# 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.

A lógica de negócio deveria ser isolada em outra camada, deixando o controller somente com a responsabilidade de receber e responder às requisições;

Testes unitários são importantes para validar as regras de negócio esperadas e o comportamento da aplicação diante de possíveis problemas, e deveriam ser escritos; assim, com testes para cada camada somados a testes de integração, a aplicação ganharia robustez e confiabilidade;

Spring boot é uma boa escolha, facilita a configuração e o publicação da aplicação, ganhando tempo de desenvolvimento valioso com poucas configurações.

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

Configurações excessivas com os XMLs; consumo alto de recursos computacionais apenas para inicializar, sendo demorado esse processo; risco de incompatibilidade entre versões da JVM e/ou bibliotecas.

1. Quais são os principais desafios ao se adotar aplicações do tipo "Embedded Servlet” em relação a aplicações desenvolvidas para um application server?

Liberdade para adicionar somente os módulos do JavaEE que forem necessários para a aplicação, já que o tipo embedded só abriga basicamente o servlet container.

Talvez seja necessário escrever mais para inicializar programaticamente, porém com menos configuração para fazer.

1. Defina os resources e métodos http para o controller abaixo utilizando annotation do Spring MVC:

@RestController

@RequestMapping(“/pedidos”)

public class PedidoResource {

@GetMapping(value = “/{id}”)

public ResponseEntity<ServiceResponse<PedidoVO>> consultar(@PathVariable(“{id}”) String idPedido) {

...

}

@GetMapping

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

...

}

@PostMapping

@ResponseStatus(HttpStatus.CREATED)

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

...

}

@PutMapping(value = “/{id}”)

@ResponseStatus(HttpStatus.OK)

public ResponseEntity<ServiceResponse<Void>> alterar(

@PathVariable(“{id}”) String idPedido,

@RequestBody PedidoVO pedido) {

...

}

@DeleteMapping(value = “{id}”)

@ResponseStatus(HttpStatus.OK)

public ResponseEntity<ServiceResponse<Void>> excluir(

@PathVariable(“id”) String idPedido,

@RequestBody PedidoVO pedido) {

...

}

@PatchMapping(value = “{id}”)

@ResponseStatus(HttpStatus.OK)

public ResponseEntity<ServiceResponse<Void>> atualizarStatus(

@PathVariable(“id”) String idPedido,

@RequestBody Status status) {

...

}

@PatchMapping(value = “{id}”)

@ResponseStatus(HttpStatus.OK)

public ResponseEntity<ServiceResponse<Void>> atualizarVencimentoPedido(

@PathVariable(“id”) String idPedido,

@RequestBody VencimentoPedidoDto vencimentoPedidoDto) {

...

}

}