Ministério da Justiça

Projeto: SICAU

Nota Técnica

Revisão	Descrição	Autor	Data
1.0	Construção do documento	Israel Branco	08/05/2020

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/MJ-CGAE-SICAU-CGL-2017 na data de 08/05/2020.

Os módulos que compõe a solução alvo desta análise, foram desenvolvidos utilizando tecnologias que 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 Sicau 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 para trabalhar como uma aplicação WEB e utiliza o framework Struts como principal *framework* MVC da aplicação.

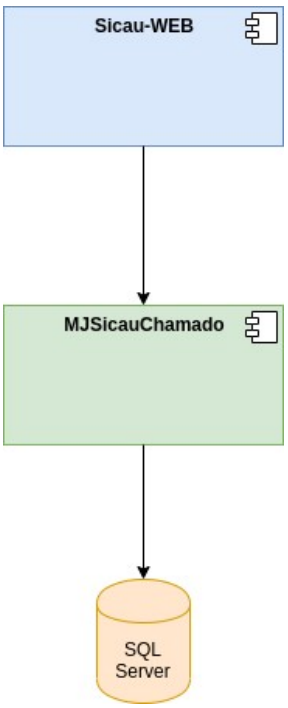

Figura 1: Representação em componentes

O repositório utilizado pelo projeto Sicau-Web não contempla o código do componente de negócio MJSicauChamado, para a realização das análises estáticas de código e demais que tangem a utilização de código fonte, fora utilizado engenharia reversa no componente baixado do repositório Artifactory.

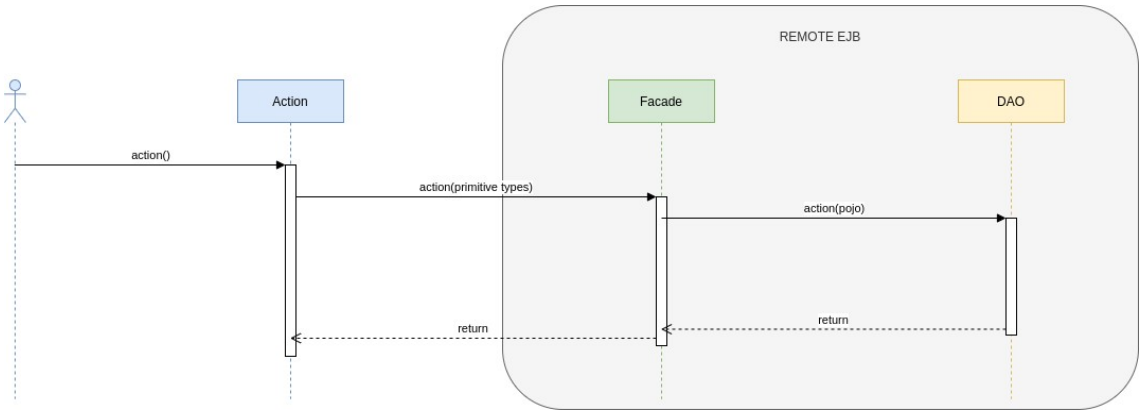

Figura 2: Fluxo principal - diagrama de sequências

3.1 Componente WEB

Este componente contem os controladores de formulários, páginas jsp, folhas de estilo e arquivos javascript. A estruturação deste projeto é intuitiva e com boa adequação dos padrões de projeto em conformidade com a nomenclatura dos pacotes.

