Ministério da Justiça

Projeto: DRCI Coopera

Nota Técnica

Revisão	Descrição	Autor	Data
1.0	Construção do documento	Israel Branco	23/04/2020

1 Sumário

2 Introdução

Este documento visa reportar o resultado da análise efetuada na aplicação DRCI-Coopera. 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 tag *coopera-ctis-nota-tecnica* no repositório https://gitlab.mj.gov.br/cgsis/coopera/-/tags/ctis-nota-tecnica com referência branch stable na data de 22/042020.

3 Apresentação do cenário atual

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

O sistema DRCI foi construído para funcionar em ambiente WEB, utiliza tecnologia Java e está 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) e utiliza banco de dados relacional *PostgreSQL.*

Sua arquitetura está composta por dois componentes sendo eles sgdrcri-entity responsável por efetuar o mapeamento ORM com o banco de dados e seus três schemas e o componente sgdrci-web responsável por conter as camadas MVC da aplicação.

O diagrama a seguir representa o modelo de componentes ao qual a aplicação está construída, suas dependências e seu modelo de comunicação.

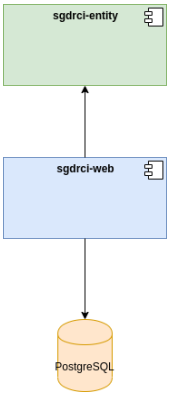

Figura 1: Diagrama de componentes

O diagrama a seguir demonstra o fluxo sequencial padrão de comunicação entre as camadas da aplicação em alto nível.

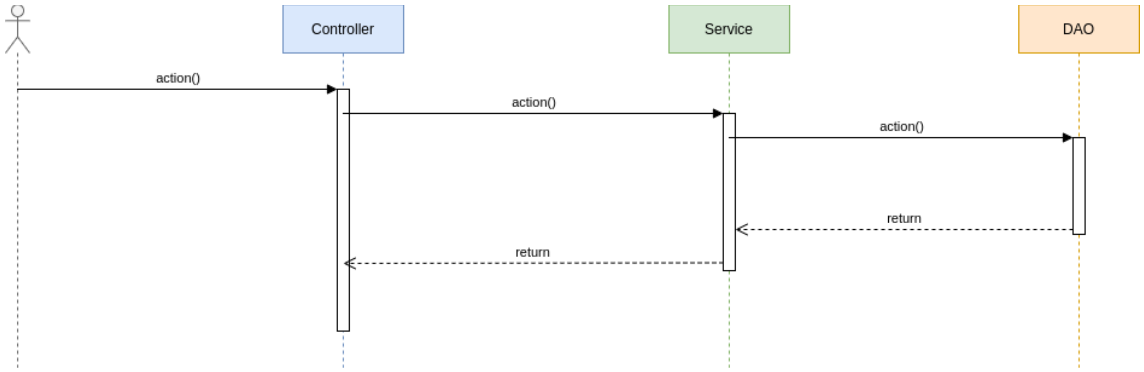

Figura 2: Diagrama de sequências

3.1 SGDRCI-Entity

Componente bem estruturado ao qual o pacote base contem as entidades mapeadas para o schema principal (sgdrci).

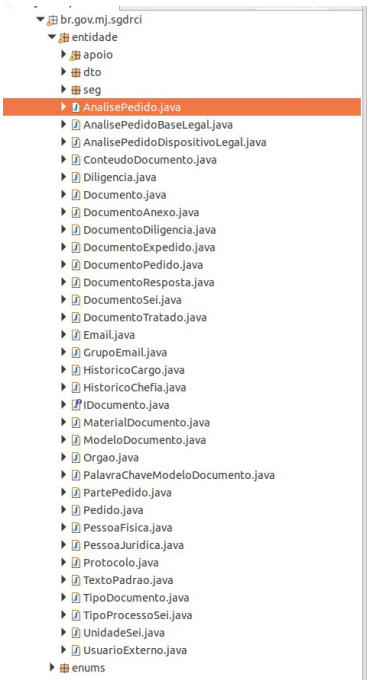

Figura 3: Estrutura do projeto sgdrci-entity

3.2 SGDRCI-Web

Este componente contém as camadas de visão, serviço e persistência da aplicação. Também possui pacotes e conjunto de classes utilitárias e de integração com o serviço SEI.

