Curso de Sistemas de Informação
Universidade Estadual de Mato Grosso do Sul

Desenvolvimento Web com *Framework Phalcon*

Marcos Vinicius O. Silveira

Prof. Dr. Ricardo Luís Lachi(Orientador)
Prof. Esp. Jónison Almeida dos Santos (Co-orientador)

Dourados -MS
Novembro de 2015

Curso de Sistemas de Informação
Universidade Estadual de Mato Grosso do Sul

Desenvolvimento Web com *Framework Phalcon*

Marcos Vinicius O. Silveira

Novembro de 2015

Banca Examinadora:

Prof. Esp. Jónison Almeida dos Santos (Co-orientador)
Área de Computação - UEMS

Prof. Msc. Diogo Fernando Trevisan
Área de Computação - UEMS

Prof. Msc. Jéssica Bassani
Área de Computação - UEMS

Desenvolvimento Web com *Framework Phalcon*

Marcos Vinicius O. Silveira

Este exemplar corresponde à redação final da monografia da disciplina Projeto Final de Curso II devidamente corrigida e defendida por Marcos Vinicius O. Silveira e aprovada pela Banca Examinadora, como parte dos requisitos para a obtenção do título de Bacharel em Sistemas de Informação.

Dourados, 24 de novembro de 2015.

Prof. Dr. Ricardo Luís Lachi (Orientador)

Prof. Esp. Jónison Almeida dos Santos
(Co-orientador)

Resumo

A linguagem PHP se tornou popular por ser de fácil aprendizado e com o tempo de desenvolvimento relativamente menor em relação a outras linguagens *webs*. Porém essas características permitem que programadores pouco experientes criem programas vulneráveis a ataques e com péssima manutenibilidade. A utilização de *frameworks* auxilia no desenvolvimento e ajuda a evitar estes problemas através dos padrões de projetos. Neste aspecto, o uso do Phalcon é de grande valia, pois resolve, além do exposto, outro problema comum: a velocidade. Por ser desenvolvido utilizando a linguagem C e disponibilizado como uma extensão para PHP possui uma velocidade superior a outros *frameworks* gerando *softwares* com baixo custo, modularizados e rápidos sem necessidade de modificações nas configurações de servidor ou utilização de hardwares robustos.

Neste trabalho é apresentado as caracteristicas do *framework phalcon*, a utilização do padrão de projetos *MVC*, e utilizando estas tecnologias efetuado o desenvolvimento de um SGC (Sistema de Gerenciamento de Conteúdo) nomeado *Pluton*.

Palavras-chave: *PHP, Framework*, *Phalcon*, *web*, padrões de projetos, *MVC*, *Pluton*.

Abstract

The PHP language has become popular because of its easy learning and with a time of development relatively short in relation to other languages. However, these features allows novice programmers to build programs that are vulnerable to attacks and with a bad maintainability. The use of frameworks in addition to assisting in the development also helps prevent these problems through some standards that are required in the development. In this respect the use of phalcon is of great value, because solves addition to the above, another common problem: the speed. by be developed using the C language and made available as an extension for PHP has it a superior speed among other frameworks generating software with low cost, modularized and fast without the need of modifications in the server settings or the use of robust hardware.

This paper presents the characteristics of phalcon framework, the use of Design Partnness MVC projects, and using these technologies made the development of a CMS (Content Management System) named Pluton.

Key-words: PHP, Framework, Phalcon, web, design pattners, MVC, Pluton.

Agradecimentos

Agradeço primeiramente a minha família pelo apoio e compreensão destes longos anos acadêmicos.

A minha mãe Gedinéia por sempre me apoiar e ter paciência com esse ser incompreensível que ela gerou e ama sem julgar.

Aos Amigos que fiz durante o curso sem os quais não conseguiria chegar até aqui, meus exemplos e motivo de sempre querer melhorar, Evandro, Guilherme e Valter.

Também gostaria de agradecer aos amigos que aguentaram aqueles momentos de insanidade e reclamação durante esta fase complicada e nunca deixaram de me incentivar e motivar, Elton Servilha, Fabiano Nascimento Santos, Evaldo Fernando dos Santos, Eduardo Menezes e a todos os amigos que de alguma forma influenciaram para a formação da pessoa que sou hoje.

Aos meus Orientadores Jónison Almeida dos Santos e Ricardo Luíz Lachi por seus auxílios, concelhos e paciência durante o desenvolvimento deste projeto.

A todos os professores do curso, que transmitiram e transmitem da melhor forma possível seus conhecimentos para nós acadêmicos e dedicam suas vidas a nos ensinar.

Gostaria ainda de registrar aqui meu agradecimento especial uma pessoa que apesar de não fazer mais parte da minha vida, sem ela este projeto não teria se realizado. A você que talvez nunca leia isso aqui, meu mais sincero muito obrigado.

Sumário

Lista de Siglas

HTTP *Hypertext Transfer Protocol*

MVC *Model, View* e *Controller*

OO Orientação a Objetos

PHP *Personal Home Page: Hipertext Preprocessor*

SGC Sistema de Gerenciamento de Conteúdos

SGBD Sistema de Gerenciamento de Banco de Dados

SQL *Structured Query Language*

YAF *Yet Another Framework*

Lista de Figuras

Capítulo 1

Introdução

Uma das maiores dificuldades no desenvolvimento de aplicações, sejam *web* ou *desktop* é a falta de padronização. Segundo Junior (2006) uma aplicação *web* deve possuir conexão com banco de dados, inserções, alterações e remoções de informações, envio de formulários, entre outras funcionalidades.

