




Ministério da Justiça

Projeto: e-Certidão

Nota Técnica

MJ	e-Certidão - Nota Técnica	
-----------	----------------------------------	------------------------------------------------------------------------------------

Revisão	Descrição	Autor	Data
1.0	Construção do documento	Israel Branco	02/04/2020

1 Sumário

2 Introdução.....	4
3 Apresentação do cenário atual.....	5
3.1 Componentes.....	7
3.2 Tecnologias utilizadas.....	9
3.3 Modelagem de dados.....	10
4 Análise técnica.....	11
4.1 SonarQube.....	11
4.2 OWASP ZAP.....	13
4.3 Análise sobre os resultados.....	14
4.3.1 Manutenibilidade de código.....	14
4.3.3 Performance e estabilidade.....	17
4.3.3 Escalabilidade.....	18
4.3.3 UX - User experience.....	18
5 Recomendações.....	20
6 Conclusão.....	21

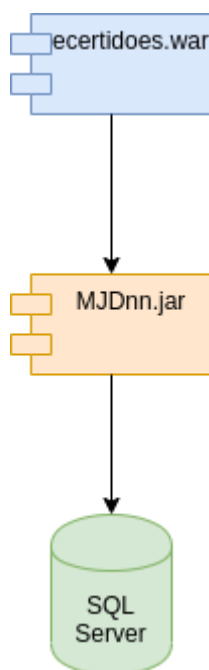
2 Introdução

Este documento visa reportar o resultado da análise efetuada no sistema e-Certificado. Para este estudo foram desconsiderados todo o contexto negocial ao qual a ferramenta está inserida juntamente com o ambiente ao qual a ferramenta opera em ambiente produtivo, 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.

3 Apresentação do cenário atual

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

O sistema e-Certidão criado para trabalhar em ambiente web sob protocolo HTTP/HTTPS está dividido em dois componentes, sendo eles um componente web empacotado em arquivo WAR (WEB archive) e outro componente EJB empacotado em arquivo JAR (Java archive), sendo o Microsoft SQL Server o banco de dados relacional utilizado.



*Figura 1:
Diagrama
de
component
es*

O componente web está disponibilizado no repositório GIT <http://git.mj.gov.br/STEFANINI/E-CERTIDOES> sendo a branch stable utilizada para esta análise. Não foi encontrado código fonte para o componente MJDnn.jar, para composição do diagrama a seguir foi utilizado técnica de engenharia reversa.

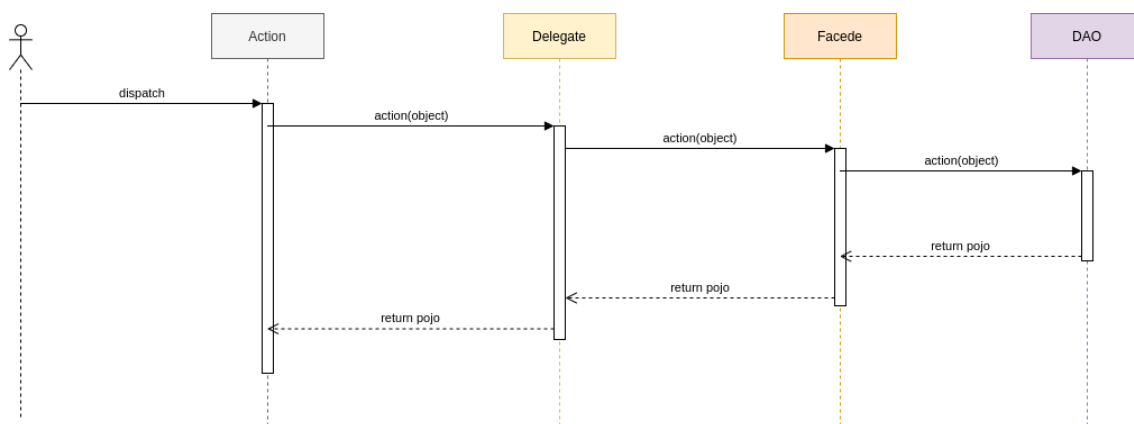
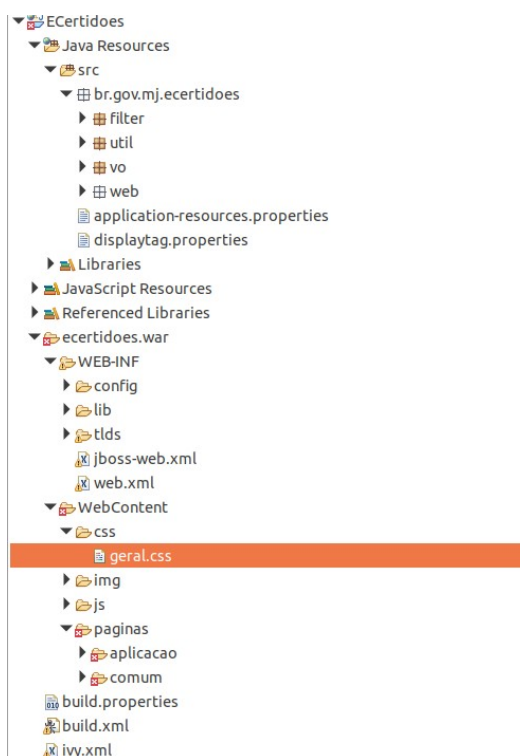


