Departamento de Polícia Rodoviária Federal

Projeto: Joker

Nota Técnica

Revisão	Descrição	Autor	Data
1.0	Construção do documento	Israel Branco	25/11/2020

1 Sumário

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-23-11-2020* no repositório https://git.prf/bruno.nobrega/Joker/-/tags/nota-tecnica-ctis-23-11-2020 com referência branch master na data de 23/11/2020.

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 stack Spring com suporte Spring Boot, 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* e não possui integração com nenhum outro sistema.

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

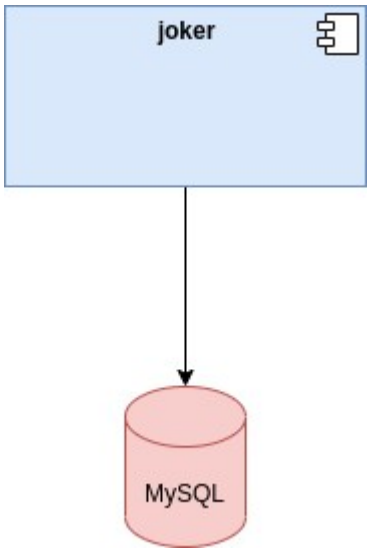

Figura 1: Diagrama de componentes

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

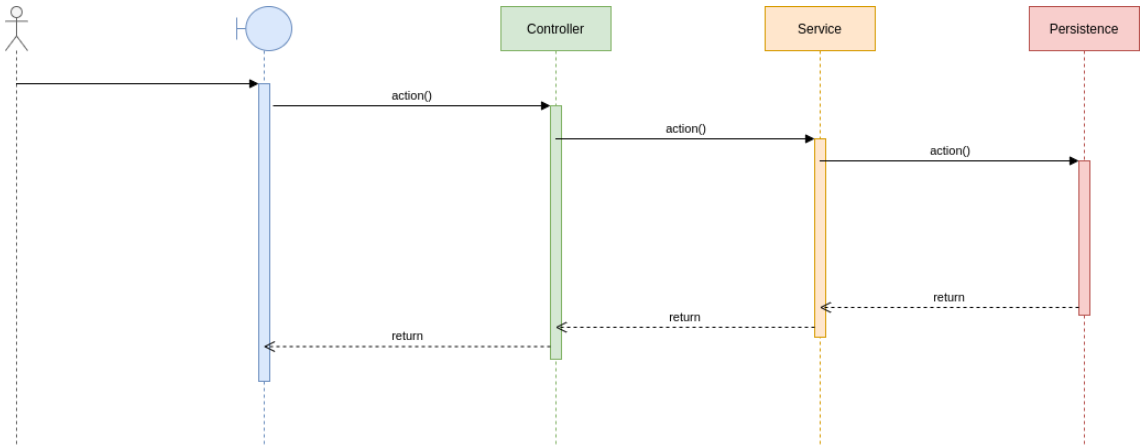

Figura 2: Diagrama de sequências - Fluxo principal

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.

