CENTRO UNIVERSITÁRIO UNISATC

BACK-END

Prof. Bruno Kurzawe

Estudo e implementação de novas tecnologias de WEB e Modelagem de soluções dinâmicas

Arquitetura de Microservices

Definição

A arquitetura de microserviços é um estilo arquitetônico que estrutura uma aplicação como um conjunto de serviços independentes, cada um executando um processo de negócio específico.

Funcionamento da Arquitetura de Microserviços

Cada serviço é:

- **Independente**: Pode ser desenvolvido, implantado e escalado independentemente.
- **Focado**: Realiza uma função de negócio específica.
- **Comunicativo**: Interage com outros serviços por meio de APIs.

Características Principais

- **Desacoplamento**: Cada serviço é independente.
- **Escalabilidade**: Escala horizontalmente conforme a demanda.
- **Resiliência**: Falhas em um serviço não afetam outros.
- **Facilidade de Atualização**: Atualizações são mais fáceis de implementar.

Vantagens da Arquitetura de Microserviços

- Agilidade no Desenvolvimento.
- Facilidade de Escala.
- Maior Resiliência.
- Melhor Utilização de Recursos.

Comunicação entre Microserviços

- HTTP/REST.
- Mensageria (ex: Kafka, RabbitMQ).
- Protocolos Binários (ex: gRPC).

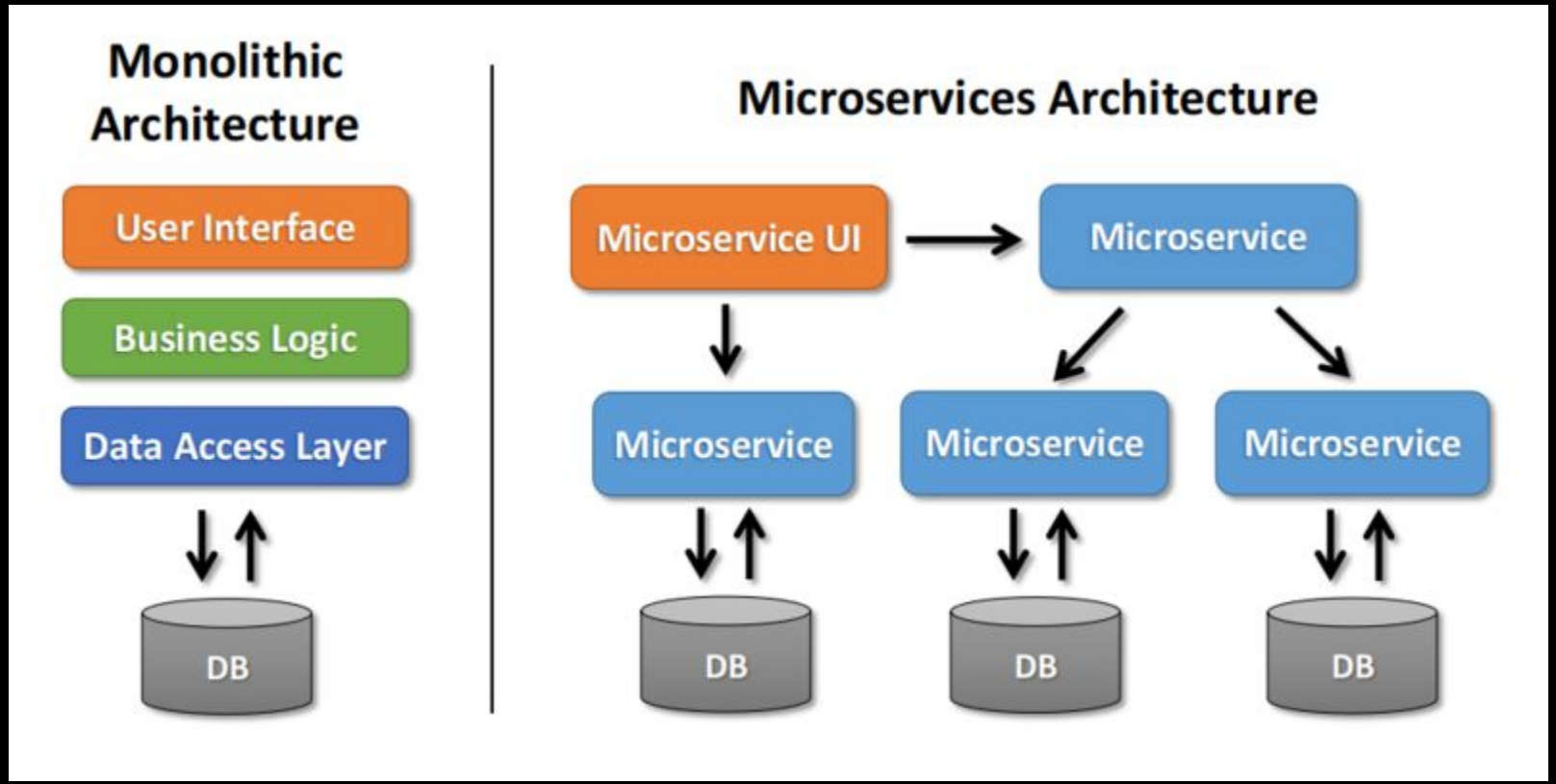
Desvantagens da Arquitetura de Microserviços

