Ministério da Justiça

Projeto: OUVIDORIA

Nota Técnica

Revisão	Descrição	Autor	Data
1.0	Construção do documento	Israel Branco	01/05/2020

1 Sumário

2 Introdução

Este documento visa reportar o resultado da análise efetuada na aplicação Ouvidoria. 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, utilizou-se a *branch master* no repositório http://git.mj.gov.br/STEFANINI/OUVIDORIA-WEB.git na data de 01/05/2020.

3 Apresentação do cenário atual

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

O sistema Ouvidoria possui unicamente um módulo em sua composição:

- ***WEB***: 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 banco de dados relacional *MySQL .*

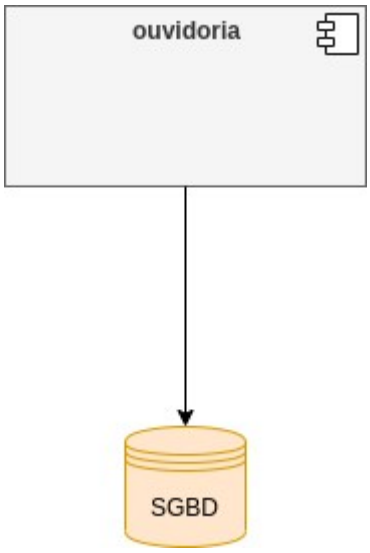

Figura 1: Representação de componentes

3.1 Componente WEB

Este componente representa o domínio da aplicação, camada de acesso a dados, controladores de formulários e apresentação WEB. A estruturação deste projeto é intuitiva e bem estruturada, também há boa adequação dos padrões de projeto em conformidade com a nomenclatura dos pacotes. A estruturação da camada de apresentação WEB também está organizada de forma lógica e intuitiva.

