Desafio VR Desenvolvimento

• 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 Sistema pare estar bem dividido , cada camada responsavel por algo como o SOLID sugere.

--> O controller recebe a requisição e está com toda lógica de negócio. Monta e repassa o domínio para a aplicação;

O Controller não deveria possuir lógica de negocio ,o Controler deve ter o mapeamento do endpoint que recebe um DTO request na chamada e retorna um DTO Response ou uma mensagem com código.

--> A aplicação tem a responsabilidade de repassar o objeto pronto para o repositório;

Devemos preencher o objeto da entity e passar esse objeto no parametro do metodo do repositoty e ele fazer a persistencia.

--> O repositório apenas persiste os objetos mapeados do hibernate através de spring data;

Não . Podemos ter os objetos mapeados do spring data como podemos ter também classes que fazem acesso a banco de dados através de query nativa .

--> O domínio apenas faz o mapeamento para o BD;

Não. Nessas classes de domínio fazemos validacoes de regras de negocio tambem por exemplo.

--> Nenhum teste unitário foi escrito.

Os testes unitários garantem maior quaidade da aplicação, ele já faz validacoes de possiveis nullpointers por exemplo e evitando futuros bugs.

- 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.

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

Ao adotar servidores de aplicação (como JBoss, WebLogic, Tomcat, etc.) em uma arquitetura de microsserviços, algumas das principais limitações são:

Sobrecarga de recursos: Servidores de aplicação tendem a ser pesados, com muitas funcionalidades integradas (transações distribuídas, serviços de segurança, EJB, etc.), o que aumenta o uso de memória e CPU. Isso vai contra a leveza desejada em microsserviços.

Tempo de inicialização elevado: Aplicações em servidores tradicionais geralmente demoram mais para iniciar, o que impacta negativamente a escalabilidade e a recuperação rápida de falhas, essenciais em ambientes com orquestração como Kubernetes.

Dificuldade de automação e deploy contínuo: Aplicações em servidores de aplicação exigem empacotamento em formatos como EAR/WAR, além de configuração e deploy específicos no servidor. Isso pode dificultar o uso de pipelines modernos de CI/CD e práticas como blue-green deployments.

Acoplamento à plataforma: Usar recursos proprietários de um servidor de aplicação (como datasources JNDI, serviços proprietários) pode tornar os microsserviços dependentes da plataforma, dificultando a portabilidade e a independência entre os serviços.

Escalabilidade limitada: Em microsserviços, cada serviço deve ser leve e escalável de forma independente. O uso de servidores de aplicação pode dificultar o scaling horizontal, principalmente quando o servidor carrega múltiplos serviços ou módulos ao mesmo tempo.

--> 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.

Os principais desafios ao adotar ferramentas com servidores embutidos (como Spring Boot com Tomcat/Jetty/Undertow embutido) incluem:

Gestão de dependências e atualizações: Como o servidor está embutido na aplicação, o time de desenvolvimento se torna responsável por garantir que as versões do servidor, bibliotecas e segurança estejam sempre atualizadas, o que antes era gerenciado separadamente pelo administrador do servidor de aplicação.

Observabilidade e monitoramento: Ferramentas tradicionais de monitoramento (como JMX/JConsole em servidores de aplicação) podem não estar disponíveis ou ser limitadas, exigindo a adoção de outras abordagens (Micrometer, Prometheus, etc.) e aumentando a complexidade da instrumentação.

Padronização em ambientes corporativos: Em grandes organizações que usam servidores de aplicação para centralizar configurações (como segurança, datasources, logging), a migração para servidores embutidos pode romper essa padronização, exigindo reconfiguração em cada microserviço individualmente.

Carga de responsabilidade no time de desenvolvimento: Com o servidor embutido, o pacote final é autossuficiente, mas isso implica que o time precisa cuidar de aspectos antes geridos pela infraestrutura, como segurança, tuning de servidor, e políticas de deploy.

Curva de aprendizado e reestruturação: Equipes acostumadas com o modelo tradicional de servidores de aplicação precisam se adaptar a novas ferramentas e práticas, como empacotamento como fat jar, uso de containers, e estratégias modernas de observabilidade.

Justificativa:

Apesar dos desafios, a adoção de servidores embutidos oferece maior autonomia, agilidade e simplicidade no deploy, sendo mais alinhada a ambientes de microsserviços e nuvem. No entanto, exige um novo modelo operacional que envolve mudanças culturais, técnicas e de responsabilidade nas equipes de desenvolvimento e operação.