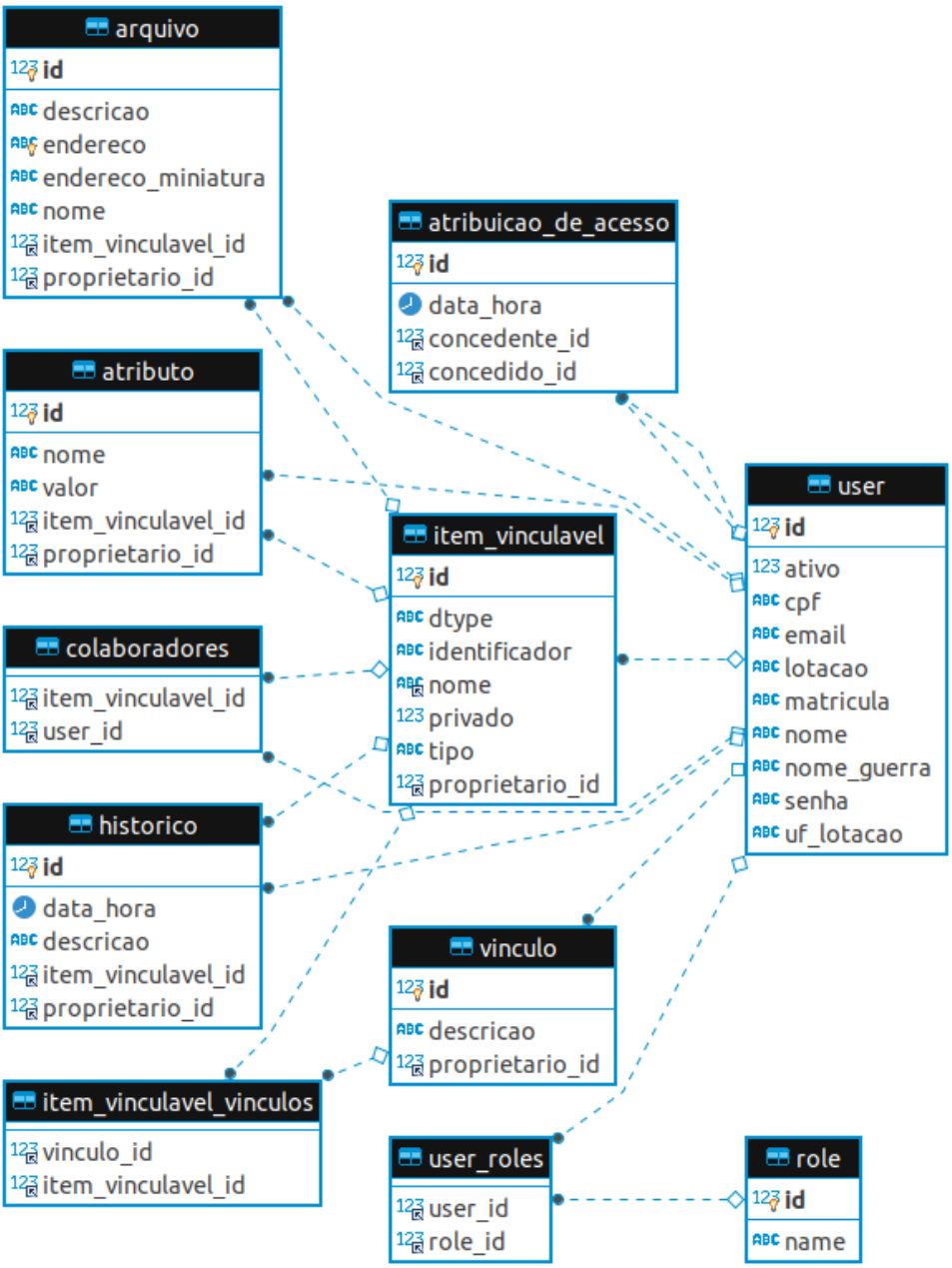

Figura 3: MER - Joker

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	1.5.9	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.

