PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Disciplina: Projeto de Software
Professor: João Paulo Carneiro Aramuni
Aluno: João Pedro Santana Marques

Capítulo 7: Arquitetura de Software

O capítulo 7 do livro apresenta um estudo sobre arquitetura de software, abordando sua definição, importância e diferentes padrões arquiteturais utilizados no desenvolvimento de sistemas.

A arquitetura de software pode ser definida de diferentes formas, sendo uma das mais comuns aquela que considera a arquitetura como o projeto em um nível mais alto, focando na organização de pacotes, componentes, módulos, subsistemas ou serviços, e não apenas em classes individuais. Outra definição considera a arquitetura de software como um conjunto de decisões críticas de projeto, que dificilmente podem ser alteradas depois de implementadas, incluindo a escolha da linguagem de programação e do banco de dados.

Os padrões arquiteturais são modelos de organização para sistemas de software e definem a relação entre seus módulos. No capítulo, serão abordados os seguintes padrões: Arquitetura em Camadas, Model-View-Controller (MVC), Microsserviços, Arquitetura Orientada a Mensagens e Publish/Subscribe. Ainda há diferença entre padrões e estilos arquiteturais, sendo os padrões voltados para soluções específicas e os estilos para uma organização geral dos módulos.

A **arquitetura em camadas** é um dos padrões arquiteturais mais utilizados desde as décadas de 1960 e 1970. Nesse modelo, as classes são organizadas em camadas hierárquicas, onde cada camada pode acessar apenas os serviços da camada imediatamente abaixo. Esse padrão é amplamente aplicado, especialmente em protocolos de rede (como HTTP, TCP e IP). As principais vantagens da arquitetura em camadas são a modularização e a organização das dependências, facilitando a manutenção do sistema. Permite também a substituição de camadas sem afetar o restante do sistema e a reutilização de componentes em diferentes contextos.

A **arquitetura em três camadas** é amplamente utilizada em sistemas de informação corporativos, separando a aplicação em três camadas distintas. Normalmente, essa arquitetura é distribuída, com a camada de interface no cliente, a camada de negócios no servidor de aplicação e o banco de dados armazenando as informações:

Interface com o Usuário (Camada de Apresentação): Responsável pela interação com o usuário, incluindo exibição de informações e captura de entradas. Pode ser uma aplicação desktop ou web, como, por exemplo, um sistema acadêmico para o lançamento de notas dos alunos.

Lógica de Negócio (Camada de Aplicação): Implementa as regras de negócio do sistema. No exemplo acadêmico, validações como a regra de que as notas devem estar entre 0 e o valor máximo da avaliação. Também pode envolver ações como o envio de e-mails aos alunos.

Banco de Dados: Armazena os dados do sistema. No caso do sistema acadêmico, as notas dos alunos são salvas após a validação na camada de lógica.

O padrão arquitetural **MVC (Model-View-Controller)** organiza o sistema em três componentes principais. Uma vantagem da arquitetura é a separação das preocupações, permitindo que diferentes desenvolvedores trabalhem nas diversas partes do sistema e facilitando a manutenção.

Visão (View): Responsável pela apresentação da interface gráfica, incluindo janelas, botões e menus. A visão é encarregada de exibir informações ao usuário e capturar suas interações.

Controladoras (Controller): Interpreta eventos de entrada (como cliques de mouse e pressionamentos de teclas) e solicita modificações no estado do modelo ou da visão. Por exemplo, ao clicar em um botão, a controladora atualiza a visão ou o modelo com base na ação do usuário.

Modelo (Model): Contém os dados e a lógica de negócios do sistema, sem depender diretamente da visão ou controladora. Ele representa o domínio do sistema e manipula informações sem se preocupar com como elas são apresentadas.

MVC foi criado para interfaces gráficas e organiza o código em três partes: modelo, visão e controladora. Já a arquitetura de três camadas é uma organização mais ampla de sistemas distribuídos, com uma camada de interface (visão), uma camada de aplicação (lógica de negócios) e uma camada de banco de dados.

As **Single Page Applications (SPAs)** são um tipo moderno de aplicação web que melhora a experiência do usuário em comparação com as aplicações web tradicionais. Nas SPAs, o código da aplicação (HTML, CSS e JavaScript) é carregado inicialmente no navegador. A partir daí, a aplicação funciona de maneira similar a um programa de desktop, sem a necessidade de recarregar a página toda vez que o usuário interage. Essas características tornam as SPAs altamente interativas e mais responsivas. No entanto, a SPA ainda depende de comunicação com o servidor para realizar funções como recuperar dados ou atualizar informações.

