



UNIVERSIDADE DE  
**vassouras**

# Laboratório de Programação Full Stack

Universidade de Vassouras

# **Aula Completa: Aplicação .NET com EF Core InMemory e Arquitetura em Camadas**

## **Objetivo da Aula**

Construir uma aplicação .NET estruturada em camadas (Domínio, Repositório, Serviço e API), utilizando o **Entity Framework Core InMemory**. Vamos desde a criação do repositório Git até a execução completa da aplicação.



## Passo 1 - Criação da Solução e Estrutura de Camadas

💡 **Observação:** A clonagem do repositório Git e a configuração de versionamento serão abordadas no final.

```
# Criação do diretório do projeto  
mkdir MinhaApp  
cd MinhaApp
```

Essa explicação orienta desde a criação da solução até o versionamento no Git. **O Git será tratado no fim** para evitar confusões.

## ✓ Padrão de Criação de Classes e Interfaces

A partir de agora, todas as classes e interfaces no projeto seguirão este roteiro para garantir consistência e organização:

### ● Passo 1: Criar Classes e Interfaces no Visual Studio

1. Clique com o botão direito no projeto onde deseja criar a classe ou interface.
2. Selecione **Adicionar** → **Novo Item**.
3. Escolha a opção **Class** para criar uma classe ou **Interface** para uma interface.
4. Defina um nome claro e objetivo, seguindo o padrão:
  - Classes: Nome no singular, começando com letra maiúscula. Exemplo: `Aluno`, `AlunoRepositorio`, `AlunoServico`.
  - Interfaces: Prefixo "I" seguido de nome descritivo. Exemplo: `IALunoRepositorio`, `IALunoServico`.
5. Clique em **Adicionar** para criar o arquivo.

## ✓ Padrão de Criação de Classes e Interfaces

A partir de agora, todas as classes e interfaces no projeto seguirão este roteiro para garantir consistência e organização:

### ● Passo 2: Criar Classes e Interfaces via CLI (Opcional)

Para quem prefere linha de comando, siga o roteiro:

1. Navegue até o diretório do projeto:

```
cd MinhaApp.Dominio
```

2. Crie a classe ou interface usando o comando:

```
dotnet new class -n Aluno  
dotnet new interface -n IAlunoRepositorio
```

Seguindo esse padrão, todas as classes e interfaces terão uma estrutura padronizada e alinhada à arquitetura em camadas. Sempre mantenha a coesão e acoplamento baixo para facilitar a manutenção e evolução do projeto.

## 2. Criação da Solução e das Camadas do Projeto

Vamos criar uma solução em branco e estruturar o projeto:

### Via CLI (VS Code/Terminal)

```
# Criação da solução
dotnet new sln -n MinhaApp

# Criação das camadas
dotnet new classlib -n MinhaApp.Dominio
dotnet new classlib -n MinhaApp.Repositorio
dotnet new classlib -n MinhaApp.Servico
dotnet new webapi -n MinhaApp.WebAPI

# Adição dos projetos na solução
dotnet sln MinhaApp.sln add MinhaApp.Dominio
dotnet sln MinhaApp.sln add MinhaApp.Repositorio
dotnet sln MinhaApp.sln add MinhaApp.Servico
dotnet sln MinhaApp.sln add MinhaApp.WebAPI

# Referências entre projetos
dotnet add MinhaApp.Repositorio reference MinhaApp.Dominio
dotnet add MinhaApp.Servico reference MinhaApp.Repositorio
dotnet add MinhaApp.WebAPI reference MinhaApp.Servico
|
```





## Via Visual Studio

1. Abra o Visual Studio e clique em **Arquivo > Novo > Projeto**.
2. Selecione **Solução em Branco (.NET Core)** e nomeie como `MinhaApp`.
3. Clique com o botão direito na solução, selecione **Adicionar > Novo Projeto**.
  - Crie um **Biblioteca de Classes (.NET Core)** para `MinhaApp.Dominio`, `MinhaApp.Repositorio` e `MinhaApp.Servico`.
  - Crie um **Aplicativo Web API (.NET Core)** para `MinhaApp.WebAPI`.
4. Adicione referências entre projetos conforme necessário.