Basicamente pode-se criar uma classe para conexão com o banco, uma outra para envio de elementos via *GET*[1] e *POST*[2] e iniciar sua aplicação a partir disto. Mas em uma aplicação complexa, com inúmeros recursos, tratamento de dados, feeds[3] e vários desenvolvedores trabalhando simultaneamente no projeto, apenas algumas classes podem não ser suficientes, o código pode acabar ficando desorganizado, sem padrão e com uma difícil manutenção.

Para resolver este tipo de problema existem os padrões de projetos (*Design Pattners*). Melo and NASCIMENTO (2007) define padrões de projetos como práticas eficientes e testadas ao longo dos anos utilizadas para solucionar problemas comuns entre projetos, tornando mais fácil a reutilização de arquiteturas bem sucedidas.

Um *framework* é uma arquitetura que através da utilização de um ou mais padrões de projetos, fornece várias ferramentas comuns a todo tipo de aplicação proporcionando um ambiente de desenvolvimento mais produtivo (Minetto, 2007).

Este projeto consiste no desenvolvimento de uma aplicação *web* com a linguagem PHP utilizando o padrão de projetos *Model, View* e *Controller* (MVC) e o *framework phalcon* com a finalidade de demonstrar as funcionalidades e facilidades da utilização destas ferramentas

1.1 Objetivo Geral

O Objetivo geral deste trabalho é conceituar o *framework phalcon*, o padrão de projetos MVC e os Sistemas de Gerenciamento de conteúdo (SGCs), descrevendo características

[1]Método utilizado para envio de pequenas informações para outra página através da URL.

[2]Utilizado para enviar informações de uma página a outra de maneira segura através de uma conexão paralela entre o navegador e o servidor *web*.

[3]São listas de atualização de conteúdos de *web sites*.

e funcionalidades com o intuito de fornecer material para profissionais interessados na utilização destas tecnologias.

1.1.1 Objetivos Específicos

- Conceituar funcionalidades, vantagens e características do *framework phalcon*.
- Descrever a utilização e criação de aplicações utilizando padrão de projetos MVC orientado a objetos.
- Definir exemplificar e desenvolver um SGC (Sistema de Gerenciamento de Conteúdo) para gerenciamento de sites com o *Framework Phalcon* e o padrão de projetos MVC.

1.2 Motivação

Grande parte dos projetos de aplicações *web* atuais utilizam padrões de projetos e *frameworks* para homogenizar e agilizar o desenvolvimento. A maioria dos *frameworks* PHP utilizam MVC e são desenvolvidos utilizando a própria linguagem PHP. *Phalcon* se destaca por ser desenvolvido em uma linguagem compilada, possuindo por isso maior velocidade. Também é um projeto *Open Source*[4] desenvolvido por uma comunidade ativa e possui um fórum próprio onde os desenvolvedores podem tirar suas dúvidas e conseguir informações sobre o projeto.

Apesar do fórum ativo onde dúvidas sobre desenvolvimento e utilização podem ser sanadas pelos participantes, existem poucos artigos e livros publicados sobre o assunto, seja em português ou em inglês, dado principalmente pelo fato de ser um *framework* consideravelmente novo com apenas 3 anos desde a sua criação.

Este trabalho é um grande ganho para a comunidade acadêmica brasileira disponibilizando um conteúdo em português para consulta de informações e funcionalidades do referido *framework.*

1.3 Metodologia

Este trabalho foi baseado em uma pesquisa bibliográfica, constituída principalmente de livros e artigos científicos, destacando os conceitos e as características das linguagens e ferramentas a serem utilizadas no desenvolvimento do projeto.

O Projeto físico foi desenvolvido em conformidade com as pesquisas realizadas e aqui contidas.

[4]Programa que pode ser utilizado, copiado ou modificado sem qualquer restrição (Campos, 2006).

1.4 Organização do Texto

O texto do trabalho está organizado em um único volume, consistindo em 6 capítulos, incluindo a introdução. No Capítulo 2 estão descritas as ferramentas e as linguagens de programação a serem utilizadas no desenvolvimento da aplicação. No Capítulo 3 é apresentada a ferramenta *Phalcon*, suas funcionalidades e algumas de suas principais classes e métodos. O Capítulo 4 consiste da descrição de um Sistema de Gerenciamento de conteúdo, tipo de ferramenta desenvolvida neste projeto. No Capítulo 5 é apresentado o sistema *pluton*, e suas características. E por último o Capitulo 6 que consiste da conclusão.

Capítulo 2

Ferramentas e Linguagens

Neste capítulo serão apresentadas as linguagens, ferramentas e *frameworks* utilizados para auxiliar no desenvolvimento do *software* proposto.

2.1 Orientação a Objetos

Segundo Dall'Oglio (2009) a Orientação a Objetos (OO) representa toda uma filosofia para construção de sistemas utilizando uma ótica mais próxima da realidade onde lidamos com objetos e representamos estruturas mais próximas do que conhecemos no mundo real, objetos estes que trabalham individualmente e colaboram entre si para construção de sistemas mais elaborados.

Atualmente existem diversas Linguagens orientadas a objetos, entre as quais podemos citar *Ruby*, *Phyton*, *Java*, *C#(C-Sharp)*, *C++*, *Javascript*, PHP, entre outras.

Para Galante et al. (2007), entre os conceitos básicos da orientação a objetos os que mais se destacam são:

- Abstração: a palavra abstrair, do português, significa extrair de um conteúdo o que ele tem de importante. Em OO, Farinelli (2011) diz que a Abstração é a capacidade de se modelar conceitos, entidades, elementos e características do mundo real considerando somente o que é relevante para resolver o problema em questão.

