Ministério da Justiça

Projeto: DRCI Internet

Nota Técnica

Revisão	Descrição	Autor	Data
1.0	Construção do documento	Israel Branco	19/03/2020

1 Sumário

2 Introdução

Este documento visa reportar o resultado da análise efetuada na aplicação DRCI. 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.

3 Apresentação do cenário atual

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

O sistema DRCI está construído para funcionar em ambiente WEB com uma segregação entre as camadas de front-end e back-end, sua arquitetura esta projetada para trabalhar de forma desacoplada e distribuída.

O backend da aplicação está construído sobre a stack Java Enterprise Edition já a aplicação front-end está construída com a especificação JSF.

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

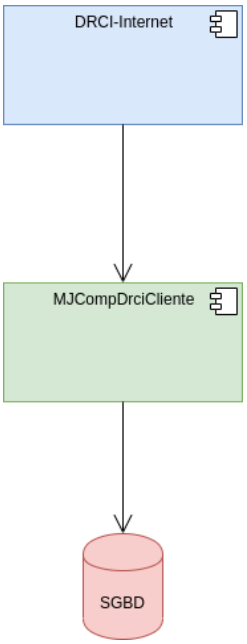

Figura 1: Diagrama de componentes

O componente back-end listado como MJCompDrciCliente trata-se de um EJB remoto e possui dependências de outros componentes de mesma características que fazem parte da arquitetura distribuída do MJ.

3.1 DRCI Internet

Esta camada representa a interface com o usuário da aplicação e sua organização não esta feita de forma lógica e intuitiva, a nomenclatura dos pacotes não induz a sua real utilização. Esta estrutura não proporciona boa capacidade produtiva para manutenções corretivas e evolutivas.

Figura 2: Estrutura do projeto front-end

3.2 MJCompDrciCliente

Este componente representa toda a camada de back-end da aplicação e na mesma analogia ao componente que representa o front-end, este também não possui boa estruturação. Há duplicações de pacotes por responsabilidades e falta de coesão em sua estruturação.

O projeto está composto da seguinte forma:

- Classes utilitárias para iteração com SGBD (Sistema Gerenciador de Banco de Dados).
- Classes POJO (Plain Old Java Object) que possuem objetivo de trafegar estado entre as camadas da aplicação (DTO (Data Transfer Object)).
- Classes que disponibilizam interfaces para a chamada de EJBs remotos para consumo da aplicação front-end com a utilização do Design Pattern Facade.

