GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

Disciplina: Desenvolvimento de Serviços em Nuvem com Java
Aluno: Matheus Evangelista

TP1 <*Desenvolvimento de Serviços em Nuvem com Java*>

Monte uma tabela comparativa com 3 implementações de arquiteturas de sistemas (linhas), seus pontos positivos (primeira coluna) e pontos negativos (segunda coluna)

Model-view-controller (MVC)

- Vantagens:
  - Facilita o reaproveitamento de código;
  - Facilidade na manutenção e adição de recursos;
  - Facilidade em manter o seu código sempre limpo;
- Desvantagens:
  - Requer uma quantidade maior tempo para analisar e modelar o sistema
  - Requer pessoal especializado
  - Não é aconselhável para pequenas aplicações

Model-view-viewModel (MVVM)

- Vantagens
  - Extensibilidade
  - Testabilidade
  - Manutenção
- Desvantagens
  - O ViewModel pode conter o estado de exibição próximo ao estado de negócios da apresentação podendo ficar bem confuso
  - No modelo MVVM, escrever um teste para um aplicativo é uma tarefa bastante difícil comparativamente com o MVP.
  - Também em alguns casos, o código pode ser visto na forma de XML, o que pode confundir o desenvolvedor e dificultar o processo de depuração.

Model-view-presenter (MVP)

- Vantagens:
  - Facilitar a depuração para aplicativos
  - Reutilização de código
  - Melhor separação de responsabilidades
- Desvantagens
  - Você precisa de várias implementações no código do fragmento/activity ou controller/view para definir e obter a entrada do usuário
  - O tamanho do código no Model View Presenter (MVP) é excessivo, o que o torna um pouco mais complexo.
  - Na arquitetura MVP, um grande número de interfaces é usado para interação entre as três camadas. Como cada interface cobre apenas uma pequena fração das interações, isso leva a uma grande variedade de métodos a serem implementados

Quais são as camadas de uma aplicação típica? Para que servem?

As camadas típicas de uma aplicação são Model-View-Controller. Isso possibilita a divisão do projeto em camadas muito bem definidas. Cada uma delas, o Model, o Controller e a View, executa o que lhe é definido e nada mais do que isso.
A utilização do padrão MVC traz como benefício o isolamento das regras de negócios da lógica de apresentação, que é a interface com o usuário. Isto possibilita a existência de várias interfaces com o usuário que podem ser modificadas sem a necessidade de alterar as regras de negócios, proporcionando muito mais flexibilidade e oportunidades de reuso das classes.

O que é um Monólito de Aplicativo e quais seus pontos positivos e negativos?

Monólito é utilizado para definir a arquitetura de alguns sistemas, refere-se a forma de desenvolver um sistema, programa ou aplicação onde todas as funcionalidades e códigos estejam em um único processo. Essas diversas funcionalidades estão em um mesmo código fonte e em sua execução compartilham recursos da mesma máquina, seja processamento, memória, bancos de dados e arquivos.
Como o sistema está inteiro em um único bloco, seu desenvolvimento é mais ágil, se comparado com outras arquiteturas, sendo possível desenvolver uma aplicação em menos tempo e com menor complexidade inicial, reparem na palavra inicial.
A medida em que uma aplicação é descrita como monolítica depende muito de sua perspectiva. Talvez uma aplicação que não está orientada a serviços pode ser considerada monolítica.

O que é Microsserviço? Quais são as suas principais características?

Microsserviço refere-se aos serviços individuais em uma arquitetura de microsserviços. Por sua vez, essa arquitetura é um estilo moderno de arquitetura para web services. São um exemplo da “Arquitetura Orientada a Serviços”, ou SOA. O SOA tornou-se uma alternativa popular à abordagem tradicional de construção de aplicações singulares e autossuficientes, as quais chamamos de monólitos. Quando se fala em microsserviços estamos nos referindo a uma funcionalidade que pode ser dividida em fragmentos menores.

Características

- Autônomos
  Cada serviço do componente de uma arquitetura de microsserviços pode ser desenvolvido, implantado, operado e escalado sem afetar o funcionamento de outros serviços. Os serviços não precisam compartilhar nenhum código ou implementação com os outros serviços. Todas as comunicações entre componentes individuais ocorrem por meio de APIs bem definidas.

- Especializados
  Cada serviço é projetado para ter um conjunto de recursos e é dedicado à solução de um problema específico. Se os desenvolvedores acrescentarem mais código a um serviço ao longo do tempo, aumentando sua complexidade, ele poderá ser dividido em serviços menores.