Ministério da Justiça

Projeto: CLASSIFICAÇÃO INDICATIVA

Nota Técnica

Revisão	Descrição	Autor	Data
1.0	Construção do documento	Israel Branco	12/03/2020

1 Sumário

2 Introdução

O sistema CLASSIFICAÇÃO INDICATIVA chamado neste documento daqui para frente como CLASSIND foi criado com o intuito de atender a necessidade de classificação indicativa de jogos eletrônicos, filmes, programas de tv e correlatos.

O sistema conta com 2 módulos para o seu funcionamento, módulo WEB que trata as consultas públicas através do portal do Ministério da Justiça e o módulo administrativo aplicação desktop que trata a classificação dos registros (Visual Básic 6).

O módulo WEB alvo desta análise, foi desenvolvido utilizando tecnologias que datam entre os períodos de 2004 a 2006, se tratando de um legado superior a 10 anos estas encontram-se obsoletas sem suporte e sem atualização de seus fabricantes.

3 Apresentação do cenário atual

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

O sistema CLASSIND 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 *Microsoft SQLServer .*

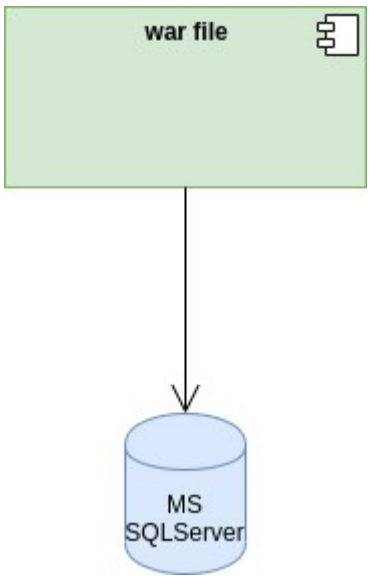

Figura 1: Diagrama de componentes

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 não é intuitiva, também não há boa adequação dos padrões de projeto em conformidade com a nomenclatura dos pacotes, por fim boa parte das classes deste componente extrapolam o limite de sua competência tornando as mesmas pouco coesa e muito extensas. A estruturação da camada de apresentação WEB é mais lógica e intuitiva.

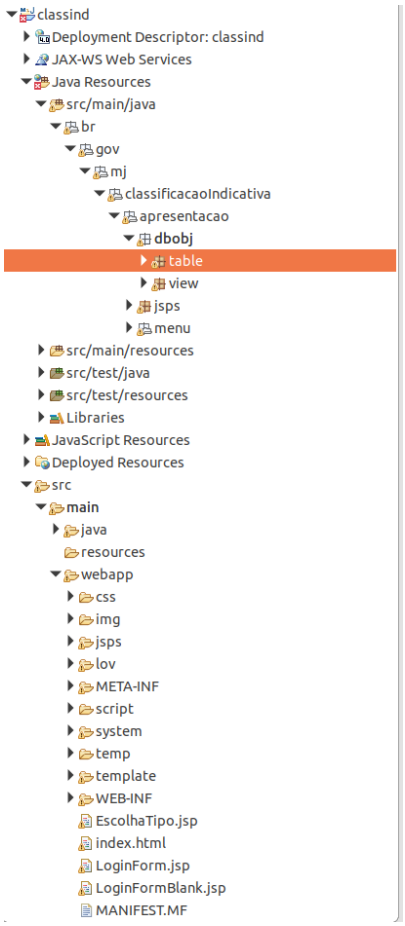

Figura 2: Estruturação do projeto

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	1.5	Linguagem de programação.	
Apache Struts	1.x	Framework MVC	
jFreeChart	0.9.8	Framework para criação de gráficos	
Egen-JDK	x	Não foi possível evidenciar o objetivo desta dependência.	
Egen-Util	x	Não foi possível evidenciar o objetivo desta dependência.	
Cewolf		Não foi possível evidenciar o objetivo desta dependência.	
jTDS	1.2	Driver jdbc para Microsoft SQL Server	

3.3 Modelagem de dados

A estrutura de banco de dados esta composta por 45 tabelas em um único schema. Há 4 tabelas que não apresentam relacionamentos nesta estrutura, sendo elas: tbClassificacaoJogo, tbClassificacao, tbClassificacaoIndicativaOrdenada e tbClassificacaoAssunto.

