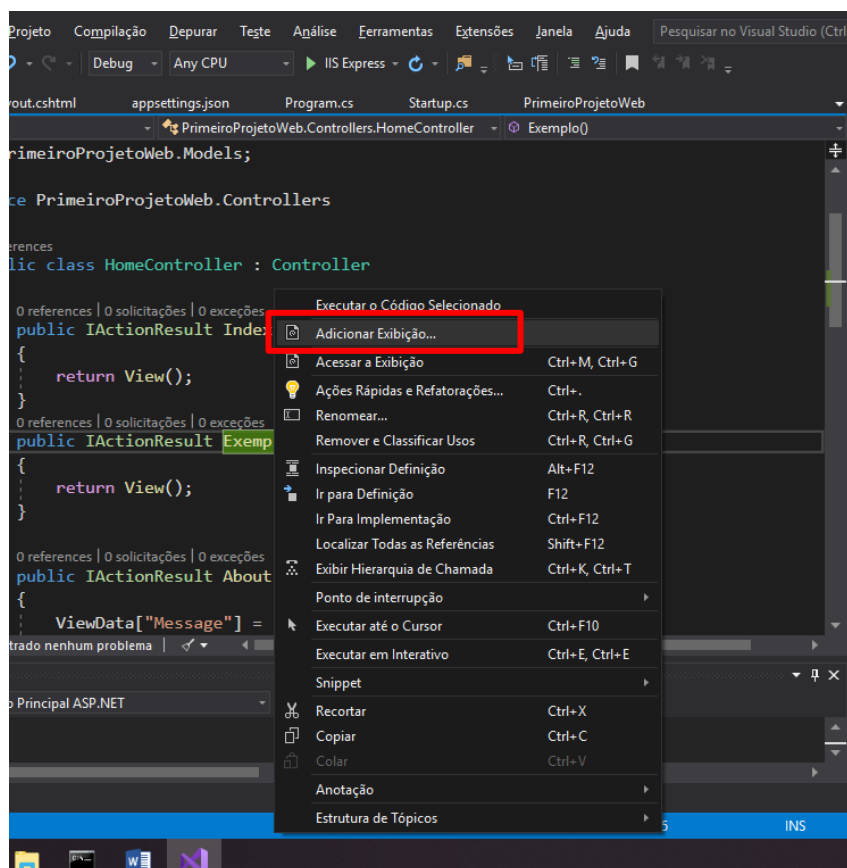


Como o controller é o detentor da nossa regra de negócios, então para que possamos chamar a nossa view precisamos criar um método que será responsável por essa chamada e que também terá o nome da nossa View. O método que possui a assinatura o IActionResult será responsável pela chamada da nossa View.

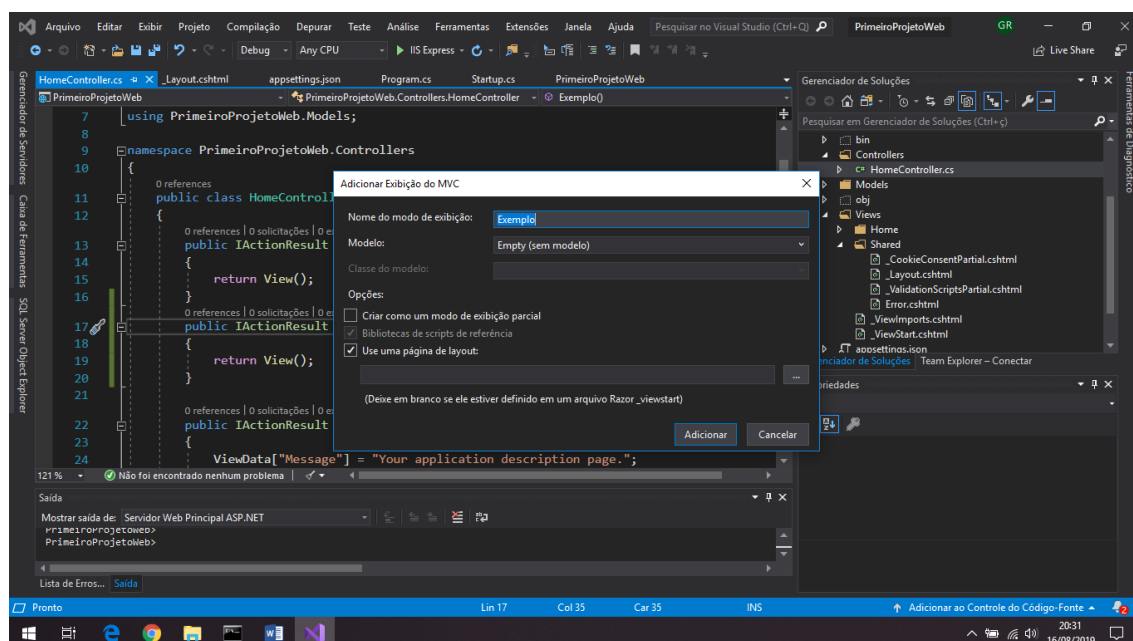


	Anotações

Depois de feita nossa assinatura, que é a chamada da View, clicamos com o botão direito em cima do nosso método exemplo e clicamos em adicionar exibição, conforme imagem abaixo:



Quando clicarmos, irá aparecer uma janela para confirmar. Basta clicar em adicionar e desta forma será criada a nossa view.