- Complexidade Operacional.
- Coordenação entre Serviços.
- Consistência de Dados Distribuídos.

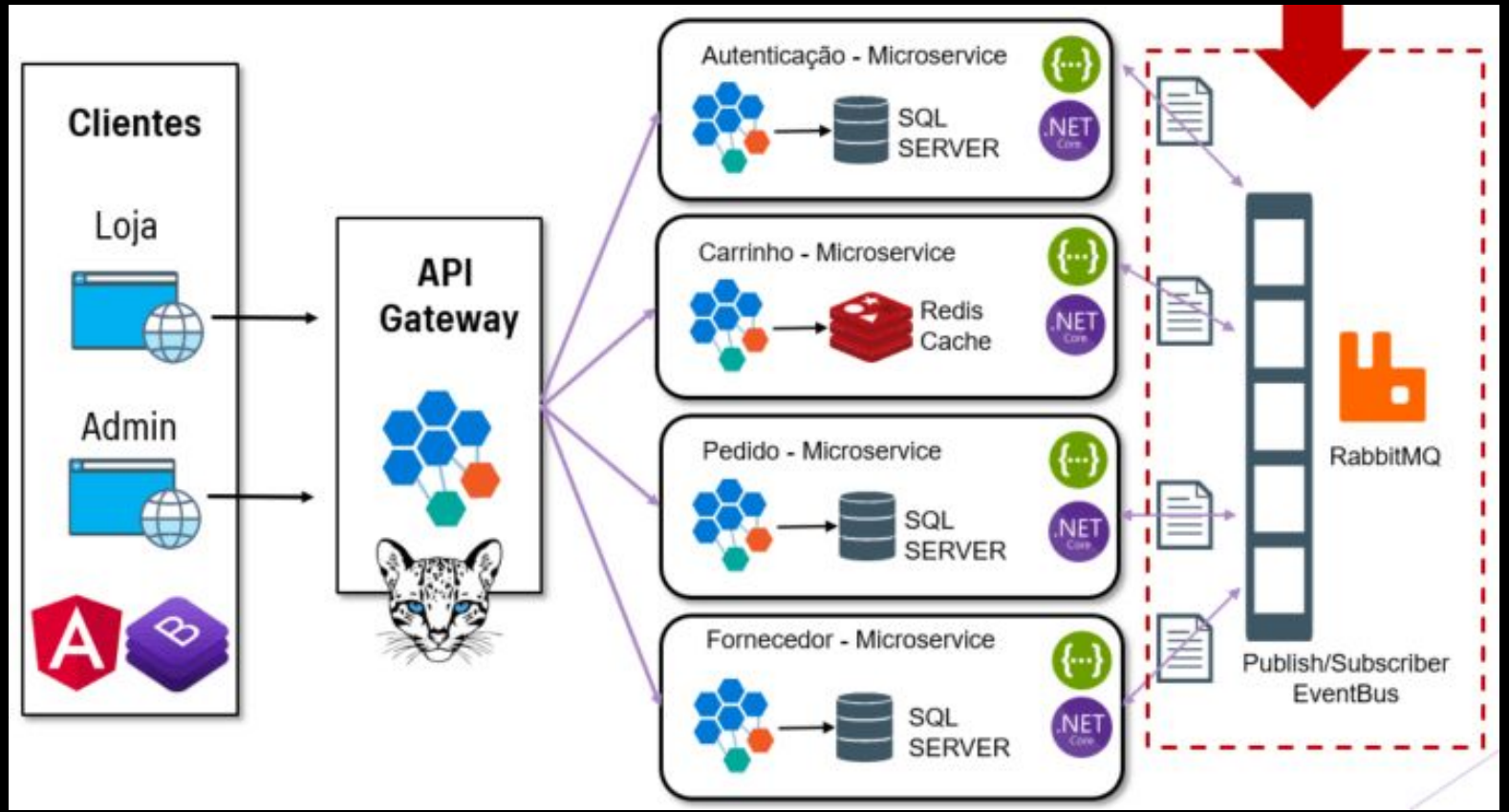
Padrões na Arquitetura de Microserviços

- Service Discovery (ex: Consul, Eureka).
- Gateway de API (ex: Zuul, API Gateway).
- Balanceamento de Carga (ex: NGINX).