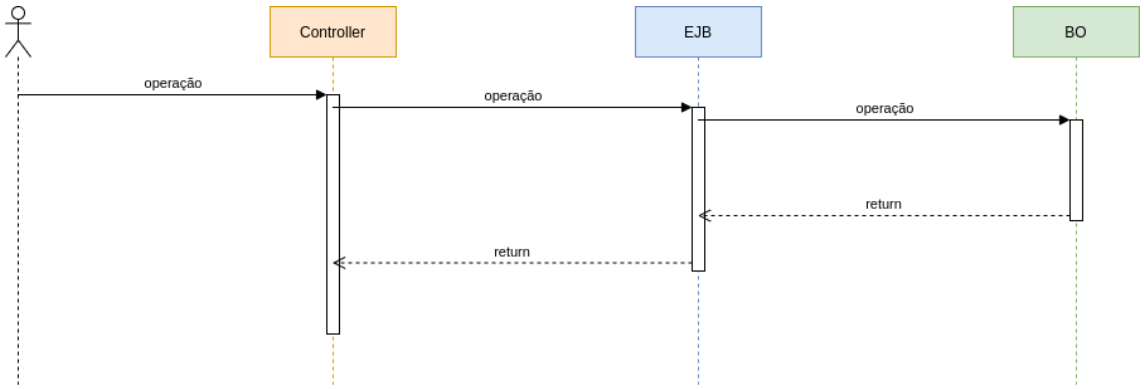

Figura 3: Sequência padrão de ações

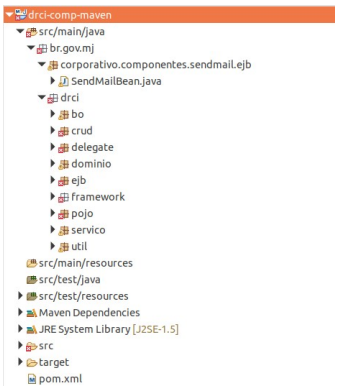

Figura 4: Estrutura do projeto back-end

3.2 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	1.6	Linguagem de programação.	
JSF	2	Framework MVC	
Javaee-API	5.0.1	Implementação JEE 5	
Jboss	4.2.3	Servidor de aplicação JEE	
Jasperreports	4.0.1	Framework para criação de relatórios	
Hibernate	3.2	Implementação ORM	
Apache POI	3.6	Componente utilizado para manipular documentos baseado em formato MS Office.	
SQLServer		Banco de dados	

		relacional	
Apache Axis	1.2	Encapsulamento do framework, utilizado para criação e consumir WebServices.	

3.3 Modelagem de dados

A estrutura de banco de dados esta composta pela utilização de de um único schema (SGDRCI).

Figura 5: Modelo entidade relacional

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:

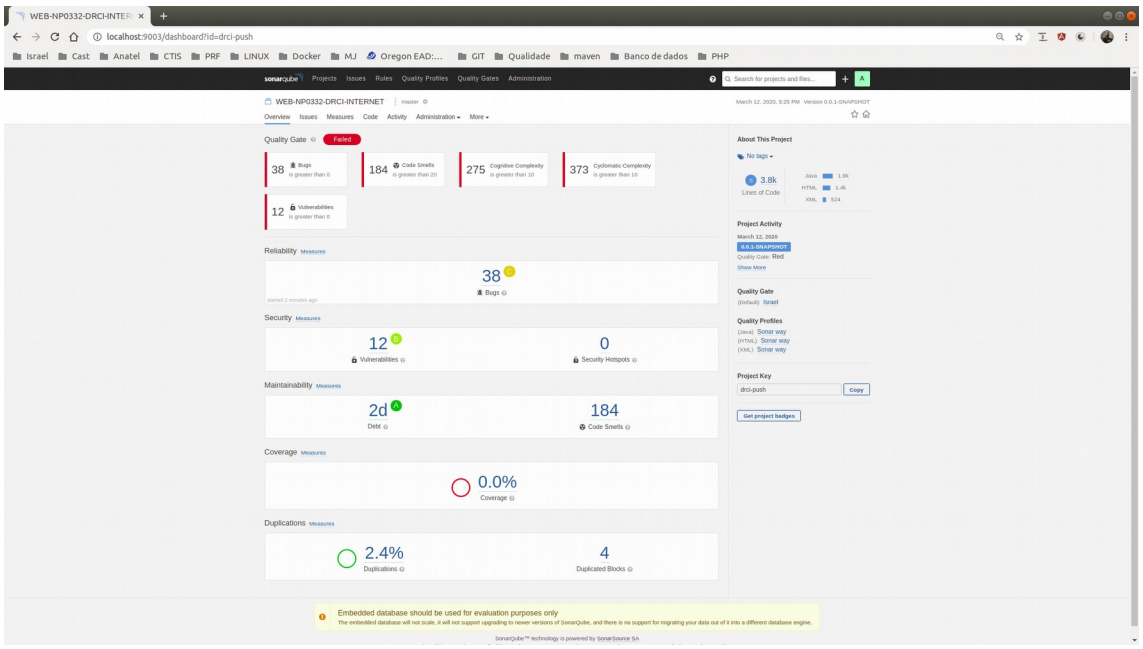

Figura 6: DRCI Insternet - Análise estática de código

- 38 bugs;
- 184 violações de más práticas;
- 275 violações de complexidade cognitiva (dificuldade de entendimento de código);
- 373 violações de complexidade ciclomática (complexidade de código);
- 12 vulnerabilidades;
- 2.4% de duplicação de código;

A ferramenta apresenta 0% de cobertura de testes, o código fonte não apresenta artefatos de testes de unidade.

