# 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**:

Falta de testes unitários, o que torna o sistema propenso a erros e difícil de manter. Toda a lógica de negócio está concentrada no controller, o que pode tornar o código difícil regras de front-end e regras de negócio.

A responsabilidade da aplicação é limitada a apenas repassar o objeto para o repositório, o que pode resultar em código duplicado.

As estratégias para melhorar o sistema é adicionar testes unitários para garantir a confiabilidade e facilidade de manutenção;

refatorar o código para separar a lógica de negócio do controller e mover para classes de serviço ou de aplicação.

adicionar validações e tratamento de erros nas camadas de domínio, aplicação e controller;

utilizar técnicas de injeção de dependência para tornar o código mais fácil de testar e manter;

**Justificativa**: Estas estratégias ajudam a tornar o código mais organizado, fácil de entender e manter, além de aumentar a qualidade do sistema. A adição de testes unitários garante a confiabilidade do sistema, a separação da lógica de negócio torna o código mais claro e fácil de manter, e a injeção de dependência torna o código mais fácil de testar e manter.

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

**Resposta**:

1. Aumento na complexidade de gerenciamento e orquestração de diversos componentes que compõe a solução;
2. Difícil depuração de erro, o que agrava se estiver usando um serviço de mensageria;
3. Dificuldade na manutenção de testes de integração automatizados;
4. Complexidade no deploy devida a cadeia de dependências.
5. 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.

**Resposta**: Já algum tempo este modelo é adotado, e o tomcat é o servidor padrão do spring-boot. O desafio está em mudança de paradigma em se deixar uma estrutura voltada para servidores para uma estrutura voltada em conteinerização e nuvem. Contudo, tal estrutura fornece uma maior escalabilidade horizontal dos serviços.

1. Teste prático (em anexo)