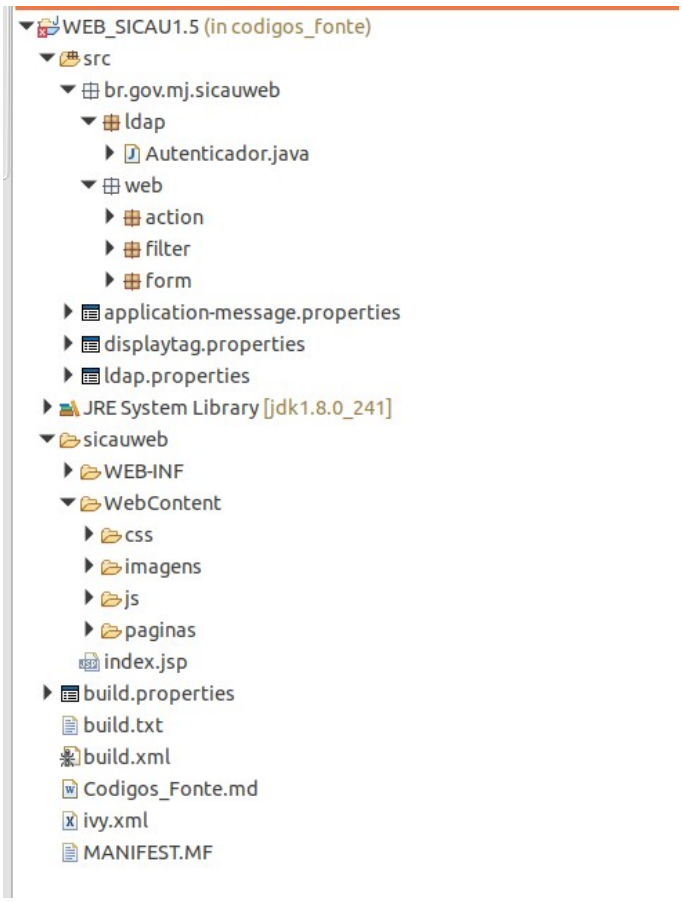

Figura 3: Estrutura do componente web

3.2 Componente MJSicauChamado

Este componente as classes de negócio, classes com mapeamento ORM, classes utilitárias, classes de serviço e classes de persistência. A estruturação deste projeto é intuitiva e com boa adequação dos padrões de projeto em conformidade com a nomenclatura dos pacotes.

Vale ressaltar que o código fonte analisado é oriundo de engenharia reversa, podendo haver divergências com o código fonte original do componente.

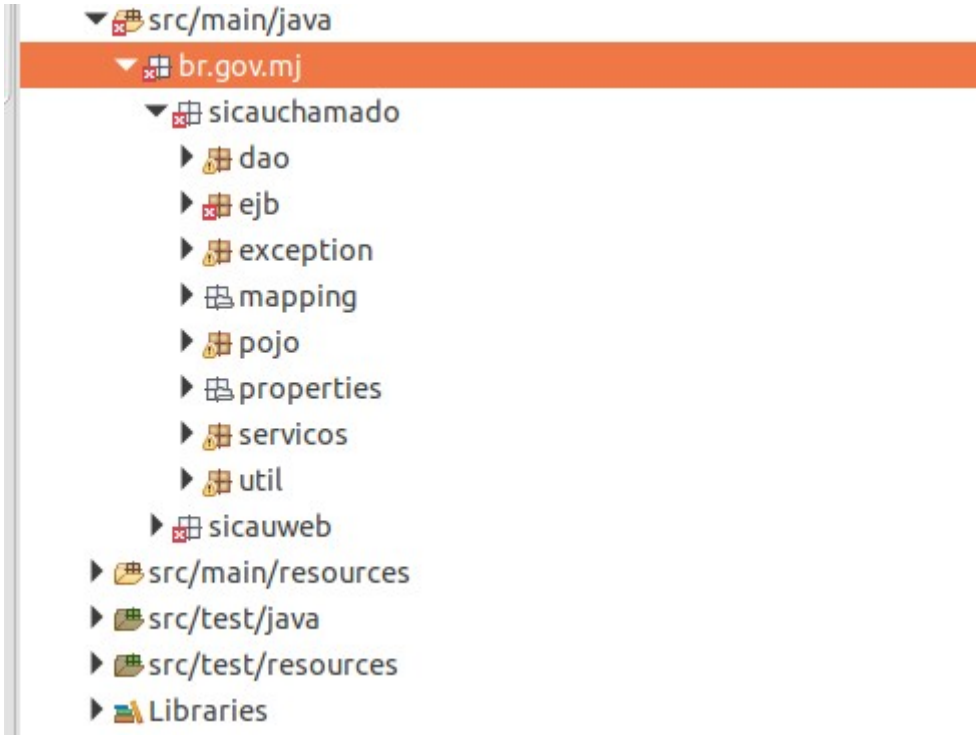

Figura 4: Estrutura do componente MJSicauChamado

3.3 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.	
EJB	2	Componente corporativo da arquitetura JEE	
Apache Struts	1.x	Framework MVC	
Hibernate	2.x	Framework ORM	
Jboss	4.x	Servidor de aplicação	

3.4 Modelagem de dados

A estrutura de banco de dados esta composta por 100 tabelas em um único schema.