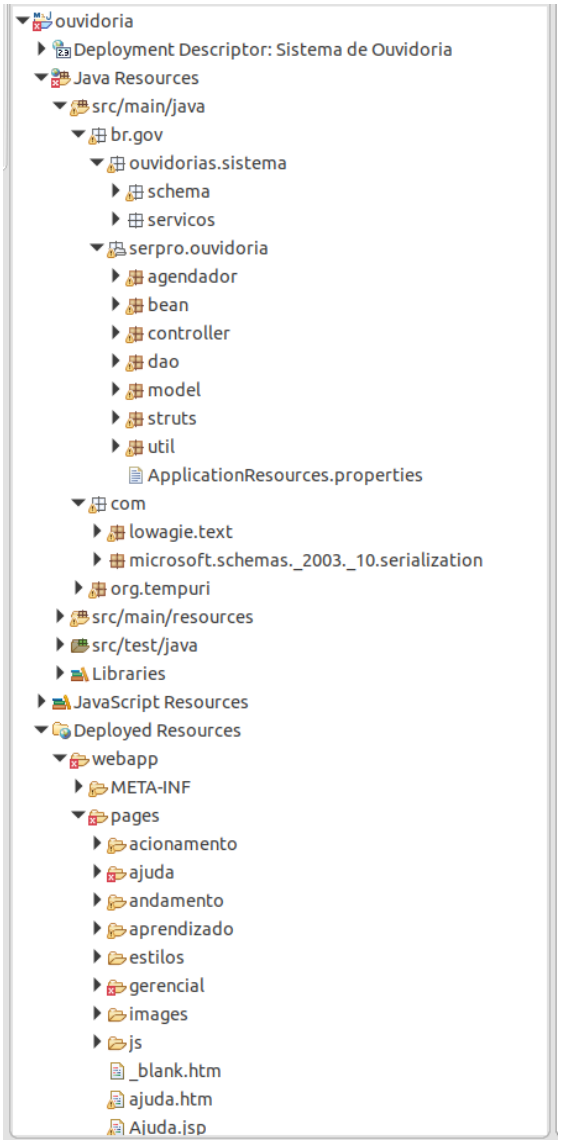

Figura 2: Estrutura do projeto

3.2 Tecnologias utilizadas

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

Nome	Versão	Utilização	Observação
Java	1.6	Linguagem de programação.	
Apache Struts	1.2.4	Framework MVC	
jFreeChart	0.9.21	Framework para criação de gráficos.	
C3P0	0.8.4.5	Gerenciador de pool de conexão JDBC.	
Hibernate	3.0.5	Framework ORM	
Axis	1.4	Framework para construção/consumo de serviços SOAP.	
Jboss AS	4.2	Servidor de aplicação.	
MySQL	x	Servidor de banco de dados.	

3.3 Modelagem de dados

A estrutura de banco de dados esta composta por 55 tabelas em um único schema. Há tabelas que não apresentam relacionamentos nesta estrutura.

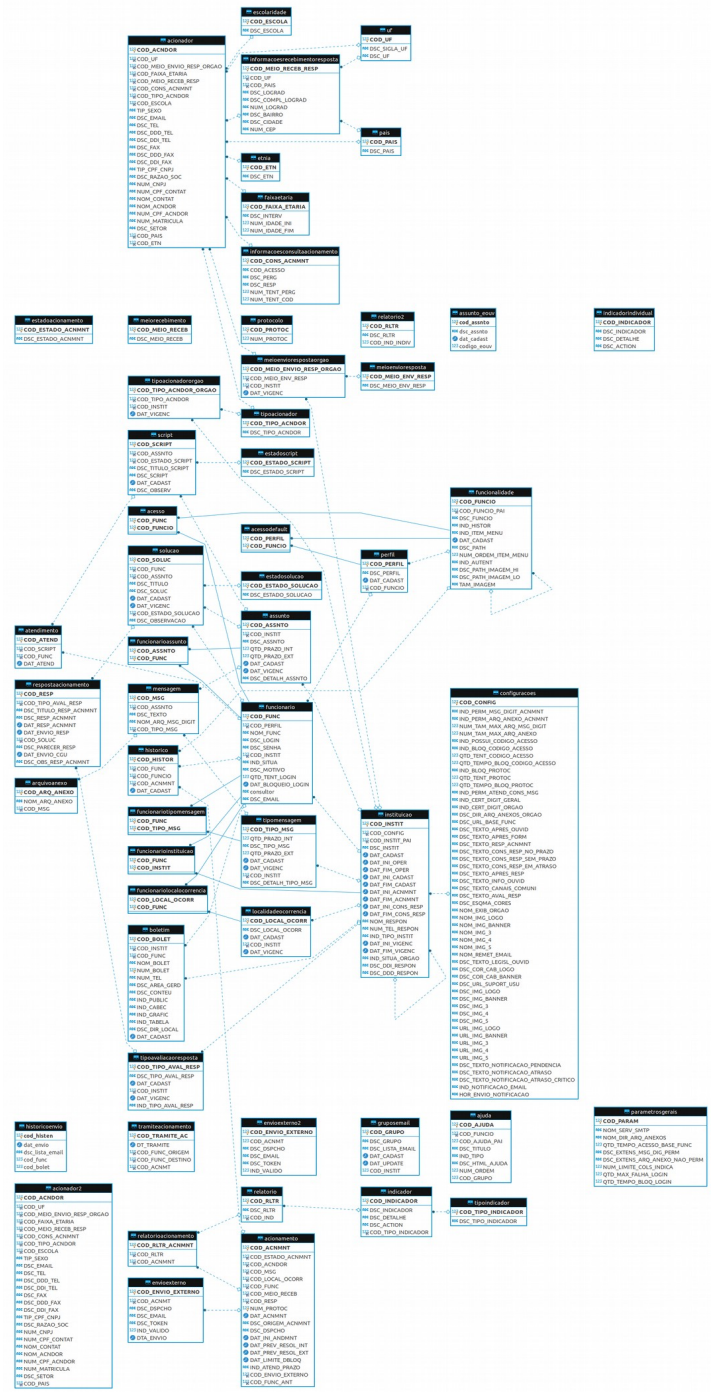

Figura 3: MER - Ouvidoria

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 aplicações):

