# 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.

A descrição acima apresenta alguns problemas e oferece oportunidades para melhorias. Podemos destacar alguns pontos de observação e sugerir estratégias para aprimorar a qualidade e a manutenção do sistema:

1. **Controller com Muita Lógica de Negócio:**
   * Concentrar toda a lógica de negócio no Controller pode tornar o código difícil de entender, testar e manter.
   * Implementar o padrão de design MVC (Model-View-Controller) corretamente. Mova a lógica de negócio para serviços dedicados (camada de serviço) e mantenha o Controller apenas para manipulação de requisições e respostas.
2. **Ausência de Testes Unitários:**
   * A falta de testes unitários torna difícil garantir a confiabilidade do sistema e dificulta a detecção de regressões.
   * Implementar testes unitários para cada camada do sistema. Utilizar frameworks como JUnit e Mockito para testes de unidade. Testes automatizados aumentam a confiança na estabilidade do código.
3. **Arquitetura Monolítica:**
   * A descrição sugere uma arquitetura monolítica, o que pode resultar em acoplamento excessivo e dificuldades de escalabilidade.
   * Avaliar a possibilidade de migrar para uma arquitetura de microsserviços. Isso pode facilitar a escalabilidade, manutenção e implementação de novas funcionalidades independentes.
4. **Responsabilidades Não Adequadamente Distribuídas:**
   * A aplicação tem a responsabilidade de preparar o objeto para o repositório, e o repositório apenas persiste objetos mapeados. Pode haver uma falta de clareza nas responsabilidades.
   * Adotar o padrão de design Repository corretamente. Repositórios devem ser responsáveis apenas pela persistência, enquanto a aplicação (ou serviços) deve se concentrar em manipular a lógica de negócio.
5. **Código sem Padronização e Boas Práticas:**
   * A descrição não menciona a padronização do código e o uso de boas práticas de programação.
   * Implementar padrões de codificação consistentes, como convenções de nomenclatura, estilo de código e boas práticas de programação Java. Utilizar ferramentas de análise estática de código (por exemplo, SonarQube) para identificar e corrigir problemas.
6. **Falta de Documentação:**
   * A ausência de documentação pode dificultar a compreensão do sistema por outros desenvolvedores.
   * Criar documentação adequada, incluindo documentação de código, README, e documentação de API. Isso ajuda na integração de novos desenvolvedores e facilita futuras manutenções.
7. **Segurança:**
   * Não há menção à segurança no sistema.
   * Implementar práticas de segurança, como autenticação e autorização. Utilizar HTTPS para comunicação segura. Garantir que o sistema esteja protegido contra ameaças comuns, como injeção de SQL e Cross-Site Scripting (XSS).
8. **Monitoramento e Log:**
   * Não há informações sobre como o sistema é monitorado e registrado.
   * Implementar uma estratégia de log eficaz e configurar ferramentas de monitoramento para identificar problemas de desempenho e falhas.
9. Descreva quais são as principais limitações ao se adotar servidores de aplicação em uma arquitetura orientada a microsserviços.

Quando se escolhe adotar servidores de aplicação em uma arquitetura orientada a microsserviços pode apresentar algumas limitações, especialmente quando comparado a abordagens mais leves e descentralizadas. Aqui estão algumas das principais limitações:

1. **Acoplamento Tecnológico:**
   * Servidores de aplicação muitas vezes impõem escolhas tecnológicas específicas. Isso pode resultar em acoplamento tecnológico, onde todos os microsserviços precisam aderir às mesmas tecnologias suportadas pelo servidor de aplicação.
   * Dificuldade em escolher as melhores tecnologias para cada microsserviço, limitando a flexibilidade da equipe de desenvolvimento.
2. **Dificuldades na Escalabilidade Independente:**
   * Servidores de aplicação podem complicar a escalabilidade independente de microsserviços. Escalar um único microsserviço muitas vezes significa escalar toda a aplicação implantada no servidor de aplicação.
   * Menos flexibilidade na gestão de recursos e custos, pois os microsserviços podem ter demandas diferentes de escalabilidade.
3. **Overhead de Recursos:**
   * Servidores de aplicação geralmente incluem muitos recursos que podem não ser necessários para microsserviços mais leves.
   * Desperdício de recursos, incluindo memória e poder de processamento, o que pode ser significativo quando muitos microsserviços estão em execução.
4. **Dificuldades em Adotar Novas Tecnologias:**
   * A adoção de novas tecnologias pode ser mais desafiadora em um ambiente baseado em servidores de aplicação, pois a transição pode exigir mudanças significativas na infraestrutura.
   * Atrasos na adoção de tecnologias mais recentes que podem beneficiar microsserviços específicos.
5. **Problemas de Manutenção e Atualização:**
   * Atualizar ou manter um servidor de aplicação pode impactar todos os microsserviços que dependem dele.
   * Dificuldades na aplicação de correções e atualizações, uma vez que todas as instâncias do servidor de aplicação precisam ser atualizadas simultaneamente.
6. 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.

Quando optamos por ferramentas que incorporam um servidor embutido em vez de servidores de aplicação separados, como JBoss ou Tomcat, existem alguns desafios e considerações a serem levadas em conta:

Desafios de Escalabilidade de Servidores embutidos muitas vezes não possuem a mesma capacidade de escalabilidade horizontal que servidores de aplicação dedicados. Isso pode ser um desafio se a aplicação precisar lidar com um grande volume de tráfego e exigir escalabilidade eficiente.

Menor Flexibilidade de Implantação com ferramentas com servidores embutidos podem limitar as opções de implantação. Isso pode ser um desafio se a aplicação precisar ser implantada em ambientes específicos ou integrada a infraestruturas mais complexas.

Acoplamento Tecnológico com ferramentas com servidores embutidos pode impor escolhas tecnológicas específicas. Isso pode resultar em acoplamento tecnológico, onde a aplicação fica vinculada às tecnologias suportadas pela ferramenta, limitando a flexibilidade.

Dificuldades na Atualização e Manutenção: de atualizar a versão do servidor embutido pode ser mais desafiador do que atualizar um servidor de aplicação dedicado. Isso pode resultar em dificuldades na aplicação de correções e atualizações.

Segurança e Controle em alguns casos, servidores embutidos podem ter configurações de segurança menos flexíveis do que servidores de aplicação dedicados. Isso pode ser um desafio se a aplicação exigir configurações específicas de segurança.

Limitações de Recursos e Desempenho dos Servidores embutidos podem ter limitações de recursos em comparação com servidores de aplicação dedicados. Isso pode impactar o desempenho da aplicação, especialmente em cenários de carga elevada.

Ao fazer a escolha entre servidores embutidos e servidores de aplicação dedicados dependerá basicamente dos requisitos específicos do projeto e das características da aplicação. Enquanto servidores embutidos podem ser adequados para projetos menores ou casos de uso específicos, aplicações mais complexas ou com requisitos de escalabilidade mais elevados podem se beneficiar de servidores de aplicação dedicados, que oferecem maior flexibilidade, controle e desempenho. Na tomada dessa decisão deve ser cuidadosamente ponderada, considerando as necessidades atuais e futuras da aplicação e da equipe de desenvolvimento.

1. Teste prático (em anexo)

