



Departamento de Polícia Rodoviária Federal

Projeto: Joker

Nota Técnica

DPRF	Joker - Nota técnica	
-------------	-----------------------------	--

Revisão	Descrição	Autor	Data
1.0	Construção do documento	Israel Branco	25/11/2020
1.1	Análise branch corregedoria	Israel Branco	03/11/2021

1 Sumário

2 Considerações iniciais.....	4
3 Apresentação do cenário atual.....	5
3.1 Tecnologias utilizadas.....	8
4 Análise técnica.....	9
4.1 SonarQube.....	9
4.2 OWASP Dependency Check.....	10
4.3 OWASP ZAP.....	11
4.4 Estrutura do projeto.....	12
4.5 Manutenibilidade de código.....	13
4.6 Confiabilidade.....	13
4.7 Performance e estabilidade.....	14
5 Recomendações.....	15

2 Considerações iniciais

Este documento visa reportar o resultado da análise efetuada na aplicação **Joker**, para este estudo foram desconsiderados todo o contexto negocial ao qual a ferramenta está inserida, também foram desconsideradas o ambiente ao qual a ferramenta esta operando sendo analisado puramente questões que tangem a qualidade de código, padrões de codificação, vulnerabilidades de dependências, modelo relacional de banco de dados e concepção arquitetural.

Para a realização desta análise, gerou-se a *nota-tecnica-ctis-20-01-2021* no repositório <https://git.prf/bruno.nobrega/joker/-/tags/nota-tecnica-ctis-20-01-2021> com referência branch corregedoria na data de 04/01/2021.

3 Apresentação do cenário atual

Esta sessão ira descrever a arquitetura, tecnologias, frameworks e dependências que compõe a base da aplicação.

O sistema Joker foi construído para funcionar em ambiente WEB utiliza tecnologia Java com stack Spring apoiada pelo framework acelerador Spring Boot. A aplicação esta estruturada arquiteturalmente como uma aplicação monolítica (entende-se por este termo quando o sistema é composto por camadas de interface com usuário, camada de aplicação de regras negociais e camada de acesso a dados combinadas em uma única aplicação), utiliza o banco de dados *MySQL*, *possui integração com sistemas internos a PRF e* fornece um conjunto de endpoints para consumo externo.

Sua arquitetura é composta por um único componente e sua estrutura está dividida em camadas bem definidas e bem segmentadas, os diagramas a seguir representam o modelo de componentes e o modelo sequencial do fluxo principal.

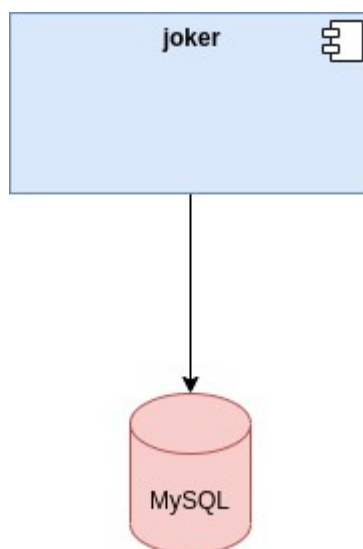


Figura 1: Diagrama de componentes

A aplicação utiliza o modelo MVC para a segregação de responsabilidades em camadas sendo que as requisições http são oriundas das páginas HTML. O diagrama a seguir representa os fluxos encontrados durante o processo de análise da aplicação.

