Departamento de Polícia Rodoviária Federal

Projeto: SPIA

Nota Técnica

Revisão	Descrição	Autor	Data
1.0	Construção do documento	Israel Branco	25/03/2021

1 Sumário

2 Considerações iniciais

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

Para a realização desta análise, gerou-se a tags *ctis-nota-tecnica-22-01-2021* nos repositórios https://git.prf/sistemas-nacionais/alerta-brasil/alerta-brasil-3/java-apps, https://git.prf/sistemas-nacionais/alerta-brasil/alerta-brasil-3/ui, https://git.prf/sistemas-nacionais/alerta-brasil/alerta-brasil-3/py-apps e https://git.prf/sistemas-nacionais/alerta-brasil/alerta-brasil-3/php-apps com referência branch master na data de 22/01/2021.

3 Apresentação do cenário atual

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

O ecossistema **SPIA** foi construído para funcionar essencialmente operando com protocolo HTTP/HTTPS tendo módulos construídos com arquitetura baseada em microsserviços e módulo que contempla um conjunto de APIs Rest para servir o front-end WEB de página única (*SPA)*. O ecossistema utiliza tecnologia Java, Python e PHP em sua estrutura backend com segregação de responsabilidades entre as camadas e coesão negocial, para a estrutura de frontend utiliza-se Typescript com o framework Angular. Para a estrutura de suporte a dados, utiliza os providers Redis para cache em memória, Postgres como banco relacional, H2 para testes de unidade e MongoDB como banco documental para armazenamento de histórico.

Os diagramas a seguir representam a concepção da arquitetura, o modelo de componentes, o modelo de sequências e o fluxo de dados gerenciado pelo cluster Apache Nifi ao qual os módulos estão dispostos respectivamente.

