
# ADVANCED BUSINESS DEVELOPMENT WITH .NET

Prof. Humberto Oliveira

# Agenda

- Arquitetura de Software
- Padrões de Arquitetura
- Camadas e Modularização

# O que é arquitetura de software?

# ARQUITETURA DE SOFTWARE

**É dificil de definir de maneira precisa o que é arquitetura de software.**

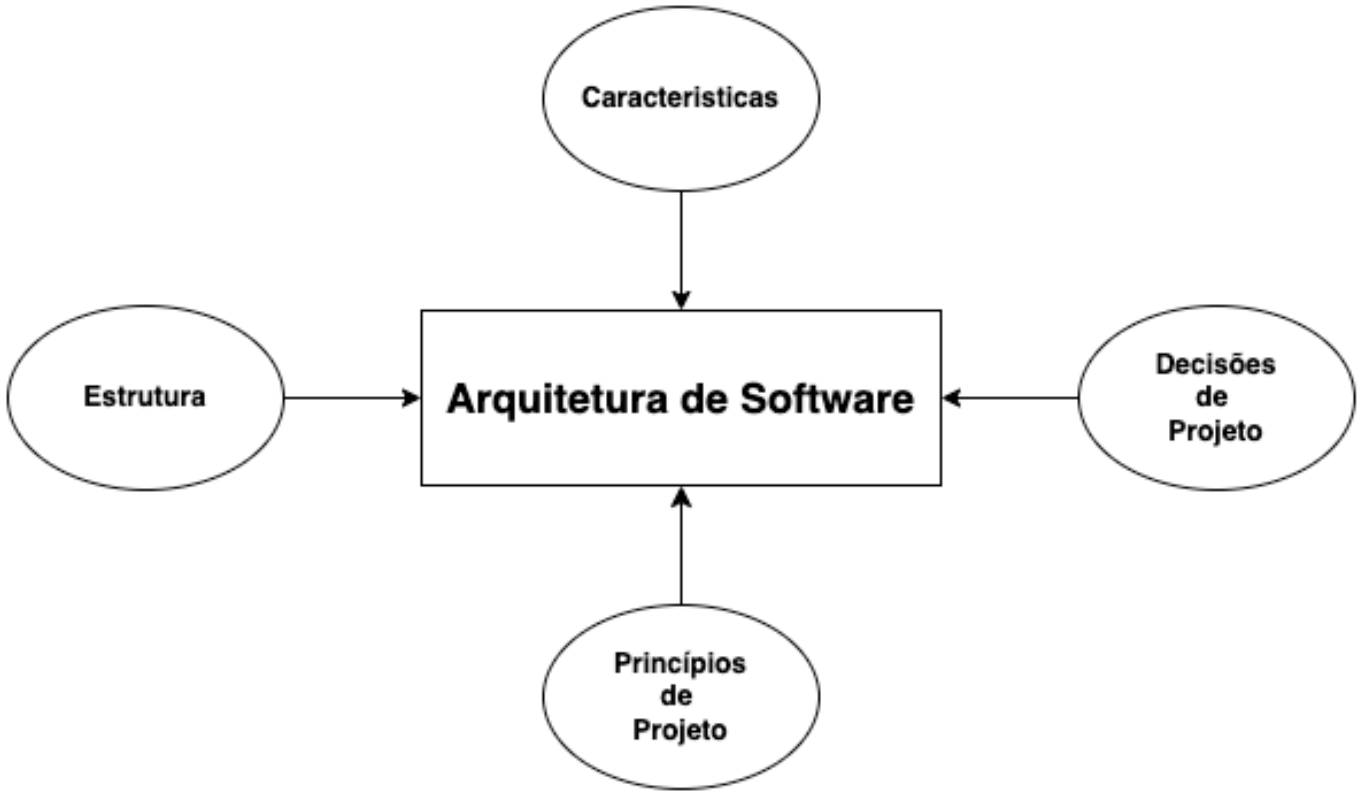
Um definição que acho pertinente é do Robert Martin (Uncle Bob), autor de livros como **Clean Code**, **Clean Architecture** que é:

*** Arquitetura de software é a estrutura do sistema definida pelos seus criadores.**

Ou seja, a estrutura envolve a divisão do sistema em componentes, o arranjo desses componentes e a forma como eles se comunicam. Essa definição orienta decisões que impactam qualidade, desempenho e escalabilidade.

