



Ministério da Justiça

Projeto: SICAU-ADM

Nota Técnica

MJ	SICAU - Nota Técnica	
-----------	-----------------------------	--

Revisão	Descrição	Autor	Data
1.0	Construção do documento	Israel Branco	18/05/2020

1 Sumário

2 Introdução.....	4
3 Apresentação do cenário atual.....	5
3.1 Organização do projeto.....	7
3.2 Tecnologias utilizadas.....	8
3.3 Modelagem de dados.....	9
4 Análise técnica.....	11
4.1 SonarQube.....	11
4.2 Análise sobre os resultados.....	14
4.2.1 Manutenibilidade de código.....	14
4.2.2 Confiabilidade.....	15
4.2.3 Performance e estabilidade.....	16
4.2.3 Escalabilidade.....	17
4.2.4 UX - User experience.....	18
5 Conclusão e recomendações.....	19

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-CGMA-SICAU-ADMIN-2015> na data de 14/05/2020.

O módulo que compõe a solução alvo desta análise fora desenvolvido 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 irá descrever a arquitetura, tecnologias, frameworks e dependências que compõe a base da aplicação.

O Sicaú-ADM está construído para funcionar em ambiente WEB, utiliza tecnologia Java e está 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 SQL Server* e utiliza o framework Struts como *framework MVC*.

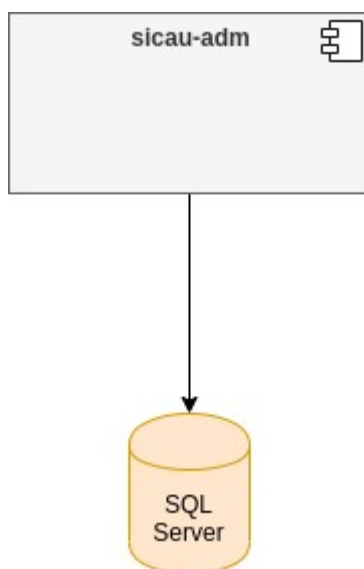


Figura 1: Diagrama de componentes

A aplicação segue padrões sequenciais conforme fluxo principal demonstrado na figura abaixo, sua arquitetura segue a utilização do padrão MVC estipulado pelo framework Struts e utiliza a biblioteca Egen JDBC para a camada de acesso a dados.

