# 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: Bom! Acredito que esse seria mais ou menos a divisão das camadas dessa aplicação:

Diagrama

Descrição gerada automaticamente

Bom! Não é fácil projetar software que tenha um bom desacoplamento e uma alta coesão, aplicando inversão de dependência, princípios OOP, SOLIDS e pelo menos a parte tática do DDD e conseguir mantê-lo assim, ainda mais quando os membros mudam muito e quem assume não consegue manter os princípios e acaba violando as regras de arquiteturas. Por isso a importância de um bom code review. Alguns até usam o Arch Unit para tentar garantir que o design não seja violado etc. Esses seria o meu guia para escrever um novo ou dá manutenção do software. Mais como sempre depende.

Com isso vejo alguns problemas: O sistema parece não ter separação clara das responsabilidades, apresentando alto acoplamento. Então poderíamos aplicar uma refatoração, como, remover as logicas de negócio do controller, deixando apenas as validações dos dados que entra pelo controller. Ex Beans Validations, dependendo, teriam um Mapper para converter os objetos de requests para o objeto de domínio. Eu tentaria evitar classes anêmicas. Mais como sempre depende da complexidade dos sistemas, já que tem alguns backends(“microservices”), que parece mais um proxy. Uma vez na camada de domínio, eu aplicaria as regras de negócios. Onde as classes não teriam nenhum mapeamento como o banco. Uma vez passado pelo domínio. Eu teria mais mapper para converter para o objeto de entidades, no qual esse será mapeada para uma base, isso acredito que já dariam uma melhor qualidade e facilitaria uma possível manutenção. E escreveria testes unitários e de integração, testes não são bala de prata contra bugs, mais aumenta muito a qualidade, e confiabilidade do software.

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

R: Bom!! Posso estar sendo ingênuo aqui, mas não vejo um servidor de aplicação como limitado. Ele só está sendo usado para o contexto errado. Já que um microserviço, tem algumas características, que um application server, não serviria. Funcionária, mas seria a escolha errada.

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: Bom! Depende. Os servidores de aplicação têm características de prover full Jakarta EE, e as vezes os softwares que a usam, não fazem uso do poder e recursos que ele entrega. Sendo adotado por grandes corporações. Com seus monolitos e integrações SOA etc. Com isso dependendo do contexto faz sentido usar alguns frameworks como Spring-boot, Micronaut, Quarkus e etc, que tem características de oferecer servidores mais leves embutidos, melhorando seu tempo de start. Podendo ser rodado em container ou Vms. Bom! Eu posso estar sendo equivocado aqui, mas não lembro de algum desafio usando esses framewaoks.

1. Teste prático (em anexo)