Desenvolvimento Web II

Aula 02 - Padrões de projeto

Prof. Fabricio Bizotto

Instituto Federal Catarinense

fabricio.bizotto@ifc.edu.br

Ciência da Computação

6 de janeiro de 2024

Roteiro

Conceitos

Os padrões de projeto para web são soluções reutilizáveis para problemas comuns de design de software que surgem no desenvolvimento de aplicativos web. Eles fornecem diretrizes e estruturas para organizar o código, melhorar a escalabilidade, a manutenibilidade e a eficiência do desenvolvimento. Os mais comuns são:

- **Modelo-Visão-Controlador (MVC)**
- **Modelo-Visão-Presenter (MVP)**
- **Modelo-Visão-ViewModel (MVVM)**

Padrões de Projeto

MVC

Model-View-Controller

Definição

É um dos padrões de arquitetura mais conhecidos e adotados pela indústria de software. Foi introduzido pela primeira vez no final da década de 1970 por Trygve Reenskaug, um cientista da computação norueguês, e desde então se tornou um elemento básico na arquitetura de aplicativos. O padrão facilita a separação de interesses dividindo o aplicativo em três componentes principais.

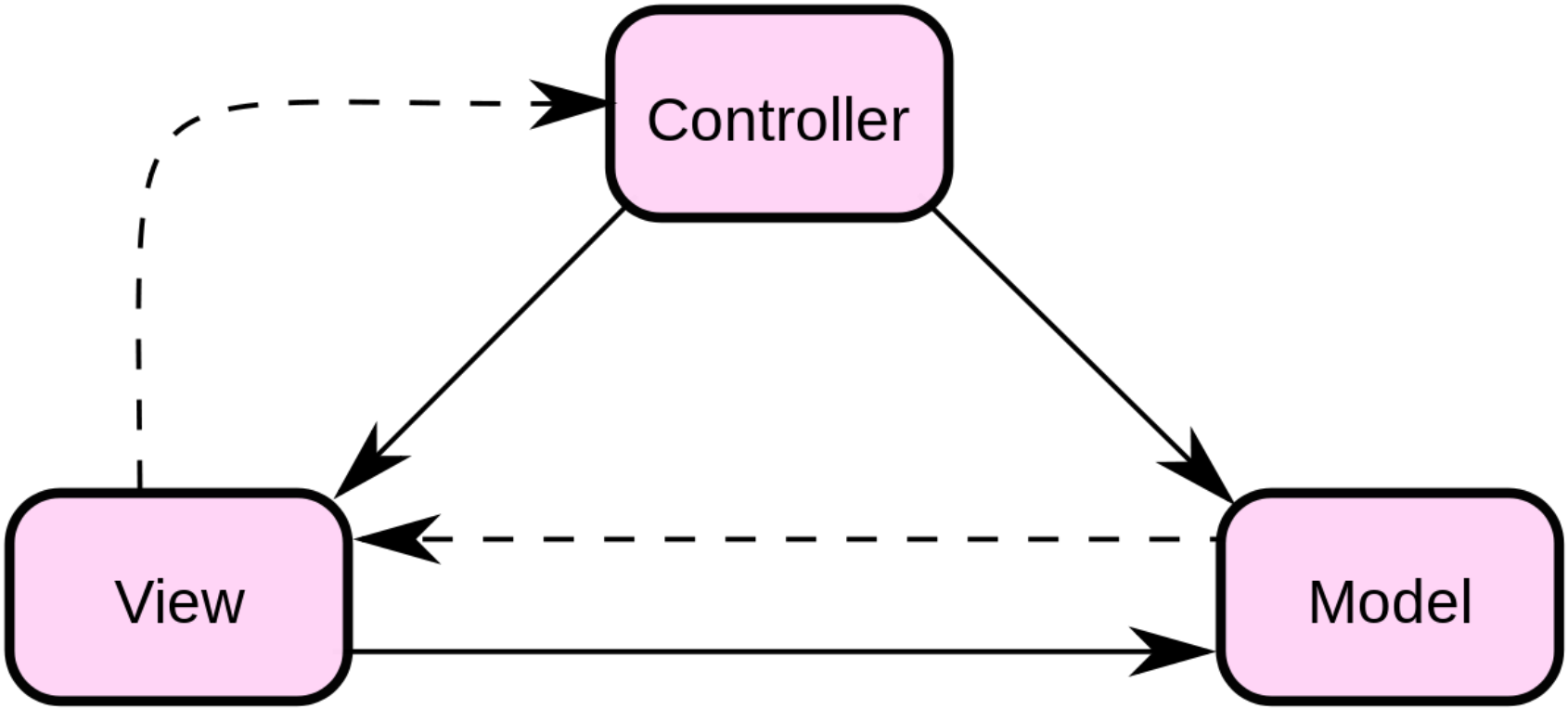


Figura: Estrutura do MVC.

Explicando

- **Model**: define quais dados o aplicativo deve conter. Se o estado desses dados mudar, o modelo geralmente notificará a visualização (*View*) (para que a exibição possa mudar conforme necessário) e, às vezes, o controlador (se for necessária uma lógica diferente para controlar a visualização atualizada).
- **View**: define como os dados do aplicativo devem ser exibidos (Interface com o Usuário). A visualização é responsável por receber a entrada do usuário e encaminhá-la para o controlador.
- **Controller**: atua como intermediário entre o modelo e a visualização. O controlador é responsável por receber a entrada do usuário da visualização e atualizar o modelo conforme necessário.