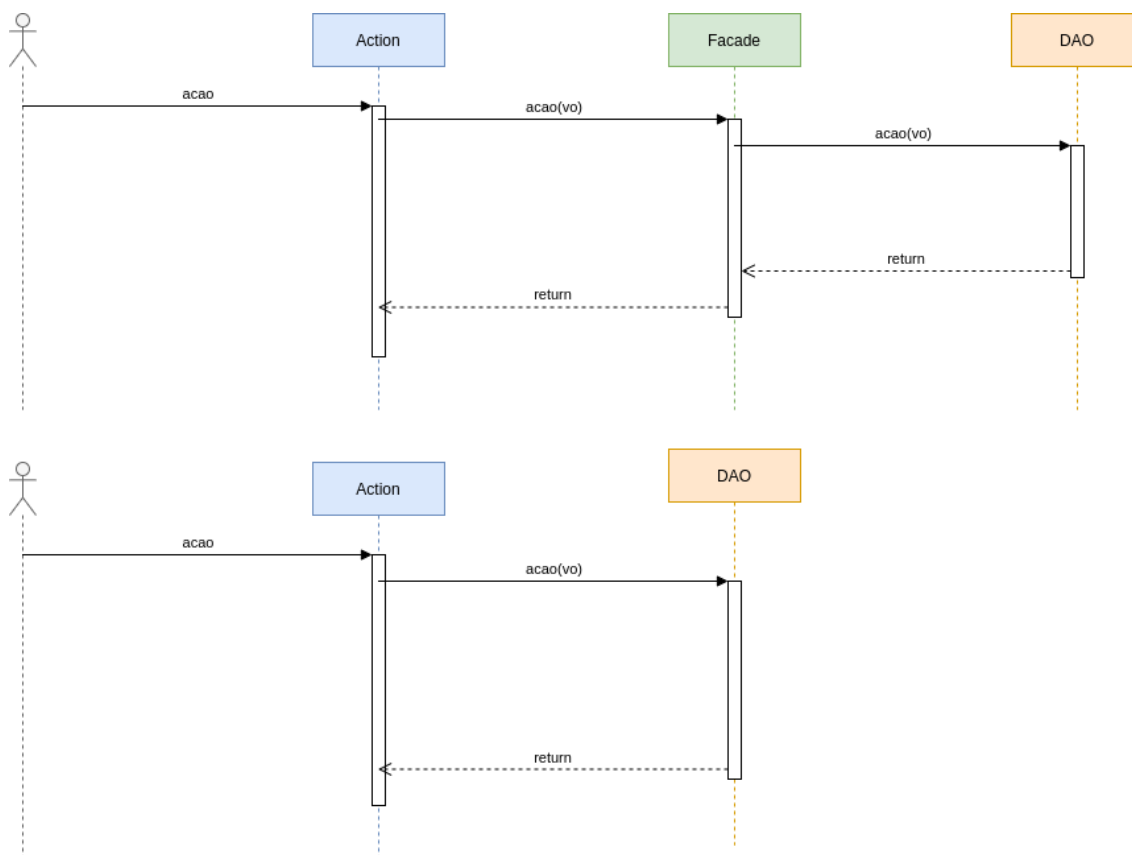


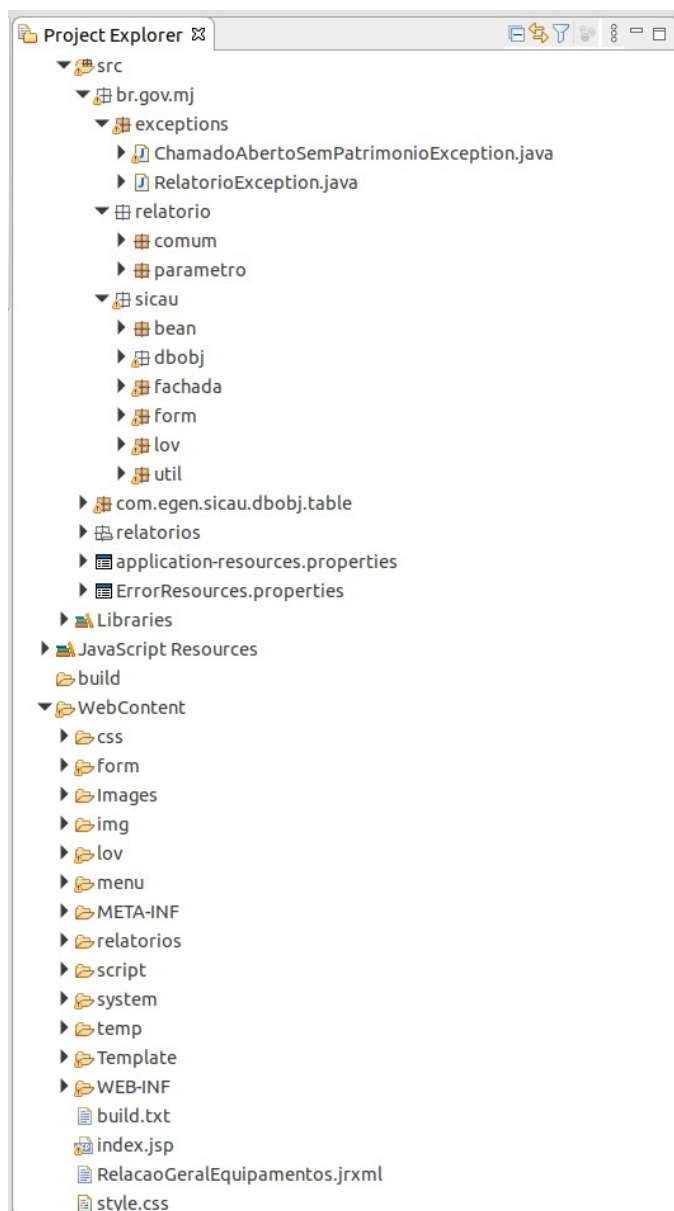
Figura 2: Diagrama de sequências - comportamento da solução

Conforme demonstrado na diagramação acima, a aplicação não apresenta comportamento uniforme em seu fluxo principal. Ambos comportamentos apresentados são encontrados na aplicação.