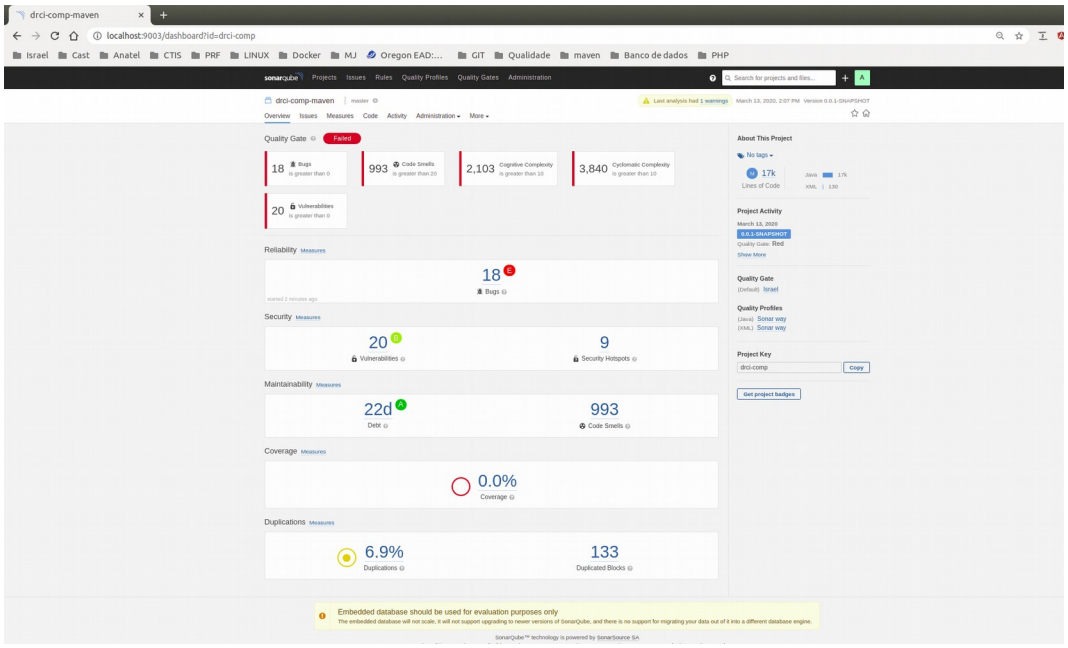

Figura 7: MJCompDrciCliente - Análise estática de código

- 18 bugs;
- 993 violações de más práticas;
- 2103 violações de complexidade cognitiva (dificuldade de entendimento de código);
- 3840 violações de complexidade ciclomática (complexidade de

código);

- 20 vulnerabilidades;
- 6.9% de duplicação de código;

A ferramenta apresenta 0% de cobertura de testes, o código fonte não apresenta artefatos de testes de unidade.

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 DRCI Internet (front-end), a seguir temos as principais informações extraídas desta análise.

Dependency	Highest Severity	CVE Count	Confidence	Evidence Count
jsf-impl-2.0.3-b05.jar	0.0	1	Highest	34
axis-saaj-1.4.jar	HIGH	4	Highest	10
jsp-api-2.1-6.1.5.jar	HIGH	11	Highest	31
dom4j-1.6.1.jar	HIGH	1	Highest	25
xalan-2.7.0.jar	HIGH	1	Highest	34
xercesImpl-2.8.1.jar	0.0	2	Low	67
axis2-saaj-api-1.3.jar	HIGH	3	Highest	20
jasperreports-4.0.1.jar	HIGH	4	High	28
commons-fileupload-1.2.1.jar	CRITICAL	5	Highest	33
bcprov-jdk14-138.jar	HIGH	13	Highest	23
poi-3.6.jar	HIGH	8	Highest	27
axis-1.2.jar	MEDIUM	3	Highest	14
primefaces-3.0.jar	MEDIUM	2	Highest	20
commons-beanutils-1.7.0.jar	HIGH	2	Highest	20
commons-collections-2.1.1.jar	CRITICAL	3	Highest	19
standard-1.1.2.jar	HIGH	1	Highest	22
jquery-ui-1.8.13.custom.min.js	MEDIUM	2		3
primefaces-3.0.jar: jquery.js	MEDIUM	3		3

A planilha acima apresenta as vulnerabilidades encontradas nas dependências do componente DRCI Internet, o detalhamento encontra-se no Anexo I deste documento.