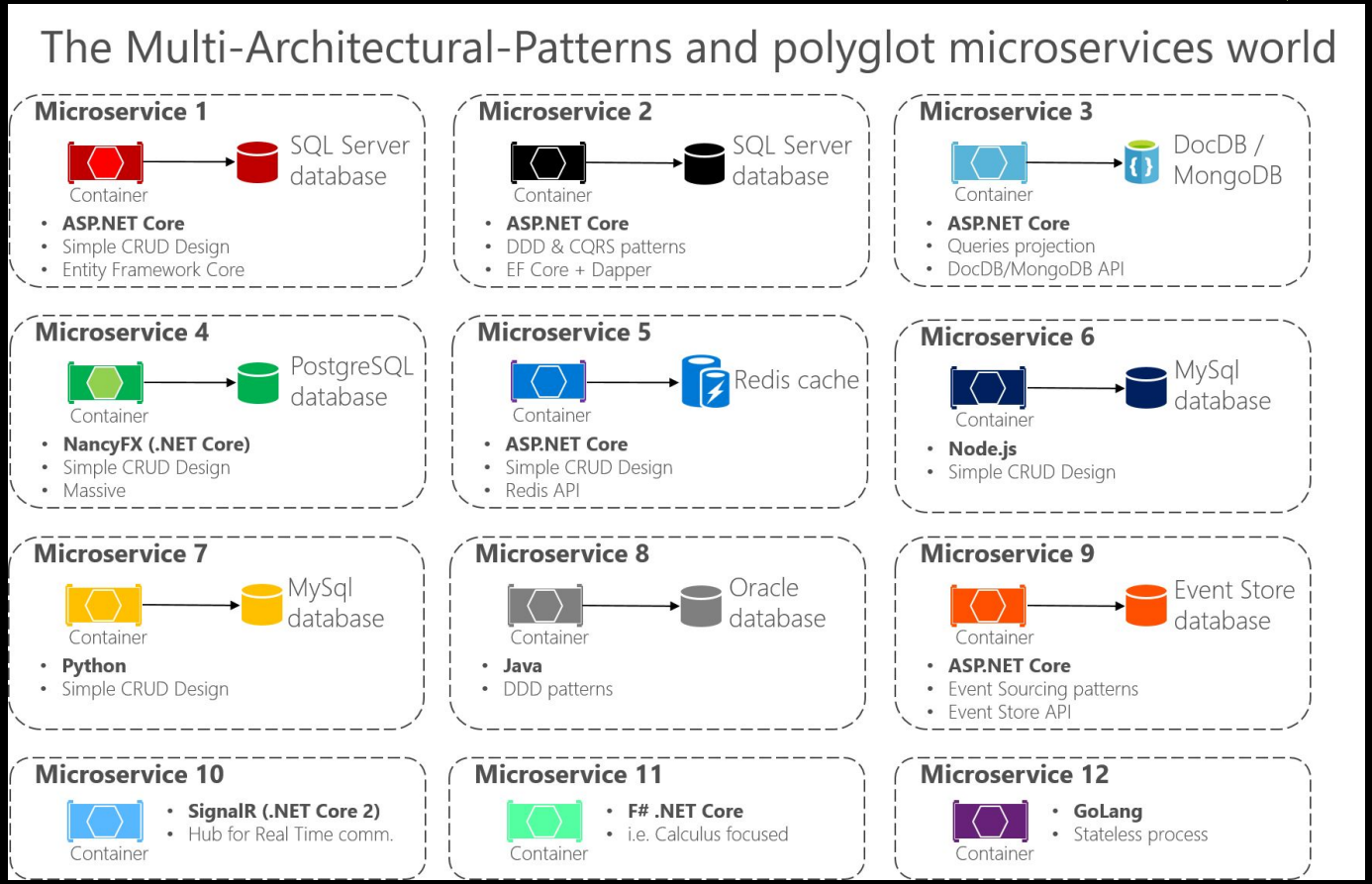
Padrões na Arquitetura de Microserviços



Padrões na Arquitetura de Microserviços



Padrões na Arquitetura de Microserviços



Empresas que Utilizam a Arquitetura de Microserviços

- Netflix.
- Spotify.
- Uber.
- Amazon.
- Airbnb.

Estudo de Caso - Netflix

- Streaming de Vídeo: Serviços separados para recomendações, reprodução, gerenciamento de contas, etc.

- Escalabilidade Dinâmica: A capacidade de escalar serviços conforme a demanda de visualizações.

Conclusão

- A arquitetura de microserviços oferece benefícios significativos, mas também apresenta desafios.

- A escolha de adotar essa arquitetura deve ser baseada nas necessidades específicas do projeto e na complexidade do domínio do negócio.

Contêineres, Docker e Kubernetes

Definição

Contêineres são unidades de software que empacotam e isolam aplicações com suas dependências, enquanto Docker é uma plataforma de código aberto para criar e executar contêineres. Kubernetes é uma plataforma de orquestração para automação, escalabilidade e gerenciamento de contêineres.

Características dos Contêineres

- Isolamento: Contêineres executam de maneira isolada.
- Portabilidade: Funcionam consistentemente em qualquer ambiente.
