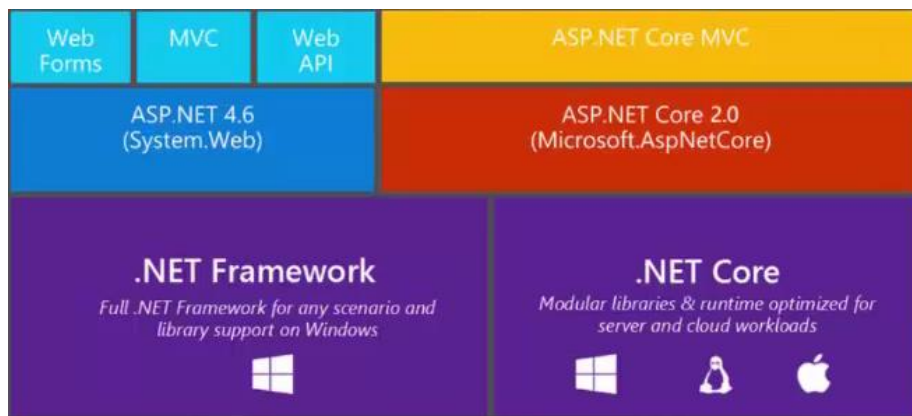


Os seguintes tópicos na aula de hoje:

- 1 - Visão geral do ASP.NET MVC,
- 2 - O padrão MVC (Model-View-Controller) ,
- 3 - Rotas,
- 4 - Razor,
- 5 - HTML Helper e TagHelper,
- 6 - Projeto exemplo sem banco dados

1 - Visão geral do ASP.NET MVC

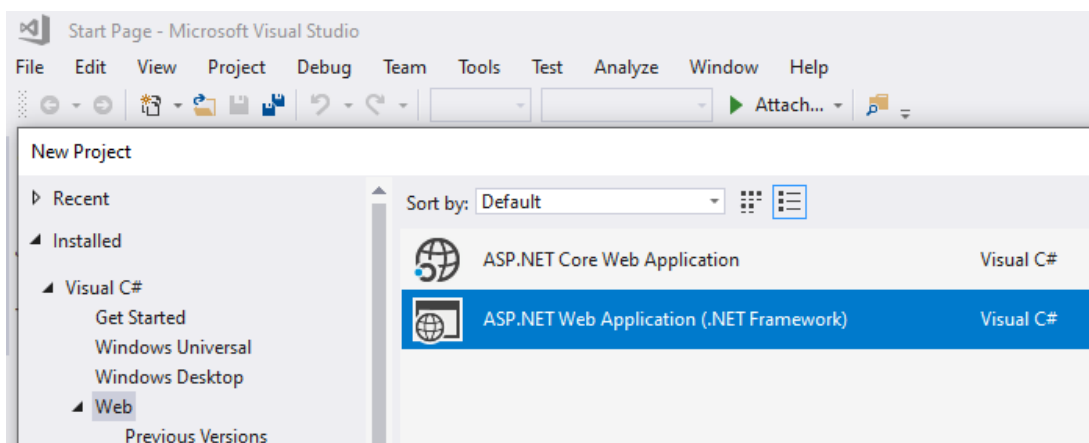
O ASP .NET Core é um framework multiplataforma de código aberto para construir aplicações web modernas e aplicações baseadas na nuvem .



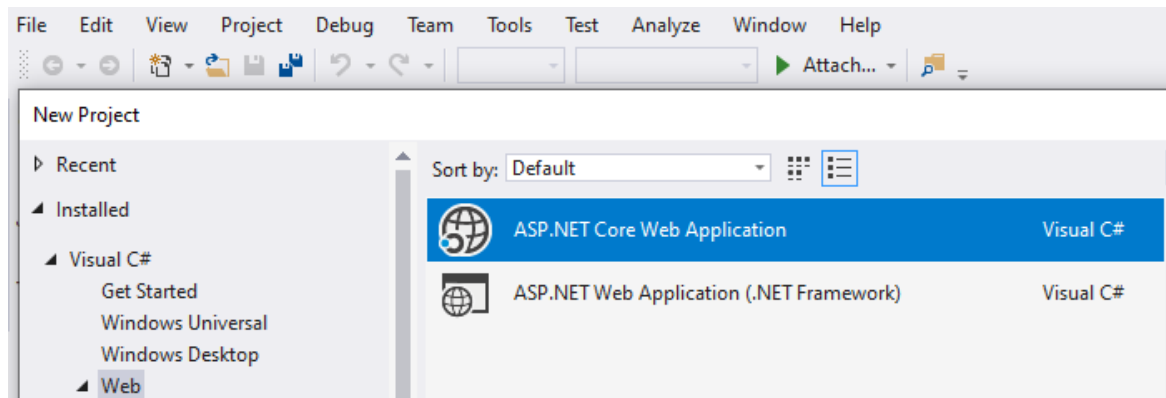
Podemos desenvolver e executar aplicações ASP .NET Core no Linux, Mac e Windows. O ASP .NET Core está na versão 6.

Agora podemos criar aplicações MVC com APIs. Tanto uma como a outra herdam o mesmo namespace.

Podemos criar aplicações ASP.NET Web Application como no exemplo abaixo, que rodam somente no Windows.



E também podemos criar aplicações ASP.NET Core Application que pode ser executada em Windows Mac e Linux.

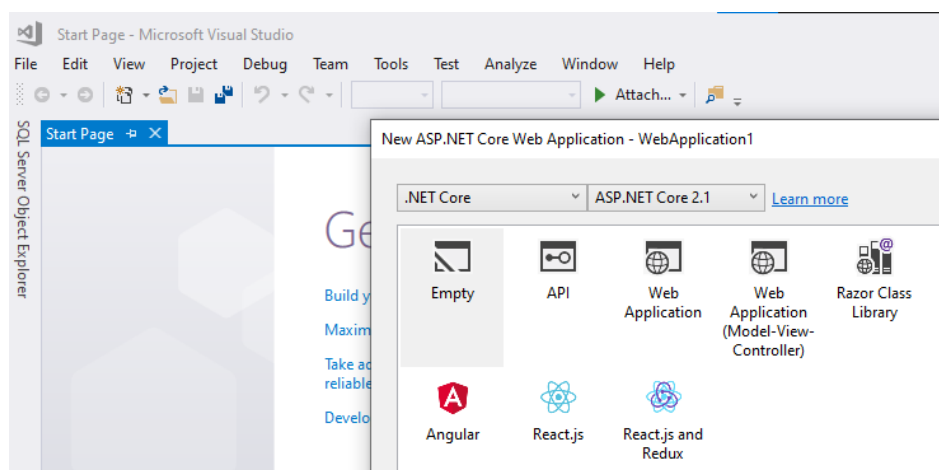


A ASP .NET Core é **independente** de qualquer **sistema de projeto proprietário** ou ambiente de desenvolvimento integrado.

