# Desafio VR Desenvolvimento

1. Recebemos um código desenvolvido por terceiros de um sistema que possui alto volume de lógica de negócio e apresenta as seguintes características:

- O sistema recebe requisições REST, está dividido em camadas e possui classes de domínio;

- O controller recebe a requisição e está com toda lógica de negócio. Monta e repassa o domínio para a aplicação;

- A aplicação tem a responsabilidade de repassar o objeto pronto para o repositório;

- O repositório apenas persiste os objetos mapeados do hibernate através de spring data;

- O domínio apenas faz o mapeamento para o BD;

- Nenhum teste unitário foi escrito.

- O sistema está escrito em java para rodar como spring boot.

Apresente observações/problemas sobre essa solução.

Comente qual(is) a(s) sua(s) estratégia(s) para melhorar este sistema em termos de qualidade e manutenção. Justifique suas decisões.

Resposta:

1º Acoplamento elevado: o acoplamento entre as camadas é elevado, o que torna o código difícil de ser modificado sem causar efeitos colaterais.

2º Falta de testes unitários: a ausência de testes unitários torna a manutenção do código mais difícil, uma vez que pequenas mudanças podem causar falhas no sistema inteiro.

3º Baixa coesão: a lógica de negócio está concentrada no controller, o que leva a uma baixa coesão e torna o código difícil de ser entendido e manutenido.

Para melhorar a qualidade e a manutenção do sistema, as seguintes estratégias podem ser adotadas:

1º Reduzir o acoplamento: separar a lógica de negócio das demais camadas, criando classes específicas para esse propósito, melhorando a coesão e facilitando a manutenção.

2º Adicionar testes unitários: escrever testes unitários para garantir que as funcionalidades do sistema estão corretas e que futuras modificações não causam falhas.

3º Refatorar o código: reorganizar o código de forma a torná-lo mais claro e fácil de ser entendido, facilitando a manutenção e a implementação de novas funcionalidades.

Essas decisões foram justificadas com o objetivo de melhorar a qualidade e a manutenibilidade do sistema, tornando-o mais robusto e escalável. Além disso, com a separação clara das responsabilidades entre as camadas e o uso de testes unitários, o risco de erros é minimizado e a equipe de desenvolvimento ganha confiança na evolução do sistema.

1. Descreva quais são as principais limitações ao se adotar servidores de aplicação em uma arquitetura orientada a microsserviços.

Resposta:

As principais limitações ao se adotar servidores de aplicação em uma arquitetura orientada a microsserviços incluem:

1º Complexidade: A divisão em vários serviços pode aumentar a complexidade da arquitetura e exigir uma gestão mais rigorosa dos componentes.

2º Latência: Comunicação entre serviços pode resultar em latência adicional e aumentar a complexidade do gerenciamento de failover.

3º Integração: Integrar serviços distintos pode ser desafiador e exigir soluções adicionais para garantir a consistência e integridade dos dados.

4º Escalabilidade: Escalar serviços individualmente pode ser mais complexo do que escalar uma aplicação monolítica, pois pode exigir ajustes em vários componentes.

5º Segurança: Proteger cada serviço individualmente pode ser mais complexo do que proteger uma aplicação monolítica, pois pode exigir soluções adicionais para garantir a segurança dos dados.

1. Atualmente, diversas aplicações escritas em Java estão deixando de serem desenvolvidas para rodarem em servidores (JBoss, Tomcat), adotando ferramentas que disponibilizam um servidor embutido na própria ferramenta. Quais são os principais desafios ao se tomar uma decisão dessas? Justifique sua resposta.

1º Aumento no trabalho para realizar testes.

2ºPerda de performance momentânea (cold-start)

3º Dificuldade no monitorando e depuração, por não ter acesso a infraestrutura onde os serviços são executados.

4º Complexidade da arquitetura, sem padrões estabelecidos.

5º Ao usar a mesma estrutura (serverless) a aplicação fica engessada em um mesmo fornecedor, tendo limites de recursos a serem usados.

6º Não é possível o compartilhamento de recursos entre as aplicações criadas, precisando assim modificar cada uma de forma isolada.

1. Teste prático (em anexo)