Figura 2: Diagrama de sequências

3.1 Componentes

O modulo WEB é composto por páginas JSP, arquivos JavaScript e CSS e controladores responsáveis pela iteração com os formulários JSP.

O projeto está organizado de forma simples com boa segregação dos pacotes e demais estruturas web (páginas jsp, arquivos css e javascript). Esta organização favorece as manutenções corretivas e evolutiva da aplicação, facilita também o entendimento do novos integrantes as equipes de desenvolvimento.



*Figura 3: Estrutura do projeto
WEB*

Sendo o componente MJDnn o responsável por toda operacionalização das regras de negócio e acesso a dados, sua estrutura está bem dividida e organizada.

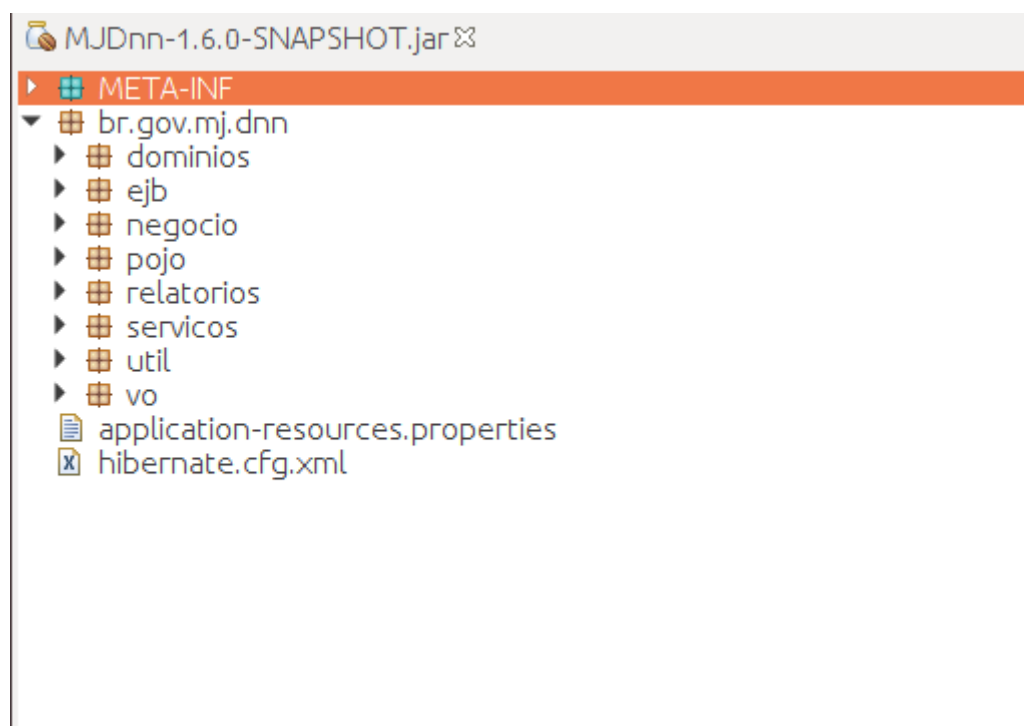


Figura 4: Estrutura do componente MJDnn

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	1.3.8	Framework MVC	
jTDS	1.2	Driver jdbc para Microsoft SQL Server	
Displaytag	1.1.1	Biblioteca para tratamento de tabelas dinâmicas (data table)	
Jasper Reports	1.1.0	Biblioteca para trabalhar com a geração de relatórios.	
EJB	2.x	Componente da plataforma JavaEE	
Hibernate	2.x	Framework ORM	



A estrutura de banco de dados esta composta por 44 tabelas em um único schema, a tabela ***NumeroProtocolo*** que não apresentam relacionamentos nesta estrutura.

