

ASP.Net Core MVC

Linguagem de Programação Comercial

Conhecendo a Estrutura do asp.net core mvc

@cassiocosta_ 9 8126-3079

Documentação oficial

- <http://docs.microsoft.com>
- <https://docs.microsoft.com/pt-br/aspnet/core/getting-started>
- <https://docs.microsoft.com/pt-br/dotnet/core/tools/> (dotnet cli)
- <https://github.com/dotnet/core/>

O que é o ASP dot net core

- ASP. [Ing. Sigla para Active Server Page]. contém comandos para scripts e tags HTML, que rodam no servidor e, por isso, independe do navegador e do sistema operacional de que o usuário dispõe.
- faz parte do dot net core framework e não é uma linguagem de programação
- Não é o servidor de páginas, como IIS, Kestrel, apache
- possibilita criar software web dinâmicos, com o uso de uma linguagem de programação. Ex. C#, VB
- é Open source.

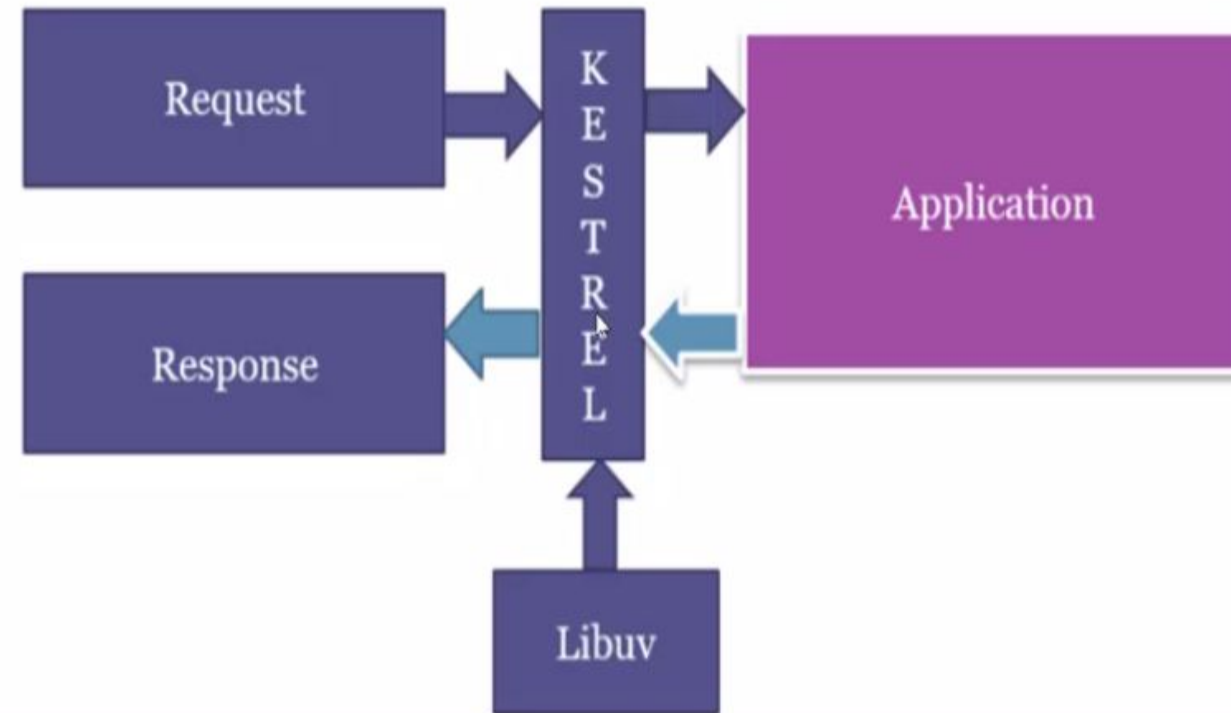
Principais recursos do ASP.NET Core

- Kestrel
- Middleware
- Dependency Injection
- Configuration

Hosting

- não é um server
- o hosting sabe qual é o server
- Responsável pela inicialização do aplicativo/software
- Gerencia o ciclo de vida de uma aplicação
- Responsável de pegar as solicitações http do server e enviar para o aplicativo.

Request-Response with Kestrel



documentação oficial em : <https://docs.microsoft.com/pt-br/aspnet/core/fundamentals/hosting>

Kestrel

- é um web server cross-plataform para ASP.Net Core baseado em libuv, um cross-plataform asynchornous I/O library;
- responsável de gerenciar as requisições http.
- Libuv - biblioteca em C multiplataforma com foco em IO async. Desenvolvido para node mais largamente usado em outras aplicações, como o kestrel.
- 2300% mais requisições por segundo do que o .net framework;
- .Net Core Performance 3x faster than NodeJs :)

doc

oficial

em:

[Fonte: https://docs.microsoft.com/pt-br/aspnet/core/fundamentals/servers/kestrel](https://docs.microsoft.com/pt-br/aspnet/core/fundamentals/servers/kestrel)

Implementando um hosting

- Todo o aplicativo no ASP.NET Core é um console, ou seja, tem o método Main.
- Crie um novo projeto do tipo console “AppHosting”
- Adicionar uma dependência para ASP.NET Core
 - Execute no CLI **dotnet add package Microsoft.AspNetCore**
 - Execute o **dotnet restore**
- Abrir o projeto no VS Code. Digite no CLI
 - **code .**
- <https://docs.microsoft.com/pt-br/dotnet/core/tools/dotnet-add-package>

Implementando o hosting

- Conforme figura, no arquivo *program.cs* importe as bibliotecas (dependências) necessárias para trabalhar com ASP.NET Core e digite o código para o método *static Main*.