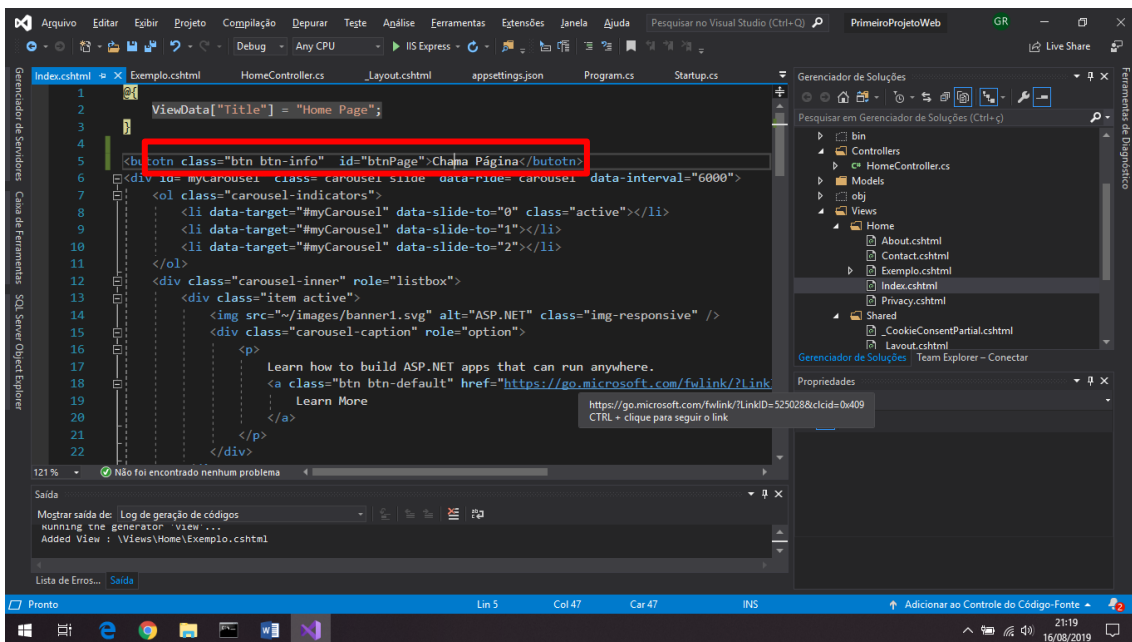


	Anotações

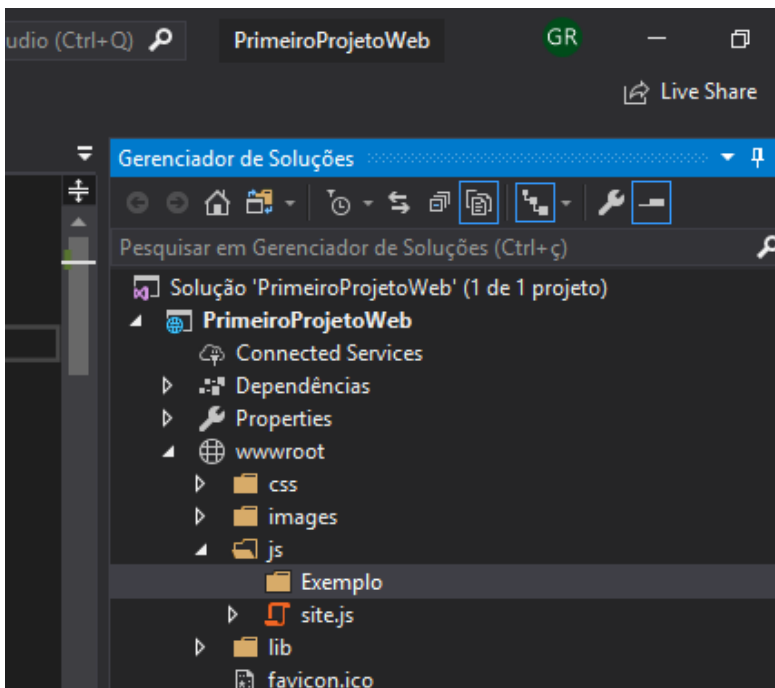
Chamando a View pelo Controller

Agora, com a nossa view pronta, vamos fazer uma chamada na nossa página principal. A página principal é o nosso Index, conforme vimos nas nossas rotas, e vamos criar um botão que irá chamar a nossa página.

Dentro da pasta Home abra a página Index.cshtml e crie um button com o id btnPage, conforme imagem abaixo:

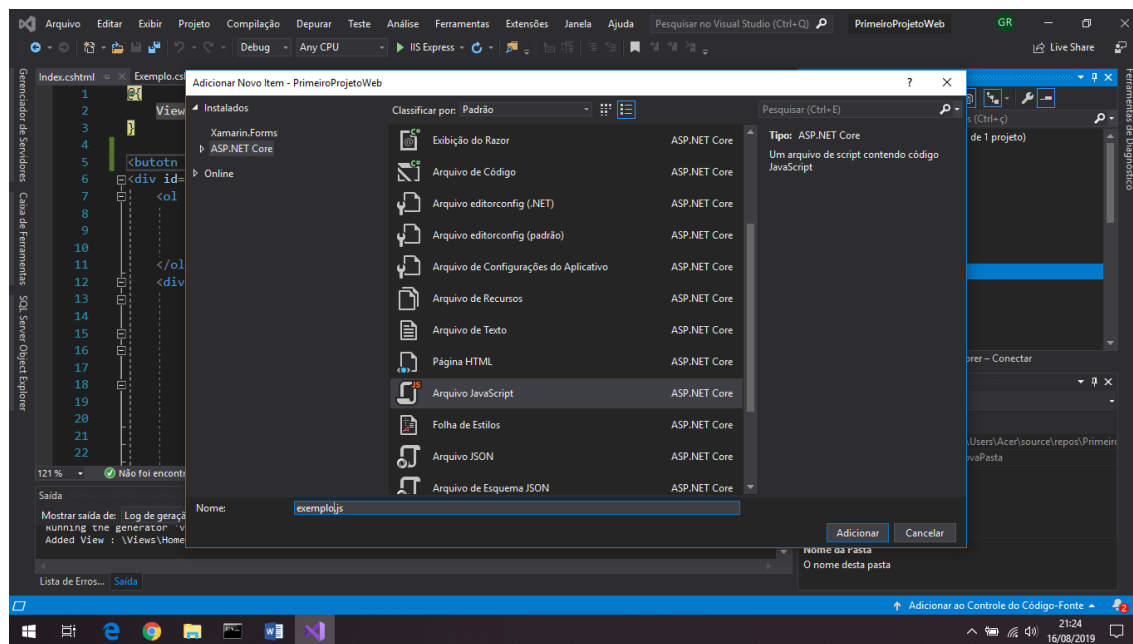


Após executar esta parte, vamos até a pasta js, que está dentro do nosso wwwroot, onde vamos criar uma pasta chamada Exemplo.

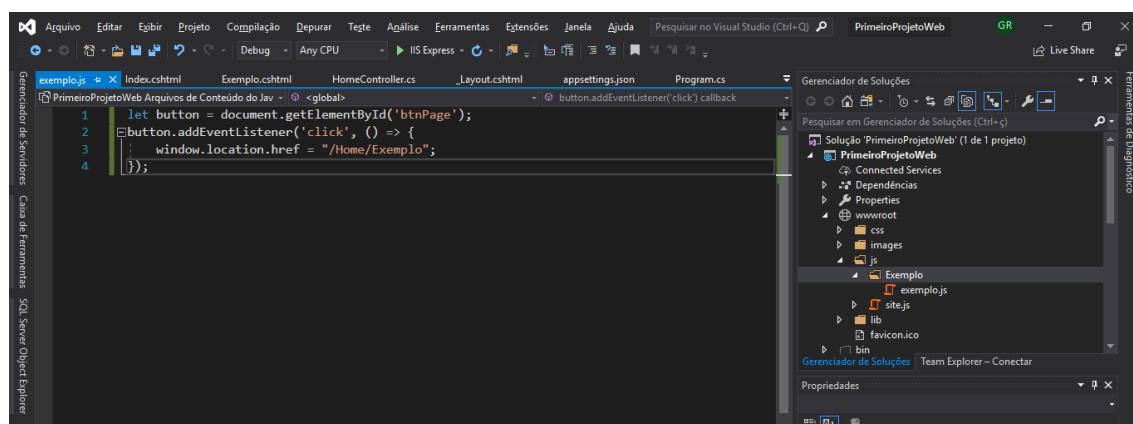


	Anotações

Criada essa pasta, vamos adicionar um arquivo javascript dentro dela. O arquivo terá o nome exemplo.js, conforme imagem abaixo:

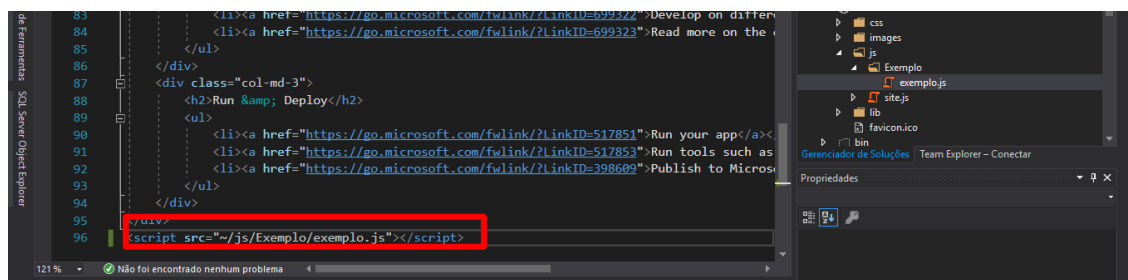


Basta clicar em adicionar e agora começamos a criar nosso arquivo.js. Dentro deste arquivo vamos executar um comando para chamar o nosso controller, que contém a nossa página, conforme imagem abaixo:

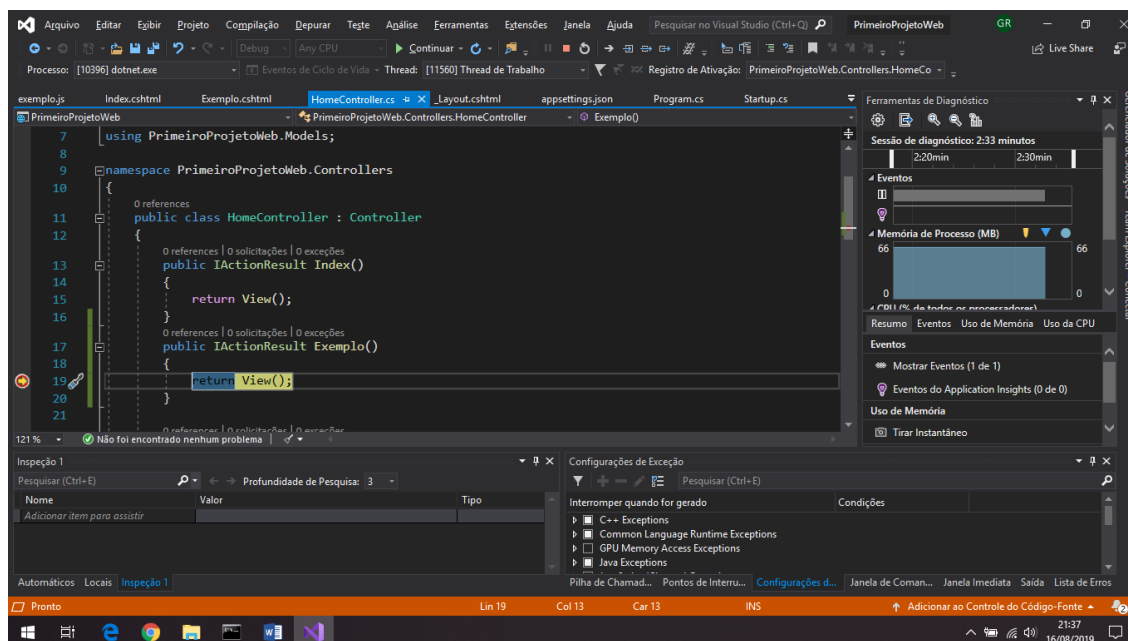
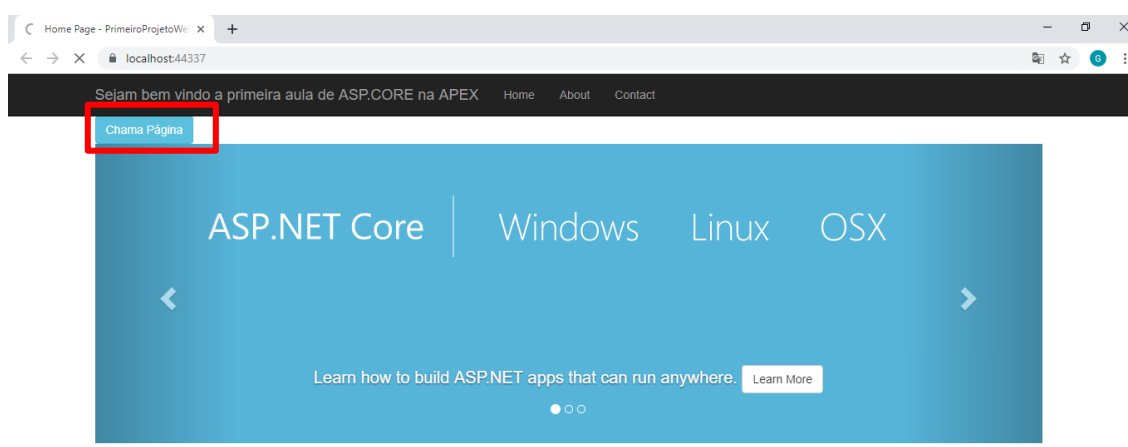


	Anotações

Para que a página possa ver esse nosso arquivo js, temos de chamar ele no script para carregar junto com o html

