**AspNet Core WebAPI:**

***API com controladores:***

***Resumo:***

Uma API Web baseada em controlador consiste em uma ou mais classes de controlador que derivam de ControllerBase.

A classe ControllerBase fornece muitas propriedades e métodos úteis para lidar com solicitações HTTP. Por exemplo, CreatedAtAction retorna um código de status 201 .

BadRequest – 400

NotFound – 404

***Atributos:***

O namespace Microsoft.AspNetCore.Mvc fornece atributos que podem ser usados para configurar o comportamento de controladores de API Web e dos métodos de ação.

*Ex:*

[HttpPost]

[ProducesResponseType(StatusCodes.Status201Created)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

*Outros:*

*[Route] – Especifica o padrão de URL para um controlador ou ação.*

*[HttpGet] - Identifica uma ação que dá suporte ao verbo de ação HTTP GET.*

***Contexto de Banco de Dados (DbContext):***

Classe principal que coordena as funcionalidades do Entity Framework para um modelo de dados.

***Scaffolding (With actions using Entity Framework):***

Ferramenta do EntityFramework que gera um Crud completo.

***DTO (Data Transfer Object):***

Subconjunto do modelo principal. É um modelo de entrada ou exibição. Pode ser usado para omitir dados sensíveis, entre outras utilidades.

***Db Browser:***

Software de navegação para banco de dados SqLite

***Record:***

Classe imutável. Bom para uso em DTO´s.

***Alguns Tipos de retorno da API:***

(Mais em: [Tipos de retorno de ação do controlador na API Web do ASP.NET Core | Microsoft Learn](https://learn.microsoft.com/pt-br/aspnet/core/web-api/action-return-types?view=aspnetcore-8.0))

***EspecÍfico(básico):***

Ex:

[HttpGet]

public Task<List<Product>> Get() =>

\_productContext.Products.OrderBy(p => p.Name).ToListAsync();

***IAsyncEnumerable:***

Ex:

[HttpGet("asyncsale")]

public async IAsyncEnumerable<Product> GetOnSaleProductsAsync()

{

var products = \_productContext.Products.OrderBy(p => p.Name).AsAsyncEnumerable();

await foreach (var product in products)

{

if (product.IsOnSale)

{

yield return product;

}

}

}

***IActionResult:***

Ex:

[HttpPost()]

[Consumes(MediaTypeNames.Application.Json)]

[ProducesResponseType(StatusCodes.Status201Created)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public async Task<IActionResult> CreateAsync\_IActionResult(Product product)

{

if (product.Description.Contains("XYZ Widget"))

{

return BadRequest();

}

\_productContext.Products.Add(product);

await \_productContext.SaveChangesAsync();

return CreatedAtAction(nameof(CreateAsync\_IActionResult), new { id = product.Id }, product);

}

***ActionResult:***

Ex:

[HttpPost()]

[Consumes(MediaTypeNames.Application.Json)]

[ProducesResponseType(StatusCodes.Status201Created)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public async Task<ActionResult<Product>> CreateAsync\_ActionResultOfT(Product product)

{

if (product.Description.Contains("XYZ Widget"))

{

return BadRequest();

}

\_productContext.Products.Add(product);

await \_productContext.SaveChangesAsync();

return CreatedAtAction(nameof(CreateAsync\_ActionResultOfT), new { id = product.Id }, product);

}

***Opção para CreatedAtAction(Fornece o código 201 created):***

return new CreatedAtActionResult(nameof(CreateAsync),

"Products",

new { id = product.Id },

product);

***HttpPatch com JsonPatch:***

***Step by step(Sem alterar os outros formatadores):***

*Nuget:*

Microsoft.AspnetCore.Mvc.NewtonsoftJson

Microsoft.AspnetCore.JsonPatch

*Add Builders:*

builder.Services.AddControllers(options =>

{

options.InputFormatters.Insert(0, MyJPIF.GetJsonPatchInputFormatter());

});

Add class Sample:

([JsonPatch na API Web do ASP.NET Core | Microsoft Learn](https://learn.microsoft.com/pt-br/aspnet/core/web-api/jsonpatch?view=aspnetcore-8.0))

*Exemplo de uso:*

[HttpPatch]

public IActionResult JsonPatchWithModelState(

[FromBody] JsonPatchDocument<Customer> patchDoc)

{

if (patchDoc != null)

{

var customer = CreateCustomer();

patchDoc.ApplyTo(customer, ModelState);

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

return new ObjectResult(customer);

}

else

{

return BadRequest(ModelState);

}

}

*Para as classes:*

namespace JsonPatchSample.Models;

public class Customer

{

public string? CustomerName { get; set; }

public List<Order>? Orders { get; set; }

}

namespace JsonPatchSample.Models;

public class Order

{

public string OrderName { get; set; }

public string OrderType { get; set; }

}

***Walkthrough:***

- Criar projeto Web API (ex: TodoApi):

- Adicionar Nuget Entity.Framework:

Para testes – Microsoft.EntityFramework.InMemory

SQLite+Migrations - Microsoft.EntityFrameworkCore + Microsoft.EntityFrameworkCore.Design + Microsoft.EntityFrameworkCore.Sqlite

- Criar classe Model:

Api > Nova Pasta > “Models”;

Models > new class > TodoItem.cs

- Criar DbContext:

Models > new class > TodoContext.cs

- Registrar DbContext na classe program.cs:

using Microsoft.EntityFrameworkCore;

using TodoApi.Models;

Para teste:

* builder.Services.AddDbContext<TodoContext>(opt =>

opt.UseInMemoryDatabase("TodoList"));

Para Sqlite:

1. No appsettings.json:

{

“ConnectionStrings”:{

“DefaultConnection”: “Data Source = sqlite.db”

},

1. Na classe program.cs:

* builder.Services.AddDbContext<TodoContext>(opt =>

opt.UseSqLite(builder.Configuration.GetConnectionString(“DefaultConnection”)));

- Criar Migrations (cd deve estar na api):

No terminal – dotnet ef migrations add <nomeDaMigrations>

(dotnet ef migrations remove <nomeDaMigrations> para deletar)

- Criar Db Sqlite:

No terminal – dotnet ef database update

- Fazer Scaffold:

Controllers > Add > New Scaffold Item > controlador de API com ações, usando Entity Framework >

* TodoItem – Classe Modelo
* TodoContext – Classe de contexto de dados > Adicionar.