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: A arquitetura atual viola princípios fundamentais do SOLID, especialmente o da responsabilidade única (SRP), ao concentrar a lógica de negócio nos controllers, o que compromete a coesão, dificulta testes unitários e reduz a escalabilidade. Além disso, o domínio é anêmico, contendo apenas atributos e mapeamentos, contrariando os fundamentos do Domain-Driven Design (DDD), que defendem um domínio rico e orientado a comportamentos. Para melhorar, propõe-se a adoção da Clean Architecture, separando bem as camadas de controller, aplicação, domínio e infraestrutura. As regras de negócio devem ser movidas para serviços de aplicação, e os comportamentos encapsulados em entidades e objetos de valor. Além disso, testes automatizados com JUnit e Mockito devem ser implementados para garantir confiabilidade e facilitar a manutenção contínua do sistema.

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

   R: Servidores tradicionais como JBoss e WebLogic não são ideais para microsserviços por serem pesados, lentos para iniciar e exigirem configurações centralizadas. Eles dificultam o escalonamento horizontal e não se integram bem com práticas modernas de DevOps, como o uso de containers e orquestradores. Microsserviços exigem leveza, isolamento e agilidade, o que contrasta com a proposta monolítica dos servidores de aplicação clássicos. Além disso, esses servidores geralmente mantêm estado em memória, o que vai contra a natureza stateless recomendada para microsserviços. O acoplamento com infraestrutura compartilhada também limita a autonomia de deploy e versionamento independente, prejudicando a entrega contínua e a resiliência do sistema distribuído.

3. 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: A principal dificuldade ao adotar servidores embutidos está no gerenciamento descentralizado, pois cada aplicação precisa cuidar de suas configurações, segurança e monitoramento. Isso demanda maior maturidade da equipe e uso de ferramentas externas para logs, métricas e autenticação. Apesar disso, essa abordagem favorece a escalabilidade e integração com containers, sendo mais adequada para microsserviços modernos.

4. Teste prático (em anexo)