Ministério da Justiça

Projeto: DESARMA

Nota Técnica

Revisão	Descrição	Autor	Data
1.0	Construção do documento	Israel Branco	04/03/2020

1 Sumário

2 Introdução

O sistema DASARMA foi criado com o intuito de atender a campanha de desarmamento da população brasileira, sendo seu principal objetivo a manutenção dos dados essenciais para o recebimento do PCE (Produto Controlado pelo Exército) e o ressarcimento de valores a população sobre a entrega do mesmo.

O sistema conta com a manutenção de 2 perfis de acesso sendo eles o administrador geral responsável por manter os dados base para a utilização do sistema e o segundo o responsável pelo recebimento de armas . As tecnologias utilizadas neste projeto datam entre os períodos de 2004 a 2006 e 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 DESARMA possui três módulos 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 .*
- ***SCHEDULER****:* componente implantado no servidor de aplicação para trabalhar rotinas de processamento em lote, utiliza banco de dados relacional *Microsoft SQLServer* e se integra a ferramenta com serviço de gestão de pagamentos do Banco do Brasil.
- ***WebService:*** componente criado para expor serviços utilizando protocolo SOAP, utiliza banco de dados relacional *Microsoft SQLServer.*

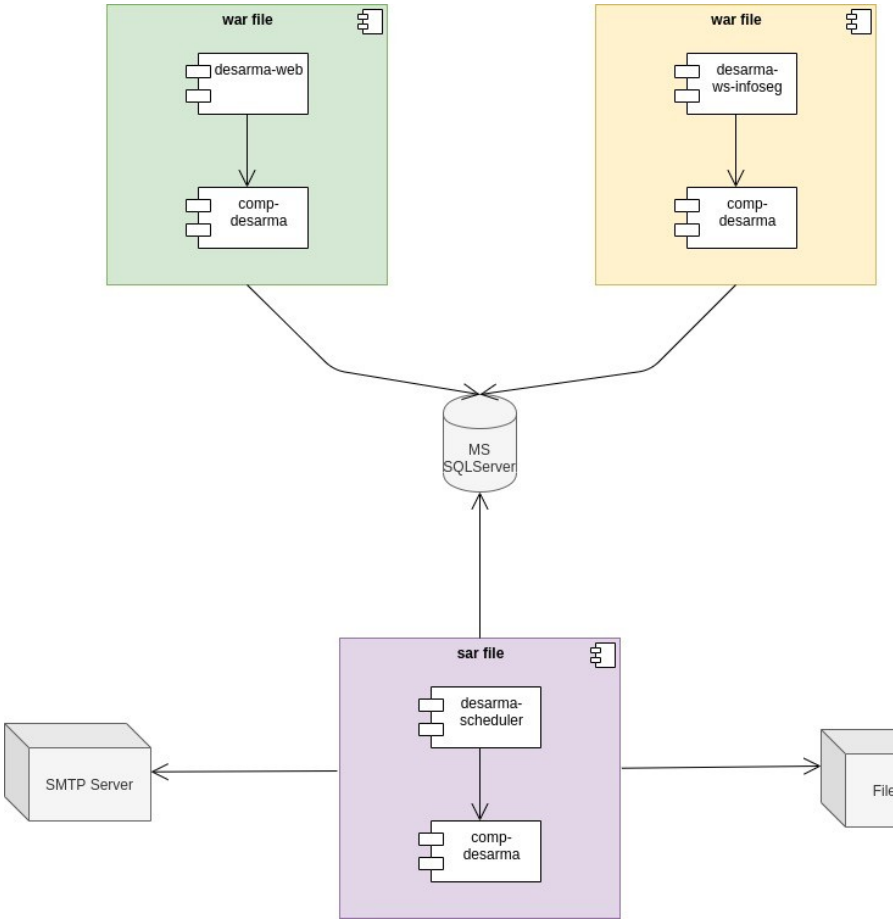

Figura 1: Diagarma de componentes

3.1 Componente Comp-Desarma

Este componente representa o domínio da aplicação, mapeamentos ORM, camada de acesso a dados, camada de serviço expostos em componentes EJB. Não há evidências que demonstrem que este componente opere regras negociais da aplicação, sua

estruturação não é intuitiva, há duplicação de nomenclatura de pacotes, 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.

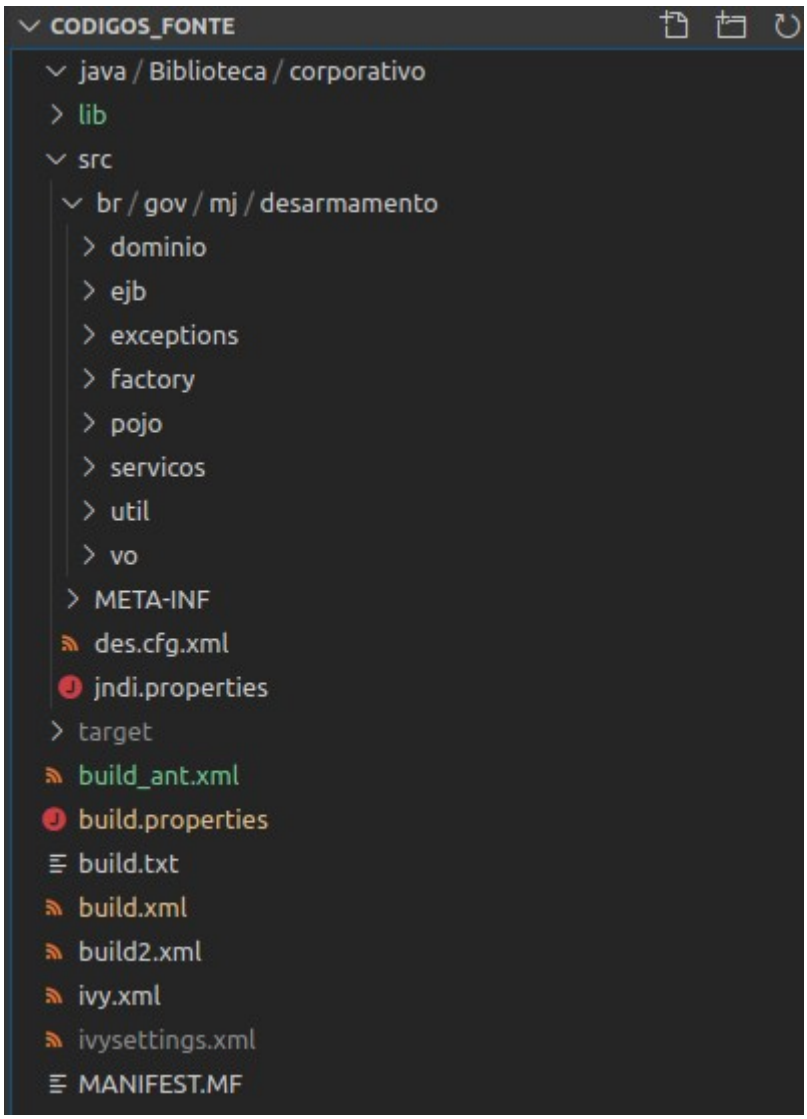

Figura 2: Estrutura do projeto em primeiro escalão

3.2 Componente Desarma-WEB

Este componente representa a camada WEB do ecossistema Desarma, possui em sua composição formulários de iteração com os usuários, relatórios, classes utilitárias para tratamento de mecanismo de segurança captch e filtros para validação de autorização de acesso à aplicação.

