




Departamento de Polícia Rodoviária Federal

**Projeto:** Boletim de ocorrências policiais - BOP2

# Nota Técnica

|             |                            |  |
|-------------|----------------------------|--|
| <b>DPRF</b> | <b>BOP2 - Nota técnica</b> |  |
|-------------|----------------------------|--|

| <b>Revisão</b> | <b>Descrição</b>        | <b>Autor</b>  | <b>Data</b> |
|----------------|-------------------------|---------------|-------------|
| 1.0            | Construção do documento | Israel Branco | 30/07/2020  |
|                |                         |               |             |
|                |                         |               |             |
|                |                         |               |             |
|                |                         |               |             |
|                |                         |               |             |
|                |                         |               |             |

# 1 Sumário

|                                      |    |
|--------------------------------------|----|
| 2 Considerações iniciais.....        | 4  |
| 3 Apresentação do cenário atual..... | 5  |
| 3.1 Tecnologias utilizadas.....      | 8  |
| 4 Análise técnica.....               | 9  |
| 4.1 SonarQube.....                   | 9  |
| 4.2 OWASP Dependency Check.....      | 12 |
| 4.3 OWASP ZAP.....                   | 13 |
| 4.4 Estrutura do projeto.....        | 14 |
| 4.5 Manutenibilidade de código.....  | 16 |
| 4.6 Confiabilidade.....              | 19 |
| 4.7 Performance e estabilidade.....  | 22 |
| 5 Recomendações.....                 | 23 |

## 2 Considerações iniciais

Este documento visa reportar o resultado da análise efetuada na aplicação **BOP2** denominada neste documento como **BOP**. Para este estudo foram desconsiderados todo o contexto comercial ao qual a ferramenta está inserida, também foram desconsideradas o ambiente ao qual a ferramenta está 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, gerou-se a tag *tag-sonda-nota-tecnica-20200629* no repositório <https://git.prf/prf/bop2/> com referência branch Marcos-v-2.3.3-c na data de 29/07/2020.



### 3 Apresentação do cenário atual

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

O sistema BOB foi 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) e utiliza o banco de dados *PostgreSQL*.

Sua arquitetura é composta por 2 componentes, sendo eles **BOP-Entidades** que contém o mapeamento ORM para a o schema dbbop e o componente **BOP-Web** que contém classes utilitárias, classes de acesso a dados, conversores, classes de serviço, e componentes visuais da aplicação tais como controladores de formulários, folhas de estilo, páginas web e demais artefatos presentes na camada de apresentação.

O diagrama a seguir representa o modelo de componentes ao qual a aplicação está construída.

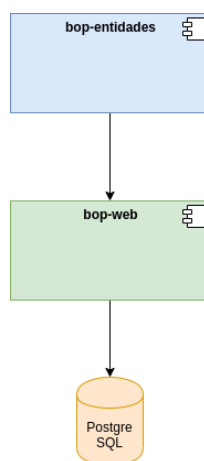
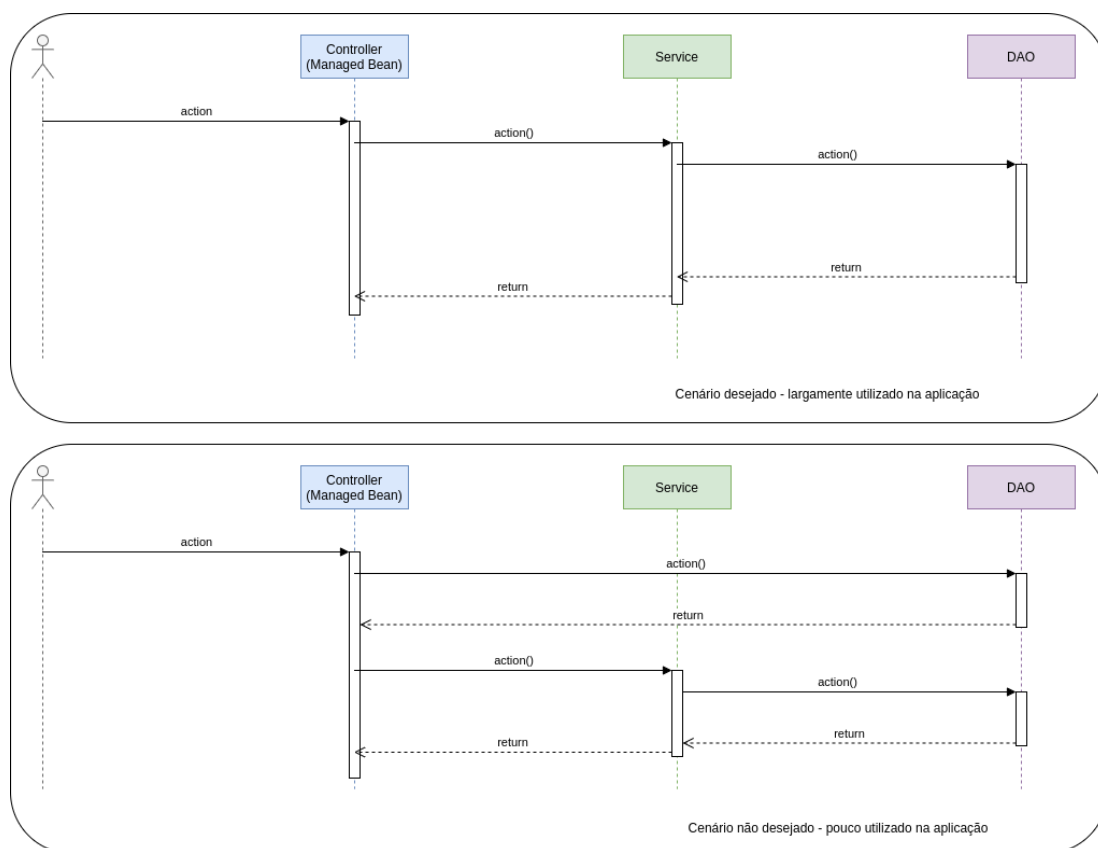