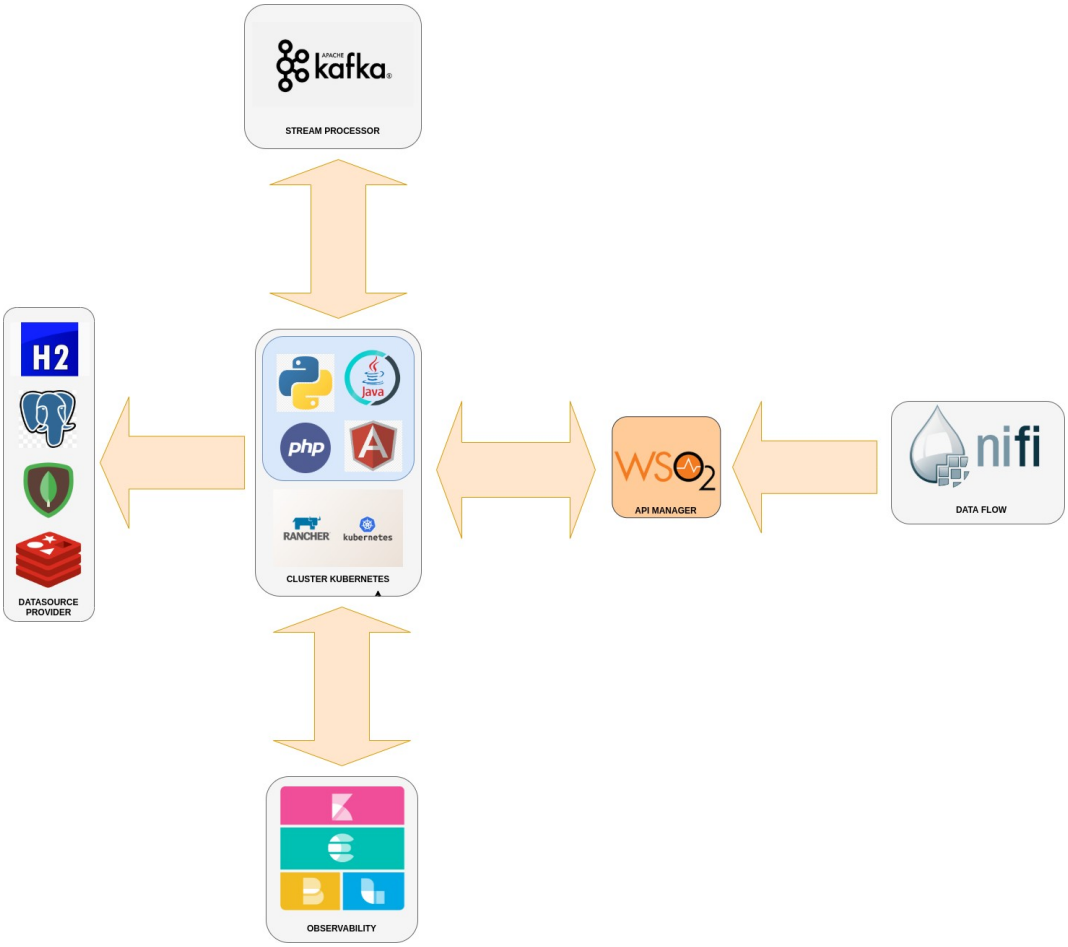

Figura 1: Concepção da arquitetura

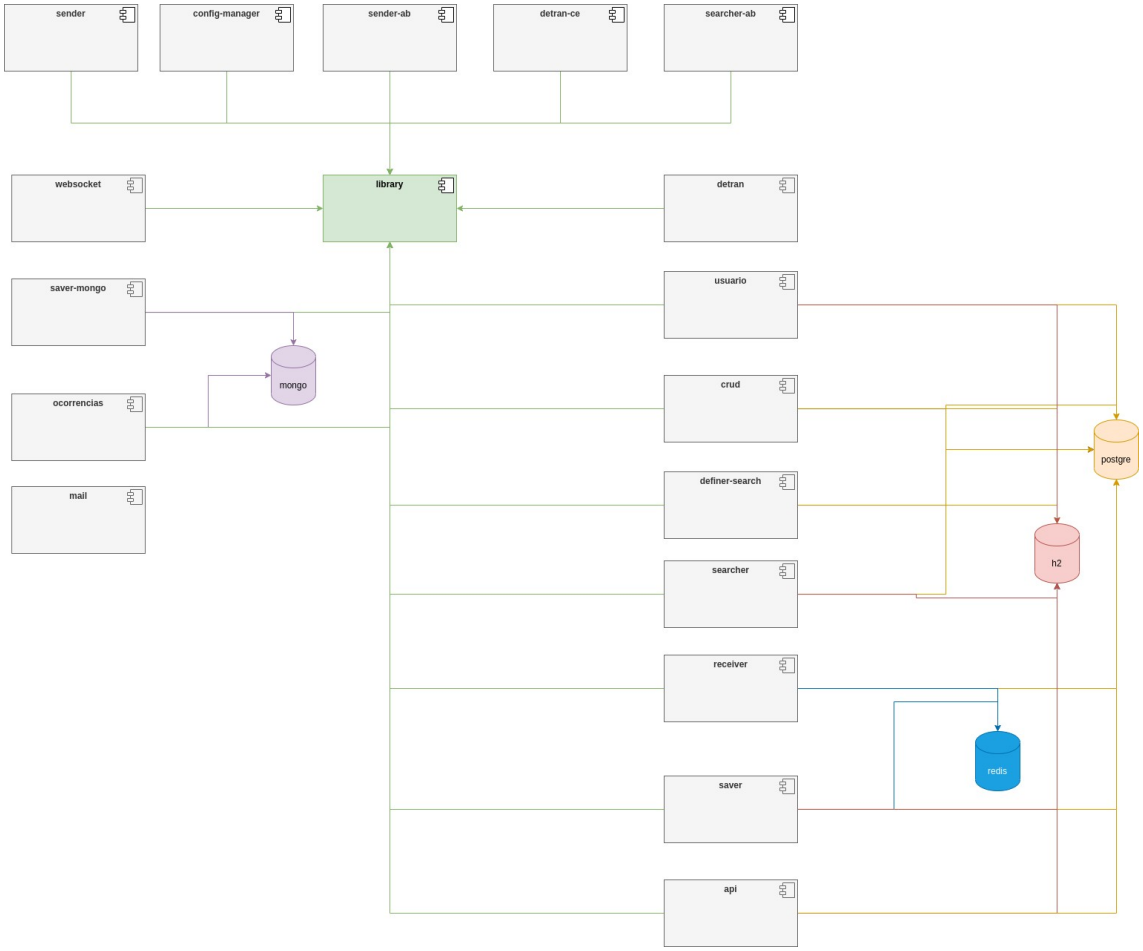

Figura 2: Diagrama de componentes - Microsserviços Java

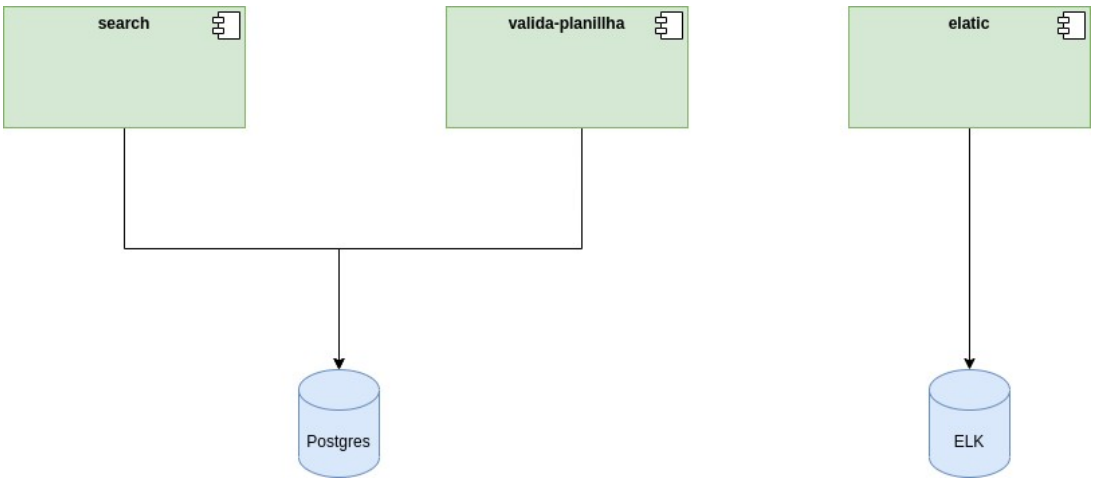

Figura 3: Diagrama de componentes - Microsserviços PHP

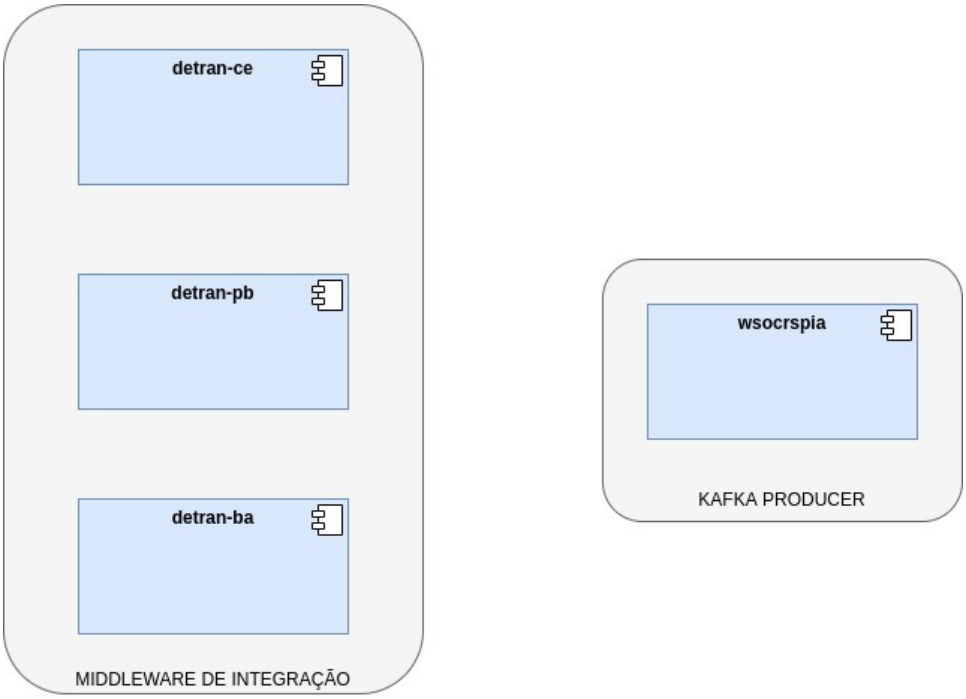

Figura 4: Diagrama de componentes - Microsserviços Python

A aplicação e os microsserviços Java utilizam o modelo MVC para a segregação de responsabilidades em camadas, as requisições http são oriundas da SPA Angular e de requisições do Apache Nifia os endpoints Rest. Os diagramas a seguir representam o fluxo encontrado durante o processo de análise da aplicação.

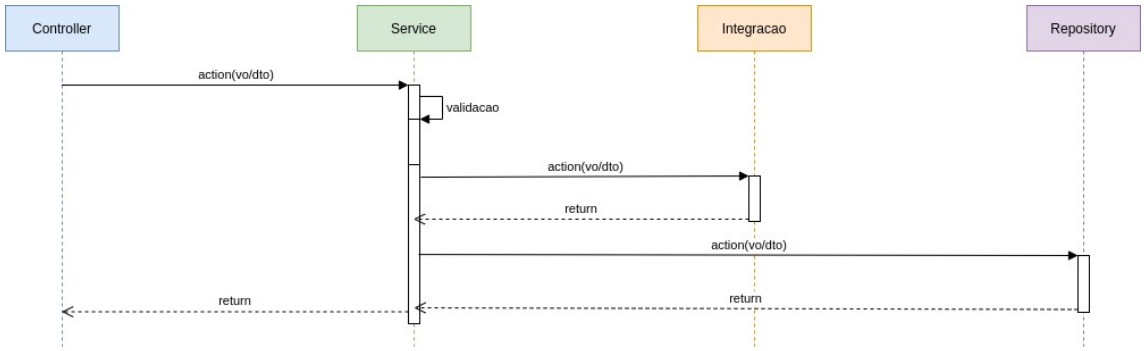

Figura 5: Diagrama de sequências - Fluxo principal - Java

As aplicações PHP Java utilizam framework Laraval, contudo não segue o modelo MVC em sua segregação de responsabilidade de camadas.