Há uma boa segregação de responsabilidade nos pacotes, contudo há também uma duplicação de responsabilidade entre os pacotes **dominio** e **vo** com o componente Comp-Desarma.

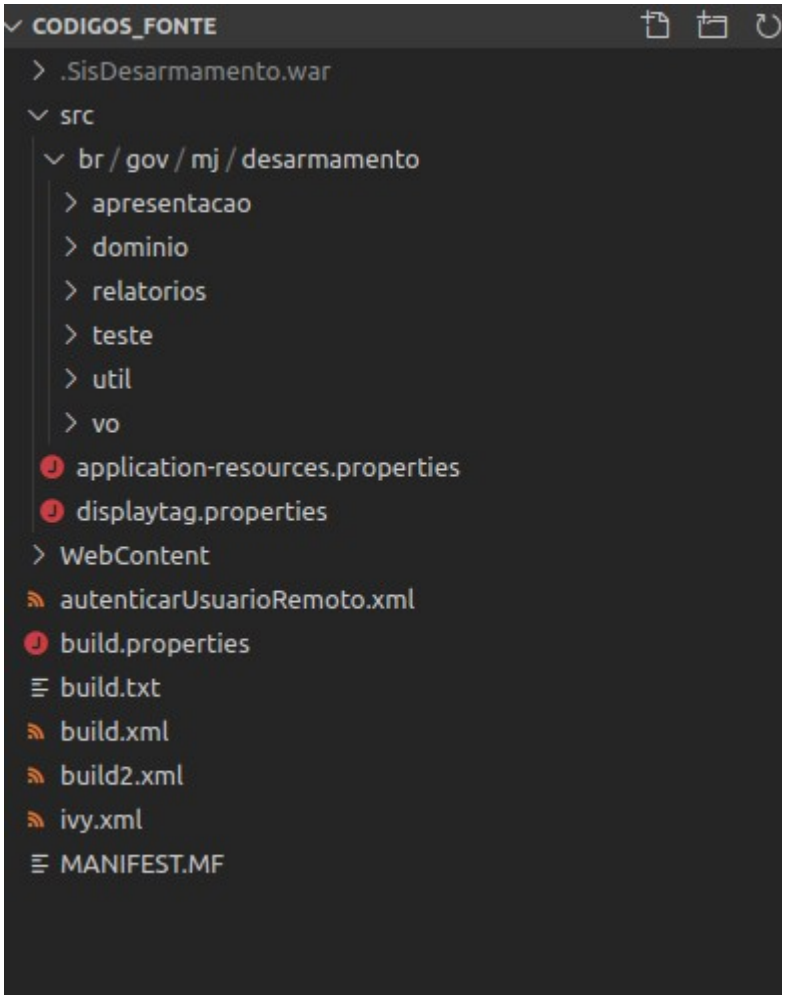

Figura 3: Estrutura do projeto em primeiro escalão

3.3 Componente Desarma-WS-Infoseg

Componente responsável por expor funcionalidade de consulta de armas entregues por período de tempo. Componente foge a estrutura padrão do projeto, utiliza o cor do SpringFramework para utilização do container de IOC do e o modulo Web para criação de WebService SOAP.

A estrutura hierárquica e a nomenclatura utilizada são pouco intuitivas quanto a segregação por responsabilidade e comportamento.

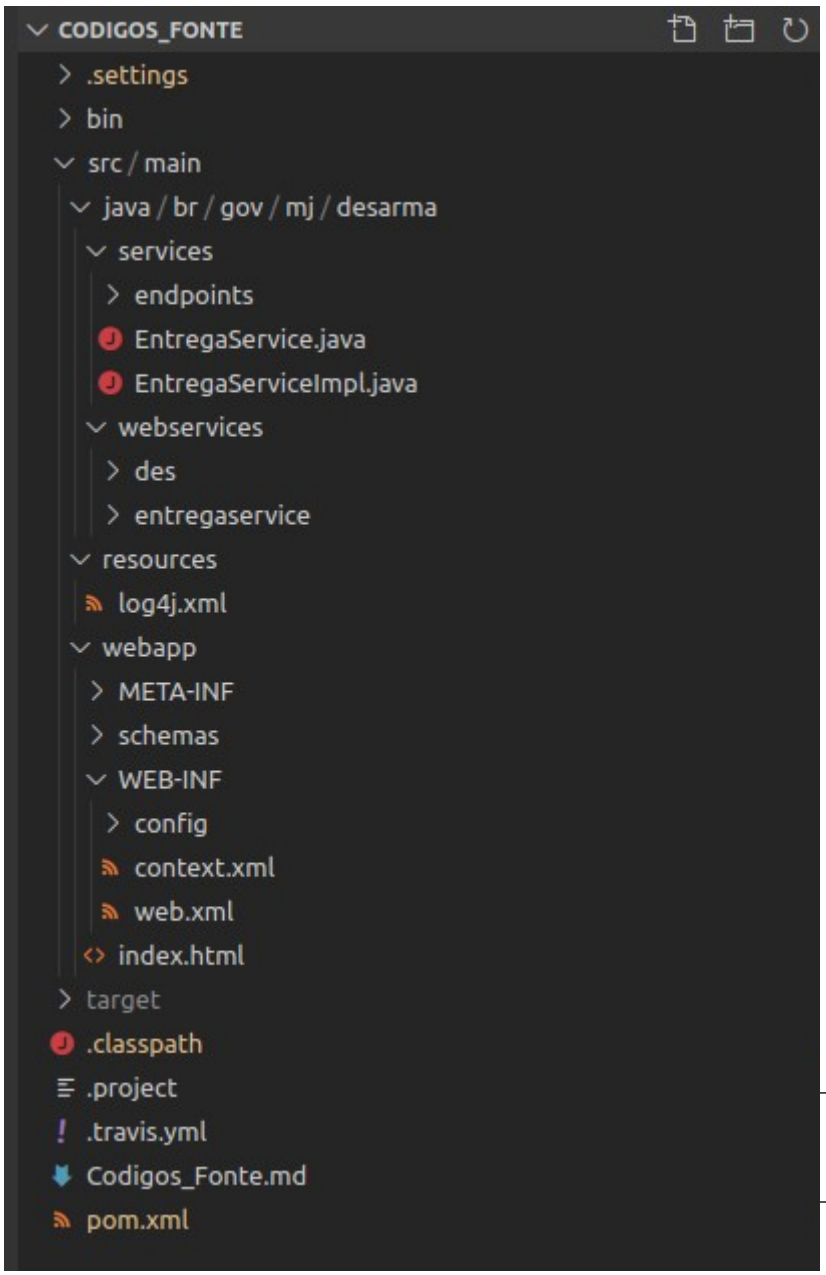

Figura 4: Estrutura do projeto em segundo escalão

3.4 Componente Desarma-Scheduler

Componente contempla comportamento único para processamento de dados com a utilização de agendamentos, sua estrutura está altamente coesa e com finalidade específica. Há somente um pacote na estrutura deste projeto, sendo este destinado a sua utilização finalística.