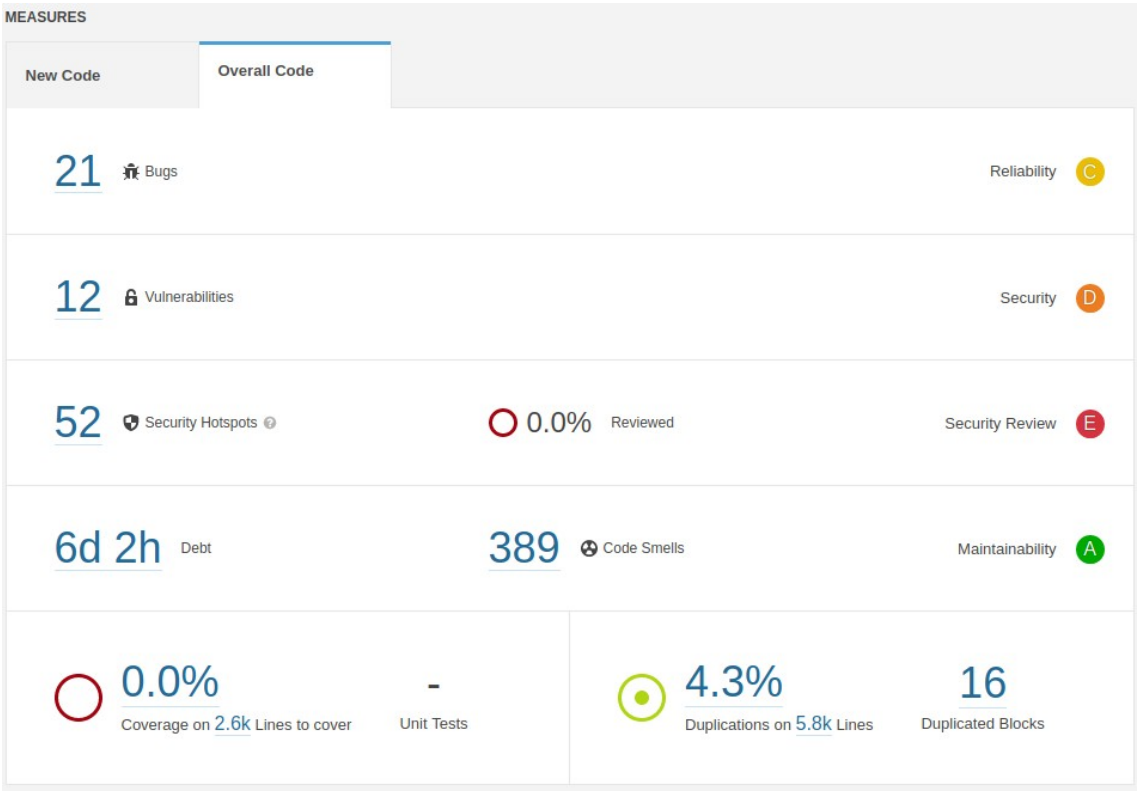

Figura 4: Análise estática de código

- 21 bugs;
- 12 vulnerabilidades de código;
- 52 violações de segurança;
- 389 violações de código ruim (complexidade cognitiva , complexidade ciclomática e débito técnico);
- 4.3% 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
snakeyaml-1.17.jar	1	Highest	28
spring-data-commons-1.13.9.RELEASE.jar	3	Highest	30
spring-security-core-4.2.3.RELEASE.jar	6	Highest	37
spring-security-web-4.2.3.RELEASE.jar	5	Highest	37
ognl-3.0.8.jar	1	Highest	19
tomcat-embed-core-8.5.23.jar	22	Highest	22
jackson-databind-2.8.10.jar	47	Highest	40
spring-boot-1.5.9.RELEASE.jar	1	Highest	34
mysql-connector-java-5.1.44.jar	5	Highest	38
spring-core-4.3.13.RELEASE.jar	10	Highest	28
bootstrap-4.0.0-alpha.jar	3		12
select2-4.0.3.jar	1	High	13
select2-bootstrap-css-1.4.6.jar	1	High	13
commons-compress-1.9.jar	1	Highest	39
bootstrap.min.js	4		3
jquery.min.js	4		3
jquery-2.0.0.jar: jquery.js	4		3
bootstrap-4.0.0-alpha.jar: bootstrap.js	4		3
select2-4.0.3.jar: jquery-2.1.0.js	4		3
jquery-2.0.0.jar: jquery-1.9.1.ajax_xhr.min.js	4		3
jquery-2.0.0.jar: jquery-basis.js	6		3
jquery-2.0.0.jar: jquery-migrate.js	2		3
bootstrap-fileinput-4.3.5.jar: purify.js	5		3
bootstrap-fileinput-4.3.5.jar: purify.min.js	5		3
jquery-2.0.0.jar: jquery-migrate.min.js	2		3
select2-4.0.3.jar: jquery-1.7.2.js	5		3
select2-4.0.3.jar: bootstrap.min.js	4		3
select2-4.0.3.jar: jquery.min.js	4		3

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.

