# 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.

1. Descreva quais são as principais limitações ao se adotar servidores de aplicação em uma arquitetura orientada a microsserviços.
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.
3. Teste prático (em anexo)
   1. Controllers não devem conter lógica de negócio, a responsabilidade deve ser passada para a camada de serviço, pois a camada de controle serve apenas como a interface da aplicação, onde é feito o mapeamento dos métodos Java para endpoints (métodos HTTP). Assim como domínio deve conter apenas classes ou tabelas que representem os modelos de dados da aplicação. Testes unitários são necessários para garantir o funcionamento de vários pontos do sistema, sem eles fica muito mais difícil para o desenvolvedor descobrir novos bugs, principalmente quando algum trecho de código for alterado, o ideal é ter uma cobertura de teste que cubra pelo menos de 80% a 90% da aplicação (podendo ser garantida por ferramentas como SonarQube por exemplo).
   2. A arquitetura de microsserviços consiste em dividir o que seria uma grande aplicação em pequenas unidades divididas por funcionalidades ou qualquer outra regra, que conversam entre si. No entanto, neste processo ocorre alguns problemas como alto uso de rede e complexidade de gerenciamento de todas instâncias. Surgindo a necessidade de adotar formas mais leves e flexíveis de hospedar e executar aplicações sabendo que servidores de aplicação são pesados e até mesmo lentos para executar. No exemplo do Java, hoje existem outros frameworks que são vistos como “Cloud-ready” como o Quarkus, que promete colocar o Java em um contexto de muito mais velocidade na execução como também menos uso de memória RAM, utilizando algumas abordagens diferentes ao Spring Boot (que utiliza JVM convencional e servidor embutido). Ainda no contexto de cloud nós vemos atualmente muitas linguagens de script sendo utilizadas para microsserviços como NodeJS e pyrthon, por serem leves e de fácil configuração.
   3. Podemos elencar três pontos principais:
      1. Configuração: Configurar um servidor embutido pode ser mais complexo do que configurar um servidor externo, pois pode haver menos recursos e documentação disponíveis.
      2. Gerenciamento de recursos: O gerenciamento de recursos, como memória e CPU, pode ser mais difícil com um servidor embutido, pois ele pode não ter as mesmas opções de configuração que os servidores externos.
      3. Monitoramento e gerenciamento: Monitorar e gerenciar uma aplicação rodando em um servidor embutido pode ser mais desafiador do que fazê-lo em um servidor externo, pois as ferramentas de monitoramento e gerenciamento podem ser limitadas.