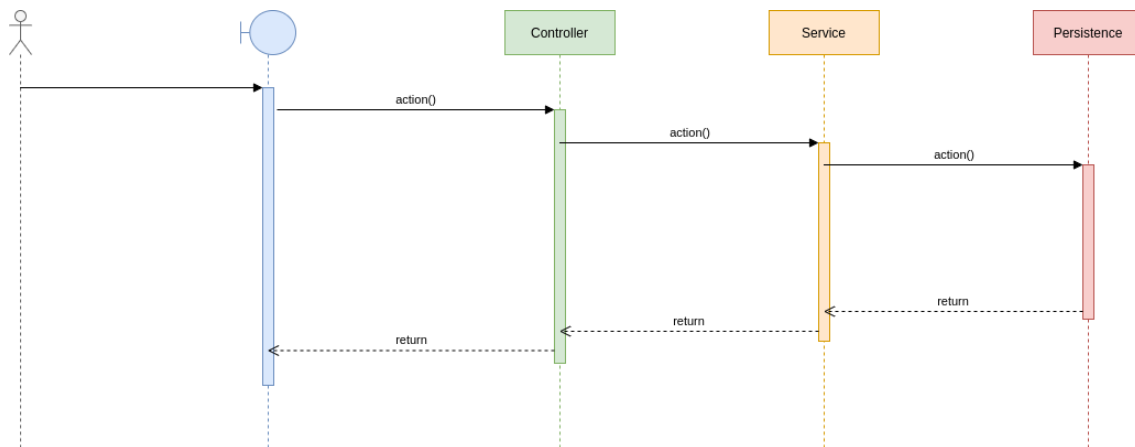


Figura 2: Diagrama de sequências - Fluxo principal



A solução não dispõe de scripts para criação de banco de dados, seu schema é criado ao subir no momento ao qual a aplicação é executada pela primeira vez, sendo que o usuário configurado para acessar o banco deve possuir privilégios para tal.