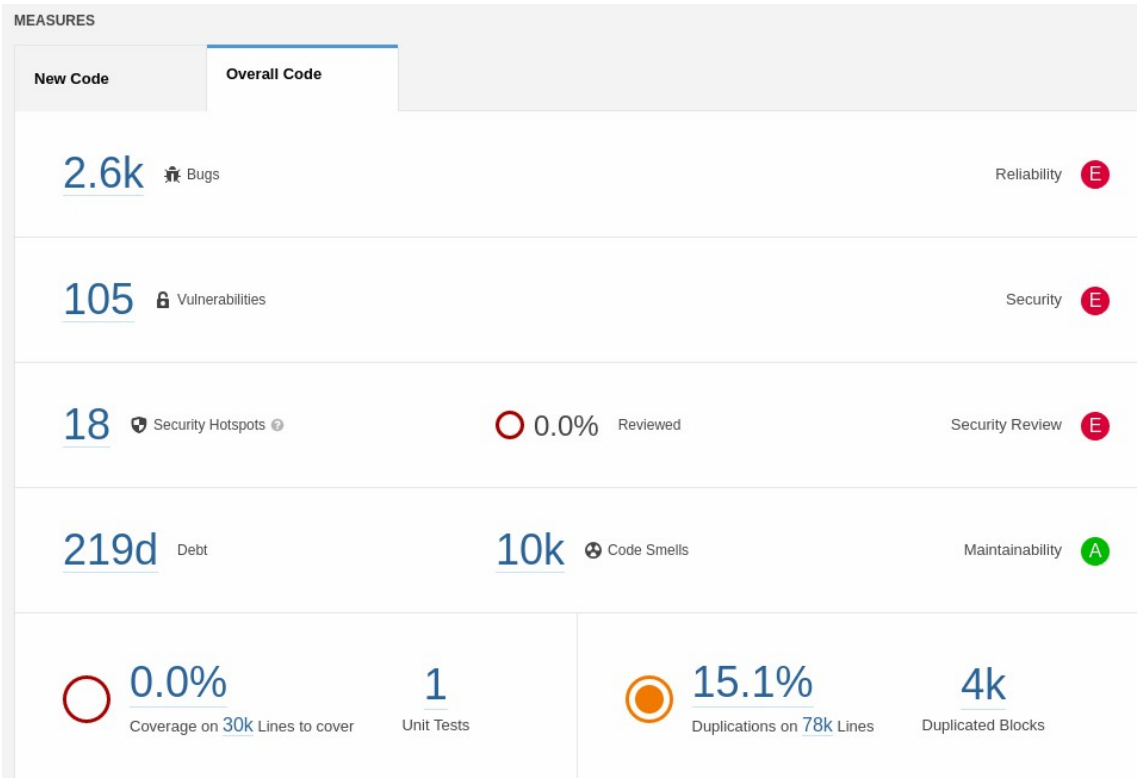

Figura 4: Análise estática de código

- 2.6 mil bugs;
- 18 violações de segurança;
- 10 mil violações de más práticas (complexidade cognitiva, complexidade ciclomática, débito técnico e outros)
- 105 vulnerabilidades;
- 15.1% de duplicação de código;

Somente um teste de unidade foi encontrado durante o processo de análise.