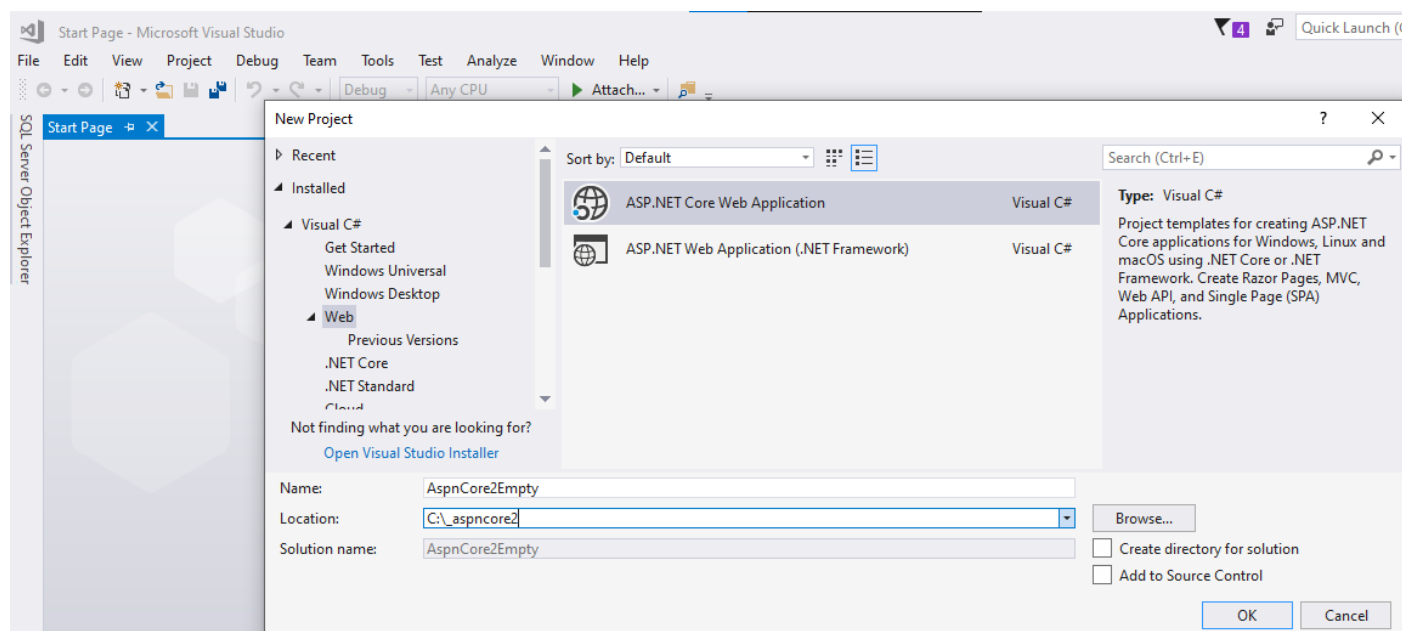
Você pode compilar um aplicativo ASP .NET Core fora do Visual Studio e em sistemas operacionais que não sejam Windows.

Opções de desenvolvimento

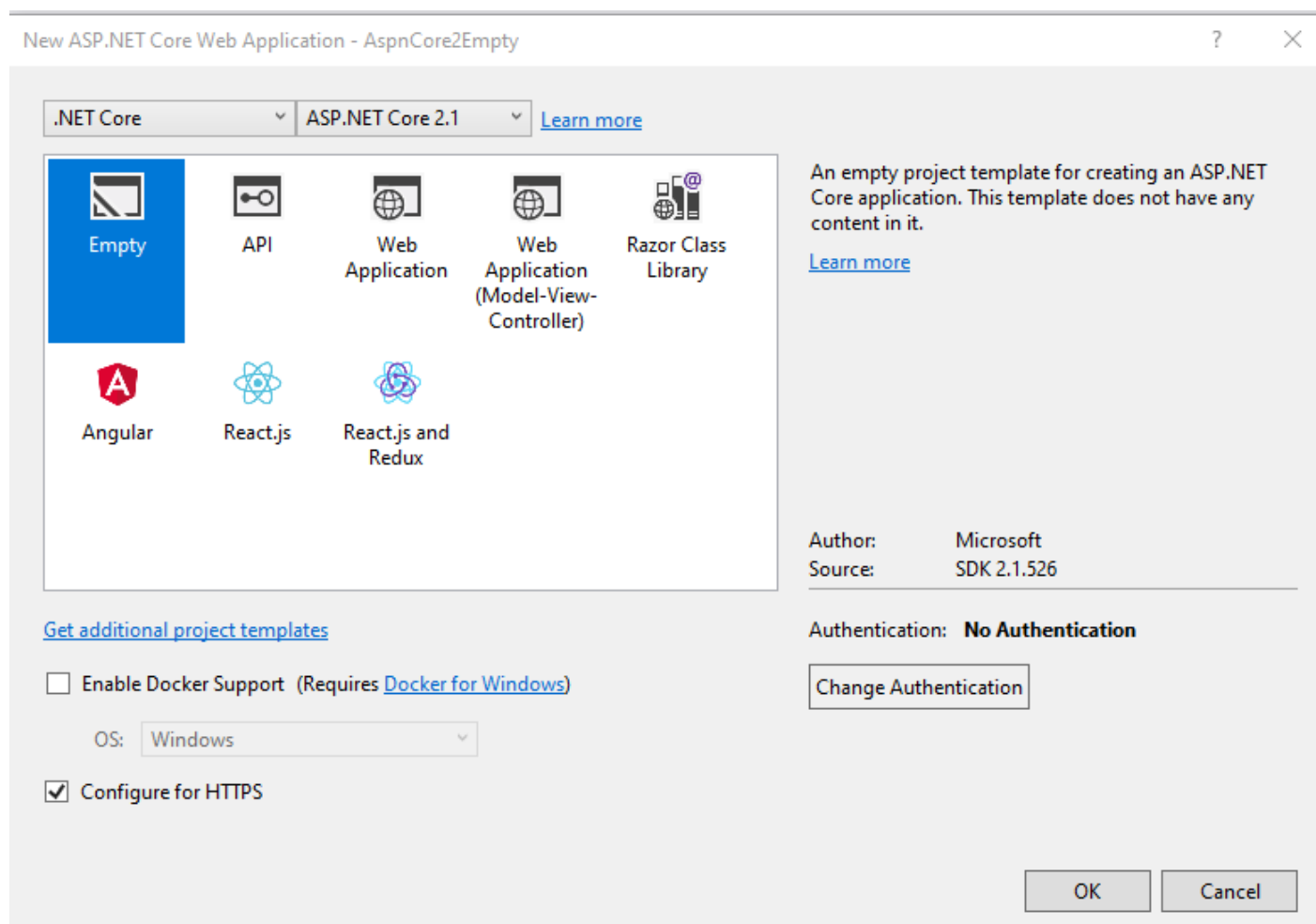
- **ASP .NET Web Forms** (manutenção de projetos em produção) (legado)
 - Não iniciar novo projetos usando Web Forms. A ASP .NET Core não suporta Web Forms.
- **ASP .NET MVC5** – (desenvolvimento de app para plataforma Windows)
 - Projetos novos para plataforma Windows que exigem recursos não suportados ainda no .NET Core (SignalR, etc)
- **ASP .NET Core 2.0** (desenvolvimento multiplataforma)
 - Projetos novos multiplataforma que não usam recursos ainda não disponíveis.