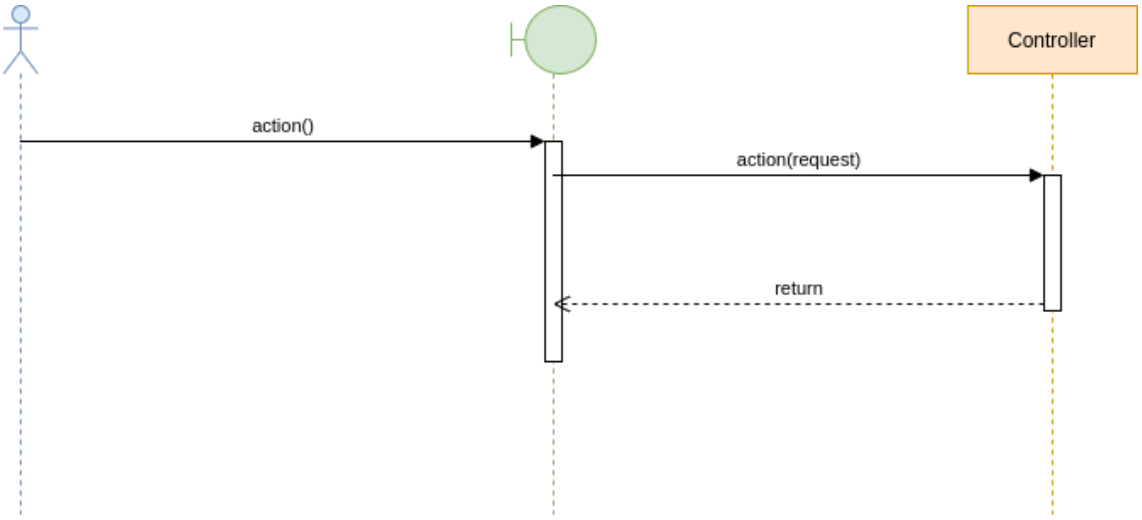

Figura 6: Diagrama de sequências - Fluxo principal - PHP

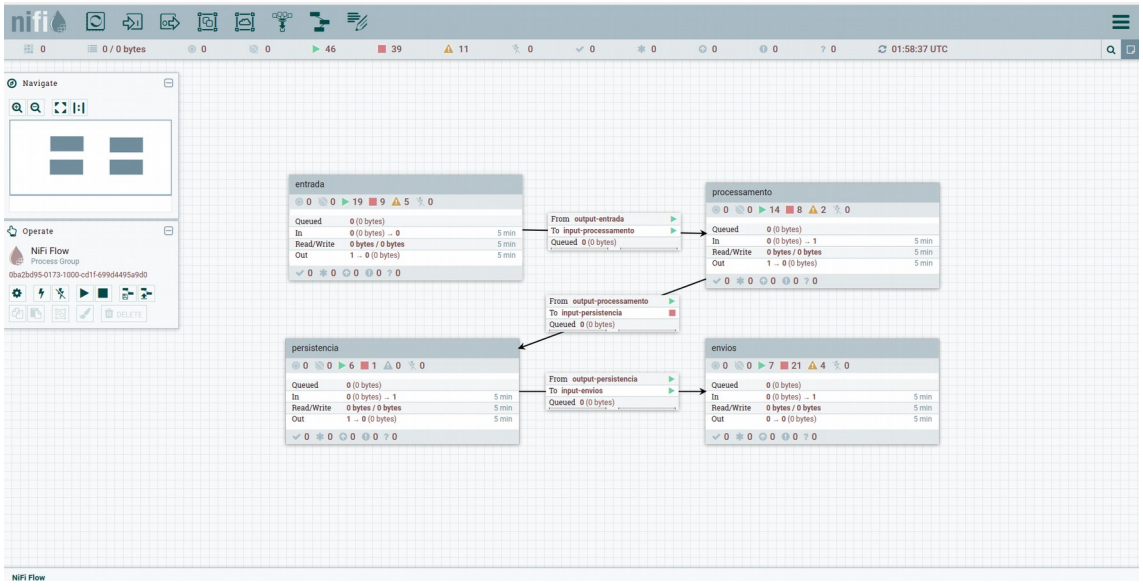

Figura 7: Apache Nifi - Macro fluxo

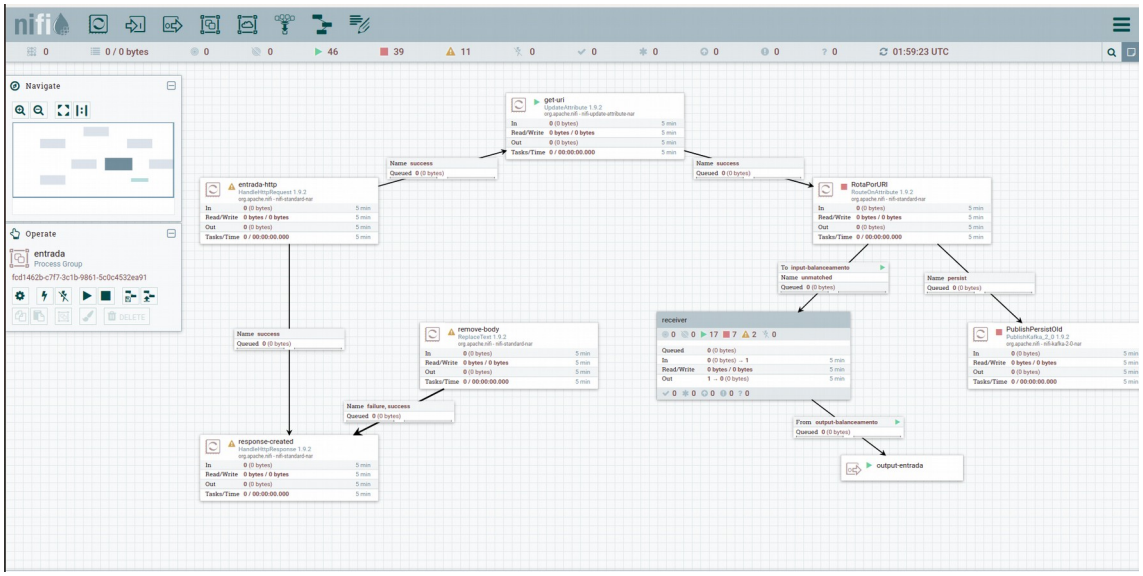

Figura 8: Apache Nifi - Fluxo de entrada de dados

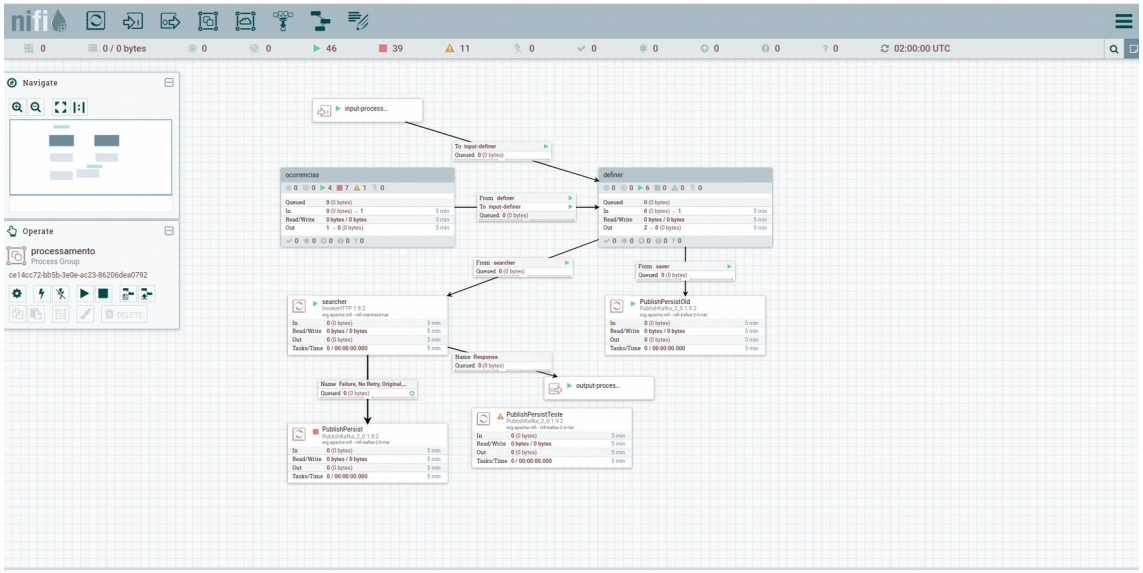

Figura 9: Apache Nifi - Fluxo de processamento de dados

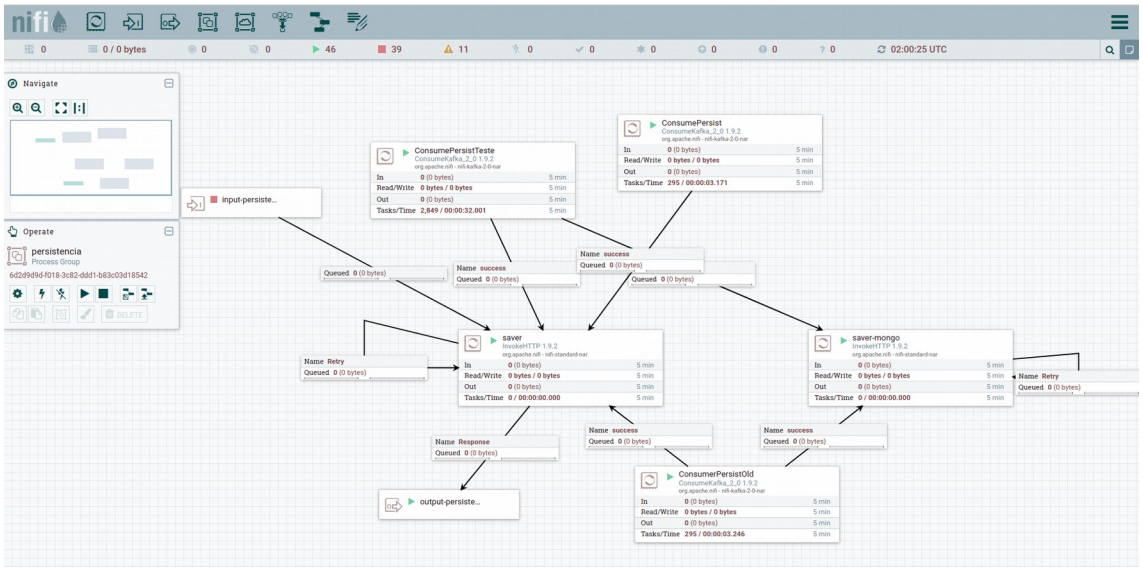

Figura 10: Apache Nifi - Fluxo de persistência de dados

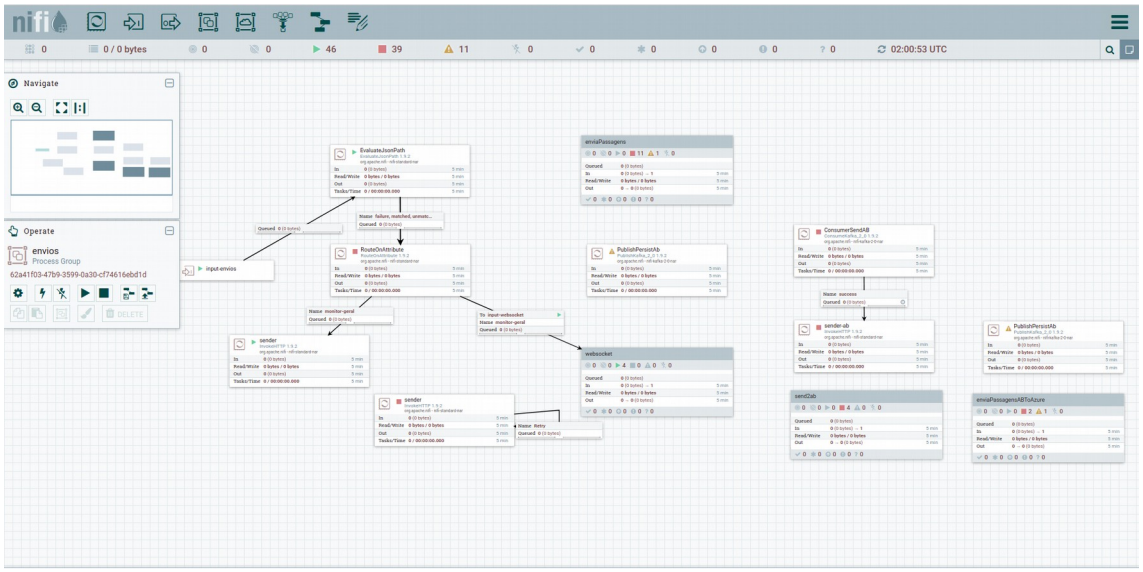

Figura 11: Apache Nifi - Fluxo de envio de dados

3.1 Tecnologias utilizadas

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

Nome	Versão	Utilização	Observação
Java	1.8	Linguagem de programação.	
Angular		Framework SPA	
Python	3.6.5	Linguagem de programação.	
PHP	7	Linguagem de programação.	