Figura 1: Representação dos componentes



A aplicação utiliza o modelo MVC para a segregação de responsabilidades em camadas sendo que as requisições http são oriundas das páginas JSF . O diagrama a seguir representa os fluxos encontrados durante o processo de análise da aplicação, o diagrama representa também a falta de padronização comportamental da mesma.



*Figura 2: Representação sequencial do fluxo principal*



A solução utiliza o schema dbbop como banco de dados core do sistema, há falta de padronização na concepção dos nomes das 64 tabelas que compõe o schema, somente algumas tabelas possuem o prefixo “tb”.

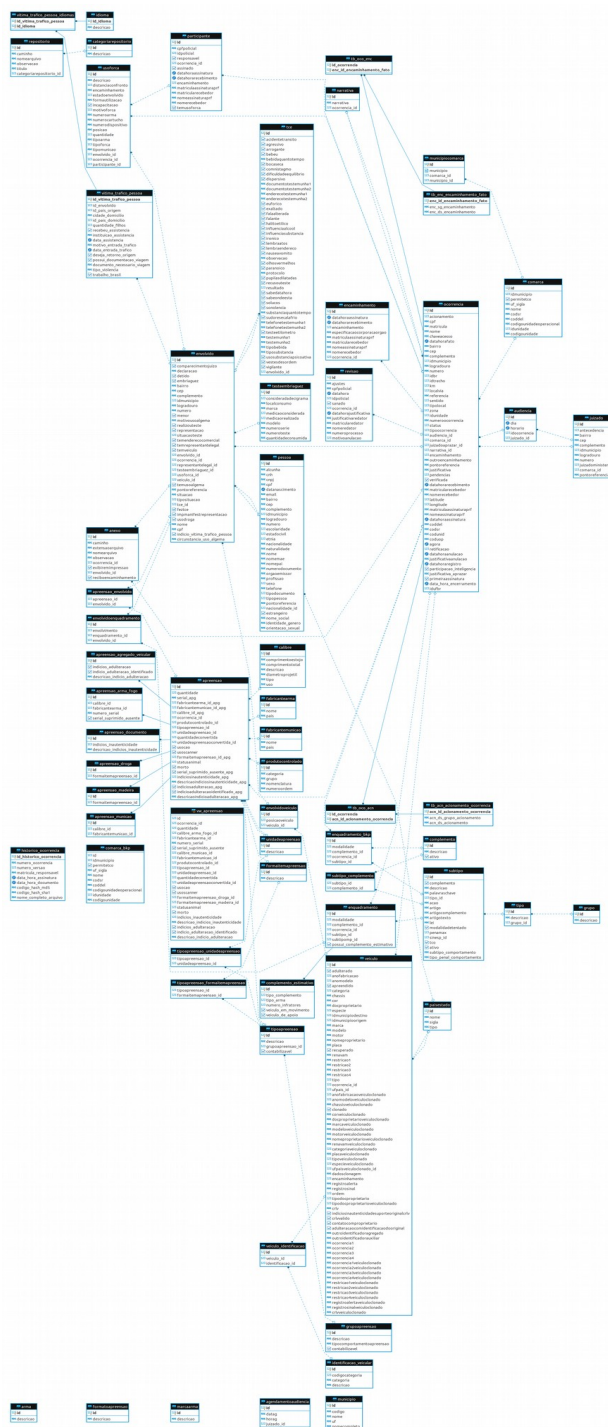


Figura 3: MER - Schema dbbop

### 3.1 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       | 1.8    | Linguagem de programação.                                   |                                  |
| Hibernate  | 5.1.3  | Framework ORM.  |                                  |
| Primefaces | 6.2.2  | Extensão de componentes JSF                                 |                                  |
| jUnit      | 4.12   | Framework para criação de testes unitários                  |                                  |
| Wildfly    | 10.x   | Servidor de aplicação JEE.                                  | Utiliza container CDI e Servlet. |
| Apache POI | 3.17   | Biblioteca para manipulação de documentos padrão Ms Office. |                                  |
| PostgreSQL |        | Banco de dados relacional                                   |                                  |