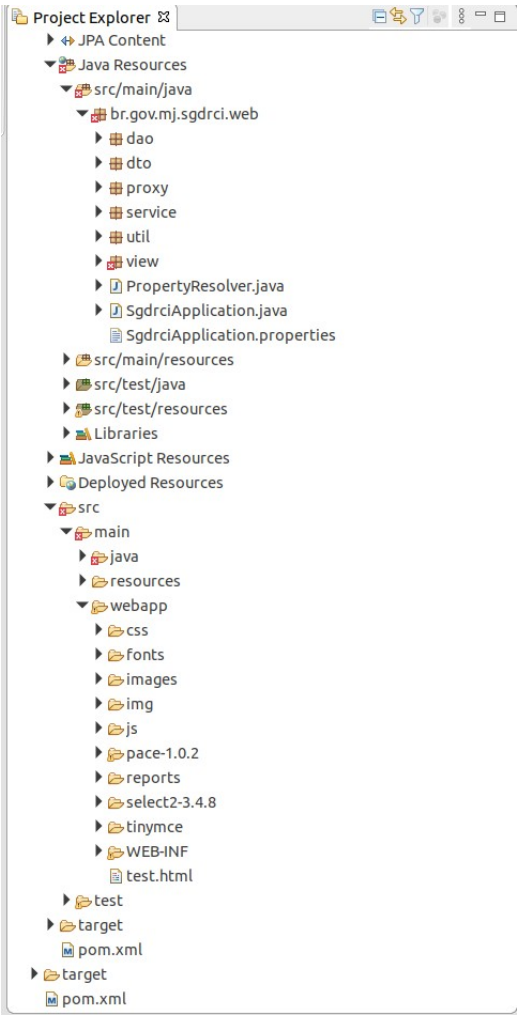

Figura 4: Estrutura do projeto sgdcri-web

O projeto apresenta boa estrutura e boa coesão na segregação dos pacotes e estruturação web.

3.3 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.	
Hibernate	4.3.7	Framework ORM.	
Hibernate-Envers	4.3.7	Módulo Hibernate utilizado para auditoria na manipulação de dados.	
Wildfly	8.x	Servidor de aplicação JEE.	Utiliza container EJB, CDI e Servlet.
Apache Wicket	6.19.0	Framework orientado a componentes utilizado no desenvolvimento de aplicações Java Web.	
Junit	4.11	Utilizado para a criação de testes automatizados.	
Jboss Arquillian	1.1.8	Framework para realização testes de integração.	
JasperReports	5.6.1	Componente utilizado para criação de relatórios.	
Apache POI	3.5	Componente utilizado para manipular documentos baseado em	

		formato MS Office.	
Mockito	1.9.5	Componente utilizado para viabilizar a utilização mocks em testes unitários.	
SEI-Client	0.3.4	Componente para integração com SEI	Componente desenvolvido pelo MJ.
PostgreSQL		Banco de dados relacional	

3.4 Modelagem de dados

A estrutura de banco de dados esta composta pela utilização de de três schemas apoio, seg e sgdrci sendo este último o schema principal da aplicação.

