Avaliação - Módulo 2: ASP.NET Core Web API – Fundamentos

- Esta avaliação é dividida em duas partes: Questões Teóricas (30%) e Aplicação Prática (70%).
- Responda às questões teóricas de forma clara e objetiva.
- Para a Aplicação Prática, siga as instruções detalhadas.
- **Entrega Mínima (Hoje):** A estrutura do projeto da API e os endpoints GET (listar todos/por ID) e POST devem estar funcionais.
- **Entrega Completa (Até a próxima semana):** API com CRUD completo (GET, POST, PUT, DELETE) e seguindo as boas práticas de arquitetura discutidas.

Parte 1: Questões Teóricas (Total: 30 pontos - 6 pontos por questão)

Responder no Forms

Parte 2: Aplicação Prática - API GerenciadorTarefasApi (Total: 70 pontos)

Cenário: Você criará uma API REST para um **Gerenciador de Tarefas Pessoais**. Esta API permitirá criar usuários, gerenciar tarefas (com detalhes opcionais) e categorizá-las com tags. Ela servirá de backend para um aplicativo frontend no próximo módulo.

Requisitos Gerais:

- Utilizar .NET 8 e C#.
- Seguir a arquitetura em camadas (Controllers, Services, Repositories, Entities) com a estrutura de pastas especificada.
- Implementar Injeção de Dependência (DI) em todas as camadas.
- Usar Entity Framework Core 8 com **PostgreSQL**.
- Configurar mapeamentos e relacionamentos usando **Fluent API** no DbContext.
- Utilizar **DTOs** para todas as entradas e saídas da API.
- Implementar **AutoMapper** para o mapeamento entre Entidades e DTOs.
- A validação de entrada deve ser feita via **Data Annotations** nos DTOs (aproveitando o [ApiController]).
- Configurar e usar **Git** para versionamento.

Etapa 1: Configuração Inicial e Estrutura (Dia da Prova)

(Peso: 10 Pontos)

1. **Inicializar Repositório Git:**
   - Crie uma pasta para o seu projeto no seu computador.
   - Abra o terminal **nessa pasta** e execute:

```
git init
git checkout -b main
#Crie um arquivo .gitignore para .NET e adicione-o
git add .gitignore
```

```
 git commit -m "feat: Adiciona .gitignore para .NET"
git checkout -b develop # Criar branch de desenvolvimento
```

- **Explicação:** Iniciamos o controle de versão, criamos a branch principal (main) e uma branch (develop) onde faremos nosso trabalho.

2. **Criação do Projeto .NET:**
   - Ainda no terminal, na mesma pasta, execute:

```
dotnet new webapi -n GerenciadorTarefasApi -f net8.0 --use-controllers
```

   - Abra a pasta GerenciadorTarefasApi no VS Code.
3. **Instalação de Pacotes NuGet:**
   - Abra o terminal integrado do VS Code (Ctrl + ') e execute:

```
# EF Core para PostgreSQL
dotnet add package Npgsql.EntityFrameworkCore.PostgreSQL
# Ferramentas do EF Core
dotnet add package Microsoft.EntityFrameworkCore.Tools
dotnet add package Microsoft.EntityFrameworkCore.Design
# AutoMapper
dotnet add package AutoMapper.Extensions.Microsoft.DependencyInjection
```

4. **Estrutura de Pastas:**
   - Recrie a estrutura de pastas conforme a fornecida (remova os arquivos/pastas template Weather*):

```
GerenciadorTarefasApi/
├── Properties/
├── Controllers/
├── Entities/
├── Infra/
│   ├── Context/
│   ├── DTOs/
│   ├── Mapping/
│   └── Repositories/
│       └── Interfaces/
├── Services/
│   └── Interfaces/
├── appsettings.json
├── appsettings.Development.json
├── Program.cs
└── GerenciadorTarefasApi.csproj
```

5. **Commit Inicial:**
   - Faça o primeiro commit na branch develop:

```
git add .
git commit -m "feat: Inicializa estrutura do projeto GerenciadorTarefasApi"
```

Etapa 2: Configuração do Banco de Dados (Dia da Prova)

(Peso: 10 Pontos)

1. **Script SQL (PostgreSQL):**

- Crie um arquivo database_script.sql na raiz do projeto (ou use uma ferramenta como DBeaver/pgAdmin) e execute o script abaixo em um banco de dados PostgreSQL chamado GerenciadorTarefasDb.