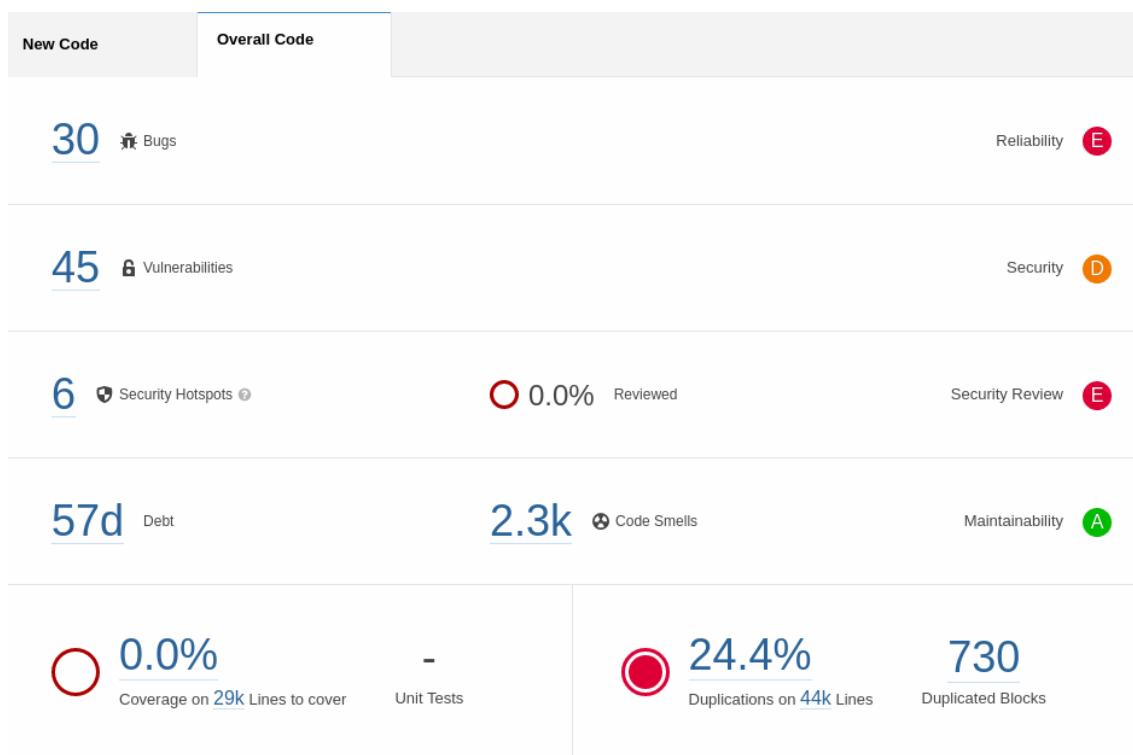


## 4 Análise técnica

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 da DPRF, contudo foram utilizadas as regras padrões de análise da ferramenta.



*Figura 4: Análise estática de código*

- 30 bugs;
- 45 vulnerabilidades de código;
- 6 violações de segurança;
- 2.3 mil violações de código ruim (complexidade cognitiva , complexidade ciclomática e débito técnico);
- 24,4% de duplicidade de código

## 4.2 OWASP Dependency Check

A utilização de bibliotecas de terceiros aumenta substancialmente a produtividade durante o processo de 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 na construção deste projeto, a seguir temos as principais informações extraídas desta análise, a relação completa desta análise está disponível no Anexo I deste documento.