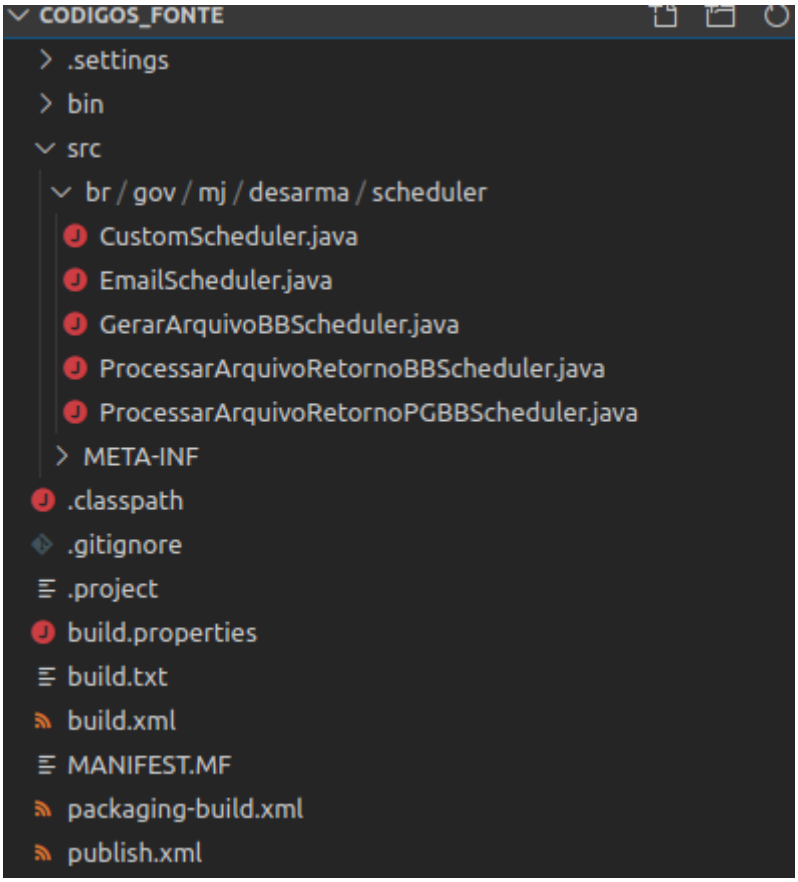

Figura 5: Estrutura do projeto em segundo escalão

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
Comp-Desarma			
Java	1.5	Linguagem de programação.	
ejb-api	1.3	Enterprise Java Bean	
Hibernate	3	Framework ORM	
JBoss	4.0.3	Servidor de aplicação JEE.	
Apache Axis	1.0.0	Encapsulamento do framework, utilizado para criação e consumir WebServices.	
MS SQL Server		Banco de dados relacional	
Desarma-Web			
Java	1.5	Linguagem de programação.	
Struts	1.3.8	Framework MVC	
Displaytag	1.1.1	Biblioteca para tratar a visualização de tabelas dinâmicas	
jcaptcha	1.0	Componente para	

		gerar captcha	
jasperreports	1.1.0	Framework para criação de relatórios	
Desarma-WS-Infoserg			
Java	1.5	Linguagem de programação.	Java
Springframwork	3.1.1	Utilizado neste projeto para prover container de IOC e MVC.	

3.3 Modelagem de dados

A estrutura de banco de dados esta composta por 34 tabelas e um único schema. Há 4 tabelas que não apresentam relacionamentos nesta estrutura, sendo elas: Indenizacao_temp, Log_FalhaRetorno, Email, InformacaoVoucher. As tabelas Indenizacao_temp e InformacaoVoucher não estão mapeadas nos componentes que compõe o projeto e podem não estar sendo utilizadas.

