DDD e SOLID aplicado para API Design

Abordagem para APIs sustentáveis em Spring Boot

Abordagem atual - MVC com Facades

Vantagens

- Início de desenvolvimento mais rápido
- Útil para projetos que em estágio inicial, para diagnóstico e rápida resolução de problemas

Desvantagens

- Em projetos grandes tende a virar um código muito abrangente
- Códigos abrangentes afetam a manutenibilidade e o melhor rastreio de bugs
- Mais difícil leitura
- Acoplamento muito grande, baixa coesão

Aplicação de SOLID

```
@Override
@Transactional(rollbackFor = { RuntimeException.class })
public CadastroVistoriaResponse salvarVistoriaComLegado(CadastroVistoriaRequest request) {
```

- Dificulta a adaptação de novos devs: é o método correto a ser chamado para criar uma nova vistoria?
- Faz muitas coisas, futuramente quais desses fluxos serão reutilizados?
- Consigo introduzir um novo comportamento num código tão longo?
- Testes unitários se tornam extensos
- Conflitos de merge mais complexos pois mais risco de 2 devs modificarem um mesmo fluxo

Exemplo: Criação de alertas

Citar método fora da transação

Aplicação de DDD em REST

```java
@RestController
@Api(tags = "DISTRIBUICAO", value = "Controlador Rest de Distribuição")
@RequestMapping(value = "/distribuicao-vistoria")
public class DistribuicaoController {

    @ApiOperation(value = "Buscar vistorias para distribuição")      You, now • Uncommitted changes
    @GetMapping(value = "/vistorias")
    public ResponseEntity<VistoriaResponse> buscarVistorias(@Valid VistoriaFiltroRequest request) {

        VistoriaResponse response = distribuicaoFacade.buscarVistorias(request);
        return ResponseEntity.status(HttpStatus.OK).body(response);
    }
```

- Escopos sem um domínio causam dificuldade em achar endpoint de busca por vistoria

Aplicação de DDD em REST

```
        );

        CadastroVistoriaResponse cadastroVistoriaResponse = this.distribuicaoVistoriaService.buscaVistorias(
                cadastroRealizacaoVistoria.getCodVistoria()
        );

        cadastroRealizacaoVistoriaResponse.setDetalhesStatusLaudo(
                detalheStatusLaudoResponseList
        );

//      this.dadosVeiculoBo.verificarCadastroDadosVeiculosResponse(
//              cadastroVistoriaResponse
//      );

        VersaoVeiculoResponse versaoVeiculoResponse = this.dadosVeiculoAutoService.buscarVersaoVeiculoByCodVersao(
                cadastroVistoriaResponse.getCadastroDadosVeiculos().getCodVersaoVeiculo()
        );

        DadoVeiculoResponse dadoVeiculoResponse = cadastroRealizacaoVistoriaBo.criarDadoVeiculoResponse(
                cadastroRealizacaoVistoria,
                cadastroVistoriaResponse,
                versaoVeiculoResponse
        );

        cadastroRealizacaoVistoriaResponse.setDadoVeiculo(
                dadoVeiculoResponse
        );

        VistoriaResponse vistoriaResponse = this.distribuicaoVistoriaService.buscaVistoriasDistribuicao(
                cadastroRealizacaoVistoria.getCodVistoria()
        );
```

- Escopos sem um domínio causam dificuldade em achar endpoint de busca por vistoria
- Endpoint `/realizacao-vistorias/{codVistoria}` de realização busca por vistorias duas vezes por falta de definição de domínio correto para resgatar vistoria

Aplicação de DDD em REST

- Alguns exemplos de como falta de domínio estabelecido afeta a performance das aplicações:
  - Avarias e Status Vistoria deveriam estar no Realização