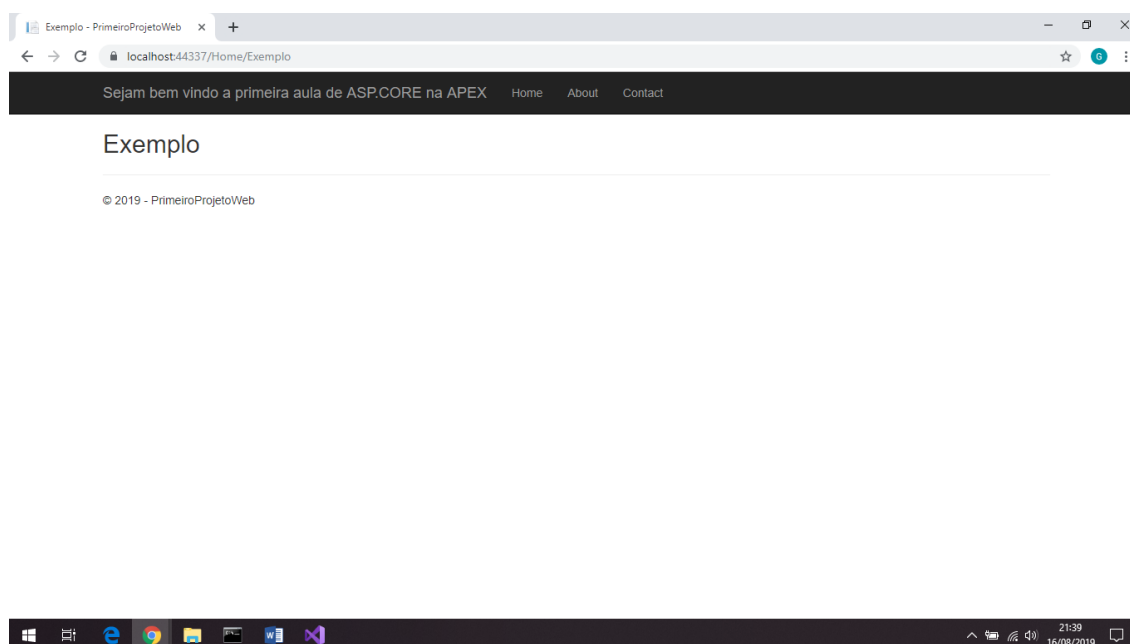


Agora vamos colocar um *break point* em nosso controller em nosso método exemplo, que chama a nossa página para que quando apertarmos o botão possamos acompanhar a execução.



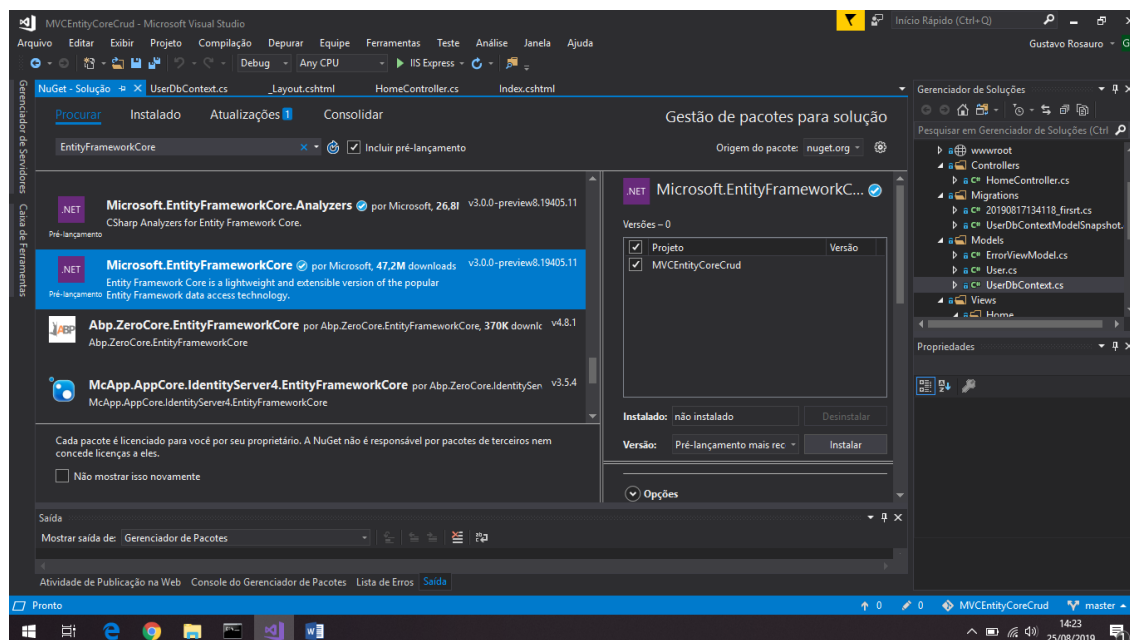
	Anotações

E por fim podemos ver que a nossa página foi exibida com sucesso conforme demonstra a imagem abaixo por meio do nosso Evento Clique quando apertamos o nosso botão.



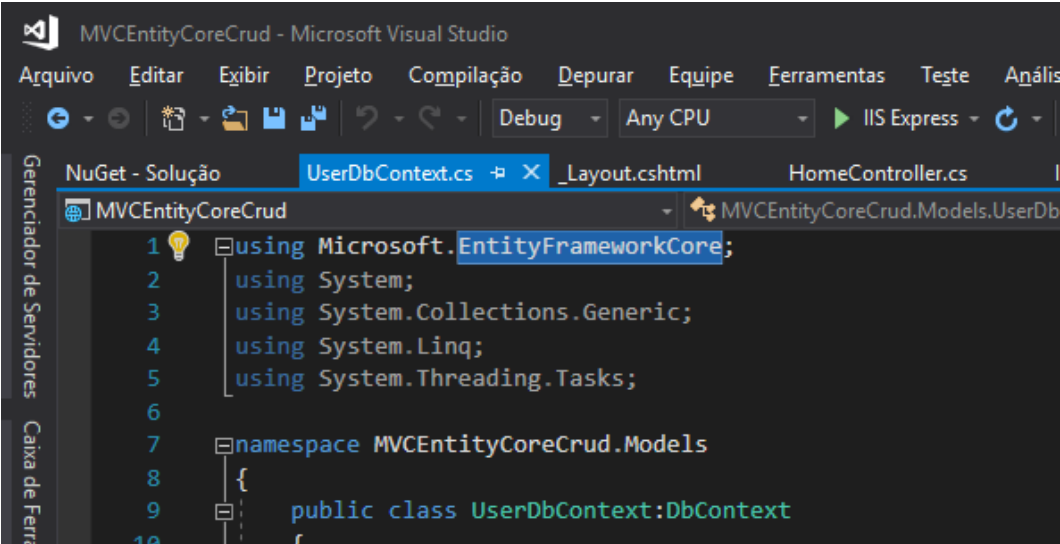
Criando estrutura do Entity Framework Core

Para que possamos criar a nossa Estrutura do Entity Framework Core temos que instalar o pacote dele. Faremos isso indo no nosso nuget e achando essa opção conforme imagem abaixo:



