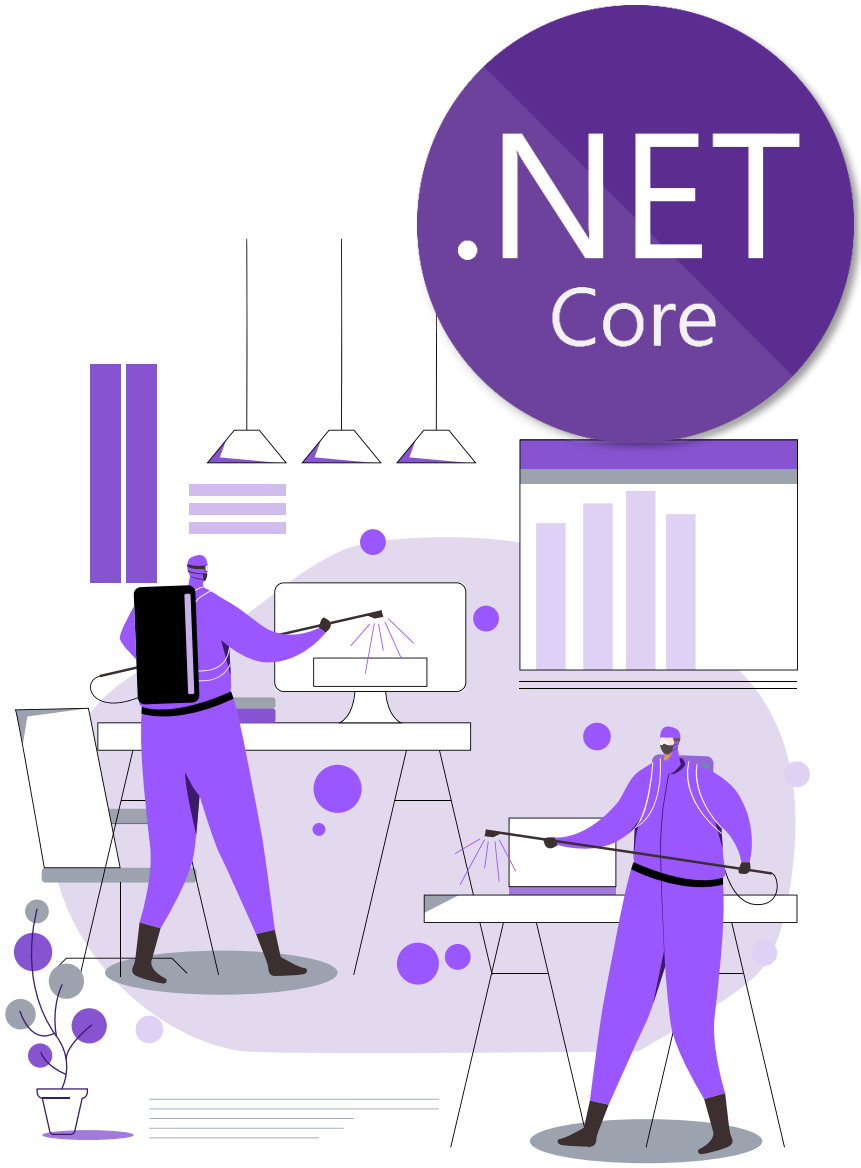


PeopleHub

Clear Architecture

PoC

Versão: 0

Sumário

Versão

Data	Rev.	Descrição	Responsável
09/02/2025	0	Versão inicial.	Flávio dos Santos

Objetivo

Este documento tem como objetivo apresentar a arquitetura da PoC PeopleHub, baseada nos princípios da Clean Architecture e Domain-Driven Design (DDD). A proposta visa garantir uma estrutura modular, escalável e testável, promovendo a separação de responsabilidades e facilitando a manutenção e evolução do sistema.

A PoC define diretrizes para implementação, incluindo a organização das camadas Presentation, Application, Domain, Infrastructure e Config, destacando suas respectivas responsabilidades. Além disso, estabelece padrões de desenvolvimento, como injeção de dependências, versionamento de API, autenticação JWT, persistência com Entity Framework Core e PostgreSQL, garantindo flexibilidade e independência de tecnologias externas.

O documento também serve como referência para o time de desenvolvimento, abordando boas práticas, nomenclaturas, princípios SOLID e gerenciamento de transações, assegurando um código limpo e alinhado às melhores práticas do mercado.

