Departamento de Polícia Rodoviária Federal

**Projeto:** Boletim de acidente de trânsito - BAT

# Nota Técnica

| Revisão | Descrição | Autor | Data |
|---|---|---|---|
| 1.0 | Construção do documento | Israel Branco | 02/06/2020 |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# 1 Sumário

# 2 Considerações iniciais

Este documento visa reportar o resultado da análise efetuada na aplicação **Novo BAT** denominada neste documento como **BAT**. 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, gerou-se a tag *ctis-nota-tecnica-20200601* no repositório https://git.prf/acidente/bat com referência branch master na data de 01/06/2020.

# 3 Apresentação do cenário atual

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

O sistema BAT 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), utiliza o banco de dados *PostgreSQL* e banco de dados *MySQL* para integração com a plataforma SEI.

Sua arquitetura é composta por quatro componentes, sendo eles **DBServo** que contém o mapeamento e camada de acesso a dados para a o schema servo, **bat-nucleo** que contém o mapeamento para o banco de dados para o schema principal da aplicação dbbat, **bat-servico** que contém classes utilitárias, classes de acesso a dados, conversores, classes de agendamento e classes de serviço, por fim temos o componente **bat-web** responsável por tratar os 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.

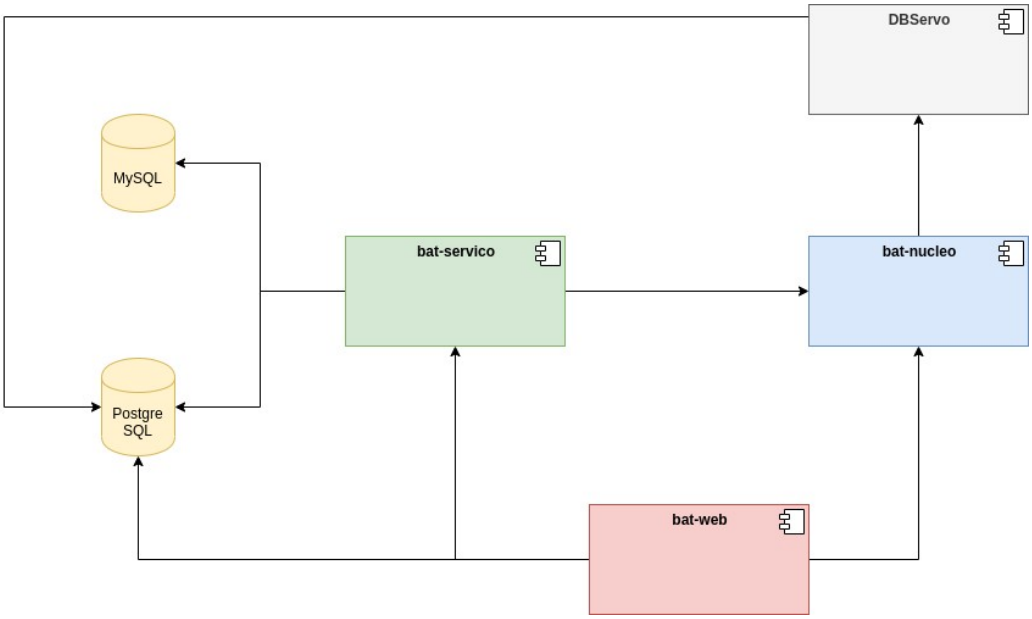


*Figura 1: Diagrama de componentes*

A aplicação utiliza o modelo MVC para a segregação de responsabilidades em camadas, as requisições http são oriundas das páginas JSF e endpoints Rest. 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.

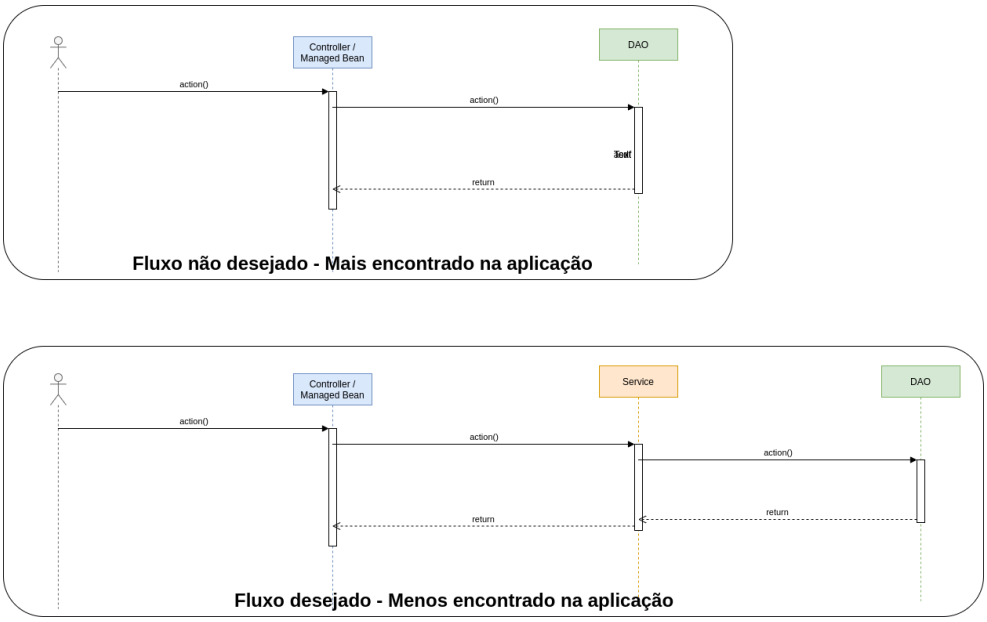


*Figura 2: Diagrama de sequências*

A solução utiliza o schema dbbat como banco de dados core do sistema, utiliza também o schema dbservo, bob, rh, pdi, multas e banco de dados MySQL para acesso as informações do sistema SEI. O alvo de nossa análise será em cima do banco de dados core da aplicação, sendo que este possui um total de 40 tabelas.