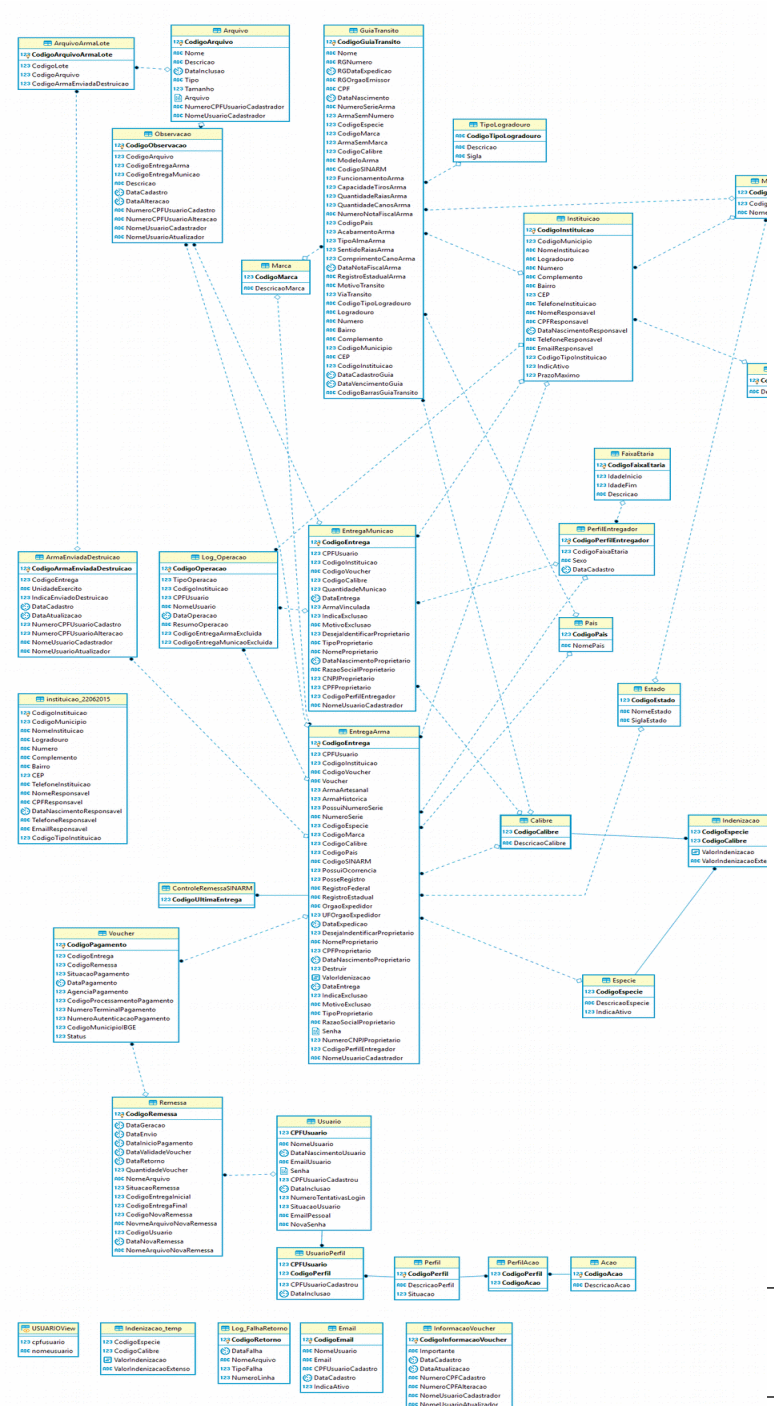

Figura 6: MER Banco Desarma

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 Comp-Desarma

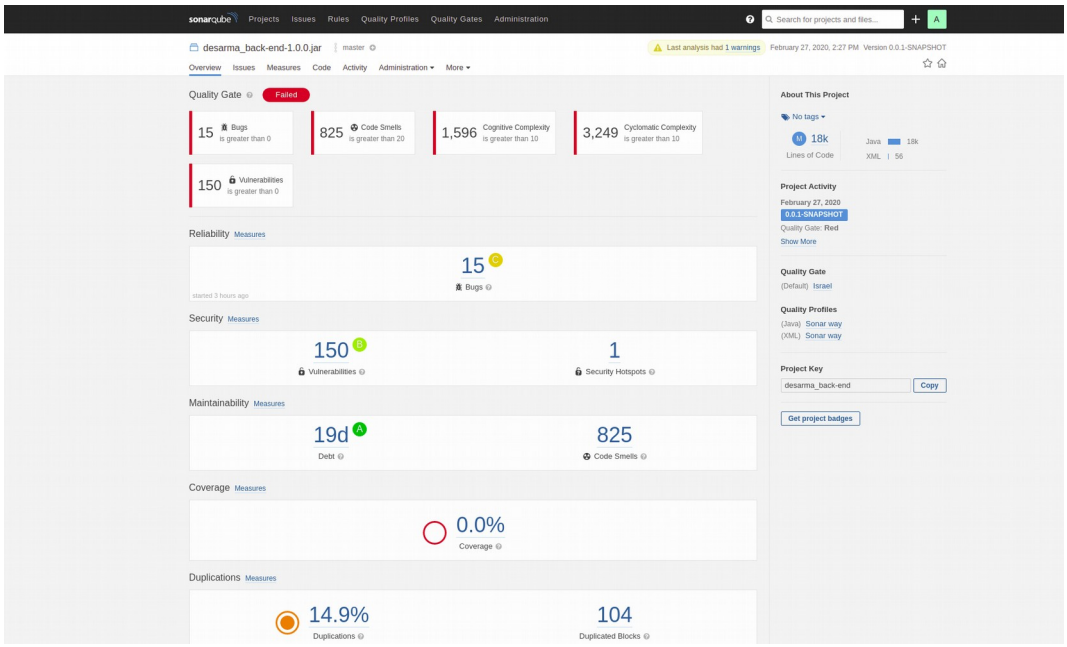

Figura 7: Análise estática de código

- 15 bugs;
- 825 violações de más práticas;
- 1596 violações de complexidade cognitiva (dificuldade de entendimento de código);
- 3249 violações de complexidade ciclomática (complexidade de código);
- 150 vulnerabilidades;
- 14.9% de duplicação de código;

4.1.2 Desarma-Web

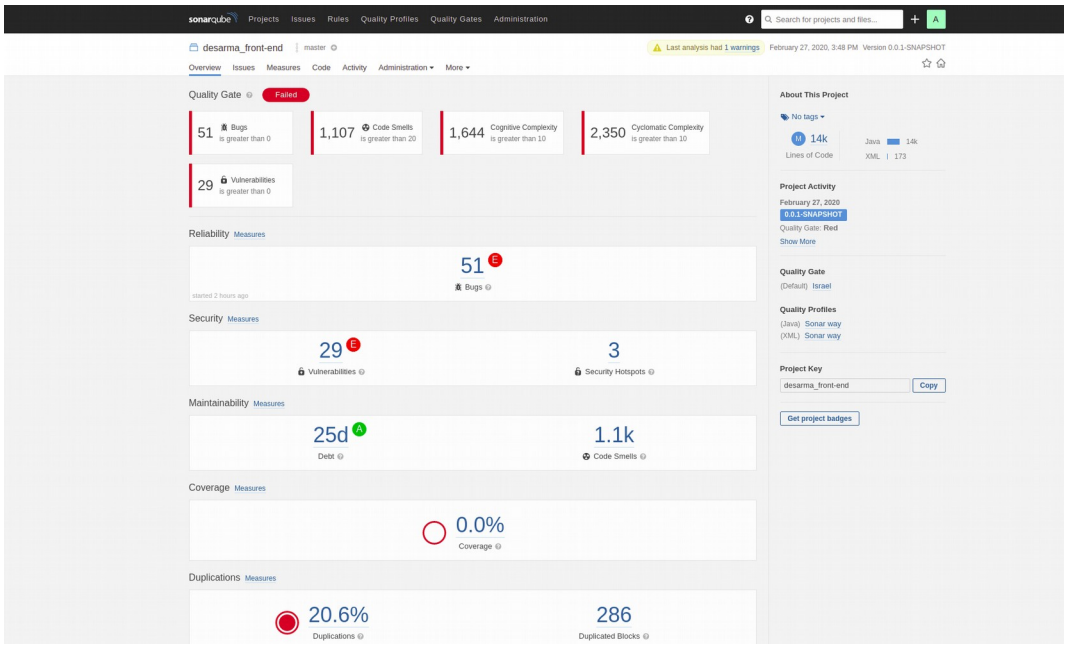

Figura 8: Análise estática de código

- 51 bugs;
- 1107 violações de más práticas;
- 1644 violações de complexidade cognitiva (dificuldade de entendimento de código);
- 2350 violações de complexidade ciclomática (complexidade de

código);

- 29 vulnerabilidades;
- 20.6% de duplicação de código;