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
axis-1.4.jar	HIGH	4	Highest	22
axis-saaj-1.4.jar	HIGH	4	Highest	10
jstl-1.0.jar	HIGH	1		20
log4j-1.2.12.jar	CRITICAL	2	Highest	14
struts-1.2.4.jar	HIGH	24	Highest	30
commons-beanutils-1.6.1.jar	HIGH	2	Highest	24
commons-fileupload-1.0.jar	CRITICAL	5	Highest	24
struts-legacy-1.1.jar	HIGH	11	Highest	18
dom4j-1.6.jar	HIGH	1	Highest	25
mysql-connector-java-5.1.37.jar	HIGH	7	Highest	38
commons-collections-3.2.1.jar	CRITICAL	3	Highest	35
commons-beanutils-core-1.7.0.jar	HIGH	2	Highest	20
commons-beanutils-bean-collections-1.7.0.jar	CRITICAL	4	Highest	20
xercesImpl-2.2.1.jar	Unknown	2		13
bcprov-jdk14-1.46.jar	Unknown	15	Highest	29
quartz-1.6.0.jar	CRITICAL	1		15
batik-dom-1.6.jar	CRITICAL	3	Low	19
batik-awt-util-1.6.jar	CRITICAL	3	Low	20
batik-svggen-1.6.jar	CRITICAL	3	Low	19
batik-xml-1.6.jar	CRITICAL	3	Low	19
batik-util-1.6.jar	CRITICAL	3	Low	19
c3p0-0.8.4.5.jar	HIGH	1	Low	15
connector-api-1.5.jar	MEDIUM	2	Low	17
commons-compress-1.14.jar	MEDIUM	2	Highest	43

A planilha acima apresenta as vulnerabilidades encontradas nas dependências de cada módulo, o detalhamento encontra-se no Anexo I deste documento.

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.

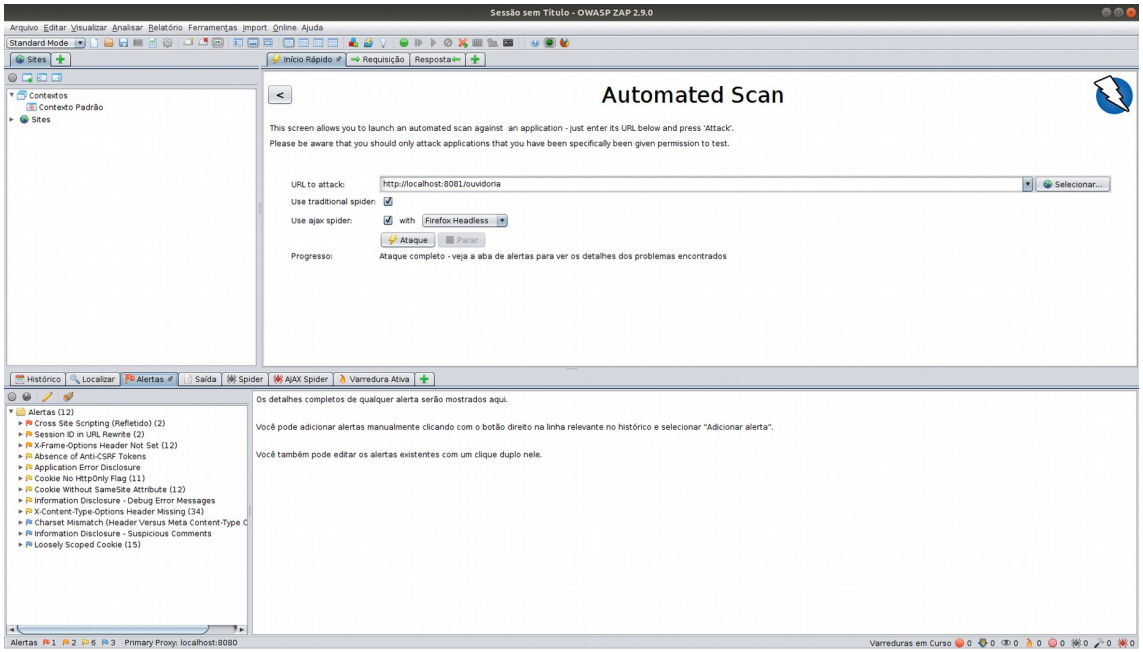

Figura 5: OWASP Zap - análise de intrusão

O relatório completo deste teste está disponível no anexo I deste documento, o detalhamento desta análise está classificada em:

- 1 vulnerabilidades de severidade alta;
- 2 vulnerabilidades de severidade média;
- 6 vulnerabilidades de baixa média;
- 3 vulnerabilidades a nível informativo;

4.4 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.