- Classes: são estruturas destinadas a descrever objetos utilizando atributos e métodos, sendo um modelo para criação de objetos semelhantes. É um dos principais conceitos da OO pois permite a reutilização efetiva de código (Dall'Oglio, 2009).

- Objetos: São instâncias de uma classe, variáveis que possuem atributos que mudam de acordo com os métodos da classe que executa. São utilizados para representar as classes e fazer com que as mesmas tenham sentido na aplicação.

- Herança é a possibilidade de uma classe utilizar métodos e atributos de outra. Uma classe pode herdar estes métodos e utiliza-los para modificar o estado de seus próprios

objetos, ou seja uma classe é criada a partir de uma já existente e herda seus atributos e comportamentos além de incluir os seus próprios.

- Polimorfismo: nos da a opção de controlar o comportamento de um objeto de diferentes maneiras dependendo da mensagem recebida. Uma classe pode conter subclasses onde cada uma implementa de acordo com suas necessidades um determinado método. Apesar de todas as subclasses possuírem o mesmo método, cada uma faz de maneira diferente.

Baseando-se nesses conceitos podemos entender que a OO agiliza o desenvolvimento e manutenção de sistemas unificando e tratando todos os processos do desenvolvimento sob uma única abordagem, proporcionando uma melhor reusabilidade de código e divisão de responsabilidades, além de facilitar a aprendizagem e a manutenção do código-fonte.

2.2 Padrão de Projeto MVC (*Model*, *View* e *Controller*)

É um modelo de três camadas (*Model*, *View* e *Controller*) Orientado a Objetos que dividem um aplicativo de modo que as funcionalidades e a apresentação dos dados fiquem em camadas separadas, normalmente a maior parte do código se encontra nas camadas *View* e *Control*. Foi desenvolvido entre 1978 e 1979 na *XEROX PARC* e mais tarde implementado na biblioteca de classe *Smaltalk-80* (Henrajani, 2007).

O MVC organiza o projeto de uma maneira que facilite a reusabilidade e manutenção do código. Sua estrutura básica pode ser visualizada na Figura 2.1 onde é possível identificar como os módulos interagem entre si.

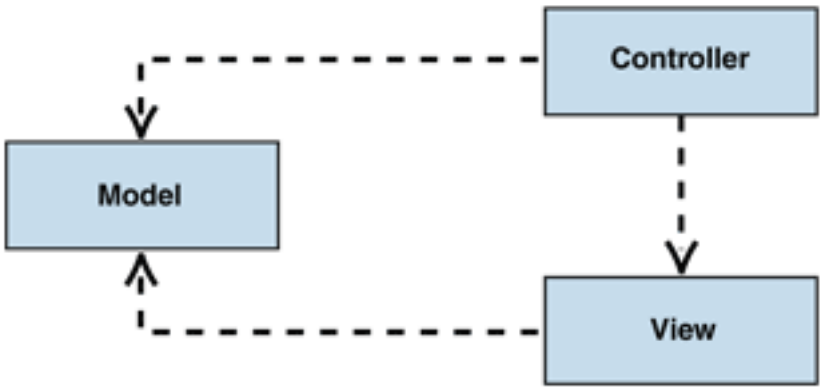


Figura 2.1: Modelo MVC.
Fonte: Brizeno (2012).

Os métodos do *model* podem ser acionados tanto pelo *controller* como pelos arquivos de visualização (*views*) e quando chamados retornam os dados solicitados. As *views* são

sempre chamadas por métodos do *controller* e, em alguns casos executam chamadas ao *model* solicitando informações para exibí-las na tela.

2.2.1 *Models*

O Modelo MVC era utilizado inicialmente para mapear as tarefas de entrada, processamento e saída para o modelo de interação com o usuário, onde o *model* representa os dados da aplicação, é ele quem tem contato com as informações armazenadas, sejam em um banco de dados, um arquivo XML ou qualquer outro lugar. É no *model* onde os dados são tratados (Gabardo, 2012).

O *model* consiste em classes e métodos para manipulação do banco de dados como consultas, inserções e exclusões de informações. Um exemplo de um *model* em PHP seria:

```
1  <?php
2
3  Class Usuarios {
4      private $login;
5      private $senha;
6
7      //Construtor
8      public static function __construct($login, $senha) {
9          $this->login = $login;
10         $this->senha = $senha;
11     }
12
13     //Método para verificar login do usuário
14     public function select() {
15
16         //lógica para verificar login e senha do usuário
17     }
18 }
```

Neste exemplo, a classe "Usuarios" é responsável por manipular os dados executando inserções, alterações, remoções e verificações dos dados referentes aos usuários.

2.2.2 *Views*

As *views* são responsáveis por exibir os dados e informações. É onde o conteúdo é renderizado para interagir com o usuário enviando suas ações, como envio de formulários ou solicitações e alterações de dados para o Controller (Gabardo, 2012).

Uma *view*, pelo padrão MVC, deve possuir a extensão ".phtml" indicando que é um arquivo HTML que pode possuir *scripts* PHP.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="UTF-8">
<title> <?php echo "TITULO"; ?> </title>
</head>
<body>
<?php echo "<h1>Olá Mundo!</h1>"; ?>
</body>
</html>
```

Este exemplo mostra uma página *web* em HTML utilizando *script* PHP para exibir informações para o usuário.

2.2.3 *Controllers*

É responsável por controlar todo o fluxo do programa, é quem define o comportamento da aplicação, interpretando as ações do usuário e mapeando as chamadas para os *Models* (Gabardo, 2012).

As ações do usuário são enviadas pela *View* para o *Controller* e que com base no processamento do *Model* executa a ação solicitada retornando uma reposta para o usuário.