# ARQUITETURA DE SOFTWARE

Segundo o livro "**Fundamentals of Software Architecture**" (Richards e Ford), os autores consideram quatros aspectos que se combinam para formar a arquitetura de software:



# ARQUITETURA DE SOFTWARE

1. **Estrutura**: Consiste no estilo arquitetural utilizado no sistema. Ex: **microsserviços**, **arquitetura em camadas**, **monolítica** e etc.

2. **Características**: Esse aspecto considera as caracteristicas do projeto. Ex: **escalabilidade**, **confiabilidade**, **disponibilidade**, **segurança**, **testabilidade** e etc.

3. **Decisões de projeto**: esse aspecto inclui as decisões arquitetônicas do projeto. Ex: O acesso a dados será feito somente pela **camada de Dados** (Repositório).

4. **Princípios de projeto**: refere-se as praticas adotadas para guiar o desenvolvimento. Ex: Todas as chamada e respostas da API serão assincronas.

# PADRÕES ARQUITETURAIS

**Padrões arquiteturais** são soluções adotadas, para resolver problemas comuns na organização do software, oferecendo estruturas e práticas para melhorar a eficiência e a qualidade do design de software.

**Alguns exemplos:**

- **Monolítica**: Um único sistema onde todos os componentes são interdependentes.
- **Microservices**: Aplicação composta por pequenos serviços independentes que se comunicam entre si.
- **Event-Driven**: Baseada em eventos que acionam ações específicas no Sistema. (*Se comunica atraves de um serviço de mensageria RabbiTMQ*)
- **Serverless**: Execução de código sob demanda em ambientes gerenciados, sem a necessidade de gerenciar servidores (Firebase) .

# PADRÕES ARQUITETURAIS – MONOLÍTICA

Uma arquitetura **monolítica** é uma abordagem tradicional no desenvolvimento de software na qual todos os componentes de uma aplicação são combinados em uma única unidade totalmente integrada**.**

**Vantagens:**

- **Simples de entender e desenvolver**: A arquitetura monolítica é simples e direta, tornando mais fácil para os desenvolvedores entender a aplicação e seus componentes.

- **Fácil de implantar**: Implantar uma aplicação monolítica é um processo direto porque todos os componentes são combinados em uma única base de código.

- **Fácil de testar**: Testar uma aplicação monolítica é mais fácil porque todos os componentes são integrados em uma única unidade, permitindo executar um único conjunto de testes para verificar a funcionalidade de toda a aplicação.

# PADRÕES ARQUITETURAIS - MONOLÍTICA

# PADRÕES ARQUITETURAIS - MONOLÍTICA

**Desvantagens**:

- **Desafios de escalabilidade**: As aplicações monolíticas podem se tornar lentas e sem resposta à medida que aumentam de tamanho, dificultando o dimensionamento da aplicação para atender às demandas crescentes.

- **Modularidade limitada**: As aplicações monolíticas são totalmente integradas, dificultando a adição de novos recursos ou a alteração de recursos existentes sem afetar toda a aplicação.

- **Falta de agilidade**: As aplicações monolíticas geralmente são criadas com uma única base de código, dificultando o trabalho das equipes em diferentes componentes em paralelo, levando a tempos de desenvolvimento lentos.