4.4.1 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 e a falta de coesão dificultam este processo de refactoring, as ilustrações que seguem demonstram os cenários apontados (OBS: a característica apresentada é utilizada de forma recorrente em diversos momentos do código).

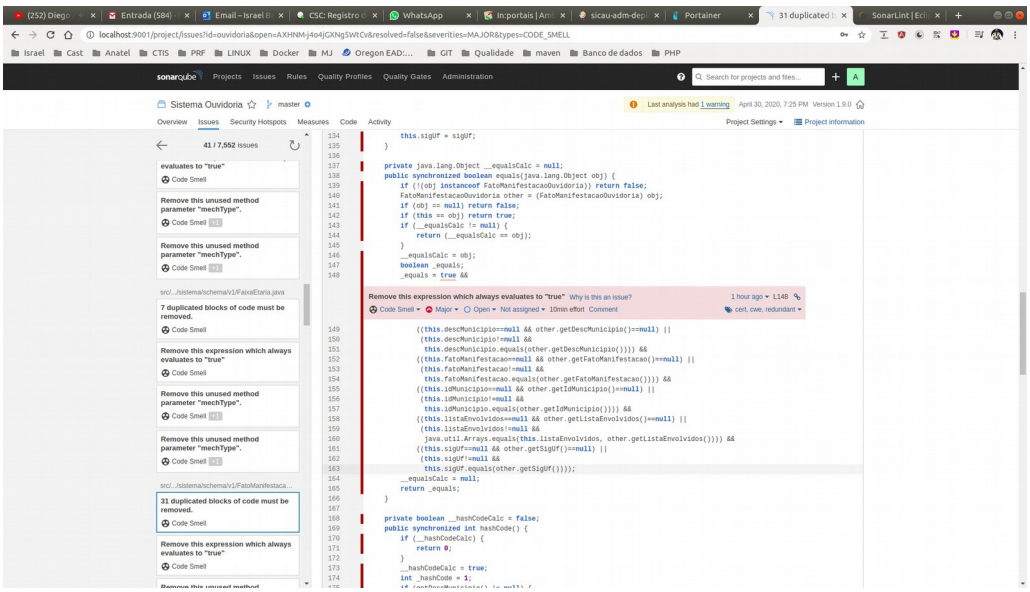

Figura 6: Complexidade ciclomática

A existência de código javascript distribuído entre as páginas JSP e a utilização de scriptlets Java nas mesmas são fatores que contribuem para a difícil manutenibilidade do código.