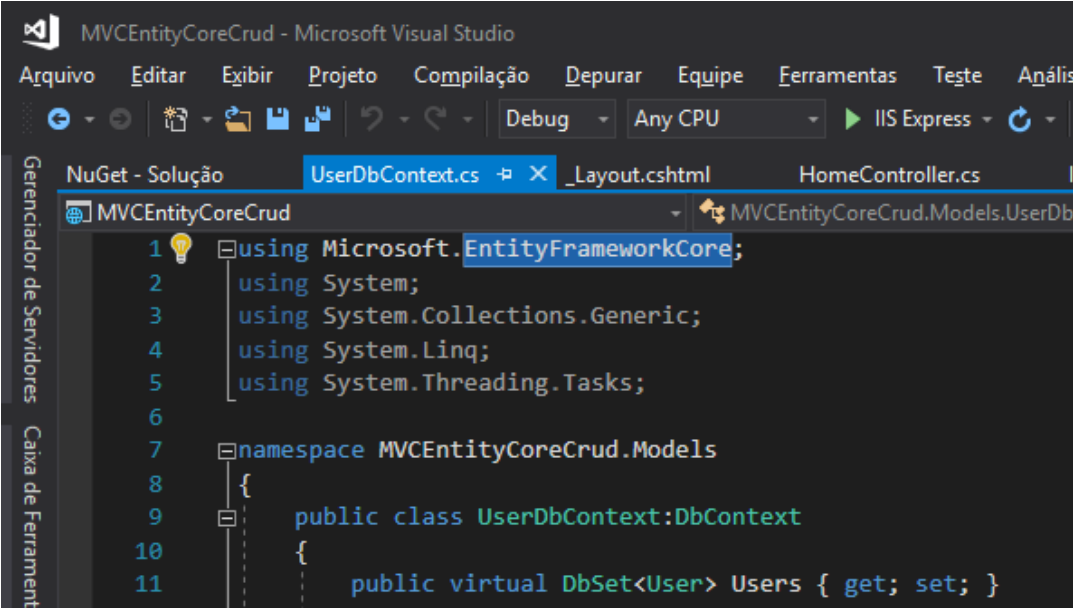
	Anotações

Após executada esta instalação, agora podemos criar uma classe com o nome exemplo UserDbContext.cs que herdará o DbContext dessa biblioteca:



```
1 using Microsoft.EntityFrameworkCore;
2 using System;
3 using System.Collections.Generic;
4 using System.Linq;
5 using System.Threading.Tasks;
6
7 namespace MVCEntityCoreCrud.Models
8 {
9     public class UserDbContext:DbContext
10     {
11     }
```

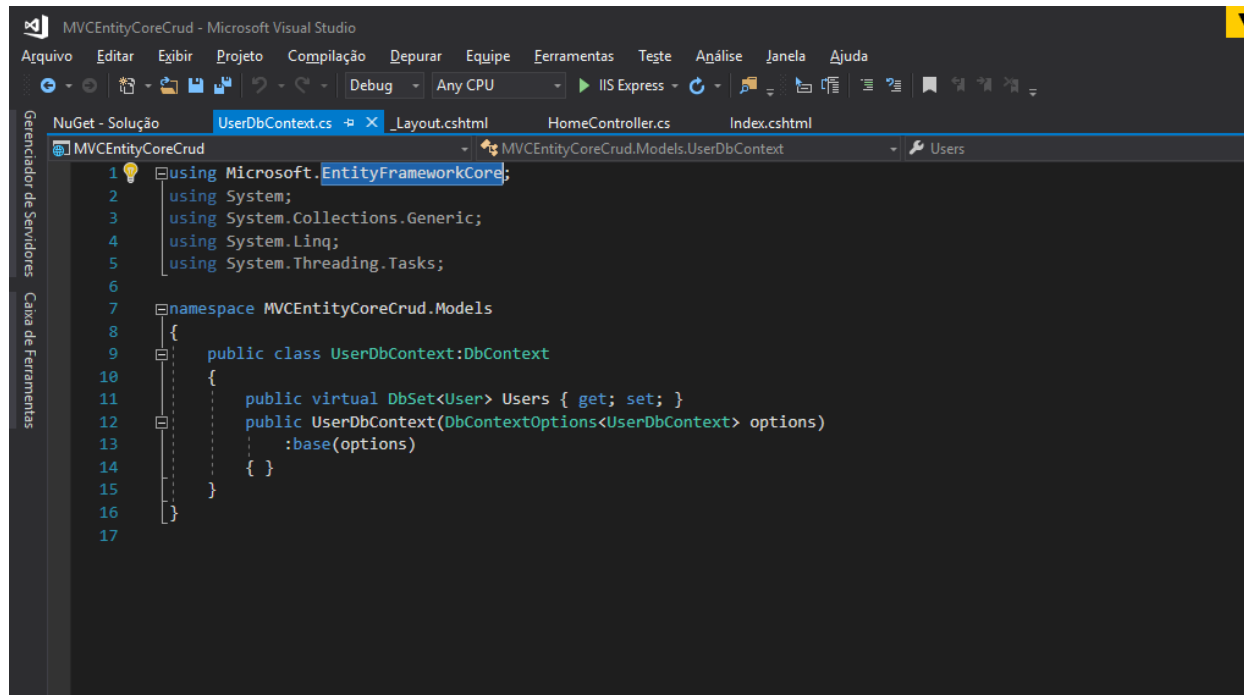
Feito isso, podemos utilizar o nosso DbSet, que nos permite utilizar o crud com nossa calsse comandos, por exemplo, que nos permite excluir, atualizar, deletar e inserir um registro em nossa tabela por meio dessa classe. Veja no exemplo abaixo:



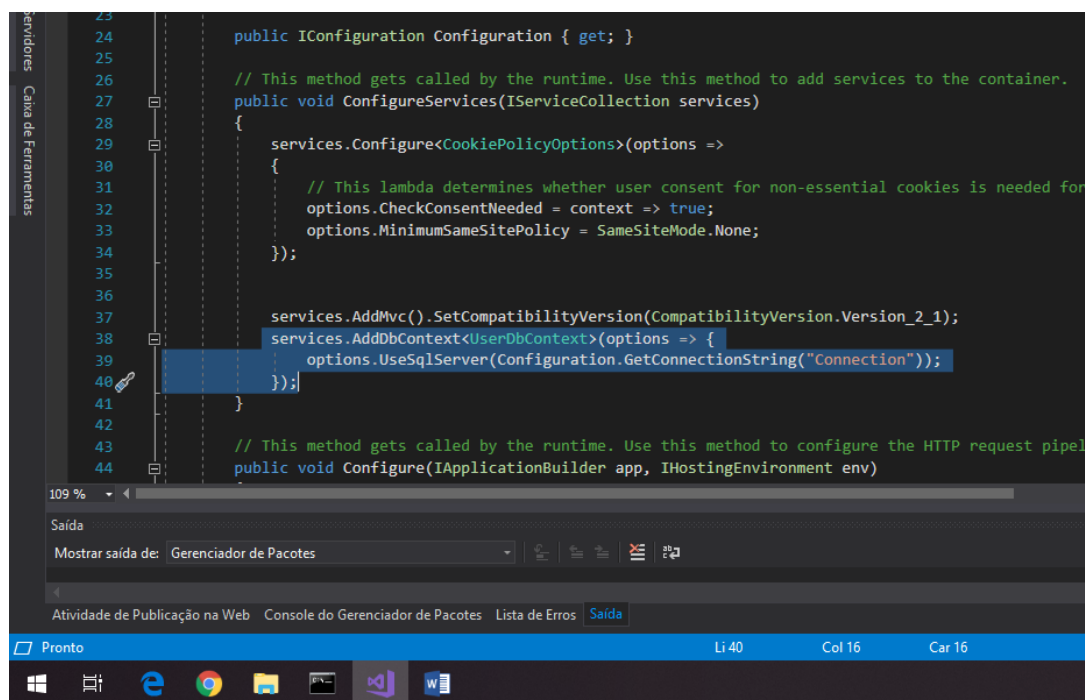
```
1 using Microsoft.EntityFrameworkCore;
2 using System;
3 using System.Collections.Generic;
4 using System.Linq;
5 using System.Threading.Tasks;
6
7 namespace MVCEntityCoreCrud.Models
8 {
9     public class UserDbContext:DbContext
10     {
11         public virtual DbSet<User> Users { get; set; }
```

	Anotações

Agora, para configurar nossa conexão, vamos utilizar uma propriedade chamada DbContextOptions. Com ela vamos fazer a ponte dessa classe com a nossa conexão do banco de dados.



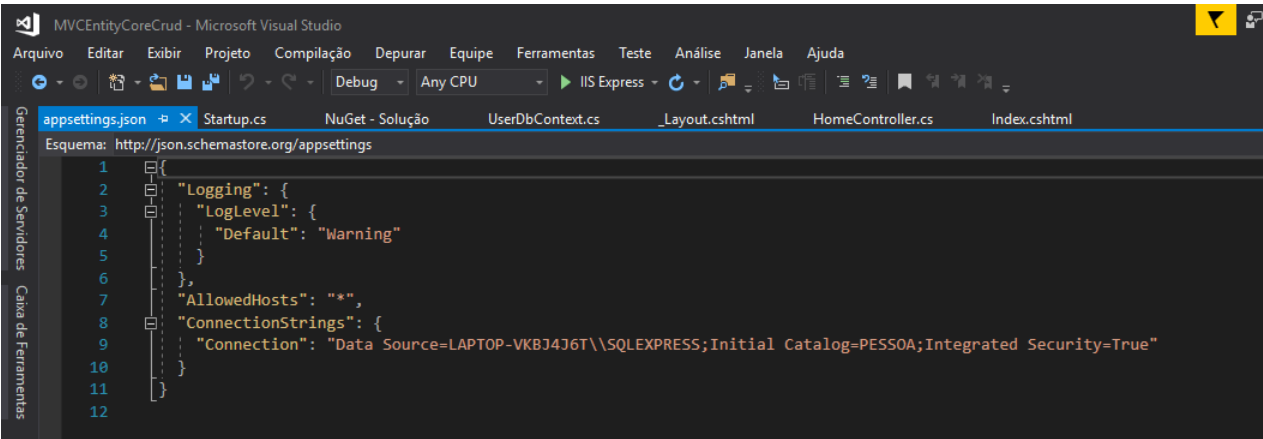
Feito essa parte, agora falta configurar a nossa configuração com o banco de dados na nossa classe Startup.cs Conforme exemplo abaixo:



Conforme visto na imagem acima, temos o connection dentro de nosso método getconnectionstring.

	Anotações

Temos agora que ir até nosso arquivo appsetting.json, onde vamos criar a nossa conexão, conforme exemplo abaixo:

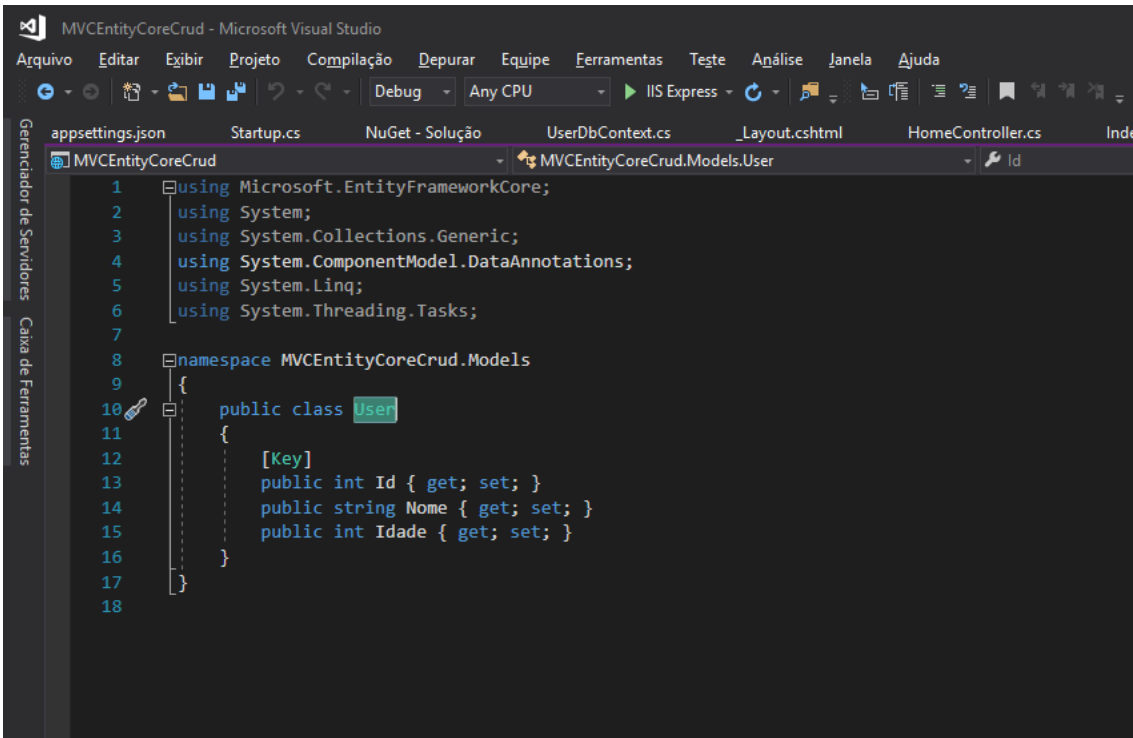


Feito isso agora podemos criar o nosso banco de dados por meio de alguns comandos que veremos no capítulo a seguir.

