



Aula 02

Prof: Henrique Augusto Maltauro

Codificar Back-end de Aplicações Web

Arquitetura de Aplicações Web

- C#: API

O Visual Studio, já tem alguns modelos pré-prontos para criar uma API com C#.

Nós estaremos utilizando a opção ASP.NET Core Web API, a qual já é criada com o padrão REST, que vamos aprender no decorrer desta aula.

Arquitetura de Aplicações Web

- C#: API



ASP.NET Core Web API

A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

C#

Linux

macOS

Windows

Cloud

Service

Web

WebAPI

Arquitetura de Aplicações Web

- C#: API

Quando executamos uma API no Visual Studio, ele cria um servidor no seu computador, e define uma URL de acesso para acessar a API.

Por padrão essa URL de acesso é localhost:7043.

Arquitetura de Aplicações Web

- HTTP (vamos rever alguns conceitos)

Do inglês Hypertext Transfer Protocol (Protocolo de Transferência de Hipertexto), o HTTP é o protocolo padrão de comunicação na internet.

Toda a estrutura de comunicação do HTTP é feita através de métodos, e esses métodos serão utilizados para se comunicar com a API, para realizar os processos da API.

Arquitetura de Aplicações Web

- HTTP (vamos rever alguns conceitos)

GET

POST

PUT

DELETE

PATCH

CONNECT

OPTIONS

TRACE

HEAD

Arquitetura de Aplicações Web

- HTTP (vamos rever alguns conceitos)

Os métodos HTTP vão receber uma requisição (request), definindo que eles façam alguma ação na API.

E vão retornar uma resposta (response), dizendo se a requisição foi executada com sucesso ou não, e até retornando dados que foram solicitados na requisição.

Arquitetura de Aplicações Web

- HTTP (vamos rever alguns conceitos)

GET: Requisita o retorno de um recurso específico.

POST: Requisita a adição de um recurso novo.

PUT: Requisita a atualização de todo o conteúdo de um recurso específico.

PATCH: Requisita a atualização de parte do conteúdo de um recurso específico.

DELETE: Requisita a remoção de recurso específico.

Arquitetura de Aplicações Web

- REST

Do inglês **Representational State Transfer** (**Transferência Representacional de Estado**), o **REST** é uma arquitetura de software que define um conjunto de regras a serem usadas para a criação de um serviço WEB, no nosso caso, uma **API**.

Arquitetura de Aplicações Web

- REST

A ideia do REST é padronizar a comunicação na internet em geral de forma abstrata, utilizando os métodos do protocolo HTTP.

Por que de forma abstrata?

Porque a informação que as chamadas de uma API REST recebe e/ou retorna, pode ser qualquer tipo de informação.

Arquitetura de Aplicações Web

- REST

Importante lembrar que REST não é a mesma coisa que RESTfull.

Sempre que vocês virem algum software definido como RESTfull, simplesmente significa que aquele software utiliza a arquitetura REST.

Arquitetura de Aplicações Web

- Controller

O controller, em português **controlador**, é um dos principais elementos da API, e ele é a **classe** responsável por receber a **requisição**, ele é a porta de entrada para a API.

Arquitetura de Aplicações Web

- C#: Controller

No C#, o controller vai ser uma classe, que pode ser criada através de um modelo pré-pronto chamado API Controller - Empty.

Arquitetura de Aplicações Web

- C#: Controller



API Controller - Empty

C#

Arquitetura de Aplicações Web

- C#: Controller

E o que define uma classe como controller são duas coisas:

- [ApiController]
- ControllerBase

Arquitetura de Aplicações Web

- C#: Controller ([ApiController])

No C#, o [ApiController] é um atributo de configuração, que define algumas configurações para aquela classe se comportar como um controller de API.

Arquitetura de Aplicações Web

- C#: Controller ([ApiController])

```
[ApiController]
public class ExemploController
{
    // Bloco de código do controller
}
```

Arquitetura de Aplicações Web

- C#: Controller (ControllerBase)

No C#, o ControllerBase é uma classe que possui todos os métodos necessários para que a classe que herdar dela, consiga realizar todos os processos de um controller.