As dependências analisadas se restringem ao componente DRCI Internet, tendo em vista que o com componente MJCompDrciCliente utiliza dependências da arquitetura de referência do MJ e demais componentes implantados no servidor de aplicação (remote EJB).

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 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).

```
public List<DrciDocumentoVO> getListaDocumentoVO(
        List<DocumentoDiligencia> listaDocumento) {
    ArrayList<DrciDocumentoVO> listaRetorno = null;
    List<DocumentoDiligencia> listaDocumentoDiligencias = listaDocumento;

    try {
        if (CollectionUtil.possuiItens(listaDocumentoDiligencias)) {
            listaRetorno = new ArrayList<DrciDocumentoVO>();
            DrciDocumentoVO documentoVO;
            Documento documento;
            for (DocumentoDiligencia documentoDiligencia : listaDocumentoDiligencias) {
                if (documentoDiligencia.getCodigoDocumento() != null
                        && documentoDiligencia.getCodigoDocumento() != 0) {
                    documento = recuperarDocumentoPeloCodigo(documentoDiligencia
                            .getCodigoDocumento());
                    Expediente expediente = (Expediente) DrciCrudDelegate
                            .getInstance().recuperar(
                                    ClassesCrud.EXPEDIENTE,
                                    documento.getCodigoDocumento());
                    if (expediente != null) {
                        documentoDiligencia.setExpediente(expediente);
                    }
                    documentoVO = carregarDocumentoVO(documento,
                            documentoDiligencia);
                } else {
                    documentoVO = carregarDocumentoVO(documentoDiligencia);
                }
                listaRetorno.add(documentoVO);
            }
            CollectionUtil.sortList(listaRetorno,
                    "codigoDocumentoDiligencia");
        }
    } catch (Exception e) {
        e.printStackTrace();
        addMensagem("Erro ao carregar lista de Documentos",
                FacesMessage.SEVERITY_ERROR);
    }

    return listaRetorno;
}
```

Figura 8: Complexidade ciclomática - DetalharProcessoMB.java - DRCI Internet

4.4.2 Confiabilidade

Há controle de transação com as operações em banco de dados utilizando JTA dentro do container EJB, esta tratativa condiz com boas práticas de desenvolvimento de aplicações. A existência do mesmo corrobora para a garantia das propriedades ACID do SGBD.

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.

4.4.3 Escalabilidade

A arquitetura baseada em remote EJB sem a manutenção de estado (Stateless) no componente denominado como back-end promove boa capacidade de escalonamento horizontal.

Esta arquitetura além de promover ambiente escalonável, favorece a utilização de ambientes redundantes com maior probabilidade de tolerância a falhas e maior capacidade de reuso.

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.

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 ambiente para desenvolvimento local para manutenções corretivas, adaptativas. Recomenda-se a criação do

mesmo juntamente com a documentação necessária para reproduzir para que estes ajustes possam ser realizados de forma mais assertivas e efetivas.

Recomenda-se também que este projeto seja organizado no repositório GIT "*novo*" (gitlab.mj.gov.b) tanto os componentes de front e back-end. Há certa dificuldade em encontrar os dois projetos referido GIT "*antigo*" (git.mj.gov.b).

6 Conclusão

A aplicação não apresenta boa estruturação em sua construção fato este que não propícia facilidade na manutenção corretiva/evolutiva. A arquitetura distribuída das dependências deste projeto somam os fatores que dificultam tanto a criação de ambiente local quanto a manutenção da aplicação.

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.