# 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.

Existem problemas arquiteturais na solução descrita. A lógica de negócio deve ser movida para classes de serviço, deixando o controlador o mais enxuto possível, somente com as respostas. Pelo que entendi também entidades transitam pelos endpoints, sendo necessária a criação de dto’s (data transfer objects) de entrada/saídas (request/response). E sua consequente conversão, interagindo com os objetos de domínio.

Após a refatoração, é necessária a construção dos testes unitários para que tenhamos a garantia do correto comportamento do sistema e uma porcentagem de aceite mínima estabelecida em uma esteira DevOps. Quando a aplicação é submetida à esteira, a maioria das empresas utiliza o Sonar Cube para a execução/validação dos testes (geralmente a cobertura precisa atingir de 70 a 80%). Pequenos problemas também precisam ser resolvidos após a análise do Sonar, como code smells por exemplo...

1. Descreva quais são as principais limitações ao se adotar servidores de aplicação em uma arquitetura orientada a microsserviços.  
     
   Uma das principais limitações seria a complexidade. Orquestrar inúmeros serviços que conversam entre si, e que se interdependem. Podem haver várias implementações de nossos serviços que conversem com serviços externos de outras organizações, e que se não ficarem atentos a mudanças, podem quebrar-se, diferente de uma arquitetura monolítica. Quando trabalhei no UOL era muito penosa a análise da orquestração entre 6 aplicações de microserviços (para tal utilizávamos ElasticSearch/Kibana) para rastreamento de log’s.  
     
   Falta de padrão. À medida que uma aplicação modularizada cresce, há uma baixa coesão e alto acoplamento. Uma única falha pode insponibilizar a aplicação.   
     
   Segurança. Por mais bem arquitetados, os microserviços não estiverem bem definidos, podem se tornar suscetíveis a ataques.   
     
   Alto tráfego de rede. Por mais que trocas de bytes JSON sejam pequenas, o alto volume pode resultar em altos consumos de banda.  
     
   Aplicação e servidor embutido, aumenta também consideravemente o tamanho da aplicação.
2. 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.  
     
   Em aplicações mais antigas, concebidas para rodar em um servidor independente, existem desvantagens como:  
     
   Você precisaria configurar um pool de conexões no servidor independente, já em servidores embarcados em um framework como o Spring, já possuem seu próprio pool de conexões (Hikari por exemplo). Apesar de gerar um arquivo jar (com servidor embutido) ou um container, é possível também gerar um arquivo war para que a aplicação Spring rode em um servidor independente também.  
     
   Outro grande problema de se rodar uma aplicação em um servidor externo, é a do compartilhamento de recursos, já que nele serão servidas diversas outras aplicações. Teríamos que dimensionar máquinas para se adequar a tal situação, tornando mais custosa a adequação da solução,  
     
   Qualquer outra aplicação que gere algum tipo de problema neste servidor compartilhado, pode derrubar o servidor, ficando indisponíveis todas as outras.  
     
   Diversas versões mais recentes ou atualizações do servidor precisariam ser implementadas, visando a solução de problemas e melhorias. Várias aplicações poderiam quebrar com isto, além de se tornar um processo penoso em vários ambientes (DEV, QA, PROD), onde precisaríamos manter versões de servidor estritamente iguais em todos. Isto dificultaria e muito, a entrega contínua.  
     
   Implementar novos servidores com configurações iniciais ou novas configurações de funcionalidades, pode ser infinitamente mais difícil do que a própria configuração de nossas aplicações.
3. Teste prático (em anexo)