3.1 Organização do projeto

O projeto contém os controladores de formulários, páginas jsp, folhas de estilo e arquivos javascript, sua estruturação é intuitiva contudo não há boa adequação dos padrões de 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.	
egen-jdbc	egen-jdbc	Componente para construção da camada de persistência.	
Apache Struts	1.1	Framework MVC	
Jasper reports	1.1.0	Ferramenta para geração de relatórios.	
Jboss	4.0.3	Servidor de aplicação.	

3.3 Modelagem de dados

A aplicação Sicau-ADM utiliza dois schemas dentro da base MJ, sendo elas o schema Sicau (composto com 100 tabelas) e o Schema controle de acessos (composto por 6 tabelas).



Figura 3: MER - banco de dados MJ schema Sicau

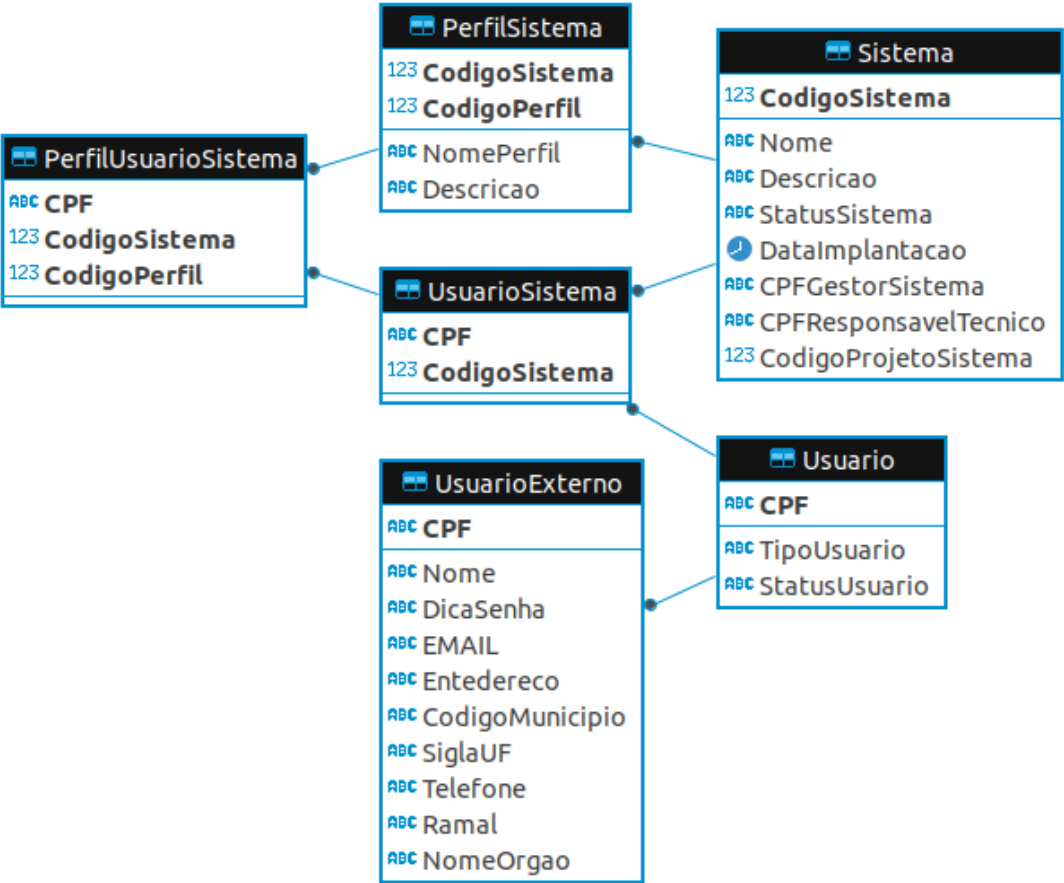


Figura 4: MER - banco de dados MJ schema ControleAcesso

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):