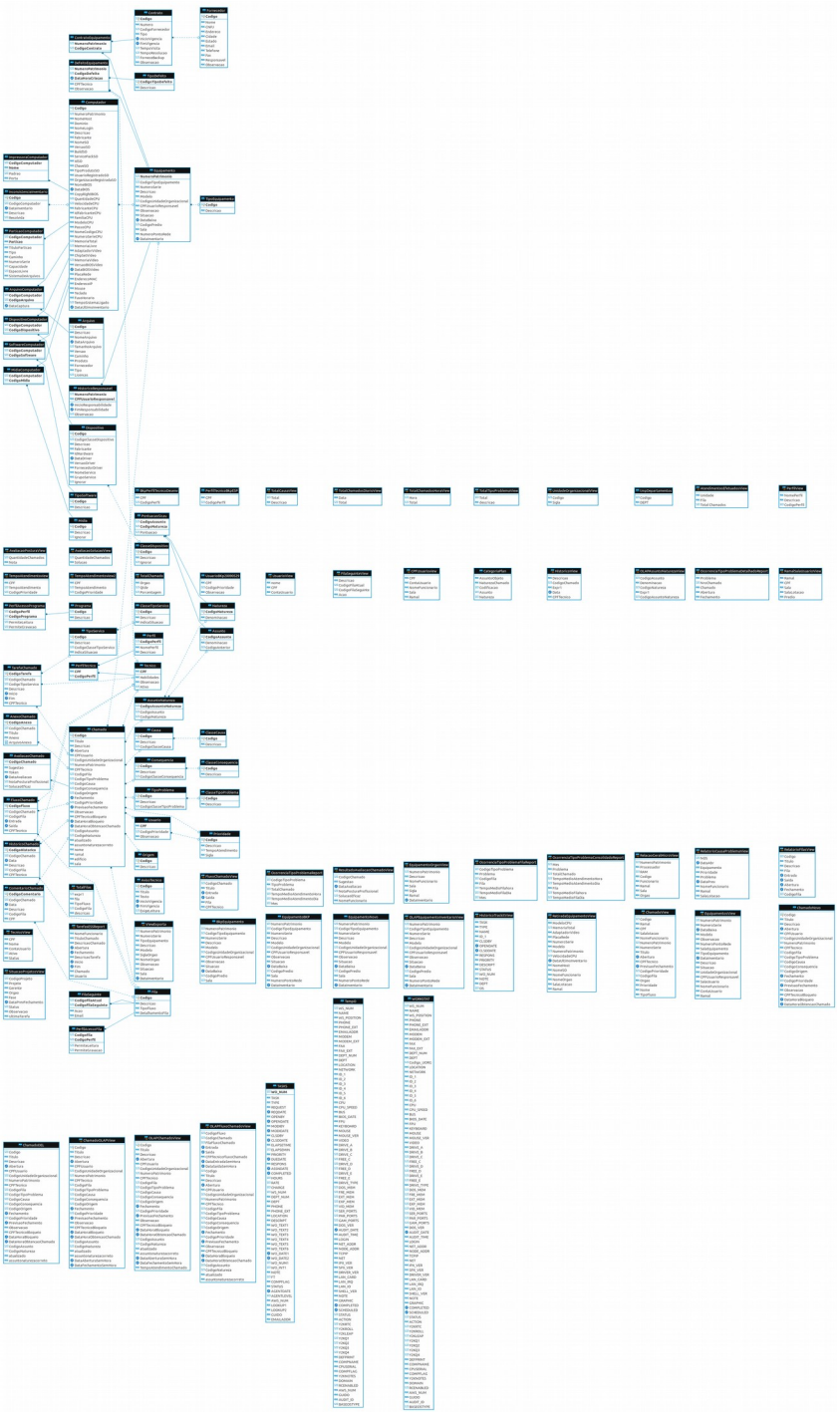

Figura 5: MER - banco de dados MJ schema Sicau

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 master):