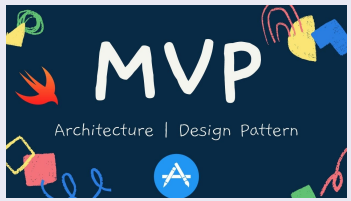
Padrões de Projeto

MVP

Model-View-Presenver

Definição

Aborda algumas das desvantagens da abordagem MVC tradicional. Originou-se no início da década de 1990 na Taligent, uma joint venture entre Apple, IBM e Hewlett-Packard. Foi ainda mais popularizado pelo Dolphin Smalltalk em 1998 e, em 2006, a Microsoft adotou o MVP para programação de interface de usuário no framework .NET.



No MVP quem manda é o View. Cada View chama seu Presenter ou possui alguns eventos que o Presenter escuta.

Exemplo

Quando o usuário clica no botão “Salvar”, o manipulador de eventos na View delega ao método “OnSave” do Presenter. O Presenter fará a lógica necessária e qualquer comunicação necessária com o Modelo e, em seguida, chamará de volta a Visualização por meio de sua interface para que a Visualização possa exibir que o salvamento foi concluído.

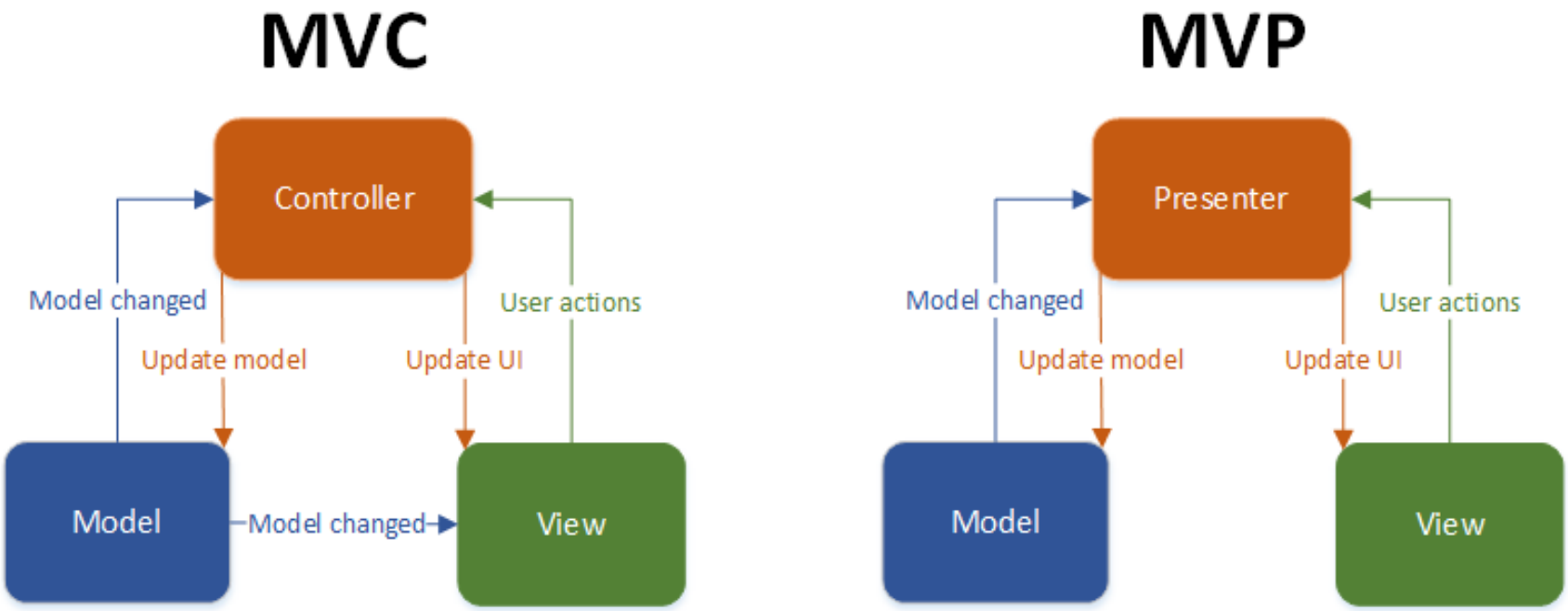
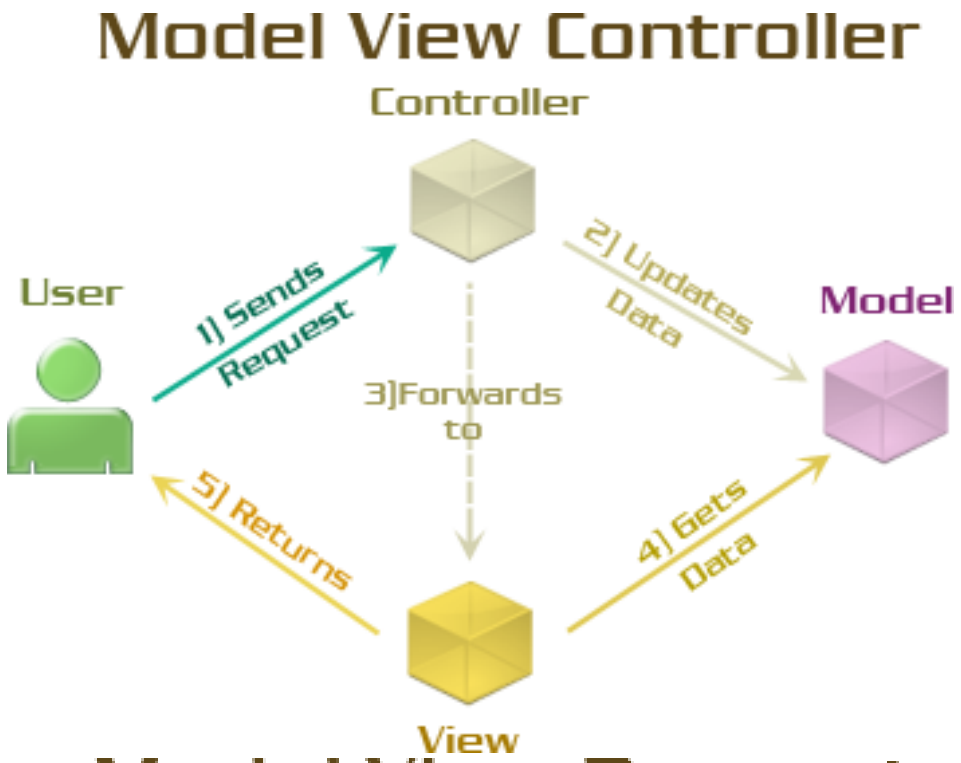