| Dependency   | Highest Severity | CVE Count | Confidence | Evidence Count |
|--|------------------|-----------|------------|----------------|
| <a href="#">ant-1.7.1.jar</a>                                    | LOW              | 1         | Highest    | 25             |
| <a href="#">bcprov-jdk14-1.38.jar</a>                            | 0.0              | 15        | Highest    | 27             |
| <a href="#">bcprov-jdk15-135.jar</a>                             | HIGH             | 14        | Highest    | 23             |
| <a href="#">cdi-api-1.2.jar</a>                                  | MEDIUM           | 1         | Low        | 36             |
| <a href="#">commons-beanutils-1.8.3.jar</a>                      | HIGH             | 2         | Highest    | 35             |
| <a href="#">commons-collections-3.1.jar</a>                      | CRITICAL         | 3         | Highest    | 25             |
| <a href="#">commons-collections-3.2.1.jar</a>                    | CRITICAL         | 3         | Highest    | 35             |
| <a href="#">commons-email-1.3.1.jar</a>                          | HIGH             | 2         | Highest    | 38             |
| <a href="#">commons-fileupload-1.2.1.jar</a>                     | CRITICAL         | 5         | Highest    | 33             |
| <a href="#">dom4j-1.6.1-jboss.jar</a>                            | HIGH             | 2         | Highest    | 25             |
| <a href="#">hibernate-validator-5.1.0.Final.jar</a>              | MEDIUM           | 1         | Highest    | 33             |
| <a href="#">httpclient-4.2.1.jar</a>                             | MEDIUM           | 2         | Highest    | 34             |
| <a href="#">httpclient-4.2.6.jar</a>                             | MEDIUM           | 2         | Highest    | 34             |
| <a href="#">itextpdf-5.0.6.jar</a>                               | HIGH             | 1         | High       | 22             |
| <a href="#">javax.faces-api-2.2.jar</a>                          | MEDIUM           | 1         | Highest    | 43             |
| <a href="#">jquery.js</a>  | medium           | 3         |            | 3              |
| <a href="#">org.mortbay.jetty-5.1.12.jar</a>                     | MEDIUM           | 6         | Highest    | 25             |
| <a href="#">poi-3.17.jar</a>                                     | LOW              | 1         | Highest    | 29             |
| <a href="#">primefaces-6.2.jar</a>                               | MEDIUM           | 1         | Highest    | 28             |
| <a href="#">primefaces-6.2.jar: jquery.js</a>                    | MEDIUM           | 2         |            | 3              |
| <a href="#">selenium-core-1.0.1.jar: dojo.js</a>                 | HIGH             | 6         |            | 3              |
| <a href="#">selenium-core-1.0.1.jar: dojo.js.uncompressed.js</a> | HIGH             | 6         |            | 3              |
| <a href="#">selenium-core-1.0.1.jar: prototype.js</a>            | HIGH             | 1         |            | 3              |
| <a href="#">shiro-core-1.3.2.jar</a>                             | HIGH             | 3         | Highest    | 32             |



### 4.3 OWASP ZAP

Ferramenta OWASP ZAP 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.

