ControllerBase=>O mais basico do controller,porém para uma webAPI implementaremos com anotations o [ApiController]

Quando vc passa a anotação [Route("api/[controller]")] você esta dizendo que vai ser o nome da controller´(exemplo ValueController vai ser Value na rota)

que será utilizada,se você quiser por outro nome troque [controller] por outro nome.

não precisa passar ambos atributos [HttpGet]

[Route("{id:int}")]

Hoje podemos através do atributo de verbo especificar normamente exemplo:[HttpGet("{id:int}")]

ps:é importante especificar o tipo do parametro passado para evitarmos mal intencionados,assim o proprio HTTP já

bate o erro 404 caso não for um int por exemplo.

Podemos mudar também especificar no próprio HttpGet a rota que queremos exemplo:

[HttpGet("obter-por-id/{id:int}")],no

caso a rota ficaria a seguinte:

obter-por-id/5.

ActionResult=>é o resultado de uma Action.Você pode não usar o ActionResult para retornar o que deseja,mas,se voce precisar retornar um BadRequest,

voce precisa usar um ActionResult.exemplo:

[HttpGet]

public ActionResult<IEnumerable<string>> Get()

{

var valores = new string[] { "value1", "value2" };

if (valores.Length < 5000)

return BadRequest();

return Ok(valores);

}

Perceba que no método acima caso usarmos na assinatura somente public IEnumerable<string>Get() teriamos de retornar um null

caso a condição do if não for respeitada,o que traria um tratamento especial para o front.

Fora que utilizando ActionResult você pode utilizar um Ok(object value),que representa um 200 http code,e no Ok() no construtor dele

você pode passar um objeto que o mesmo será convertido para JSON.

Retornar um ActionResult é de boa pratica para retornos de solicitações de alteração,para o front comprender se aquela requisição

obteve sucesso ou não.

Caso você não quiser tipar seu ActionResult,porém,você precisa Passar através dos métodos ObjectsResults o retorno do mesmo como por exemplo:

[HttpGet]

public ActionResult GetValores()

{

var valores = new string[] { "value1", "value2" };

if (valores.Length < 5000)

return BadRequest();

return valores;

}

[HttpGet]

public ActionResult GetValores()

{

var valores = new string[] { "value1", "value2" };

if (valores.Length < 5000)

return BadRequest();

return Ok(valores);

}

Passamos nossa coleção para o método Ok que nos retornara um Resultado com nossa coleção.

[FromBody]=>ao usarmos essa anotação na frente de um parâmetro queremos dizer a WebApi que o valor passado para

o método esta no Corpo da requisição. Exemplo:

// POST api/values

[HttpPost]

public void Post([FromBody] string value)

{

}

ps:A partir do Asp.Net Core 2.1 não precisamos mais utilizar a anotação [FromBody] caso o parametro for um objeto complexo:

// POST api/values

[HttpPost]

public void Post(MeuObjetoComplexo value)

{

}

Sabendo que nosso objeto criado esta sendo passado no parametro o aspnet core identifica automaticamente que

irá vir no corpo da requisição.

[FromRoute]=>Você esta especificando que o parâmetro que o método espera esta vindo da rota.

Exemplo:

// PUT api/values/5

[HttpPut("{id}")]

public void Put([FromRoute]int id, [FromBody] string value)

{

}

ps:A partir da versão 2.1 do aspNet Core você não precisa utilizar o FromRoute caso você especifique na anottation que

você irá receber um id como parametro.Exemplo explicativo abaixo:

// PUT api/values/5

[HttpPut("{id}")]/\* especificamos que na rota iremos receber um id\*/

public void Put(int id/\* Como esta especificado acima não preciso colocar [FromRoute]\*/, [FromBody] string value)

{

}

[FromForm]=>você esta especificando ao método que o dado esta vindo atraves de um Form utilizando o atributo FormData

do request.a requisição do front consta no content type que você esta enviando um FormData.Exemplo:

// PUT api/values/5

[HttpPut("{id}")]

public void Put([FromRoute]int id, [FromForm] string value)

{

}

[FromHeader]=>quando sabemos que a requisição irá mandar algum dado que esta dentro da header então especificamos

o FromHeader.Exemplo:

// DELETE api/values/5

[HttpDelete("{id}")]

public void Delete([FromHeader]int id)

{

}

[FromQuery]=>Quando você tem algum atributo sendo passado via Query String porém ele não faz parte da rota.Exemplo:

[HttpDelete("{id}")]

public void Delete([FromQuery]int id)

{

}

Suponhamos que ele não esteja sendo suportado na rota,porém sabemos que na requisição ele esta sendo passado via query, então, usaremos o fromQuery nesse exemplo, lembrando que o parâmetro da queryString tem que ser IGUAL(nomenclatura) ao parâmetro esperado na WebApi.

[FromServices]=>Ele não é sobre como receber um dado e sim para voce injetar um interface ou uma classe e o mesmo fará a resolução

via injeção de dependencia.Lembrando que o mesmo não é uma funcionalidade do protocolo do Http e sim uma funcionalidade

do ASP NET Core.

Http code 201=> ok e criado

ProducesResponseType=>Especificar e para documentadores automáticos de APIs como Swagger por exemplo,o mesmo ja entende

quais são os tipos de retornos possiveis e documentar para consumidores da API.

[HttpPost]

[ProducesResponseType(typeof(Produto),StatusCodes.Status201Created)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public ActionResult Post(Produto product)

{

if (product.Id == 0)

return BadRequest();

//add no banco

//Produz um ActionResult 201

return CreatedAtAction(actionName: nameof(Post), product);

}