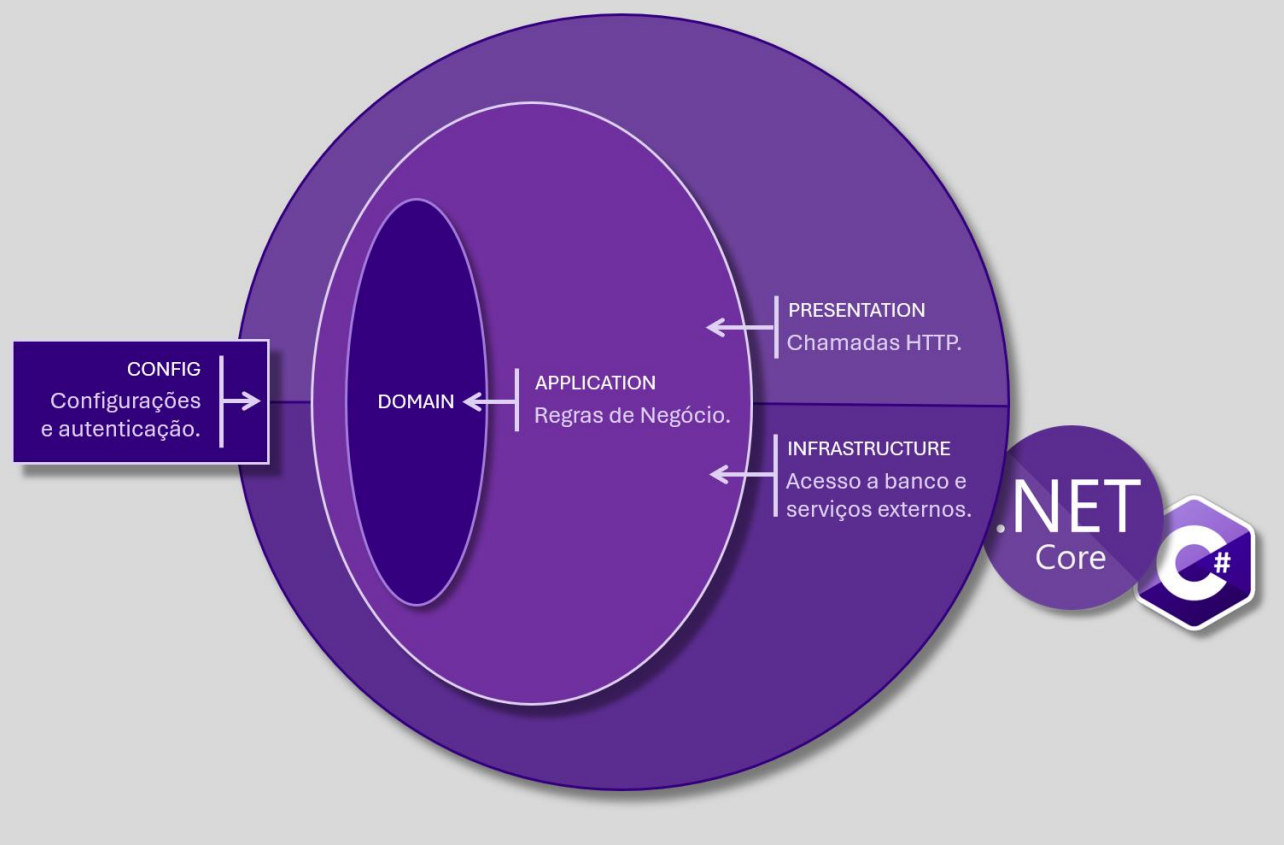
Diagrama de Arquitetura

Definição

A **Clean Architecture**, proposta por Robert C. Martin (Uncle Bob), é um modelo arquitetural amplamente utilizado que organiza o código em camadas bem definidas e modulares, garantindo uma separação clara de responsabilidades, maior testabilidade, flexibilidade para mudanças e fácil manutenção ao longo do tempo.

Ela segue o princípio de dependências direcionais, onde as camadas externas (como infraestrutura e apresentação) dependem da camada de aplicação, e a camada aplicação depende exclusivamente da camada de domínio. Isso garante que as regras de negócio sejam isoladas e independentes de detalhes de implementação, como frameworks, bancos de dados ou interfaces externas, tornando o sistema mais flexível, testável e sustentável no longo prazo.

Além disso, a Clean Architecture incentiva o uso abstrações bem definidas para desacoplar os módulos do sistema, facilitando a substituição de implementações sem impactar a lógica central. A injeção de dependências reforça essa flexibilidade, permitindo a troca de tecnologias e frameworks sem afetar a camada de domínio, garantindo uma arquitetura sustentável, escalável e bem estruturada.