Figura: Estrutura do MVP.

MVC vs MVP

- O MVC não coloca o View no comando, os Views atuam como escravos que o Controlador pode gerenciar e direcionar.
- No MVC, as visualizações são sem estado, ao contrário das visualizações no MVP, onde são com estado e podem mudar com o tempo.
- No MVP, as Views não têm lógica e devemos mantê-las o mais burras possível. Por outro lado, Views em MVC podem ter algum tipo de lógica.
- No MVP, o Presenter é desacoplado da View e se comunica com ela através de uma interface. Isso permite zombar do View em testes unitários, ou seja, podemos testar o Presenter sem o View.
- No MVP, as visualizações são completamente isoladas do modelo. No entanto, no MVC, as Views podem se comunicar com o Modelo para mantê-lo atualizado com os dados mais atualizados.


Model View Controller
Controller
User
1] Sends Request
2] Updates Data
Model
3]Forwards to
5] Returns
4] Gets Data
View


Model View Presenter
User
View
Presenter
Model
1] Interacts
2] Notifies
3]Updates
6] Displays
5] Updates
4] Gets Data

Modelos Arquiteturais

N camadas

N-tier

Principais Arquiteturas de Software

Arquitetura em N camadas (N-tier) - Definição

A arquitetura em N camadas é um padrão de arquitetura de software no qual a aplicação é dividida em camadas lógicas ou físicas.

