Projeto Fiap.Challenge.Wtc - Arquitetura Hexagonal

Visão Geral

Este projeto foi desenvolvido seguindo os princípios da **Arquitetura Hexagonal** (também conhecida como Ports and Adapters), promovendo alta testabilidade, baixo acoplamento e separação clara de responsabilidades.

Estrutura do Projeto

```
Fiap.Challenge.Wtc/
├── src/
│   ├── Fiap.Challenge.Wtc.API/          # Camada de Apresentação (Web
API)
│   ├── Fiap.Challenge.Wtc.Application/   # Camada de Aplicação (Use
Cases)
│   ├── Fiap.Challenge.Wtc.Domain/        # Camada de Domínio (Regras de
Negócio)
│   └── Fiap.Challenge.Wtc.Infrastructure/ # Camada de Infraestrutura
(Implementações)
├── tests/
│   ├── Fiap.Challenge.Wtc.Tests.Unit/        # Testes Unitários
│   └── Fiap.Challenge.Wtc.Tests.Integration/ # Testes de Integração
└── docs/                                     # Documentação
```

Camadas da Arquitetura

1. Domain (Domínio) 🏛️

Localização: `src/Fiap.Challenge.Wtc.Domain/`

O núcleo da aplicação, contendo as regras de negócio puras e independentes de tecnologia.

Responsabilidades:

- Definir entidades de domínio
- Implementar value objects
- Definir interfaces de repositório (ports)
- Definir serviços de domínio
- Definir exceções de domínio

Estrutura:

```
Domain/
├── Entities/          # Entidades de domínio
├── ValueObjects/      # Value Objects
├── Repositories/      # Interfaces de repositório (ports)
```

```
├── Services/          # Serviços de domínio
└── Exceptions/        # Exceções específicas do domínio
```

2. Application (Aplicação) ⚙️

Localização: `src/Fiap.Challenge.Wtc.Application/`

Orquestra os use cases da aplicação e define os contratos para serviços externos.

Responsabilidades:

- Implementar use cases (casos de uso)
- Definir DTOs para transferência de dados
- Definir interfaces para serviços externos
- Coordenar transações e validações

Estrutura:

```
Application/
├── UseCases/          # Implementação dos casos de uso
├── DTOs/              # Data Transfer Objects
├── Interfaces/        # Interfaces para serviços externos
└── Common/            # Classes comuns (Result, etc.)
```

3. Infrastructure (Infraestrutura) 🔧

Localização: `src/Fiap.Challenge.Wtc.Infrastructure/`

Implementa os adapters para tecnologias externas e persistência de dados.

Responsabilidades:

- Implementar repositórios
- Configurar acesso a dados
- Implementar serviços externos
- Configurar injeção de dependência

Estrutura:

```
Infrastructure/
├── Data/              # Configurações de banco de dados
├── Repositories/      # Implementações dos repositórios
├── Services/          # Implementações de serviços externos
└── Configuration/     # Configurações e extensões DI
```

4. API (Apresentação) 🌐

Localização: `src/Fiap.Challenge.Wtc.API/`

Camada de apresentação que expõe os endpoints da aplicação.

Responsabilidades:

- Definir controllers e endpoints
- Gerenciar autenticação e autorização
- Tratar exceções globalmente
- Configurar middlewares

Estrutura:

```
API/
├── Controllers/     # Controllers da Web API
├── Middleware/      # Middlewares customizados
└── Configuration/   # Configurações da API
```

Princípios da Arquitetura Hexagonal

1. **Inversão de Dependência**

- O domínio define interfaces (ports)
- A infraestrutura implementa essas interfaces (adapters)
- A aplicação depende apenas de abstrações

2. **Separação de Responsabilidades**

- **Domain**: Regras de negócio puras
- **Application**: Orquestração e casos de uso
- **Infrastructure**: Detalhes de implementação
- **API**: Interface com o mundo exterior

3. **Testabilidade**

- Domínio e aplicação são facilmente testáveis
- Dependências externas podem ser facilmente "mockadas"
- Testes unitários e de integração separados

Fluxo de Dados

```
HTTP Request → API Controller → Use Case → Domain Service → Repository
Interface
                                                                    ↓
HTTP Response ← API Controller ← Use Case ← Domain Service ← Repository
Implementation
```

Vantagens desta Arquitetura

1. **Alta Testabilidade**: Domínio e aplicação independentes de infraestrutura
2. **Flexibilidade**: Fácil troca de tecnologias de persistência ou comunicação
3. **Manutenibilidade**: Separação clara de responsabilidades
4. **Evolução**: Permite crescimento sustentável da aplicação
5. **Reutilização**: Lógica de negócio reutilizável em diferentes contextos

Comandos Úteis

Compilar a solução:

```
dotnet build
```

Executar testes:

```
dotnet test
```

Executar a API:

```
dotnet run --project src/Fiap.Challenge.Wtc.API
```

Executar testes unitários:

```
dotnet test tests/Fiap.Challenge.Wtc.Tests.Unit
```

Executar testes de integração:

```
dotnet test tests/Fiap.Challenge.Wtc.Tests.Integration
```

Próximos Passos

1. Implementar entidades específicas do domínio
2. Criar use cases específicos da aplicação
3. Implementar repositórios com Entity Framework ou tecnologia escolhida
4. Adicionar autenticação e autorização
5. Configurar logging e monitoramento
6. Implementar validações mais robustas
7. Adicionar documentação da API com Swagger

Esta documentação deve ser atualizada conforme o projeto evolui e novos recursos são implementados.