- Eficiência: Compartilham o mesmo kernel do sistema operacional.

Vantagens do Uso de Contêineres e Docker

- Consistência: Ambientes de desenvolvimento e produção idênticos.
- Rapidez: Inicialização e execução rápidas.
- Escalabilidade: Facilidade em escalar horizontalmente.
- Versionamento: Controle preciso de versões de aplicações.

Desvantagens dos Contêineres e Docker

- Complexidade de Rede: Gerenciamento de redes em ambientes distribuídos.
- Segurança: Requer atenção especial para garantir ambientes seguros.
- Overhead: Alguma sobrecarga de recursos em comparação com execução nativa.

Características do Kubernetes

- Orquestração Automática: Distribuição e escalabilidade automática de contêineres.
- Desvios de Falhas: Detecção e recuperação automática de falhas.
- Deployments Declarativos: Descreve o estado desejado da aplicação.

Vantagens da Orquestração com Kubernetes

- Escalabilidade: Facilita a escalabilidade horizontal.
- Alta Disponibilidade: Garante disponibilidade contínua de serviços.
- Atualizações Contínuas: Permite implementações sem tempo de inatividade.

Desvantagens do Kubernetes

- Complexidade Inicial: Implementação pode ser complexa para iniciantes.
- Recursos Necessários: Pode exigir recursos significativos.