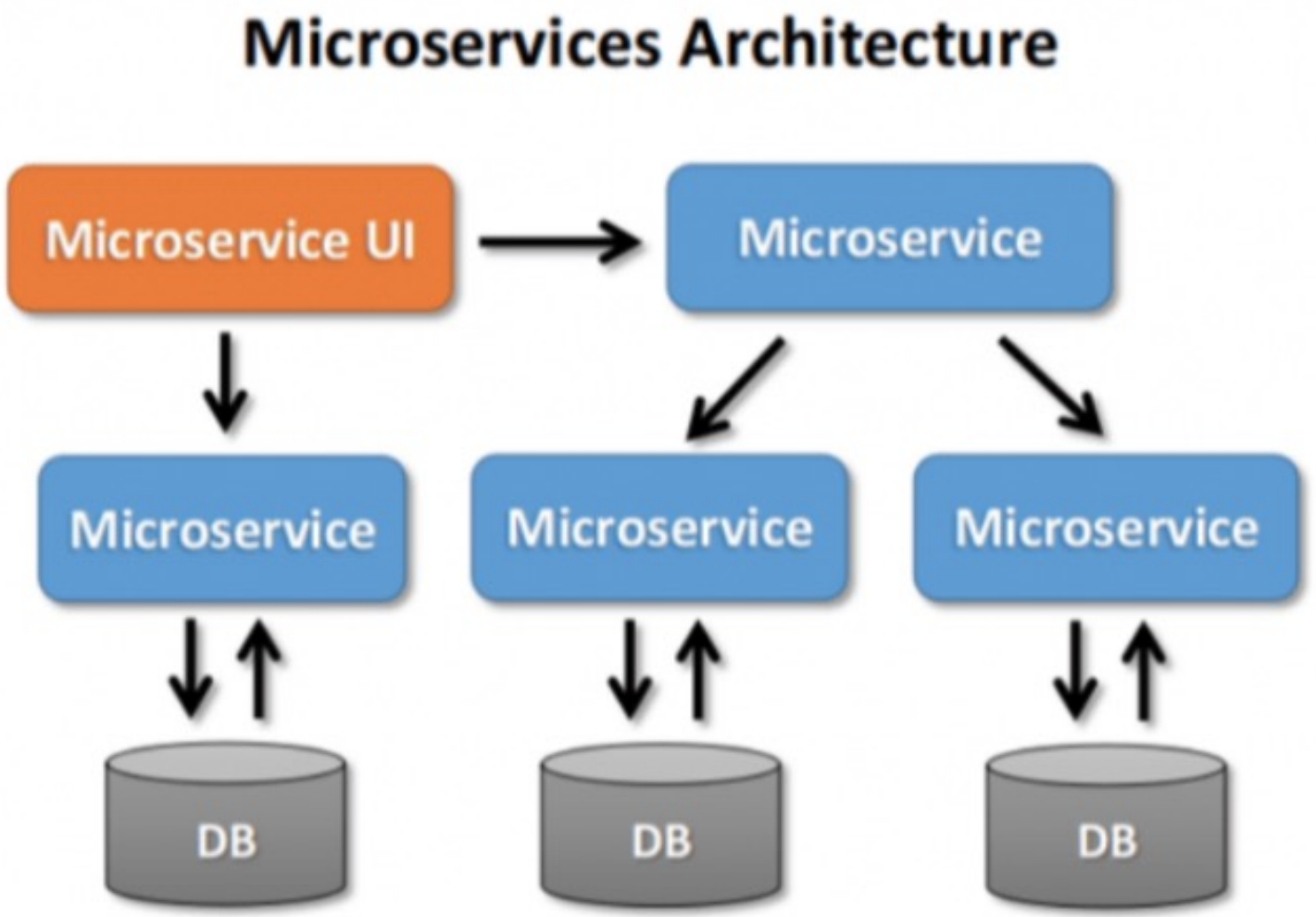
# PADRÕES ARQUITETURAIS – MICROSERVIÇOS

Os **microserviços** são uma abordagem arquitetônica que divide uma aplicação complexa em uma coleção de serviços pequenos, levemente acoplados, que se comunicam entre si por meio de APIs.

