



Ministério da Justiça

**Projeto:** RECALL

# Nota Técnica

<b>MJ</b>	<b>RECALL - Nota Técnica</b>	
-----------	------------------------------	--

<b>Revisão</b>	<b>Descrição</b>	<b>Autor</b>	<b>Data</b>
1.0	Construção do documento	Israel Branco	17/04/2020

# 1 Sumário

2 Introdução.....	4
3 Apresentação do cenário atual.....	5
3.1 Estrutura do projeto.....	6
3.2 Tecnologias utilizadas.....	7
3.3 Modelagem de dados.....	8
4 Análise técnica.....	9
4.1 SonarQube.....	9
4.2 OWASP ZAP.....	10
4.4 Análise sobre os resultados.....	11
4.4.1 Manutenibilidade de código.....	11
4.4.3 Performance e estabilidade.....	13
4.4.3 Escalabilidade.....	13
5 Recomendações.....	14
6 Conclusão.....	15

## 2 Introdução

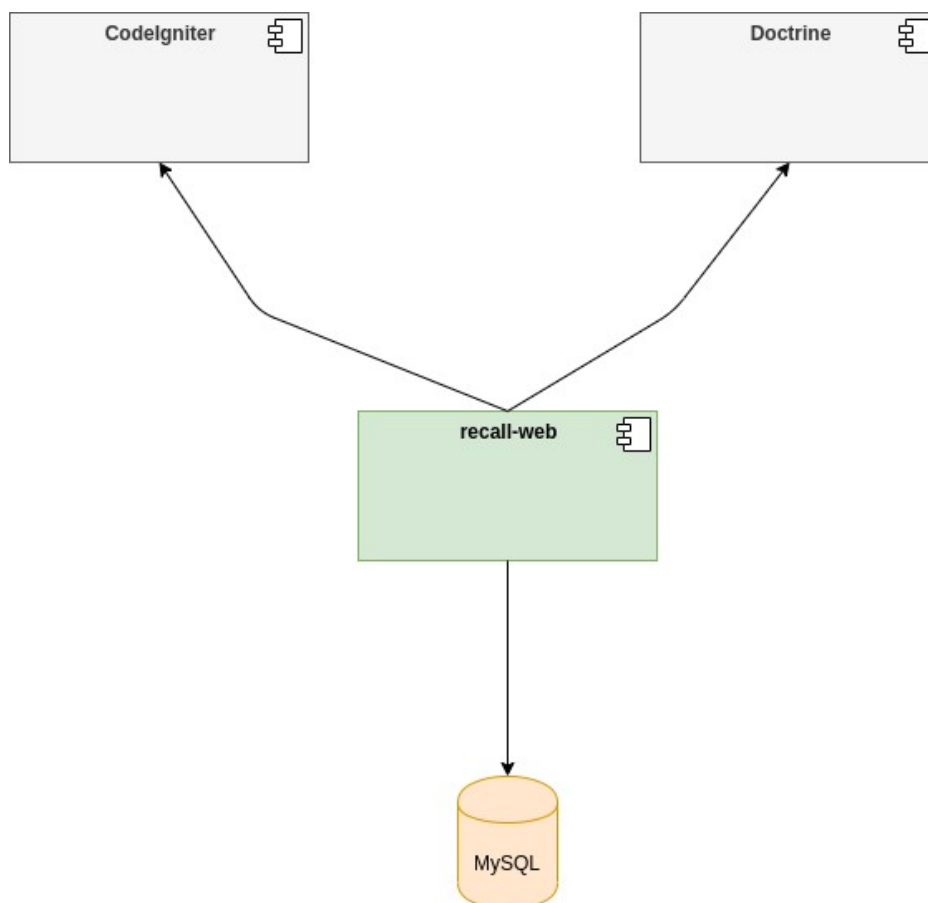
Este documento visa reportar o resultado da análise efetuada na aplicação RECALL. Para este estudo foram desconsiderados todo o contexto comercial ao qual a ferramenta está inserida, também foram desconsideradas o ambiente ao qual a ferramenta está operando sendo analisado puramente questões que tangem a qualidade de código, padrões de codificação, vulnerabilidades de dependências, modelo relacional de banco de dados e concepção arquitetural.