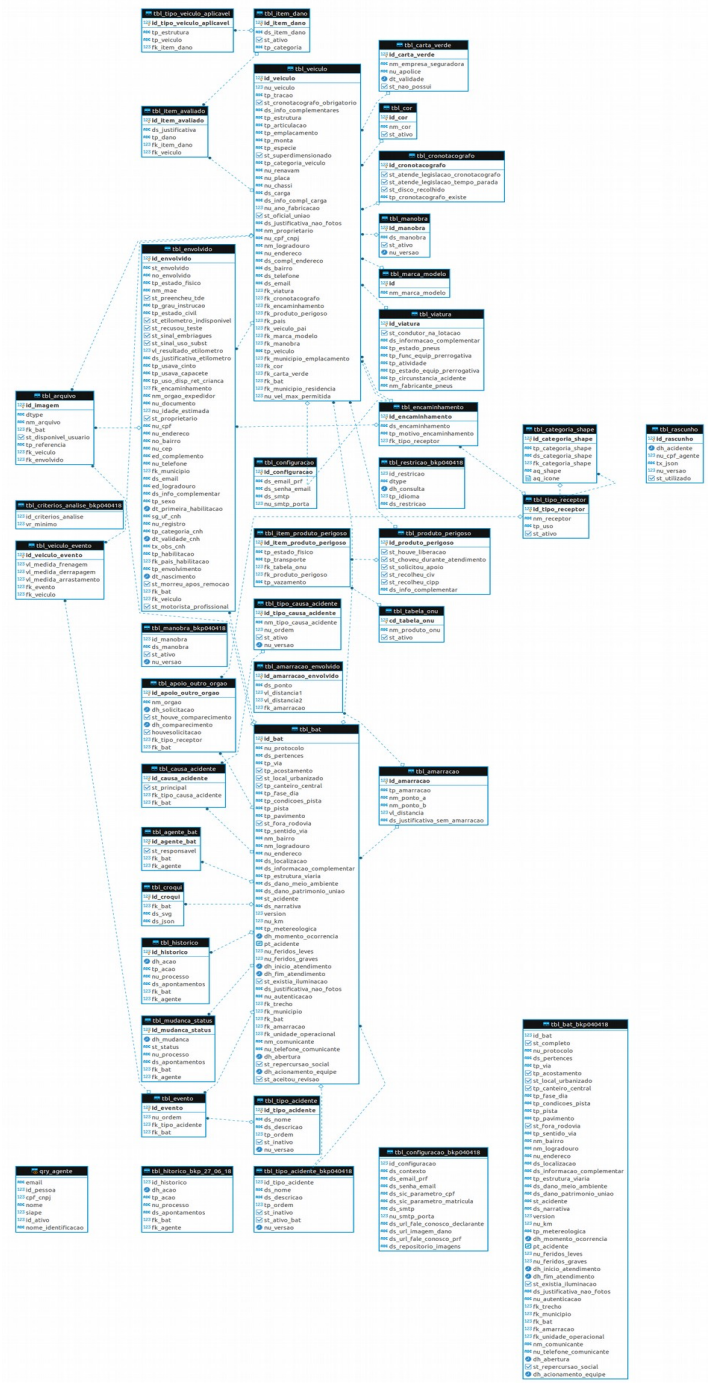

*Figura 3: MER schema dbbat*

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 | 5.1.0 | Framework ORM. | |
| Primefaces | 6.1 | 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. |
| PostgreSQL | | Banco de dados relacional | |
| MySQL | | 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. As análises seguem a sequência bat-nucleo, bat-servico e bat-web.

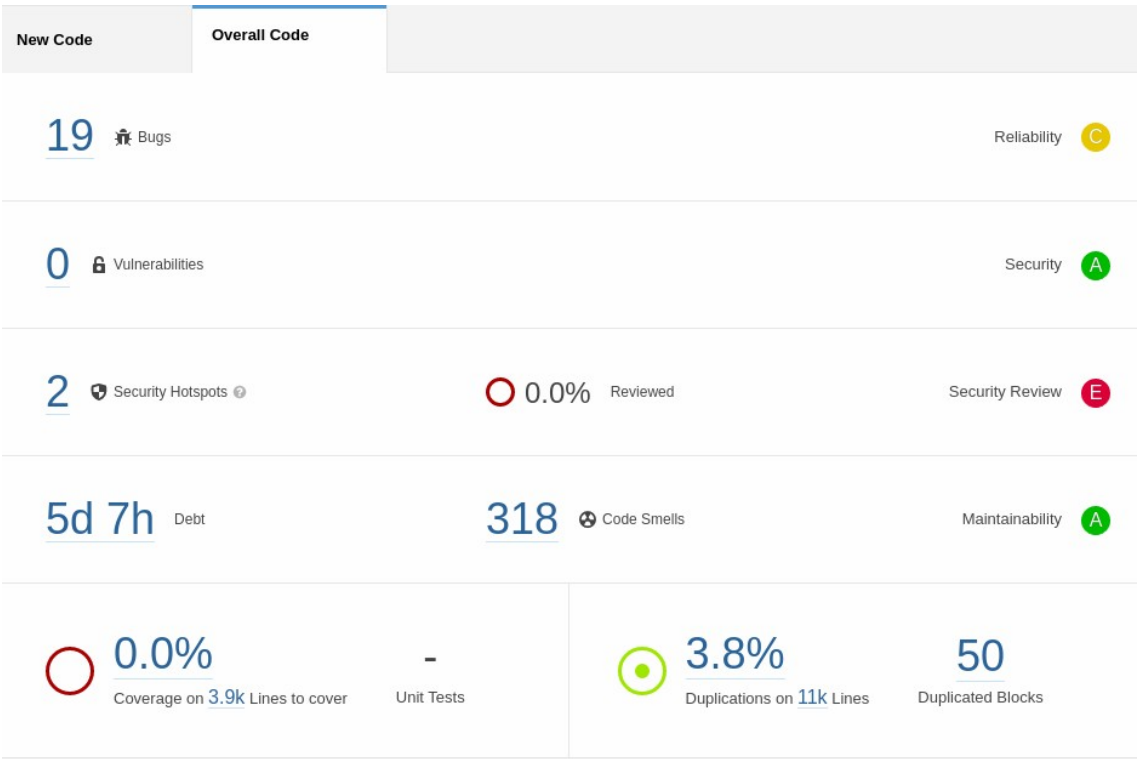


*Figura 4: Análise estática de código : bat-nucleo*

- 19 bugs;
- 0 vulnerabilidades de código;
- 2 violações de segurança;
- 318 violações de código ruim (complexidade cognitiva , complexidade ciclomática e débito técnico);
- 3,8% de duplicidade de código