Uma observação importante é que este modelo de dados não possui padronização na utilização de nomenclatura das tabelas e estão agrupadas em um banco de dados que possui demais objetivos que extrapolam o domínio do CLASSIND.

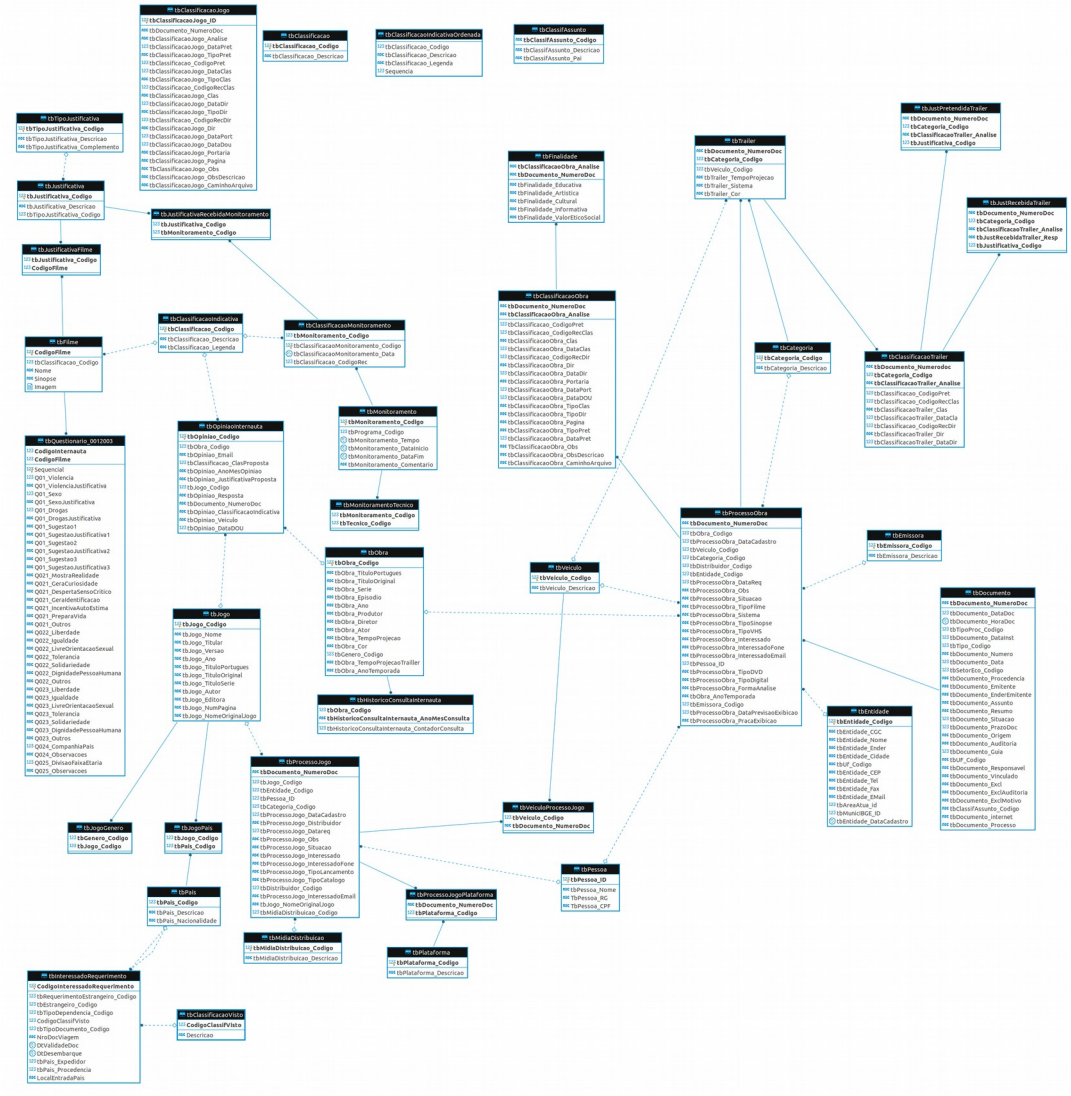

Figura 3: MER Banco CLASSIND

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 (branch Stable):