Docker	x	Ambiente de container	
Apache Nifi	x	Orquestração de fluxo de dados entre os microsserviços	
ELK	x	Elastic Stack	
PostgreSQL	x	Banco de dados relacional utilizado no contexto transacional	
MongoDB	x	Banco de dados documental utilizado para guarda de históricos	
Redis	x	Banco de dados em memória utilizado para cache	
H2	x	Banco de dados em memória utilizado para testes de unidade	

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. As análises seguem a sequência das aplicações Java (todas), PHP e Python.

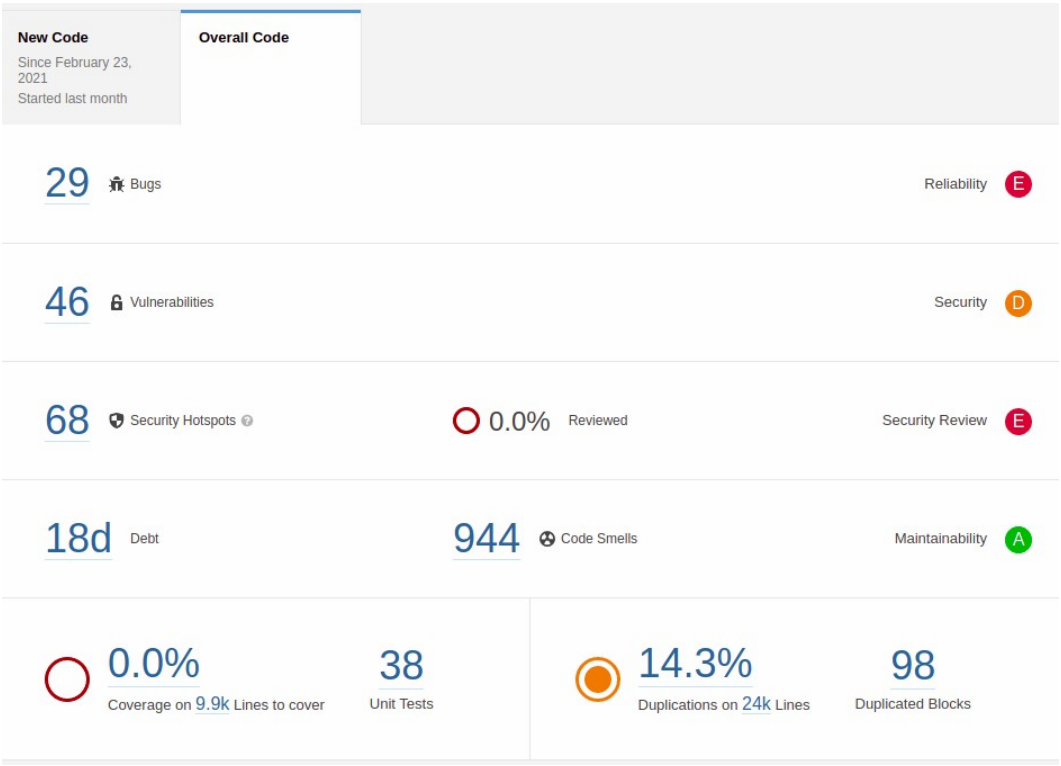

Figura 12: Análise estática de código - Java

- 29 bugs;
- 46 vulnerabilidades de código;
- 68 violações de segurança;
- 944 violações de código ruim (complexidade cognitiva , complexidade ciclomática e débito técnico);
- 14,3% de duplicidade de código;