SQL

```
-- Tabela de Usuários (Exemplo de entidade principal)
CREATE TABLE "TB_USUARIOS" (
    "id_usuario" SERIAL PRIMARY KEY,
    "nome_usuario" VARCHAR(100) NOT NULL,
    "email_usuario" VARCHAR(150) NOT NULL UNIQUE
);

-- Tabela de Tarefas (Relacionamento 1:N com Usuario)
CREATE TABLE "TB_TAREFAS" (
    "id_tarefa" SERIAL PRIMARY KEY,
    "titulo_tarefa" VARCHAR(100) NOT NULL,
    "descricao_tarefa" VARCHAR(500) NULL,
    "data_criacao" TIMESTAMP WITH TIME ZONE NOT NULL,
    "data_conclusao" TIMESTAMP WITH TIME ZONE NULL,
    "concluida" BOOLEAN NOT NULL DEFAULT false,
    "id_usuario" INT NOT NULL, -- Chave Estrangeira para Usuario
    CONSTRAINT "fk_usuario_tarefa" FOREIGN KEY ("id_usuario") REFERENCES
"TB_USUARIOS"("id_usuario")
);

-- Tabela de Detalhes da Tarefa (Relacionamento 1:1 com Tarefa)
CREATE TABLE "TB_DETALHES_TAREFA" (
    "id_tarefa" INT PRIMARY KEY, -- Mesma chave primária de Tarefas
    "prioridade" INT NOT NULL DEFAULT 0, -- Ex: 0=Baixa, 1=Média, 2=Alta
    "notas_adicionais" TEXT NULL,
    CONSTRAINT "fk_tarefa_detalhes" FOREIGN KEY ("id_tarefa") REFERENCES
"TB_TAREFAS"("id_tarefa") ON DELETE CASCADE -- Se deletar a tarefa, deleta os detalhes
);

-- Tabela de Tags
CREATE TABLE "TB_TAGS" (
    "id_tag" SERIAL PRIMARY KEY,
    "nome_tag" VARCHAR(50) NOT NULL UNIQUE
);

-- Tabela de Junção Tarefa <-> Tag (Relacionamento N:N)
CREATE TABLE "TB_TAREFAS_TAGS" (
    "id_tarefa" INT NOT NULL,
    "id_tag" INT NOT NULL,
    PRIMARY KEY ("id_tarefa", "id_tag"),
    CONSTRAINT "fk_tarefatag_tarefa" FOREIGN KEY ("id_tarefa") REFERENCES
"TB_TAREFAS"("id_tarefa") ON DELETE CASCADE,
    CONSTRAINT "fk_tarefatag_tag" FOREIGN KEY ("id_tag") REFERENCES
"TB_TAGS"("id_tag") ON DELETE CASCADE
);

-- Dados Iniciais (Opcional, mas útil para testes)
INSERT INTO "TB_USUARIOS" ("nome_usuario", "email_usuario") VALUES ('Admin User',
'admin@tarefas.com');
INSERT INTO "TB_TAGS" ("nome_tag") VALUES ('Trabalho'), ('Pessoal'), ('Urgente');
```

2. **Connection String:**
   - Configure a connection string no appsettings.Development.json:

```
JSON
{
  // ...
  "ConnectionStrings": {
    "DefaultConnection":
"Host=localhost;Database=GerenciadorTarefasDb;Username=postgres;Password=SUA_SENHA_DO_POS
TGRES"
  }
}
```

Etapa 3: Implementação da API (Entrega Mínima - Dia da Prova)

(Peso: 30 Pontos - Foco no TarefasController GET/POST)