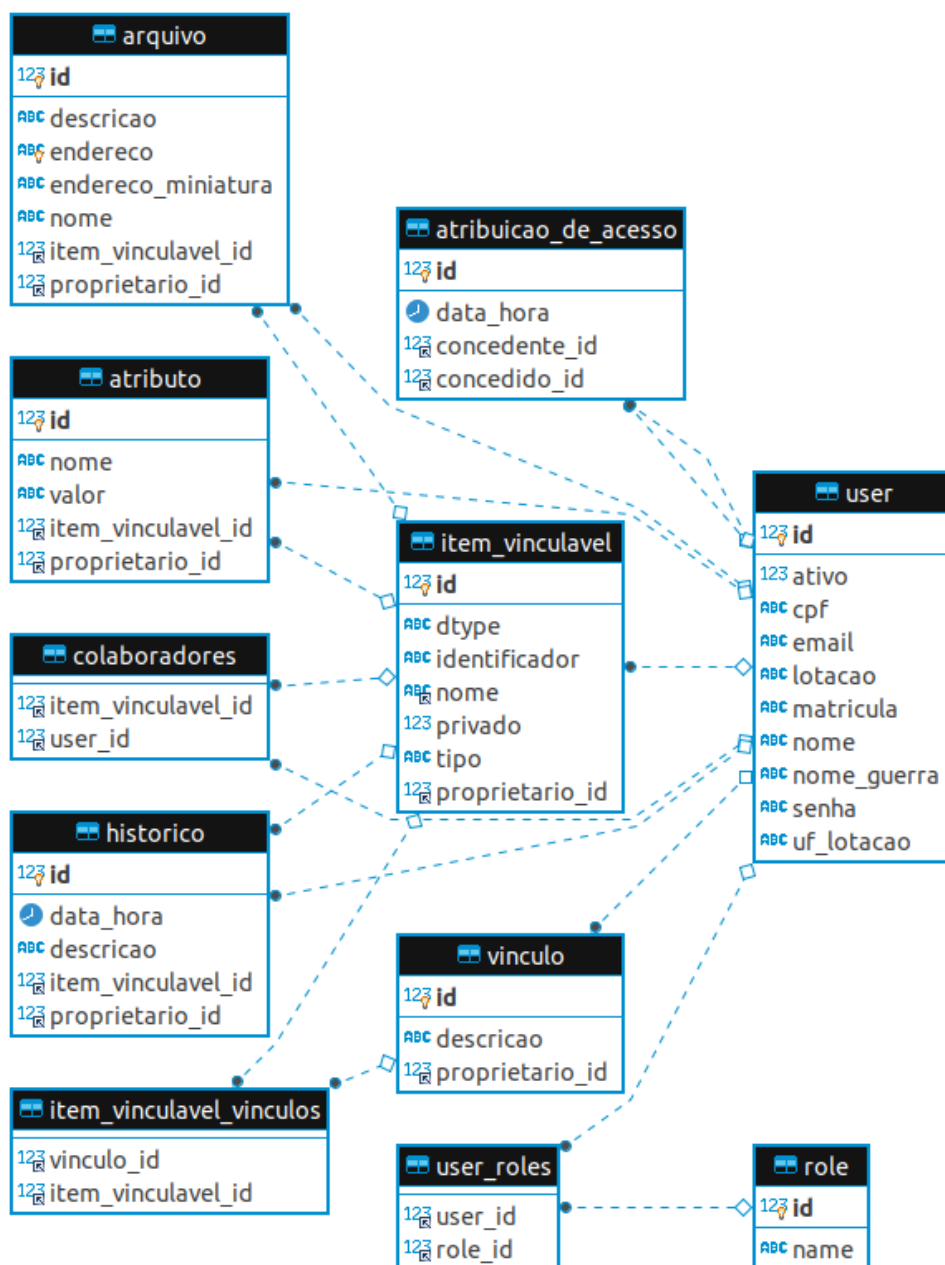


Figura 3: MER - Joker

3.1 Tecnologias utilizadas

Esta sessão descreve as tecnologias, frameworks e principais bibliotecas utilizadas na construção do projeto conjuntamente com suas versões e propósitos de utilização.

Nome	Versão	Utilização	Observação
Java	1.8	Linguagem de programação.	
Spring Boot	2.2.7	Diversos módulos	
Maven	3.3.x	Gerenciador de build/dependências	
Thymeleaf		Camada de apresentação	
MySQL		Banco de dados relacional	

4 Análise técnica

Este tópico descreve a ferramenta do ponto de vista técnico, tanto nos aspectos de codificação, análise estática de código, análise de vulnerabilidade de dependências e particularidades de implementação.

4.1 SonarQube

Ferramenta utilizada para verificação de estática de código. Para esta análise não foram utilizadas as métricas de qualidade implantadas no SonarQube da DPRF, contudo foram utilizadas as regras padrões de análise da ferramenta.