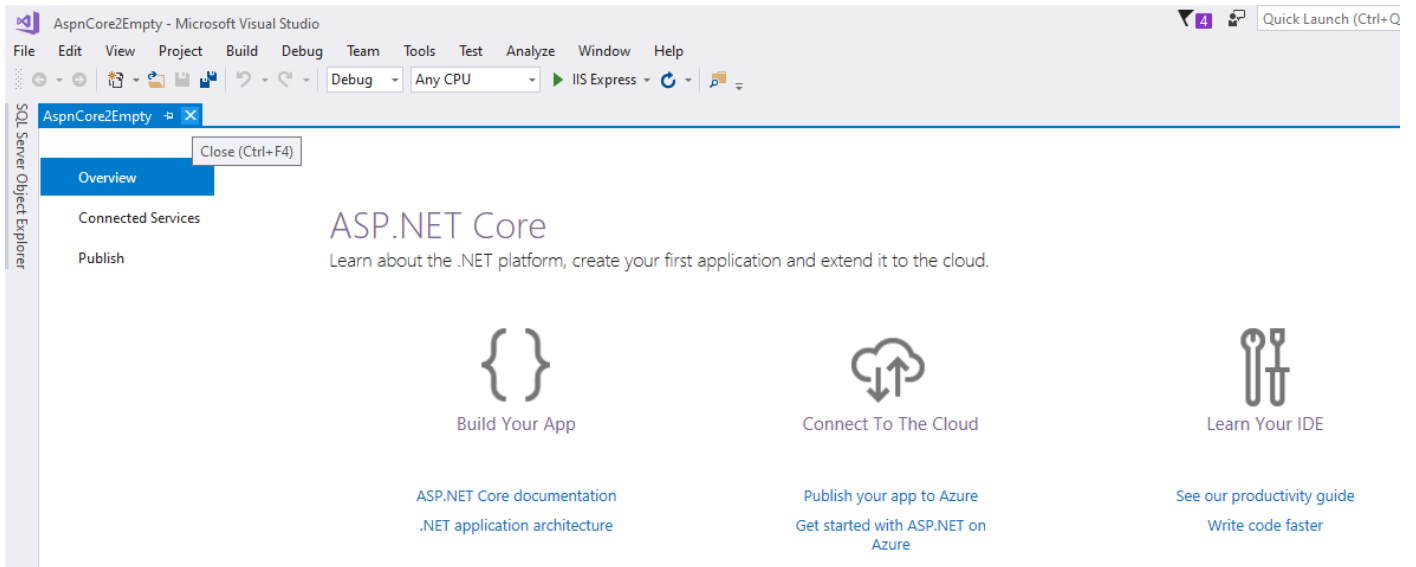
Criar um projeto vazio usando o VS 2017 e examinar a estrutura do projeto



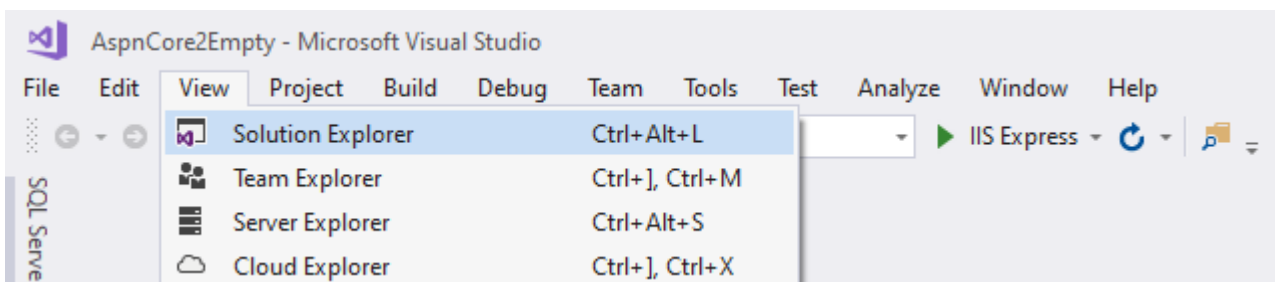
Na página seguinte vou escolher o tipo Empty.



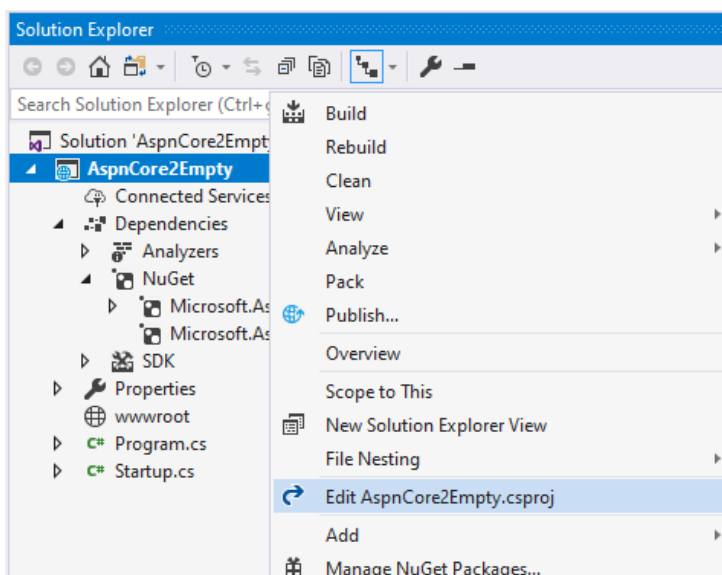
Vamos e fechar clicando no X.



Abrimos a Solution Explorer.