```
<?php

Class SessionController {
  Public function EfetuaLoginAction(){
  // Código para efetuar login
  }
```

No exemplo acima temos um controlador responsável pelo gerenciamento de sessão e uma *action* responsável por efetuar o *login*. A *action* deve utilizar os métodos do *Model* para ter acesso aos dados e executar as ações necessárias conforme as solicitações do usuário.

2.3 Desenvolvimento *web*

No desenvolvimento de páginas *web* pode ser necessário a utilização de várias linguagens de programação, como por exemplo uma para trabalhar na máquina cliente, e uma outra na máquina servidor. Esses tipos de linguagens são descritos nesta seção.

2.3.1 HTML

HTML é a sigla em inglês para *Hyper Text Markup Language*, que, em português, significa linguagem para marcação de hipertexto.

Segundo Silva (2011), o conceito de hipertexto admite uma quantidade infinita de considerações e discussões, por isso, para nível de definição, resume hipertexto como todo

o conteúdo inserido em um documento para *web* que possui como principal característica a possibilidade de se interligar a outros documentos da *web*.

Atualmente o HTML se encontra na versão 5 (HTML5) e apesar de ter passado por diversas versões a W3C [1] considera oficialmente apenas as versões HTML 2.0, HTML 3.2, HTML 4.01 e a HTML5, as demais versões ou são anteriores à criação do W3C ou não foram lançadas oficialmente.

Em sua última versão, o HTML incorporou novos elementos e funcionalidades que possibilitam melhores experiências e mais simplicidade no desenvolvimento. Foi criada por um grupo de desenvolvedores de empresas como *Mozilla*, *Opera* e *Apple* fundado em 2004, chamado *What Working Group* (WHATWG), em um momento onde o W3C estava com suas atenções voltadas para a criação da segunda versão do XHTML (Anthes, 2012).

Em 2006 o W3C tomou conhecimento do trabalho da equipe do WHATWG e se uniu para auxiliar no desenvolvimento do HTML o que mais tarde causou o detrimento do XHTML 2.

Uma das maiores mudanças do HTML5 foi o fornecimento de novas ferramentas para trabalhar com *scripts client-side*, diminuindo a quantidade de código destes *scripts* e tornando a aplicação mais leve e funcional (Eis and Ferreira, 2012).

2.3.2 *Client-side script*

São responsáveis pelas operações executadas no cliente, ou seja, no navegador e sem qualquer contato com o servidor externo, um exemplo de linguagem client-side é o *javascript*. Estes *scripts* são preferencialmente utilizados para validação de entrada de dados sem utilizar recursos do servidor e por consequência não provocar trafego de dados na rede. Entre as validações mais comuns estão a validação de CPF, telefones, *e-mails* e senhas (Junior, 2006).

Códigos de linguagens *client-side* são inseridos em meio a *scripts* HTML com utilização de *tags* que indicam sua inserção.

Javascript

Javascript é uma linguagem que permite inserir lógica em páginas escritas em HTML. Os *scripts* podem estar "soltos" na página, sendo executados na sequência em que aparecem no arquivo de código-fonte, ou relacionados à ocorrência de eventos sendo ativados apenas quando um evento pré-determinado ocorre (Grillo and FORTES, 2008).

A utilização do *javascript* em páginas HTML pode ser de duas maneiras, no mesmo arquivo onde está o HTML entre as tags *<script type ='text/javascript'>* e *</script>* ou em um arquivo separado, chamado pelas mesmas tags através do parâmetro *"src="* que indica a localização do arquivo de *script*.

Um exemplo de um código fonte javascript é:

```
1 <script type= "text/javascript">
```

[1] Principal orgão de padronização da *World Wide Web* (http://www.w3.org/)