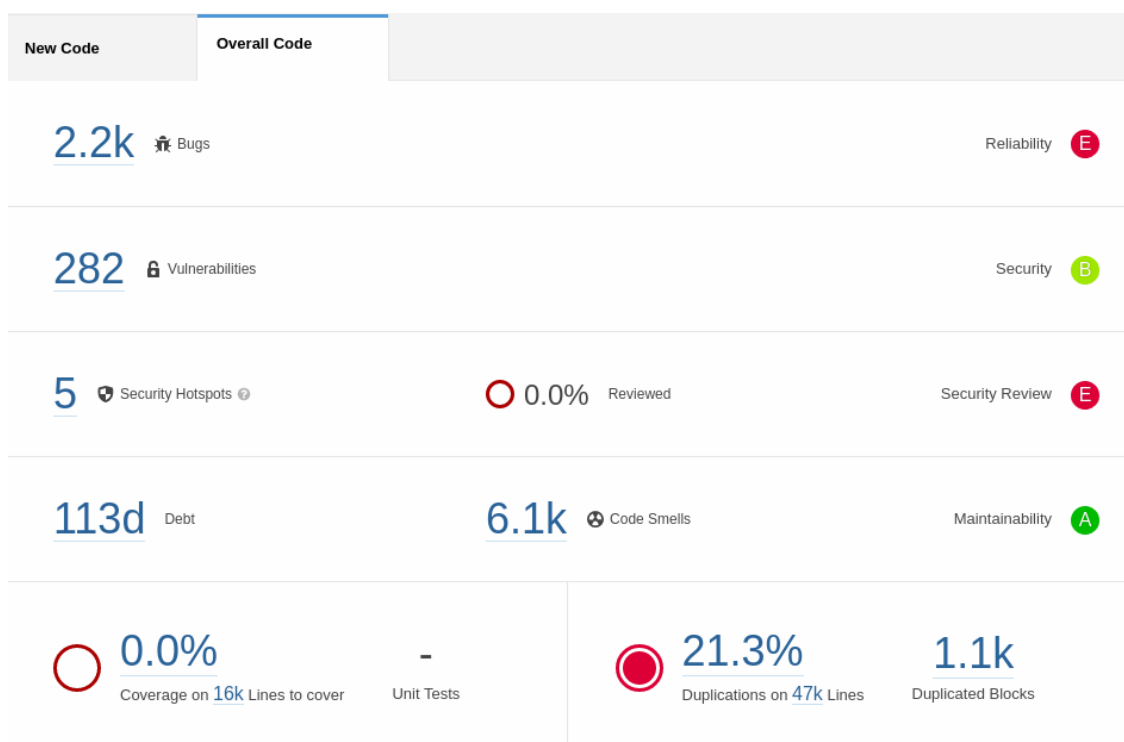


Figura 5: Sonarqube - Análise estática de código

- 2.2 mil bugs;
- 282 violações de vulnerabilidade de código;
- 6.1 mil vulnerabilidades violações de más práticas;
- 6.1 mil violações de código ruim (ex: complexidade cognitiva, complexidade ciclomática, debito técnico e outros)
- 21.3% 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, a seguir temos as principais informações extraídas desta análise.

Dependency	Highest Severity	CVE Count	Confidence	Evidence Count
standard-1.0.0.jar	HIGH	1	Highest	25
commons-beanutils-core-1.7.0.jar	HIGH	2	Highest	20
commons-collections-2.0.jar	CRITICAL	3	Highest	18
jasperreports-1.1.0.jar	HIGH	4		17
poi-2.5.1-final-20040804.jar	HIGH	8	Highest	14
commons-beanutils-1.7.0.jar	HIGH	2	Highest	20
struts-1.1.jar	HIGH	24	Highest	19
struts-legacy-1.1.jar	HIGH	11	Highest	18
commons-fileupload-1.0.jar	CRITICAL	5	Highest	24

A planilha acima apresenta as vulnerabilidades encontradas nas dependências do projeto, 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.

