# 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.

Para melhorar a manutenibilidade do código as minhas sugestões seriam as seguintes:

1. Antes de fazer as devidas refatorações, criar testes de integração (Não foi explicado se possui) para garantir a funcionalidade dos endpoints REST da aplicação.
2. Mover a lógica de negócios para as classes de domíno da aplicação, para facilitar o reuso em outros controlers/endpoints, assim como facilitar a escrita de testes.
3. Criar testes unitários para validar a funcionalidade das classes de domínio que foram criadas.
4. Remover toda e qualquer referência de banco de dados ou persistência das classe de domínio da aplicação, migrando tudo para uma camada de persistência (Que contém as entidades e repositórios) e utlizando DTOs para transferir os dados.
5. Os repositório ficam responsáveis por persistir as entidades, assim como consultas destas entidades.
6. Criar testes unitários para validar a funcionalidades dos repositórios e entidades.

Fazendo estas modificações o sistema ficará mais simples de manter e modificações poderão ser feitas com mais confiança, tendo em vista as garantias que os testes trazem.

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

Como uma arquitetura de microsserviços consiste em uma coleção de pequenos serviços autônomos, quer dizer, cada serviço é independente e deve implementar uma única funcionalidade comercial em contexto limitado, segundo *Martin* Fowler , especialista em arquitetura de software, “**não considere migrar para microsserviços a menos que você já tenha um sistema que seja muito complexo para gerenciar como um monólito**”. Como as aplicações rodam em múltiplos servidores, nada pode ser compartilhado entre os microserviços, sendo assim, deparamos o comportamento de um monólito. Dito isso, temos a complexidade no qual o sistema formado por partes autônomas e especializadas forma um todo bem complexo, distribuído; governança, com vários componentes completamente distintos trabalhando juntos em um único aplicativo, a governança pode deixar a desejar, não esqueça que pode demorar na inicialização, no qual ficará inacessível o serviço, temos que levar em conta o tamanho destes,

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.

As configurações de servidor de aplicação passam a ser individuais, sendo necessário para cada aplicação configurar logs, monitoramento e segurança e atualizações. Não é possível compartilhar recursos entre aplicações, tendo que modificar a aplicação para que ela se torne stateless. E sempre que for atualizar a aplicação deve se levar em conta que será necessário o restart do servidor, independete da mudança.

1. Teste prático (em anexo)