```
2   Document.write("Meu primeiro codigo javascript");
3 </script>
```

É executado no computador cliente e por conta disto seu código pode ser visualizado e até mesmo modificado através da utilização de algumas ferramentas. Por esse motivo não é utilizado para operações com banco de dados ou de segurança, para essas operações existem os *scripts Server-side*.

2.3.3 *Server-side scripts*

As linguagens *server-side* são processadas por um servidor *web*, que interpreta os dados e retorna o resultado para ser exibido no navegador, o que torna impossível a visualização do código da aplicação pelo cliente, por conta disso são utilizadas principalmente para consulta e manipulação de dados (Campos and de Souza Ribeiro, 2007).

PHP

A abreviação PHP vem de "*Hypertext PreProcessor*", que é uma linguagem de programação de código aberto muito utilizada para a criação de *scripts* que são executados no servidor *web* para a manipulação de páginas HTML (Alexandre and SANTOS, 2003).

Foi criada em 1994 por Rasmus Lerdorf. Inicialmente era apenas um código fonte em C utilizado para monitorar acessos ao seu currículo na internet. Rasmus nomeou esse conjunto de *script* como "*Personal Home Page Tools*" (PHP *Tools*). Algum tempo depois Rasmus reescreveu o PHP Tools adicionando novas funções permitindo conexão com banco de dados e o desenvolvimento de páginas *webs* simples, como um livro de visitas (Dall'Oglio, 2009).

O PHP evoluiu ao longo dos anos e hoje na sua atual versão (5.6) é uma das linguagens *web* mais utilizadas, possuindo cerca de 200.000 (duzentas mil) páginas ativas (PHP, 2014b).

É uma linguagem completa, simples e de fácil aprendizado. Tem performance e estabilidade excelentes, suporta conexões com os mais diversos banco de dados além de uma grande variedade de padrões e protocolos (Alexandre and SANTOS, 2003).

Uma vantagem importante do PHP é ser um software gratuito e de código-fonte aberto, que pode ser encontrado juntamente com sua documentação em seu site oficial (PHP, 2014a).

Outra característica do PHP é que ele é embutido no HTML, ou seja, HTML e PHP podem estar em um mesmo arquivo, podendo mesclar códigos PHP e HTML facilitando o desenvolvimento, como por exemplo, montar uma página *web* com HTML e exibir os dados retornados do servidor em PHP.

Para consultar e retornar destes dados é necessário um banco de dados onde estejam armazenados, PHP possui suporte a diversos bancos, como o *MySQL*, *PostgreSQL*, *Sybase*, *Oracle* e muitos outros. Cada banco suportado possui uma série de funções para executar diversas operações.

Além de suporte a diversos bancos também é multiplataforma, possuindo versões tanto para *Windows*, *Linux* ou *MacOS*.

cURL

Curl é uma ferramenta multiplataforma para transferência de dados de um servidor para outro através de diversos protocolos (*DICT*, *FILE*, *FTP*, *FTPS*, *GOPHER*, *HTTP*, *HTTPS*, *IMAP*, *IMAPS*, *LDAP*, *LDAPS*, *POP3*, *POP3S*, *RTMP*, *RTSP*, *SCP*, *SFTP*, *SMB*, *SMBS*, *SMTP*, *SMTPS*, *TELNET* e *TFTP*), além de suporte a *proxy*, autenticação de usuários, *upload FTP*, *HTTP Post*, entre outros (Stenberg (2012)).

Foi criada por Daniel Stenberg baseada no projeto do brasileiro Rafael Sagula, HttpGet. Inicialmente a ferramenta só atendia solicitações *Http* mas em sua segunda versão além da alteração do nome para *urlget* também ganhou suporte a *download FTP*. O projeto só passou a ter o nome *cURL* a partir da versão 4 do mesmo lançada em 1998 com suporte a SSL. Com o passar dos anos o projeto foi crescendo e ganhando suporte a novas ferramentas, atualmente está na versão 7.44.0[2].

O projeto também disponibiliza uma API em C para utilização da ferramenta, e diversas linguagens a utilizam para efetuar solicitações, entre elas o *PHP*. Um exemplo simples da utilização de cURL em *PHP* seria:

```
1 <?php
2   // Inicia o cURL acessando uma URL
3   $cURL = curl_init('/dados.txt');
4   // Define a opção que diz que você quer receber o resultado encontrado
5   curl_setopt($cURL, CURLOPT_RETURNTRANSFER, true);
6   // Executa a consulta, conectando-se ao site e salvando o resultado na
        variável $resultado
7   $resultado = curl_exec($cURL);
8   // Encerra a conexão com o site
9   curl_close($cURL);
```

No exemplo acima os dados de um arquivo são coletados e armazenados em uma variável, além de armazenar podemos também manipular e filtrar ou exibir os dados retornados.

```
1  <?php
2
3  $ch = curl_init();
4
5  // informar URL e outras funções ao CURL
6  curl_setopt($ch, CURLOPT_URL, "http://www.google.nl/");
7  curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
8
9  // Acessar a URL e retornar a saída
10 $output = curl_exec($ch);
11
```

[2] `http://curl.haxx.se/download.html`

```
12 // liberar
13 curl_close($ch);
14
15 // Substituir 'Google' por 'PHP Curl'
16 $output = str_replace('Google', 'PHPCurl', $output);
17
18 // Imprimir a saída
19 echo $output;
```

Neste exemplo, os dados retornados são modificados e depois exibidos na tela com as modificações executadas.

Capítulo 3

Framework Phalcon

Phalcon foi desenvolvido em 2011 por Andres Guiterrez, Eduardo Carvajal, Nikolaos Domopulos e Nokolay Kirsh e hoje possui centenas de desenvolvedores no serviço de *Web Hosting github* [1].

É um *framework full stack* [2] de código-fonte livre para desenvolvimento PHP criado como uma extensão C e otimizado para possuir um alto desempenho. Também possui baixo acoplamento [3] e é totalmente MVC permitindo uma utilização parcial ou total de suas funcionalidades conforme as necessidades do programador (Miller, 2014).

As extensões são funcionalidades extras para a linguagem de programação alvo, que auxiliam no desenvolvimento de aplicações. O PHP possui cerca de cento e cinquenta extensões [4] que vem em seu pacote de código fonte oficial, tendo em média duas mil e quinhentas funções provenientes destas .

Uma extensão C é compilada e carregada juntamente com o PHP quando o servidor *web* é iniciado, disponibilizando suas classes e funções para serem utilizadas em qualquer aplicação no servidor onde está instalado, pois por ser desenvolvido em uma linguagem compilada seu código fonte não é interpretado (Phalcon, 2014b).

Mesmo *phalcon* sendo completamente desenvolvido em C, não é necessário nenhum conhecimento desta linguagem de programação para utiliza-lo, pois suas funcionalidades são disponibilizadas como classes e métodos do PHP. Por exemplo para configurar uma conexão com um banco de dados utilizando *phalcon* basta utilizar o seguinte código(Phalcon, 2014b):

[1] https://github.com/phalcon

[2] Possui funcionalidades para resolver os principais problemas de uma aplicação *web*, não sendo limitado as funcionalidades básicas.

[3] Acoplamento é o grau de dependência entre dois artefatos (classe, método, componente, tabela ou documento) do projeto. Quanto maior a dependência maior é o grau de acoplamento(Doederlein, 2012).

[4] http://br1.php.net/manual/en/extensions.alphabetical.php