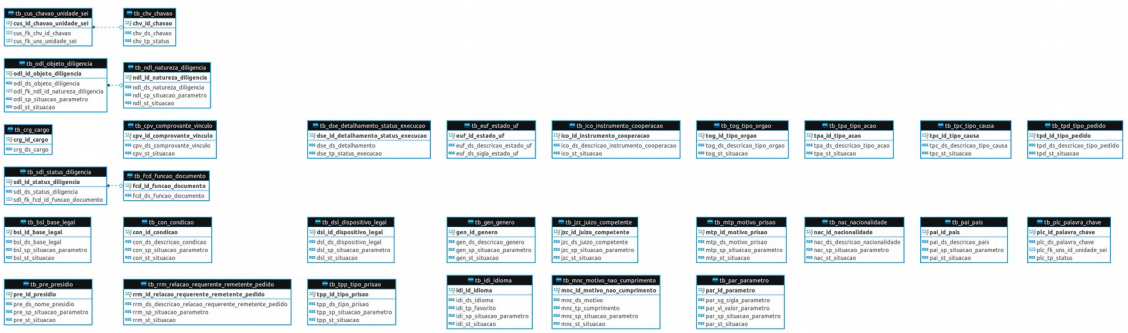

Figura 5: Schema Apoio

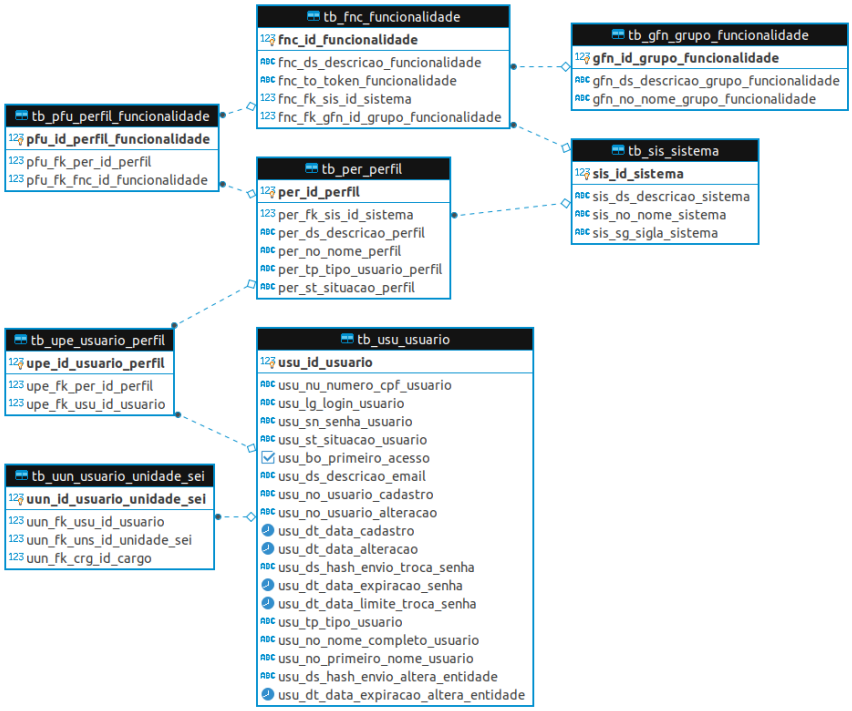

Figura 6: Schema Seg

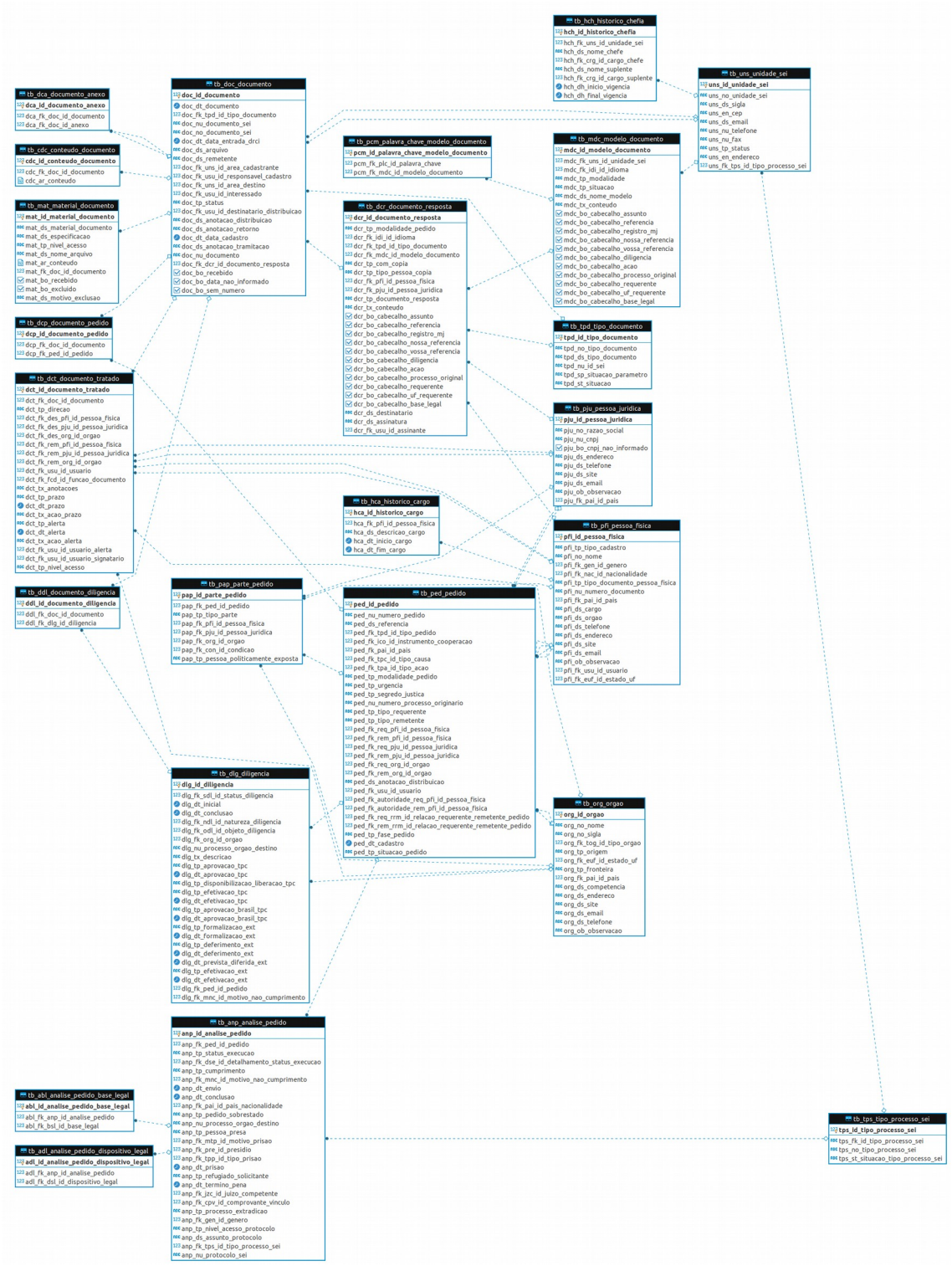

Figura 7: Schema sgdrci

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 do Ministério da Justiça, contudo foram utilizadas as regras padrões de análise da ferramenta. Os resultados foram os seguintes para as dois componentes da solução:

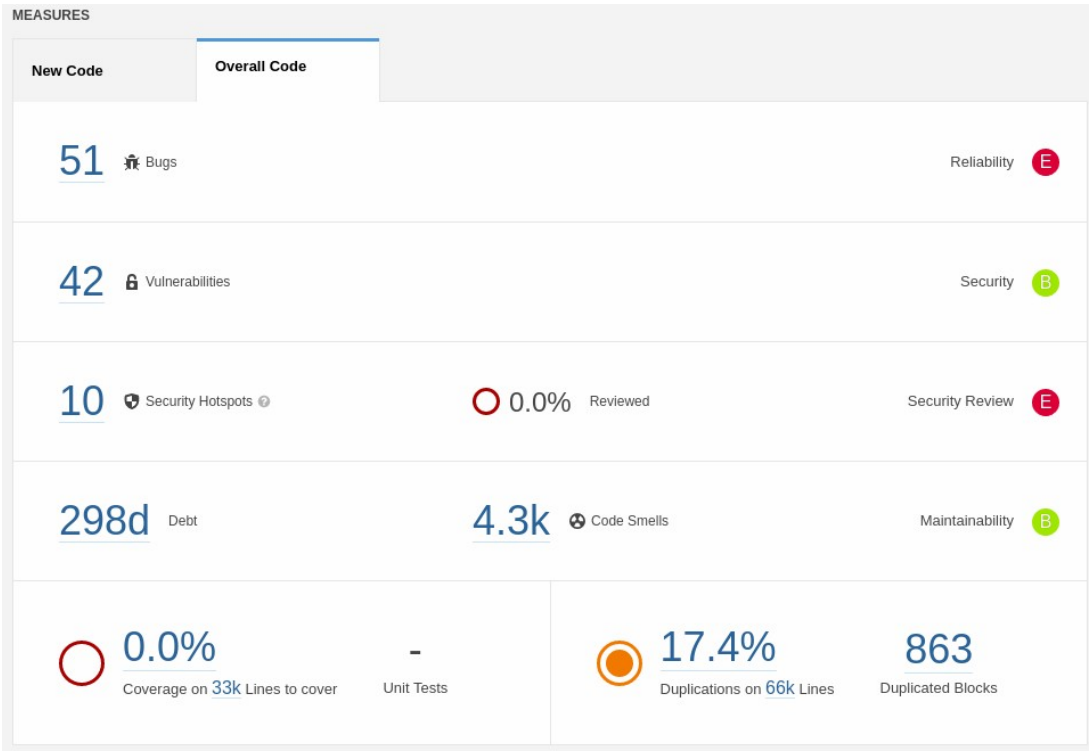

Figura 8: Análise estática de código

- 51 bugs;
- 42 vulnerabilidades;
- 10 vulnerabilidade de acesso de segurança
- 4.3 mil de violações de código ruim e débito técnico
- 863 registros que correspondem a 17.4% de duplicidade de código;

A ferramenta apresenta 0% de cobertura de testes contudo o código fonte apresenta somente 4 artefatos de testes (InteressadoDTOTest, ConstantsTest, SeiApiTest e PropertyTest).

