

Dell IT Academy



Recursos

- Documentação
 - <https://learn.microsoft.com/en-us/aspnet/core>
- Fontes
 - <https://github.com/dotnet/aspnetcore>
- Exemplos
 - <https://www.todobackend.com/>

ASP.NET CORE WEB API

O que é?

- ASP.NET Core Web API é um dos frameworks .NET para o desenvolvimento de Web Services no estilo REST

CONTROLADOR

Controlador e Ações

- Um controlador (uma classe) é utilizado para definir um grupo de ações
- Uma ação (um método) é utilizado para tratar uma requisição
 - Todos métodos públicos definem uma ação por padrão
- Requisições são mapeadas para controladores e ações via roteamento
- **IMPORTANTE:** controlador é uma abstração de nível de camada de interface, evitar ao máximo implementar lógica de negócio em um método de ação

Regras de Roteamento

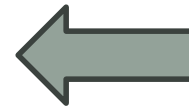
- O Web Service necessita mapear as requisições HTTP ao código que irá trata-las
 - URI de requisição
 - Método (verbo) do HTTP
- ASP.NET Core MVC possui um *middleware* de gerenciamento de rotas
- Ver:
 - <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/routing>
 - <https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/routing>

Regras de Roteamento

- Regras criadas utilizando tabela de roteamento

```
app.MapControllerRoute(  
    name: "default",  
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

- Regras criadas através de atributos



```
[Route("customers/{customerId}/orders")]  
public IEnumerable<Order> GetOrdersByCustomer(int  
customerId) { ... }
```


Roteamento - Métodos de Ação

- Ações são métodos públicos do *controller* que respondem a requisições do HTTP
- Ações são mapeadas com base em:
 - O método HTTP utilizado na requisição
 - Ex.: GET, PUT, POST, DELETE, etc
 - Regras de mapeamento de ação no roteamento

Roteamento - Métodos de Ação

```
public class ProductsApiController : ControllerBase
{
    [HttpGet("/products/{id}", Name = "Products_List")]
    public IActionResult GetProduct(int id) { ... }
}
```

```
[Route("products")]
public class ProductsApiController : ControllerBase
{
    [HttpGet]
    public IActionResult ListProducts() { ... }

    [HttpGet("{id}")]
    public ActionResult GetProduct(int id) { ... }
}
```

Roteamento - Métodos de Ação

- Rotas suportam *tokens de substituição*:
 - [action], [area], [controller]

```
[Route("[controller]/[action]")]
public class ProductsController : ControllerBase
{
    [HttpGet] // Matches '/Products/List'
    public IActionResult List() {
        // ...
    }

    [HttpGet("{id}")] // Matches '/Products/Edit/{id}'
    public IActionResult Edit(int id) {
        // ...
    }
}
```

Roteamento - Métodos de Ação

- É possível atribuir restrições sobre as regras de mapeamento dos parâmetros
- Formato geral *{parâmetro:restrição}*

```
[HttpPost("product/{id:int}")]  
public IActionResult ShowProduct(int id)  
{  
    // ...  
}
```

- Ver <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/routing#route-template-reference>

AÇÕES

Parâmetros

- ASP.NET Core mapeia dados de uma requisição HTTP em parâmetros de métodos de ação
- É um processo conhecido como *model binding*
- Busca por valores segue uma ordem:
 - Valores de formulário (corpo da requisição POST)
 - Valores em templates de roteamento
 - Valores de *query string* (requisição GET)
- Ação deve verificar status explicitamente:
 - Propriedade *ModelState.IsValid*
- Ver <https://learn.microsoft.com/en-us/aspnet/core/mvc/models/model-binding>

Parâmetros

- Diversos atributos controlam a vinculação de dados a partir da requisição HTTP:
 - [BindRequired] – resulta em status de erro se a vinculação não ocorrer
 - [BindNever] – vinculação não deve ocorrer
 - [FromHeader], [FromQuery], [FromRoute], [FromForm], [FromBody] – indicam a fonte específica da vinculação
 - [FromServices] – indica vinculação via o serviço de injeção de dependências

Parâmetros