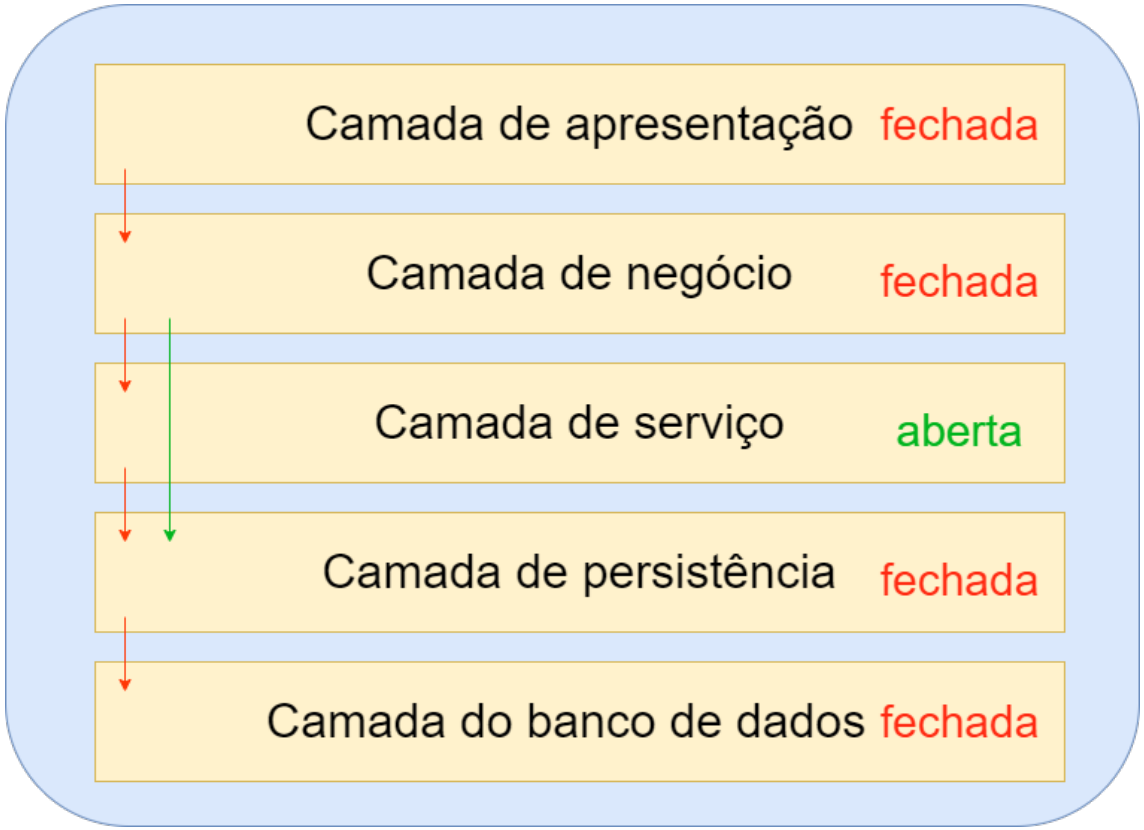


Figura: Arquitetura em camadas - Fluxo.

Vantagens

- **Separação de Responsabilidades**: A separação clara das responsabilidades em diferentes camadas (como apresentação, lógica de negócios e acesso a dados) facilita a manutenção e a evolução do sistema.
- **Escalabilidade**: A escalabilidade é facilitada, pois cada camada pode ser dimensionada independentemente das outras, permitindo a otimização de recursos.
- **Facilidade de Testes**: Cada camada pode ser testada separadamente, o que simplifica os testes unitários e facilita a identificação e correção de falhas.
- **Manutenção**: Alterações em uma camada específica não devem afetar as outras, tornando a manutenção mais simples e menos propensa a efeitos colaterais indesejados.

Desvantagens

- **Complexidade Inicial**: A implementação de uma arquitetura em camadas pode ser mais complexa inicialmente, especialmente para projetos pequenos ou simples.
- **Comunicação entre camadas**: A comunicação entre camadas pode resultar em algum overhead, especialmente em sistemas distribuídos, o que pode impactar o desempenho. Alguns exemplos são latência, serialização e desserialização de dados, etc.
- **Duplicação de Lógica**: Pode ocorrer uma duplicação de lógica entre as camadas, o que pode levar a inconsistências se não for gerenciado adequadamente.