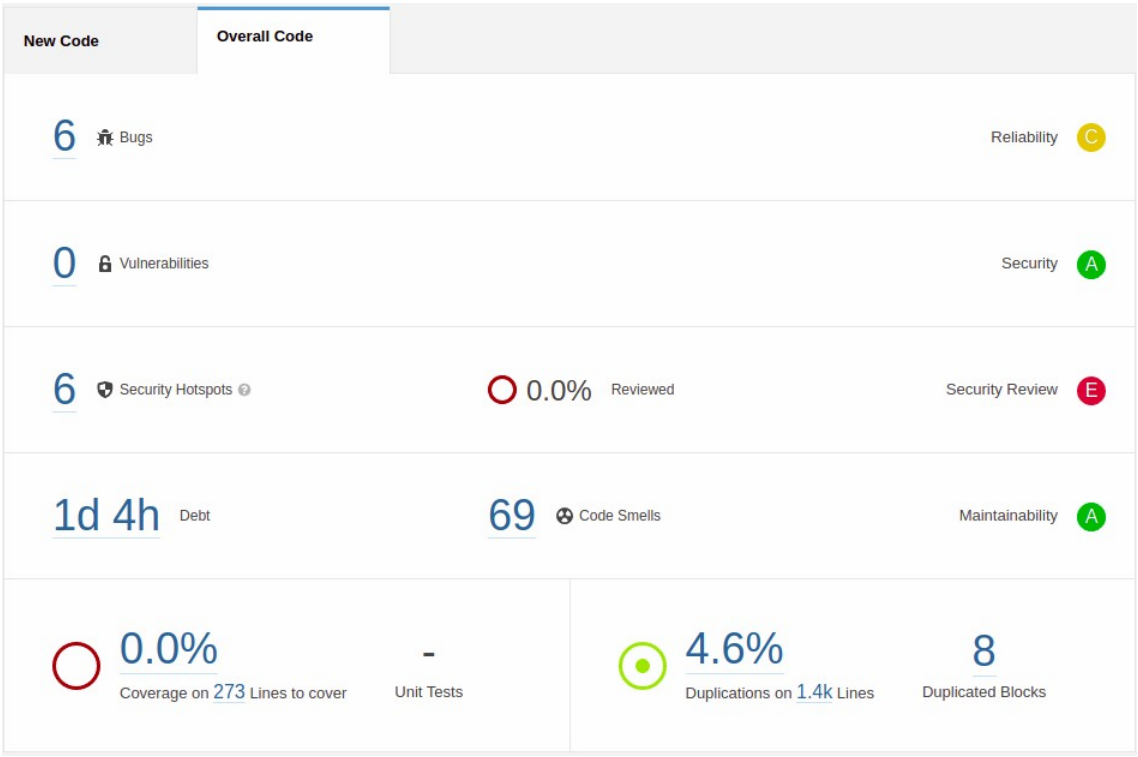

Figura 13: Análise estática de código – PHP Elastic

- 6 bugs;
- 0 vulnerabilidades de código;
- 6 violações de segurança;
- 69 violações de código ruim (complexidade cognitiva , complexidade ciclomática e débito técnico);
- 4,6% de duplicidade de código;

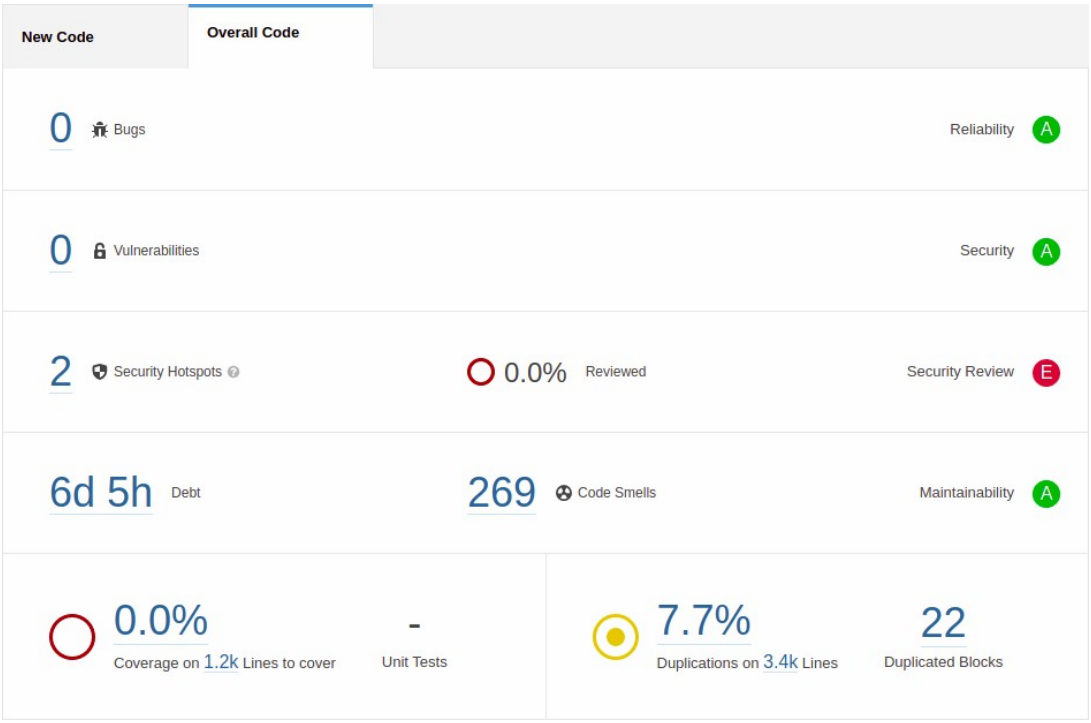

Figura 14: Análise estática de código – PHP Search

- 0 bugs;
- 0 vulnerabilidades de código;
- 2 violações de segurança;
- 269 violações de código ruim (complexidade cognitiva , complexidade ciclomática e débito técnico);
- 7,7% de duplicidade de código;

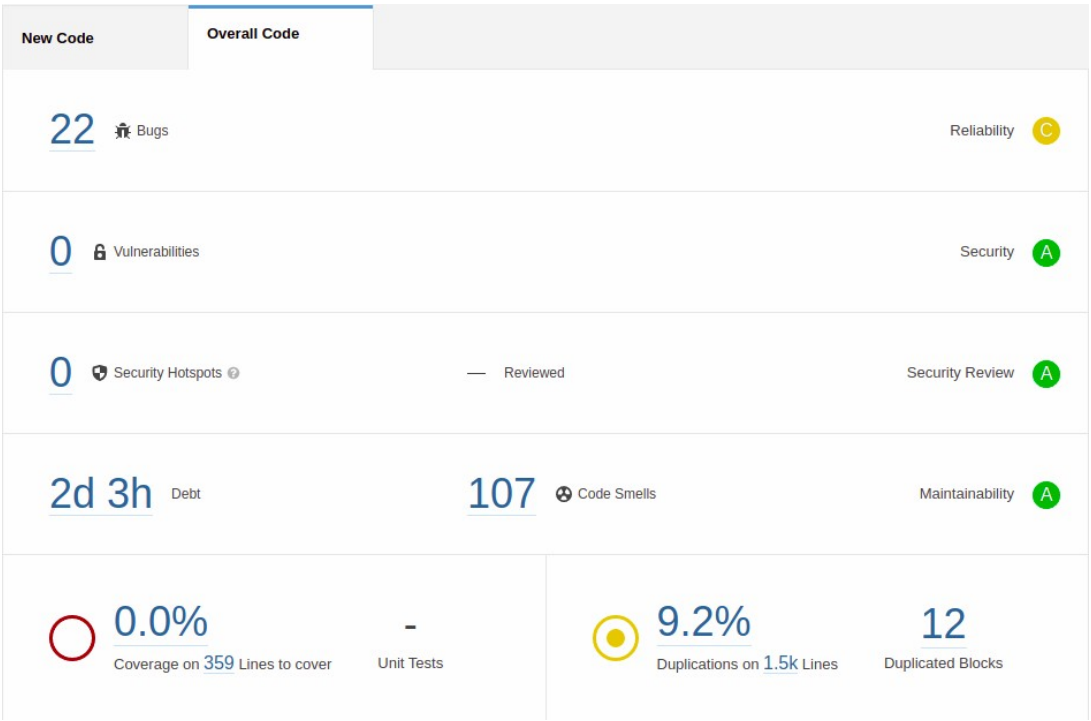

Figura 15: Análise estática de código – PHP ValidaPlanilha

- 22 bugs;
- 0 vulnerabilidades de código;
- 0 violações de segurança;
- 107 violações de código ruim (complexidade cognitiva , complexidade ciclomática e débito técnico);
- 9,2% de duplicidade de código;