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 no projeto, a seguir temos as principais informações extraídas desta análise.

Dependency	Highest Severity	CVE Count	Confidence	Evidence Count
WEB				
bsh-2.0b4.jar	HIGH	1	Highest	23
commons-collections4-4.0.jar	HIGH	1	Highest	37
wicket-datetime-6.23.0.jar	CRITICAL	2	Highest	35
bootstrap-3.2.0.jar	MEDIUM	8		13
jquery-1.11.1.jar	medium	3		13
guava-14.0.1.jar	MEDIUM	1	Highest	21
protobuf-java-2.4.1.jar	HIGH	1		25
jquery-ui-1.10.4.jar	MEDIUM	1	High	13
wicket-core-6.19.0.jar	CRITICAL	3	Highest	46
jasperreports-5.6.1.jar	HIGH	4	Low	29
commons-collections-3.2.1.jar	CRITICAL	3	Highest	35
castor-1.2.jar	MEDIUM	1	Highest	19
jackson-databind-2.1.4.jar	CRITICAL	35	Highest	38
wicketstuff-editable-grid-6.17.0.jar	CRITICAL	4	High	23
poi-3.5-FINAL.jar	HIGH	8	Highest	24
log4j-1.2.16.jar	CRITICAL	1	Highest	29
postgresql-9.0-801.jdbc4.jar	CRITICAL	3	Highest	18
dom4j-1.6.1.jar	HIGH	1	Highest	25
resteasy-jaxrs-3.0.11.Final.jar	HIGH	2		21
httpclient-4.2.6.jar	MEDIUM	2	Highest	34
jackson-core-asl-1.6.3.jar	CRITICAL	3	High	32
jackson-mapper-asl-1.6.3.jar	CRITICAL	13	High	30
axis-1.4.jar	HIGH	4	Highest	15
bcprov-jdk14-136.jar	HIGH	16	Highest	23
jQuery-2.1.4.min.js	medium	3		3
jquery-ui.min.js	MEDIUM	1		3
bootstrap.min.js	MEDIUM	4		3
bootstrap.js	MEDIUM	4		3
wicket-datetime-6.23.0.jar: yuiloader.js	MEDIUM	3		3
wicket-datetime-6.23.0.jar: yuiloader-min.js	MEDIUM	3		3
wicket-datetime-6.23.0.jar: yahoo.js	MEDIUM	3		3
wicket-datetime-6.23.0.jar: yahoo-min.js	MEDIUM	3		3
wicket-datetime-6.23.0.jar: dom.js	MEDIUM	3		3
wicket-datetime-6.23.0.jar: dom-min.js	MEDIUM	3		3
wicket-datetime-6.23.0.jar: event.js	MEDIUM	3		3
wicket-datetime-6.23.0.jar: event-min.js	MEDIUM	3		3
wicket-datetime-6.23.0.jar: yahoo-dom-event.js	MEDIUM	3		3
wicket-datetime-6.23.0.jar: calendar.js	MEDIUM	3		3
wicket-datetime-6.23.0.jar: calendar-min.js	MEDIUM	3		3
wicket-bootstrap-core-0.9.7.jar: jquery-migrate-1.2.1.js	medium	1		3
wicket-bootstrap-core-0.9.7.jar: jquery-migrate-1.2.1.min.js	medium	1		3
wicket-bootstrap-core-0.9.7.jar: bootlint.js	medium	3		3
wicket-bootstrap-extensions-0.9.7.jar: bootstrap.min.js	MEDIUM	4		3
jquery-ui-1.10.4.jar: jquery-ui.min.js	MEDIUM	1		3
jquerypp-1.0.1.jar: string.js	HIGH	1		3
jquery-ui-1.10.4.jar: jquery.ui.dialog.js	MEDIUM	1		3
jquery-ui-1.10.4.jar: jquery-ui.js	MEDIUM	1		3
jquery-ui-core-6.2.2.jar: jquery-ui-1.10.1.custom.min.js	MEDIUM	1		3
wicket-jquery-ui-core-6.20.0.jar: moment.min.js	low	1		3
wicket-core-6.19.0.jar: jquery-2.1.3.js	medium	3		3
wicket-core-6.19.0.jar: jquery-1.11.2.js	medium	3		3
wicket-core-6.19.0.jar: jquery-1.11.2.min.js	medium	3		3
wicket-core-6.19.0.jar: jquery-2.1.3.min.js	medium	3		3
wicketstuff-inmethod-grid-6.17.0.jar: event.min.js	MEDIUM	5		3
wicketstuff-inmethod-grid-6.17.0.jar: event.js	MEDIUM	5		3
wicketstuff-inmethod-grid-6.17.0.jar: yahoo.min.js	MEDIUM	5		3
wicketstuff-inmethod-grid-6.17.0.jar: yahoo.js	MEDIUM	5		3
wicketstuff-inmethod-grid-6.17.0.jar: dom.js	MEDIUM	5		3
wicketstuff-inmethod-grid-6.17.0.jar: dom.min.js	MEDIUM	5		3
ENTITY				
bsh-2.0b4.jar	HIGH	1	Highest	23
commons-collections4-4.0.jar	HIGH	1	Highest	37
poi-3.10.1.jar	HIGH	5	Highest	29
dom4j-1.6.1.jar	HIGH	1	Highest	25
resteasy-jaxrs-3.0.11.Final.jar	HIGH	2		21
httpclient-4.2.6.jar	MEDIUM	2	Highest	34
jackson-mapper-asl-1.9.12.jar	CRITICAL	14	High	30