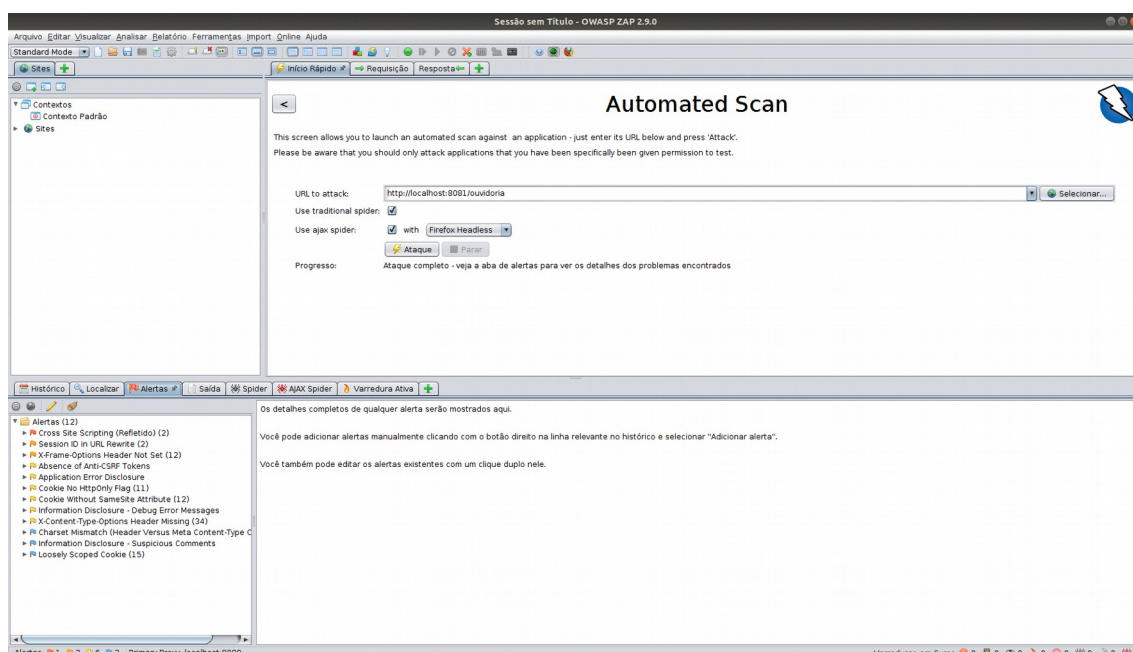


Figura 6: Relatório OWASP ZAP - análise de intrusão

- 1 vulnerabilidade de severidade alta;
- 2 vulnerabilidades de severidade média;
- 6 vulnerabilidades de baixa média;
- 3 vulnerabilidades a nível informativo;

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 inúmeros 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.

A falta de padronização de comportamento sequencial demonstrador na figura 2 exemplifica a falta de padrões comportamentais adotados durante o processo de construção do software.

A alta complexidade ciclomática está presente em várias partes do código, este modelo de codificação não só aumenta a dificuldade de refactoring de código como dificulta os testes funcionais do mesmo.



```

199 private void update_action(final AvisoTecnicoActionForm f, final JdbcUtil j, final HttpServletRequest request) throws
200     final HttpSession session = request.getSession(true);
201     final AvisoTecnico table = new AvisoTecnico();
202     final Vector sets = new Vector(10, 2);
203     table.setCodigo(FormatNumber.parseInt(f.getCodigo()));
204     table.setTitulo(f.getTitulo());
205     if (f.getTitulo() != null) {
206         sets.addElement("titulo");
207     }
208     else {
209         sets.addElement("");
210     }
211     table.setTexto(f.getTexto());
212     if (f.getTexto() != null) {
213         sets.addElement("texto");
214     }
215     else {
216         sets.addElement("");
217     }
218     table.setIniciovigencia(FormatDate.parseTimestamp(f.getIniciovigencia(), ""));
219     if (f.getIniciovigencia() != null) {
220         sets.addElement("iniciovigencia");
221     }
222     else {
223         sets.addElement("");
224     }
225     table.setFimvigencia(FormatDate.parseTimestamp(f.getFimvigencia(), ""));
226     if (f.getFimvigencia() != null) {
227         sets.addElement("fimvigencia");
228     }
229     else {
230         sets.addElement("");
231     }
232     table.setExigileitura(FormatNumber.parseInt(f.getExigileitura()));
233     if (f.getExigileitura() != null) {
234         sets.addElement("exigileitura");
235     }
236     else {
237         sets.addElement("");
238     }
239     final String[] set = new String[sets.size()];
240     for (int i = 0; i < sets.size(); ++i) {
241         set[i] = (String) sets.elementAt(i);

```

*Figura 7: Complexidade ciclomática exemplo -
AvisoTecnicoAction*

4.2.2 Confiabilidade

Não somente uma evidência de tratativa de controle transacional da classe FachadaChamada na aplicação, todas as demais classes que utilizam as classes JDBC Util não utiliza tão recurso e a falta desta boa prática não garante as propriedades ACID do SGBD.