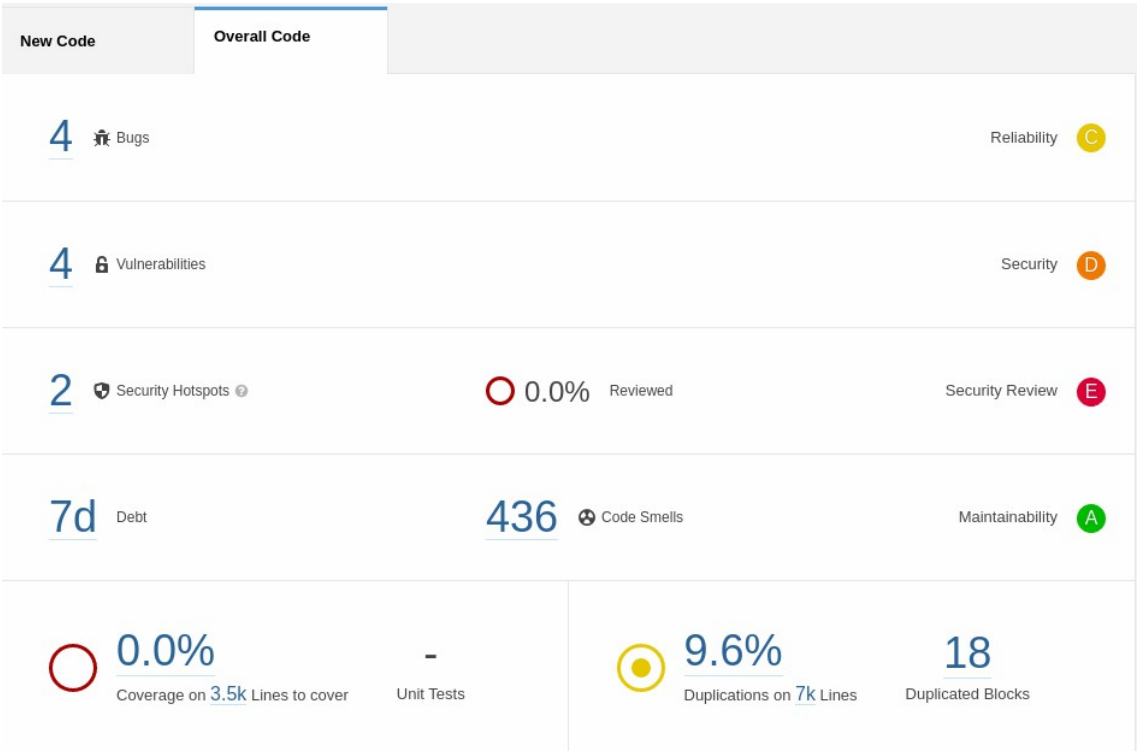


*Figura 5: Análise estática de código : bat-servico*

- 4 bugs;
- 4 vulnerabilidades de código;
- 2 violações de segurança;
- 436 violações de código ruim (complexidade cognitiva , complexidade ciclomática e débito técnico);
- 9,6% de duplicidade de código