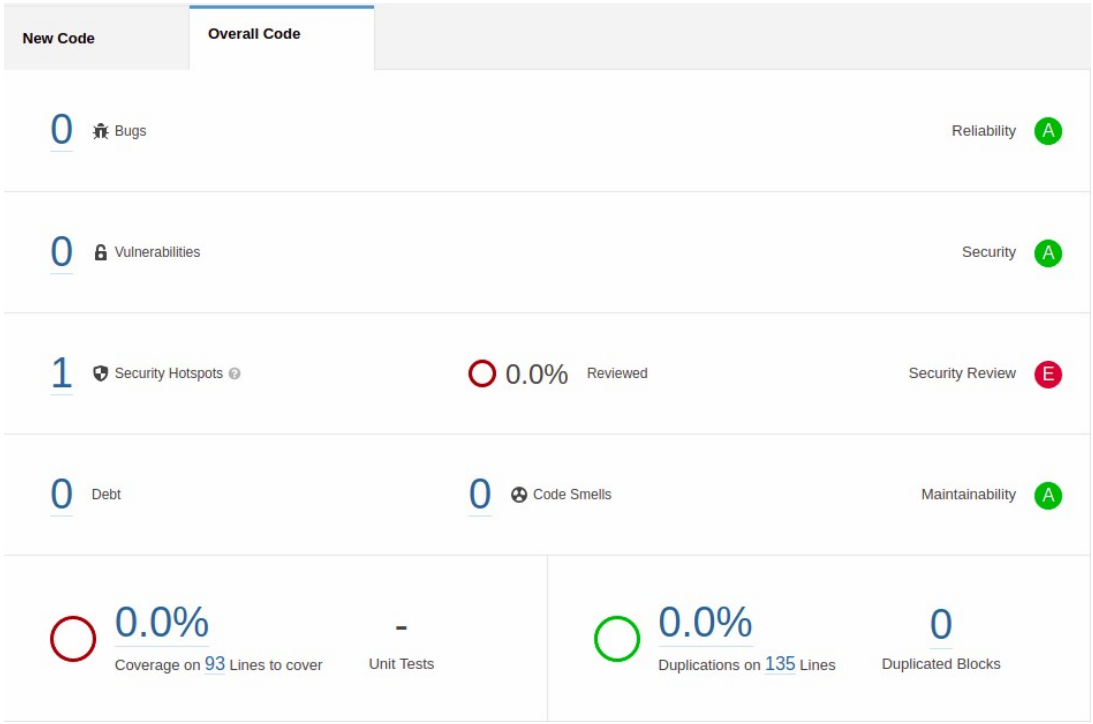

Figura 16: Análise estática de código – Python Detran Ba

- 0 bugs;
- 0 vulnerabilidades de código;
- 1 violações de segurança;
- 0 violações de código ruim (complexidade cognitiva , complexidade ciclomática e débito técnico);
- 0% de duplicidade de código;

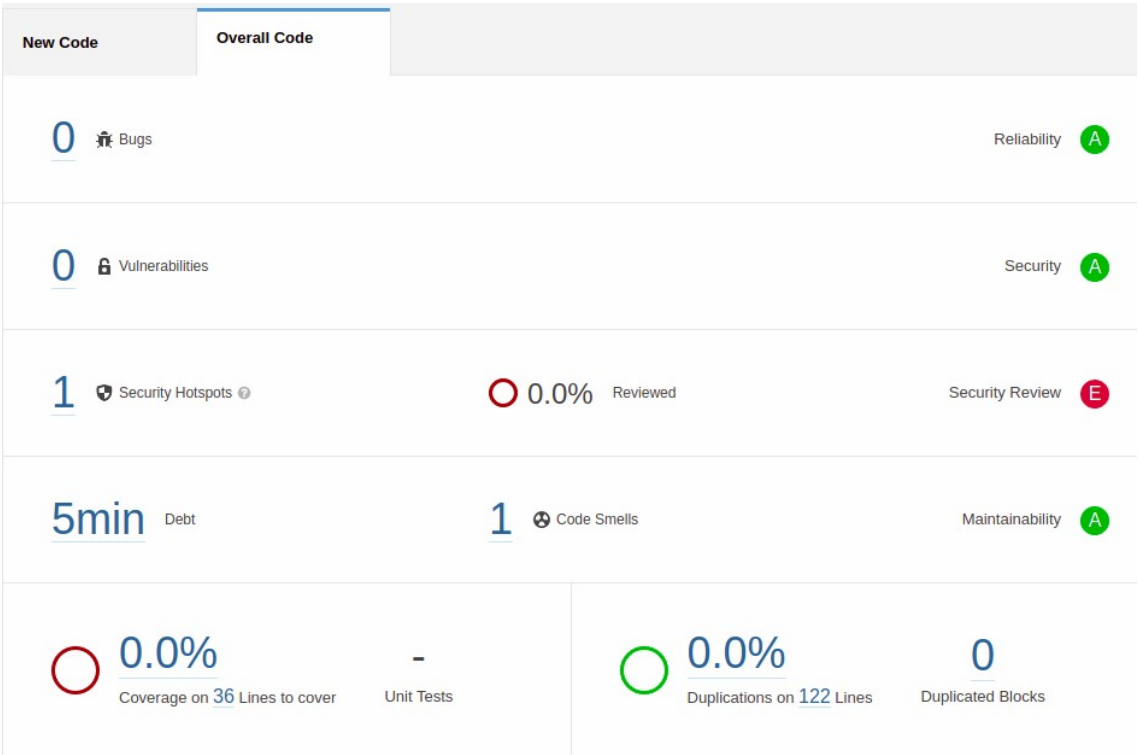

Figura 17: Análise estática de código – Python Detran Pb

- 0 bugs;
- 0 vulnerabilidades de código;
- 1 violações de segurança;
- 1 violação de código ruim (complexidade cognitiva , complexidade ciclomática e débito técnico);
- 0% de duplicidade de código;

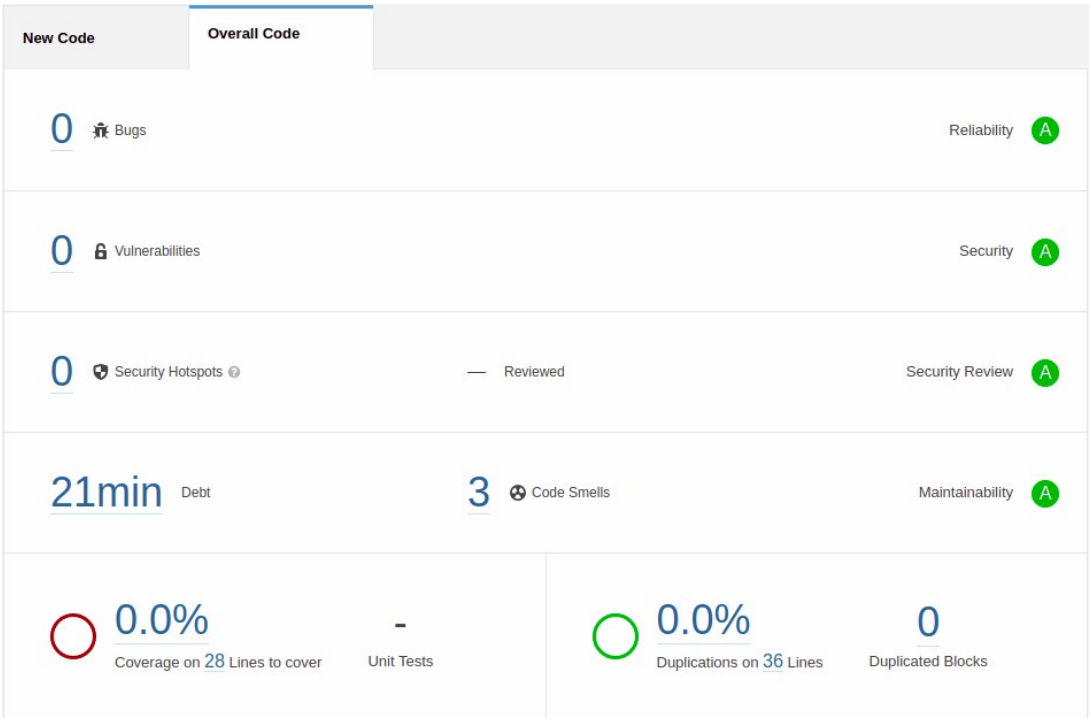

Figura 18: Análise estática de código – Python Detran Ce

- 0 bugs;
- 0 vulnerabilidades de código;
- 0 violações de segurança;
- 3 violações de código ruim (complexidade cognitiva , complexidade ciclomática e débito técnico);
- 0% de duplicidade de código;