A planilha acima apresenta as vulnerabilidades encontradas nas dependências nos componentes da solução, o detalhamento encontra-se no Anexo I deste documento.

4.3 Análise sobre os resultados

Este tópico tratará tecnicamente a análise baseado nos resultados obtidos pelas ferramentas citadas juntamente com a análise amostral do código fonte.

4.3.1 Manutenibilidade de código

Os relatórios apresentados pela ferramenta SonarQube demonstram uma série de vícios adotados durante o processo de construção do software e alinhado a estes vícios.

A inexistência de cobertura de testes de unidade que trazem a dificuldade no processo de refactoring da aplicação, uma vez que não há condições de mensurar impactos durante o processo de manutenção corretiva/adaptativa. A alta complexidade ciclomática dificulta o processo de refactoring, a ilustração abaixo demonstra o cenário apresentados (OBS: a característica apresentada é utilizada de forma recorrente em diversos momentos do código).

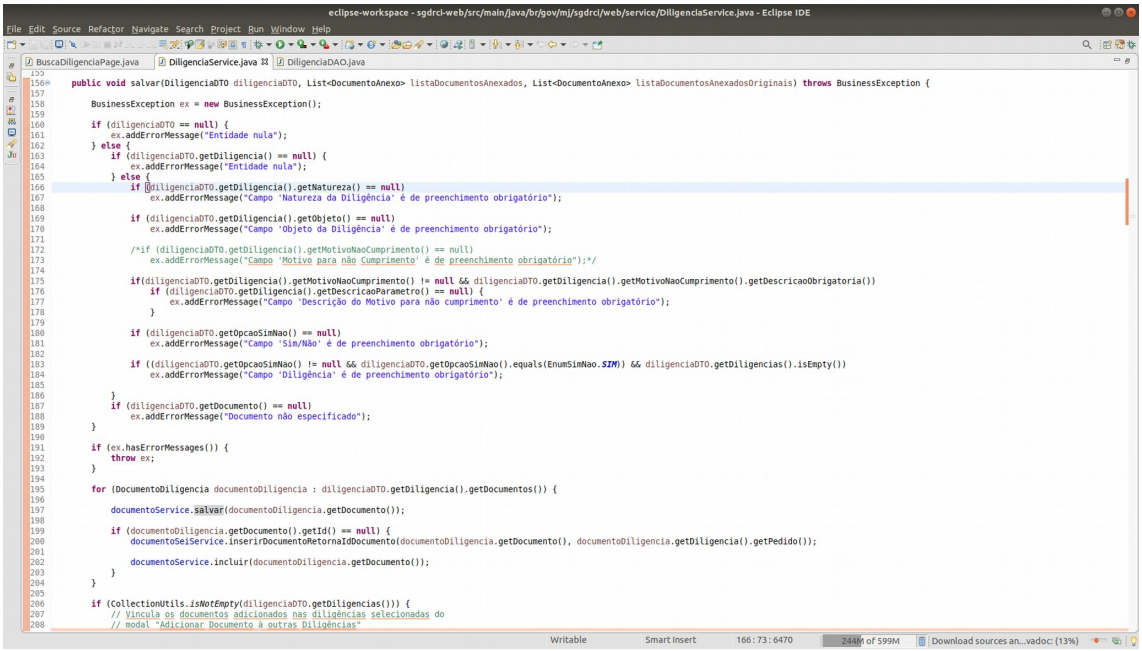

Figura 9: Complexidade ciclomática - DiligenciaService.java

Outro cenário apresentado no código fonte é a falta adequada para tratamento das exceções, o código apresentado pelo SonarQube a seguir é apresentado de forma recorrente na aplicação.

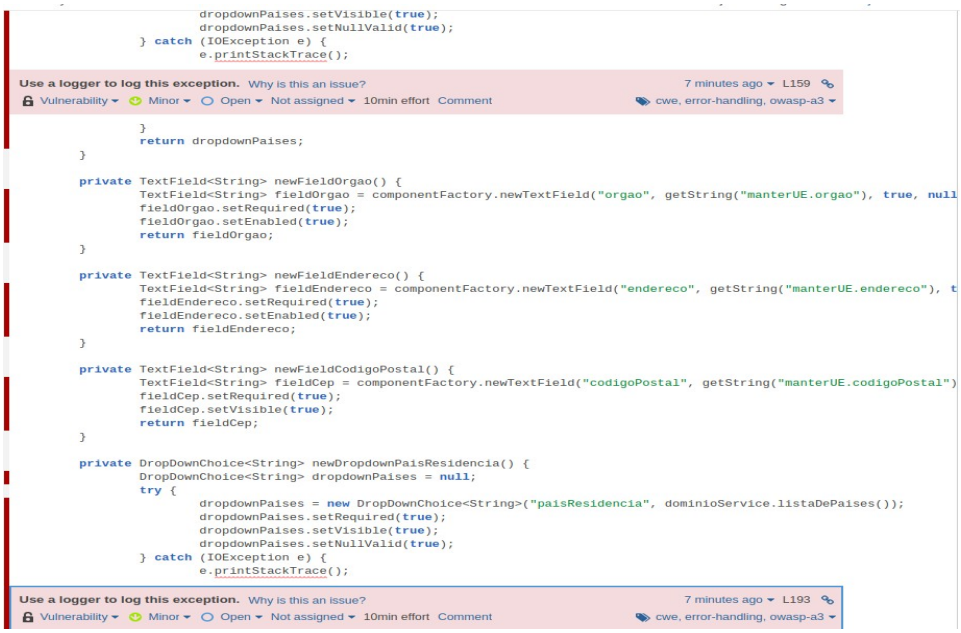

Figura 10: Tratamento de logs

4.3.2 Confiabilidade