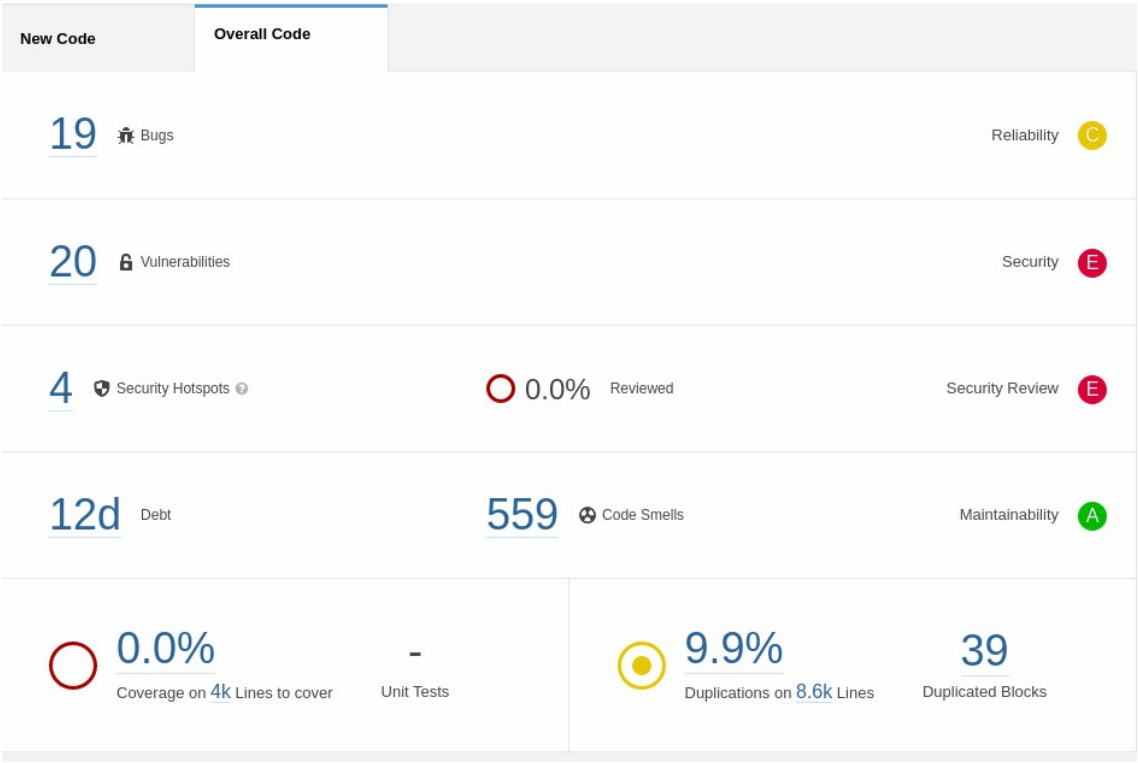


*Figura 6: Análise estática de código : bat-web*

- 19 bugs;
- 20 vulnerabilidades de código;
- 2 violações de segurança;
- 559 violações de código ruim (complexidade cognitiva , complexidade ciclomática e débito técnico);
- 9,9% de duplicidade 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 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 |
|---|---|---|---|---|
| **bat-nucleo** | | | | |
| httpclient-4.2.1.jar | MEDIUM | 2 | Highest | 34 |
| commons-beanutils-1.8.0.jar | HIGH | 2 | Highest | 35 |
| cdi-api-1.1.jar | MEDIUM | 1 | Low | 31 |
| hibernate-validator-5.2.2.Final.jar | HIGH | 1 | Highest | 34 |
| **bat-servico** | | | | |
| commons-email-1.3.3.jar | HIGH | 2 | Highest | 38 |
| httpclient-4.2.1.jar | MEDIUM | 2 | Highest | 34 |
| dom4j-1.6.1.jar | CRITICAL | 2 | Highest | 25 |
| shiro-core-1.3.2.jar | CRITICAL | 2 | Highest | 32 |
| postgresql-9.4.1207.jar | HIGH | 3 | Highest | 48 |
| hibernate-validator-5.2.2.Final.jar | HIGH | 1 | Highest | 34 |
| **bat-web** | | | | |
| bootstrap-4.1.0.jar | MEDIUM | 5 | | 13 |
| jquery-3.0.0.jar | MEDIUM | 2 | | 13 |
| commons-email-1.3.3.jar | HIGH | 2 | Highest | 38 |
| commons-beanutils-1.9.2.jar | HIGH | 1 | Highest | 37 |
| shiro-core-1.3.2.jar | CRITICAL | 2 | Highest | 32 |
| primefaces-6.1.jar | MEDIUM | 2 | Highest | 23 |
| batik-css-1.9.jar | CRITICAL | 1 | Highest | 37 |
| hibernate-validator-5.2.2.Final.jar | HIGH | 1 | Highest | 34 |
| batik-dom-1.9.1.jar | CRITICAL | 1 | Highest | 24 |
| itextpdf-5.5.11.jar | HIGH | 1 | High | 36 |
| httpclient-4.3.4.jar | MEDIUM | 2 | Highest | 34 |
| bootstrap.min.js | MEDIUM | 4 | | 3 |
| primefaces-6.1.jar: jquery.js | medium | 3 | | 3 |

## 4.3 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.

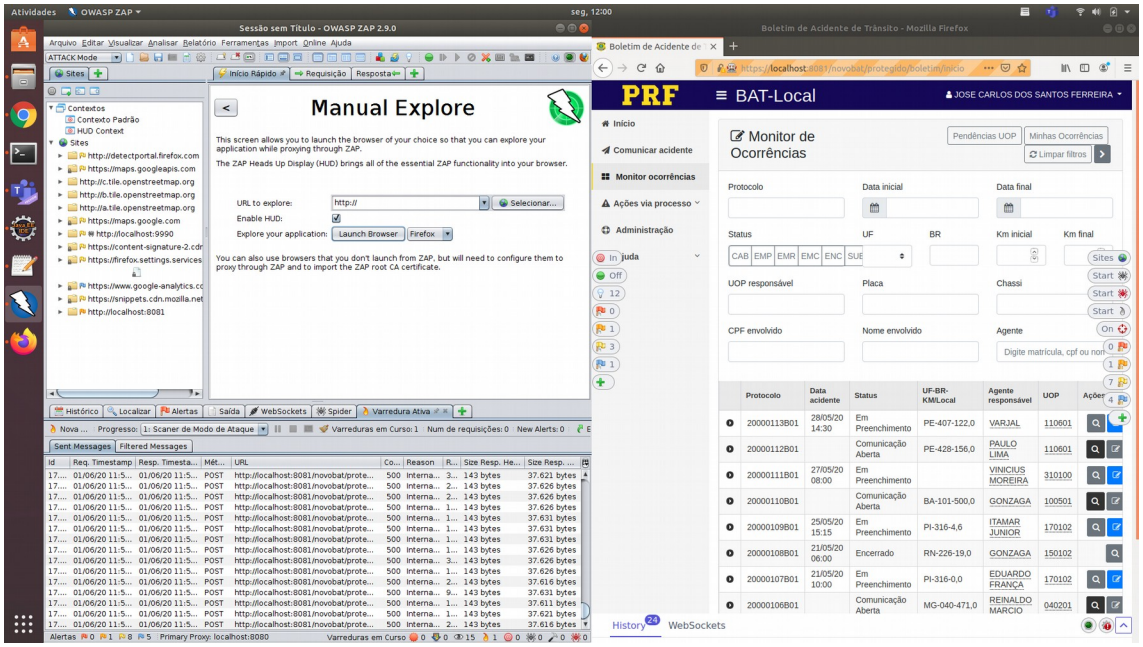


*Figura 7: OWASP ZAP - ferramenta em execução*

- 0 vulnerabilidade de severidade alta;
- 1 vulnerabilidade de severidade média;
- 15 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 possui organização razoavelmente adequada, há segregação por contexto funcional, contudo há repetição de empacotamento dentro dos projetos.