Par a realização desta análise foi criado a tag 001-nota\_tecnica\_ctis com origem na branch stable na 12/04/2020 no repositório <https://gitlab.mj.gov.br/cgsis/recall> do Ministério da Justiça.

### 3 Apresentação do cenário atual

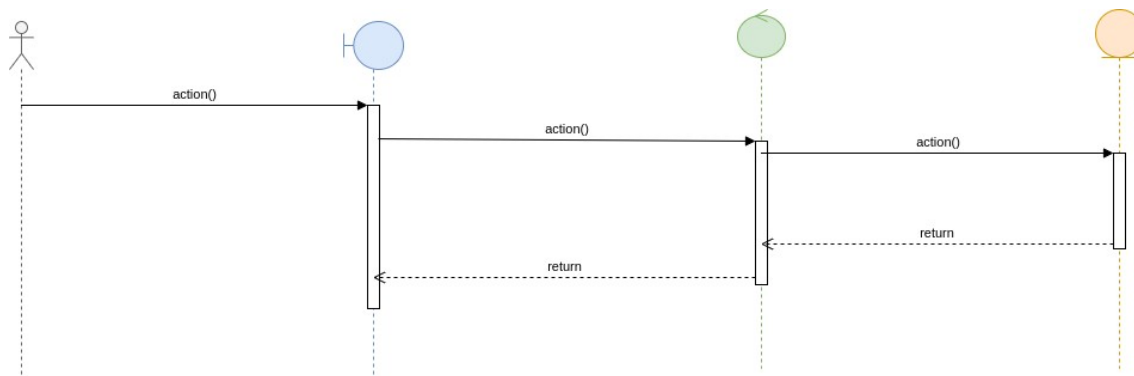
Esta sessão ira descrever a arquitetura, tecnologias, frameworks e dependências que compõe a base da aplicação.

RECALL está construído para funcionar em ambiente WEB com tecnologia PHP 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 MySQL.



*Figura 1: Diagrama de componentes*

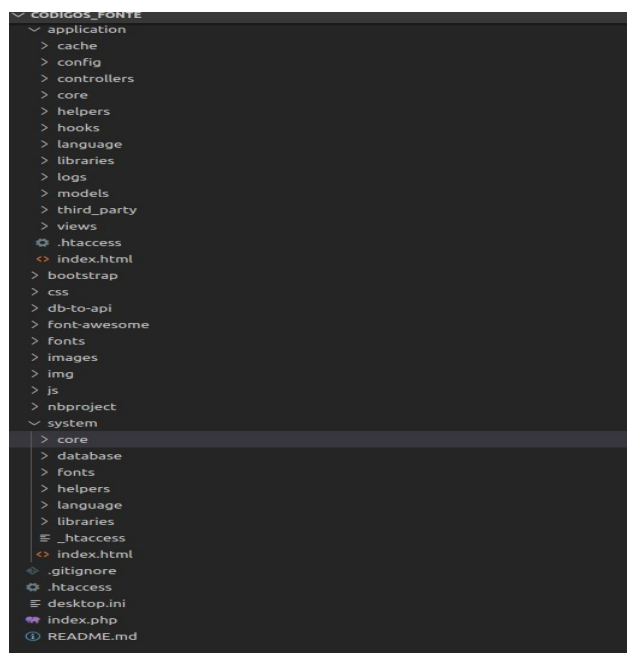
O software está estruturado sobre a arquitetura MVC com segregação das camadas segundo sua necessidade negocial.