4.1.1 WEB

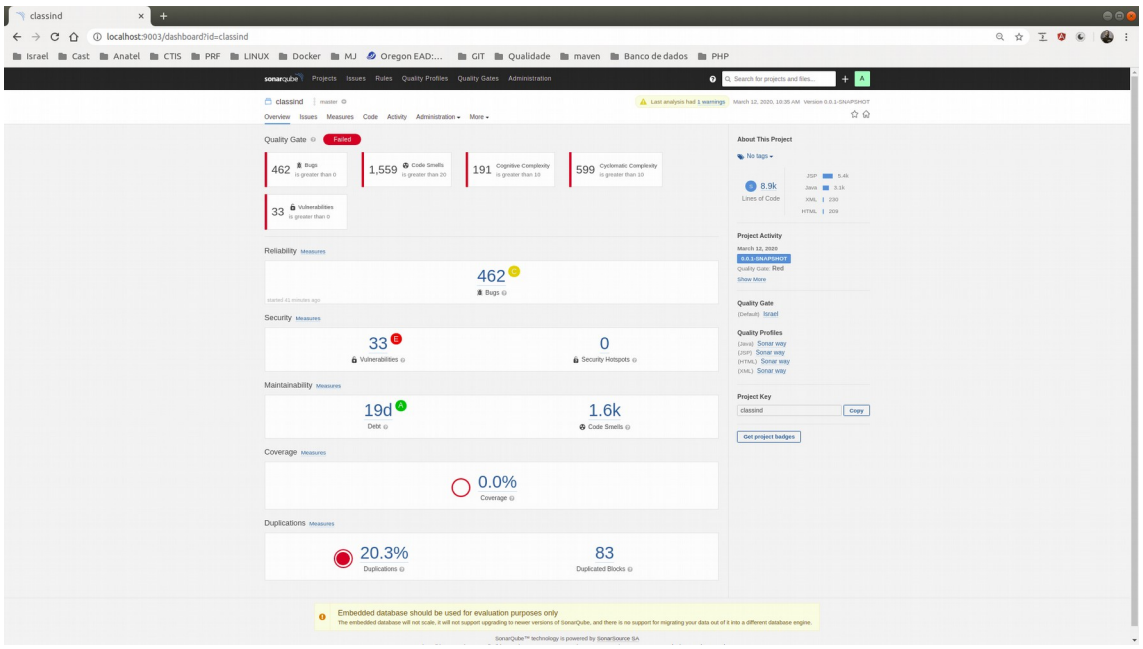

Figura 4: Análise estática de código

- 465 bugs;
- 1559 violações de más práticas;
- 191 violações de complexidade cognitiva (dificuldade de entendimento de código);
- 559 violações de complexidade ciclomática (complexidade de código);
- 33 vulnerabilidades;
- 14.9% de duplicação 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 no projeto back-end, a seguir temos as principais informações extraídas desta análise.

Dependency	Highest Severity	CVE Count	Confidence	Evidence Count
struts-1.2.6.jar	HIGH	10	Highest	20
commons-beanutils-1.2.jar	HIGH	2	Highest	19
jsch-0.1.27.jar	MEDIUM	1	Highest	24
plexus-archiver-2.6.3.jar	MEDIUM	1	Highest	27
commons-compress-1.8.1.jar	MEDIUM	1	Highest	40
plexus-utils-3.0.18.jar	0.0	2	Highest	29

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

Vale ressaltar que as dependências listadas com a exceção da lib do framework struts foram incluídas neste projeto para que consigamos executá-lo. Acredita-se devam estar embutidas no servidor de aplicação utilizado para a execução em ambiente produtivo.

A lib do framework struts pode não corresponder a versão adequada uma vez que esta veio na pasta lib do projeto quando baixado diretamente do repositório git. Esta e outras dependências que não obtivemos acesso a versão, foram incorporadas no projeto por analogia ao tempo da tecnologia e a versão da JDK utilizada.