Presentation Layer

A ***Presentation Layer*** é responsável por expor a API REST do sistema, tratando as requisições HTTP recebidas dos clientes e direcionando-as corretamente para a ***Application Layer***, garantindo a validação e roteamento adequado das chamadas.

Ela segue os princípios de responsabilidades bem definidas, garantindo que a Presentation Layer da API atue exclusivamente como um ponto de entrada para as requisições, sem conter regras de negócios, delegando esse processamento para as camadas apropriadas.

Principais Responsabilidades:

- Definir endpoints HTTP para CRUD de Pessoas Físicas e Jurídicas.
- Aplicar autenticação via JWT.
- Tratar exceções globais e formatar respostas padronizadas.

Infrastructure Layer

A **Infrastructure Layer** da PeopleHub é responsável por implementar os detalhes da aplicação, fornecendo suporte para acesso a dados, serviços externos, autenticação, auditoria e integração com o ambiente operacional. Ela funciona como a ponte entre a **Application Layer** e os frameworks/bibliotecas externas, garantindo que a lógica de negócios permaneça desacoplada de detalhes de implementação.

Principais Responsabilidades

Acesso a Dados (Persistence)

Implementa repositórios (Repositories) que interagem diretamente com o banco de dados, utilizando o Entity Framework Core para comunicação com o PostgreSQL.

Exemplo na PoC:

- PersonRepository e UserAccountRepository implementam a interface de repositório do Domain Layer (IPersonRepository, IUserAccountRepository).
- PeopleHubDbContext define a estrutura das tabelas, mapeando entidades como IndividualPersonEntity, LegalPersonEntity, UserAccountEntity e AuditLogEntity.

Gerenciamento de Transações (Unit of Work)

UnitOfWork (IUnitOfWork) encapsula operações de banco de dados, garantindo a execução atômica de múltiplas operações dentro de uma mesma transação.

Exemplo: A classe RegisterIndividualPersonUseCase na Application Layer chama _unitOfWork.CommitAsync() após inserir um novo usuário, garantindo que os dados sejam persistidos corretamente.

Mapeamento de Entidades (ORM)

Utiliza Fluent API no Entity Framework Core para definir configurações de tabelas no banco de dados, garantindo consistência no armazenamento.

Autenticação e Identidade

AuthenticatedUserService fornece métodos para obter informações do usuário autenticado via IHttpContextAccessor.

JwtBearer é configurado em InfrastructureModule para validar tokens JWT.

Registro de Auditoria (Logging)

AuditLogRepository grava logs de ações do usuário na tabela audit_logs, garantindo rastreabilidade. Chamado indiretamente pela Application Layer via RegisterLogAsync(), como visto no AuthenticateUserAccountUseCase.

Configuração e Injeção de Dependências

InfrastructureModule e DatabaseConfiguration configuram:

- Registro de serviços no DI Container.
- Conexão com o banco de dados (UseNpgsql).
- Injeção de repositórios e UnitOfWork.

Application Layer

A Application Layer da PoC PeopleHub desempenha um papel fundamental na orquestração dos casos de uso, garantindo que a lógica de aplicação seja bem estruturada e separada das regras de negócio. Essa camada é responsável por coordenar a interação entre a apresentação (Controllers) e o domínio, garantindo correta execução das operações sem expor detalhes da implementação interna.

Principais Responsabilidades

A Application Layer na PoC PeopleHub foi projetada para seguir os princípios da **Clean Architecture**, garantindo separação de responsabilidades, modularidade e testabilidade.

Através da orquestração dos casos de uso, gerenciamento transacional, validações, auditoria e autenticação, essa camada assegura que a aplicação funcione de forma consistente e escalável, mantendo o domínio isolado e protegido de detalhes técnicos externos.

Orquestração dos Casos de Uso

- A PoC possui diversas classes de UseCases que atuam na intermediação entre as requisições da API e a lógica do negócio.
- Cada UseCase encapsula um fluxo específico, garantindo clareza e responsabilidade única para cada operação.

Intermediação entre a Apresentação e Domínio

As Controllers da Presentation Layer chamam diretamente os serviços da Application Layer como PersonService e UserAccountService, que por sua vez executam as operações por meio de UseCases.

Isso evita que regras de negócio sejam implementadas diretamente na API, seguindo os princípios da Clean Architecture.

Gerenciamento de Transações com Unit of Work

O Unit of Work, representado pela interface IUnitOfWork, garante que todas as operações sejam executadas de forma atômica, evitando inconsistências nos dados.

