# 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.

R= 1 - “**O controller recebe a requisição e está com toda lógica de negócio**” De acordo com alguns padrões de projetos, um exemplo é o MVC. O controller não deveria ter a logica de negocio , o ideal seria ter um camada chama de “**Service**” onde ela ficaria responsável por fazer essa logica de negocio.

2 -  **O repositório apenas persiste os objetos mapeados do hibernate através de spring data.**  O Spring Data além de fazer a persistência ele é um poderoso framework para fazer consultas como projections, specfications. Além de ter abstração das consultas como findBy+”determinado campo”. Existem algum outros exemplos na documentação.

3 - **Nenhum teste unitário foi escrito.** O problema de não ter nenhum teste unitário, seria que você não consegue garantir que determinada funcionalidade se mantenha constantemente correta possibilitando a regressão do código.

Exemplo: Você cria um funcionalidade e o teste ocorre de forma manual e funciona corretamente, só que em determinado momento de implementação outra pessoa modifica essa mesma funcionalidade e assim pode impactar em algum trecho do sistema. Caso estivesse teste unitários cobrindo o fluxo da implementação e outra pessoa modificasse parte dessa implementação, o teste unitário iria garantir que o fluxo antigo iria se manter.

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

R= Ao optar por uma arquitetura de microsserviços, existem alguns pontos a serem definidas, exemplo: **Granularidade** dos serviços, os módulos terão **base de dados independentes ou um única base de dados**? A comunicação será Assíncrona ou Síncrona? Além das preocupações transversais como observabilidade e segurança.

Nas preocupações transversais poderemos ter um tipo de entrada Única para arquitetura de microservices (**API Gateway Pattern**) para clientes externos, outra preocupação transversal é que como eles vão se comunicar internamente via rest utilizando Ips e endereços dinâmicos? como vou fazer esse balanceamento de carga? qual endereço certo para enviar a requisição? E uma solução para essas perguntas seria implementar isso de forma dinâmica para saber qual o endereço certo de determinado serviço para ter a comunicação (**Service Registry Discovery Pattern**) - **vai monitoriar registrar e vai ter o controler dos microservices e toda vez que ele recorre de uma comunicação interna ele vai acionar o (service registry) e ele vai saber com quem se comunicar.**

Além dessas preocupações transversais temos também preocupações a falhas (**Circuit Breaker Pattern**) como vai ser tratada falhas dos microservices.

Ao se optar por uma arquitetura de microsserviços temos uma complexa, e com um custo de manutenção bem maior.

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.

R= O Framework SpringBoot possui em servidor Tomcat embutido rodando ao criar o projeto, essa pratica é um grande ventagem, pois antes do frameworks, teríamos que criar alguns arquivos de configuração para configurar o servidor. Assim o springBoot já fica responsável por toda essa configuração, dando agilidade para a implementação das regras de negocio que deverá ser aplicada no projeto.

1. Teste prático (em anexo)