4.1.3 Desarma-WS-Infoseg

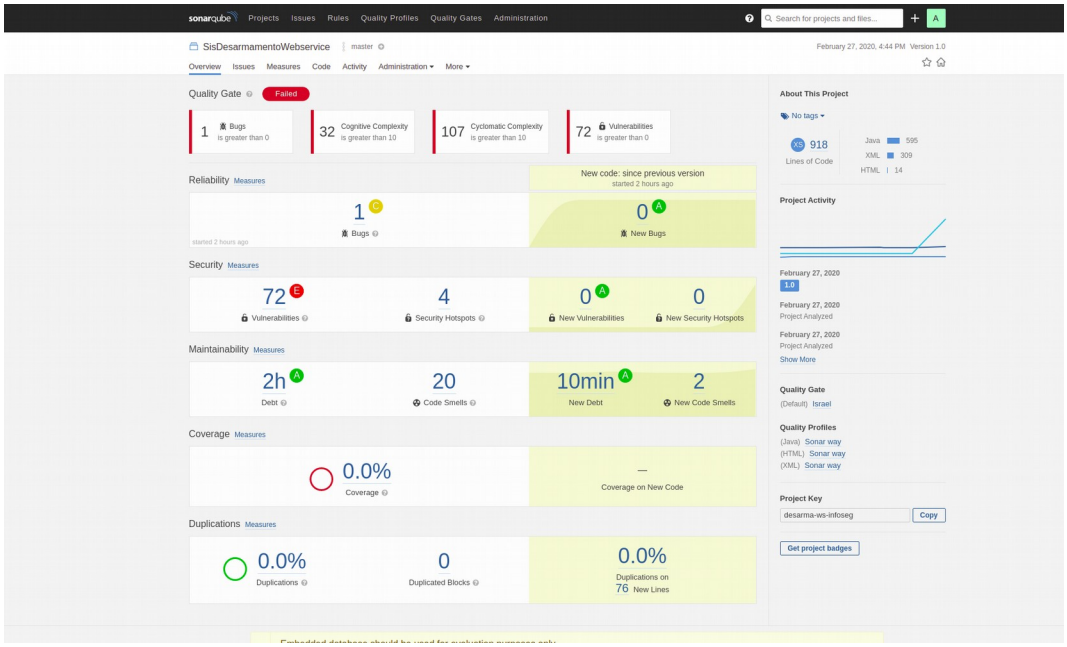

Figura 9: Análise estática de código

- 1 bugs;
- 32 violações de complexidade cognitiva (dificuldade de entendimento de código);
- 107 violações de complexidade ciclomática (complexidade de código);
- 72 vulnerabilidades;

4.1.4 Desarma-Scheduler

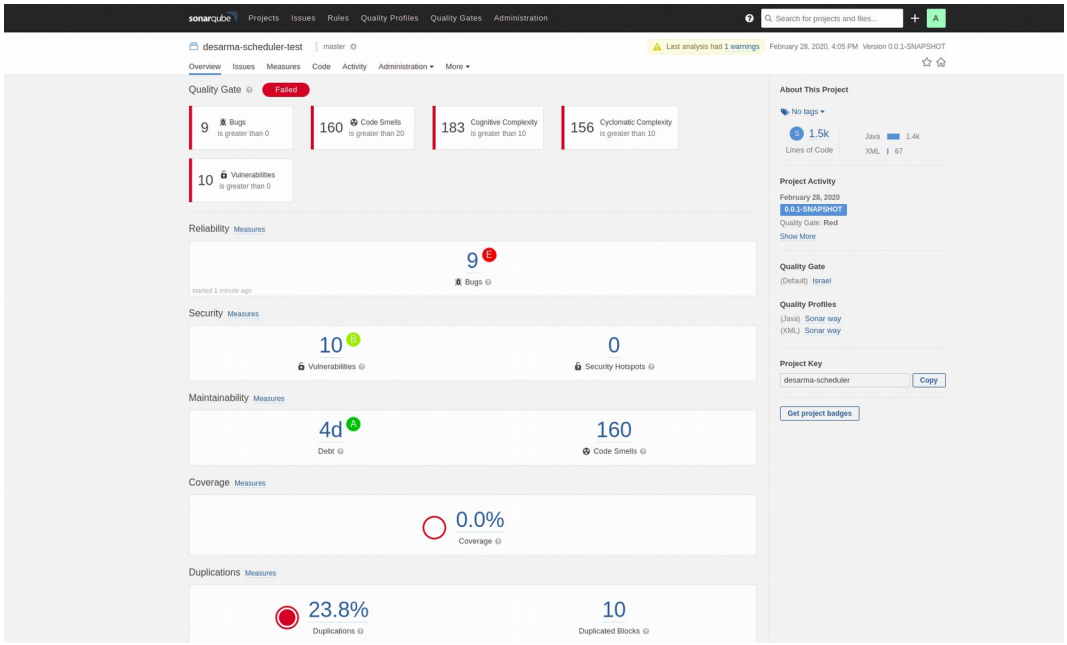

Figura 10: Análise estática de código

- 9 bugs;
- 106 violações de más práticas;
- 183 violações de complexidade cognitiva (dificuldade de entendimento de código);
- 156 violações de complexidade ciclomática (complexidade de código);
- 10 vulnerabilidades;
- 23.8% 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
Comp-Desarma				
dom4j-1.6.1.jar	HIGH	1	Highest	25
commons-collections-2.1.1.jar	CRITICAL	3	Highest	19
axis-1.0.jar	MEDIUM	4	Highest	13
Desarma-Web				
commons-beanutils-1.7.0.jar	HIGH	2	Highest	20
myfaces-api-1.1.0.jar	MEDIUM	1	Highest	12
commons-collections-3.1.jar	CRITICAL	3	Highest	25
commons-fileupload-1.1.1.jar	CRITICAL	5	Highest	30
log4j-1.2.13.jar	CRITICAL	1	Highest	18
poi-3.0-FINAL.jar	HIGH	8	Highest	24
struts-core-1.3.8.jar	HIGH	20	Highest	27
struts-el-1.3.8.jar	HIGH	6	Highest	25
jstl-1.0.2.jar	HIGH	1		20
struts-tiles-1.3.8.jar	HIGH	6	Highest	29
standard-1.1.0.jar	HIGH	1	Highest	22
axis-1.0.0.jar	MEDIUM	3	Highest	14
jasperreports-1.1.0.jar	HIGH	4		17
poi-2.5.1-final-20040804.jar	HIGH	8	Highest	14
Desarma-WS-Infoseg				
log4j-1.2.16.jar	CRITICAL	1	Highest	29
spring-core-3.1.1.RELEASE.jar	CRITICAL	13	Highest	28
spring-asm-3.1.1.RELEASE.jar	CRITICAL	11	Highest	27
commons-collections-3.2.jar	CRITICAL	3	Highest	30
spring-oxm-3.1.1.RELEASE.jar	CRITICAL	13	Highest	28
spring-ws-core-2.1.4.RELEASE.jar	CRITICAL	1	Low	27
spring-web-3.2.4.RELEASE.jar	CRITICAL	10	Highest	26
spring-webmvc-3.2.4.RELEASE.jar	CRITICAL	9	Highest	28
xercesImpl-2.8.0.jar	0.0	2	Low	67
xalan-2.7.0.jar	HIGH	1	Highest	34
dom4j-1.6.1.jar	HIGH	1	Highest	25
Desarma-Scheduler				
jboss-as-ejb3-6.1.0.Final.jar	HIGH	1	Low	30
dom4j-1.6.1.jar	HIGH	1	Highest	25
bsh-1.3.0.jar	HIGH	1	Low	15
quartz-1.8.3.jar	CRITICAL	1		22
hornetq-core-2.1.2.Final.jar	MEDIUM	1	Highest	15
netty-3.2.1.Final.jar	0.0	7	Highest	28
commons-beanutils-1.8.0.jar	HIGH	2	Highest	35
opensaml-1.1b.jar	HIGH	4	Highest	13
log4j-1.2.16.jar	CRITICAL	1	Highest	29
jboss-as-security-6.1.0.Final.jar	HIGH	1	Highest	28
xmlsec-1.4.3.jar	MEDIUM	1		15
xercesImpl-2.9.1.jar	0.0	2	Low	67
commons-collections-3.1.jar	CRITICAL	3	Highest	25

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

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).