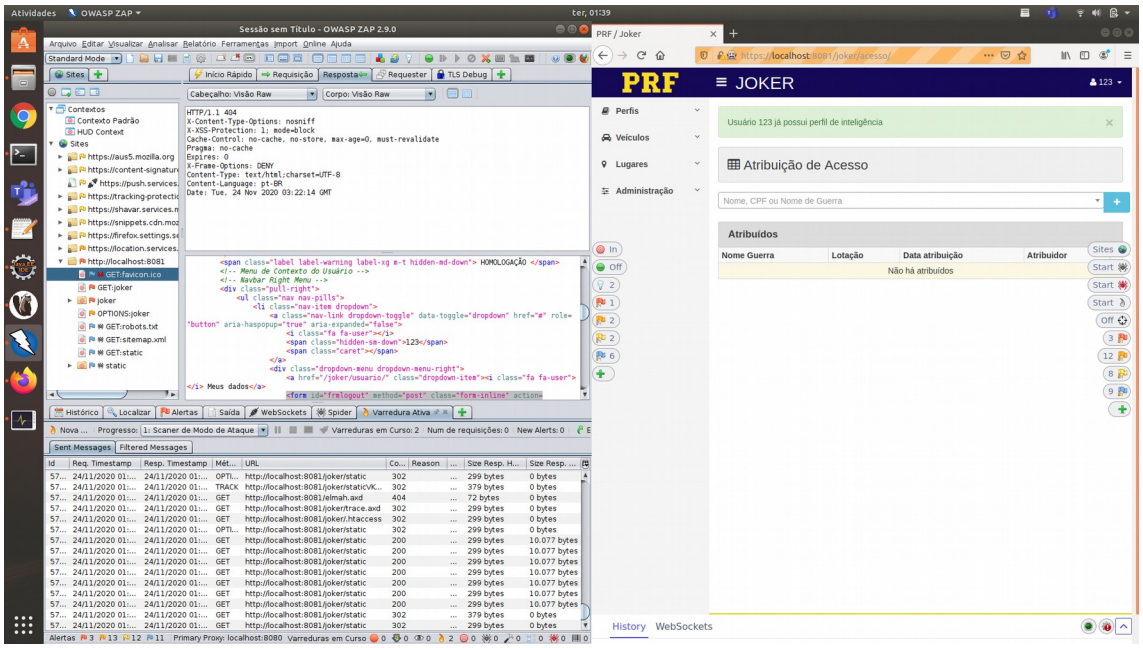

Figura 5: OWASP ZAP - ferramenta em execução

- 3 vulnerabilidade de severidade alta;
- 13 vulnerabilidade de severidade média;
- 30 vulnerabilidades de baixa média;
- 29 vulnerabilidades a nível informativo;

O relatório completo dos testes aplicados estão disponíveis no anexo I deste documento.

4.4 Estrutura do projeto

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

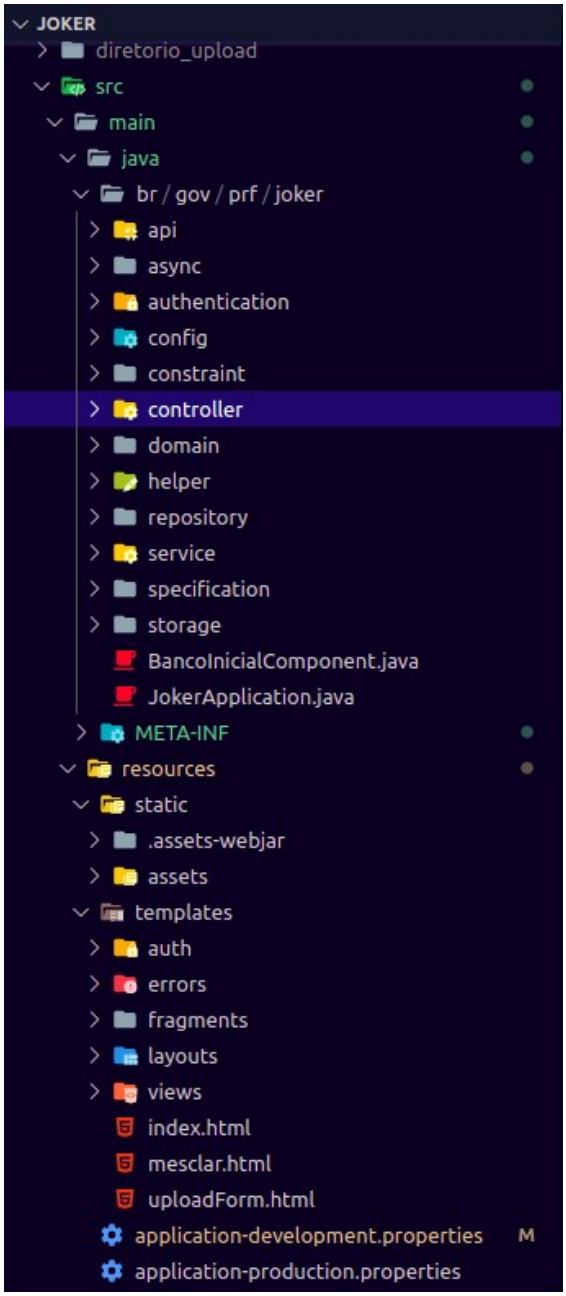

Figura 6: Estrutura do projeto

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 mesuração 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.

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 a associação dos relatórios de análise de dependências com os relatórios de análise de intrusão para que sejam analisados as principais vulnerabilidades da aplicação e associá-las as dependências que oferecem tais riscos para os devidos ajustes. Esta recomendação esta embasada na interseção de resultados das ferramentas utilizadas e na otimização e na 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 autenticação/autorização. Ao contrário disso dispõe de mecanismo redundante aos existentes para manutenção dos usuários e

criptografia com algoritmo MD5 para a senhas. Sugere-se a alteração destes recursos por pelos citados afim de centralizar, reutilizar e garantir a segurança da aplicação.