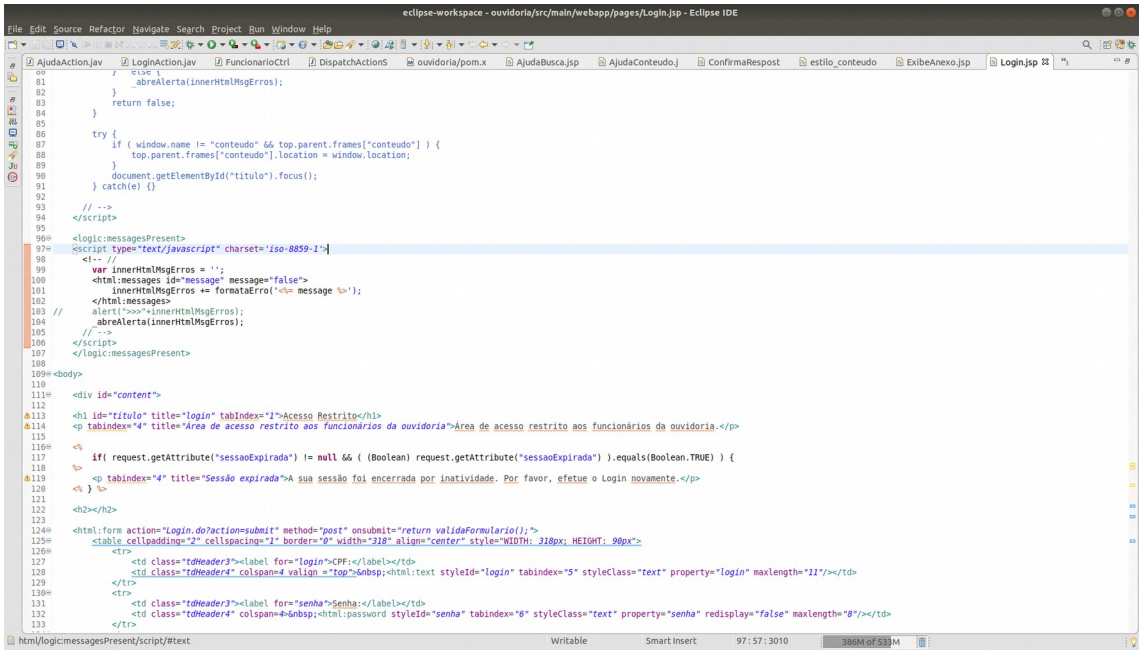

Figura 7: Arquivos JSP contendo Scriptlets Java e código Javascript

4.4.2 Confiabilidade

Todas as operações em banco de dados realizado na aplicação apresentam controle transacional, contudo este controle é feito a nível de JDBC e não a nível de aplicação. Este fator pode gerar inconsistências quando há a manutenção de dados em mais de uma tabela por operação.

Outro fator encontrado durante a análise do código foi a presença de URL's estáticas dentro do código. Este fator torna a aplicação vulnerável e suscetível a falhas quando houver mudanças das dependências externas.