Exemplo: No RegisterIndividualPersonUseCase, após adicionar uma nova pessoa física, a transação é finalizada com _unitOfWork.CommitAsync();.

Autenticação e Autorização

A autenticação é realizada no AuthenticateUserAccountUseCase, onde um JWT Token é gerado e assinado com a chave de segurança configurada na JwtConfiguration.

Isso garante que apenas usuários autenticados possam acessar os endpoints protegidos.

Independência de Infraestrutura

A camada de aplicação interage com repositórios IPersonRepository, IUserAccountRepository) sem depender diretamente da implementação do banco de dados, promovendo desacoplamento e permitindo mudanças futuras sem impactos na regra de negócio.

Domain Layer

A Domain Layer é o núcleo da aplicação e contém a lógica de negócio essencial, independente de frameworks, bancos de dados ou tecnologia de armazenamento. Seguindo os princípios da

Clean Architecture e Domain-Driven Design (DDD), essa camada encapsula entidades, value objects, regras de negócio e enums.

Principais Responsabilidades

Definir as entidades principais do domínio

Representa os conceitos centrais da aplicação, como Pessoa Física (IndividualPersonEntity) e Pessoa Jurídica (LegalPersonEntity).

Modelar os Value Objects

Como CPF, CNPJ, Telefone, E-mail e Endereço, garantindo imutabilidade e consistência nos dados.

Encapsular regras de negócio

A lógica de validação, atualização e consistência dos dados é mantida nesta camada, isolada das demais.

Garantir independência de tecnologias

O domínio não depende de banco de dados, APIs externas ou frameworks, mantendo-se puro e testável.

Config Layer

A Config Layer da PoC PeopleHub é responsável por centralizar e gerenciar as configurações essenciais da aplicação, garantindo que os serviços críticos, como autenticação e injeção de dependências, sejam configurados corretamente. Essa camada segue os princípios da Clean Architecture, isolando configurações e facilitando a manutenção e escalabilidade do sistema.

Principais Responsabilidades

A Config Layer não faz parte diretamente do núcleo da Clean Architecture, mas atua como um suporte essencial para a aplicação. Seguindo os princípios de separação de responsabilidades e independência de frameworks, essa camada permite que a infraestrutura e a aplicação sejam configuradas dinamicamente sem comprometer a lógica de negócio.

- Isolamento – As configurações ficam centralizadas e separadas do código-fonte principal.
- Facilidade de manutenção – Alterações na configuração não afetam diretamente as outras camadas.
- Segurança – Evita exposição de segredos e facilita o gerenciamento de autenticação.

Gerenciamento de Configurações

Centraliza as configurações da aplicação, como autenticação JWT, conexões com banco de dados e integração com outras camadas.

Configuração de Autenticação JWT

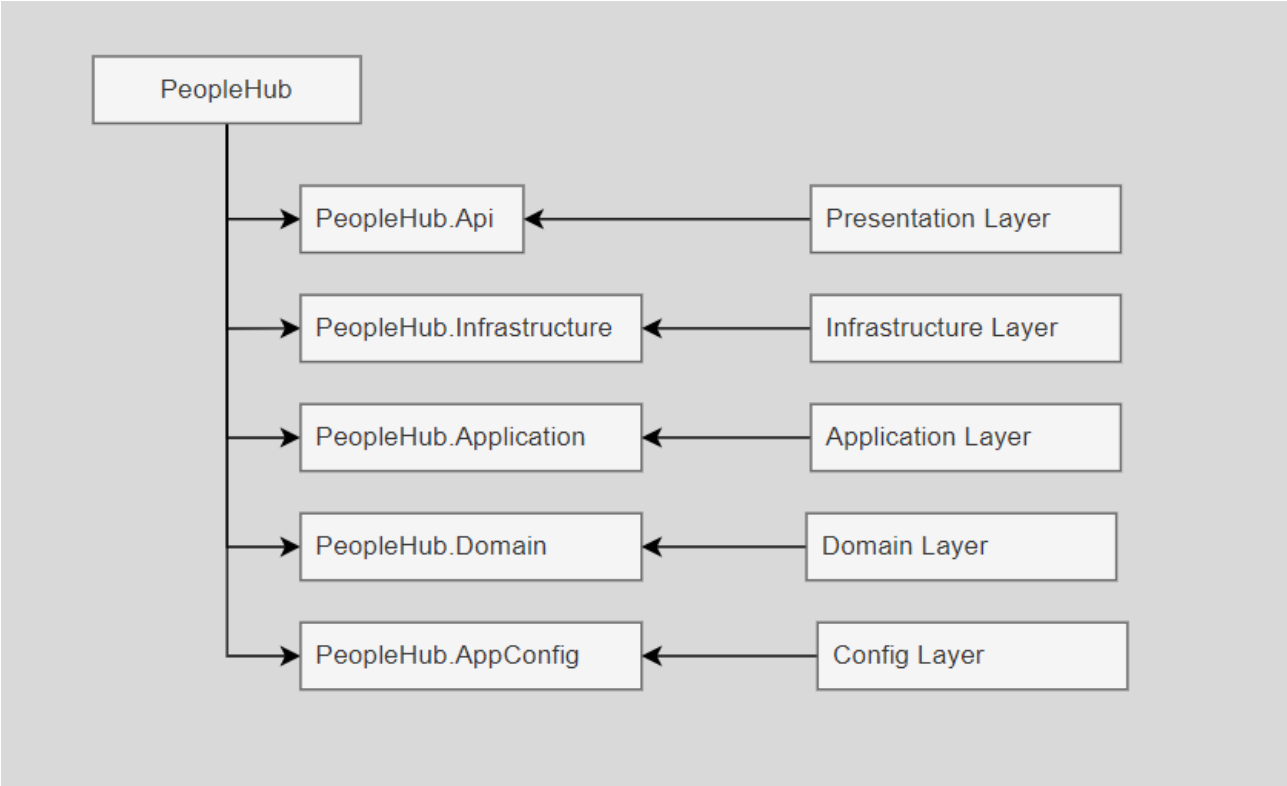
Define os parâmetros necessários para a autenticação baseada em JSON Web Token (JWT), como emissor, audiência e chave secreta.