```
[HttpPost]
public IActionResult Create([FromBody] TodoItem item)
{
    if (item == null)
    {
        return BadRequest();
    }

    _context.TODOItems.Add(item);
    _context.SaveChanges();

    return CreatedAtRoute("GetTodo", new { id = item.Id },
item);
}
```


Resultados

- ASP.NET Core converte o valor de retorno de métodos de ação em mensagem de retorno do HTTP
- Tipos de retorno:
 - tipo específico
 - *ActionResult*
 - *ActionResult<T>*
- Ver <https://learn.microsoft.com/en-us/aspnet/core/web-api/action-return-types>

Resultados

```
[HttpGet]  
public IEnumerable<Product> Get()  
{  
    return _repository.GetProducts();  
}
```

Resultados

- *ActionResult* é uma interface que representa um *factory* de objetos *ActionResult* o qual representa diferentes respostas do HTTP
 - *BadRequestResult*, *NotFoundResult*, *OkObjectResult*, etc
- Vantagens:
 - Separação do código do controlador do código de criação das respostas do HTTP
 - Encapsula detalhes de baixo nível da criação de mensagens de resposta
 - Facilita o teste unitário dos controladores

Resultados

```
[HttpGet("{id}")]
[ProducesResponseType(200, Type = typeof(Product))]
[ProducesResponseType(404)]
public IActionResult GetById(int id)
{
    if (!_repository.TryGetProduct(id, out var product))
    {
        return NotFound();
    }

    return Ok(product);
}
```

Ações Assíncronas

- Utilizam uma combinação das palavras reservadas *async*, *await* e o tipo *Task*

```
[HttpPost]
[ProducesResponseType(201, Type = typeof(Product))]
[ProducesResponseType(400)]
public async Task<IActionResult> CreateAsync([FromBody]
Product product)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    await _repository.AddProductAsync(product);

    return CreatedAtAction(nameof(GetById), new { id =
product.Id }, product);
}
```

Resultados

- *ActionResult*<*T*> está disponível somente a partir da versão 2.1
- Vantagens:
 - Metadados de descrição de resultado podem ser excluídos
 - Conversão implícita para o tipo de retorno correto

Resultados

```
[HttpGet("{id}")]
[ProducesResponseType(200)]
[ProducesResponseType(404)]
public ActionResult<Product> GetById(int id)
{
    if (!_repository.TryGetProduct(id, out var product))
    {
        return NotFound();
    }

    return product;
}
```

SERIALIZAÇÃO

Serialização

- ASP.NET Core tem suporte padrão para os seguintes tipos de dados na serialização:
 - JSON – classe *JsonResult* e factory *Json*
 - XML – deve ser configurado explicitamente como *middleware*
 - texto – classe *ContentResult* e factory *Content*
- Tipos adicionais podem ser configurados através de formataadores customizados
- Ver:
 - <https://learn.microsoft.com/en-us/aspnet/core/web-api/advanced/formatting>
 - <https://learn.microsoft.com/en-us/aspnet/core/web-api/advanced/custom-formatters>

Serialização

```
// GET: api/authors
[HttpGet]
public JsonResult Get()
{
    return Json(_authorRepository.List());
}
```

```
// GET api/authors/about
[HttpGet("About")]
public ContentResult About()
{
    return Content("An API listing authors of docs.asp.net.");
}
```

Serialização

- ASP.NET Core suporta o processo de negociação de conteúdo do protocolo HTTP
 - Cabeçalho *Accept* define o tipo de conteúdo solicitado
 - Se não for solicitado, padrão do framework é JSON
- Para forçar um tipo específico de formatador na serialização utiliza-se o atributo *[Produces]*
 - Aplicado ao controlador, ação ou escopo global

```
[Produces("application/json")]  
public class AuthorsController/
```