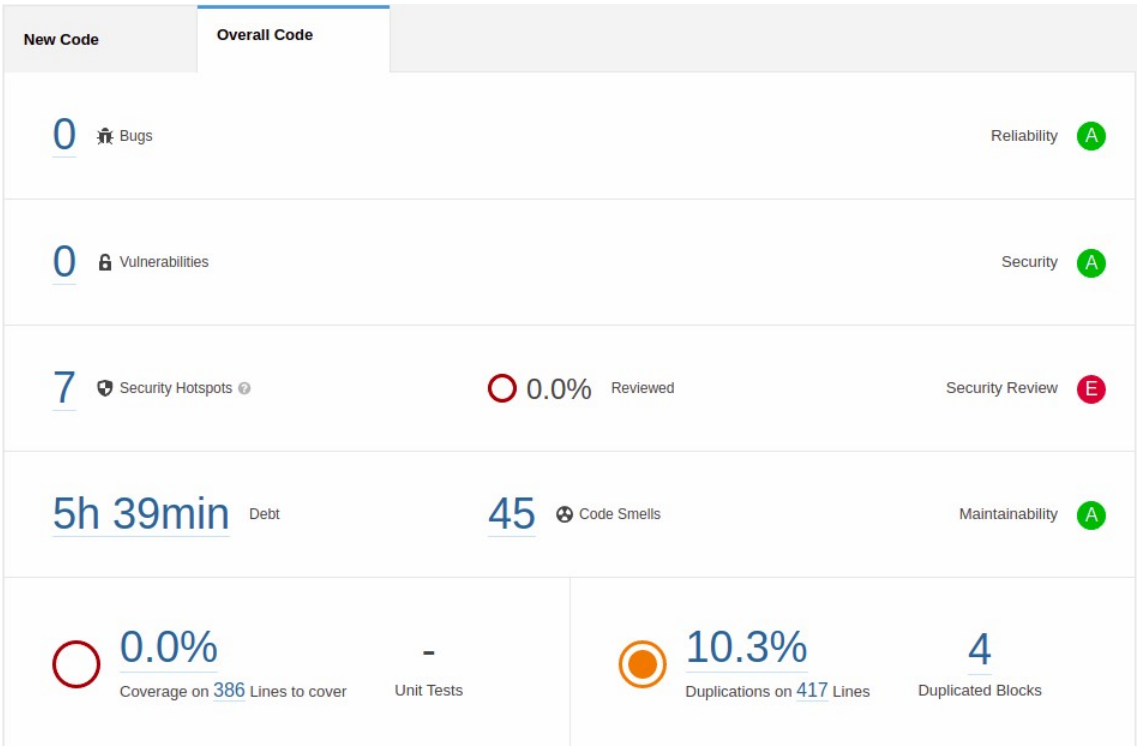

Figura 19: Análise estática de código – Python OCR

- 0 bugs;
- 0 vulnerabilidades de código;
- 7 violações de segurança;
- 45 violações de código ruim (complexidade cognitiva , complexidade ciclomática e débito técnico);
- 10,3% 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 Java. 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	Highest Severity	CVE Count	Confidence	Evidence Count
spring-security-oauth2-2.3.3.RELEASE.jar	HIGH	3	Highest	28
spring-security-core-5.4.2.jar	HIGH	1	Highest	30
jackson-mapper-asl-1.9.13.jar	CRITICAL	14	High	30
spring-security-jwt-1.0.9.RELEASE.jar	HIGH	1	Highest	32
bcprov-jdk15on-1.56.jar	CRITICAL	4	Low	47
CONFIG-MANAGER				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
log4j-api-2.11.2.jar	LOW	1	Highest	46
jackson-databind-2.9.8.jar	CRITICAL	49	Highest	41
tomcat-embed-core-9.0.19.jar	CRITICAL	13	Highest	36
spring-cloud-config-server-2.1.0.M3.jar	HIGH	3	Highest	31
spring-security-crypto-5.1.5.RELEASE.jar	HIGH	2	Highest	29
spring-security-rsa-1.0.7.RELEASE.jar	HIGH	1	Highest	33
bcprov-jdk15on-1.60.jar	MEDIUM	1	Low	49
httpclient-4.5.8.jar	MEDIUM	1	Highest	34
snakeyaml-1.23.jar	HIGH	1	Highest	26
spring-core-5.1.7.RELEASE.jar	HIGH	2	Highest	30
CRUD				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
log4j-api-2.11.2.jar	LOW	1	Highest	46
snakeyaml-1.23.jar	HIGH	1	Highest	26
jackson-databind-2.9.8.jar	CRITICAL	49	Highest	41
tomcat-embed-core-9.0.19.jar	CRITICAL	13	Highest	36
hibernate-core-5.3.10.Final.jar	HIGH	2	Low	39
dom4j-2.1.1.jar	CRITICAL	1	Highest	16
spring-core-5.1.7.RELEASE.jar	HIGH	2	Highest	30
postgresql-42.2.5.jar	HIGH	1	Highest	50

DEFINER-SEARCH				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
log4j-api-2.11.2.jar	LOW	1	Highest	46
snakeyaml-1.23.jar	HIGH	1	Highest	26
tomcat-embed-core-9.0.19.jar	CRITICAL	13	Highest	36
spring-core-5.1.7.RELEASE.jar	HIGH	2	Highest	30
hibernate-core-5.3.10.Final.jar	HIGH	2	Low	39
dom4j-2.1.1.jar	CRITICAL	1	Highest	16
netty-transport-4.1.36.Final.jar	CRITICAL	5	Highest	31
httpclient-4.5.6.jar	MEDIUM	1	Highest	34
postgresql-42.2.5.jar	HIGH	1	Highest	50
spring-security-rsa-1.0.7.RELEASE.jar	HIGH	1	Highest	33
bcprov-jdk15on-1.60.jar	MEDIUM	1	Low	49
jackson-databind-2.9.8.jar	CRITICAL	49	Highest	41
spring-cloud-config-client-2.1.0.M3.jar	HIGH	3	Highest	33
spring-security-crypto-5.1.5.RELEASE.jar	HIGH	2	Highest	29
DETRAN				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
log4j-api-2.11.2.jar	LOW	1	Highest	46
snakeyaml-1.23.jar	HIGH	1	Highest	26
jackson-databind-2.9.8.jar	CRITICAL	49	Highest	41
tomcat-embed-core-9.0.19.jar	CRITICAL	13	Highest	36
spring-core-5.1.7.RELEASE.jar	HIGH	2	Highest	30
httpclient-4.5.6.jar	MEDIUM	1	Highest	34
DETRAN-CE				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
log4j-api-2.11.2.jar	LOW	1	Highest	46
snakeyaml-1.23.jar	HIGH	1	Highest	26
jackson-databind-2.9.8.jar	CRITICAL	49	Highest	41
tomcat-embed-core-9.0.19.jar	CRITICAL	13	Highest	36
spring-core-5.1.7.RELEASE.jar	HIGH	2	Highest	30
httpclient-4.5.6.jar	MEDIUM	1	Highest	34
LIBRARY				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
spring-core-5.1.7.RELEASE.jar	HIGH	2	Highest	30
jackson-databind-2.9.8.jar	CRITICAL	49	Highest	41
tomcat-embed-core-9.0.19.jar	CRITICAL	13	Highest	36
hibernate-core-5.3.10.Final.jar	HIGH	2	Low	39
dom4j-2.1.1.jar	CRITICAL	1	Highest	16
log4j-api-2.11.2.jar	LOW	1	Highest	46
snakeyaml-1.23.jar	HIGH	1	Highest	26
MAIL				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
log4j-api-2.11.2.jar	LOW	1	Highest	46
snakeyaml-1.23.jar	HIGH	1	Highest	26
jackson-databind-2.9.8.jar	CRITICAL	49	Highest	41
tomcat-embed-core-9.0.19.jar	CRITICAL	13	Highest	36
spring-core-5.1.7.RELEASE.jar	HIGH	2	Highest	30
OCORRENCIAS				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
log4j-api-2.11.2.jar	LOW	1	Highest	46
snakeyaml-1.23.jar	HIGH	1	Highest	26
jackson-databind-2.9.8.jar	CRITICAL	49	Highest	41
tomcat-embed-core-9.0.19.jar	CRITICAL	13	Highest	36
spring-core-5.1.7.RELEASE.jar	HIGH	2	Highest	30
RECEIVER				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
log4j-api-2.11.2.jar	LOW	1	Highest	46
snakeyaml-1.23.jar	HIGH	1	Highest	26
jackson-databind-2.9.8.jar	CRITICAL	49	Highest	41
tomcat-embed-core-9.0.19.jar	CRITICAL	13	Highest	36
spring-core-5.1.7.RELEASE.jar	HIGH	2	Highest	30
netty-transport-4.1.36.Final.jar	CRITICAL	5	Highest	31
hibernate-core-5.3.10.Final.jar	HIGH	2	Low	39
dom4j-2.1.1.jar	CRITICAL	1	Highest	16
postgresql-42.2.5.jar	HIGH	1	Highest	50
httpclient-4.5.8.jar	MEDIUM	1	Highest	34
PRFServo-2.0.0.20.jar (shaded: org.apache.httpcomponents:h	MEDIUM	2	Highest	13

