**DDD e Clean Architecture no .NET**

## O que é DDD?

**DDD (Domain-Driven Design)** é uma forma de desenvolver sistemas focando no **domínio do negócio**, ou seja, nas regras reais que o sistema precisa resolver.

### Princípios principais do DDD:

* **Modelagem de Domínio:** Cria entidades, Value Objects e agregados que representam fielmente o negócio.
* **Ubiquitous Language:** Linguagem comum entre devs e o negócio.
* **Isolamento do Domínio:** Regras de negócio separadas de infraestrutura.
* **Repositórios:** Para abstrair acesso a dados.

Na prática, você cria classes como Receita, Usuario e Ingrediente, aplicando as regras de negócio dentro dessas entidades.

## O que é Clean Architecture?

**Clean Architecture** organiza a estrutura do projeto para que: - O **Domínio fique no centro** e não dependa de nada. - Infraestrutura e APIs dependam do domínio, nunca o contrário. - Facilite testes, manutenção e evolução.

* **Estrutura em camadas:**

**1. Entidades (Entities):** Contêm as regras de negócio mais genéricas e independentes.

**2. Casos de Uso (Use Cases):** Contêm a lógica específica da aplicação e orquestram as entidades.

**3. Interface de Entrada (Interface Adapters):** Traduzem dados entre o mundo externo e os casos de uso.

**4. Infraestrutura (Frameworks & Drivers):** Contém detalhes como banco de dados e frameworks web.

* **Princípios-chave:**
* Independência de tecnologia: permite trocar tecnologias sem afetar as regras de negócio.
* Testabilidade: facilita a criação de testes automatizados.

• Separação de interesses: cada camada tem uma responsabilidade clara

## Como funcionam juntos no .NET?

No **.NET**, A junção do **DDD** com a **Clean Architecture** é poderosa porque combina o foco no domínio do negócio com uma estrutura de software bem organizada. O DDD fornece o modelo de domínio e a linguagem ubíqua, enquanto a Clean Architecture oferece uma forma de estruturar o código de maneira que o domínio fique no centro da aplicação. Isso resulta em sistemas mais fáceis de manter, testar e evoluir, com regras de negócio bem definidas e isoladas de detalhes técnicos.

### Estrutura típica:

* **Domain:** Entidades, VOs, interfaces de repositório.
* **Application:** Casos de uso e serviços de aplicação.
* **Infrastructure:** Implementação de repositórios e integrações.
* **API:** Controllers e endpoints.

### Fluxo:

1. A **API recebe a requisição**.
2. Chama um **caso de uso** na camada Application.
3. O caso de uso **aplica as regras de negócio**.
4. Usa repositório (infraestrutura) para persistência.
5. A **API retorna a resposta ao usuário**.

## Benefícios de usar DDD + Clean Architecture

* Regras de negócio isoladas.
* Facilita testes unitários no domínio.
* Facilita manutenção e evolução.
* Facilita evolução sem quebrar o sistema.
* Alinha o código ao negócio.

## Exemplo prático:

Em um sistema de receitas:

- **Domain:** Receita, Ingrediente com regras (“receita precisa ter pelo menos um ingrediente”).

- **Application:** CriarReceitaUseCase para orquestrar a criação.

- **Infrastructure:** Implementa IReceitaRepository usando EF Core.

- **API:** POST /receitas chama o CriarReceitaUseCase.

Usando DDD e Clean Architecture, você constrói um sistema limpo, organizado e escalável, pronto para crescer sem bagunça.