4.2.3 Performance e estabilidade

Não foi analisado a aplicação em funcionamento para avaliar demais requisitos não funcionais, contudo foi percebido durante a execução da aplicação a inexistência de recursos de paginação de resultados. Certamente a falta desta boa prática degrada não somente os recursos de rede, banco de dados, servidor de aplicação e recursos locais dos usuários da aplicação.

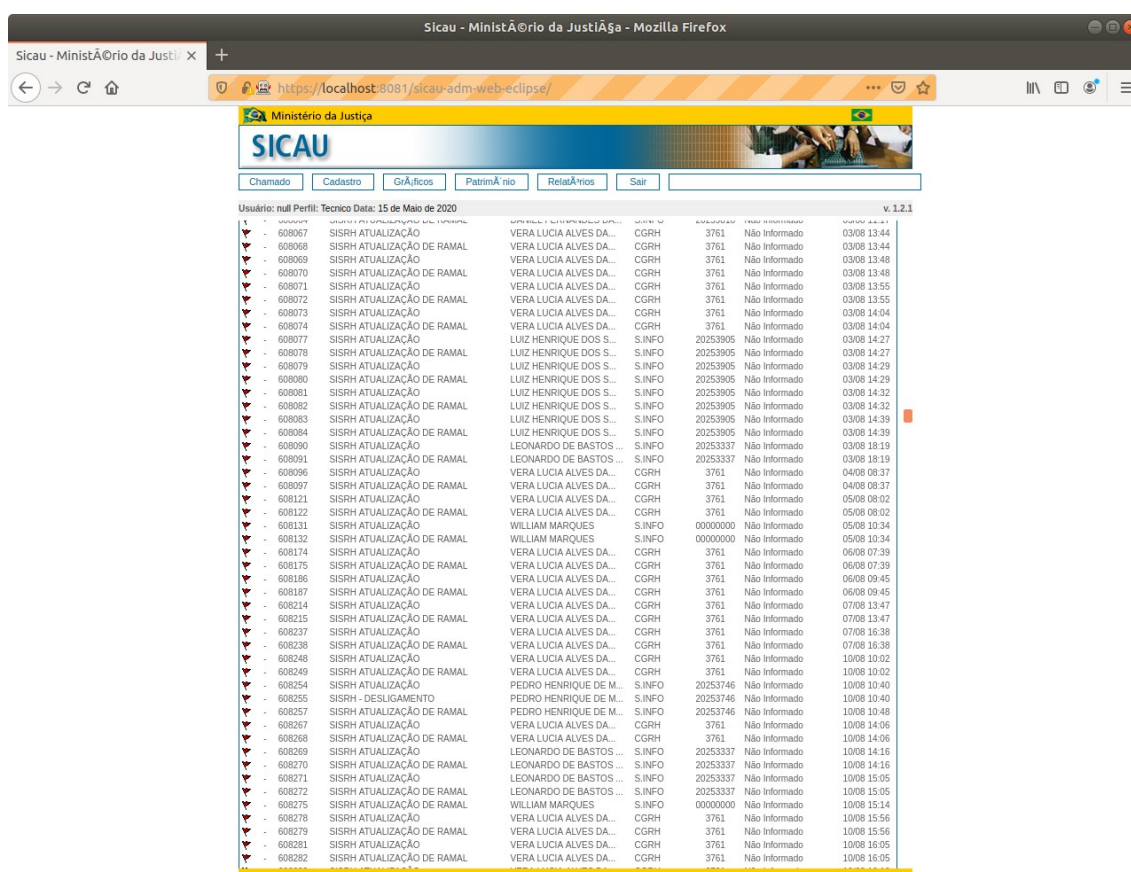


Figura 8: Performance - falta de utilização do recurso de paginação de resultados

A aplicação não apresenta recurso visual que indica ao usuário que sua requisição está em operação, isso faz com que o usuário não tenha a percepção de funcionamento da aplicação podendo induzir o

usuário a execução de uma nova requisição na aplicação.