Empresas que Utilizam Docker e Kubernetes

- Google
- Microsoft
- Amazon
- IBM
- Spotify

Cases de Sucesso com Docker e Kubernetes

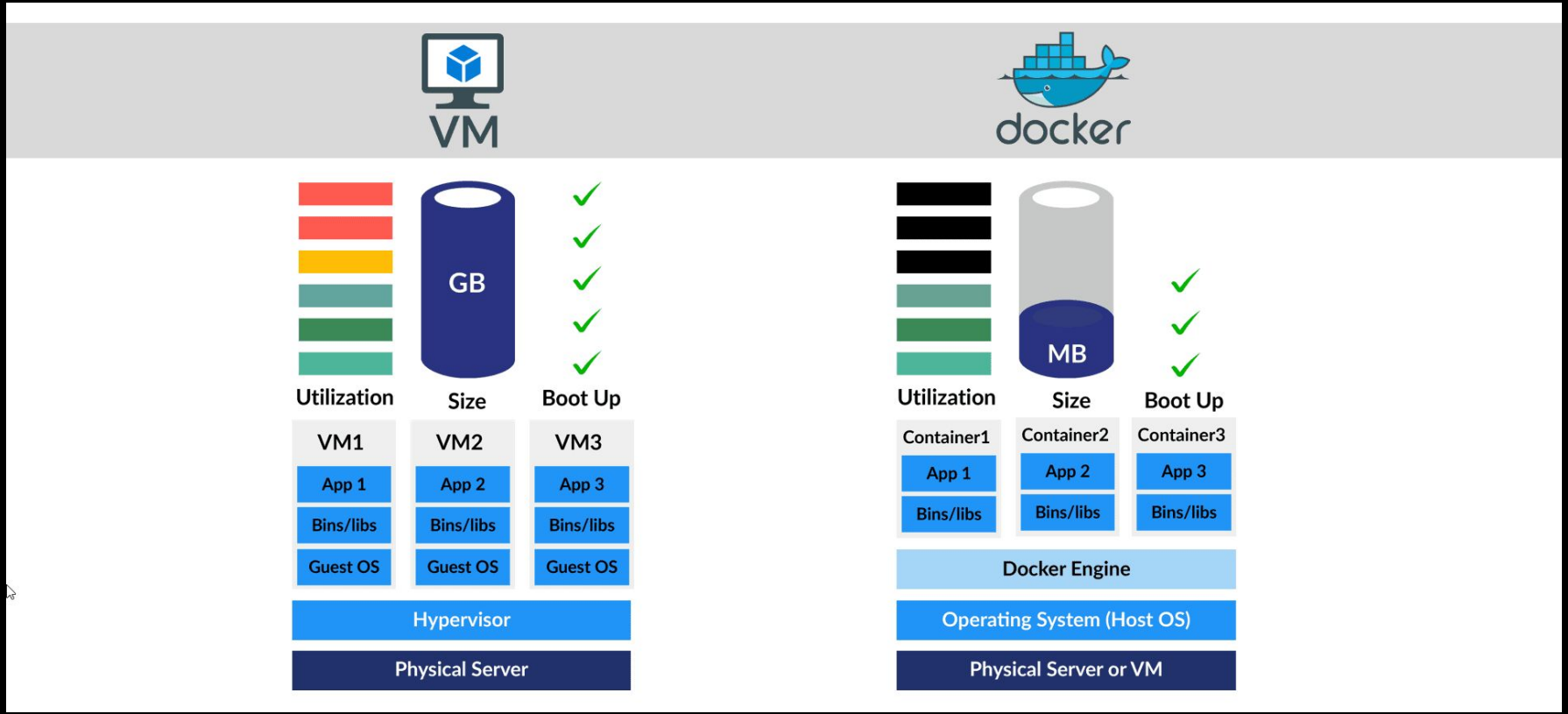
- Spotify: Utiliza contêineres para isolamento e escalabilidade eficiente.
- Uber: Usa Kubernetes para orquestração e gerenciamento simplificado de contêineres.

Conclusão

- Contêineres e suas tecnologias associadas revolucionaram o desenvolvimento e implantação de aplicações.

- A combinação de Docker e Kubernetes oferece uma solução poderosa para empacotamento, distribuição e orquestração de aplicações em escala.

VMs x Containers



Abstração do Sistema Operacional

- Máquinas Virtuais: Cada VM inclui uma cópia completa do sistema operacional, conhecido como "hipervisor", que emula o hardware físico para suportar múltiplos sistemas operacionais em uma máquina física. Isso leva a uma abordagem mais pesada, já que cada VM requer seu próprio sistema operacional completo, incluindo kernel e drivers.