Principais Arquiteturas de Software

Arquitetura em N camadas (N-tier) - Representação

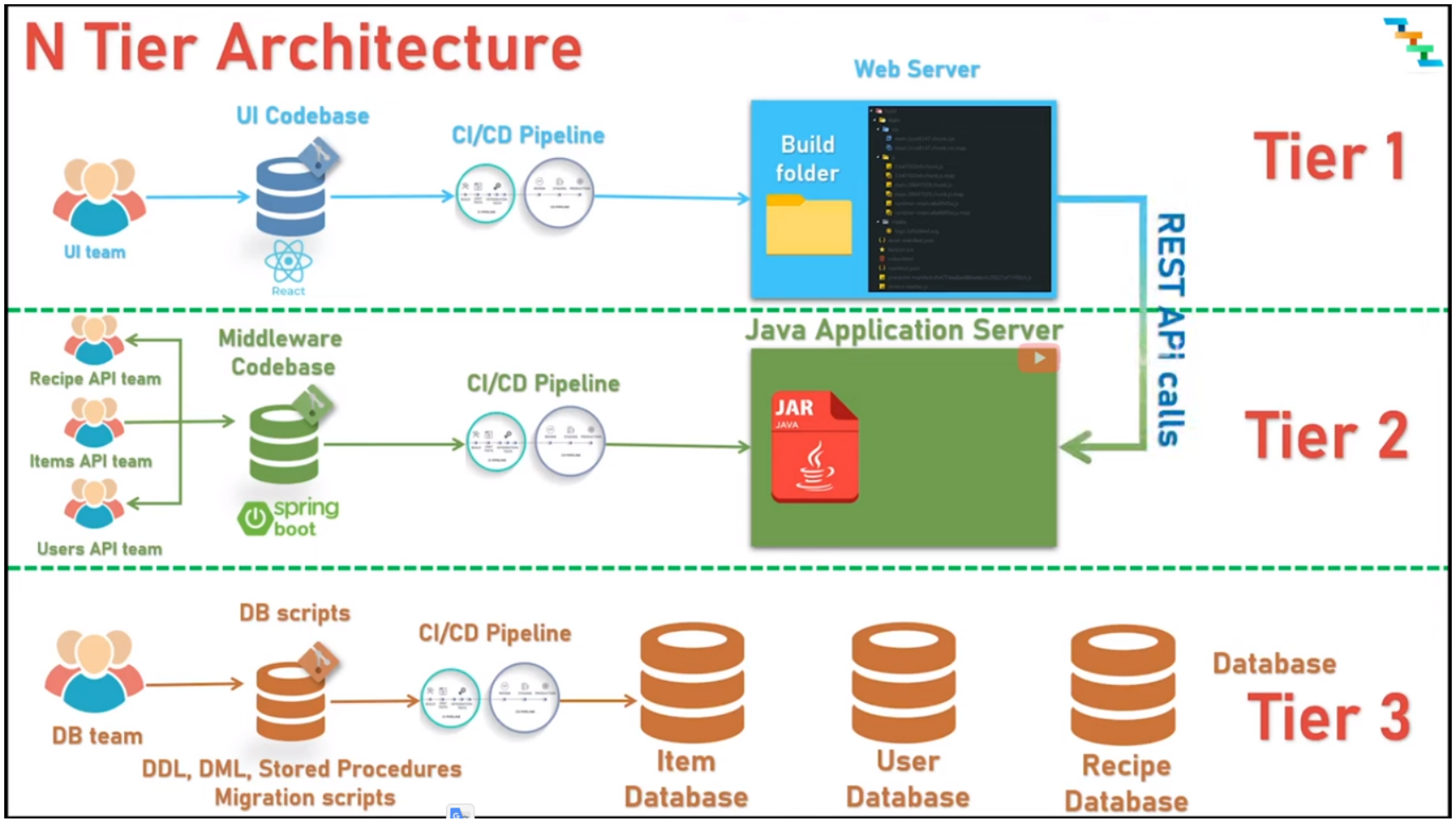


Figura: Arquitetura em N camadas (N-tier).

Principais Arquiteturas de Software

Arquitetura em 2 camadas - Cliente-Servidor

A arquitetura de software em duas camadas, também conhecida como arquitetura cliente-servidor, é um modelo simples no qual a lógica de aplicação é dividida em duas partes principais: a camada de apresentação (cliente) e a camada de dados (servidor). Essa arquitetura é bastante direta e é frequentemente utilizada em aplicações pequenas e simples.

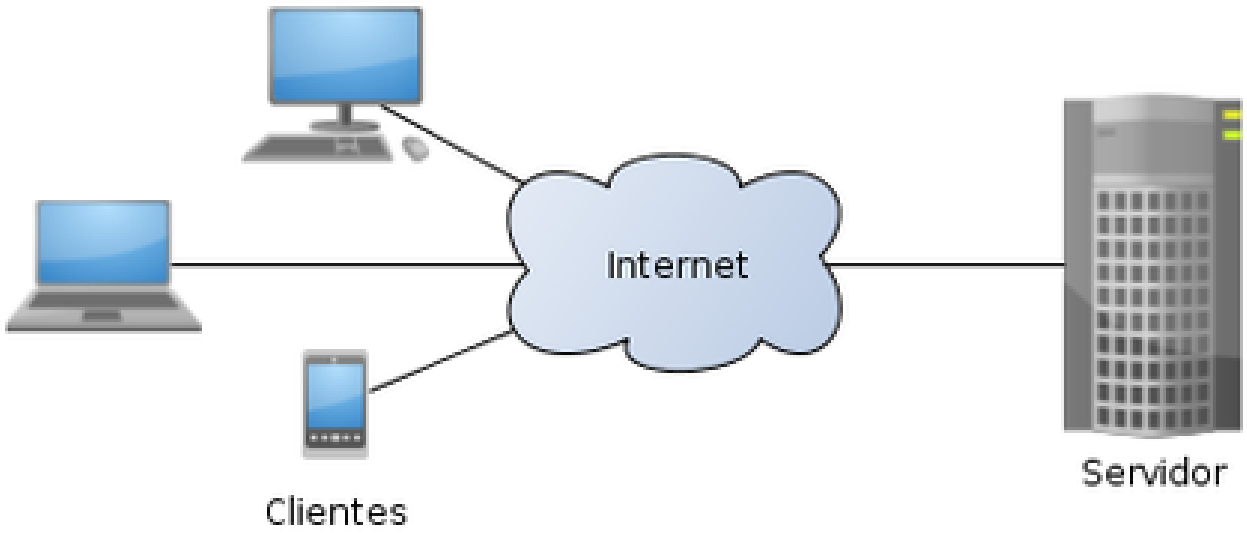


Figura: Arquitetura Cliente-Servidor.

Modelos Arquiteturais

Microserviços

Microservices

Principais Arquiteturas de Software

Microserviços - Definição

- A arquitetura de microsserviços é uma abordagem arquitetônica e organizacional do desenvolvimento de software na qual o software consiste em pequenos serviços independentes.
- É similar à arquitetura orientada a serviços (SOA), mas com algumas diferenças importantes, tais como, por exemplo, o tamanho dos serviços e a forma como eles se comunicam.
- Esses serviços são mantidos por pequenas equipes autossuficientes.
- Cada serviço é desenvolvido, implantado e gerenciado de forma independente.
- Cada serviço é responsável por uma única funcionalidade.
- Os serviços se comunicam entre si através de APIs bem definidas.

Vantagens

- **Separação de Responsabilidades**: Tudo é desenvolvido através de pequenas unidades de código e publicado em processos de deploy automatizados e independentes.
- **Tecnologias**: Essa abordagem permite que cada serviço seja desenvolvido usando a *stack* mais adequadas para o problema que está sendo resolvido.
- **Aplicabilidade**: em aplicações de grande porte e complexas que precisam ser escaladas rapidamente e com equipes distribuídas.

Desvantagens

- **Complexidade**: A complexidade de uma arquitetura de microsserviços é maior do que a de uma arquitetura monolítica, pois existem mais componentes para gerenciar.
- **Comunicação**: A comunicação entre os serviços pode resultar em algum overhead, especialmente em sistemas distribuídos, o que pode impactar o desempenho.
- **Testes**: Os testes de integração são mais complexos, pois envolvem a comunicação entre os serviços.
- **Gerenciamento**: O gerenciamento de uma arquitetura de microsserviços (governança) é mais complexo, pois existem mais componentes para gerenciar.