*Figura 2: Fluxo básico - View->Controller->Model*

### 3.1 Estrutura do projeto

O projeto está organizado de forma simples e com boa coesão e segregação de funcionalidades e a estruturação do projeto favorece a visibilidade do fluxo sequencial básico demonstrado na Figura 2.



*Figura 3: Organização do projeto*

### 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
PHP	5.6	Linguagem de programação	
CodeIgniter		Framework MVC	
Doctrine		Framework ORM	
MySQL	5.7	Servidor de banco de dados	
Apache	2.x	Servidor de aplicação	



### 3.3 Modelagem de dados

A estrutura de banco de dados utiliza um único schema composto por 39 tabelas.

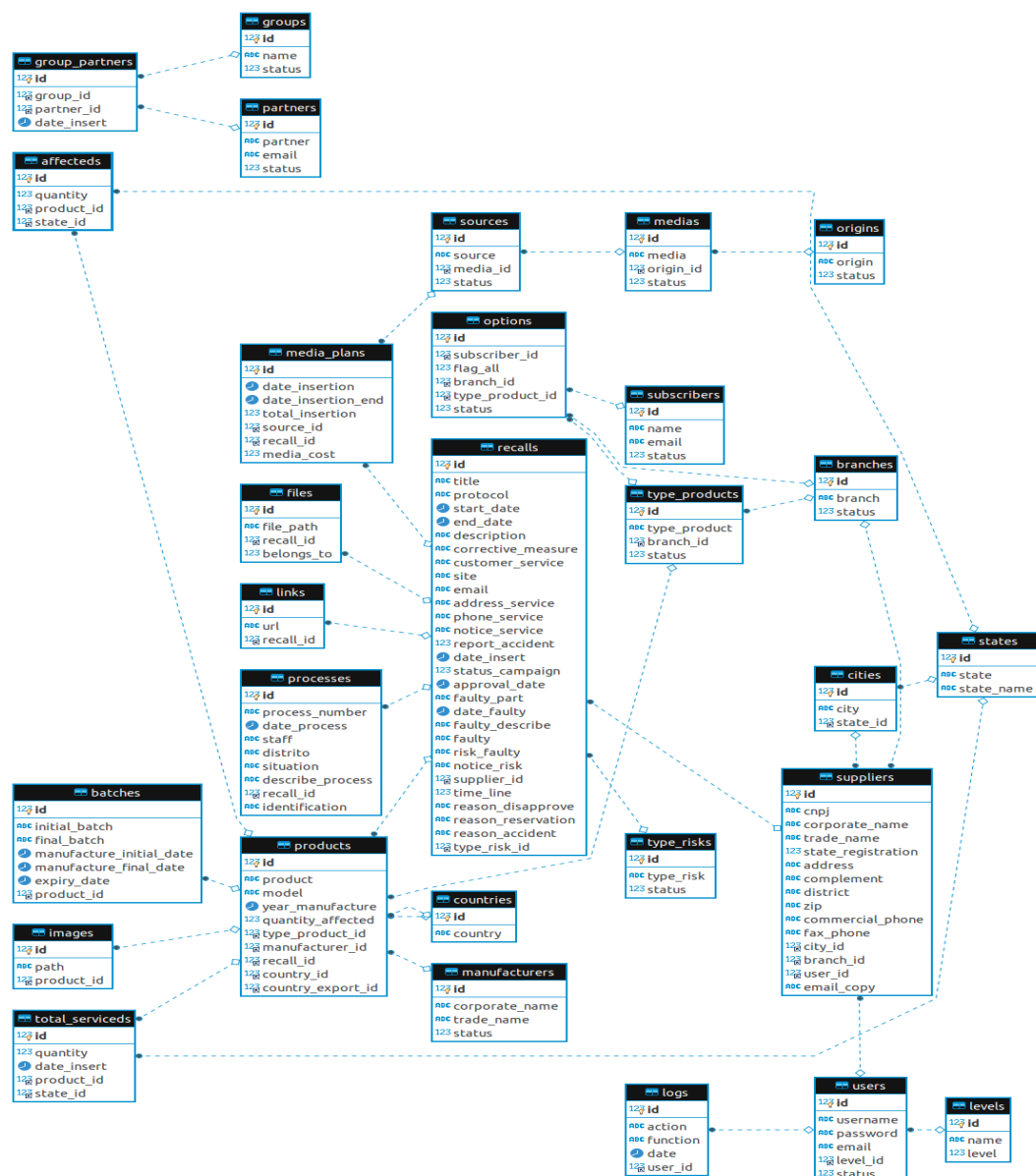


Figura 4: Modelo entidade relacional



## 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 a aplicação:

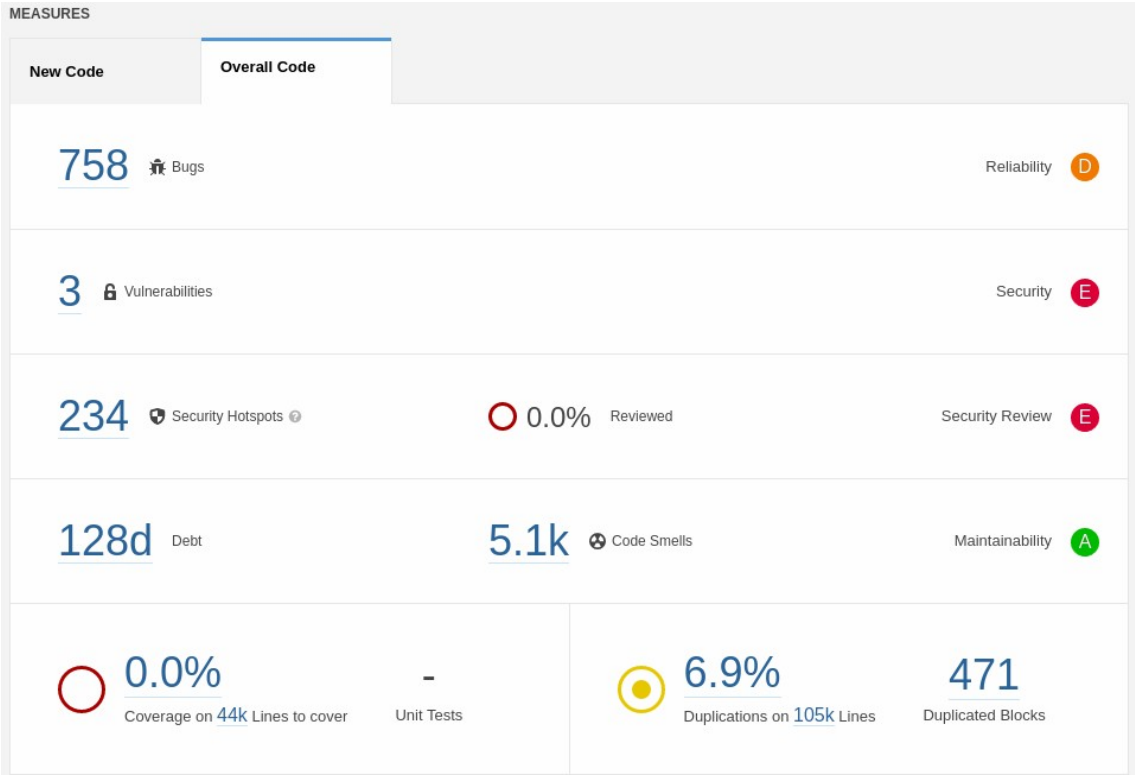


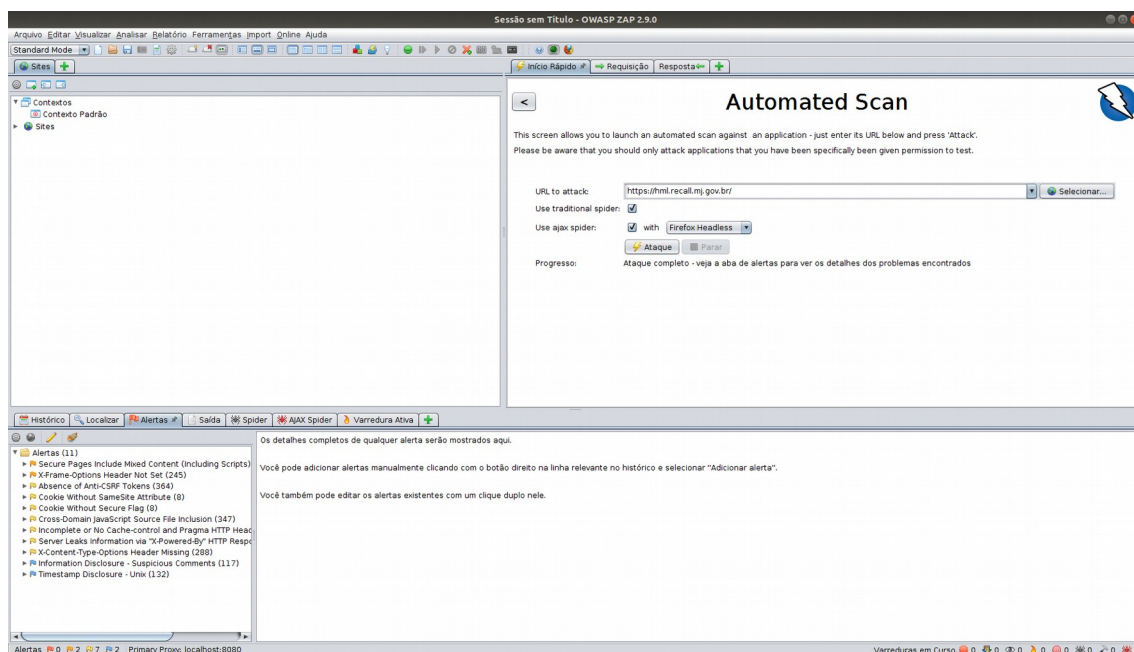
Figura 5: Análise estática de código

- 758 bugs;
- 3 vulnerabilidades;
- 234 violações de segurança;
- Mais de 5 mil violações de práticas ruins de desenvolvimento (débito técnico, complexidade ciclomática e outras);
- 6.8% de duplicação de código;

Vale ressaltar que no momento desta análise, o projeto não dispunha de evidências de artefatos de teste de unidade.

## 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: OWASP ZAP*

O anexo I deste documento contempla a análise completa da ferramenta OWASP ZAP.

