Aula 14

Prof: Henrique Augusto Maltauro

Codificar Back-end de Aplicações Web

C#

- IActionResult

Nas aulas passadas vimos o uso do IActionResult para padronizar as respostas das rotas, e facilitar a definição do Status Code da resposta.

Contudo, os IActionResult que aprendemos funcionam somente em controllers, pois é nativo da classe BaseController, a qual é herdada por padrão nos controllers.

C#

- IActionResult

Vamos rever os IActionResult que possam ser utilizados em todas as classes, e ter o entendimento de que esses novos IActionResult apresentados precisam todos de instanciamento, ou seja, precisa da palavra-chave new.

C#

- IActionResult (OkResult e OkObjectResult)

Equivalente ao IActionResult Ok temos o OkResult e o OkObjectResult para a resposta de código 200.

Sendo que o OkObjectResult recebe um objeto como parâmetro, que irá definir o corpo da resposta.

C#

- IActionResult (OkResult e OkObjectResult)

```
public IActionResult MetodoExemplo()
{
    // Status Code: 200
    return new OkResult();
}
```

C#

- IActionResult (OkResult e OkObjectResult)

```
public IActionResult MetodoExemplo()
{
    // Status Code: 200
    return new OkObjectResult("Objeto");
}
```

C#

- IActionResult (CreatedResult)

Equivalente ao IActionResult Created temos o CreatedResult para a resposta de código 201, recebendo obrigatoriamente dois parâmetros.

O primeiro parâmetro deve ser uma string para identificar a rota de GET do recurso criado, e o segundo é um objeto que irá definir o corpo da resposta.

C#

- IActionResult (CreatedResult)

```
public IActionResult MetodoExemplo()
{
    // Status Code: 201
    return new CreatedResult("api/Exemplo/1", "Objeto");
}
```

C#

- IActionResult (AcceptedResult)

Equivalente ao IActionResult Accepted temos o AcceptedResult para a resposta de código 202, podendo receber dois parâmetros.

O primeiro parâmetro deve ser uma string para identificar a rota de GET do recurso criado, e o segundo é um objeto que irá definir o corpo da resposta.

C#

- IActionResult (AcceptedResult)

```
public IActionResult MetodoExemplo()
{
    // Status Code: 202
    return new AcceptedResult();
}
```

C#

- IActionResult (AcceptedResult)

```
public IActionResult MetodoExemplo()
{
    // Status Code: 202
    return new AcceptedResult("api/Exemplo/1", "Objeto");
}
```

C#

- IActionResult (NoContentResult)

Equivalente ao IActionResult NoContent temos o NoContentResult para a resposta de código 204, e como o objetivo do código 204 é apresentar uma resposta sem conteúdo, ele não recebe nenhum parâmetro.

C#

- IActionResult (NoContentResult)

```
public IActionResult MetodoExemplo()
{
    // Status Code: 204
    return new NoContentResult();
}
```

C#

- IActionResult (RedirectResult)

Equivalente aos IActionResult Redirect, RedirectPreserveMethod, RedirectPermanent e RedirectPermanentPreserveMethod temos o RedirectResult para as respostas de código 301, 302, 307 e 308, recebendo obrigatoriamente um parâmetro string e dois parâmetros booleanos opcionais.

C#

- IActionResult (RedirectResult)

O parâmetro obrigatório indica um redirecionamento de rota, ele serve para indicar a nova rota que deve ser utilizada.

Os dois parâmetros booleanos opcionais irão definir qual é o código da resposta.

C#

- IActionResult (RedirectResult)

```
public IActionResult MetodoExemplo()
{
    // Status Code: 301
    return new RedirectResult("api/Exemplo/NovaRota", true);
}
```

C#

- IActionResult (RedirectResult)

```
public IActionResult MetodoExemplo()
{
    // Status Code: 302
    return new RedirectResult("api/Exemplo/NovaRota");
}
```

C#

- IActionResult (RedirectResult)

```
public IActionResult MetodoExemplo()
{
    // Status Code: 307
    return new RedirectResult("api/Exemplo/NovaRota", false, true);
}
```

C#

- IActionResult (RedirectResult)

```
public IActionResult MetodoExemplo()
{
    // Status Code: 308
    return new RedirectResult("api/Exemplo/NovaRota", true, true);
}
```

C#

- IActionResult (BadRequestResult e BadRequestObjectResult)

Equivalente ao IActionResult BadRequest temos o BadRequestResult e o BadRequestObjectResult para a resposta de código 400.

Sendo que o BadRequestObjectResult recebe um objeto como parâmetro, que irá definir o corpo da resposta.

C#

- IActionResult (BadRequestResult e BadRequestObjectResult)

```
public IActionResult MetodoExemplo()
{
    // Status Code: 400
    return new BadRequestResult();
}
```

C#

- IActionResult (BadRequestResult e BadRequestObjectResult)

```
public IActionResult MetodoExemplo()
{
    // Status Code: 400
    return new BadRequestObjectResult("Objeto");
}
```

C#

- IActionResult (UnauthorizedResult e UnauthorizedObjectResult)

Equivalente ao IActionResult Unauthorized temos o UnauthorizedResult e o UnauthorizedObjectResult para a resposta de código 401.

Sendo que o UnauthorizedObjectResult recebe um objeto como parâmetro, que irá definir o corpo da resposta.