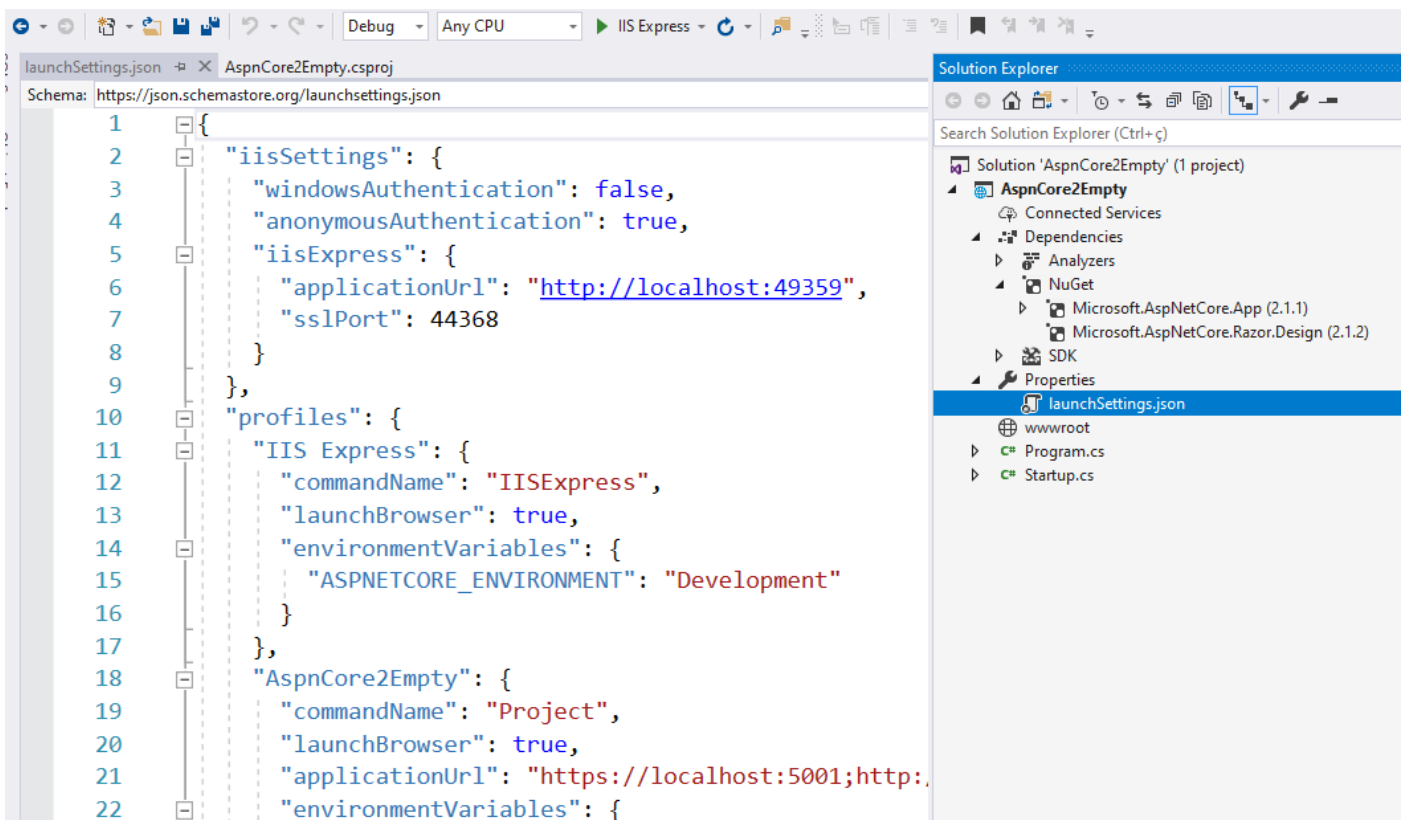
Clicando (botão direito do mouse) em AspCore2Empty > Edit AspCore2Empty.csproj



Podemos notar que o código está mais limpo.

```
AspnCore2Empty.csproj
1 <Project Sdk="Microsoft.NET.Sdk.Web">
2
3   <PropertyGroup>
4     <TargetFramework>netcoreapp2.1</TargetFramework>
5   </PropertyGroup>
6
7   <ItemGroup>
8     <Folder Include="wwwroot\" />
9   </ItemGroup>
10
11  <ItemGroup>
12    <PackageReference Include="Microsoft.AspNetCore.App" />
13    <PackageReference Include="Microsoft.AspNetCore.Razor.Design" Version="2.1.2" PrivateAssets="All" />
14  </ItemGroup>
15
16 </Project>
```

Nessa parte temos o endereço das páginas.

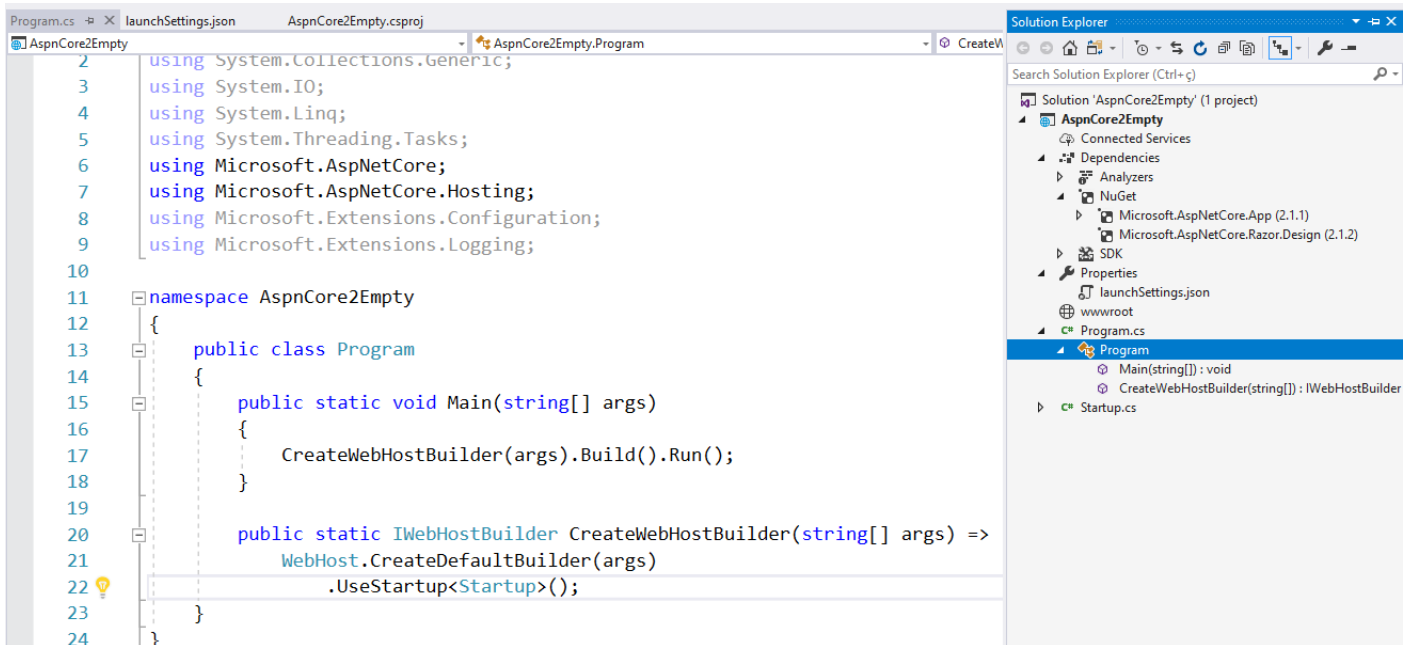


The screenshot shows the Visual Studio IDE with the `launchSettings.json` file open. The file is configured for IIS Express and includes settings for the `AspnCore2Empty` project. The `profiles` section defines two profiles: `IIS Express` and `AspnCore2Empty`. The `IIS Express` profile is set to launch the browser and uses the `ASPNETCORE_ENVIRONMENT` variable set to `Development`. The `AspnCore2Empty` profile is also set to launch the browser and uses the `ASPNETCORE_ENVIRONMENT` variable set to `Development`. The `applicationUrl` for the `AspnCore2Empty` profile is `https://localhost:5001;http://localhost:5000`.