4.2.3 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 levando 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.

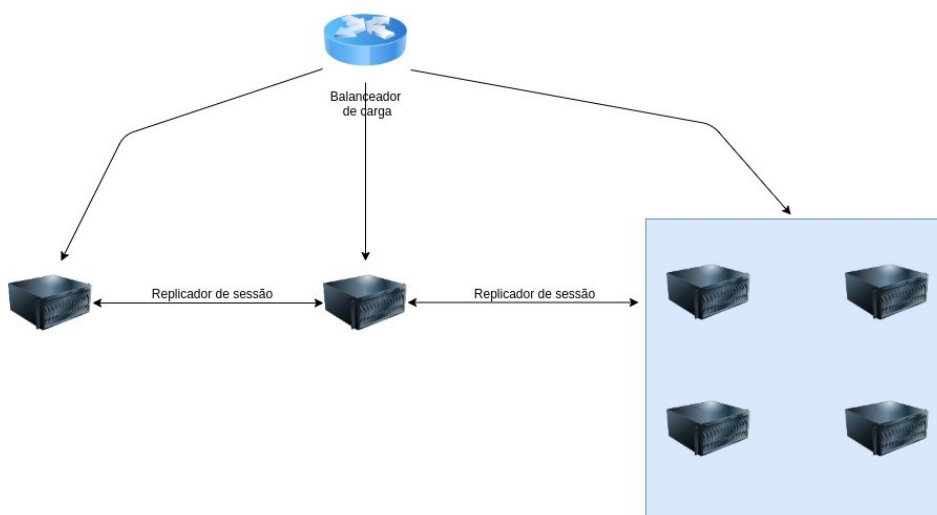


Figura 9: Escalabilidade do monólito

4.2.4 UX - User experience

Se tratando de um legado com certo tempo de construção, a aplicação não apresenta recursos modernos de navegação, intuitividade em sua execução e não apresenta boa apresentação gráfica tanto na aplicação web quanto na apresentação de relatórios.

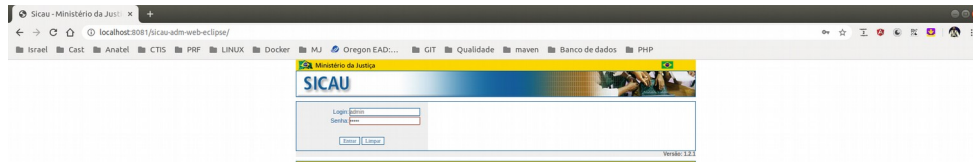
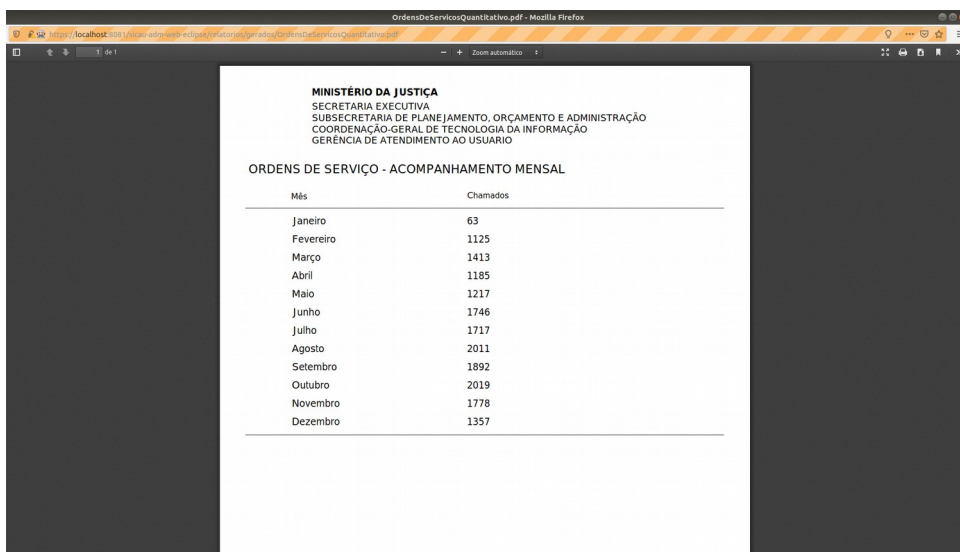