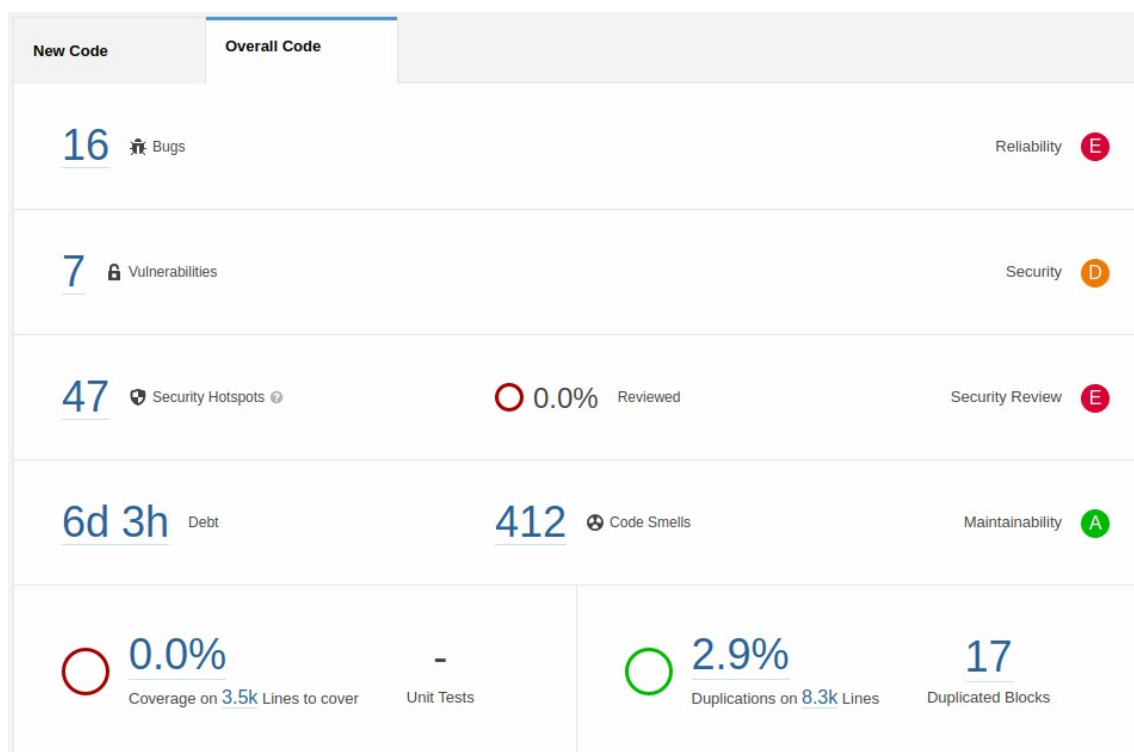


Figura 4: Análise estática de código

- 16 bugs;
- 7 vulnerabilidades de código;
- 47 violações de segurança;
- 412 violações de código ruim (complexidade cognitiva , complexidade ciclomática e débito técnico);
- 2.9% de duplicidade de código

4.2 OWASP Dependency Check

A utilização de bibliotecas de terceiros aumenta substancialmente a produtividade na construção de um software, contudo estas podem trazer consigo vulnerabilidades que afetam diretamente a segurança da aplicação. A ferramenta Dependency Check tem como propósito efetuar análise de vulnerabilidade de dependências utilizadas na construção deste projeto, a seguir temos as principais informações extraídas desta análise, a relação completa desta análise está disponível no Anexo I deste documento.

Dependency	CVE Count	Confidence	Evidence Count
hibernate-core-5.4.15.Final.jar	2	Low	39
log4j-api-2.12.1.jar	1	Highest	46
snakeyaml-1.25.jar	1	Highest	28
spring-security-core-5.2.4.RELEASE.jar	1	Highest	29
spring-core-5.2.6.RELEASE.jar	1	Highest	30
lang-tag-1.5.jar	4	High	43
spring-security-oauth2-core-5.2.4.RELEASE.jar	1	Highest	31
nimbus-jose-jwt-7.8.1.jar	1	Highest	46
groovy-2.5.11.jar	1	Highest	44
tomcat-embed-core-9.0.34.jar	7	Highest	39
spring-security-rsa-1.0.9.RELEASE.jar	1	Highest	33
spring-cloud-netflix-ribbon-2.2.2.RELEASE.jar	1	Highest	35
guava-16.0.jar	2	Highest	20
bootstrap-4.1.3.jar	1		13
select2-4.0.5.jar	1	High	13
select2-bootstrap-css-1.4.6.jar	1	High	13
jackson-databind-2.10.4.jar	1	Highest	39
ltxpdf-5.5.11.jar	1	High	36
bcprov-jdk15on-1.49.jar	14	Low	37
httpClient-4.5.12.jar	1	Highest	34
bootstrap.min.js	4		3
jquery.min.js	4		3
datatables-1.10.19.jar; jquery.js	4		3
jquery-3.3.1.jar; jquery.js	3		3
select2-4.0.5.jar; jquery-2.1.0.js	4		3
jquery-3.3.1.jar; jquery.slim.js	3		3
jquery-3.3.1.jar; jquery.min.js	3		3
jquery-3.3.1.jar; jquery.slim.min.js	3		3
bootstrap-4.1.3.jar; bootstrap.min.js	1		3
bootstrap-4.1.3.jar; bootstrap.bundle.js	1		3
bootstrap-4.1.3.jar; bootstrap.bundle.min.js	1		3
bootstrap-4.1.3.jar; bootstrap.js	1		3
bootstrap.sass:4.1.0	4		7
bootstrap:4.1.0	4		7
select2-4.0.5.jar; jquery-1.7.2.js	5		3
bootstrap-fileinput-4.4.8.jar; purify.js	5		3
bootstrap-fileinput-4.4.8.jar; purify.min.js	5		3

DPRF	Joker - Nota técnica	
-------------	-----------------------------	--

4.3 OWASP ZAP

Ferramenta funciona como scanner de segurança, utilizada para realização de testes de vulnerabilidade de aplicações WEB. Atualmente trata-se de um dos projetos mais ativos na comunidade de software livre.

Dado a complexidade para ajustes de configuração e dependências com sistemas internos, não foi possível efetuar a análise de vulnerabilidade com a ferramenta supracitada. Testes complementares podem ser realizados em ambientes de não produtivos da ferramenta.