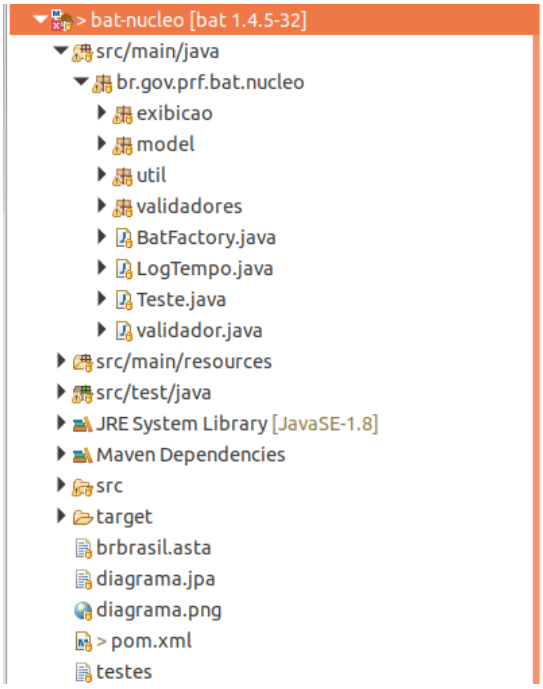


*Figura 8: Estrutura do projeto bat-nucleo*

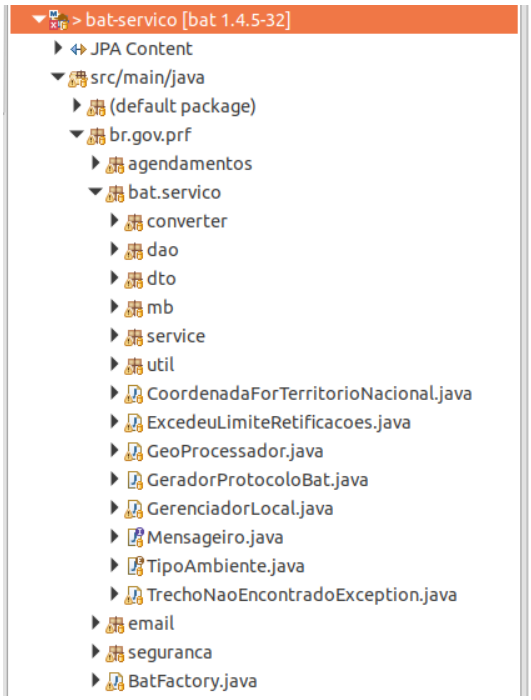


*Figura 9: Estrutura do projeto bat-service*

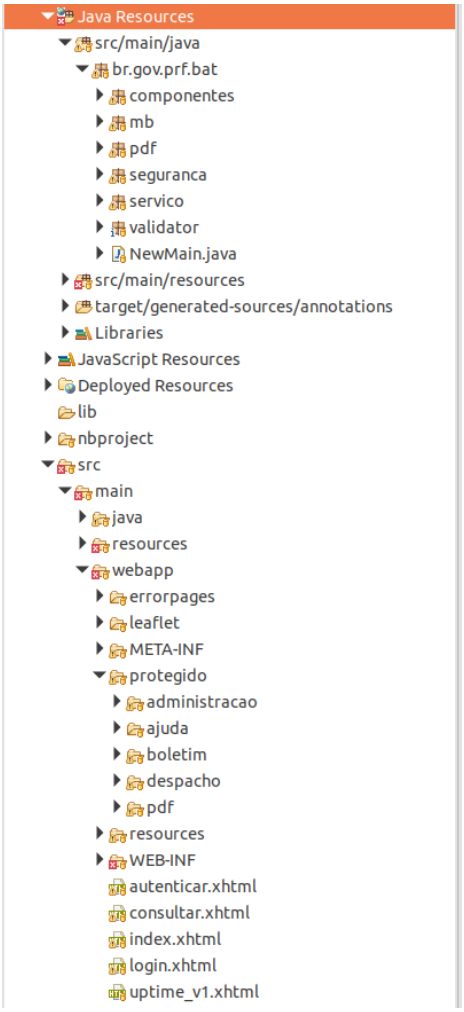


*Figura 10: Figura 9: Figura 8: Estrutura do projeto bat-web*

Percebemos nas ilustrações a existência do pacote serviço dentro do componente bat-web, percebemos também a existência do pacote MB no componente bat-sevice. Ambos empacotamentos estão duplicados e fora do contexto funcional ao qual se propõe o componente ao qual está presente.

## 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 ineficiencia na construção dos testes de unidade de unidade e a falta de cobertura de testes adequada 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 falta de padronização na utilização da segregação de responsabilidade por camadas fere o padrão comportamental da solução.

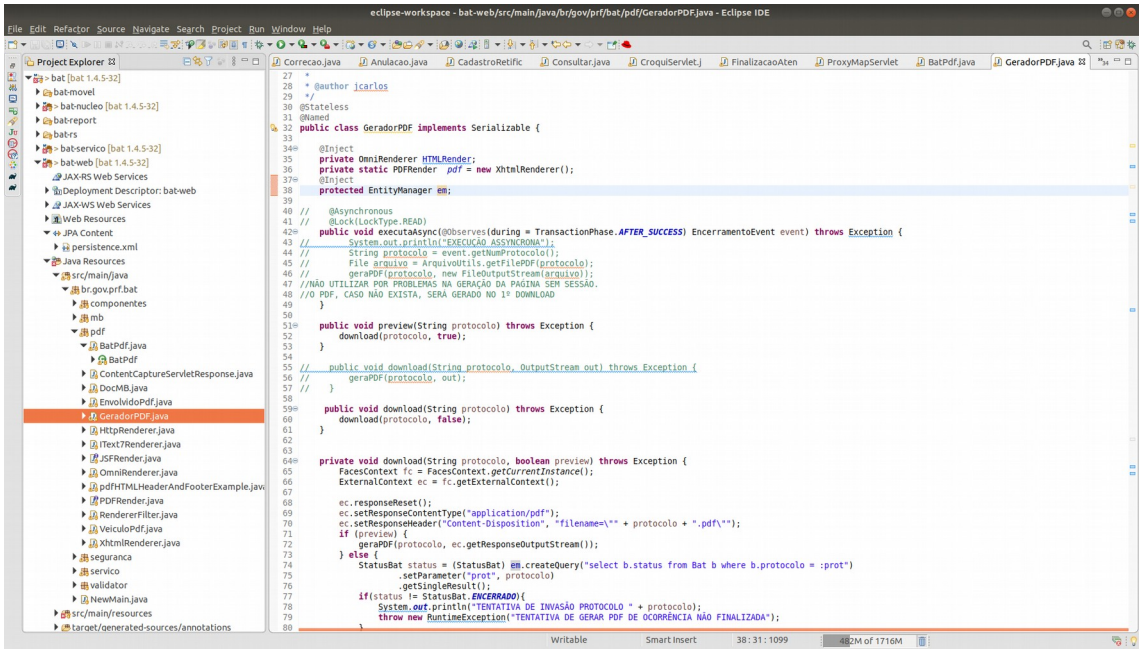


*Figura 11: Managed Bean acessando dados diretamente*

Classes de acesso a dados efetuando validações negociais, não mantendo o nível de isolamento e responsabilidade da classe.