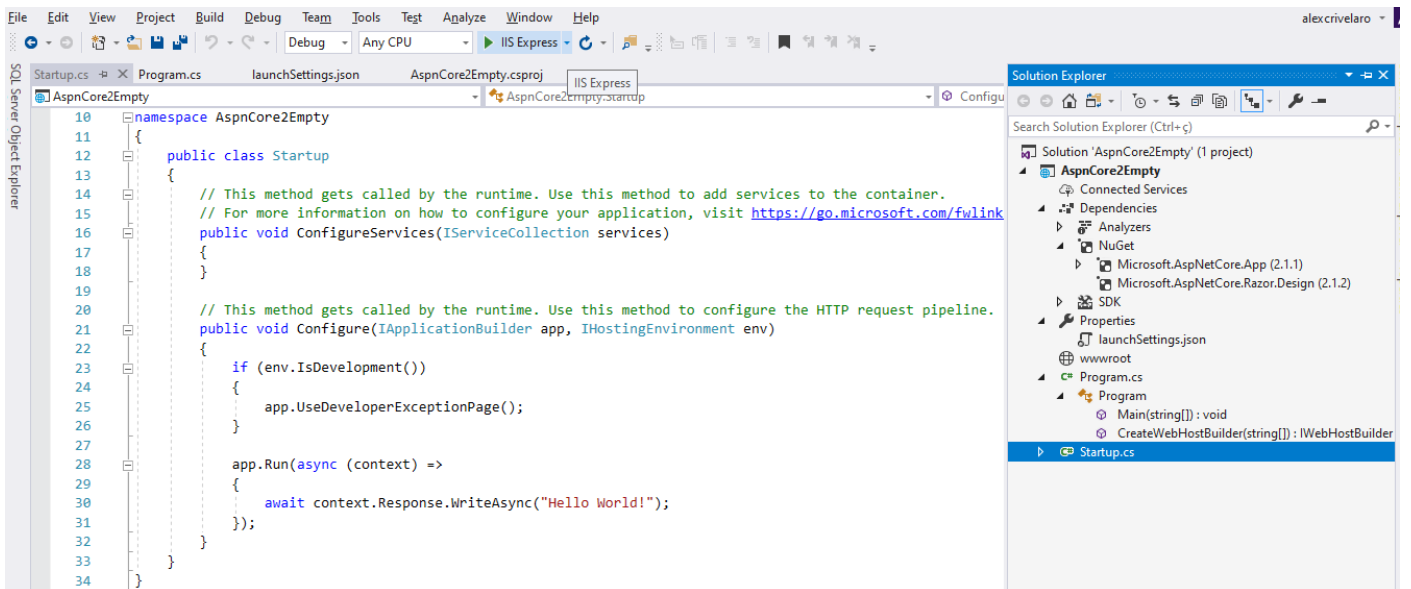
```
1 {
2   "iisSettings": {
3     "windowsAuthentication": false,
4     "anonymousAuthentication": true,
5     "iisExpress": {
6       "applicationUrl": "http://localhost:49359",
7       "sslPort": 44368
8     }
9   },
10  "profiles": {
11    "IIS Express": {
12      "commandName": "IISExpress",
13      "launchBrowser": true,
14      "environmentVariables": {
15        "ASPNETCORE_ENVIRONMENT": "Development"
16      }
17    },
18    "AspnCore2Empty": {
19      "commandName": "Project",
20      "launchBrowser": true,
21      "applicationUrl": "https://localhost:5001;http://localhost:5000",
22      "environmentVariables": {
```

The Solution Explorer on the right shows the project structure for `AspnCore2Empty`, including `Connected Services`, `Dependencies`, `Analyzers`, `NuGet` packages (`Microsoft.AspNetCore.App (2.1.1)` and `Microsoft.AspNetCore.Razor.Design (2.1.2)`), `SDK`, `Properties`, and the `launchSettings.json` file.

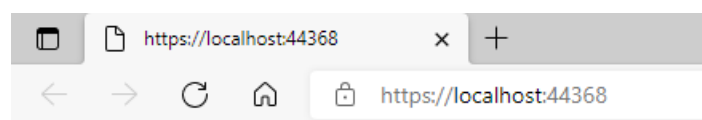
Em program temos o método Main, onde ocorre toda a inicialização.



A classe Startup.cs mostra o “Hello world!” clicando em IIS Express.



Resultado final da aplicação. O método Main() chama a classe Startup.cs do método Configure.

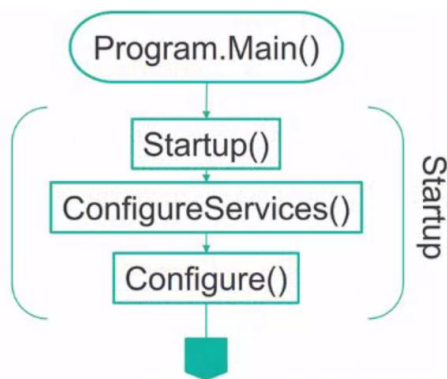


Hello World!

O entendendo o processo de inicialização das aplicações ASP .NET Core

Aplicações ASP .NET Core são aplicações do tipo .NET Core Console.

Todas bibliotecas de hospedagem são executadas a partir de Program.cs



Main() – Configura host, logging, configuration e chamao Startup()

Startup() – inicializa a app

ConfigureServices () - Adicionar Serviços
Verifica se há serviços e adiciona

Configure () - Configura os Middlewares

Olhando para nosso programa:

```
1 using System;
2 using System.Collections.Generic;
3 using System.IO;
4 using System.Linq;
5 using System.Threading.Tasks;
6 using Microsoft.AspNetCore;
7 using Microsoft.AspNetCore.Hosting;
8 using Microsoft.Extensions.Configuration;
9 using Microsoft.Extensions.Logging;
10
11 namespace AspnCore2Empty
12 {
13     public class Program
14     {
15         public static void Main(string[] args)
16         {
17             CreateWebHostBuilder(args).Build().Run();
18         }
19
20         public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
21             WebHost.CreateDefaultBuilder(args)
22                 .UseStartup<Startup>();
23     }
24 }
```

O método Main é responsável por iniciar o WebHost invocando a classe Startup que executa a aplicação.

