# Desafio VR Desenvolvimento

1. Recebemos um código desenvolvido por terceiros de um sistema que possui volume alto de lógica de negócio e apresenta a seguinte característica:

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 a sua estratégia para melhorar este sistema em termos de qualidade e manutenção.

Resposta:

1 Parte:

Os controles acumulam responsabilidades que não lhe são próprias, a concentração de regras de negócios e a

contato com classes de domínios impõe um auto acoplamento tanto para manutenção corretiva, evolutiva e também, a

dependência de um Sistema de Gerenciamento de Banco de Dados (SGBD) específico.

Impede a apropriada validação por camadas, como por exemplo: as validações de requisição, de negócio e de domínio.

Além de dificultar a implementação de testes, a falta de divisão de responsabilidade tem como consequência os testes

unitários inchados, onde seria necessário carregar muitas dependências para fazer testes mínimos.

2 Parte:

A divisão em camadas tem como objetivo separar as responsabilidades. Neste caso, visualizo ao mínimo três

camadas bem distintas: a camada de controle, de serviço e de repositório. Juntamente com está divisão seguiria a

implementação de conversores de objetos de transferências entre as camadas.

Na camada de controle deixaria a responsabilidade de direcionar ao serviço apropriado, para coletar as

informações apropriadas a requisição, limitando o acesso ao repositório somente através do serviço.

Na layer de serviço definiria implementações agrupados por assuntos bem distintos, onde cada regra de negócio

possa ser avaliada individualmente e cada implementação tendo uma interface correspondente, para garantir evoluções e

alterações em uma regra específica sem causar risco as demais.

Os repositórios sendo acessados exclusivamente pela camada de serviço, somando a utilização de conversores

entre as camadas, possibilitaria a fácil substituição do SGBD.

O uso de conversores de objetos de transferência mantém claro gerenciamento de dependências entre as camadas, tornando-as coesas e evidente o ponto onde haverá impacto nas alterações futuras. Além de permitir a validação dos objetos nas camadas apropriadas, conseguimos verificar quais as informações devem ser trafegadas entre elas.

Como já citado, a divisão em camadas torna os testes unitários mais leves quanto a injeção de serviços ou/e

mocks, tonando-os mais granuláveis e verificáveis, tanto na questão de erro de implementação quanto na aderência ao

negócio.

A própria implementação de testes com a devida cobertura torna o sistema mais maduro e seguro para alterações futuras

permitindo testes regressivos.

1. Descreva quais são as principais limitações ao se adotar servidores de aplicação em uma arquitetura orientada a microsserviços.
2. Quais são os principais desafios ao se adotar aplicações do tipo "Embedded Servlet” em relação a aplicações desenvolvidas para um application server?
3. Defina os resources e métodos http para o controller abaixo utilizando annotation do Spring MVC:

@RestController

public class PedidoResource {

public ResponseEntity<ServiceResponse<PedidoVO>> consultar(String idPedido) {

...

}

public ResponseEntity<ServicePageableResponse<List<PedidoVO>>> listar(PedidoListarDto pedidoListarDto) {

...

}

public ResponseEntity<ServiceResponse<Void>> criar(PedidoVO pedido) {

...

}

public ResponseEntity<ServiceResponse<Void>> alterar(

String idPedido,

PedidoVO pedido) {

...

}

public ResponseEntity<ServiceResponse<Void>> excluir(

String idPedido,

PedidoVO pedido) {

...

}

public ResponseEntity<ServiceResponse<Void>> atualizarStatus(

String idPedido,

Status status) {

...

}

public ResponseEntity<ServiceResponse<Void>> atualizarVencimentoPedido(

String idPedido,

VencimentoPedidoDto vencimentoPedidoDto) {

...

}

}