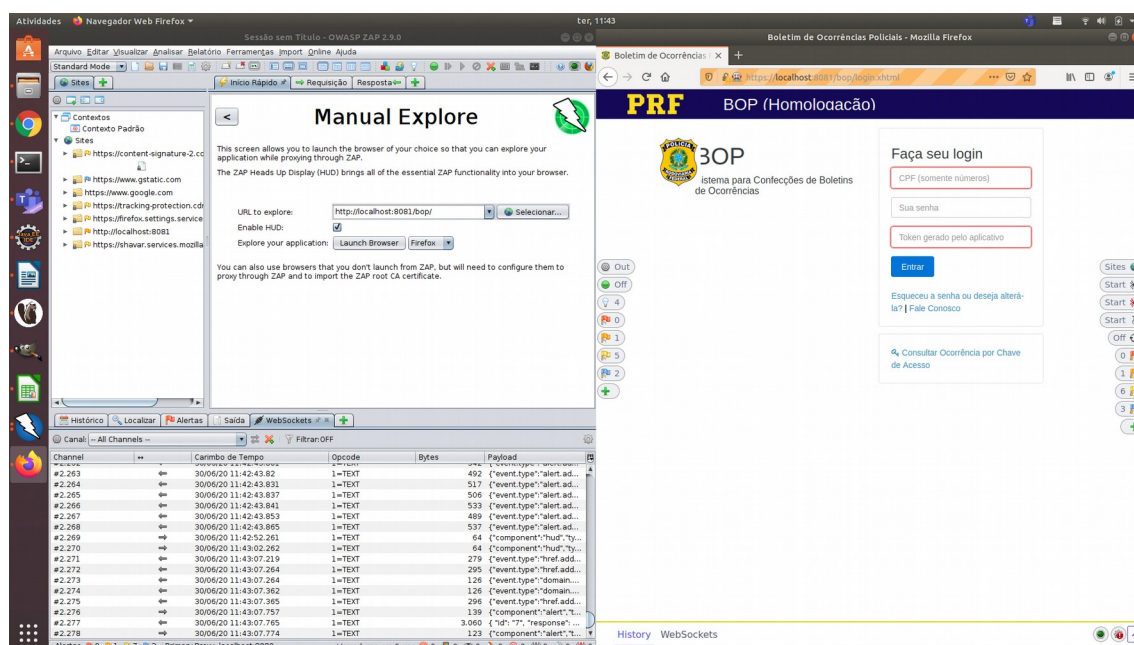


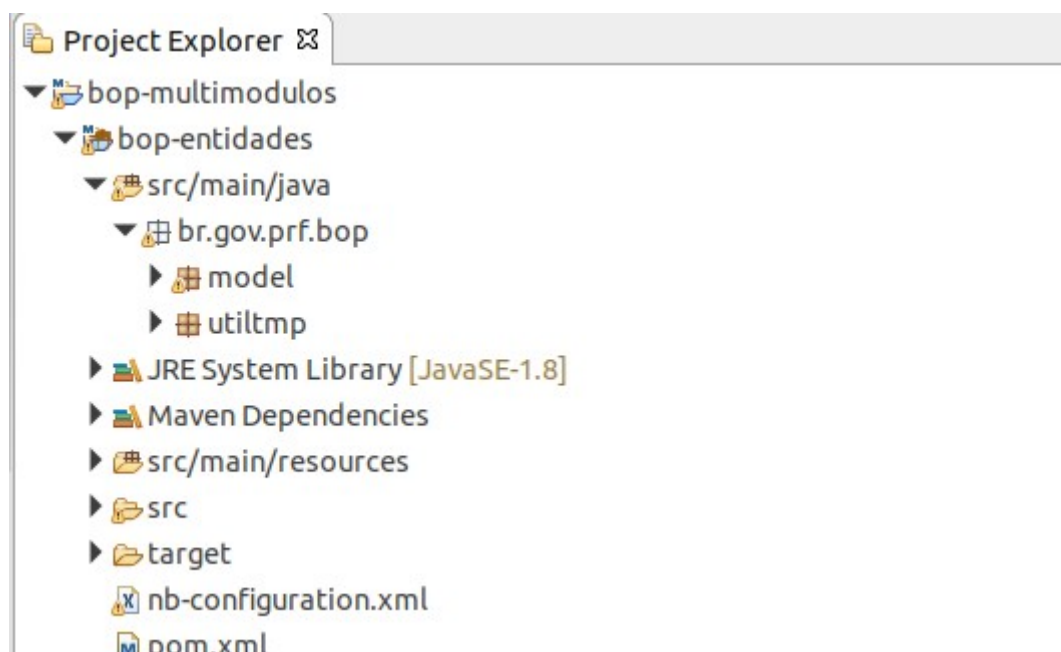
Figura 5: OWASP ZAP - Análise de intrusão de software

- 2 vulnerabilidade de severidade alta;
- 2 vulnerabilidade de severidade média;
- 13 vulnerabilidades de baixa média;
- 12 vulnerabilidades a nível informativo;

O relatório completo dos testes aplicados estão disponíveis no anexo I deste documento.

#### 4.4 Estrutura do projeto

Os componentes que envolvem o escopo da ferramenta possui boa organização sendo que há segregação por contexto funcional na estrutura dos dois componentes do projeto.



*Figura 6: Estrutura do componente BOP-Entidades*



*Figura 7: Estrutura do componente BOP-Web*



## 4.5 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 testes de unidade trazem consigo aumento significativo na dificuldade no 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 na utilização da segregação de responsabilidade por camadas fere o padrão comportamental da solução.

A seguir teremos exemplos encontrados durante a análise do código que ferem boas práticas, tais como padrão de segregação por responsabilidade de camadas, coesão comercial/estrutural, falta de composição e utilização de controle transacional.