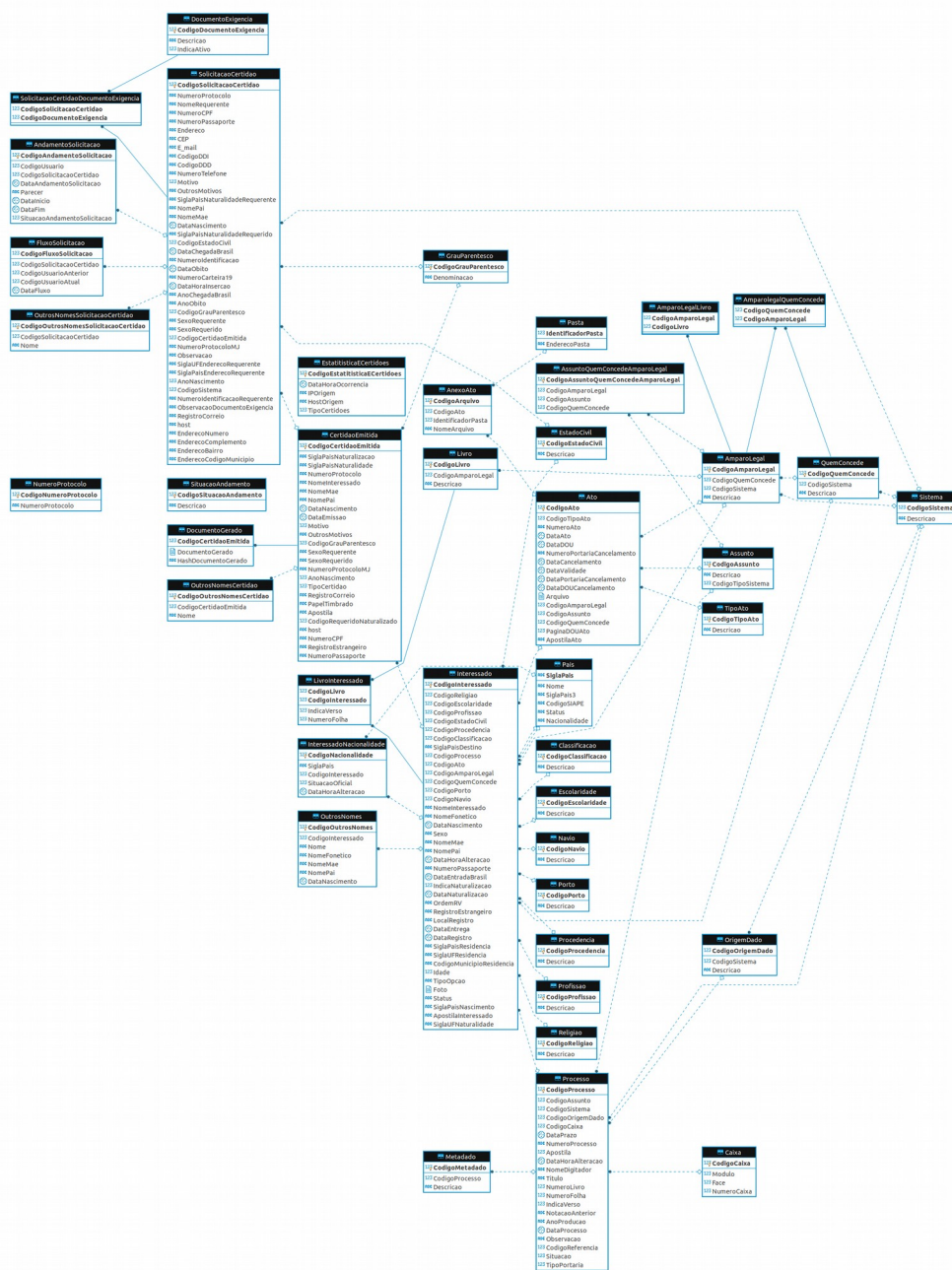


Figura 5: Mer - Banco MJCorporativo - Schema DNN

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):