**Vantagens** :

- **Escalabilidade**: Permite o dimensionamento independente de serviços individuais, facilitando o gerenciamento de demandas crescentes e aumento do tráfego.
- **Modularidade**: Cada microserviço pode ser desenvolvido e implantado de forma independente, facilitando a adição de novos recursos ou a alteração de recursos existentes sem afetar toda a aplicação.
- **Agilidade**: A capacidade de trabalhar em cada microserviço em paralelo torna o processo de desenvolvimento mais rápido e eficiente.
- **Resiliência**: Pode fornecer melhor resiliência e tolerância a falhas, pois falhas em um serviço não afetam toda a aplicação.

# PADRÕES ARQUITETURAIS – MICROSERVIÇOS



# PADRÕES ARQUITETURAIS – MICROSERVIÇOS

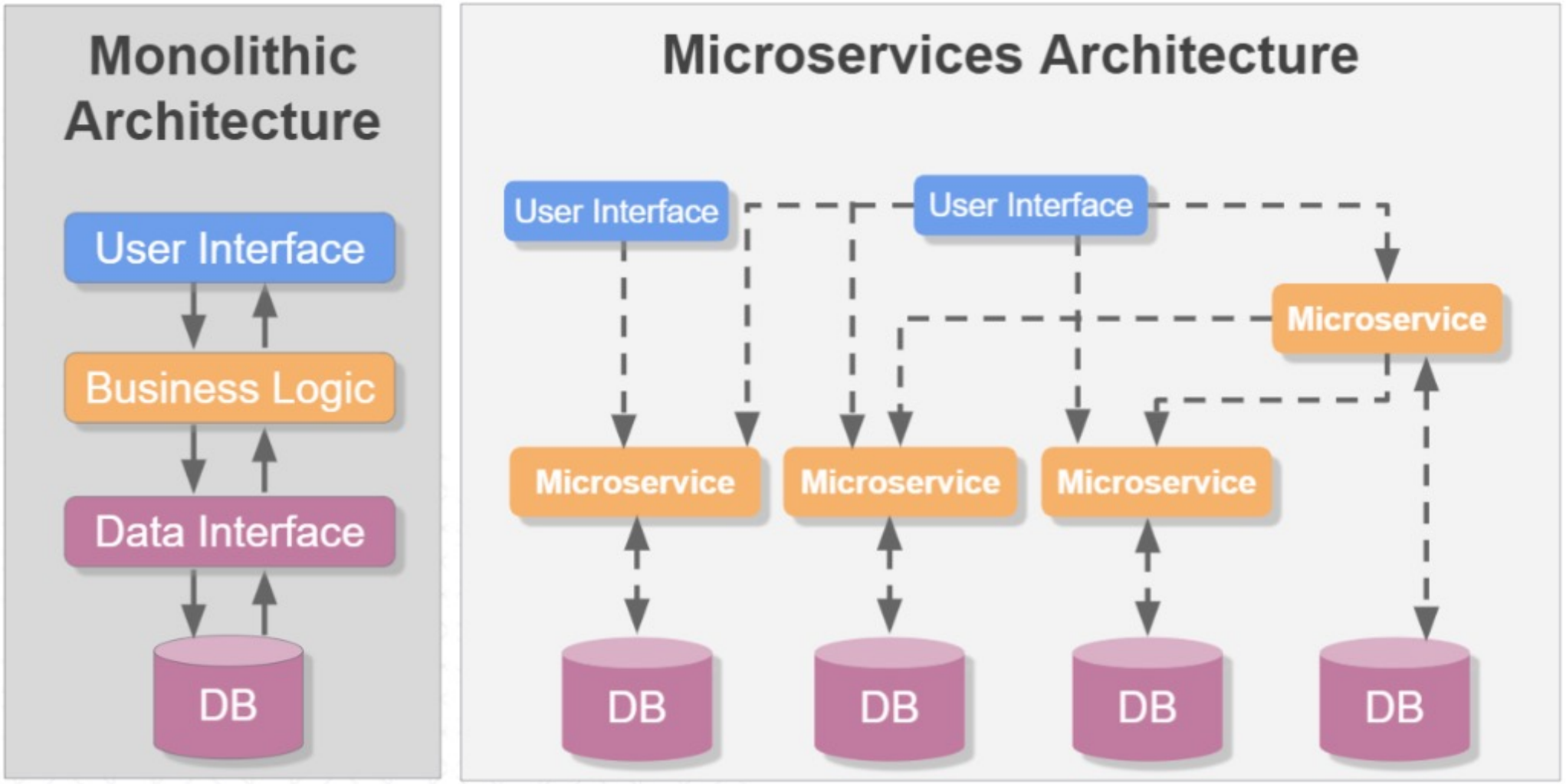
**Desvantagens** :

