📘 Guia Completo: Criando uma Web API com C# (ASP.NET Core)

🛠️ 1. Pré-requisitos

Antes de começar, instale:

.NET SDK (recomendado: versão 7 ou 8)

Visual Studio 2022+ ou Visual Studio Code

Postman (para testar a API)

⚙️ 2. Criando o Projeto

Via terminal ou CMD:

```
bash
CopiarEditar
dotnet new webapi -n MinhaApi
cd MinhaApi
```

Via Visual Studio:

Novo Projeto → ASP.NET Core Web API

Desmarque "Enable OpenAPI Support" se não quiser Swagger

Selecione .NET 7/8

🧱 3. Estrutura do Projeto

```
bash
CopiarEditar
/Controllers        → Endpoints da API
/Models             → Classes de dados
```

```
/Data               → Contexto do banco (EF Core)
/DTOs (opcional)    → Objetos para entrada/saída de dados
/Services (opcional)→ Lógica de negócio
Program.cs          → Configuração da aplicação
appsettings.json    → Configurações (como banco de dados)
```

📄 4. Criando o Modelo

```
csharp
CopiarEditar
// Models/Produto.cs
public class Produto
{
    public int Id { get; set; }
    public string Nome { get; set; } = string.Empty;
    public decimal Preco { get; set; }
}
```

🗄️ 5. Configurando o Banco de Dados (EF Core + SQLite)

Instalar pacotes:

```
bash
CopiarEditar
dotnet add package Microsoft.EntityFrameworkCore.Sqlite
dotnet add package Microsoft.EntityFrameworkCore.Tools
```

Criar contexto:

```
csharp
CopiarEditar
// Data/AppDbContext.cs
using Microsoft.EntityFrameworkCore;
using MinhaApi.Models;

public class AppDbContext : DbContext
{
```

```
    public AppDbContext(DbContextOptions<AppDbContext> options) :
base(options) {}
    public DbSet<Produto> Produtos => Set<Produto>();
}
```

Adicionar ao Program.cs:

```
csharp
CopiarEditar
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlite("Data Source=produtos.db"));
```

Criar e aplicar migrations:

```
bash
CopiarEditar
dotnet ef migrations add Inicial
dotnet ef database update
```

🚀 6. Criando o Controller

```
csharp
CopiarEditar
// Controllers/ProdutosController.cs
using Microsoft.AspNetCore.Mvc;
using MinhaApi.Models;
using Microsoft.EntityFrameworkCore;

[ApiController]
[Route("api/[controller]")]
public class ProdutosController : ControllerBase
{
    private readonly AppDbContext _context;
    public ProdutosController(AppDbContext context) => _context =
context;

    [HttpGet]
    public async Task<ActionResult<IEnumerable<Produto>>> Get() =>
```

```
        await _context.Produtos.ToListAsync();

    [HttpGet("{id}")]
    public async Task<ActionResult<Produto>> Get(int id)
    {
        var produto = await _context.Produtos.FindAsync(id);
        return produto == null ? NotFound() : Ok(produto);
    }

    [HttpPost]
    public async Task<ActionResult<Produto>> Post(Produto produto)
    {
        _context.Produtos.Add(produto);
        await _context.SaveChangesAsync();
        return CreatedAtAction(nameof(Get), new { id = produto.Id },
produto);
    }

    [HttpPut("{id}")]
    public async Task<IActionResult> Put(int id, Produto produto)
    {
        if (id != produto.Id) return BadRequest();
        _context.Entry(produto).State = EntityState.Modified;
        await _context.SaveChangesAsync();
        return NoContent();
    }

    [HttpDelete("{id}")]
    public async Task<IActionResult> Delete(int id)
    {
        var produto = await _context.Produtos.FindAsync(id);
        if (produto == null) return NotFound();
        _context.Produtos.Remove(produto);
        await _context.SaveChangesAsync();
        return NoContent();
    }
}
```

🧪 7. Testando com Swagger ou Postman

Execute o projeto: `dotnet run`

Acesse: https://localhost:5001/swagger (ou use o Postman)

Teste rotas: GET, POST, PUT, DELETE

-

🧰 8. Validação de Dados

```
csharp
CopiarEditar
using System.ComponentModel.DataAnnotations;

public class Produto
{
    public int Id { get; set; }

    [Required]
    public string Nome { get; set; } = string.Empty;

    [Range(0.01, 9999.99)]
    public decimal Preco { get; set; }
}
```

⚠️ 9. Tratamento de Erros

Middleware global (Program.cs):

```
csharp
CopiarEditar
app.UseExceptionHandler(errorApp =>
{
    errorApp.Run(async context =>
    {
        context.Response.StatusCode = 500;
        context.Response.ContentType = "application/json";
        await context.Response.WriteAsync("{\"erro\":\"Ocorreu um
```

```
erro interno.\"}");
    });
});
```

🔐 10. Autenticação e Autorização (Opcional)

Pode usar JWT com `Microsoft.AspNetCore.Authentication.JwtBearer`

Configurar políticas de autorização no `Program.cs`

Decorar controllers com `[Authorize]` ou `[AllowAnonymous]`

📦 11. Publicação e Deploy

Publicar local: `dotnet publish -c Release`

Deploy comum: Azure, AWS, Railway, Render ou IIS

📚 Recursos Úteis

Docs oficiais: https://learn.microsoft.com/aspnet/core

Cursos grátis: Balta.io, Dev.io, Youtube (dotnet, C# WebAPI)

Exemplos prontos: GitHub - `aspnet/samples`, `devmentors`, etc.