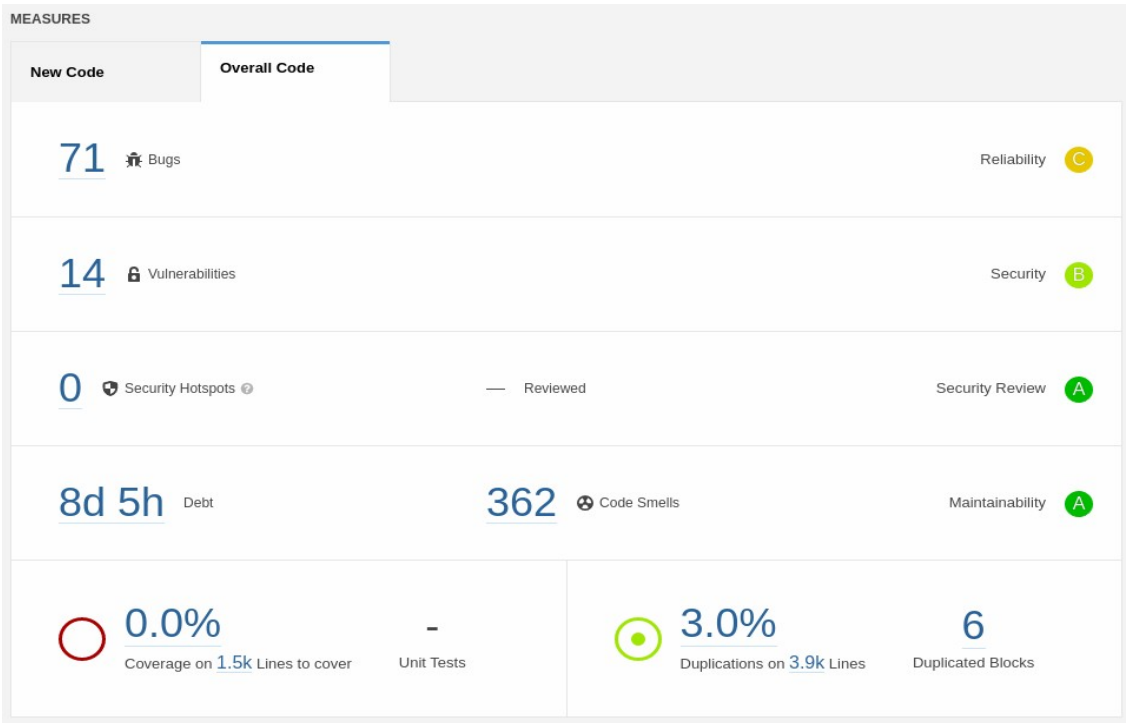

Figura 6: Análise estática de código

- 71 bugs;
- 14 vulnerabilidades violações de más práticas;
- 362 violações de código ruim (ex: complexidade cognitiva, complexidade ciclomática, debito técnico e outros)
- 3.0% 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	23
commons-fileupload-1.2.1.jar	CRITICAL	5	Highest	34
struts-core-1.3.8.jar	HIGH	20	Highest	27
struts-el-1.3.8.jar	HIGH	7	Highest	25
struts-tiles-1.3.8.jar	HIGH	7	Highest	29
commons-collections-3.1.jar	CRITICAL	3	Highest	25
dom4j-1.6.1.jar	CRITICAL	2	Highest	25

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

4.2 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.

4.2.1 Manutenibilidade de código

Os relatórios apresentados pela ferramenta SonarQube demonstram poucos vícios adotados durante o processo de construção do software, também demonstra a inexistência de cobertura de testes de unidade o que por sua vez dificulta o processo de refactoring da aplicação, uma vez que não há condições de mensurar impactos durante o processo de manutenção corretiva/adaptativa.

Como não há evidências da existência do código fonte do componente de negócio MJSicauChamado, há limitações para a manutenção corretiva/evolutiva da aplicação.

4.2.2 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.

4.2.3 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.

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

4.2.3 Escalabilidade

A arquitetura baseada em containers de Servelets e EJB em separado promoem uma boa capacidade de escalonamento na horizontal com a utilização de cluster para alta disponibilidade.

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

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.

Recomenda-se que seja executado análise de vulnerabilidade da ferramenta com ferramentas de análise de intrusão, como por exemplo OWASP ZAP. No momento da escrita desta nota técnica, não havia disponibilidade da aplicação nos ambientes de desenvolvimento, homologação e produção.

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.

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 juntamente disponibilização do código fonte do componente de negócio MJSicauChamado.

6 Conclusão

A aplicação apresenta uma boa estruturação em sua construção o que facilita sua manutenção corretiva/evolutiva, contudo a inexistência do código fonte citado inúmeras vezes neste documento impossibilitam a sua manutenção. A inatividade da comunidade/fabricante para os frameworks, bibliotecas e demais componentes de terceiros utilizados neste projeto dificultam a resolução de determinados problemas.

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.

Caso este projeto ainda demande muita utilização, recomenda-se a reconstrução do mesmo com a adoção de tecnologias mais modernas com foco na arquitetura proposta pela CTIS ao Ministério da Justiça.