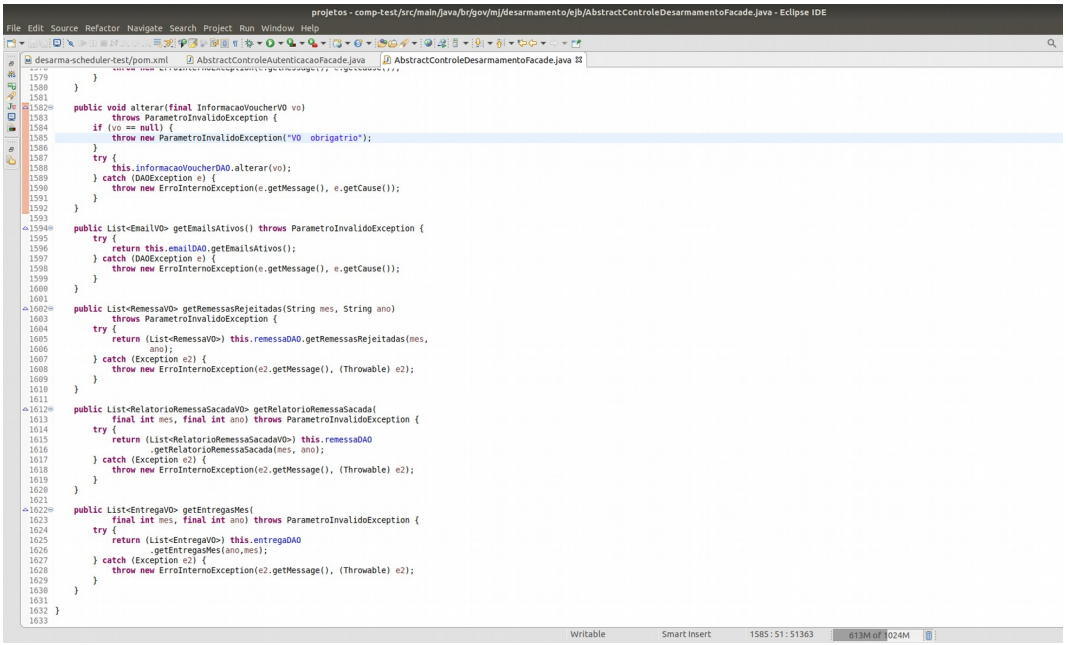

Figura 11: Baixa coesão - AbstrctControleDesarmamentoFacede

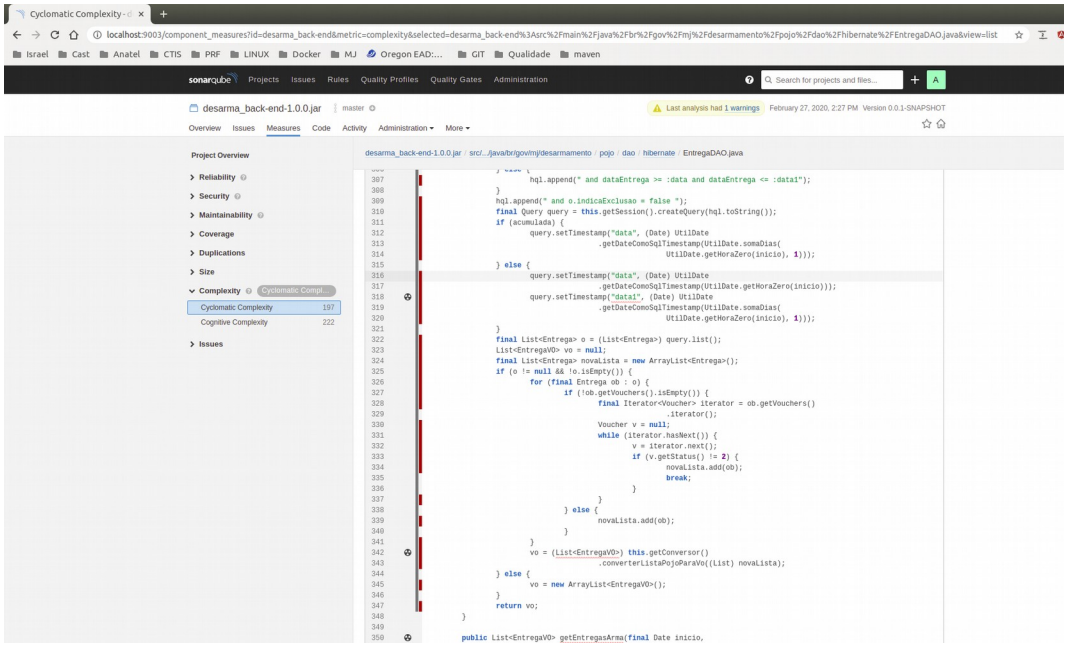

Figura 12: Alta complexidade ciclomática - EntregaDAO

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.

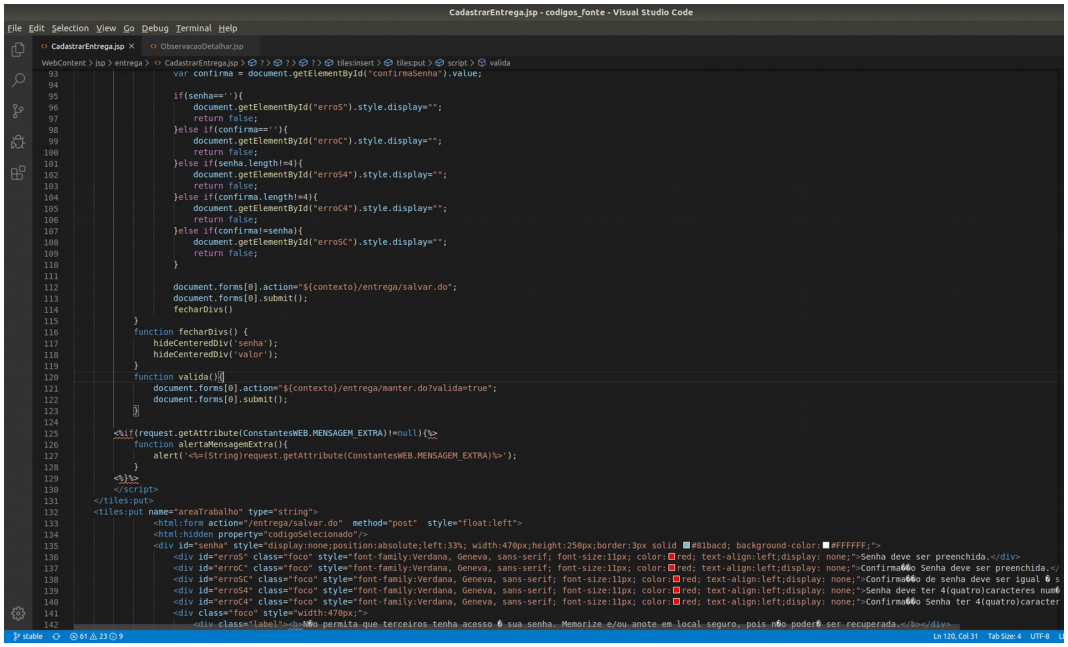

Figura 13: Arquivos JSP contendo Scriptlets Java e código Javascript

4.4.2 Confiabilidade

Não é possível avaliar se está havendo controle transacional a nível de banco de dados, uma vez que todas as classes de persistência herdam da classe abstrata AbstractHibernateDAO estando esta contida no componente de arquitetura de referência.