Implementa a validação dos tokens, garantindo a segurança e integridade das requisições na API.

Integração entre Camadas

Responsável por registrar e configurar os serviços da Application Layer e Infrastructure Layer no Dependency Injection Container, garantindo que todas as dependências estejam corretamente resolvidas.

Estrutura do Projeto



Diretrizes de Implementação

Estrutura e Organização do Código

O projeto segue Clean Architecture e DDD (Domain-Driven Design), garantindo modularidade e separação de responsabilidades.

Todo o código deve estar organizado dentro das camadas apropriadas: Presentation, Application, Domain, Infrastructure e Config.

Use Cases são implementados na camada de Application, garantindo que a lógica de aplicação esteja separada da interface e da infraestrutura.

Interfaces (Contracts) devem ser definidas na Application Layer, garantindo que a Infrastructure Layer as implemente corretamente sem dependências inversas.

Padrões de Nomenclatura

As classes devem seguir convenções coerentes:

- UseCases: Verbo + Entidade + UseCase (Exemplo: RegisterUserAccountUseCase)
- Repositories: Entidade + Repository (Exemplo: UserAccountRepository)
- DTOs: Ação + Entidade + Dto (Exemplo: RegisterUserAccountDto)
- Métodos dentro dos Use Cases devem seguir a estrutura:
- ExecuteAsync: Método padrão para executar as operações.

Princípios de Código Limpo

Aplicação de SOLID, especialmente:

- S - Single Responsibility Principle: Cada classe deve ter um único propósito.
- D - Dependency Inversion Principle: As dependências devem ser injetadas via Interfaces, evitando acoplamento forte.
- Aplicação de Fail Fast: Validações devem ocorrer o mais cedo possível para evitar execução de lógica desnecessária.
- Uso de Exceções Específicas: Preferir ArgumentException ou exceções customizadas ao invés de exceções genéricas.
- Uso de DTOs para comunicação entre camadas, garantindo que a Domain Layer permaneça isolada.

Tratamento de Erros e Logging

- Todas as exceções devem ser registradas usando um serviço de logging (AuditLogService).
- Validações devem retornar respostas padronizadas (ApiResponseDto<T>).
- A Presentation Layer não deve conter lógica de negócios, apenas validações mínimas para garantir integridade dos dados.

Tecnologias Utilizadas

Tecnologia	Descrição
.NET 8 (ASP.NET Core)	Framework principal para desenvolvimento da API RESTful, garantindo alto desempenho e suporte de longo prazo.
Asp.Versioning.Mvc	Package utilizado para implementar versionamento de APIs .NET Core
Npgsql	O package Npgsql.EntityFrameworkCore.PostgreSQL é o provedor para o PostgreSQL.
BCrypt.Net-Next	Biblioteca para hashing de senhas usando o algoritmo BCrypt.