- Contêineres: Compartilham o mesmo kernel do sistema operacional hospedeiro, mas isolam processos, bibliotecas e recursos. Isso significa que os contêineres são mais leves e rápidos para iniciar, pois não precisam emular um sistema operacional completo.

Isolamento

- Máquinas Virtuais: Oferecem isolamento mais robusto, já que cada VM executa seu próprio sistema operacional independente. Isso proporciona uma separação mais completa, mas ao custo de uma maior sobrecarga e consumo de recursos.

- Contêineres: Compartilham o mesmo kernel, o que os torna mais eficientes e leves. No entanto, o isolamento não é tão forte quanto o das VMs. Contêineres são ideais para ambientes onde o isolamento não precisa ser tão rígido, como em aplicações distribuídas e microserviços.

Overhead e Eficiência

- Máquinas Virtuais: Tendem a ter um overhead maior de recursos devido à duplicação dos sistemas operacionais completos. Cada VM precisa alocar recursos para o sistema operacional, drivers e aplicativos.

- Contêineres: São mais eficientes, pois compartilham recursos do sistema operacional hospedeiro. O overhead é menor, resultando em tempos de inicialização mais rápidos e maior densidade de carga de trabalho em uma máquina física.

Portabilidade

- Máquinas Virtuais: Mais portáteis, pois encapsulam todo o ambiente, incluindo o sistema operacional.

- Contêineres: Também portáteis, mas são dependentes do ambiente do contêiner, como o Docker. A portabilidade é facilitada porque os contêineres incluem apenas o que é necessário para executar a aplicação, não o sistema operacional completo.

GraphQL

Definição

GraphQL é uma linguagem de consulta para APIs, criada pelo Facebook, que oferece uma alternativa eficiente e poderosa às APIs REST tradicionais. Vamos explorar como o GraphQL se alinha aos princípios fundamentais de design de software.

Vantagens do GraphQL

- Consulta Personalizada: Os clientes solicitam apenas os dados necessários.
- Redução Overfetching/Underfetching: Evita o problema de obter mais ou menos dados do que o necessário.
- Versatilidade: Suporta diversas fontes de dados e tipos de clientes.
- Versão Única: Evita a necessidade de várias versões de API.

Desvantagens do GraphQL

- Complexidade Inicial: Pode parecer complexo, especialmente para projetos simples.

- Segurança: Requer cuidados extras para evitar ataques de negação de serviço (DDoS) através de consultas complexas.

Características do GraphQL

- Sistema de Tipos: Define a estrutura dos dados.
- Consulta Declarativa: Clientes especificam exatamente o que precisam.
- Mutations: Permite modificar dados no servidor.
- Subscriptions: Oferece suporte a atualizações em tempo real.

Princípios de Design de Software

- SOLID: Princípios de responsabilidade única, aberto/fechado, substituição de Liskov, segregação de interface e inversão de dependência.

- DRY (Don't Repeat Yourself): Evita duplicação de código para manter a consistência.

- KISS (Keep It Simple, Stupid): Favorece a simplicidade e a clareza no design.

- Princípio da Menor Surpresa: O comportamento de uma parte do sistema não deve surpreender outra parte.

Alinhamento de GraphQL aos Princípios de Design

- Simplicidade: Reduz overfetching/underfetching, simplificando a obtenção de dados.

- Flexibilidade: A consulta declarativa permite que os clientes obtenham exatamente o que precisam.

- Escalabilidade: Adequado para projetos de diferentes tamanhos e complexidades

Empresas que Utilizam GraphQL

- Facebook: Criador do GraphQL, utiliza extensivamente em seus produtos.
- GitHub: Usa GraphQL em sua API para fornecer aos desenvolvedores flexibilidade nas consultas.
- Twitter: Adotou GraphQL para melhorar a eficiência das consultas em sua API.

Exemplo Prático de Uso de GraphQL

Considerações Finais

- GraphQL oferece vantagens significativas: Flexibilidade, eficiência e simplicidade.

- Integração com Princípios de Design: Alinha-se aos fundamentos do design de software, promovendo boas práticas.

Fim da aula 12…