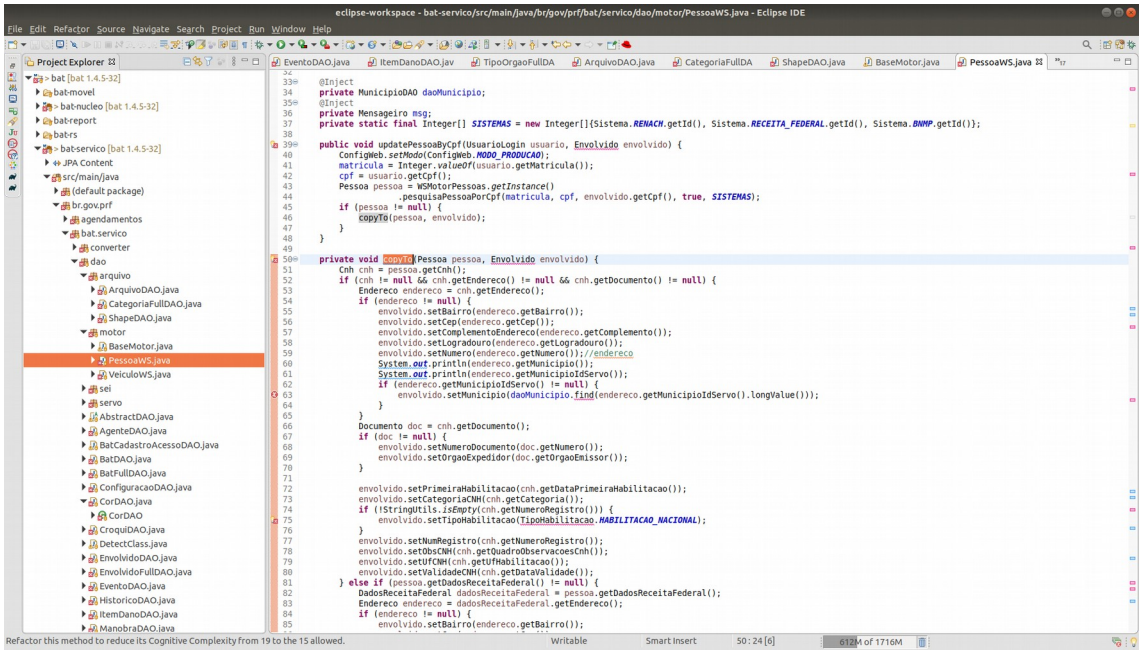


*Figura 12: DAO efetuando validações negociais*

A utilização inadequada de padrões de projeto (design pattern) criacionais aumenta a dificuldade da manutenção do código. Padrões do tipo Abstract Factory ou Factory Method deferiam ser utilizados para a resolução de problemas como este.

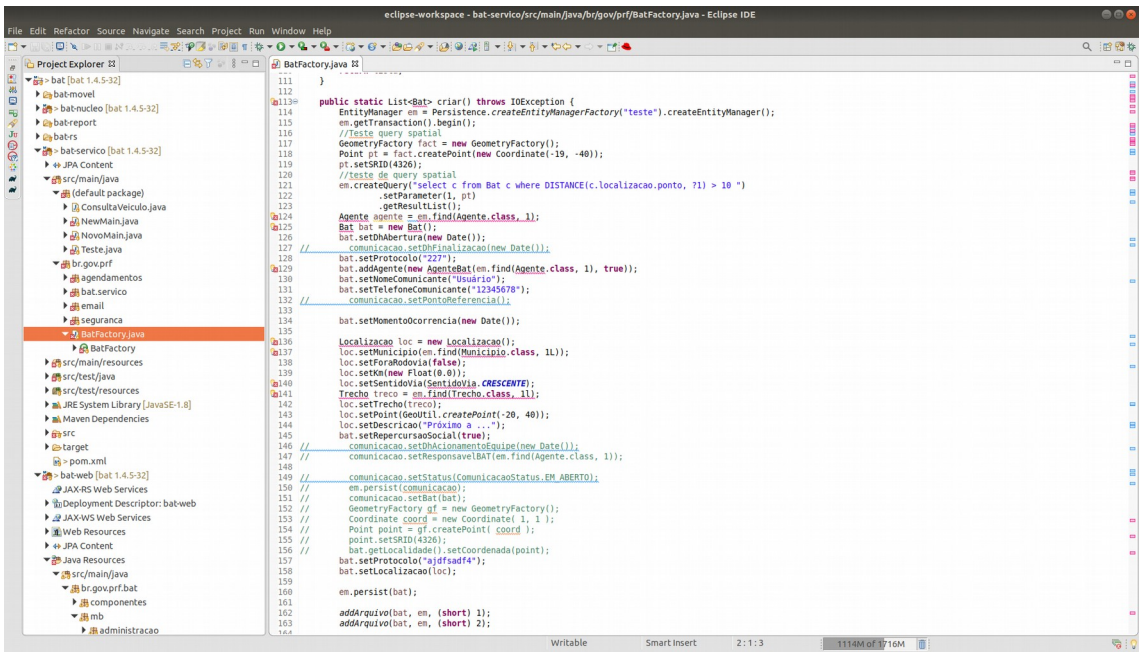


*Figura 13: Inadequação de utilização de pattern criacional*

A tratativa inadequada dos logs da aplicação certamente não auxiliam o administrador/desenvolvedor quanto a descoberta de falhas no sistema. A ilustração abaixo demonstra 296 ocorrências de utilização do comando System.ou.print* nos componentes da aplicação.