O código fonte não evidencia tratamento de controle transacional a nível de aplicação e caso não ocorra na arquitetura de referência, não há como garantir a consistência de dados em funcionalidades que necessitem mais de uma operação em banco de dados.

4.4.3 Performance e estabilidade

Aplicação não apresenta recurso de paginação de consultas em banco de dados, 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.

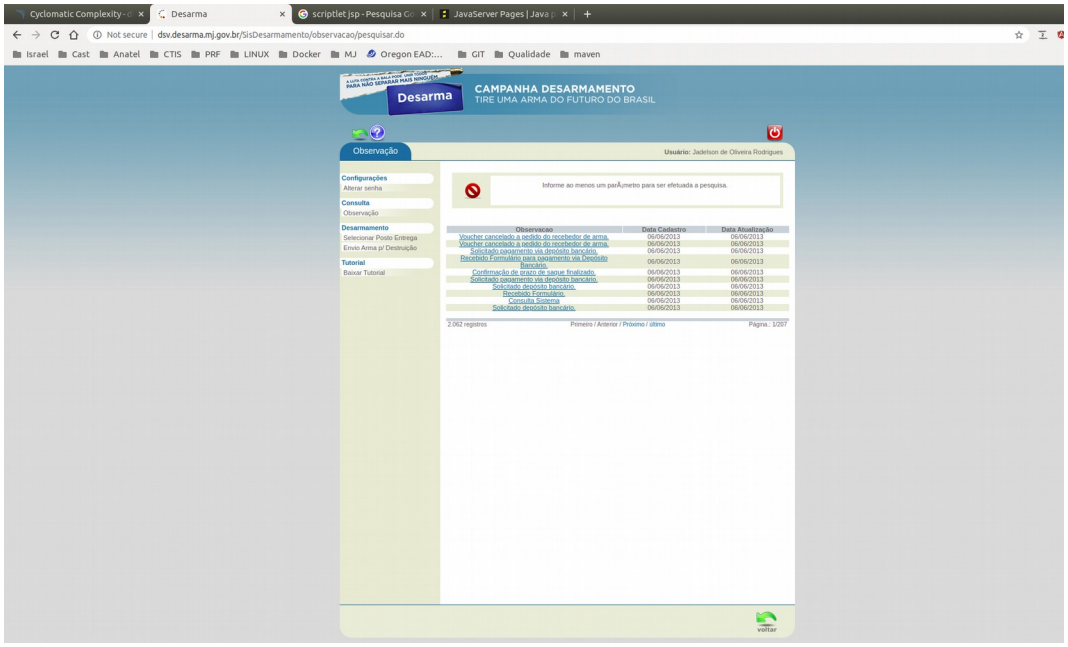

Figura 14: Paginação efetuada no componente web (displaytag)

A aplicação utiliza em diversas funcionalidades o armazenamento de informações em sessões de usuário. Além do elevado consumo de memória que este recurso pode causar, não é possível afirmar que todos os objetos setados em sessão estão sendo destruídos ao término de sua utilização.

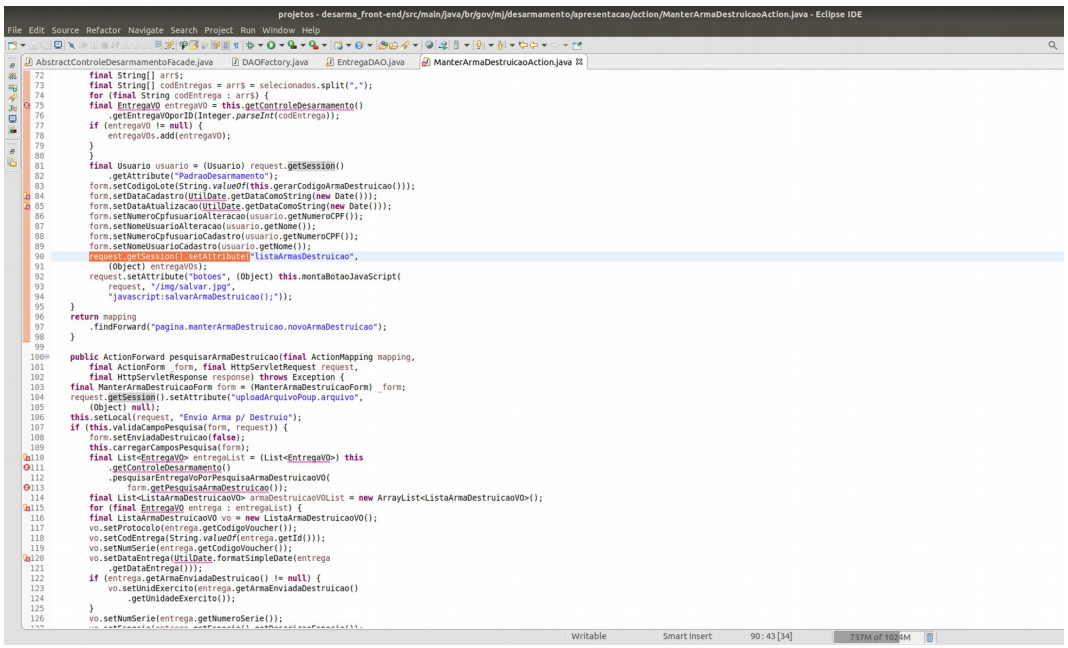

Figura 15: Armazenamento de dados em sessão de usuário

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.