Principais Arquiteturas de Software

Microserviços - Representação

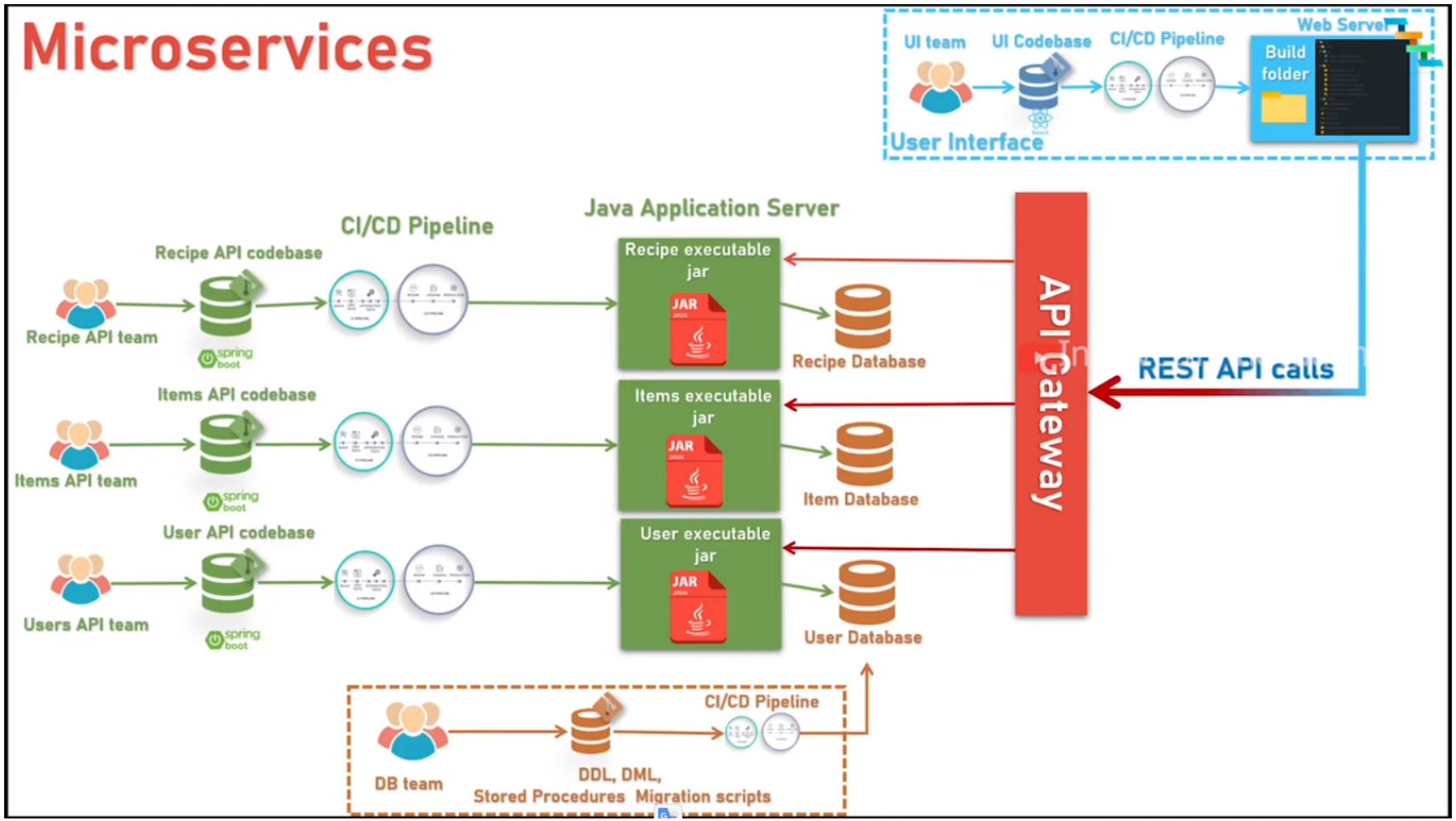


Figura: Arquitetura de Microsserviços.