Figura 5: Diretório lib do projeto CLASSIND

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.

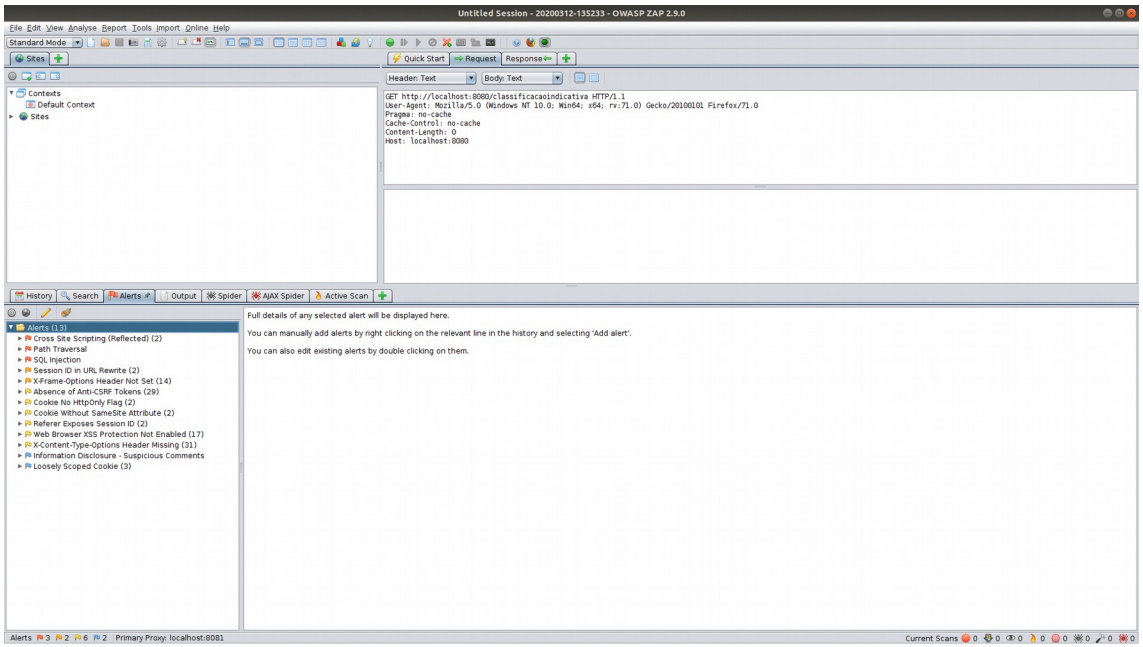

Figura 6: OWASP ZAP - Teste de penetração

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

- 3 vulnerabilidades de severidade alta;
- 2 vulnerabilidades de severidade média;
- 6 vulnerabilidades de baixa média;
- 2 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).