4.4 Estrutura do projeto

O projeto possui boa organização estrutural e segregação por contexto funcional.

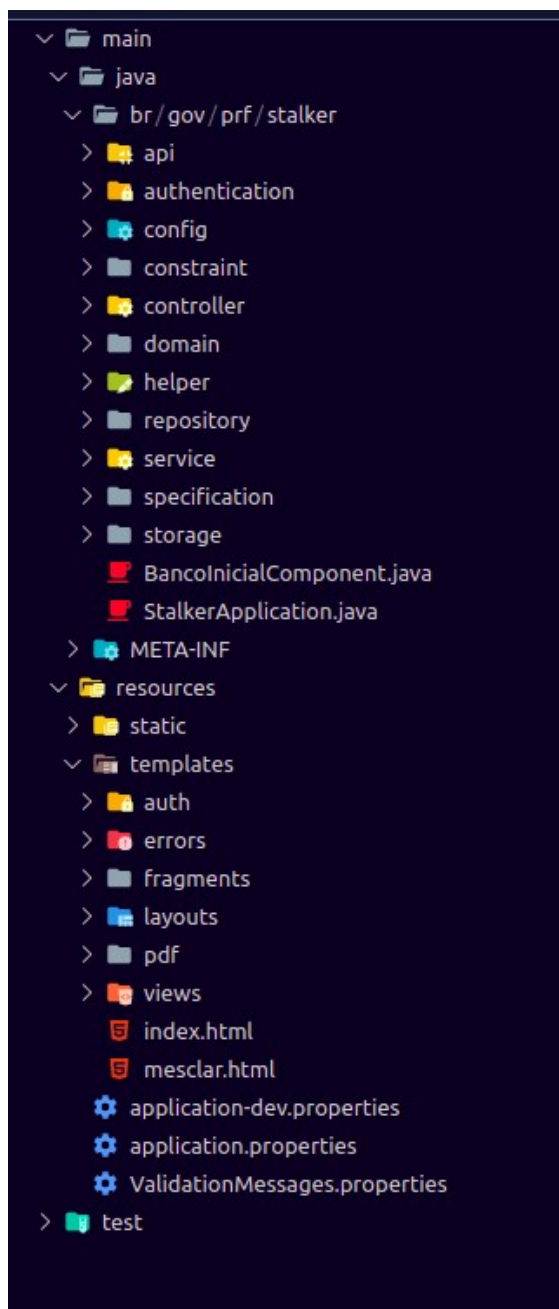


Figura 5: Estrutura do projeto

4.5 Manutenibilidade de código

Os relatórios apresentados pela ferramenta SonarQube demonstram poucos vícios adotados durante o processo de construção do software, a boa estrutura organizacional promove boas condições para manutenções corretivas/evolutivas, contudo a falta de testes de unidade dificultam a mensuração de impactos.

A aplicação apresenta boa estrutura arquitetural e padronização de código aliado a baixa complexidade ciclomática e a boa coesão, fatores estes que facilitam a manutenção do código.

4.6 Confiabilidade

O controle de transação efetuado na aplicação está sendo feito em operações controladas a nível de aplicação na camada superior a camada de acesso a dados (DAO – Data Access Object). Esta prática é a recomendada para que haja garantia das propriedades ACID do banco de dados, contudo, esta ação não está presente em todas as partes da aplicação.