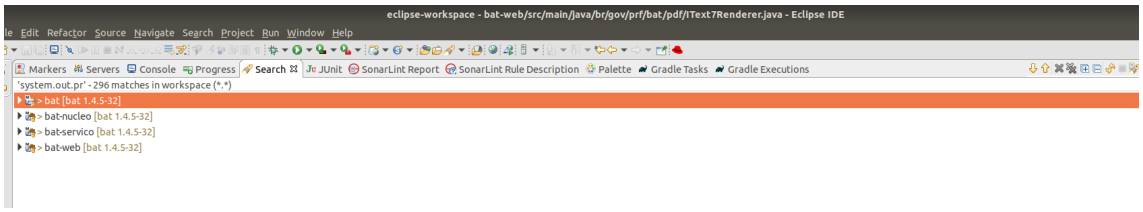


*Figura 14: 296 ocorrências da utilização do comando System.out.print**

## 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, esta ação não está presente em todas as partes da aplicação.

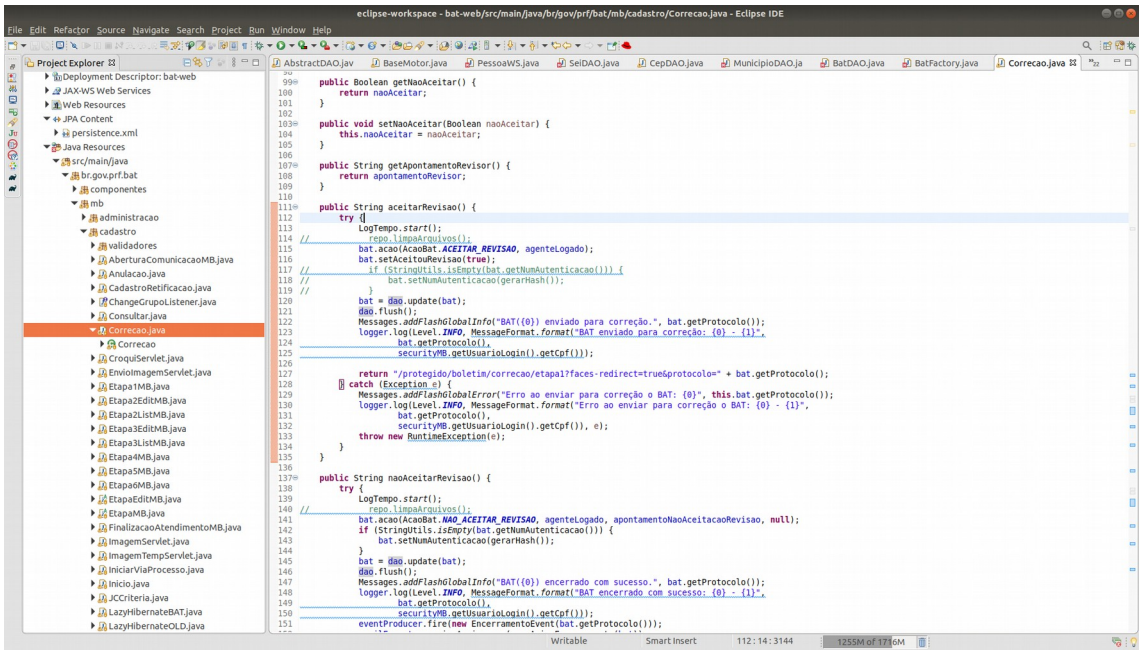


*Figura 15: Managed Bean sem controle transacional*

A imagem a seguir demonstra o que seria o comportamento esperado pela aplicação no que tange ao uso do controle transacional nas operações que envolvem operações DML em banco de dados.

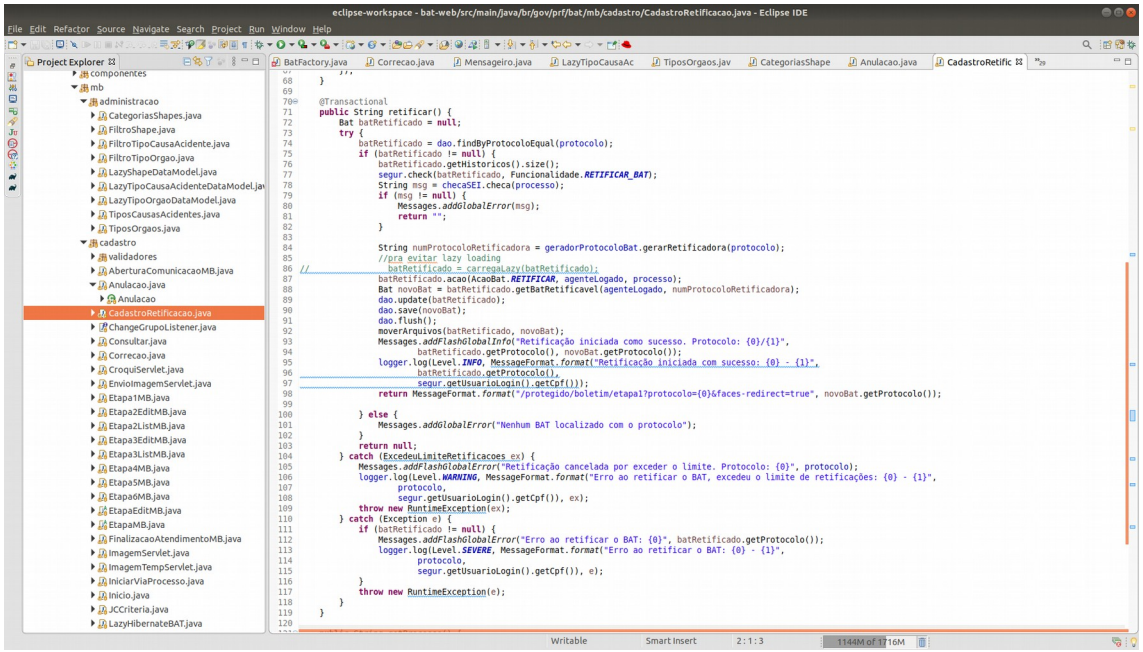


*Figura 16: Managed Bean executando operações em banco com controle transacional*

As operações executadas em lote no core da aplicação por intermédio de execuções agendadas também não utilizam controle transacional nas operações de manipulação de dados.