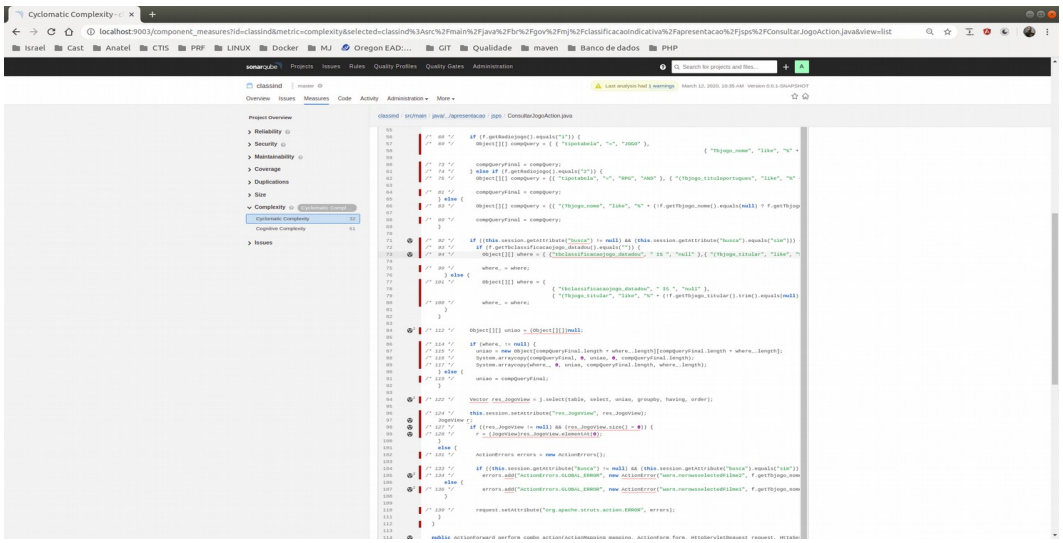

Figura 7: Alta complexidade ciclomática e falta de coesão – ConsultarJogoAction.java

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.

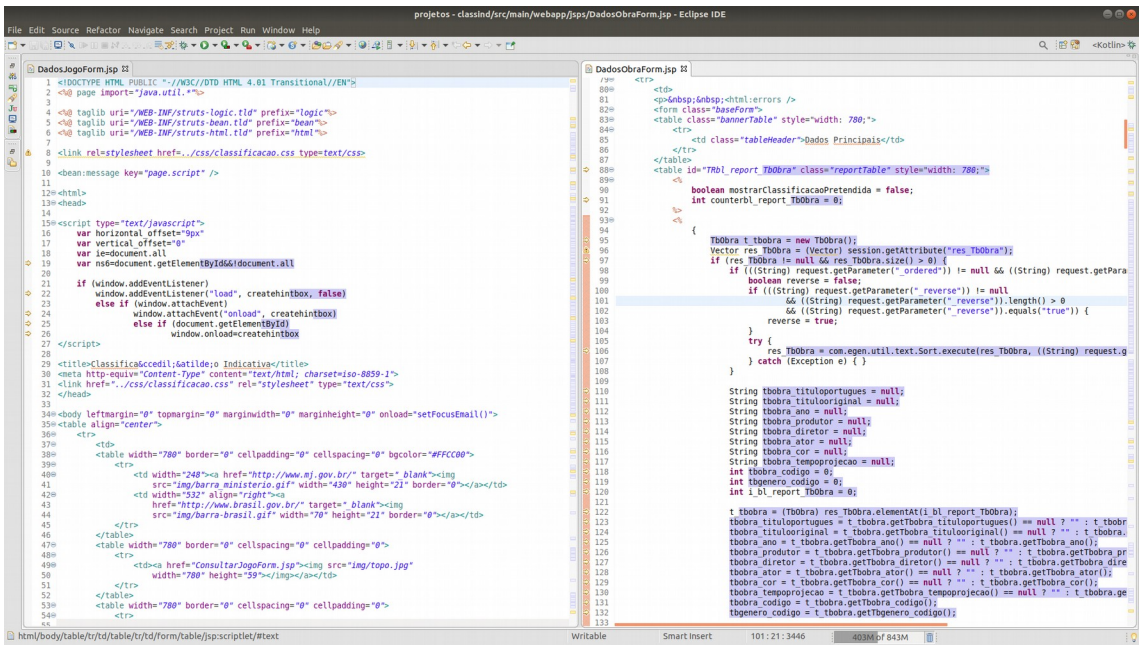

Figura 8: Arquivos JSP contendo Scriptlets Java e código Javascript

O javascript utilizado na figura 7 no painel a esquerda se repete em demais páginas.

4.4.2 Confiabilidade

Todas as operações em banco de dados realizado na aplicação são meramente consultas e não há tratativas de controle transacional nas operações, contudo este não chega a ser um problema dado a característica de iteração com o banco de dados.

4.4.3 Performance e estabilidade

Aplicação não apresenta recurso de paginação de consultas em banco de dados, a paginação está sendo realizada no formulário JSP

fator este que degrada a performance da mesma tanto na utilização desnecessária de recursos de banco de dados, armazenamento em memória no servidor de aplicação Java e utilização de recursos de rede.