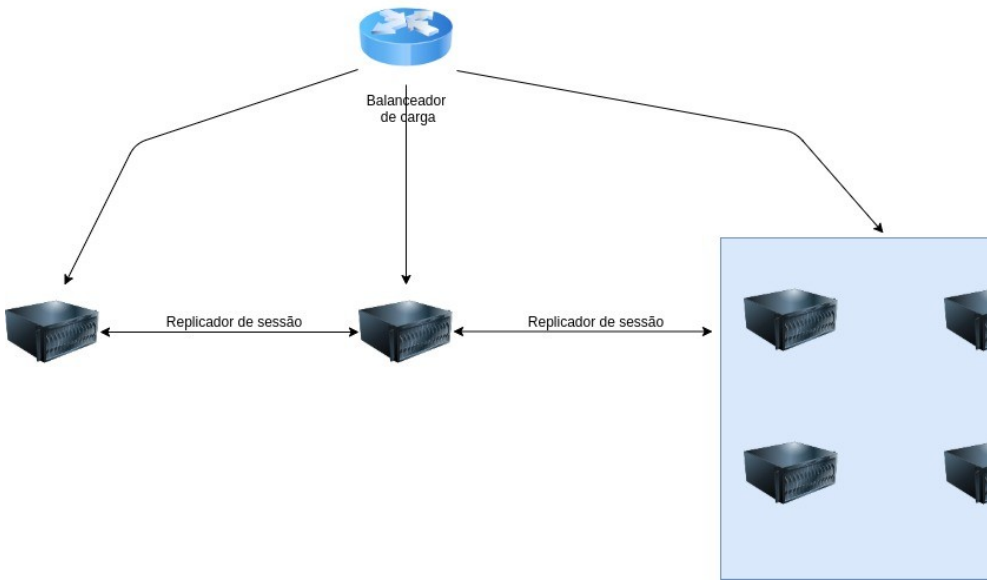

Figura 16: Escalabilidade do monólito

4.4.3 UX – User experience

A aplicação apresenta diversas ocorrências de encode dentro do código fonte e isso transparece em sua utilização.

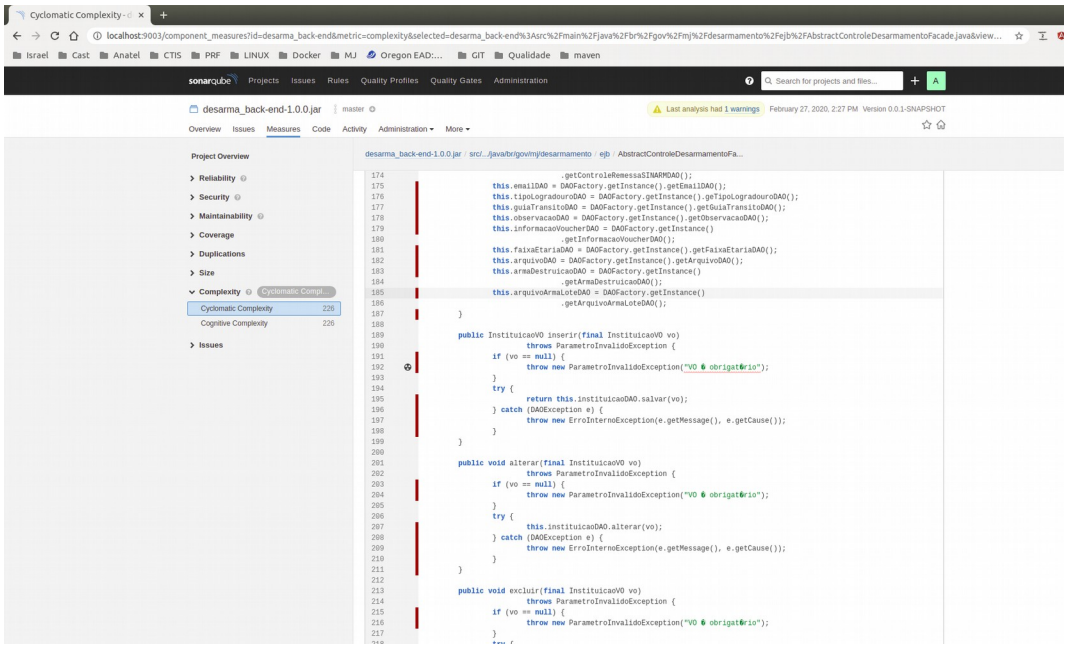

Figura 17: Erros de encode no código fonte

Figura 18: Erros de encode na aplicação web

Incompatibilidade tecnológica com os navegadores atuais trazem dificuldades e/ou impedimento de funcionalidades no sistema.

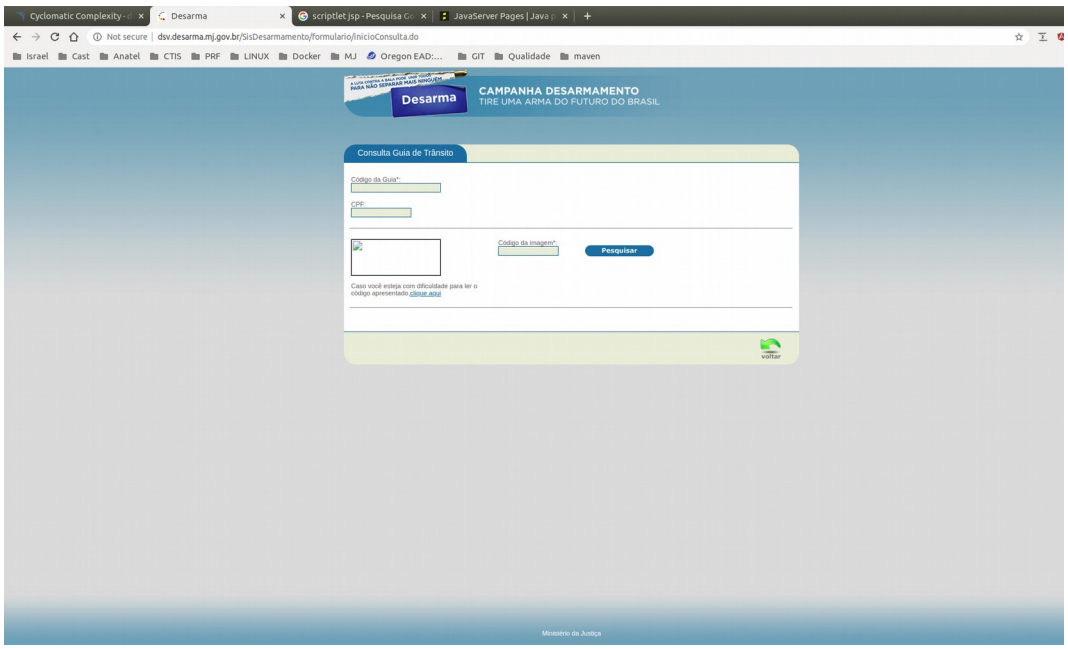

Figura 19: Erro de visualização do Captcha

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 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. Ainda sobre as dependências, recomenda-se que seja removido as dependências que não estão sendo utilizadas no projeto a exemplo o driver jdbc do PostgreSQL, uma vez que neste projeto utiliza-se o SGBD Oracle.

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.

Recomenda-se que seja utilizado o gerenciador de build/dependência Maven em detrimento ao Apache Ant/Ivy, a utilização do utilitário Apache Ivy trás consigo configurações extras nas IDEs de desenvolvimento e no gerenciador de integração continua.

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.

Por ser um legado com baixa utilização, acredita-se que não haja interesse em substituir a tecnologia deste projeto, contudo 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.