Arquitetura de Aplicações Web

- C#: Controller (ControllerBase)

```
[ApiController]
public class ExemploController : ControllerBase
{
    // Bloco de código do controller
}
```

Arquitetura de Aplicações Web

- Controller (roteamento)

O roteamento é a principal forma utilizada na internet para a comunicação entre computadores e aplicações. Ele é composto de um caminho, que nada mais é do que a URL de acesso.

Aquela URL que você digita no navegador de internet é uma rota, e o acesso aos controllers e processos de uma API é feito da mesma forma.

Arquitetura de Aplicações Web

- C#: Controller (roteamento)

No C#, nós temos alguns atributos de configuração que são utilizados para definir o roteamento, e assim definir qual é a rota específica para acessar determinado processo da API.

Arquitetura de Aplicações Web

- C#: Controller (roteamento [Route])

O [Route] é um atributo de configuração, que define uma rota para acessar tanto o controller como cada um dos seus métodos.

Esse atributo vai receber uma string, a qual vai definir qual é a rota de acesso.

Arquitetura de Aplicações Web

- C#: Controller (roteamento [Route])

```
[ApiController]
[Route("api/exemplo")]
public class ExemploController : ControllerBase
{
    // Bloco de código do controller
}
```

Arquitetura de Aplicações Web

- C#: Controller (roteamento [Route])

```
[ApiController]
[Route("api/exemplo")]
public class ExemploController : ControllerBase
{
    [Route("metodo/teste")]
    public string MetodoTeste()
    {
        // Bloco de código do método a ser
        // executado na rota:
        // api/exemplo/metodo/teste
    }
}
```


Arquitetura de Aplicações Web

- C#: Controller (roteamento [Route])

Nesta `string` que o `[Route]` recebe, podemos definir algumas configurações de `rotas dinâmicas`.

Podemos definir uma `rota dinâmica` para o `controller` com o nome do `controller` através da configuração `[controller]`.

Arquitetura de Aplicações Web

- C#: Controller (roteamento [Route])

```
[ApiController]
[Route("api/[controller]")] // api/Exemplo
public class ExemploController : ControllerBase
{
    // Bloco de código do controller
}
```

Arquitetura de Aplicações Web

- C#: Controller (roteamento [Route])

Podemos definir uma rota dinâmica para um método que possui parâmetros através da configuração {parametro}.

Arquitetura de Aplicações Web

- C#: Controller (roteamento [Route])

```
[ApiController]
[Route("api/[controller]")] // api/Exemplo
public class ExemploController : ControllerBase
{
    [Route("metodo/teste/{id}")]
    // api/Exemplo/metodo/teste/1
    // api/Exemplo/metodo/teste/2
    // api/Exemplo/metodo/teste/3
    // etc...
    public string MetodoTeste(long id)
    {
        // Bloco de código do método
    }
}
```

Arquitetura de Aplicações Web

- C#: Controller (roteamento)

Mas para os métodos da classe controller, definir somente as rotas de acesso através do [Route] não basta, devemos definir também qual é o método HTTP que aquele processo está relacionado.

E para isso, o C# tem alguns atributos de configuração que realizam esse processo.

Arquitetura de Aplicações Web

- C#: Controller (roteamento [HttpGet])

O [HttpGet] é um atributo de configuração que define um método como relacionado ao processo do método HTTP GET.

Arquitetura de Aplicações Web

- C#: Controller (roteamento [HttpGet])

```
[ApiController]
[Route("api/[controller]")]
public class ExemploController : ControllerBase
{
    [HttpGet]
    [Route("{id}")]
    public string GetById(long id)
    {
        // Bloco de código do método GET
    }
}
```

Arquitetura de Aplicações Web

- C#: Controller (roteamento [HttpPost])

O [HttpPost] é um atributo de configuração que define um método como relacionado ao processo do método HTTP POST.