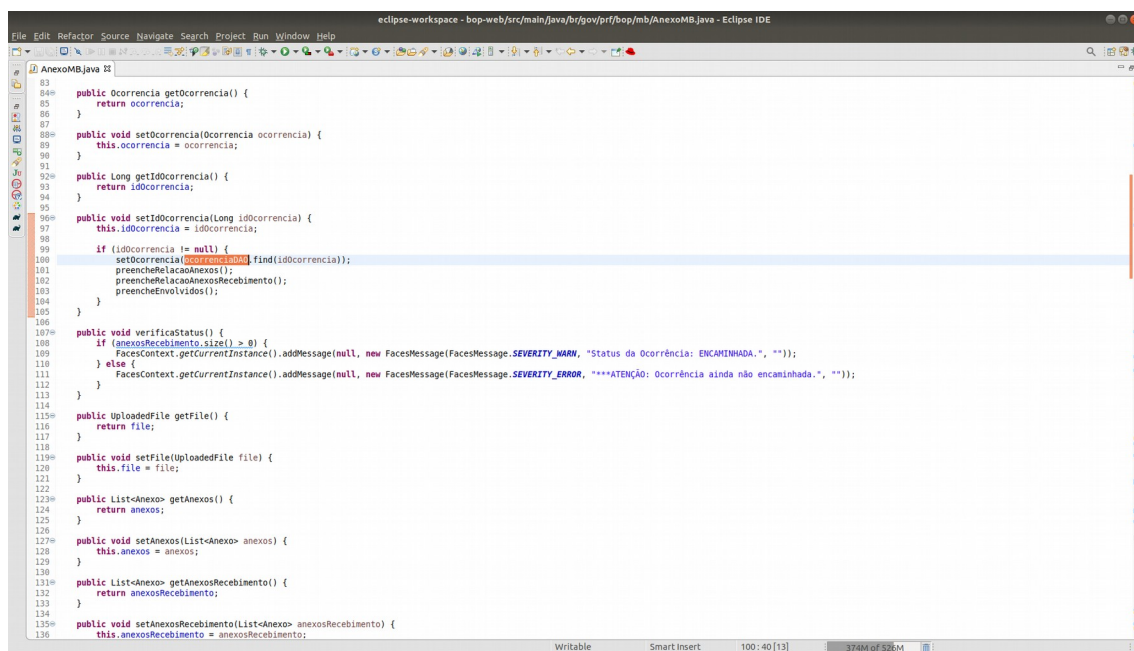


Figura 8: Managed Bean acessando camada de dados



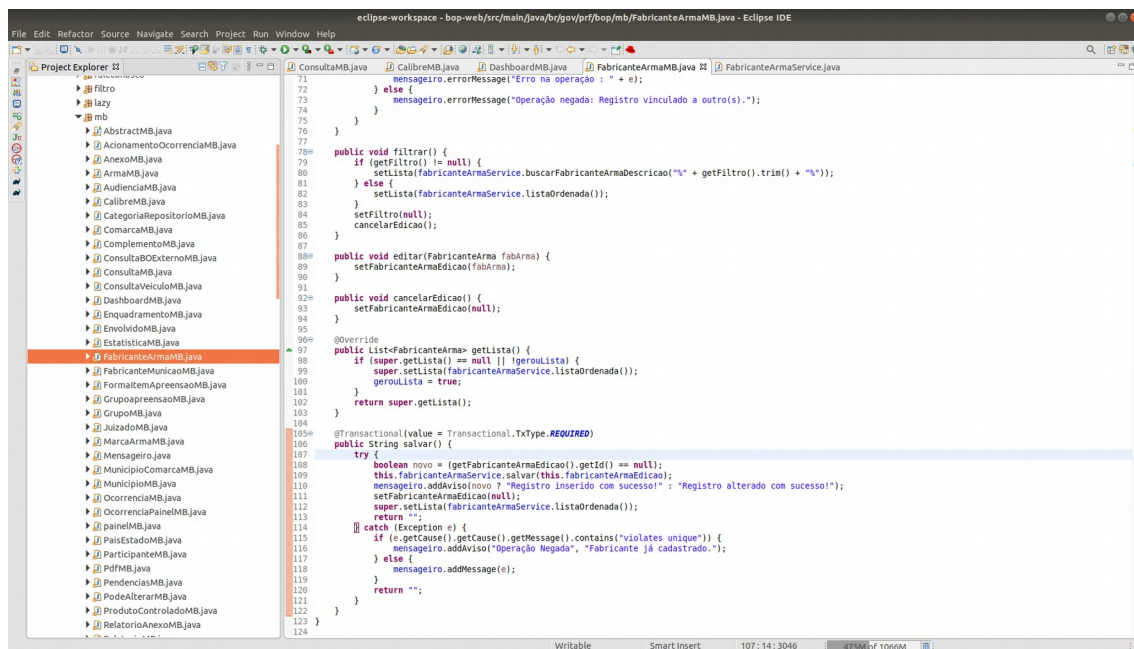


Figura 9: Managed Bean efetuando controle transacional

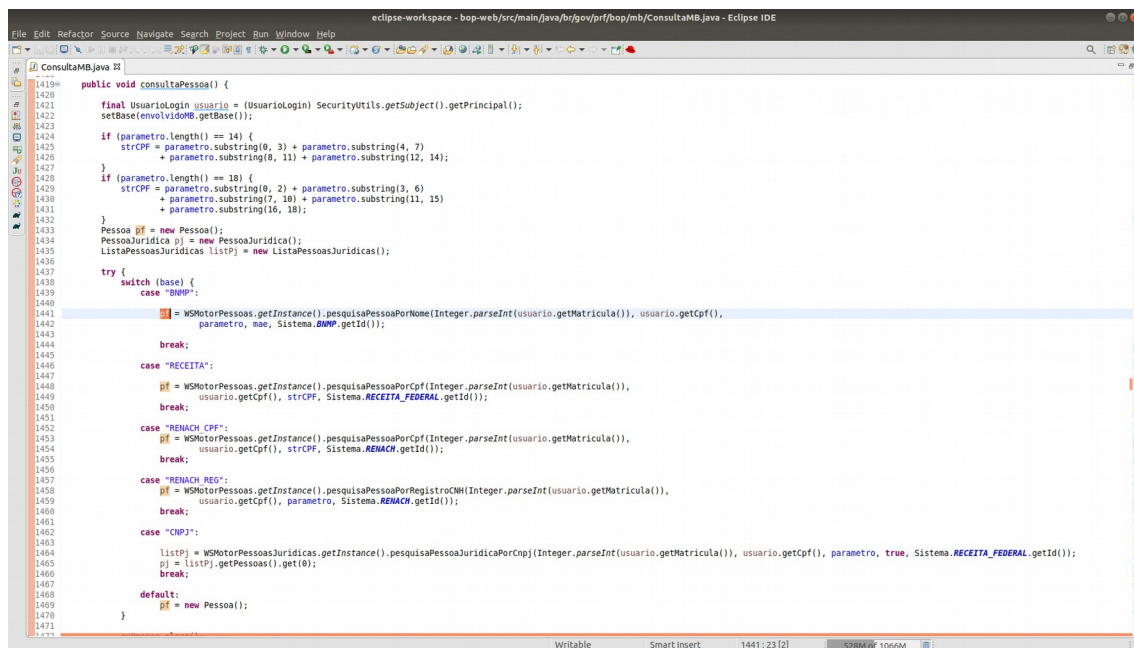
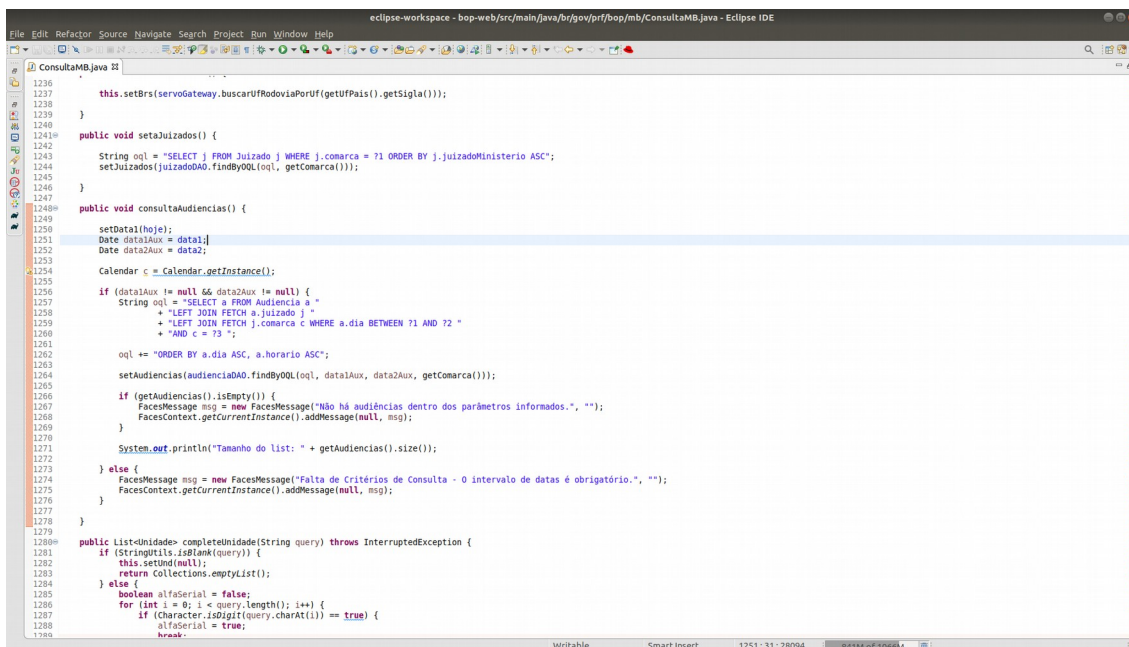


Figura 10: Managed Bean realizando regra negocial



*Figura 11: Managed Bean escrevendo SQL e acessando camada de dados*

A tratativa inadequada dos logs da aplicação certamente não auxiliam o administrador/desenvolvedor quanto a descoberta de falhas no sistema. A ilustração abaixo demonstra 124 ocorrências de utilização do comando `System.out.print*` nos componentes da aplicação.



*Figura 12: 124 Ocorrências de System.out.print\**



#### 4.6 Confiabilidade

O controle de transação efetuado na aplicação está sendo feito em operações controladas a nível de aplicação na camada superior a camada de acesso a dados (DAO – Data Access Object). Esta prática é a recomendada para que haja garantia das propriedades ACID do banco de dados, contudo, também encontramos este controle transacional sendo efetuado nas camadas de que fazem bind com os formulários JSF (Managed Bean) conforme demonstrado na Figura 9.