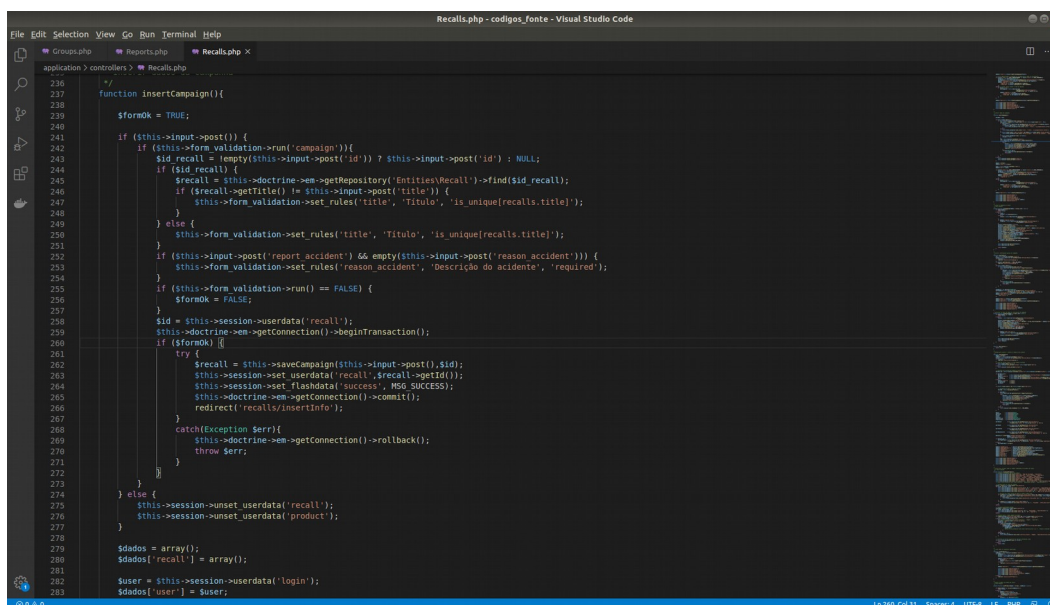
## 4.4 Análise sobre os resultados

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 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 traz a dificuldade no processo de refactoring da aplicação uma vez que não há condições de mensurar impactos durante o processo de manutenção corretiva/adaptativa. Alinhado a esta questão, a alta complexidade ciclomática dificulta este processo de refactoring, a ilustração abaixo demonstra ambos cenários apresentados em um único método (OBS: a característica apresentada é utilizada de forma recorrente em diversos momentos do código).



```

226
227 function insertCampaign(){
228
229     $formOk = TRUE;
230
231     if ($this->input->post()) {
232         if ($this->form-validation->run('campaign')){
233             $id_recall = empty($this->input->post('id')) ? $this->input->post('id') : NULL;
234             if ($id_recall) {
235                 $recall = $this->doctrine->em->getRepository('Entities\Recall')->find($id_recall);
236                 if ($recall->getTitle() != $this->input->post('title')) {
237                     $this->form-validation->set_rules('title', 'Título', 'is_unique[recalls.title]');
238                 }
239             } else {
240                 $this->form-validation->set_rules('title', 'Título', 'is_unique[recalls.title]');
241             }
242             if ($this->input->post('report accident') && empty($this->input->post('reason accident')) {
243                 $this->form-validation->set_rules('reason accident', 'Descrição do acidente', 'required');
244             }
245             if ($this->form-validation->run() == FALSE) {
246                 $formOk = FALSE;
247             }
248             $id = $this->session->userdata('recall');
249             $this->doctrine->em->getConnection()->beginTransaction();
250             if ($formOk) {
251                 try {
252                     $recall = $this->saveCampaign($this->input->post(), $id);
253                     $this->session->set_userdata('recall', $recall->getId());
254                     $this->session->set_flashdata('success', MSG_SUCESSO);
255                     $this->doctrine->em->getConnection()->commit();
256                     redirect('recalls/insertInfo');
257                 } catch (Exception $err){
258                     $this->doctrine->em->getConnection()->rollback();
259                     throw $err;
260                 }
261             } else {
262                 $this->session->unset_userdata('recall');
263                 $this->session->unset_userdata('product');
264             }
265             $dados = array();
266             $dados['recall'] = array();
267             $user = $this->session->userdata('login');
268             $dados['user'] = $user;
269         }
270     }
271 }
272
273
274
275
276
277
278
279
280
281
282
283

```

Figura 7: Complexidade ciclomática

Outra característica encontrada no código fonte deste projeto é a quantidade excessiva de linhas escritas nas principais classes Controller's, estas chegam a ter entre 1000 a 3000 linhas de códigos. A quantidade excessiva de linhas denota a possível extrapolação de sua zona de atividade, técnicas como composição e delegação poderiam ser aplicadas as mesmas.

Outra característica apresentada no projeto é a falta de camada para tratamento de regras negociais, toda regra está sendo executada nas classes Controller's.

Os fatores apresentados prejudicam a manutenibilidade do código.

