FMX Academy

INFORMAÇÕES			
MÓDULO		**AULA**	**INSTRUTOR**
06	**ASP.NET Core Web API – Avançado e Segurança Inicial**	**01 - Aula de Revisão: ASP.NET Core Web API (Módulo 05)**	**Guilherme Paracatu**

1. Aula de Revisão: ASP.NET Core Web API (Módulo 05)

Objetivo: Consolidar os pilares de APIs REST, EF Core 8 e Arquitetura

FERRAMENTAS: VS Code, Projeto ApiEstoque, PostgreSQL.

1.1. O que é uma API?

- **O Garçom Digital:** A API é o contrato que define como softwares diferentes (frontend, mobile) conversam com o backend.
- **Abstração:** Ela esconde a complexidade da "cozinha" (servidor). O cliente pede um dado e a API entrega, sem que o cliente precise saber como o banco de dados funciona.

1.2. REST: O Estilo Arquitetural

- **Não é protocolo:** São "regras de etiqueta" para a web ser escalável e padronizada.
- **Recursos:** Tudo é um recurso (Tarefa, Usuário) identificado por uma URI.
  - **Exemplo de URI:** /api/tarefas (Sempre use substantivos, nunca verbos).
- **Representações:** Trocamos "fotos" do estado do recurso, geralmente em formato JSON.

1.3. O Protocolo HTTP como Base

- **Stateless (Sem Estado):** O servidor não guarda sessão do cliente. Cada requisição deve ser completa e conter tudo o que o servidor precisa saber. Isso facilita o balanceamento de carga e a escalabilidade.
- **Verbos HTTP (Ações CRUD):**
  - **GET:** Ler dados.
  - **POST:** Criar um novo recurso.
  - **PUT:** Substituir um recurso inteiro (Cuidado: pode apagar dados se o objeto enviado estiver incompleto).
  - **DELETE:** Remover um recurso.
  - **PATCH:** Atualizar apenas partes de um recurso.

1.4. Códigos de Status (O Feedback)

- **2xx (Sucesso):** 200 OK, 201 Created (após POST), 204 No Content (após DELETE/PUT).
- **4xx (Erro do Cliente):** 400 Bad Request (erro de validação), 401 Unauthorized, 404 Not Found.
- **5xx (Erro do Servidor):** 500 Internal Server Error.

2. Entity Framework Core 8 e Performance

2.1. O ORM e o DbContext

- **O Tradutor:** O EF Core mapeia suas classes C# para tabelas SQL.
- **DbContext:** É a "central de comando". Ele gerencia a conexão, rastreia alterações e traduz LINQ para SQL.

2.2. Mapeamento de Dados

- **Data Annotations:** Atributos como [Table] ou [Key] direto na classe. São simples, mas "sujam" a entidade.
- **Fluent API:** Configuração feita no OnModelCreating dentro do DbContext. **É o método preferível**, pois centraliza a configuração e mantém as entidades limpas.

2.3. Relacionamentos

- **1:N (Um para Muitos):** Ex: Um Usuário tem várias Tarefas.
- **N:N com Payload:** Ex: Tarefas e Tags via TarefaTag. Requer uma entidade de junção e dois relacionamentos 1:N na Fluent API.

2.4. Performance: O Vilão N+1

- **O Problema:** Fazer 1 consulta para a lista e mais "N" consultas para os dados relacionados em um loop (causado por Lazy Loading).
- **A Solução (Eager Loading):** Use .Include(x => x.Propriedade) para trazer tudo em um único JOIN no SQL.

2.5. DTOs e AutoMapper

- **Por que usar?** NUNCA exponha suas entidades do banco diretamente na API.
- **Vantagens:** Resolve referências cíclicas (loops no JSON), aumenta a segurança e desacopla a API da estrutura do banco.
- **AutoMapper:** Automatiza essa conversão entre Entidade e DTO.

2.6. Arquitetura em Camadas

- **Controller:** "Magro". Recebe o HTTP, chama o Service e retorna o resultado.
- **Service:** "O Cérebro". Contém a lógica de negócio, validações e orquestra os dados.
- **Repository:** "O Braço". Acesso exclusivo aos dados via DbContext.

2.7. Injeção de Dependência (DI)

- Baseado no princípio DIP (SOLID), usamos Interfaces (IService, IRepository) para que as camadas não dependam de classes concretas.
- Configurado via builder.Services.AddScoped<> no arquivo Program.cs.

Desafio Prático: Refatoração de API (PostgreSQL + EF Core 8)

O Cenário (Código "Sujo")
Projete este trecho de código no Visual Studio. Ele contém erros conceituais graves que discutimos na revisão:

Script SQL: Preparando o Terreno (PostgreSQL)

```
-- Criar tabelas
CREATE TABLE Fabricantes (
    Id SERIAL PRIMARY KEY,
    Nome VARCHAR(100) NOT NULL
);

CREATE TABLE Produtos (
    Id SERIAL PRIMARY KEY,
    Nome VARCHAR(100) NOT NULL,
    Preco DECIMAL(10,2) NOT NULL,
    FabricanteId INTEGER REFERENCES Fabricantes(Id)
);

-- Popular dados aleatórios
INSERT INTO Fabricantes (Nome) VALUES ('TechMaster'), ('Global Industries');
INSERT INTO Produtos (Nome, Preco, FabricanteId) VALUES
('Teclado Mecânico', 250.00, 1),
('Mouse Gamer', 150.00, 1),
('Monitor 4K', 1200.00, 2);
```

```
// EXEMPLO DE COMO NÃO FAZER
[ApiController]
[Route("api/PegarTarefas")] // Erro 1: URI com verbo
public class TarefaController : ControllerBase
{
    private readonly AppDbContext _context;

    public TarefaController(AppDbContext context) // Erro 2: Injetando contexto direto
no Controller
    {
        _context = context;
    }

    [HttpGet]
    public List<Tarefa> Get() // Erro 3: Retornando a Entidade do banco diretamente
    {
        // Erro 4: Problema N+1 (não tem .Include para os dados do Usuário)
        var tarefas = _context.Tarefas.ToList();
        return tarefas;
    }
}
```

Desafio:
Resolvam os seguintes pontos no projeto:

1. **Corrigir a Rota e o Contrato (REST)**
   - A URI não deve ter verbos como "ListarProdutos".
   - **Ação:** Altere a rota para apenas api/produtos (usando o substantivo).

2. **Blindar a API (DTOs)**
   - Nunca exponha a Entidade Produto diretamente, pois isso gera risco de segurança e referências cíclicas.
   - **Ação:** Criar um ProdutoDTO e garantir que o Controller retorne apenas esse objeto.

3. **Salvar a Performance (Eager Loading)**
   - O código acima sofre do "Problema N+1", o vilão da performance que faz múltiplas consultas desnecessárias ao banco.
   - **Ação:** Utilizar o método .Include(x => x.Usuario) na consulta LINQ para buscar os dados relacionados em um único JOIN.

4. **Respeitar as Camadas (Arquitetura)**
   - O Controller está "gordo", fazendo o papel do Repository e do Service ao acessar o DbContext diretamente.
   - **Ação:** Criar/Ajustar uma **Interface** IProdutoRepository.
     - Mover o acesso ao banco para a camada de **Repository**.
     - Injetar a Interface no Controller via construtor (Injeção de Dependência).