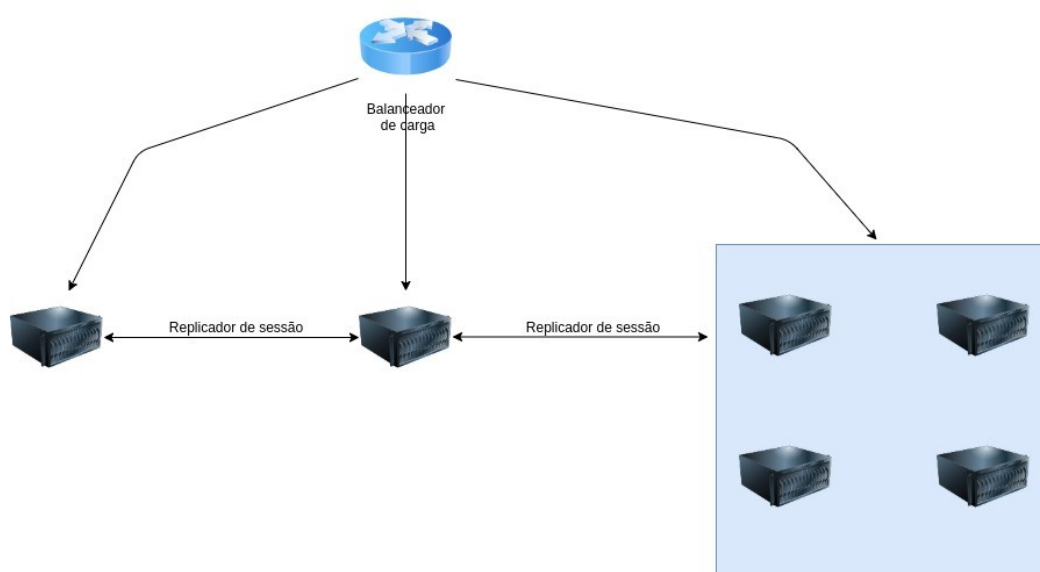
A manutenção da consistência de dados é algo fortemente desejado, contudo esta não garante toda a confiabilidade da solução. A quantidade elevada de bugs, vulnerabilidades no código e nas bibliotecas de terceiros encontradas nos relatórios apresentados trazem riscos a confiabilidade da ferramenta.

#### 4.7 Performance e estabilidade

Não foi analisado o funcionamento da aplicação para avaliar demais requisitos não funcionais, recomenda-se a utilização de ferramentas de APM para mensurar performance e recursos de máquina utilizados.

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 e onerosos.

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 13: Escalabilidade do monólito*

## 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 associação dos relatórios de análise de dependências com os relatórios de análise de intrusão para que sejam analisados 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 a implantação de ferramentas de APM para que sejam criadas métricas e alarmes que auxiliem na continuidade do serviço em ambiente produtivo(monitoramento de processamento e memória por exemplo), tendo em vista que este tipo de ferramenta fornece mecanismos para determinarmos o comportamento da solução (auxiliam no refactoring de código) e também subsidia para o correto dimensionamento da infraestrutura.

Recomenda-se o desacoplamento das execuções em lote do aplicação, uma vez que esta prática promove a concorrência de recursos da aplicação principal e dificulta a escalabilidade horizontal.

Recomenda-se também o ajuste dos fluxos de execução da

|             |                            |  |
|-------------|----------------------------|--|
| <b>DPRF</b> | <b>BOP2 - Nota técnica</b> |  |
|-------------|----------------------------|--|

aplicação, uma vez que não um comportamento uniforme, temos furos arquiteturais e a duplicidade de responsabilidades entre os componentes da aplicação. Esta prática não promove o princípio da abstração da orientação a objetos aplicado ao modelo MVC.

Recomenda-se também a criação do ambiente DPRF Segurança em ambiente de desenvolvimento necessário para a efetivação do processo de autenticação. A aplicação BOP em ambiente de homologação e produção aponta para o ambiente de produção do sistema DPRF Segurança, o ambiente de testes da aplicação aponta para o ambiente de Homologação do DPRF Segurança. Da forma que o ambiente está sendo configurado dificulta as manutenções corretivas/evolutivas a serem realizadas pela equipe técnica terceirizada.

Para fins de organização, padronização e documentação, recomenda-se que seja mantido de forma permanente 3 branches no repositório GIT e que estas representem especificamente os ambientes ao qual estão implantadas, sendo elas master, homologação e desenvolvimento juntamente com documento readme com as diretrizes necessárias para execução da aplicação. A master branch atualmente não contempla o código fonte da versão em produção sendo esta, desatualizada e obsoleta.