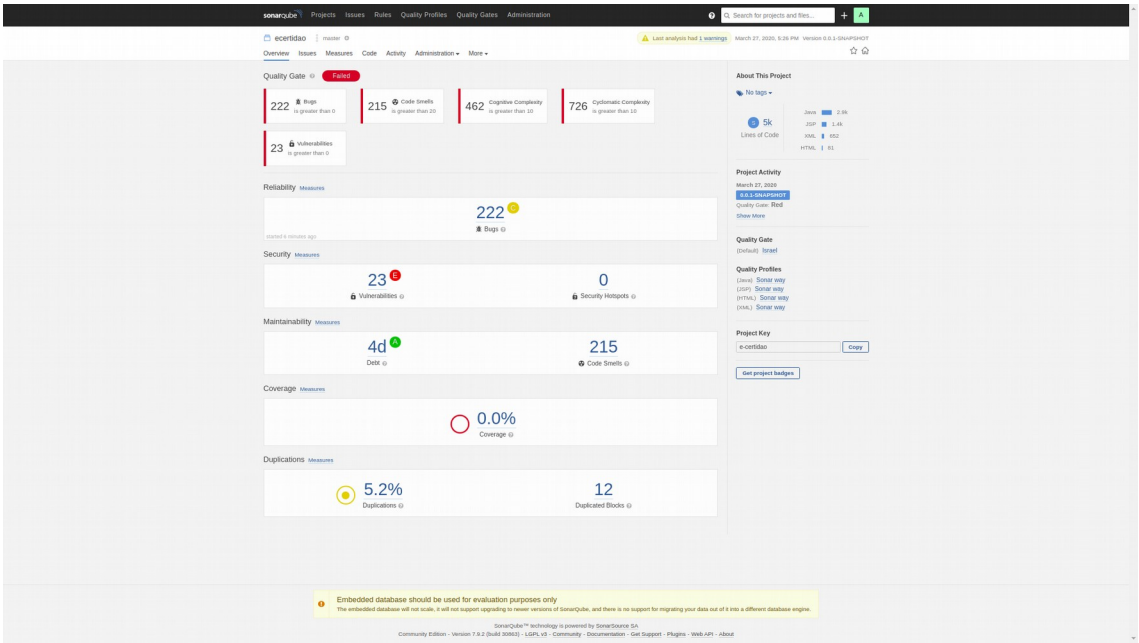


Figura 6: Análise estática de código

- 222 bugs;
- 215 violações de más práticas;
- 462 violações de complexidade cognitiva (dificuldade de entendimento de código);
- 726 violações de complexidade ciclomática (complexidade de código);
- 23 vulnerabilidades;
- 5.2% 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
commons-beanutils-1.7.0.jar	HIGH	2	Highest	20
myfaces-api-1.1.0.jar	MEDIUM	1	Highest	12
commons-fileupload-1.0.jar	CRITICAL	5	Highest	24
commons-collections-3.1.jar	CRITICAL	3	Highest	25
log4j-1.2.13.jar	HIGH	1	Highest	18
poi-3.0-FINAL.jar	HIGH	8	Highest	24
jasperreports-1.1.0.jar	HIGH	4		17
poi-2.5.1-final-20040804.jar	HIGH	8	Highest	14
struts-core-1.3.8.jar	HIGH	21	Highest	27
struts-el-1.3.8.jar	HIGH	7	Highest	25
jstl-1.0.2.jar	HIGH	1		20
standard-1.0.2.jar	HIGH	1	Low	18
struts-tiles-1.3.8.jar	HIGH	7	Highest	29
jquery.min.js	MEDIUM	3		3

A planilha acima apresenta as vulnerabilidades encontradas nas

dependências de cada módulo, o detalhamento encontra-se no Anexo I deste documento.

4.2 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.