Modelos Arquiteturais

Comparativos

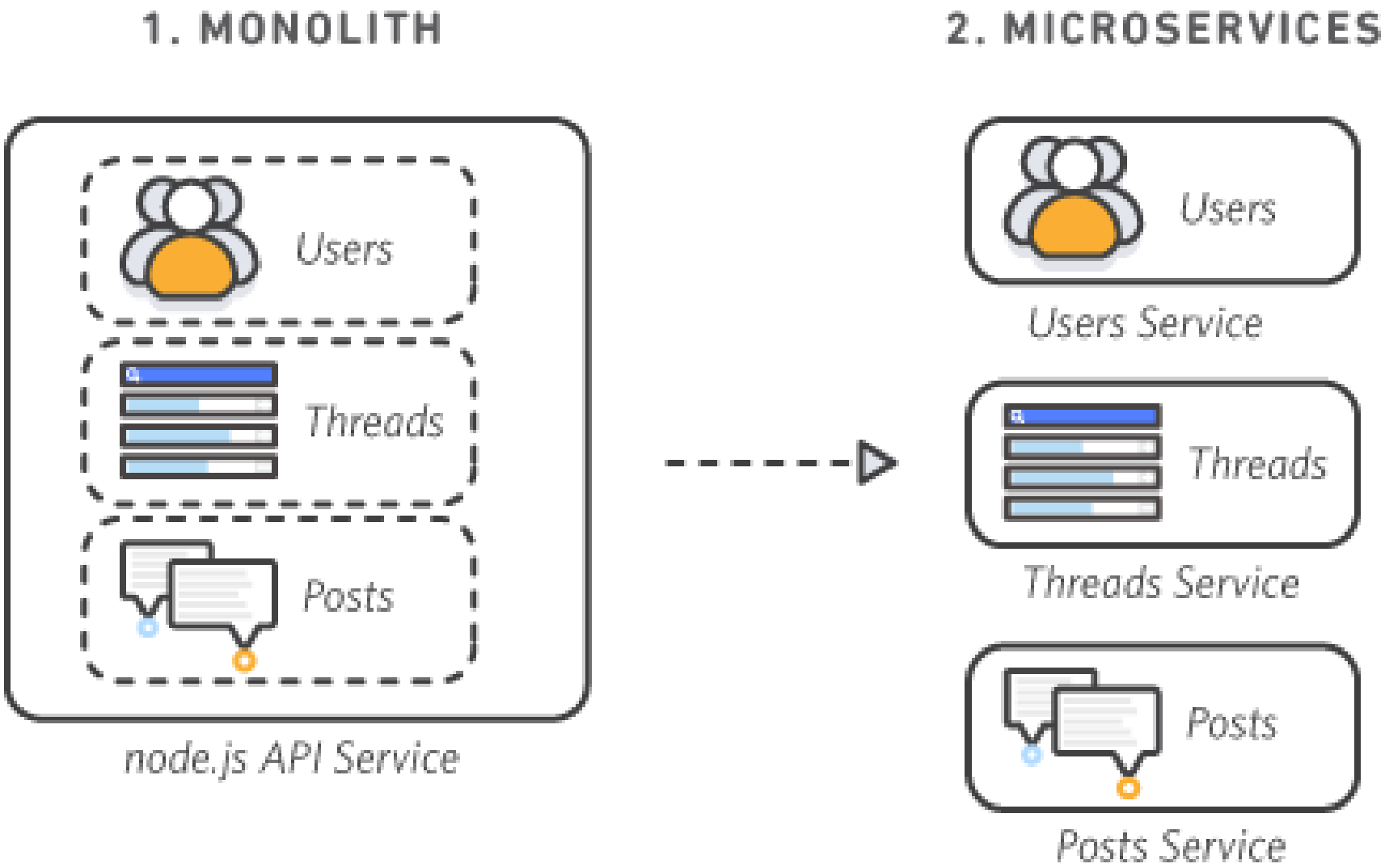


Figura: Monolito vs. Microsserviços.