Figura 10: UX - apresentação da solução



MINISTÉRIO DA JUSTIÇA SECRETARIA EXECUTIVA SUBSECRETARIA DE PLANEJAMENTO, ORÇAMENTO E ADMINISTRAÇÃO COORDENAÇÃO-GERAL DE TECNOLOGIA DA INFORMAÇÃO GERÊNCIA DE ATENDIMENTO AO USUÁRIO	
ORDENS DE SERVIÇO - ACOMPANHAMENTO MENSAL	
Mês	Chamados
Janeiro	63
Fevereiro	1125
Março	1413
Abril	1185
Maio	1217
Junho	1746
Julho	1717
Agosto	2011
Setembro	1892
Outubro	2019
Novembro	1778
Dezembro	1357

Figura 11: Apresentação de relatórios



A apresentação dos gráficos da ferramenta não é usual e não facilita a visualização das legendas, não sendo esta apresentação útil para o usuário final.

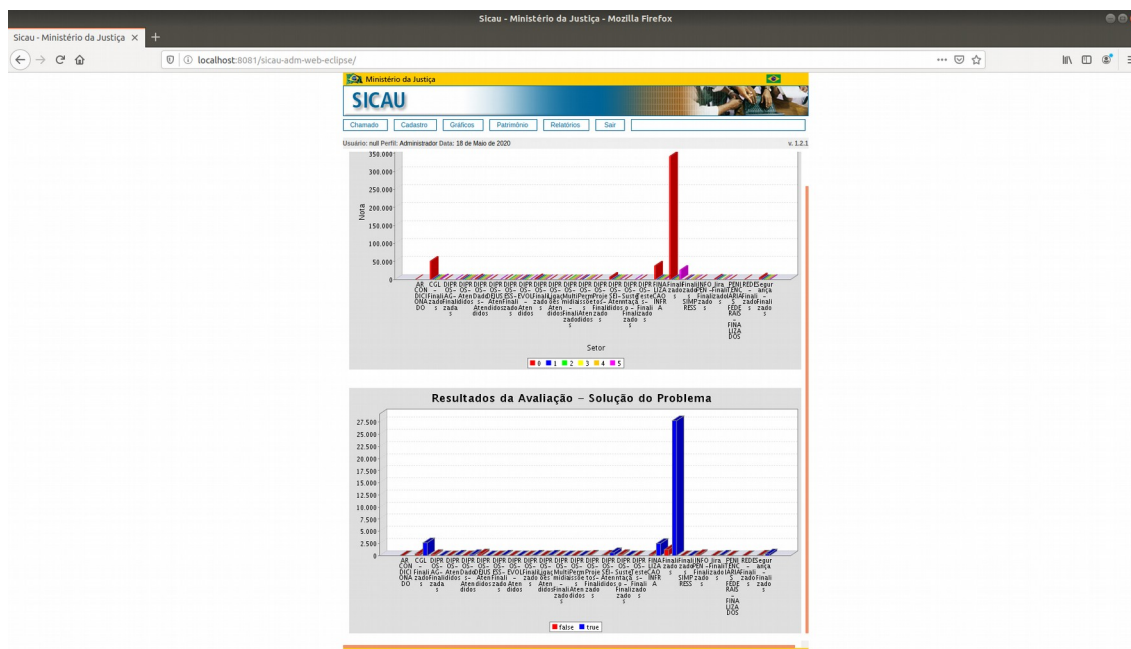


Figura 12: Apresentação dos gráficos - Sicau-ADM

5 Conclusão e recomendações

Pelos inúmeros erros apontados nas ferramentas de análise estática de código, gerenciamento de dependências e análise de intrusão aliado a falha na tratativa arquitetural juntamente com as tecnologias defasadas utilizadas neste projeto, não recomenda-se a manutenção desta ferramenta no parque tecnológico do Ministério da Justiça.

Recomenda-se a reconstrução da mesma segundo a arquitetura de referência proposta pela CTIS ao Ministério da Justiça.