Configuração na ASP.NET Core

É definida aos pares no formato **nome: valor**

Os dados de configuração do aplicativo podem vir de:

- Arquivo, como JSON, XML, INI
- Variáveis ambientais
- Argumentos da linha de comando
- Uma coleção na memória
- Provedores personalizados

Formato mais utilizado : JSON

Os dados são separados por vírgulas(,)

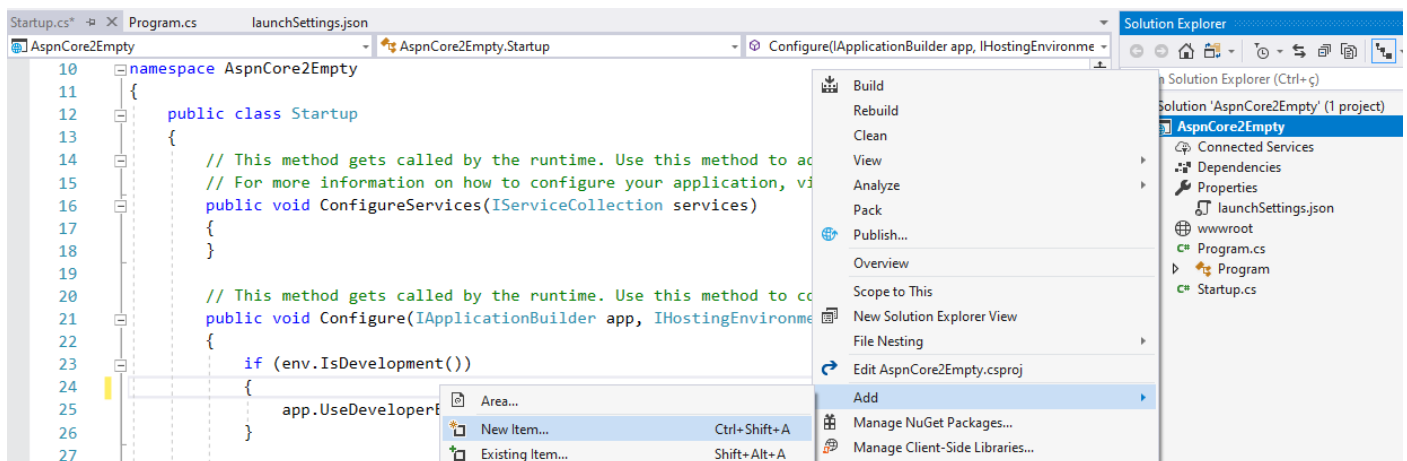
As chaves { } contém objetos

Os colchetes [] expressam matrizes/vetores

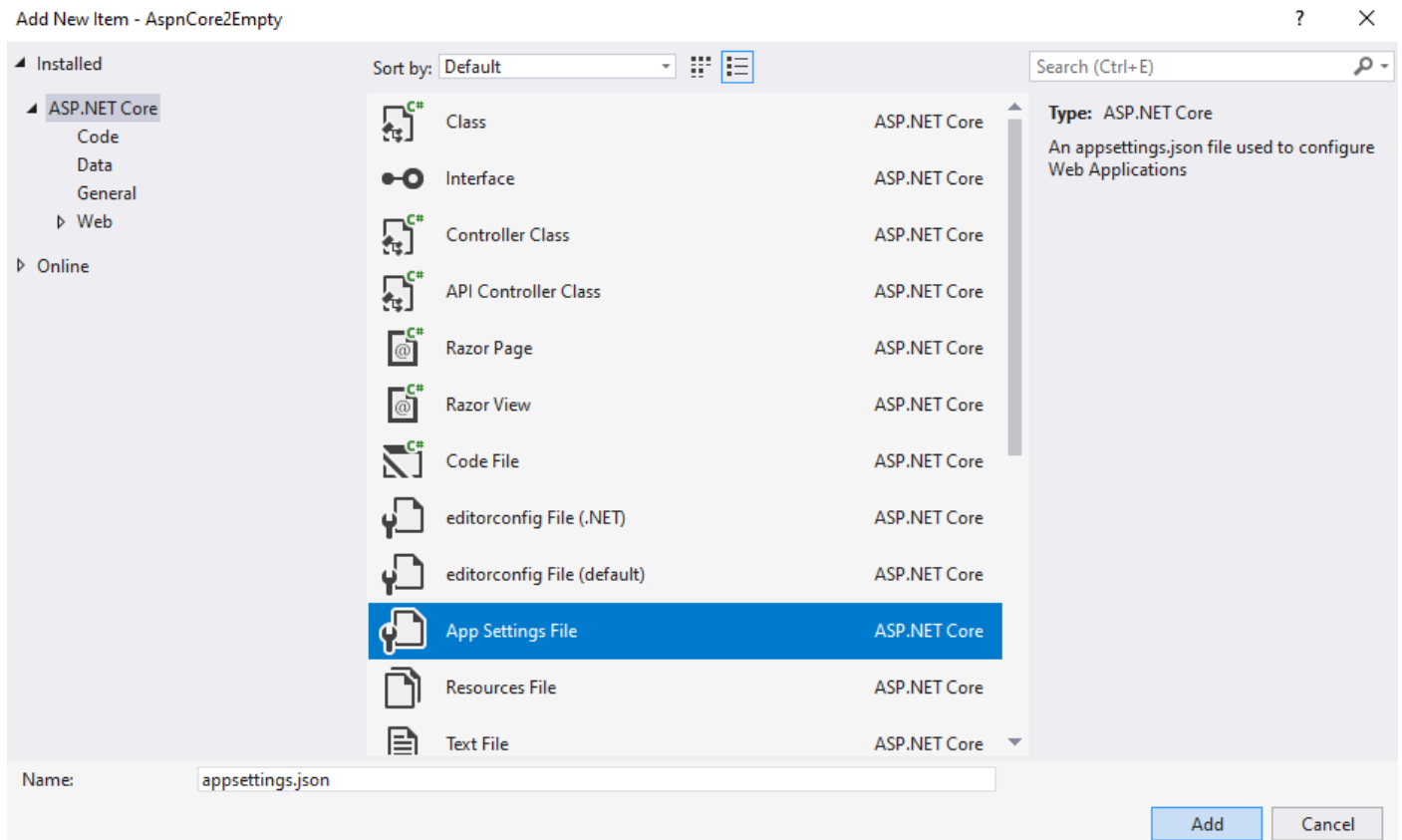
Vamos ler esse arquivo de configuração, como podemos criar esse arquivo e ler essas mesmas informações.