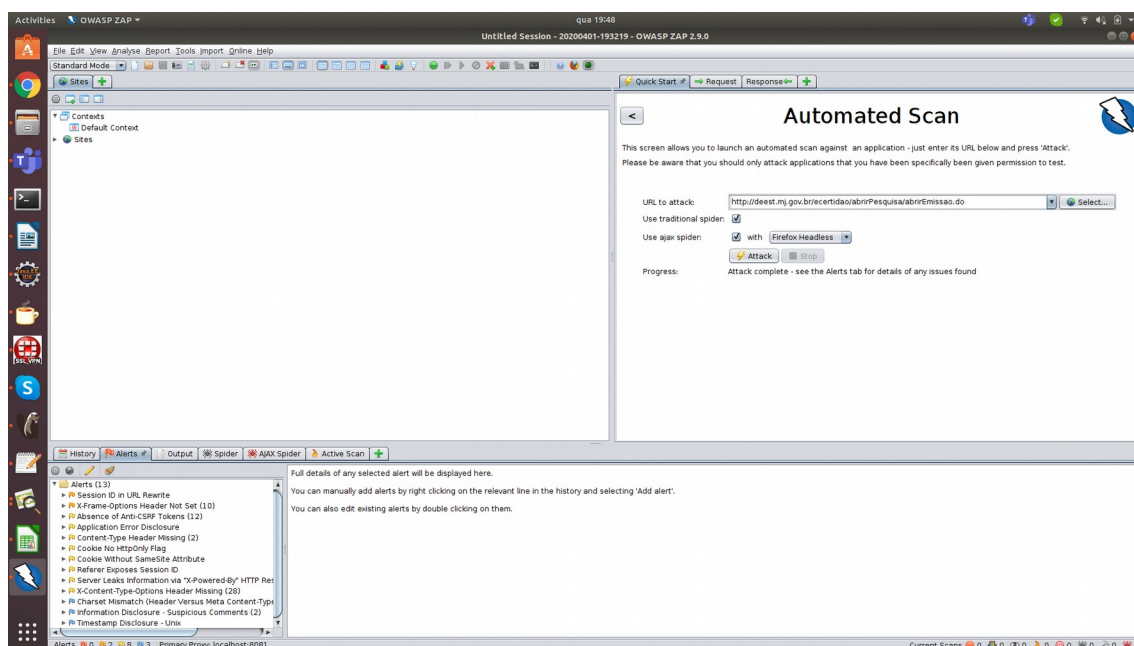


Figura 7: Análise de intrusão

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

- 0 vulnerabilidades de severidade alta;
- 2 vulnerabilidades de severidade média;
- 8 vulnerabilidades de baixa média;
- 3 vulnerabilidades a nível informativo;



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 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).

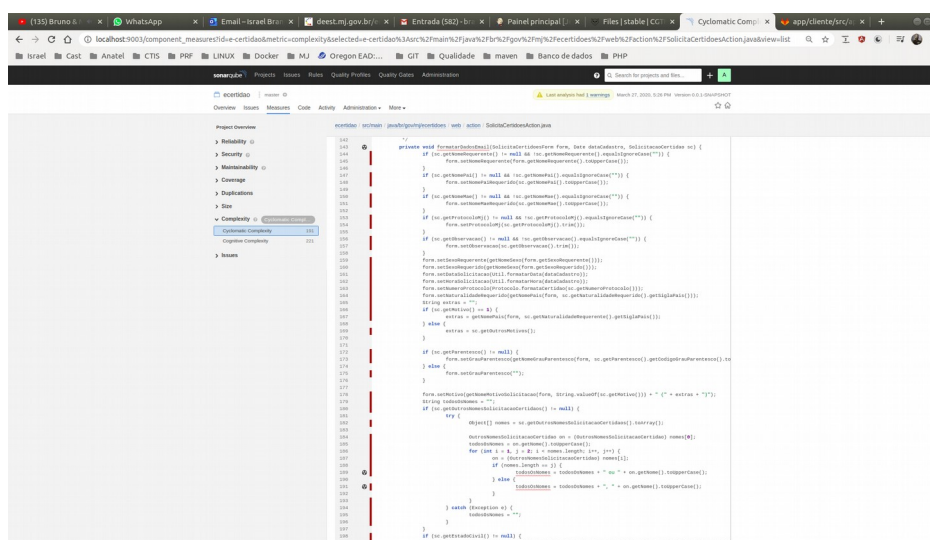
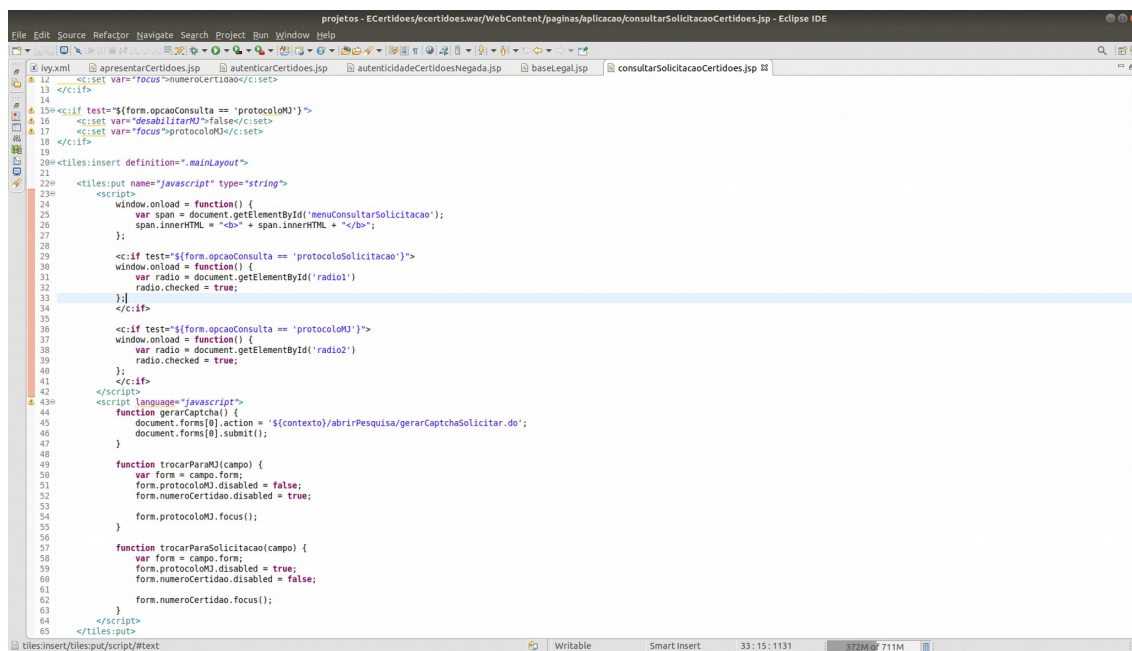