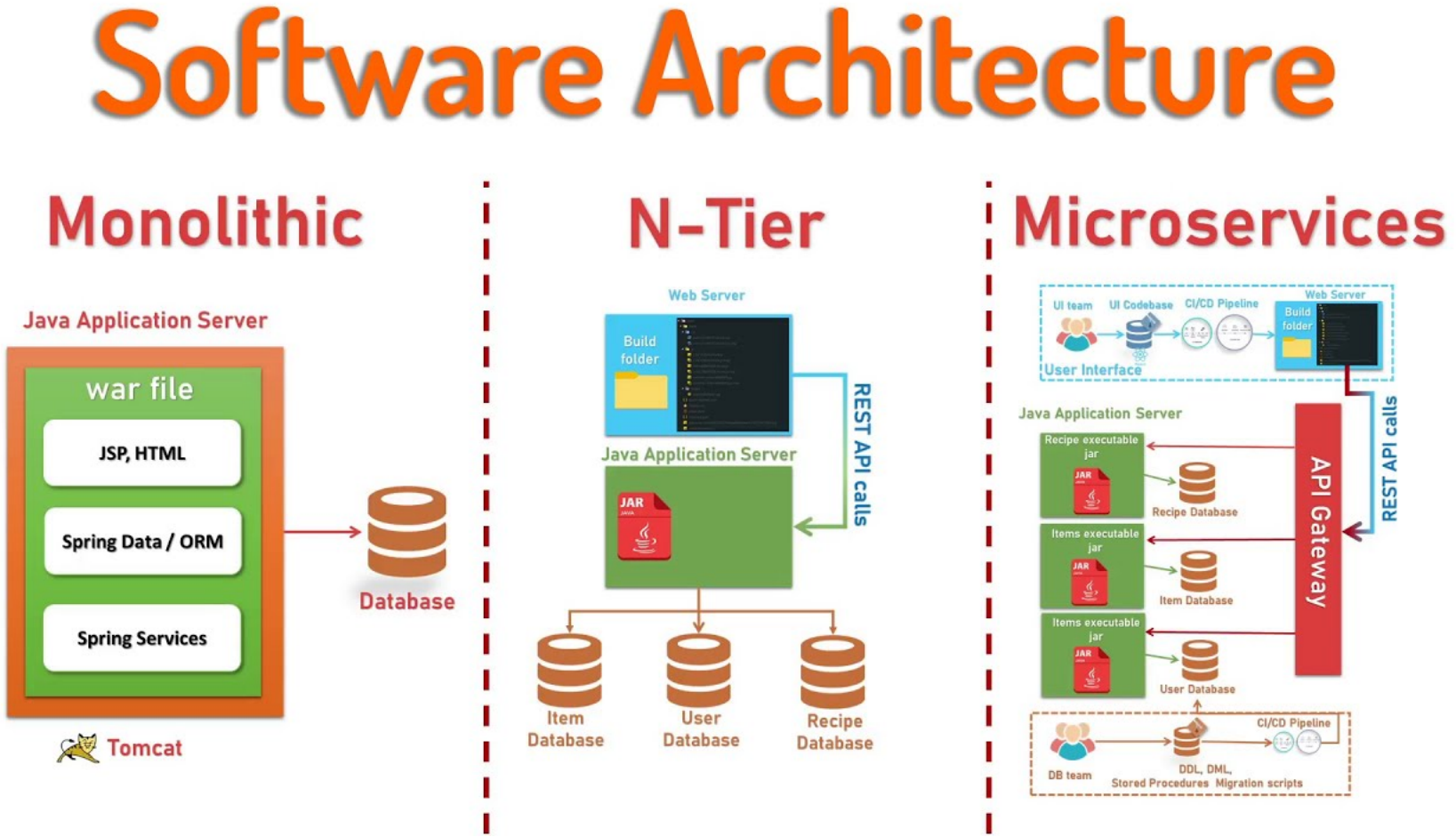


Figura: Monolito, N Camares e Microsserviços.

Sistema de Comércio Eletrônico - Módulo de Recomendações:

Monolito - Exemplo

Um componente que fornece recomendações de produtos com base no histórico de compras, comportamentos de navegação e preferências do usuário. Todos os módulos serão construídos na mesma base de código e implantados como uma única unidade.

N Camadas - Exemplo

Será organizado como parte da camada de lógica de negócios (Backend).

Microserviços - Exemplo

Um microserviço que fornece recomendações personalizadas aos clientes com base em seu histórico de compras, preferências e comportamentos de navegação. Será um serviço separado e independente.

Modelos Arquiteturais

Outras Arquiteturas

Arquitetura Orientada a Serviços (SOA)

A arquitetura de microsserviços é uma evolução do estilo de arquitetura SOA. Embora cada serviço de SOA seja um recurso de negócios completo, cada microsserviço é um componente de software muito menor, especializado em apenas uma única tarefa. Os microsserviços abordam as deficiências da SOA para tornar o software mais compatível com ambientes corporativos modernos baseados na nuvem (AWS 2024a).

Exemplo

Sistema de Comércio Eletrônico - Serviço de Pagamento: Responsável pelo processamento de transações financeiras, integração com gateways de pagamento e validação de transações.

Arquitetura Orientada a Eventos

Uma arquitetura orientada por eventos usa eventos para acionamento e comunicação entre serviços desacoplados e é comum em aplicações modernas criadas com microsserviços (AWS 2024b).

Exemplo

Sistema de Comércio Eletrônico - Pedido Realizado: quando um cliente faz um pedido, um evento é gerado e enviado para o serviço de processamento de pedidos, que pode então iniciar o processamento do pedido.

Serverless

É um modelo de desenvolvimento nativo em nuvem para criação e execução de aplicações sem o gerenciamento de servidores. Os servidores ainda são usados nesse modelo, mas eles são abstraídos do desenvolvimento de aplicações (Hat 2023).

Exemplo

Serviço de Armazenamento (Storage): Os usuários fazem upload de imagens para um serviço de armazenamento na nuvem, como o Amazon S3. A função serverless terá a lógica para processar a imagem e gerar thumbnails em diferentes tamanhos.

Material Complementar

Vídeos, Podcasts, Livros, etc

- **Aplicação Monolítica // Dicionário do Programador.**
  Canal **Código Fonte TV**.
- **Arquitetura de Software: Monolítica x SOA x Microserviços.**
  Canal **Marcos Dósea**.
- **Microservices // Dicionário do Programador.**
  Canal **Código Fonte TV**.
- **Microservices na prática.**
  Canal **Full Cycle**.
- **Microservices na prática.**
  Podcast Hipsters Ponto Tech **Monolitos**.

Recapitulando

QUIZ

Vamos praticar um pouco o que vimos até agora?

QUIZ - Desenvolvimento Web - Arquitetura de Software

Referências

[1] AWS, “Qual é a diferença entre SOA e microsserviços?” (), endereço: https://aws.amazon.com/pt/compare/the-difference-between-soa-microservices (acesso em 06/01/2024).

[2] AWS, “Pattern: Monolithic Architecture,” (), endereço: O%20que%20%C3%A9%20uma%20arquitetura%20orientada%20por%20eventos? (acesso em 06/01/2024).

[3] R. Hat, “O que é serverless?” (7 de ago. de 2023), endereço: https://www.redhat.com/pt-br/topics/cloud-native-apps/what-is-serverless (acesso em 06/01/2024).