C#

- IActionResult (UnauthorizedResult e UnauthorizedObjectResult)

```
public IActionResult MetodoExemplo()
{
    // Status Code: 401
    return new UnauthorizedResult();
}
```

C#

- IActionResult (UnauthorizedResult e UnauthorizedObjectResult)

```
public IActionResult MetodoExemplo()
{
    // Status Code: 401
    return new UnauthorizedObjectResult("Objeto");
}
```

C#

- IActionResult (NotFoundResult e NotFoundObjectResult)

Equivalente ao IActionResult NotFound temos o NotFoundResult e o NotFoundObjectResult para a resposta de código 404.

Sendo que o NotFoundObjectResult recebe um objeto como parâmetro, que irá definir o corpo da resposta.

C#

- IActionResult (NotFoundResult e NotFoundObjectResult)

```
public IActionResult MetodoExemplo()
{
    // Status Code: 404
    return new NotFoundResult();
}
```

C#

- IActionResult (NotFoundResult e NotFoundObjectResult)

```
public IActionResult MetodoExemplo()
{
    // Status Code: 404
    return new NotFoundObjectResult("Objeto");
}
```

C#

- IActionResult (ConflictResult e ConflictObjectResult)

Equivalente ao IActionResult Conflict temos o ConflictResult e o ConflictObjectResult para a resposta de código 409.

Sendo que o ConflictObjectResult recebe um objeto como parâmetro, que irá definir o corpo da resposta.

C#

- IActionResult (ConflictResult e ConflictObjectResult)

```
public IActionResult MetodoExemplo()
{
    // Status Code: 409
    return new ConflictResult();
}
```

C#

- IActionResult (ConflictResult e ConflictObjectResult)

```
public IActionResult MetodoExemplo()
{
    // Status Code: 409
    return new ConflictObjectResult("Objeto");
}
```

C#

- IActionResult (StatusCodeResult e ObjectResult)

Equivalente ao IActionResult StatusCode temos o StatusCodeResult e o ObjectResult para outros códigos de resposta.

Sendo que o StatusCodeResult recebe um int como parâmetro, que irá definir o código de resposta, e o ObjectResult recebe um objeto como parâmetro, que irá definir o corpo da resposta.

C#

- IActionResult (StatusCodeResult e ObjectResult)

```
public IActionResult MetodoExemplo()
{
    // Status Code: 500
    return new StatusCodeResult(500);
}
```

C#

- IActionResult (StatusCodeResult e ObjectResult)

```
public IActionResult MetodoExemplo()
{
    // Status Code: 500
    return new ObjectResult("Objeto") { StatusCode = 500 };
}
```

C#
Enum

C#

- Enum

O enum, ou tipo de enumeração, é um tipo de valor definido por um conjunto de constantes nomeadas do tipo numérico subjacente.

Ou seja, é um tipo de valor definido por um conjunto de valores numéricos, sendo que cada um desses números possui um nome.

C#

- Enum

São constantes fortemente tipadas e estáticas, ou seja, não é possível nem preciso acessar os seus valores instanciando um objeto.

Muito útil quando precisamos criar estruturas que serão pouco alteradas ao longo do desenvolvimento de uma aplicação.

C#

- Enum

Similar a todas as estruturas do C#, ele pode receber um modificador de acesso, é identificado por uma palavra-chave (enum), possui um nome como identificador, e é definido por chaves { }, onde dentro delas estarão todos os valores nomeados.

C#

- Enum

```
public enum StatusServico
{
    Novo,
    Aprovado,
    Recusado,
    EmProgresso,
    Cancelado,
    Concluido
}
```

C#

- Enum

O acesso é feito a partir do enum a ser utilizado, seguido do valor nomeado.

C#

- Enum

```
public class Servico
{
    public StatusServico Status { get; set; }

    public Servico()
    {
        Status = StatusServico.Novo;
    }
}
```

C#

- Enum

Por padrão, os valores das constantes nomeadas são do tipo int, sempre começando do valor zero e aumentando sequencialmente em um.

Porém, pode-se especificar explicitamente qualquer outro tipo numérico inteiro como um tipo subjacente de um tipo de enumeração, além de poder definir manualmente o valor de cada uma das constantes.

C#

- Enum

```
public enum StatusServico
{
    Novo = 1,
    Aprovado = 3,
    Recusado = 5,
    EmProgresso = 7,
    Cancelado = 9,
    Concluido = 11
}
```

C#

- Enum

```
public enum StatusServico : long
{
    Novo = 1,
    Aprovado = 3,
    Recusado = 5,
    EmProgresso = 7,
    Cancelado = 9,
    Concluido = 11
}
```

C#

- Enum

Importante lembrar que **o enum não é uma classe**, e sim um conjunto de valores constantes.

Sendo assim, não é possível definir atributos ou métodos dentro dele.

JSON

- C#: Serialização e Deserialização

Quando trabalhamos com enum no C#, precisamos de uma configuração extra para que o processo de serialização e deserialização do JSON funcione.

JSON

- C#: Serialização e Deserialização([JsonConverter])

O atributo de configuração [JsonConverter], vai marcar cada atributo da classe/objeto como tendo um processo de conversão do valor quando o mesmo passar pelo processo de serialização e deserialização. Ele recebe obrigatoriamente um parâmetro que irá definir o tipo de conversão a ser realizada.

No caso do enum utilizamos typeof(JsonStringEnumConverter).

JSON

- C#: Serialização e Deserialização([JsonConverter])

```
public class Servico
{
    [JsonPropertyName("status")]
    [JsonConverter(typeof(JsonStringEnumConverter))]
    public StatusServico Status { get; set; }

    public Servico()
    {
        Status = StatusServico.Novo;
    }
}
```

Desafio

Desafio

Pesquisar uma maneira de criar um método para utilizar a partir de um enum.