Figura 8: Complexidade ciclomática aplicação WEB

A existência de scriptlets Java nas páginas contribui para a difícil manutenibilidade do código.



```

12 <script var="focus">numeroCertidao</c:set>
13 </c:if>
14
15 <:if test="${form.opcaoConsulta == 'protocoloMJ'}">
16 <:set var="desabilitarMJ">false</c:set>
17 <:set var="focus">protocoloMJ</c:set>
18 </c:if>
19
20 <tiles:insert definition="mainLayout">
21
22 <tiles:put name="javascript" type="string">
23 <script>
24 window.onload = function() {
25     var span = document.getElementById('menuConsultarSolicitacao');
26     span.innerHTML = "<div>" + span.innerHTML + "</div>";
27 };
28
29 <:if test="${form.opcaoConsulta == 'protocoloSolicitacao'}">
30 window.onload = function() {
31     var radio = document.getElementById('radio1')
32     radio.checked = true;
33 };
34 </c:if>
35
36 <:if test="${form.opcaoConsulta == 'protocoloMJ'}">
37 window.onload = function() {
38     var radio = document.getElementById('radio2')
39     radio.checked = true;
40 };
41 </c:if>
42 </script>
43 <script language="javascript">
44 function gerarCaptcha() {
45     document.forms[0].action = '${contexto}/abrirPesquisa/gerarCaptchaSolicitar.do';
46     document.forms[0].submit();
47 }
48
49 function trocarParaMJ(campo) {
50     var form = campo.form;
51     form.protocoloMJ.disabled = false;
52     form.numeroCertidao.disabled = true;
53     form.protocoloMJ.focus();
54 }
55
56 function trocarParaSolicitacao(campo) {
57     var form = campo.form;
58     form.protocoloMJ.disabled = true;
59     form.numeroCertidao.disabled = false;
60     form.numeroCertidao.focus();
61 }
62 </script>
63 </script>
64 </tiles:put>
65 </tiles:insert>

```

Figura 9: Scriptlets nas páginas JSP

A complexidade ciclométrica encontrada no componente back-end também é alta mas com a falta do código fonte, não foi possível analisá-la com as ferramentas utilizadas para a realização desta análise.