Arquitetura de Aplicações Web

- C#: Controller (roteamento [HttpPost])

```
[ApiController]
[Route("api/[controller]")]
public class ExemploController : ControllerBase
{
    [HttpPost]
    [Route("pessoa/novo/{nome}")]
    public string PostPessoa(string nome)
    {
        // Bloco de código do método POST
    }
}
```

Arquitetura de Aplicações Web

- C#: Controller (roteamento [HttpPut])

O [HttpPut] é um atributo de configuração que define um método como relacionado ao processo do método HTTP PUT.

Arquitetura de Aplicações Web

- C#: Controller (roteamento [HttpPut])

```
[ApiController]
[Route("api/[controller]")]
public class ExemploController : ControllerBase
{
    [HttpPut]
    [Route("pessoa/atualiza/{nome}")]
    public string PutPessoa(string nome)
    {
        // Bloco de código do método PUT
    }
}
```

Arquitetura de Aplicações Web

- C#: Controller (roteamento [HttpPatch])

O [HttpPatch] é um atributo de configuração que define um método como relacionado ao processo do método HTTP PATCH.

Arquitetura de Aplicações Web

- C#: Controller (roteamento [HttpPatch])

```
[ApiController]
[Route("api/[controller]")]
public class ExemploController : ControllerBase
{
    [HttpPatch]
    [Route("pessoa/atualiza/{nome}")]
    public string PatchPessoa(string nome)
    {
        // Bloco de código do método PATCH
    }
}
```

Arquitetura de Aplicações Web

- C#: Controller (roteamento [HttpDelete])

O [HttpDelete] é um atributo de configuração que define um método como relacionado ao processo do método HTTP DELETE.

Arquitetura de Aplicações Web

- C#: Controller (roteamento [HttpDelete])

```
[ApiController]
[Route("api/[controller]")]
public class ExemploController : ControllerBase
{
    [HttpDelete]
    [Route("{id}")]
    public string DeleteById(long id)
    {
        // Bloco de código do método DELETE
    }
}
```

Arquitetura de Aplicações Web

- Swagger

O Swagger é um framework composto por diversas ferramentas que, independente da linguagem, auxilia a descrição, consumo e visualização das rotas de uma API REST.

Basicamente ele vai listar todas as rotas da API, e disponibilizar uma forma de executar cada uma delas.

Arquitetura de Aplicações Web

- C#: Swagger

No C#, podemos criar uma API já com o Swagger através da opção **Enable OpenApi support**, quando estivermos definindo as configurações iniciais da API.

Arquitetura de Aplicações Web

- C#: Swagger

ASP.NET Core Web API C# Linux macOS Windows Cloud Service Web WebAPI

Framework ⓘ

.NET 6.0 (Long Term Support) ▾

Authentication type ⓘ

None ▾

☒ Configure for HTTPS ⓘ

☐ Enable Docker ⓘ

Docker OS ⓘ

Linux ▾

☒ Use controllers (uncheck to use minimal APIs) ⓘ

☒ Enable OpenAPI support ⓘ

☒ Do not use top-level statements ⓘ

Arquitetura de Aplicações Web

- Controller

Apesar de parecer maravilhoso até o momento, não é interessante, nem recomendado, ter toda a lógica do software diretamente dentro da função que recebe o roteamento, nem mesmo dentro do controller.

Para isso, vamos fazer o uso do conceito de handler.

Arquitetura de Aplicações Web

- Handler

O **handler**, em português **manipulador**, tem por objetivo possuir todo o processo lógico da **API**, e é através dele que os **controllers** terão acesso a esse processo lógico.

Arquitetura de Aplicações Web

- C#: Handler

No C#, o handler vai ser basicamente uma classe que vai possuir diversos métodos com os diversos processo lógicos da API, e o controller vai acessar esses métodos para serem executados.

Por convenção, cria-se um handler para cada controller, com o mesmo nome do controller.