Comandos do Entity Framework Core

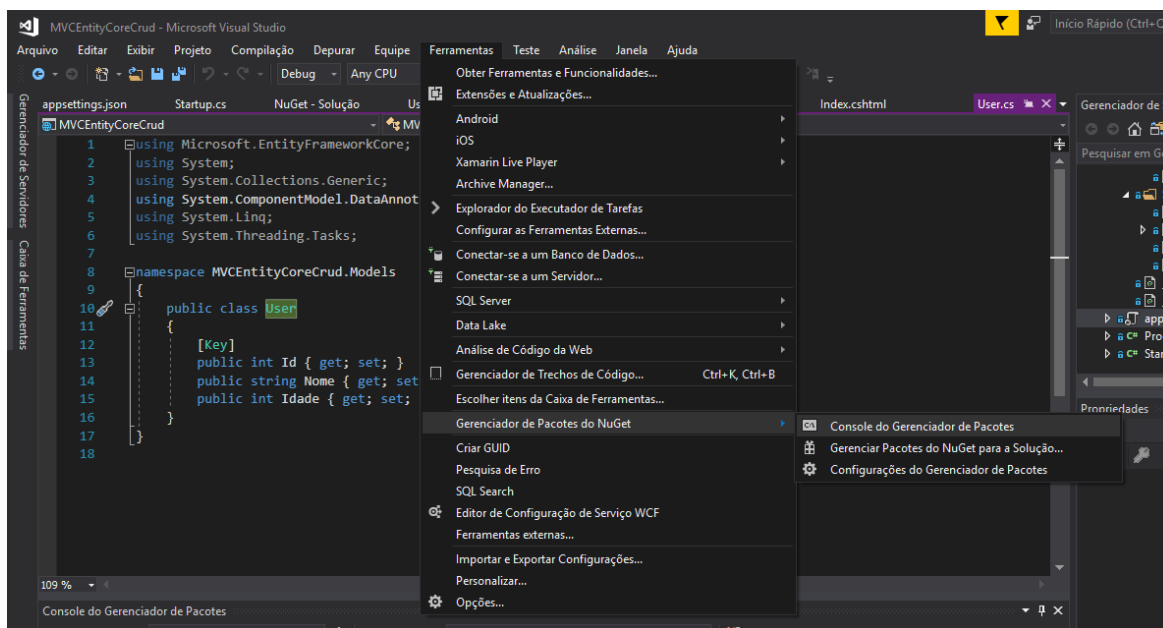
Após executar todo o processo que vimos no capítulo anterior vamos agora ver a nossa classe User.

Ao colocar os parâmetros da nossa classe User temos também que informar o key annotation, para que na hora de criar a tabela o nosso entity saiba que o campo id é uma chave primeira auto incremental, conforme imagem abaixo:

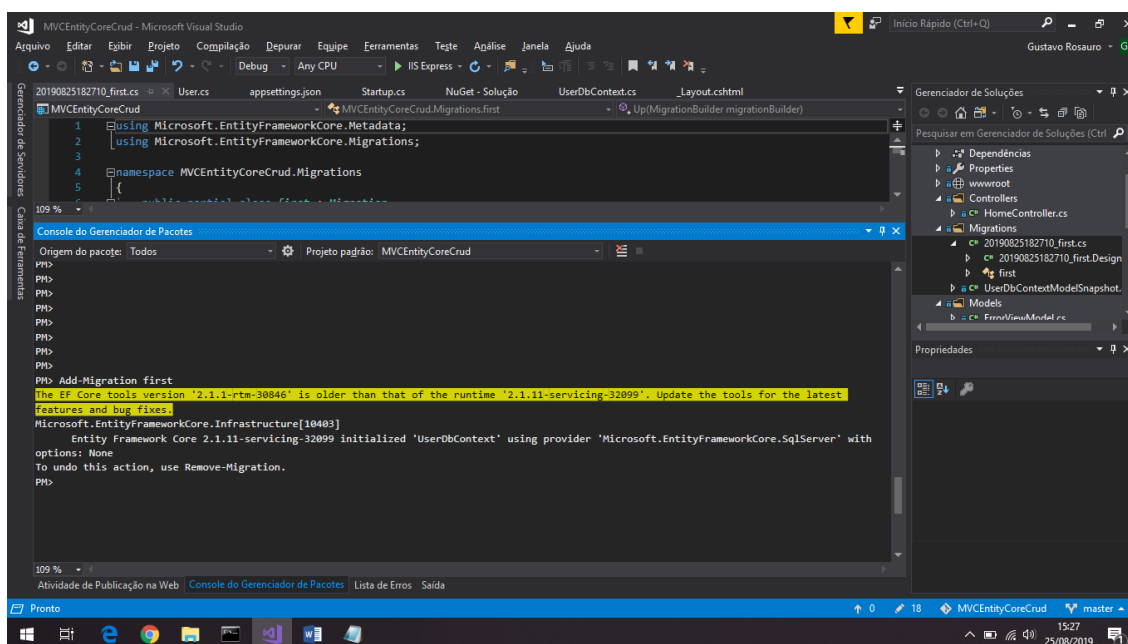


	Anotações

Após feito isso, vamos abrir o nosso Console Gerenciador de pacotes do nugget:

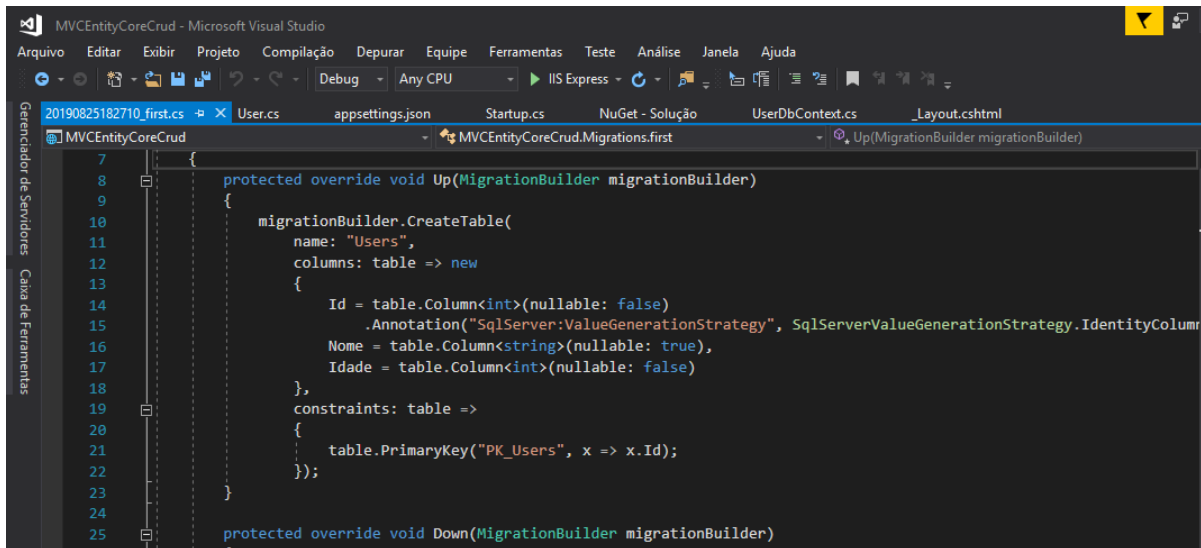


Para criarmos a tabela em nossa base temos que primeiro executar o comando Add-Migration para que ele gere nosso script:



	Anotações

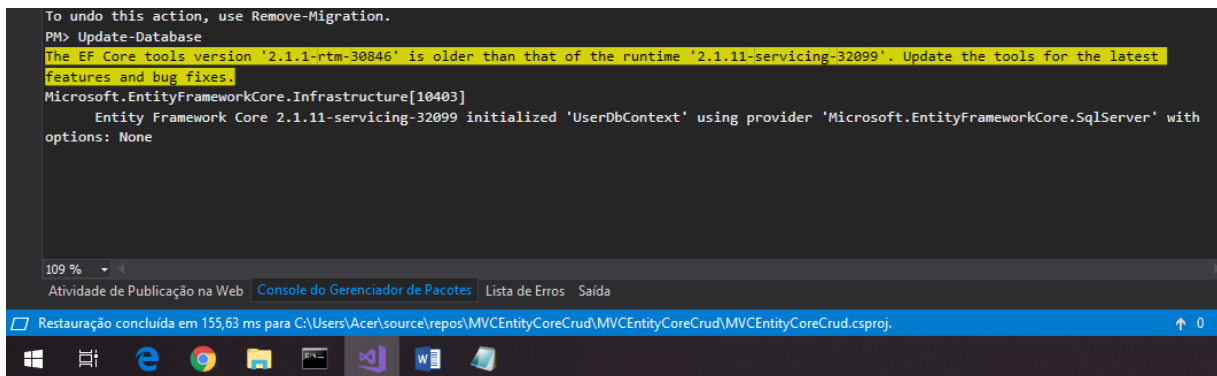
Após executar o comando você percebe que foi criado um diretório migrations. Podemos clicar nesse diretório e ver que foi gerado um comando para criar a nossa tabela baseada em nossa classe User.



```
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.CreateTable(
        name: "Users",
        columns: table => new
        {
            Id = table.Column<int>(nullable: false)
                .Annotation("SqlServer:ValueGenerationStrategy", SqlServerValueGenerationStrategy.IdentityColumn),
            Nome = table.Column<string>(nullable: true),
            Idade = table.Column<int>(nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Users", x => x.Id);
        });
}

protected override void Down(MigrationBuilder migrationBuilder)
```

Para criar agora essa estrutura em nossa base temos de executar outro comando Update-Database. Esse comando irá criar para nós uma tabela na base de dados passada na connectionstring:



```
PM> Update-Database
The EF Core tools version '2.1.1-rtm-30846' is older than that of the runtime '2.1.11-servicing-32099'. Update the tools for the latest features and bug fixes.
Microsoft.EntityFrameworkCore.Infrastructure[10403]
  Entity Framework Core 2.1.11-servicing-32099 initialized 'UserDbContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer' with options: None

109 %
Atividade de Publicação na Web Console do Gerenciador de Pacotes Lista de Erros Saída
Restauração concluída em 155,63 ms para C:\Users\Acer\source\repos\MVCEntityCoreCrud\MVCEntityCoreCrud\MVCEntityCoreCrud.csproj. 0
```

Feito isso nossa base receberá nossa classe.

Criando o Crud com o Entity Framework Core

Para manipular os dados em nossa base agora fica muito mais fácil, basta somente pegar as nossas classes UserDbContext, User e Instanciar no nosso Controller .

	Anotações

```

1  using System;
2  using System.Collections.Generic;
3  using System.Diagnostics;
4  using System.Linq;
5  using System.Threading.Tasks;
6  using Microsoft.AspNetCore.Mvc;
7  using MVCEntityCoreCrud.Models;
8
9  namespace MVCEntityCoreCrud.Controllers
10 {
11     public class HomeController : Controller
12     {
13         private readonly UserDbContext _context;
14         public HomeController(UserDbContext context)
15         {
16             _context = context;
17         }
18         User cliente = new User();

```

Feito isso, vamos para o nosso primeiro método inserir um novo usuário em nosso sistema.

```

23 [HttpPost]
24 public void InserirNovoUsuario(string nome, int idade)
25 {
26     cliente.Nome = nome;
27     cliente.Idade = idade;
28     _context.Users.Add(cliente);
29     _context.SaveChanges();
30 }

```

Como podemos ver por meio do nosso DbSet, lá do nosso UserDbContext conseguimos pegar o método Add que adiciona um novo usuário e o savechanges() sempre commita a nossa alteração, seguindo a mesma ideia do nosso sqltransaction.

Para nos mostrar os nossos usuários também é bem simples, basta instanciarmos o nosso método conforme imagem abaixo:

	Anotações

```
[HttpGet]
public IEnumerable<User> UsersList()
{
    return _context.Users;
}
```

Basta somente Retornamos nossos Users do DbSet.

Remover nosso usuário também é bem simples, basta somente passar o id para o método e por meio dele damos um find, encontramos qual é nosso usuário, removemos ele e salvamos nossa alteração. Conforme exemplo abaixo:

```
[HttpPost]
public void DeletarRegistro(int id)
{
    _context.Remove(_context.Users.Find(id));
    _context.SaveChanges();
}
```

E por último, precisamos alterar o usuário.

Para isso temos que usar o find, buscando pelo id também, e após encontrarmos passamos os novos valores para ele utilizando o state modified, e salvando nossa alteração conforme exemplo abaixo:

```
47
48
49 [HttpPut]
50 public void AlterarRegistro(int id, string nome, int idade)
51 {
52     var usuario = _context.Users.Find(id);
53     usuario.Nome = nome;
54     usuario.Idade = idade;
55     _context.Entry(usuario).State = Microsoft.EntityFrameworkCore.EntityState.Modified;
56     _context.SaveChanges();
57 }
```

	Anotações