# 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**

Por mais confusa que a aplicação esteja, ainda tem solução. Mudaria um pouco a arquitetura. Criaria um pacote service e colocaria toda a regra de negócios lá, tirando essa responsabilidade do controller. O Controller apenas chamaria um service com o serviço que deseja executar.

Outro ponto, ainda no controller, seria substituir a classe de domínio por classes de DTO. Deixando a classe de domínio separada da camada mais externa, sendo acessada apenas pelo service que a comanda.

Faria um mapper que faria esse bind entre o controller e o service, deixando as camadas externas sem acesso pelas camadas internas, só pelo service.

Como é um sistema com muita regra de negócio, acredito que tenha muitos domínios também. Nesse caso eu faria todos os scripts de criação e manutenção das tabelas para serem rodadas pelo flyway, tirando essa responsabilidade de criação do hibernate. Possibilitando uma melhor manutenção no futuro e um controle melhor de quem alterou o que.

Faria testes unitários.

Essa seria a solução que eu daria para manter o sistema como está. Mas no futuro poderiamos quebrar esse sistema em vários outros microserviços, o deixando menos complexo.

Usando o Kotlin também ajudaria com a verbosidade, já que ele tem 100% de compatibilidade com java, só que tem menos linhas de código quando bem escrito.

Por ter muita regra, imagino um sistema pesado, usando o Spring Native ajudaria a diminuir esse tamanho.

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

**Resposta:**

O principal, ao meu ver, é o tamanho. Os servidores de aplicação implementam o Java EE na sua totaliadade, ou seja, tem muita coisa lá que você não vai precisar, então só de rodar um jboss da vida, a máquina precisa dispor de uma quantidade considerável de memória.

Outro ponto relacionado a memória, é a dificuldade de escalar uma aplicação assim, sendo mais caro que escalar um aplicação sem servidor de aplicação.

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.

Vai depender do que você espera da sua aplicação. Se for microserviço, você quer ter a liberdade de rodar ele em qualquer lugar. Qualquuer cloud que rode um war já te atende, seus custos de escala também o são. Já com servidores de aplicação mais pesados não é assim tão simples. Pois eles são naturalmente mais pesados, por ter funcionalidades disponíveis que você não necessariamente vai usar.

1. Teste prático (em anexo)