# 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: Pelo que entendi os controladores recebem como parâmetro a própria classe de domínio, onde, não necessariamente se faz interessante a exposição ou requisição de todos os atributos da classe para transição nas camadas MVC. Se for o caso seria interessante a criação de DTO’s (Data Transfer Objects) que possuem a responsabilidade de transitar apenas informações pertinentes a request em questão, o DTO através de uma camada de serviço processa as informações mapeando os atributos para as classes de domínio para aí sim persistir na base de dados via JPA(Java Persistence API), da mesma forma que pós processamento caso necessário um body como resposta da request, como em um POST por exemplo, além de um HTTP Response 200 seriam interessantes os mapeamentos dos dados do domínio para um VO(View Object), visando mais uma vez privar atributos que não são pertinentes a uma exposição desnecessária. Quanto ao sistema não possuir testes unitários implementados, além da não garantia da qualidade dos serviços consumidos por cada API, em manutenções futuras perdemos a rastreabilidade de funcionalidades importantes que podem ser modificadas ocasionando bugs e falhas em geral da aplicação, para isso existem métodos de testes que podem ser aplicados garantindo a integridade de cada API, como por exemplo o TDD(Test Driven Development) que é uma metodologia onde o teste é implementado antes mesmo do serviço da API, para que a regra de negócio seja validada e preservada desde o nascimento do software em questão, poderíamos também implementar o BDD(Behavior Driven Development) o qual “consideram” uma evolução do TDD. No BDD não é apenas a equipe de desenvolvedores que escrevem os testes, é responsabilidade da trindade (PO, QA, DEV) a escrita deles, o que é uma vantagem de certa forma pois a responsabilidade com a qualidade e atendimentos dos requisitos é granularizado, garantindo uma melhor qualidade no software como um todo.

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

R: A latência pode ser uma grande limitação quando existe a comunicação entre as API’s e as mesmas estão alocadas em diferentes servers/nodes, uma arquitetura de microserviços pode possuir também uma maior complexidade em sua implementação e gestão em comparação a uma arquitetura monolítica, a escalabilidade apesar de ser um dos maiores bônus de uma arquitetura de microserviços, pode se tornar complexa em sua primeira implementação por conta da granularidade dos serviços, por fim os testes também podem ser um pouco mais complexos devido a necessidade de garantir as integrações feitas entre as API’s.

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: Acredito que a principal dificuldade seja a compatibilidade das aplicações com os novos ambientes, a transição muito provavelmente exigiria uma manutenção nas aplicações, outro possível ponto é a configuração, apesar de novos ambientes “containerizados” oferecerem muito mais vantagens no quesito escalabilidade, portabilidade etc. A configuração inicial pode ser um pouco mais complexa, o lado bom é que a documentação do Docker por exemplo, é muito bem construída.

1. Teste prático (em anexo)