Há controle de transação com as operações em banco de dados na camada de acesso a dados (DAO – Data Access Object). Embora esta esteja sendo utilizado, esta prática é melhor utilizada quando aplicada a camada de serviço, uma vez que ali é possível garantir as propriedades ACID da transação como um todo. O exemplo abaixo identifica uma vulnerabilidade no que tange a consistência de persistência quando utilizado uma lista de objetos.

```
public void salvarDocumentoDaLista(List<DocumentoDiligencia> listaDocumento) {

    for (DocumentoDiligencia documentoDiligencia : listaDocumento) {
        if (documentoDiligencia.getDocumento().getId() == null) {
            documentoSeiService.inserirDocumentoRetornaIdDocumento(documentoDiligencia.getDocumento(), documentoDiligencia.getDiligencia().getPedido());
            salvar(documentoDiligencia.getDocumento());
        }
    }

}
```

Figura 11: Vulnerabilidade na consistência de dados

Neste exemplo, quando houver uma exceção na tratativa de persistencia da lista, somente uma parte da informação poderá estar persistida em banco de dados.

4.3.3 Performance e estabilidade

Não foi analisado a aplicação em funcionamento para avaliar demais requisitos não funcionais. Durante o processo de análise de código fonte, não fora encontrado evidências que demonstrem impactos em performance da aplicação.

4.3.3 Escalabilidade

A arquitetura monolítica citada no tópico 3 deste documento prejudica a escalabilidade da ferramenta, os recursos empreendidos para a escalabilidade vertical (aumento de recursos de processamento, disco, memória e demais) são limitados.