Precisamos conectar as camadas da forma correta, garantindo o fluxo de dados:

1. No Visual Studio, clique com o botão direito em cada projeto e vá em **Adicionar** → **Referência...**
2. Adicione as referências conforme o diagrama:


- `MinhaApp.Repositorio` → `MinhaApp.Dominio`
- `MinhaApp.Servico` → `MinhaApp.Repositorio`
- `MinhaApp.WebAPI` → `MinhaApp.Servico`

Estrutura final da solução:

**MinhaApp**

- ├─ `MinhaApp.Dominio`
- ├─ `MinhaApp.Repositorio`
- ├─ `MinhaApp.Servico`
- └─ `MinhaApp.WebAPI`

### 3. Instalação do Entity Framework Core InMemory

 O Entity Framework Core InMemory será instalado em `MinhaApp.Repositorio` e `MinhaApp.WebAPI` para acessar o banco de dados e configurar a Injeção de Dependência.

#### Via CLI (VS Code/Terminal)

```
# Instalando o EF Core InMemory no Repositório
```

```
dotnet add MinhaApp.Repositorio package Microsoft.EntityFrameworkCore
```

```
dotnet add MinhaApp.Repositorio package Microsoft.EntityFrameworkCore.InMemory
```

```
# Instalando o EF Core InMemory no WebAPI
```

```
dotnet add MinhaApp.WebAPI package Microsoft.EntityFrameworkCore
```

```
dotnet add MinhaApp.WebAPI package Microsoft.EntityFrameworkCore.InMemory
```



## Via Visual Studio

1. Clique com o botão direito no projeto **MinhaApp.Repositorio**.
2. Vá em **Gerenciador de Pacotes NuGet > Procurar**.
3. Instale `Microsoft.EntityFrameworkCore` e `Microsoft.EntityFrameworkCore.InMemory`.
4. Repita os passos acima para **MinhaApp.WebAPI**.

## 4. Camada de Domínio - Entidade Aluno

A camada de domínio representa as regras de negócio e as entidades fundamentais.

### Modelo Aluno (MinhaApp.Dominio)

```
namespace MinhaApp.Dominio
{
    public class Aluno
    {
        public int Id { get; set; }
        public string Nome { get; private set; }
        public decimal Nota { get; private set; }

        public Aluno(string nome, decimal nota)
        {
            Nome = nome;
            Nota = nota;
        }

        public bool EstaAprovado() => Nota >= 7;
    }
}
```

## 5. Camada de Repositório - Acesso a Dados

O repositório faz a comunicação com o banco de dados usando o EF Core InMemory.

### Interface do Repositório - IAlunoRepositorio (MinhaApp.Repositorio)

```
using MinhaApp.Dominio;

namespace MinhaApp.Repositorio
{
    public interface IAlunoRepositorio
    {
        void Salvar(Aluno aluno);
        IEnumerable<Aluno> Listar();
    }
}
```



## DbContext - AppDbContext (MinhaApp.Repositorio)

```
using Microsoft.EntityFrameworkCore;
using MinhaApp.Dominio;

namespace MinhaApp.Repositorio
{
    public class AppDbContext : DbContext
    {
        public DbSet<Aluno> Alunos { get; set; }

        public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }
    }
}
```



## Repositório de Aluno - AlunoRepositorio (MinhaApp.Repositorio)

```
using MinhaApp.Dominio;

namespace MinhaApp.Repositorio
{
    public class AlunoRepositorio : IAlunoRepositorio
    {
        private readonly AppDbContext _context;

        public AlunoRepositorio(AppDbContext context)
        {
            _context = context;
        }

        public void Salvar(Aluno aluno)
        {
            _context.Alunos.Add(aluno);
            _context.SaveChanges();
        }

        public IEnumerable<Aluno> Listar()
        {
            return _context.Alunos.ToList();
        }
    }
}
```



## 6. Camada de Serviço - Regras de Negócio e DTO

A camada de serviço aplica regras de negócio e usa o DTO (Data Transfer Object).