- **Complexidade**: Mais complexa do que a arquitetura monolítica, dificultando o entendimento dos desenvolvedores sobre a aplicação e seus componentes.

- **Maior sobrecarga operacional**: Gerenciar um grande número de microserviços pode ser desafiador e pode aumentar a sobrecarga operacional, como implementação, monitoramento e teste de serviços individuais.

- **Comunicação entre serviços**: A comunicação entre microserviços pode apresentar problemas de latência e confiabilidade, levando à degradação do desempenho e dificuldade na depuração.

# PADRÕES ARQUITETURAIS



# CAMADAS E MODULARIZAÇÃO

No contexto da arquitetura de software, **camadas** e **modularização** são conceitos cruciais para a construção de sistemas escaláveis, manuteníveis e de alta qualidade. Ambos os conceitos ajudam a organizar o código de maneira que facilite a separação de responsabilidades e a reutilização, contribuindo para uma arquitetura mais robusta e eficiente.

Alguns exemplos de arquiteturas que utilizam essa organização:

- **Clean Architecture:** Organiza o código em camadas circulares, priorizando a independência de frameworks e interfaces de usuário.

- **Onion Architecture :** Enfatiza o núcleo da aplicação, rodeado por anéis concêntricos de dependências.

- **Hexagonal Architecture:** Também conhecida como **Ports** and **Adapters**, foca em criar um núcleo de aplicação independente com interfaces para interações externas, promovendo flexibilidade e facilidade de teste.