Clique com o botão direito do mouse

no projeto (AspnCore2Empty) > Add > New Item



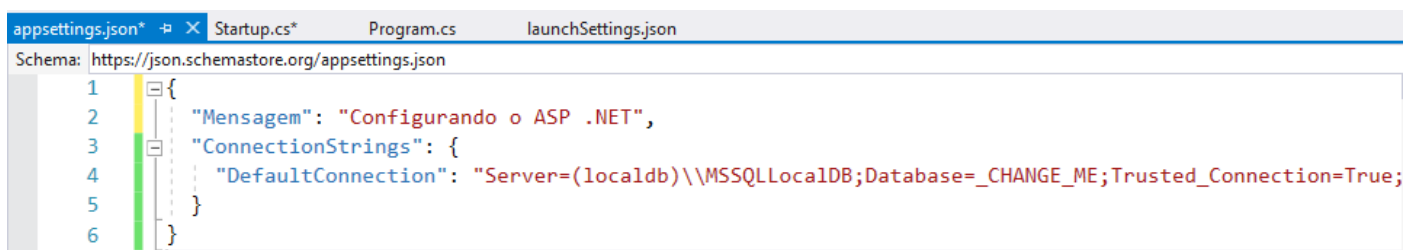
Ele me sugere o nome appsettings.json e clico em Add



E adicionarei mais uma linha de configuração separado por vírgula.

"Mensagem": "Configurando o ASP .NET",

Com isso acessaremos esse arquivo e ler essas informações



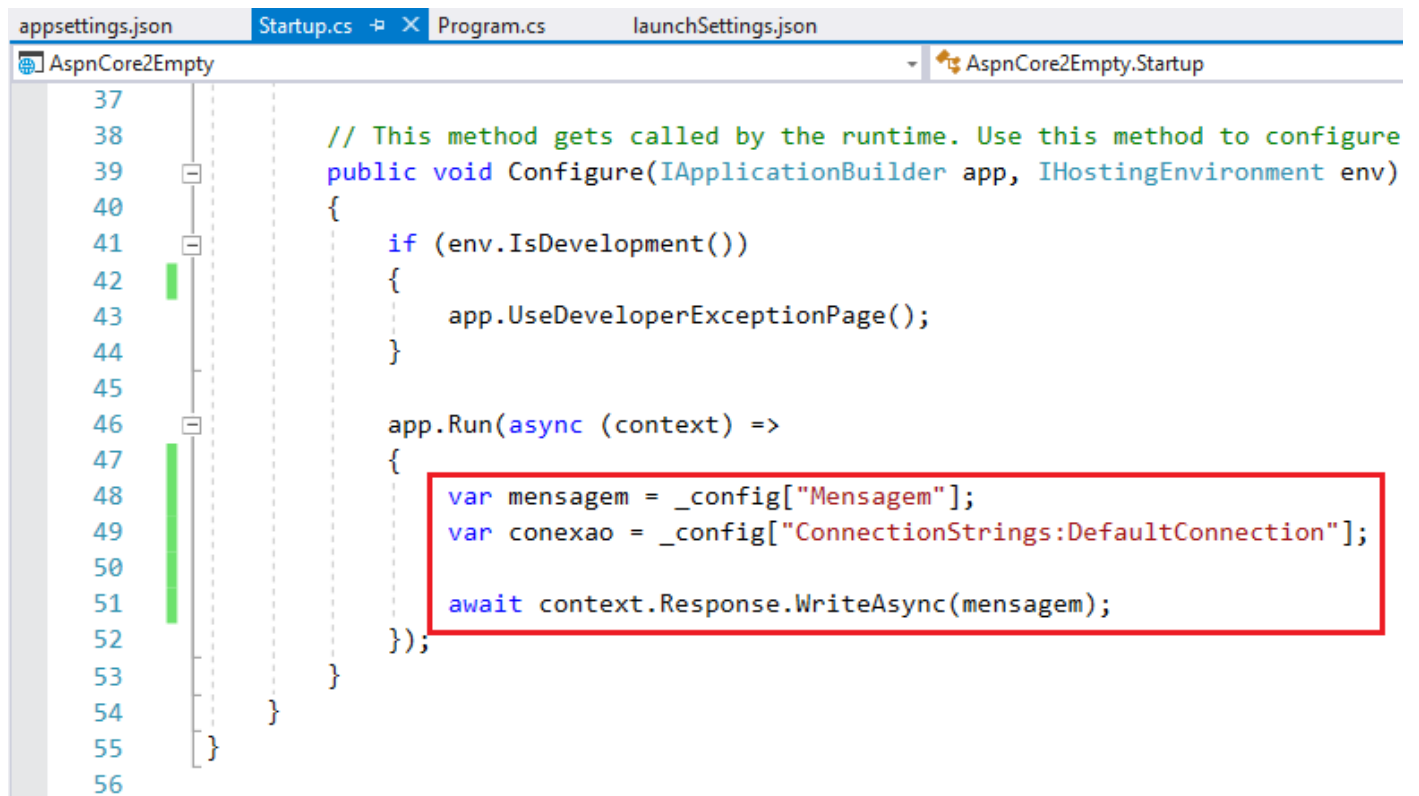
Na minha classe Startup ao adicionar o IConfiguration é necessário usar a biblioteca: `using Microsoft.Extensions.Configuration;`

E também o `using System.IO;`

Adicionaremos o código abaixo:

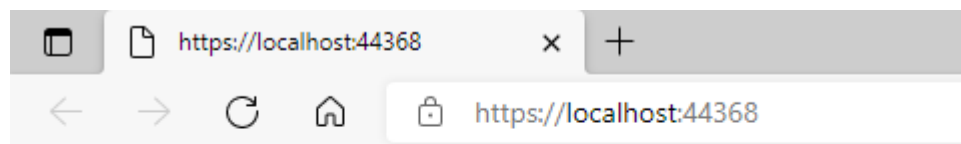
```
appsettings.json | Startup.cs | Program.cs | launchSettings.json
└─ AspnCore2Empty ──┬─ AspnCore2Empty.Startup
                     |
1  └─ using System;
2  └─ using System.Collections.Generic;
3  └─ using System.IO;
4  └─ using System.Linq;
5  └─ using System.Threading.Tasks;
6  └─ using Microsoft.AspNetCore.Builder;
7  └─ using Microsoft.AspNetCore.Hosting;
8  └─ using Microsoft.AspNetCore.Http;
9  └─ using Microsoft.Extensions.Configuration;
10 └─ using Microsoft.Extensions.DependencyInjection;
11
12 └─ namespace AspnCore2Empty
13     {
14     └─ public class Startup
15         {
16         └─ public IConfiguration _config { get; set; }
17
18         └─ public Startup()
19             {
20             └─ var builder = new ConfigurationBuilder()
21                 {
22                 └─ // define o caminho do diretório
23                     .SetBasePath(Directory.GetCurrentDirectory())
24
25                     └─ // provedor de configuração JSON
26                         .AddJsonFile("appsettings.json");
27
28                     └─ // cria a estrutura de configuracao
29                         _config = builder.Build();
30                 }
29             }
30         }
31     }
```