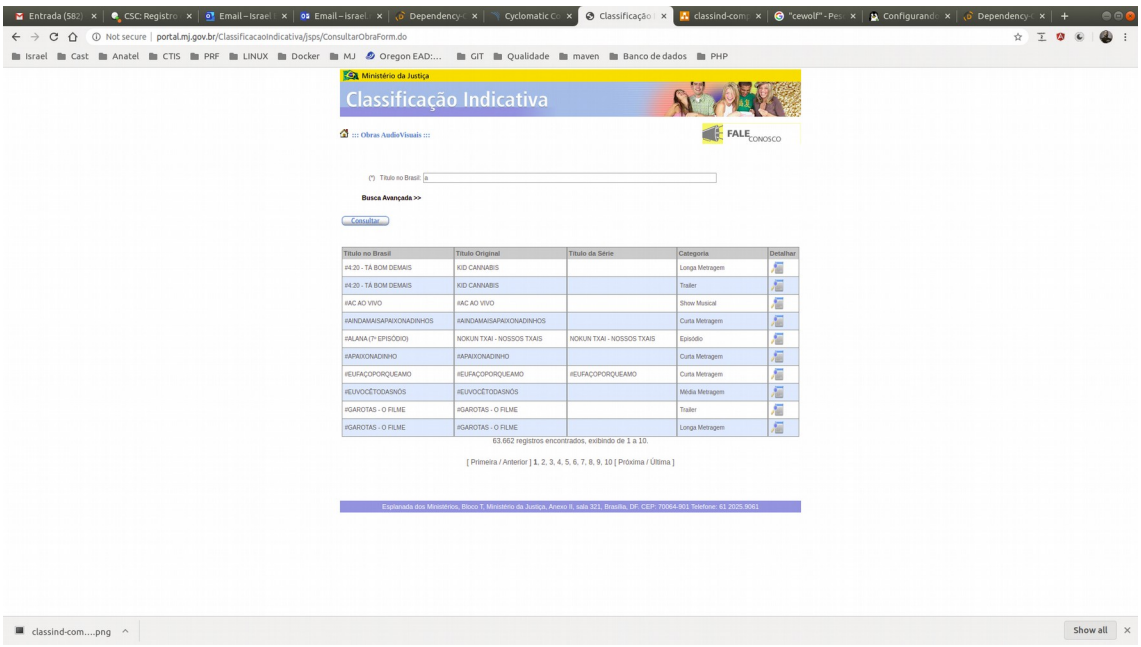

Figura 9: Paginação sendo realizada no formulário JSP

Não há critério para quantidade mínimas de caracteres utilizadas para a realização das consultas.

4.4.3 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.