### Criação do DTO - AlunoDto (MinhaApp.Servico)

```
namespace MinhaApp.Servico
{
    public class AlunoDto
    {
        public string Nome { get; set; }
        public decimal Nota { get; set; }
    }
}
```



## Interface de Serviço - IAlunoServico (MinhaApp.Servico)

```
namespace MinhaApp.Servico
{
    public interface IAlunoServico
    {
        void Adicionar(AlunoDto alunoDto);
        IEnumerable<AlunoDto> Listar();
    }
}
```



# Implementação do Serviço - AlunoServico (MinhaApp.Servico)

```
using MinhaApp.Dominio;
using MinhaApp.Repositorio;

namespace MinhaApp.Servico
{
    public class AlunoServico : IAlunoServico
    {
        private readonly IAlunoRepositorio _repositorio;

        public AlunoServico(IAlunoRepositorio repositorio)
        {
            _repositorio = repositorio;
        }

        public void Adicionar(AlunoDto alunoDto)
        {
            var aluno = new Aluno(alunoDto.Nome, alunoDto.Nota);
            _repositorio.Salvar(aluno);
        }


        public IEnumerable<AlunoDto> Listar()
        {
            return _repositorio.Listar().Select(aluno => new AlunoDto
            {
                Nome = aluno.Nome,
                Nota = aluno.Nota
            });
        }
    }
}
```

## 7. Camada de Apresentação - Controller da API

Controlador responsável por expor a API.

### **AlunoController - MinhaApp.WebAPI**

#### Passo a Passo no Visual Studio:

1. Clique com o botão direito em **Controllers** dentro do projeto **MinhaApp.WebAPI**.
  2. Selecione **Adicionar** → **Controller**.
  3. Escolha **API Controller - Empty**.
  4. Nomeie o controlador como **AlunoController** e clique em **Adicionar**.
-  O controlador será criado vazio, pronto para ser preenchido manualmente com as dependências injetadas e os métodos necessários.

## 7. Camada de Apresentação - Controller da API

Controlador responsável por expor a API.

### **AlunoController - MinhaApp.WebAPI**

```
using Microsoft.AspNetCore.Mvc;
using MinhaApp.Servico;

namespace MinhaApp.WebAPI.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class AlunoController : ControllerBase
    {
        private readonly IAlunoServico _servico;

        public AlunoController(IAlunoServico servico)
        {
            _servico = servico;
        }

        [HttpPost]
        public IActionResult Adicionar(AlunoDto dto)
        {
            _servico.Adicionar(dto);
            return Ok("Aluno adicionado com sucesso");
        }

        [HttpGet]
        public IActionResult Listar()
        {
            return Ok(_servico.Listar());
        }
    }
}
```

## Program.cs (MinhaApp.WebAPI)

```
using MinhaApp.Repositorio;
using MinhaApp.Servico;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<AppDbContext>(opt => opt.UseInMemoryDatabase("AlunosDB"));
builder.Services.AddScoped<IALunoRepositorio, AlunoRepositorio>();
builder.Services.AddScoped<IALunoServico, AlunoServico>();
builder.Services.AddControllers();

var app = builder.Build();

app.UseRouting();
app.UseEndpoints(endpoints => endpoints.MapControllers());
app.Run();
```

✓ Pronto! Aplicação em camadas completa e funcional!

## ✓ Passo Final - Versionamento com Git

Com a solução completa e funcionando, vamos configurar o Git e versionar o código.

### ● 1. Criando o Repositório no GitHub

1. Acesse <https://github.com> e faça login.
2. Clique em **New Repository** e preencha as informações:
  - **Repository name:** MinhaApp
  - **Description:** Projeto em camadas com EF Core InMemory
  - **Public/Private:** Escolha conforme a necessidade
  - **Add a .gitignore template:** Selecione **Visual Studio**
  - Clique em **Create Repository**



## ● 2. Configurando o Git no Projeto Local

Após configurar a solução no Visual Studio ou pela CLI:

1. No diretório raiz da solução ( `MinhaApp` ), execute no terminal:

```
# Inicializando o repositório Git local
git init

# Adicionando o repositório remoto criado no GitHub
git remote add origin https://github.com/SEU_USUARIO/MinhaApp.git

# Adicionando os arquivos e criando o commit inicial
git add .
git commit -m "feat: Criação da solução em camadas com EF Core InMemory"

# Enviando para o repositório remoto
git branch -M main
git push -u origin main
```

# Laboratório de Programação Full Stack

Universidade de Vassouras



UNIVERSIDADE DE  
**vassouras**