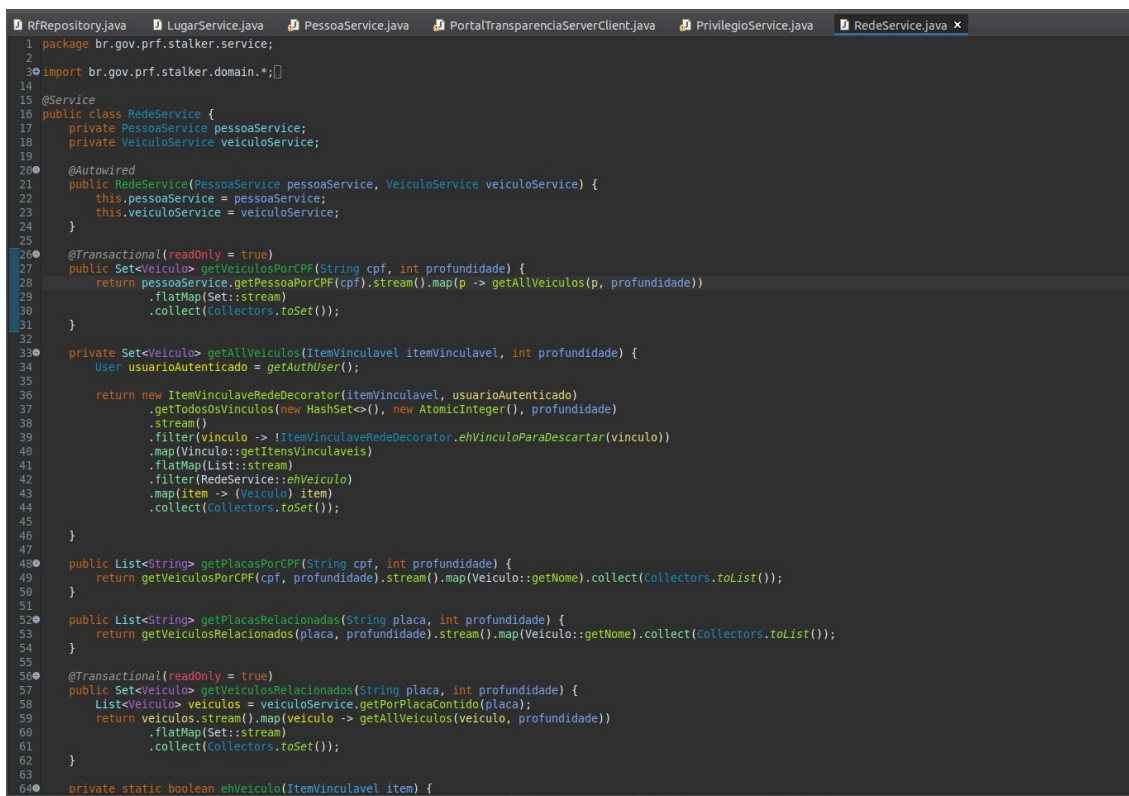
A manutenção da consistência de dados é algo fortemente desejado contudo esta não garante toda a confiabilidade da solução, é importante ressaltar a necessidade de revisão de dependências apresentada nos resultados no relatório da ferramentas de análise de dependências.

4.7 Performance e estabilidade

Não foi analisado o funcionamento da aplicação para avaliar demais requisitos não funcionais, recomenda-se a utilização de ferramentas de APM para mensurar performance e recursos de máquina utilizados.

A arquitetura monolítica citada no tópico não impede uma boa escalabilidade horizontal tendo em vista que não há manutenção de estado na aplicação.