Os **microsserviços** surgiram como uma solução arquitetural para os gargalos enfrentados pelas arquiteturas monolíticas, especialmente em contextos de desenvolvimento ágil. Embora o desenvolvimento ágil possibilite iterações rápidas e entregas frequentes de novas versões, as arquiteturas monolíticas criam dificuldades na implementação de mudanças rápidas devido ao seu modelo de execução, onde todos os módulos compartilham o mesmo espaço de memória e são executados como um único processo. A popularização dos microsserviços é facilitada pelo uso de plataformas de computação em nuvem, que permitem que os serviços sejam hospedados em máquinas virtuais de maneira escalável.

Em vez de um monolito com todos os módulos integrados em um único processo, os microsserviços dividem o sistema em serviços independentes. Além disso, os microsserviços se comunicam exclusivamente através de interfaces públicas. Uma das principais vantagens dos microsserviços é a escalabilidade mais granular: com microsserviços, é possível replicar apenas os serviços que necessitam de maior capacidade, otimizando recursos. Como os microsserviços são autônomos, falhas em um serviço não comprometem o sistema inteiro. Essa independência também permite que cada um tenha seu próprio ciclo de desenvolvimento e releases, facilitando atualizações mais rápidas sem afetar o funcionamento dos outros serviços.

Em uma arquitetura de microsserviços, cada serviço deve ser autônomo em termos de gerenciamento de dados, ou seja, cada microsserviço deve ser responsável pelos dados de que precisa para fornecer seu serviço, sem depender de um banco de dados compartilhado.

Apesar de todas as vantagens dos microsserviços, esta arquitetura apresenta uma complexidade adicional em comparação com a arquitetura monolítica. Isso ocorre porque microsserviços são sistemas distribuídos, introduzindo desafios de complexidade, latência e transações distribuídas.

A **Arquitetura Orientada a Mensagens** é um modelo em que a comunicação entre clientes e servidores é mediada por um terceiro serviço responsável por gerenciar uma fila de mensagens.

Clientes: Atuando como produtores de informações, os clientes inserem mensagens na fila de mensagens. Essas mensagens contêm dados relevantes que o servidor precisa processar.

Servidores: Atuando como consumidores de mensagens, os servidores retiram as mensagens da fila e processam as informações que elas contêm.

Fila de Mensagens: A fila segue o princípio FIFO (First In, First Out), ou seja, a primeira mensagem a ser inserida na fila é a primeira a ser processada pelo servidor.

Com o uso de filas de mensagens, a comunicação entre os clientes e servidores torna-se assíncrona. O uso de filas de mensagens promove dois tipos principais de desacoplamento entre os componentes: desacoplamento no espaço - os clientes e os servidores não precisam se conhecer diretamente; desacoplamento no tempo - clientes e servidores não precisam estar disponíveis ao mesmo tempo.

Em uma arquitetura **publish/subscribe**, os componentes são organizados como publicadores (publishers) e assinantes (subscribers) de eventos. O evento é uma mensagem que é publicada por um publicador em um serviço central de publish/subscribe. Semelhante às filas de mensagens, arquiteturas publish/subscribe oferecem desacoplamento no espaço e no tempo. No entanto, existem duas diferenças principais entre filas de mensagens e publish/subscribe: comunicação 1 para N em publish/subscribe vs 1 para 1 em filas de mensagens; notificações assíncronas no publish/subscribe e pull de mensagens na fila de mensagens.

O conceito de Big Ball of Mud (grande bola de lama) descreve uma arquitetura de software onde não há uma organização clara ou estrutura definida entre os módulos do sistema. Em vez de existir uma arquitetura bem definida, há uma explosão de dependências entre os componentes.

Aplicação Prática de Microsserviços em um E-commerce:
Assim como na aplicação prática do capítulo 5, o sistema pode ser dividido em vários microsserviços independentes:
Serviço de catálogo de produtos (responsável por exibir os produtos).
Serviço de pagamento (gerenciamento de pagamentos e transações).
Serviço de recomendação de produtos (sugestões personalizadas para os clientes).
Serviço de gerenciamento de carrinho de compras.