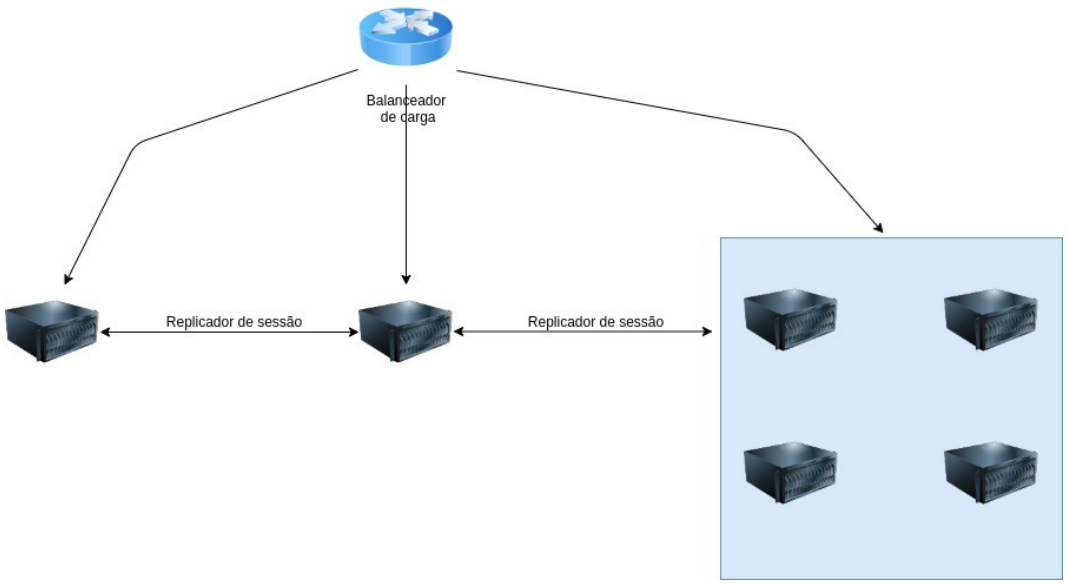

Figura 10: Escalabilidade do monólito

4.4.3 UX – User experience

A aplicação não apresentou incompatibilidade com browsers modernos, contudo a mesma não trabalha de forma responsiva.

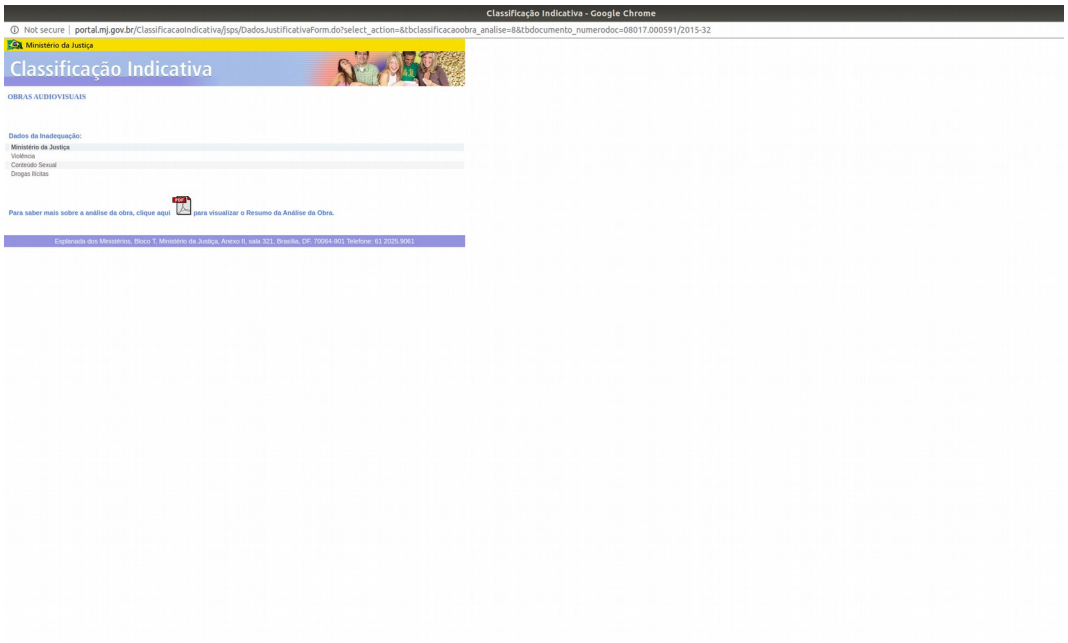

Figura 11: Responsividade

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.

Não há gerenciadores de build no core do projeto, recomendasse que seja utilizado ou o apache ANT uma vez que as dependências do projeto encontram-se na pasta lib, ou o apache Maven. Neste segundo caso, será necessário adicionar as dependências a um repositório de componentes (Artifactory, Nexus ou similar).

Atualmente não há ambiente local para desenvolvimento e sustentação do sistema, recomenda-se que o mesmo seja criado juntamente com manuais para sua reprodução. A falta deste ambiente prejudica e dificulta as manutenções corretivas/evolutivas da ferramenta.

6 Conclusão

A aplicação não apresenta uma boa estruturação em sua construção o que dificulta sua manutenção corretiva/evolutiva. A inatividade da comunidade/fabricante para os frameworks, bibliotecas e demais componentes de terceiros utilizados neste projeto dificultam a resolução de determinados problemas.

Caso este projeto ainda demande muita utilização, recomenda-se a reconstrução do mesmo com a adoção de tecnologias mais modernas.

Salienta-se a necessidade da utilização de ambiente local para as manutenções corretivas/evolutivas, essa tratativa trará menor risco de implementação e trará celeridade ao processo de homologação.