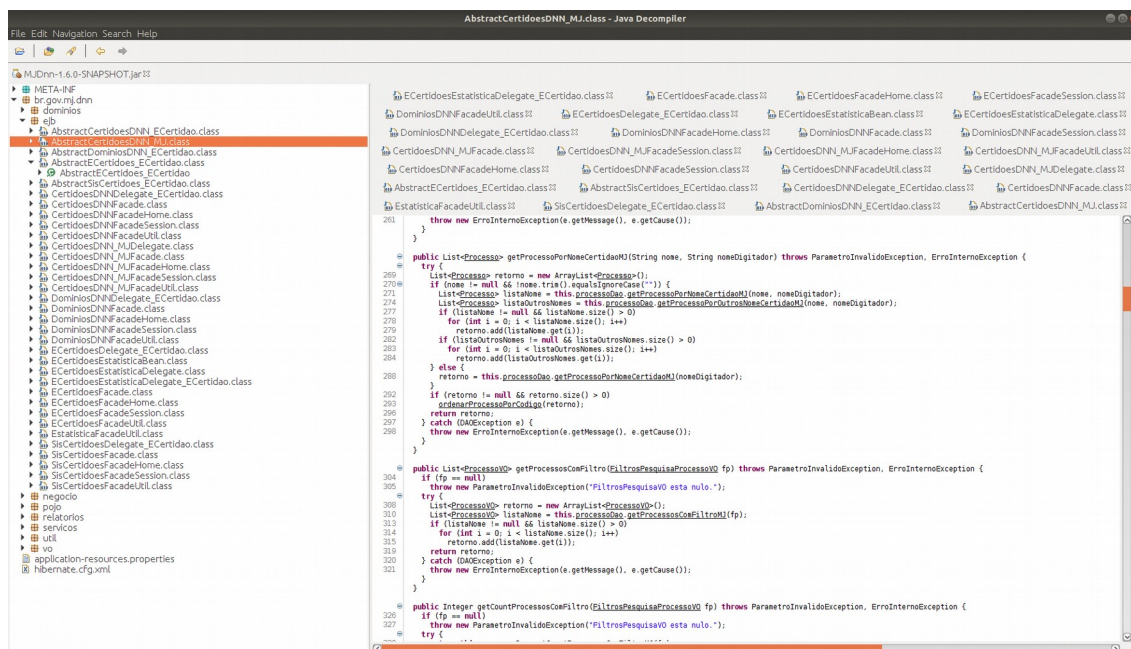


Figura 10: Complexidade ciclométrica componente MJDnn

O componente MJDnn foi construído utilizando tecnologia EJB versão 2 e Hibernate versão 2, ambas versões dado a data de sua especificação/criação não utilizam anotações (especificação JavaSE 5). A complexidade de utilização de ambos é superior as versões subsequentes dos mesmos, fator este que aumenta a complexidade e legibilidade de código e consequentemente, caem com a capacidade de manutenibilidade.

4.3.2 Confiabilidade

Todas operações em banco de dados são realizadas sob o controle transacional a nível de JDBC na camada de persistência dentro do componente MJDnn, esta tratativa garante as propriedades ACID do banco de dados.

A tratativa dada para a camada de persistência não favorece o reuso, há muito código sendo feito de forma repetitiva e a tratativa dada para o controle transacional aumenta a complexidade ciclomática do código.