A camada de visão possui extrema coesão e não há aplicabilidade de regras de negócio ou controle de formulário em sua composição.

#### 4.4.2 Confiabilidade

Não há evidências de controle transacional em nenhuma camada da aplicação, a falta deste tratamento fere boas práticas de desenvolvimento de aplicações e a falta do mesmo não garante as propriedades ACID do SGBD.

Características como demonstrado na figura a seguir são frequentes na aplicação, nota-se uma série de alteração de dados sem o devido controle transacional na operação que orchestra as mesmas demais chamadas. Este tipo de prática pode trazer inconsistências nos dados da aplicação.

```
$this->doctrine->em->persist($product);
$this->doctrine->em->flush();

$this->saveCountryExports($product);
$this->saveImages($product);
$type = UPLOAD_IMAGE; // tipo de do arquivo para fazer upload
$this->do_upload($type,$product->getId());
$this->saveBatch($product);
$this->saveAffecteds($product);
```

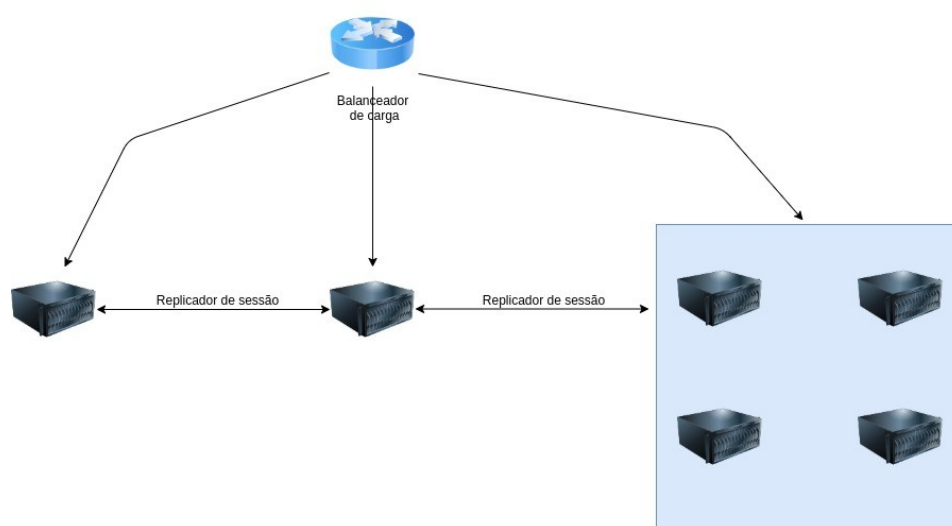
#### 4.4.3 Performance e estabilidade

Não foi analisado a aplicação em funcionamento para avaliar demais requisitos não funcionais. Durante o processo de análise de código fonte, não foram encontradas evidências que demonstrem impactos em performance da aplicação.

#### 4.4.3 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 empreendidos nos demais nós existentes. Nesta arquitetura não há a possibilidade de escalar somente as funcionalidades/módulos que mais são demandados.



*Figura 8: Escalabilidade horizontal*

## 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.

Tendo em vista a não utilização de gerenciadores de pacotes/dependências (ex Composer), não foi possível determinar as versões dos frameworks utilizados na construção da ferramenta. Neste sentido, estas dependências foram excluídas no processo de análise estática de código, contudo sugere-se os ajustes de segurança analisados pela ferramenta de intrusão OWASP ZAP.

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.

## 6 Conclusão

A aplicação apresenta estruturação razoável em sua construção fato este que não propícia facilidade na manutenção corretiva/evolutiva. A arquitetura monolítica e a quantidade de código nas classes Controller's dificultam a escalabilidade e a manutenibilidade do código.

Em relação aos ajustes de código da aplicação, sugestiona-se que os mesmos sejam efetuados com o auxílio/suporte de testes unitários tendo em vista que este caminho trará maior assertividade e menor risco para o não comprometimento da estabilidade da ferramenta.