Um ponto relevante a ser observado é a tratativa do controle transacional efetuado em processos de consultas em banco de dados, esta prática não é recomendada por degradação de performance e por não apresentar efetividade na consistência de dados.



```

1 package br.gov.pr.f.stalker.service;
2
3 import br.gov.pr.f.stalker.domain.*;
4
5 @Service
6 public class RedeService {
7     private PessoaService pessoaService;
8     private VeiculoService veiculoService;
9
10    @Autowired
11    public RedeService(PessoaService pessoaService, VeiculoService veiculoService) {
12        this.pessoaService = pessoaService;
13        this.veiculoService = veiculoService;
14    }
15
16    @Transactional(readOnly = true)
17    public Set<Veiculo> getVeiculosPorCPF(String cpf, int profundidade) {
18        return pessoaService.getPessoaPorCPF(cpf).stream().map(p -> getAllVeiculos(p, profundidade))
19            .flatMap(Set::stream)
20            .collect(Collectors.toSet());
21    }
22
23    private Set<Veiculo> getAllVeiculos(ItemVinculavel itemVinculavel, int profundidade) {
24        User usuarioAutenticado = getAuthUser();
25
26        return new ItemVinculavelDecorator(itemVinculavel, usuarioAutenticado)
27            .getTodosOsVinculos(new HashSet<>(), new AtomicInteger(), profundidade)
28            .stream()
29            .filter(vinculo -> !ItemVinculavelDecorator.ehVinculoParaDescartar(vinculo))
30            .map(vinculo::getItensVinculaveis)
31            .flatMap(List::stream)
32            .filter(RedeService::ehVeiculo)
33            .map(item -> (Veiculo) item)
34            .collect(Collectors.toSet());
35    }
36
37    public List<String> getPlacasPorCPF(String cpf, int profundidade) {
38        return getVeiculosPorCPF(cpf, profundidade).stream().map(Veiculo::getNome).collect(Collectors.toList());
39    }
40
41    public List<String> getPlacasRelacionadas(String placa, int profundidade) {
42        return getVeiculosRelacionados(placa, profundidade).stream().map(Veiculo::getNome).collect(Collectors.toList());
43    }
44
45    @Transactional(readOnly = true)
46    public Set<Veiculo> getVeiculosRelacionados(String placa, int profundidade) {
47        List<Veiculo> veiculos = veiculoService.getPorPlacaContido(placa);
48        return veiculos.stream().map(veiculo -> getAllVeiculos(veiculo, profundidade))
49            .flatMap(Set::stream)
50            .collect(Collectors.toSet());
51    }
52
53    private static boolean ehVeiculo(ItemVinculavel item) {
54
55    }
56
57    private static boolean ehVeiculo(ItemVinculavel item) {
58
59    }
60
61    private static boolean ehVeiculo(ItemVinculavel item) {
62
63    }
64

```

Figura 6: Controle transacional em processos de consulta em banco de dados

5 Recomendações

É altamente recomendado que seja efetuado refactoring de código dos bugs e vulnerabilidades de código apontadas pelo SonarQube , estas atividades certamente trarão maior confiabilidade a ferramenta e estabilidade em seu uso. Para os demais itens apontados pela ferramenta SonarQube durante o processo de análise de código são altamente desejáveis, contudo este processo de ajuste de código é moroso e trás consigo risco em potencial e está diretamente aliado a falta de cobertura de testes de unidade.

Ajustar as dependências que trazem maior risco para a aplicação é altamente recomendável, contudo este trabalho deve ser feito de forma analítica e cautelosa afim de não prejudicar a estabilidade da ferramenta. Sugere-se que o sistema seja disponibilizado em ambiente não produtivo para que seja possível a análise de vulnerabilidade (Pentest) e com base no relatório desta análise, que sejam feitos as associações dos relatórios de análise de dependências com os relatórios de análise de intrusão para que sejam avaliados as principais vulnerabilidades da aplicação e associá-las as dependências que oferecem tais riscos para os devidos ajustes. Esta recomendação visa a otimização e assertividade do trabalho de refactoring.

Recomenda-se a implantação de ferramentas de APM para que sejam criadas métricas e alarmes que auxiliem na continuidade do serviço em ambiente produtivo(monitoramento de processamento e memória por exemplo), tendo em vista que este tipo de ferramenta fornece mecanismos para determinarmos o comportamento da solução (auxiliam no refactoring de código) e também subsidia para o correto dimensionamento da infraestrutura.

A ferramenta não utiliza os serviços de RH e DPRF Segurança para efetuar a manutenção de usuários e serviços de

DPRF	Joker - Nota técnica	
-------------	-----------------------------	--

autenticação/autorização. Ao contrário disso dispõe de mecanismo redundante aos existentes para manutenção dos usuários e perfis, mesmo sendo utilizado mecanismos de segurança do framework Spring, recomenda-se a integração para evitar controles paralelos de informações no parque tecnológico da DPRF. Sugere-se a alteração destes recursos por pelos citados afim de centralizar, reutilizar e garantir a segurança da aplicação.

Recomenda-se a segregação dos endpoints em projeto específico para tal e que sua exposição passe a ser efetuada através da utilização do barramento de serviços (ESB) WSo2, evitando assim a utilização de recursos paralelos para proteção destes endpoints (OAuth2), aumentando o poder de escalabilidade e promovendo o desacoplamento do mesmo. Recomenda-se também que este projeto utilize da mesma segregação MVC demonstrada na figura 2.