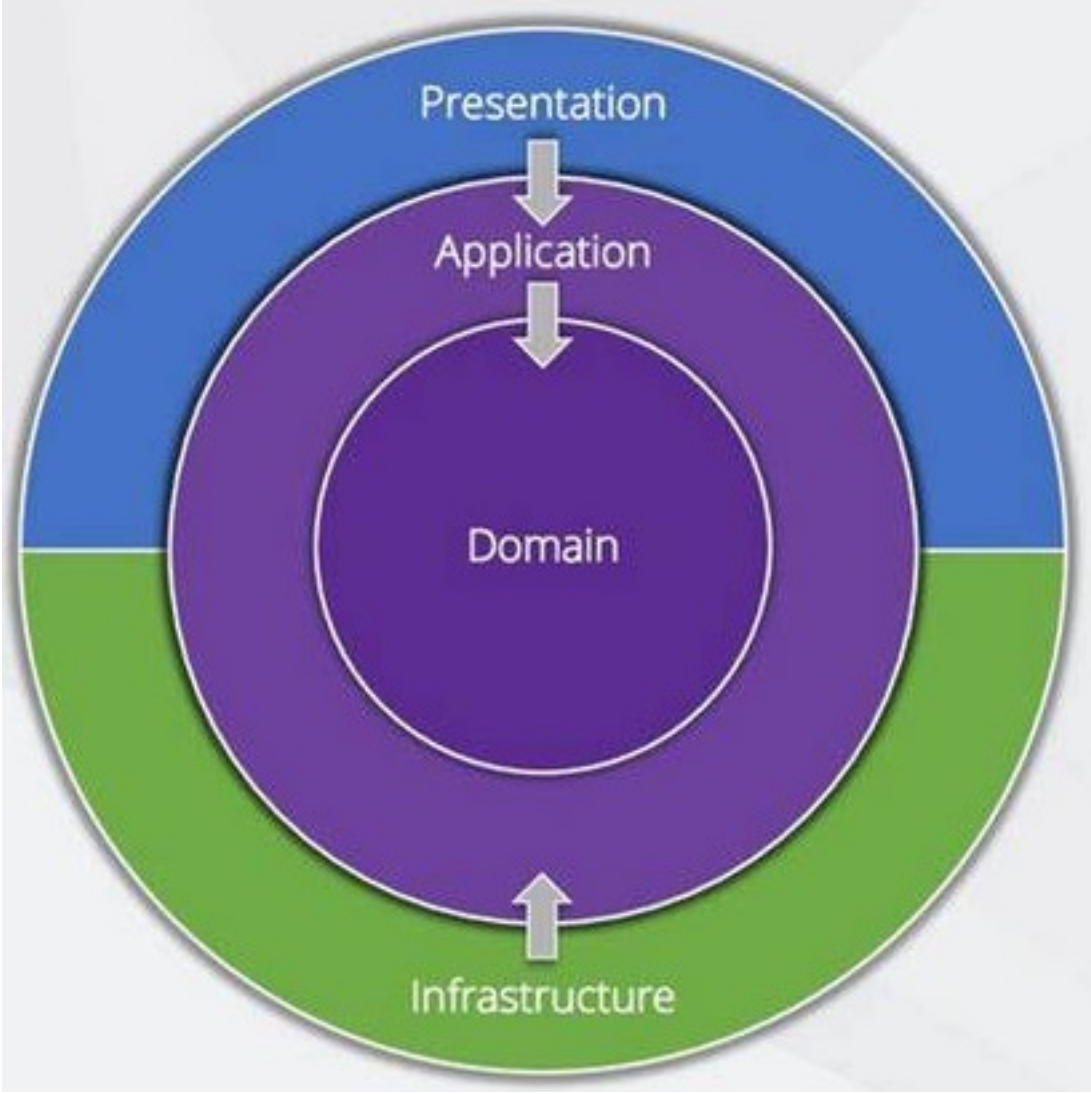
# CAMADAS E MODULARIZAÇÃO

**Apresentação (ou Interface do Usuário)**: Responsável pela interação e exibição de informações ao usuário.

**Aplicação (ou Camada de Serviços)**: Implementa a lógica de aplicação e coordena atividades entre apresentação e domínio.

**Domínio (ou Lógica de Negócio)**: Encapsula a lógica de negócios central do sistema.

**Infra estrutura/Persistencia (ou Acesso a Dados)**: Gerencia a interação com o armazenamento de dados e mapeamento entre banco de dados e objetos de domínio.



# CAMADAS E MODULARIZAÇÃO

**Modularização** é o processo de dividir um sistema em módulos independentes que podem ser desenvolvidos, testados e mantidos separadamente. Cada módulo deve ter uma responsabilidade claramente definida e uma interface bem especificada para interagir com outros módulos.

Os principais benefícios da modularização incluem:

- **Reutilização**: Módulos podem ser reutilizados, reduzindo duplicação de código e melhorando a consistência.
- **Facilidade de Manutenção**: Alterações em um módulo têm menos impacto em outros, facilitando a manutenção.
- **Desenvolvimento Paralelo**: Equipes diferentes podem trabalhar em módulos distintos simultaneamente.
- **Escalabilidade**: Sistemas modulares são mais fáceis de escalar com novos módulos adicionados ou removidos.
- **Isolamento de Erros**: Problemas em um módulo são mais fáceis de isolar e corrigir, reduzindo falhas no sistema.

# FIM