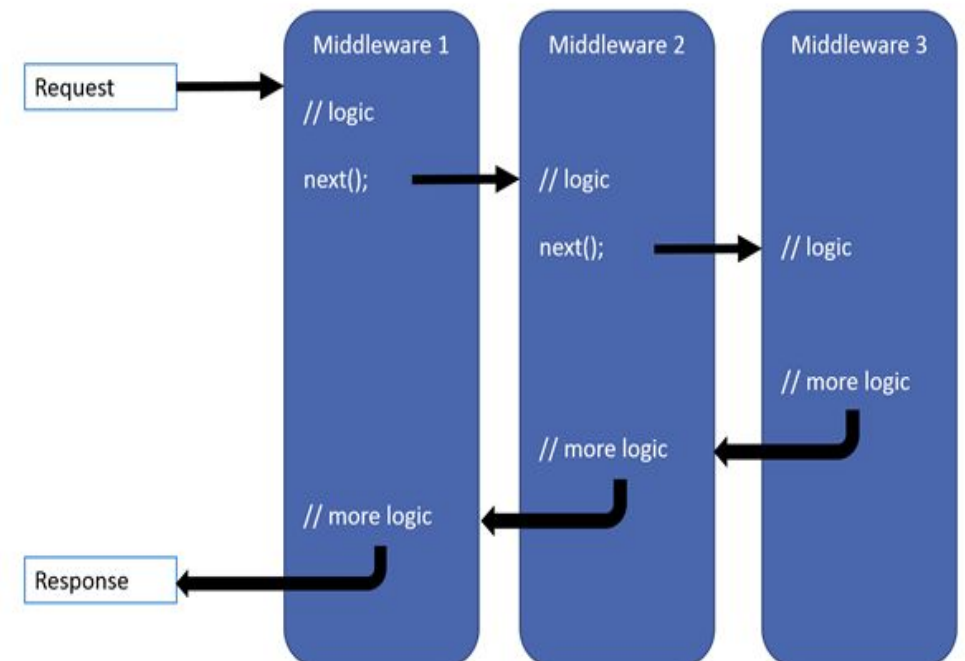
```
1 using System;
2 using Microsoft.AspNetCore.Builder;
3 using Microsoft.AspNetCore.Hosting;
4 using Microsoft.AspNetCore.Http;
```

```
0 references
8 class Program
9 {
    0 references
10 static void Main(string[] args)
11 {
12     var host = new WebHostBuilder()
13         .UseKestrel()
14         .Configure(
15             app => {
16                 app.Run(context => context.Response.WriteAsync("Olá Mundo"));
17             }
18         )
19         .Build();
20     host.Run();
21 }
22 }
```

- Logo após, coloque para rodar a aplicação.
- dotnet run** 😊 e abra no seu browser
- sobre o operador “=>” acesse:
 - <https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/operators/lambda-operator>

Middlewares

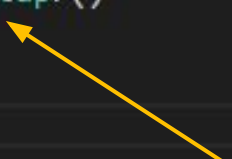
- Middleware é definido no Asp.Net Core para passar por componentes que formam um *pipeline* entre um servidor e aplicativo para inspecionar, rotear ou modificar mensagens de solicitação e resposta para um propósito específico.
- *Pipeline* se refere, por sua vez, aos diversos elementos envolvidos no fluxo de processamento de requisições e respostas de um sistema remoto, normalmente um projeto Web.
- O objetivo principal é a utilização mais racional dos recursos requeridos por um projeto Web, ativando apenas as funcionalidades essenciais dentro de cada contexto.
- Um *middleware* nada mais do que é um componente de *software* que faz parte do pipeline de execução de uma aplicação.
- da para comparar com um hamburger. onde as extremidades são as aplicações de servidor e o meio são os middlewares.
- <https://docs.microsoft.com/pt-br/aspnet/core/fundamentals/middleware>



Implementando a classe Startup

- Os Middlewares ficam nas configurações dentro do método mais.
- Para melhor organização, cria-se uma classe chamada de Startup. Refatore o método Main.

```
9      public class Program
10     {
11         0 references
12         public static void Main(string[] args)
13         {
14             var host = new WebHostBuilder()
15                 .UseKestrel()
16                 .UseStartup<Startup>()
17                 .Build();
18             host.Run();
19         }
20     }
```



- **WebHostBuilder**: uma classe que ao instanciar um objeto, cria e retorna uma instância do **WebHost** que é a lib que cuida do ciclo de vida das requisições http do servidor.
- UserKestrel: Manda utilizar o **Kestrel**, pode usar outros, como IIS.
- **UseStartup**: Configura para inicializar com a classe **Startup**.
- E construí o host com o **Build**
- Executa o host com **host.Run()**

Implementando a classe Startup

- Na classe Startup implemente o código da figura.
- Fazer um build na aplicação. Se tudo ok, execute a aplicação.
- Abra a aplicação no seu browser.

```
1 reference
5 public class Startup
6 {
7     0 references
8     public void Configure(IApplicationBuilder app)
9     {
10         app.Run(async context =>
11         {
12             await context.Response.WriteAsync("Hello, World!");
13         });
14     }
15 }
```

Analizando os Middlewares numa aplicação MVC

- Crie uma aplicação mvc - no cli digite: `dotnet new mvc -o AppHostingMvc`
- Abra no seu vscode; code .
- Analise a classe `Startup.cs`

ASP.Net Core

Crédito pós aula:

Acesse o link e estudo sobre MVC para a aula que vem.

<https://docs.microsoft.com/en-us/aspnet/core/mvc/overview>