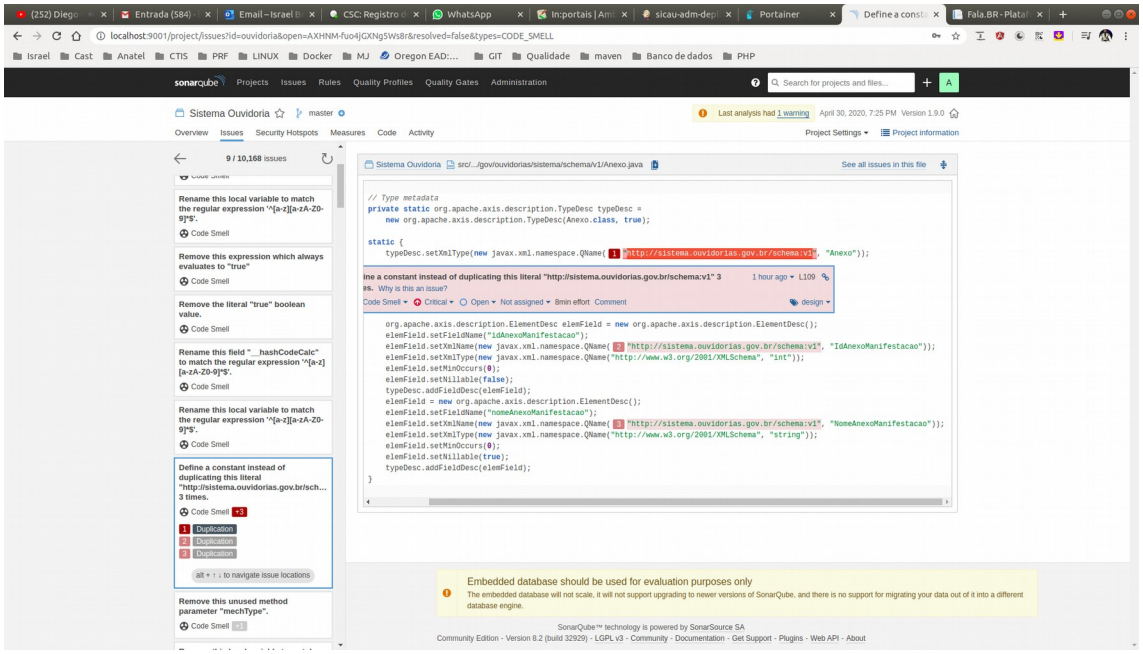

Figura 8: URL's estáticas

4.4.3 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.

4.4.4 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.

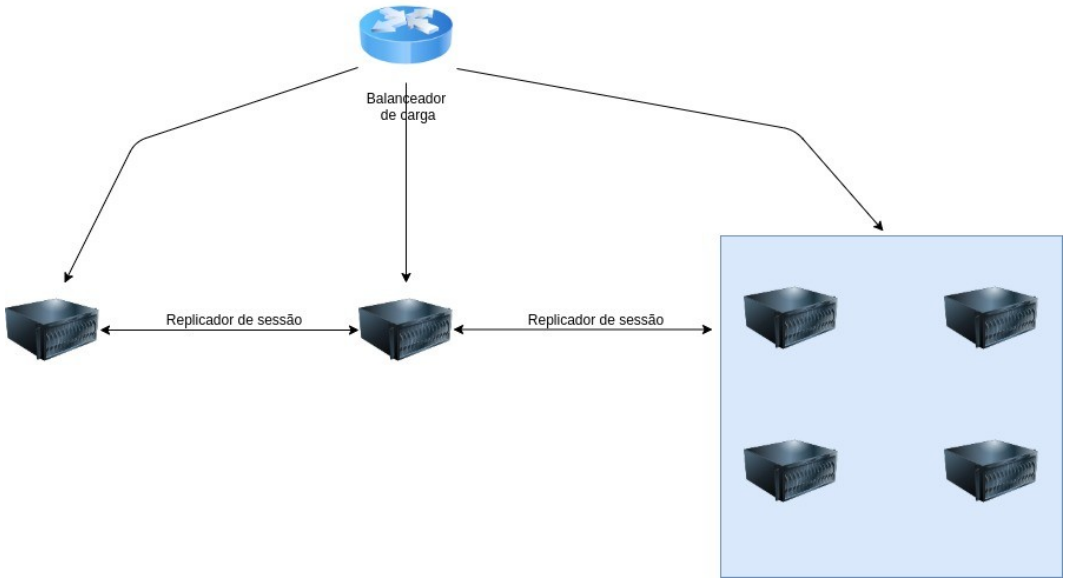

Figura 9: Escalabilidade do monólito

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.

Solucionar problemas de vulnerabilidade apresentado pelo relatório das ferramentas OWASP ZAP e Dependency Check.

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.

A aplicação apresenta funcionalidades de execução de tarefas agendadas com a utilização do framework Quartz, recomenda-se que estas execuções sejam isoladas do core do monólito tendo em vista que esta pratica além de utilizar de forma concorrente os recursos de máquina da aplicação, dificulta a escalabilidade horizontal citada no tópico 4.4.4.

6 Conclusão

A aplicação apresenta uma boa estruturação em sua construção e mesmo que utilize tecnologias obsoletas em sua construção, sua estrutura propicia bom entendimento para a realização de manutenções corretivas/evolutivas. A inatividade da comunidade/fabricante para os frameworks, bibliotecas e demais componentes de terceiros utilizados neste projeto dificultam a resolução de determinados problemas.

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.

Caso haja a necessidade de adequações nas páginas WEB para atender os critérios de usabilidade, recomenda-se que a ferramenta seja reconstruída utilizando a arquitetura de referência projetada para o Ministério da Justiça.