**Conceito de DDD:**

*Domain Driven Design* significa projeto orientado à domínio. Ele veio a partir de um livro de Eric Evans, que é praticamente um catálogo de padrões, baseados nas experiências do autor. Mas o que seria um padrão?

Um padrão é uma regra de três partes que expressa a relação entre um determinado contexto, um problema e uma solução.

O DDD também é visto por alguns como a volta da orientação a objetos. Quando falamos em POO, pensamos logo em herança, polimorfismo, heranças e encapsulamento. Porém, a essência da POO tem outros conceitos, tais como:

- Alinhamento de código com o negócio: O contato de desenvolvedores com os especialistas de domínio é algo essencial quando se faz DDD.

- Favorecer a reutilização: Os blocos de construção, como veremos adiante, facilitam aproveitar um mesmo conceito de domínio ou mesmo um código em vários lugares.

- Mínimo de acoplamento: Com um modelo bem feito, as várias partes de um sistema interagem sem que haja muita dependência entre módulos ou classes de objetos de conceitos distintos.

- Independência de tecnologia: DDD não foca em tecnologia, mas sim em entender as regras do negócio e como elas devem estar refletidas no código e no modelo de domínio. A tecnologia utilizada é importante, mas não é uma preocupação do DDD.

**Principais pontos sobre o DDD:**

- Não é uma arquitetura: Embora existam vários lugares falando sobre arquitetura DDD, o DDD não é uma arquitetura por si só e sim um paradigma que é aplicado à arquitetura a qual você deseja criar ou seguir.

- O DDD não é uma tecnologia.

- Ao utilizar os conceitos do DDD, temos um software responsável, escalável, testável, com manutenção fácil e tranquilo, além de escrito com boas práticas.

**Entendimento de uma arquitetura com os conceitos DDD:**

Ao utilizar o DDD, temos quatro camadas, sendo elas a de apresentação, aplicação, domínio e infraestrutura.

- Camada de Apresentação: É responsável por abranger tudo no que diz à interações com o cliente. No nosso caso, a camada de apresentação é uma aplicação de serviço, por ser uma API Rest.

- Camada de Aplicação: É responsável por fazer com que a camada de apresentação se comunique com o domínio. Nela são implementados: Classes de serviço da aplicação (Controllers), Interfaces, DTOs e AutoMapper. Aqui não devem ser contidas regras de negócio, ela somente coordena a execução de uma tarefa e delega para os objetos de domínio.

- Camada de Domínio: É responsável por ter uma modelagem sólida e é aqui que a mágica do DDD acontece. Nessa camada temos as Entidades, Interfaces para serviços e repositórios, classes dos serviços de domínio e validações, caso necessário.

- Camada de Infraestrutura: Dá suporte as demais camadas, é dividida em duas camadas com seus respectivos conteúdos.

- Data: Possui os repositórios, DataModel (mapeamento) e persistência de dados.

- Crosscutting: Camada que atravessa todas as outras, possuindo referência à todas elas. Assim sendo, ela é responsável pela injeção de dependências.