E para exibir a mensagem adicionaremos mais essas linhas da configuração da mensagem e da configuração da conexão do arquivo JSON.



```
37
38 // This method gets called by the runtime. Use this method to configure
39 public void Configure(IApplicationBuilder app, IHostingEnvironment env)
40 {
41     if (env.IsDevelopment())
42     {
43         app.UseDeveloperExceptionPage();
44     }
45
46     app.Run(async (context) =>
47     {
48         var mensagem = _config["Mensagem"];
49         var conexao = _config["ConnectionStrings:DefaultConnection"];
50
51         await context.Response.WriteAsync(mensagem);
52     });
53 }
54
55
56
```

No navegador temos a seguinte mensagem:

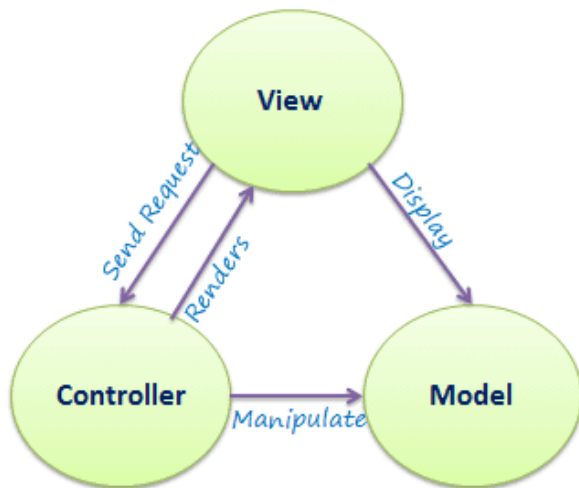


Configurando o ASP .NET

2 - O padrão MVC (Model-View-Controller) ,

MVC – É um padrão arquitetura de software.

Esse padrão separa o aplicativo em 3 componentes principais.



Model - Responsável por manter os dados e lógica de negócio da app

View - É a interface com o usuário da app que exibe os dados. Página HTML ou a tela de uma aplicação desktop, tudo que exibe dados.

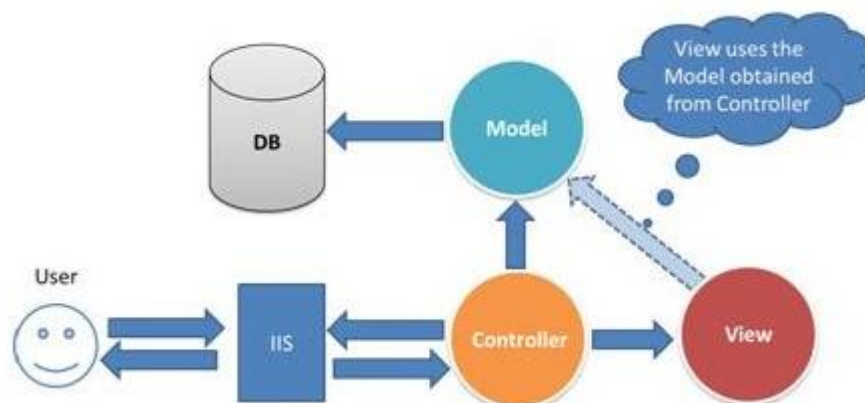
Controler - Recebe e trata as requisições e renderiza a view a propriedade com os dados

São os componentes que lidam com a interação do usuário, trabalham com o modelo e finalmente selecionam uma view para renderizar e exibir para o usuário

O Objetivo do padrão MVC é obter um baixo acoplamento entre a Model e a View, permitir o encapsulamento da lógica do negócio no modelo para que ele possa ser testado de maneira exaustiva, e fazer com que as Views sejam leves e desacopladas do Model para facilitar os testes.

O padrão MVC separa os diferentes componentes da aplicação de forma a dar mais controle em cada parte individual da aplicação, tornando assim a aplicação mais fácil de desenvolver, modificar e testar.

Fluxo da requisição do usuário



Inicialmente o usuário faz uma solicitação de algum recurso no servidor. Ele faz isso digitando uma url no navegador.

A requisição chega no controlador e lá existe um mecanismo de roteamento que vai ser responsável por decidir qual requisição será tratada pelo controlador.

O Controlador recebe a requisição, e se houver necessidade, ele vai e acessa o Model.

O Model vai operar em um banco de dados, ou em uma outra fonte de dados, e retornar as informações na forma de dados, ou na forma de objetos de negócio através do Controlador.

O Controlador escolherá a View apropriada para exibir os dados e montará essa View, usando HTML e componentes como caixa de texto, ou label.

Depois de montado, o Controlador passará os dados do modelo na View escolhida, preencher essa View e retornar para o usuário

Basicamente esse é o fluxo de requisição no modelo MVC.

A ASP .NET Core implementa o padrão MVC.

Implementação ASP .NET Core MVC

- Agrega diversos recursos que ajuda os desenvolvedores a manter as boas práticas e dar produtividade. Alguns desses recursos são:

View Engine

Model Binding

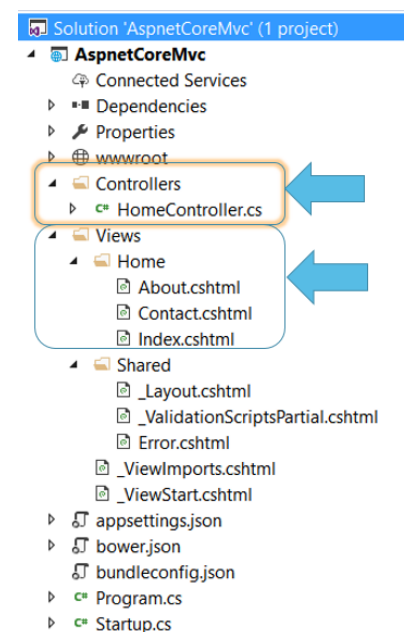
Tag-Helpers

Routing

Scaffolding

Configuração sobre Convenção

Podemos definir o Model na Pasta Models da App



Entendendo o MVC

URL não é igual a página

No modelo de desenvolvimento MVC uma **URL** corresponde a uma **ação** de um **Controller** e não a uma página em disco.

No modelo de desenvolvimento MVC as requisições do navegador são mapeadas para **ações** do **Controller**.

<http://localhost/home/index> => home -> controller
Index -> action

O controlador irá fazer a ação e devolver uma página.

O desenvolvimento é centrado na lógica da aplicação. (leia-se mapeamento)