A escalabilidade vertical do monólito é possível levanto em consideração o aumento de nós no cluster, contudo esta escalabilidade é prejudicada tendo em vista que temos que escalar a aplicação como um todo, necessitando assim da mesma quantidade de recursos empreendidas nos demais nós existentes. Nesta arquitetura não há a possibilidade de escalar somente as funcionalidades/módulos que mais são demandados.

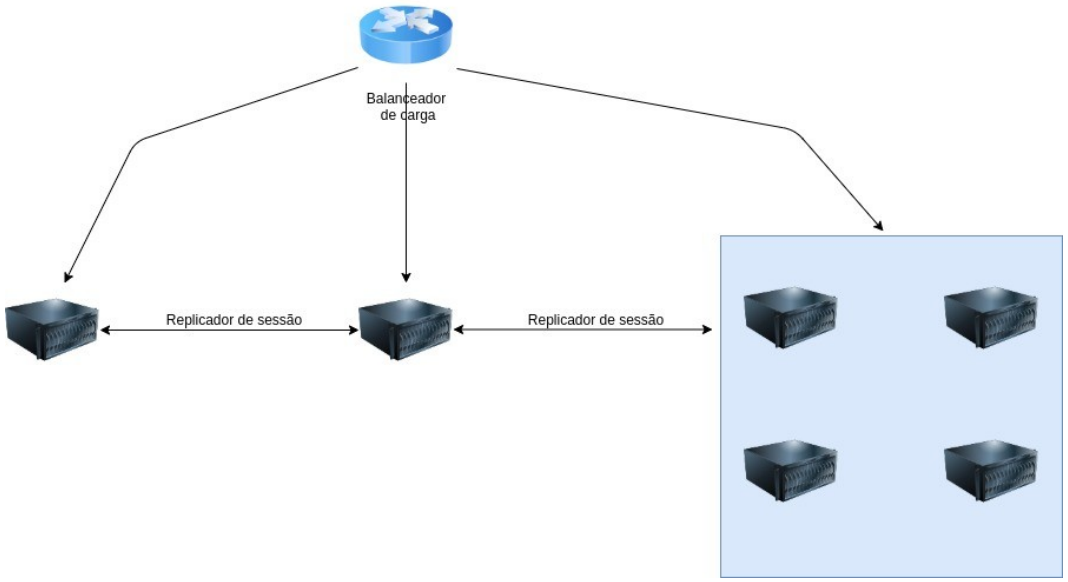

Figura 12: Escalabilidade do monólito

5 Recomendações

É altamente recomendado que seja efetuado refactoring de código dos bugs e vulnerabilidades 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.

Para melhor descoberta de erros e defeitos durante a execução da aplicação, sugere-se ajustes na tratativa dos fluxos de exceções apontados na análise de manutenibilidade de código.

Para a manutenção da consistência dos dados, sugere-se também que sejam tratados os pontos levantados na análise de confiabilidade.

Recomenda-se utilizar a especificação JSR 380 (Bean Validation) para as validações de obrigatoriedade dos formulários, certamente está prática ajudará a diminuir a complexidade ciclomática do código.

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 utilização de ferramentas de pentest (análise de vulnerabilidade) tais como OWASP ZAP (https://owasp.org/www-project-zap/) para que seja analisado 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 também que seja instalado o agente da

ferramenta de APM do Ministério da Justiça nos ambientes de homologação e produção, criar métricas e alarmes auxiliam na continuidade do serviço (monitoramento de processamento e memória por exemplo) tendo em vista que esta ferramenta fornece mecanismos para determinarmos o comportamento da solução (auxiliam no refactoring de código) também subsidia para o correto dimensionamento da infraestrutura.

6 Conclusão

A aplicação apresenta boa estruturação em sua construção, fato este que não propícia facilidade na manutenção corretiva/evolutiva. Não há pontos no código fonte que demonstrem implicações quanto a performance da aplicação, contudo uma análise de comportamento e funcionamento da aplicação com ferramenta de APM seria necessário para tal conclusão (conforme sugerido nas recomendações).

Em relação aos ajustes de código da aplicação, sugestiona-se que os mesmos sejam efetuados com o auxilio/suporte de testes unitários tendo em vista que este caminho trará maior assertividade e menor risco para o não comprometimento da estabilidade da ferramenta.