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 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.

* Descreva quais são as principais limitações ao se adotar servidores de aplicação em uma arquitetura orientada a microsserviços.
* 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.
* Teste prático (em anexo)

**Principais problemas encontrados:**

- Primeiro problema é a regra de negócio está no controller, quando deveria está no service;

- O uso do mapeamento do Hibernate e do Spring Data torna a solução acoplada a essas tecnologias e dificulta a mudança para outros fornecedores;

- A falta de testes unitários não da a garantia de que o código é um código de qualidade;

Para melhorar, algumas estratégias poderiam ser:

- Refatorar a lógica de negócio do controller para uma camada de serviço intermediária, seguindo o princípio de responsabilidade única. Isso irá melhorar a manutenção, legibilidade e testabilidade do código, ou seja, cada camada terá sua responsabilidade, e toda a lógica de negócio ficaria na camada de serviço;

- Escrever cenários de testes e implementar testes unitários para garantir a qualidade do código;

- O sistema está engessado com as tecnologias de mapeamento do banco de dados, o correto será desaclopar isso usando padrões de design e interfaces.

**Principais limitações ao se adotar servidores de aplicação em uma arquitetura orientada a microsserviços:**

- Se a arquitetura não for bem projetada, pode se tornar um código difícil de se manter;

- Pode acontecer uma sobrecarga de recursos em cima do servidor, uma arquitetura de microsserviçoes requer um ambiente bem projetado, se isso não acontecer os servidores podem sobrecarregar e acabar gastando recursos desnecessários;

- O uso de servidores de aplicação pode levar a um maior acoplamento entre os serviços, o que pode tornar mais difícil a manutenção e evolução dos microsserviços;

- O gerenciamento de um servidor de aplicação geralmente requer a presença de uma equipe de DevOps, o que acaba se tornando caro, embora seja necessário.

**Problemas ao optar por utilizar um servidor embutido em uma ferramenta Java:**

- O sistema ficará totalmente dependente a essa ferramenta, as suas configurações, e a maneira como essa ferramenta se comporta, grande parte dessas ferramentas são engessadas e não aceitam personalizações, geralmente você só pega e usa;

- Dependendo da ferramenta, a falta de suporte por parte da comunidade pode ser um problema, exemplo, o Quarkus Framework, é uma ótima ferramenta, porém a comunidade é fraca e deixa muito a desejar em relação ao suporte.