SAVER				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
log4j-api-2.11.2.jar	LOW	1	Highest	46
snakeyaml-1.23.jar	HIGH	1	Highest	26
jackson-databind-2.9.8.jar	CRITICAL	49	Highest	41
tomcat-embed-core-9.0.19.jar	CRITICAL	13	Highest	36
spring-core-5.1.7.RELEASE.jar	HIGH	2	Highest	30
hibernate-core-5.3.10.Final.jar	HIGH	2	Low	39
dom4j-2.1.1.jar	CRITICAL	1	Highest	16
netty-transport-4.1.36.Final.jar	CRITICAL	5	Highest	31
postgresql-42.2.5.jar	HIGH	1	Highest	50
httpclient-4.5.6.jar	MEDIUM	1	Highest	34
SEARCHER				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
log4j-api-2.11.2.jar	LOW	1	Highest	46
snakeyaml-1.23.jar	HIGH	1	Highest	26
jackson-databind-2.9.8.jar	CRITICAL	49	Highest	41
tomcat-embed-core-9.0.19.jar	CRITICAL	13	Highest	36
spring-core-5.1.7.RELEASE.jar	HIGH	2	Highest	30
hibernate-core-5.3.10.Final.jar	HIGH	2	Low	39
dom4j-2.1.1.jar	CRITICAL	1	Highest	16
httpclient-4.5.8.jar	MEDIUM	1	Highest	34
postgresql-42.2.5.jar	HIGH	1	Highest	50
SENDER				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
log4j-api-2.11.2.jar	LOW	1	Highest	46
snakeyaml-1.23.jar	HIGH	1	Highest	26
jackson-databind-2.9.8.jar	CRITICAL	49	Highest	41
tomcat-embed-core-9.0.19.jar	CRITICAL	13	Highest	36
spring-core-5.1.7.RELEASE.jar	HIGH	2	Highest	30
httpclient-4.5.6.jar	MEDIUM	1	Highest	34
SENDER-AB				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
log4j-api-2.11.2.jar	LOW	1	Highest	46
snakeyaml-1.23.jar	HIGH	1	Highest	26
jackson-databind-2.9.8.jar	CRITICAL	49	Highest	41
tomcat-embed-core-9.0.19.jar	CRITICAL	13	Highest	36
spring-core-5.1.7.RELEASE.jar	HIGH	2	Highest	30
httpclient-4.5.6.jar	MEDIUM	1	Highest	
SEARCHER-AB				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
log4j-api-2.11.2.jar	LOW	1	Highest	46
snakeyaml-1.23.jar	HIGH	1	Highest	26
jackson-databind-2.9.8.jar	CRITICAL	49	Highest	41
tomcat-embed-core-9.0.19.jar	CRITICAL	13	Highest	36
spring-core-5.1.7.RELEASE.jar	HIGH	2	Highest	30
httpclient-4.5.6.jar	MEDIUM	1	Highest	34
SAVER-MONGO				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
log4j-api-2.11.2.jar	LOW	1	Highest	46
snakeyaml-1.23.jar	HIGH	1	Highest	26
jackson-databind-2.9.8.jar	CRITICAL	49	Highest	41
tomcat-embed-core-9.0.19.jar	CRITICAL	13	Highest	36
spring-core-5.1.7.RELEASE.jar	HIGH	2	Highest	30
USUARIO				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
hibernate-core-5.3.10.Final.jar	HIGH	2	Low	39
dom4j-2.1.1.jar	CRITICAL	1	Highest	16
log4j-api-2.11.2.jar	LOW	1	Highest	46
snakeyaml-1.23.jar	HIGH	1	Highest	26
jackson-databind-2.9.8.jar	CRITICAL	49	Highest	41
tomcat-embed-core-9.0.19.jar	CRITICAL	13	Highest	36
spring-security-crypto-5.1.5.RELEASE.jar	HIGH	2	Highest	29
postgresql-42.2.5.jar	HIGH	1	Highest	50
spring-core-5.1.7.RELEASE.jar	HIGH	2	Highest	30
WEBSOCKET				
Dependency	**Highest Severity**	**CVE Count**	**Confidence**	**Evidence Count**
log4j-api-2.11.1.jar	LOW	1	Highest	46
snakeyaml-1.23.jar	HIGH	1	Highest	26
jackson-databind-2.9.8.jar	CRITICAL	49	Highest	41
tomcat-embed-core-9.0.14.jar	CRITICAL	16	Highest	36
spring-core-5.1.4.RELEASE.jar	HIGH	2	Highest	30

4.3 Manutenibilidade de código

Os relatórios apresentados pela ferramenta SonarQube demonstram poucos vícios adotados durante o processo de construção de todos os módulos (incluindo todas as tecnologias utilizadas), contudo a inexistência de testes de unidade e alta complexidade negocial envolvida no ecossistema SPIA dificuldade nos processos de manutenções corretivas/adaptativas e evolutivas da solução.

4.4 Confiabilidade

Para as aplicações Java, 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.

As aplicações PHP utilizam o banco somente para consulta e não necessitam de controle transacional, as aplicações Python basicamente trabalham como middlewares e também não necessitam de controle transacional.

4.5 Performance e escalabilidade

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 em todos os módulos que compõe a solução.

A arquitetura baseada em microsserviços citada no ópico 3 deste documento facilita o processo de escalabilidade

vertical/horizontal da solução. Esta facilidade está baseada no conceito de orquestração de containers aplicados dentro da ferramenta Kubernetes e se faz necessária dado o alto volume de dados processados diariamente.

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, 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.

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.

Recomenda-se que as aplicações em PHP sejam refatoradas para tirar o melhor proveito do modelo MVC fornecido pelo framework Laravel.

Para facilitar a inserção de novos egressos no time de desenvolvimento, recomenda a criação de um breve manual que discorra sobre os microsserviços construídos, o fluxo de dados aplicado no Apache Nifi e as demais ferramentas utilizadas entre os módulos.

5 Conclusão

Em sua totalidade o código produzido para a solução SPIA é de boa qualidade e com boa estruturação, a infraestrutura montada para suportar o alto volume é elástica o suficiente para suportar o crescimento da demanda de processamento de dados.

Por existir muita similaridade negocial com o sistema AB Brasil, recomenda-se a interseção entre ambos para que possa tirar melhor proveito da infraestrutura montada para suportar ambos os sistemas. Esta proposta certamente irá racionalizar o uso dos recursos computacionais e centralizar a manutenção de ambos os sistemas.