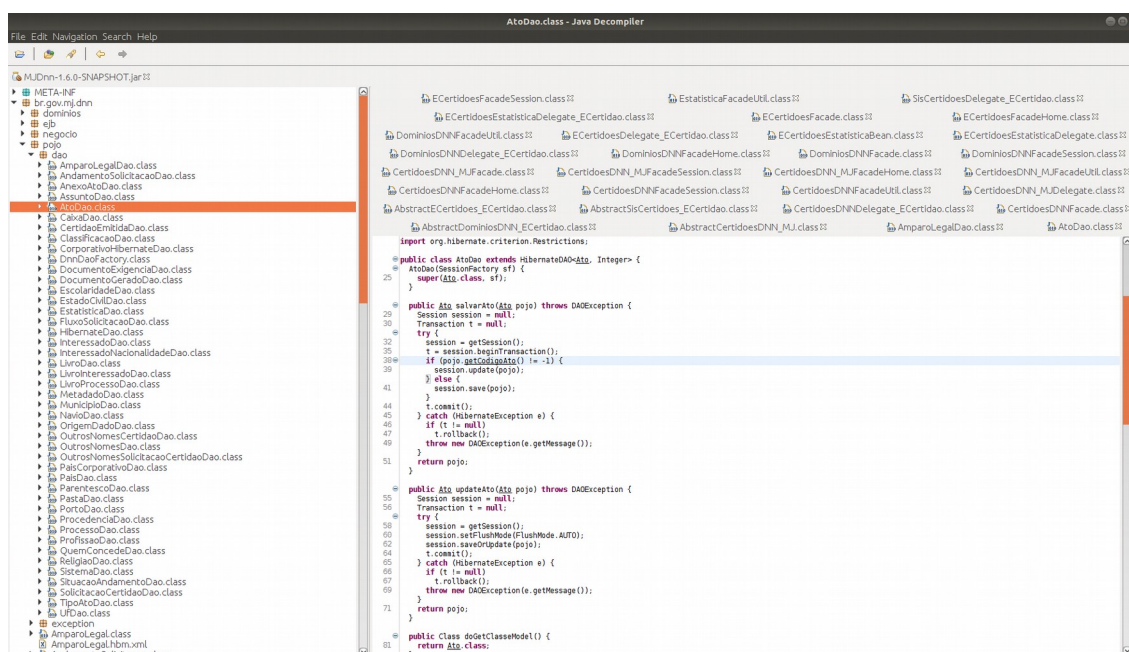


Figura 11: Exemplo - camada de persistência

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 baseada em remote EJB sem a manutenção de estado (Stateless) no componente MJDnn promove boa capacidade de escalonamento horizontal. Não há manutenção de sessão na aplicação WEB tendo em vista a sua característica negocial, essa característica facilita a escalabilidade horizontal uma vez que não há necessidade de replicação de sessão entre os nós.

A arquitetura proposta para os dois componentes promove a criação de ambientes escalonáveis, favorece a utilização de ambientes redundantes com maior probabilidade de tolerância a falhas e alta disponibilidade.

4.3.3 UX - User experience

A aplicação não apresentou incompatibilidade com a utilização de navegadores modernos, contudo apresentou erros de encode em uma das telas.

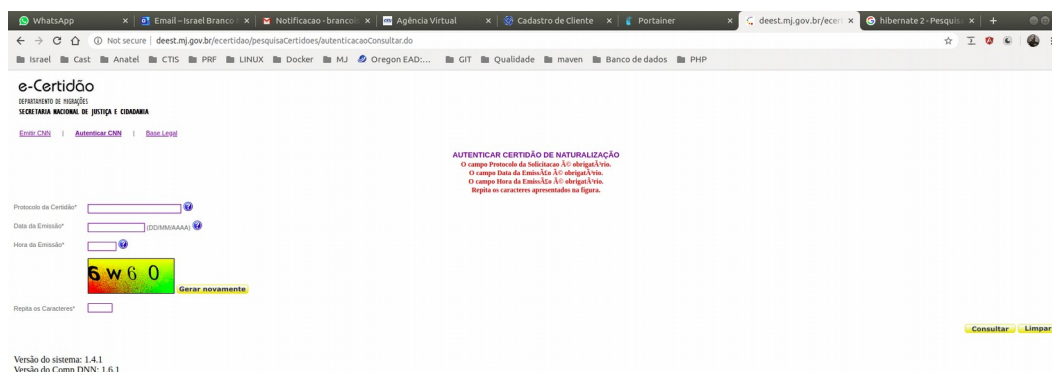



Figura 12: Erro encode - Funcionalidade: autenticação de consulta

MJ	e-Certidão - Nota Técnica	
-----------	----------------------------------	------------------------------------------------------------------------------------

Outra característica é a falta de padronização de interface com as ferramentas do governo federal, juntamente com ausência de atratividade gráfica.

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.

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.

Por fim recomenda-se que o ambiente produtivo opere com protocolo HTTPS afim de minimizar os riscos e vulnerabilidades na navegação da aplicação.

6 Conclusão

A aplicação segue padrões de codificação em conformidade com as tecnologias utilizadas, contudo a utilização destas tecnologias tais como Hibernate 2, EJB 2 e demais trazem consigo complexidade adicional ao projeto. A boa estrutura organizacional da aplicação WEB e do componente MJDnn auxilia nas manutenções corretivas/evolutivas.

Tendo em vista que este projeto utiliza tecnologias obsoletas e a inatividade da comunidade/fabricante para os frameworks, bibliotecas e demais componentes de terceiros utilizados neste projeto, fato este que pode dificultar a resolução de determinados problemas ou aumentar a curva de aprendizado de novos ingressos a equipe de desenvolvimento deste projeto.

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 e caso este projeto demande muita utilização, recomenda-se a sua reconstrução com a adoção de tecnologias mais modernas.