1. **Entidades (Entities/):**
   - Crie as classes C# Usuario.cs, Tarefa.cs, DetalhesTarefa.cs, Tag.cs, TarefaTag.cs espelhando as tabelas do banco. Adicione as propriedades de navegação (ICollection<>, referência simples). Deixe as classes "limpas" (sem Data Annotations de mapeamento).
2. **DbContext (Infra/Context/TarefaContext.cs):**
   - Crie o TarefaContext herdando de DbContext.
   - Adicione os DbSet<> para todas as entidades.
   - Implemente o OnModelCreating usando **Fluent API** para:
     - Mapear cada entidade para sua respectiva tabela (ToTable).
     - Mapear cada propriedade para sua coluna (Property().HasColumnName()). Configure IsRequired(), HasMaxLength(), IsUnique() conforme o script SQL.
     - Configurar a **Chave Primária Composta** para TarefaTag.
     - Configurar explicitamente os relacionamentos **1:1** (Tarefa <-> DetalhesTarefa), **1:N** (Usuario -> Tarefa) e os **dois 1:N** que formam o N:N (Tarefa -> TarefaTag e Tag -> TarefaTag). Use HasOne, WithOne, HasMany, WithMany, HasForeignKey, OnDelete.
3. **DTOs (Infra/DTOs/):**
   - Crie TarefaDto.cs (para retorno) e CriarTarefaDto.cs (para entrada do POST). Adicione Data Annotations de validação ([Required], [StringLength]) no CriarTarefaDto.
4. **AutoMapper (Infra/Mapping/MappingProfile.cs):**
   - Crie o MappingProfile herdando de Profile.
   - Configure os mapeamentos: CreateMap<Tarefa, TarefaDto>(); e CreateMap<CriarTarefaDto, Tarefa>();.
5. **Repositórios (Infra/Repositories/):**
   - Crie a interface ITarefaRepository.cs na subpasta Interfaces.
   - Crie a implementação TarefaDBRepository.cs que recebe TarefaContext via DI. Implemente os métodos ObterTodos() e ObterPorId(int id) (usando .Include() se necessário para carregar o Usuario ou DetalhesTarefa) e Adicionar(Tarefa novaTarefa).
6. **Serviços (Services/):**
   - Crie a interface ITarefaService.cs na subpasta Interfaces.
   - Crie a implementação TarefaService.cs que recebe ITarefaRepository e IMapper via DI. Implemente os métodos ObterTodos() (mapeia para List<TarefaDto>), ObterPorId(int id) (mapeia para TarefaDto?) e Adicionar(CriarTarefaDto tarefaDto) (mapeia DTO para Entidade, define DataCriacao e Concluida = false, chama repositório, retorna a entidade criada).
7. **Injeção de Dependência (Program.cs):**
   - Registre o DbContext, o ITarefaRepository/TarefaDBRepository, o ITarefaService/TarefaService e o AutoMapper.
8. **Controller (Controllers/TarefasController.cs):**
   - Crie o TarefasController herdando de ControllerBase com [ApiController] e [Route].
   - Injete ITarefaService.
   - Implemente os endpoints **GET /api/tarefas** e **GET /api/tarefas/{id}** retornando ActionResult<List<TarefaDto>> e ActionResult<TarefaDto>.

- Implemente o endpoint **POST /api/tarefas** recebendo [FromBody] CriarTarefaDto e retornando ActionResult<TarefaDto> com status 201 Created (use CreatedAtAction).

9. **Commit do Mínimo Viável:**
   - Após testar os endpoints GET e POST via Swagger, faça um commit:

```
git add .
git commit -m "feat: Implementa GET e POST para TarefasController com DI, EF Core e AutoMapper"
```

Etapa 4: Entrega Completa (Até a Próxima Semana)

(Peso: 20 Pontos)

1. **CRUD Completo para TarefasController:**
   - Implemente PUT /api/tarefas/{id} (recebe AtualizarTarefaDto).
   - Implemente DELETE /api/tarefas/{id}.
   - Implemente um endpoint PATCH /api/tarefas/{id}/concluir (ou similar) para marcar uma tarefa como concluída.
2. **Implementação Completa para UsuariosController:**
   - Crie DTOs (UsuarioDto, CriarUsuarioDto).
   - Configure o mapeamento no MappingProfile.
   - Crie IUsuarioRepository e UsuarioDBRepository.
   - Crie IUsuarioService e UsuarioService.
   - Registre no Program.cs.
   - Implemente o UsuariosController com CRUD completo (GET, POST, PUT, DELETE). Considere carregar as tarefas do usuário (.Include()) no GET por ID.
3. **Implementação Completa para TagsController:**
   - Crie DTOs (TagDto, CriarTagDto).
   - Configure o mapeamento no MappingProfile.
   - Crie ITagRepository e TagDBRepository.
   - Crie ITagService e TagService.
   - Registre no Program.cs.
   - Implemente o TagsController com CRUD completo (GET, POST, PUT, DELETE).
   - **Desafio:** Implemente um endpoint em TarefasController para associar uma Tag existente a uma Tarefa (ex: POST /api/tarefas/{tarefaId}/tags/{tagId}).
4. **Finalizar e Mesclar no Git:**
   - Faça commits incrementais conforme avança.
   - Quando terminar, mescle a branch develop na main:

```
git checkout main
git merge develop
git push origin main #(Se estiver usando um repositório remoto)
git checkout develop # Voltar para a branch de desenvolvimento
```

Critérios de Avaliação:

- Estrutura do projeto correta.
- Uso adequado de DI, Interfaces, Repositories, Services.
- Configuração correta do EF Core (DbContext, Fluent API).
- Implementação correta dos relacionamentos (1:1, 1:N, N:N).
- Uso correto de DTOs e AutoMapper.
- Implementação funcional dos endpoints CRUD.
- Código limpo, organizado e seguindo as convenções C#.
- Versionamento com Git.

Boa sorte com a avaliação!

Este projeto prático consolida tudo o que foi aprendido no Módulo 2.