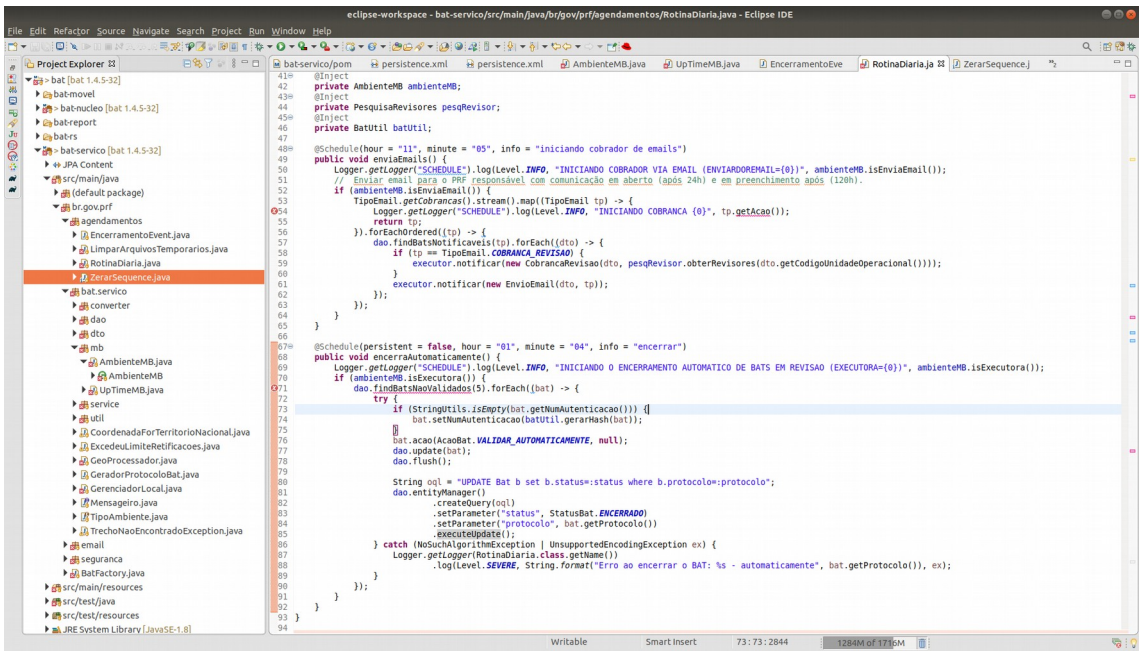


*Figura 17: Execuções em lote sem controle trasacional*

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 levanto 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.

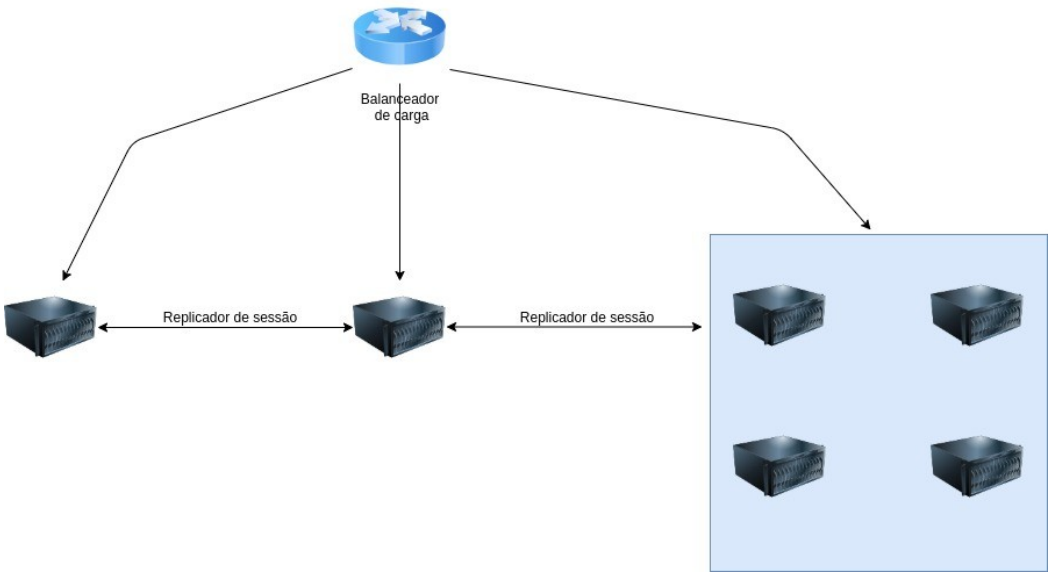


*Figura 18: Escalabilidade do monólito*

# 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

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

Por diretriz de utilização de micro serviços implantada na DPRF, recomenda-se que sejam substituídos todo o contexto de acesso as bases de dados (com a exceção do schema dbbat) pela utilização dos mesmos por intermédio da utilização do barramento de serviços.

Recomenda-se também que seja substituídos os paths estáticos utilizados na classe ArquivoUtil (conforme imagem abaixo) por variáveis de ambiente configuráveis no servidor de aplicação. Esta tratativa trás mais dinamismo e independência de código.

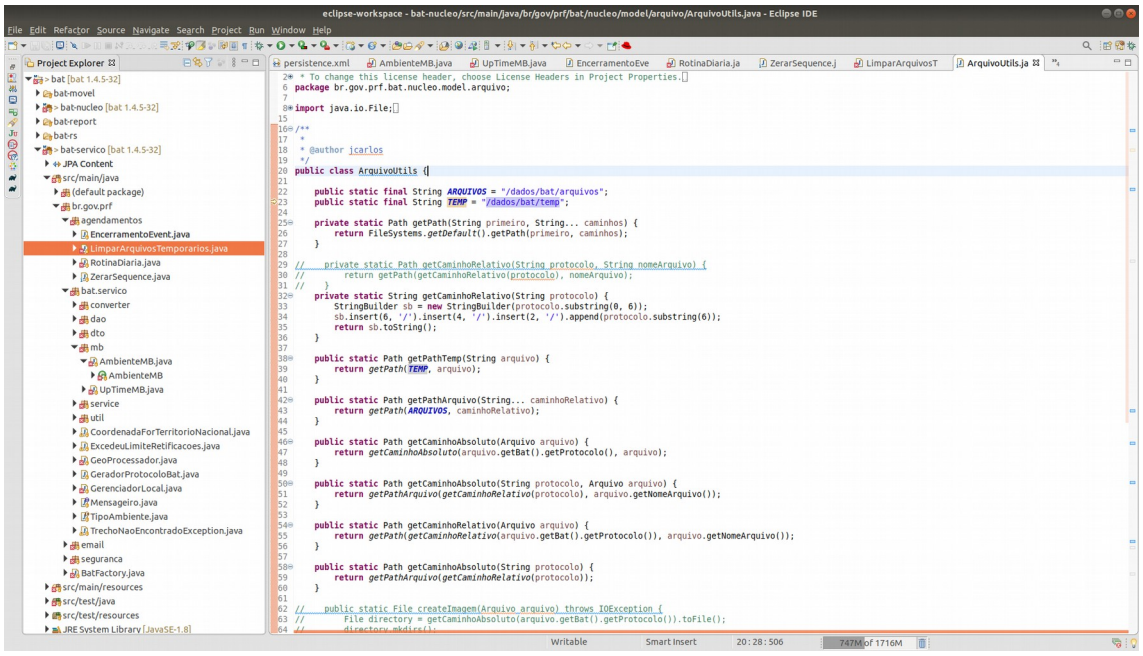


*Figura 19: Path estático em código fonte*

Para fins de organização, padronização e documentação, recomenda-se que seja mantido de forma permanente 3 branchs 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.