```php
<?php
//Configurando conexão com um banco de dados MySQL
//Criando a DI (Dependency Injection)
$di = new PhalconnDInFactoryDefault( );
$di->set('db', function(){
    return new Phalcon\Db\Adapter\Pdo\Mysql(array(
        'host' => 'localhost',
        'username' => 'root',
        'password' => '',
        'dbname' => 'tutorial'
    ));
});
```

Também possui *Tags* para gerar *scripts* HTML através do PHP de forma menos complicada, como por exemplo (Phalcon, 2014b):

```php
<?php
    echo "<h1>Olá Mundo!</h1>";
    echo Phalcon\Tag::linkTo("cadastro", "Cadastre-se aqui");
```

No código visto acima é utilizado método *linkTo* para criar um *link* para um novo método chamado cadastro, o HTML resultante do *script* acima seria (Phalcon, 2014b):

```html
<h1>Ola Mundo!</h1>
<a href="cadastro">Cadastre-se Aqui</a>
```

Phalcon possui suporte a diversos bancos de dados como *MySQL*, *PostgresSQL* e até a bancos de dados Orientados a Documentos como o *MongoDB*, além de vários servidores *webs* entre eles *Apache*, *Nginx* e o *Cherokee*. Também é multiplataforma possuindo versões para *Linux*, *Windows* ou *MacOS*.

Ou seja, ao utilizar *phalcon* não é necessário utilizar tecnologias ou sistemas operacionais específicos, podendo utilizar-se dentre os mais comuns e mais utilizados servidores, bancos e sistemas operacionais.

A premissa de *Phalcon* é a velocidade, pois como foi dito anteriormente, ele é completamente desenvolvido em C, e ao contrário dos *frameworks* em PHP ou qualquer outra linguagem interpretada, seu código fonte é compilado apenas uma vez, ficando disponível para ser utilizado sem a necessidade de interpretar o código a cada chamada de suas funções, aumentando drasticamente sua velocidade (Miller, 2014).

Em testes realizados pela equipe de desenvolvimento, *Phalcon* se mostrou até 8 vezes mais rápido que os demais frameworks testados, entre eles *CodeIgniter*, *Symfony* e *Zend*. Os testes foram executados em um computador com Processador *Intel Core i5*, 2.04Ghz, Memória de 4GB 1333Mhz DDR3, Sistema Operacional *Mac OS X Lion* 10.7.4 utilizando o servidor *web Apache* 2.2.22 e PHP versão 5.3.15 (Phalcon, 2014a).

Um dos testes executados foi o *Benchmark Hello Word* que é utilizado para encontrar a menor carga de um *framework* identificando o tempo gasto na execução de uma tarefa

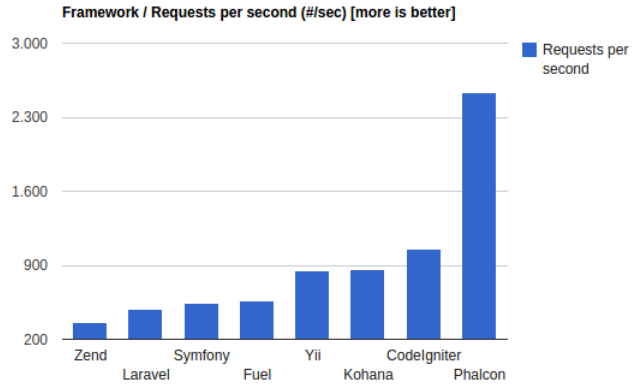


Figura 3.1: Teste velocidade *frameworks* - Requisições por segundo.
Fonte: Phalcon (2014a).

simples, medindo o tempo necessário para iniciar uma tarefa, executar uma ação e liberar os recursos no final da requisição.

Uma classe "*say*" e um método "*hello*" foram criados para cada *framework*, também uma *view* para exibir a mensagem. Com a ferramenta ab[5] foram enviadas 2000(duas mil) requisições por 10(dez) conexões concorrentes para cada *framework*.

Na Figura 3.1 é exibido quantas requisições por segundo cada *framework* foi capaz de executar, *phalcon* aceitou mais de 2500 (duas mil e quinhentas) requisições, mais que o dobro do segundo colocado. Já na Figura 3.2 podemos ver o tempo médio levado para executar todas as requisições simultâneas, novamente *phalcon* se mostra mais de duas vezes mais rápido que o segundo colocado.

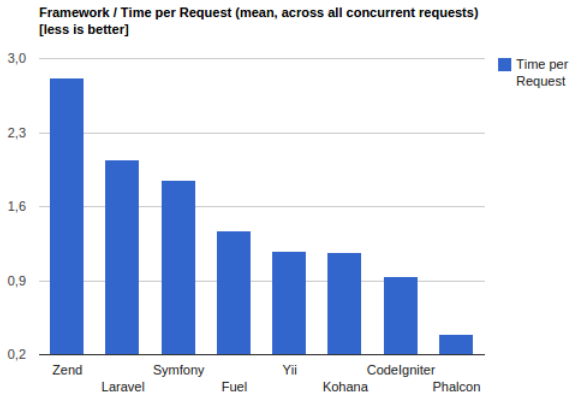


Figura 3.2: Teste velocidade *frameworks* - Tempo de execução para cada requisição
Fonte: Phalcon (2014a).

[5]Módulo do *apache* para calcular quantidade de pedidos por segundo o servidor é capaz de executar

Existem outros *frameworks* desenvolvidos em linguagens compiladas, como por exemplo o *Yaf*, porém não são *full-stack*, ou seja possuem apenas funcionalidades básicas, o que exige ou a utilização de outros *frameworks*, ou da propria linguagem PHP, o que em ambos os casos foge da proposta de *frameworks* compilados, que é a velocidade.

3.1 Classes e métodos

Phalcon possui uma enorme gama de classes para auxiliar no desenvolvimento de aplicações, dentre quais algumas merecem destaque:

3.1.1 *Phalcon\Di\FactoryDefault*

Dependency Injection é um padrão de projetos que visa diminuir dependências entre as classes do projeto facilitando a troca de dependências em tempo de execução (Lockhart (2015)).

A classe *FactoryDefault* é uma variante da interface responsável pela injeção de dependência *Phalcon\DiInterface*, ela fornece um pacote completo instanciando todas as dependências do *framework* acessíveis em qualquer parte do sistema, assim o desenvolvedor não precisa registrar serviços manualmente e o *framework* é disponibilizado de forma completa. Para utilizá-la basta inserir no arquivo de inicialização do projeto o seguinte código:

```
1 <?php
2
3 class Application extends \Phalcon\Mvc\Application {
4
5     protected function _registerServices() {
6         $di = new \Phalcon\DI\FactoryDefault();
7     }
8 }
```

Porém em pequenas aplicações pode não ser viável instanciar todos os recursos do *framework* e sim somente os que serão utilizados. Para tal utiliza-se a classe *Phalcon\Di* para inicializar individualmente cada serviço:

```
1 <?php
2
3 use Phalcon\Http\Request;
4
5 $di = new Phalcon\Di;
6
7 $di->set("request", function () {
8     return new Request();
9 });
```

No exemplo acima é instanciada a dependência da classe *Phacon\Http\Request*, que agora pode ser acessada em qualquer parte da aplicação:

```
1 <?php
2
3 $this->request;
```

Desta forma é possível ter uma aplicação mais leve instanciando somente os recursos realmente utilizados.

3.1.2 *Phalcon\Mvc\Router*

O componente *router* é utilizado para mapear o acesso a controladores. Um *router* analisa uma *URI*(Identificador Uniforme de Recursos)[6] para determinar qual o caminho a ser acessado.

Phalcon\Mvc\Router fornece recursos avançados de roteamento, no modelo *MVC* podem ser definidas rotas para *controllers/actions*. Um exemplo seria:

```
1  <?php
2
3  use Phalcon\Mvc\Router;
4
5
6  $router = new Router();
7
8
9  $router->add(
10     "/settings",
11     array(
12         "controller" => "settings",
13         "action"     => "index"
14     )
15 );
```

Este método define uma rota para o controlador *settings* e a *action index*, ou seja, ao acessar pelo navegador *localhost/settings* é redirecionado para *localhost/settings/index*. Um objeto *Router* não executa um controlador ou método, apenas recolhe a informação para que uma outra classe, *Phalcon\Mvc\Dispatcher*[7], redirecione a navegação.

Em uma aplicação podem existir diversas rotas e indicar uma a uma pode ser trabalhoso, para tal, pode-se utilizar uma maneira genérica para criar diversas rotas de uma só vez:

```
1 <?php
2
3 use Phalcon\Mvc\Router;
```

[6]Cadeia de caracteres utilizada para identificar ou denominar recursos na *internet* podendo ser desde um site até uma imagem (Mealling and Denenberg (2002)).

[7] Classe responsável por pegar o objeto do pedido, extrair nome do módulo, controlador, *action* e parâmetros adicionais contidos nele e em seguida instanciar o controlador e chamar a *action* extraída. Miller (2014)

```
4
5  $router = new Router();
6
7  $router->add(
8      "/admin/:controller/a/:action/:params",
9      array(
10         "controller" => 1,
11         "action"     => 2,
12         "params"     => 3
13     )
14 );
```

Dessa maneira ao acessar *admin/login/index* o usuário será redirecionado para o controlador *login* e método *index*, ou pode acessar qualquer outra rota existente no sistema.

3.1.3 *Phalcon\Mvc\Controller*

Controladores devem fornecer métodos (*actions*) que lidam com pedidos das *views*. Em *phalcon* por padrão, todos os métodos públicos de um controlador são acessiveis via URI. Por exemplo, ao acessar a URI *http://localhost/pluton/settings/index* o seguinte caminho é executado:

- *pluton*: Diretório do projeto
- *settings*: Controlador
- *index*: Método

Qualquer coisa informada após o método é interpretada como parâmetro. Digamos que tenha uma requisição *http://localhost/pluton/post/index/2015/Primeira-postagem* seja efetuada. "2015" e "Primeira-postagem" são enviados como parâmetros para o método *index*.

Em *Phalcon* os controladores devem possuir o sufixo *Controller* e os métodos o sufixo *Actions*. Um exemplo prático seria:

```
1 <?php
2
3 class PostController extends Phalcon\Mvc\Controller{
4
5     public function indexAction($year = NULL, $post = NULL){
6
7     }
8 }
```

Alguns métodos são responsáveis pelo carregamento de arquivos de visualização, quando este é o caso os arquivos são carregados do diretório padrão indicado no projeto, localizando uma pasta com o nome do controlador e um arquivo com o nome do método.

No exemplo acima o método *indexAction* irá buscar o arquivo *index* na pasta *views\base* e a exibe no navegador.

Além de carregar arquivos de visualização, os métodos também podem acessar dados retornados pelo usuário via *HTTP*, para tal a classe *Phalcon\Mvc\Request* é utilizada:

```
1  <?php
2
3  class PostController extends Phalcon\Mvc\Controller{
4
5      public function baseAction(){
6          if($this->request->isPost){
7              $id = $this->request->getPost("id");
8          }
9      }
10
11     }
12 }
```

Desta forma evita-se acessar a váriavel global *$_POST* diretamente proporcionando maior segurança. Da mesma forma existem métodos para acessar as variáveis *$_GET* e *$_REQUEST*: *$this->request->get()*.

Ainda utilizando essas duas ações, é possível validar o tipo ou o valor dos dados recebidos informando-o a frente do parâmetro:

```
1  <?php
2
3  $this->request->getPost("id", "int");
4
5  $this->request->get("email", "string");
6
7  //Informando um valor default
8  $this->request->getPost("id", "int", 150);
```

Outra classe importante é a Phalcon\Http\Response, com ela é possível redirecionar solicitações para outros controladores ou métodos dependendo da necessidade do programador. Por exemplo, podemos verificar se um usuário efetuou login e caso não tenha efetuado redirecionamos ele para tal página:

```
1  <?php
2
3  class IndexController extends Phalcon\Mvc\Controller{
4
5      public function index() {
6
7          //Inicia a sessão
8          $this->session->start();
9          //Caso a váriavel id não esteja instanciada na sessão redireciona para
               a tela de login
```

```
10          if (empty($this->session->get("id"))) $this->response->request("login/
                index");
11      }
12  }
```

Também é possivel informar erros através do método *setStatusCode*:

```
1  <?php
2
3  $this->response->setStatusCode(404, "Not Found");
```

3.1.4 *Phalcon\Mvc\Model*

Como visto no capítulo anterior, modelos ou *Models* são classes que representam os dados e as regras para manipulação dos mesmos pelo sistema. Em *phalcon* cada modelo é uma representação de uma tabela no banco de dados, sendo assim um modelo *Users* representa a tabela *users*.

Phalcon\Mvc\Model é a base para todos os modelos, ela fornece independência total do banco de dados, capacidades de busca avançada além da relação entre modelos(Miller (2014)).

Uma classe *Models* deve ser criada dentro do diretório padrão de modelos determinada no projeto e estender a classe *Phalcon\Mvc\Model*:

```
1  <?php
2
3  class Users extends Phalcon\Mvc\Model{
4
5
6  }
```

Os principais métodos utilizados aqui são *find()*, *findFirst()*, suas extensões *findBy* e *findFirstBy()* além do método *save()*. *find()* quando utilizado sem nenhum paramêtro retorna todos os dados de uma determinada tabela, assim como o *findFirst* retorna o primeiro dado, para filtrar uma busca é possível utilizar alguns parâmetros:

- *conditions*: Comparando com *sql*, seria como o *where* do método, um exemplo seria "*id = :id:*"
- *order*: define a ordem do retorno dos dados. Algo como "*date_register DESC*" retorna os dados ordenados pela data de registro de maneira decrescente.
- *limit*: Valor inteiro que limita a quantidade de dados retornados.
- *offset*: Indica a partir de qual registro os dados devem ser retornados, por exemplo, se for informado o valor 20, retorna todos os dados a partir do vigésimo registro da tabela.

- *bind*: Associado ao *conditions*, seta o valor para o parâmetro informado entre ':'.

Exemplificando:

```php
<?php


//Sem informar qualquer parametro
Users::find();
Users::findFirst();

//Filtrando a consulta
//Retorna todos os usuarios do tipo 1 a partir do décimo registro
Users::find(array(
    "conditions" => "type :id:",
    "order" => "date_register DESC",
    "limit" => 10,
    "offset" => 10,
    "bind" => array("type" => 1)
));

//Retorna o usuario com id igual a 1
Users::findFirst(array(
    "conditions" => "id = :id:",
    "bind" => array("id" => 1)
));

//Retorna todos os usuários do tipo 1
Users::findByType(1);

//Retorna o primeiro usuário do tipo 1
Users::findFirstByType(1);
```

Para salvar dados existe o método *save()*, ele pode ser utilizado tanto para inserir novos registros ou atualizar registros existentes, sua ação depende da maneira que o modelo é iniciado. Se iniciar um novo modelo, setar valores para os campos da tabela e executar o método *save*, ele irá inserir um novo registro no banco, porém se for executada uma consulta utilizando *findFirst* ou *findFirstBy*, atualizar os registros e executar o *save()* os dados que foram retornados pela consulta serão atualizados:

```php
<?php

//Cria um novo usuario
$users = new Users();

$users->name = "Fulano";
$user->email = "fulano@email.com";
$user->tipo = 2;
$user->save();
```

```
10
11 //Atualiza o nome do primeiro usuario do tipo 2
12 $user = Users::findFirstByTipo(2);
13 $user->name = "Fulano da Silva";
14 $user->save();
```

3.1.5 *Phalcon\Mvc\View*

Essa classe é responsável pela manipulação das *views* da aplicação. Como visto anteriormente, *phalcon* exibe de forma automática a *view* relacionada ao controlador e método executados buscando-a na pasta definida.

É possível desabilitar *views*, exibir uma *view* não relacionada ao método executado, enviar valores entre outros métodos vistos a seguir:

- *getContent()*: Insere o conteúdo da *view* de menor hierarquia na *view* atual quando chamado.
- *setVar()*: Envia uma única variável para a *view*.
- *setVars()*: Envia uma ou mais variáveis para a *view*.
- *render()*: Renderiza a *view* informada.
- *pick()*: Renderiza *views* de outros controladores;
- *partial()*: Renderiza uma *view* como parte de outra já carregada, normalmente executado dentro de outra *view*.
- *disable()*: Desabilita a renderização de uma *view* para o método atual.

Exemplos de utilização:

```
1  <?php
2  //Envia uma variavel para a view
3  $this->view->setVar($post_content);
4  $this->view->render("post", "index");
5
6
7  //Envia um array para a view
8  $vars = array("id" => 1, "content" => "text text text text");
9  $this->view->setVars($vars);
10 $this->view->render();
11
12
13 //Renderiza uma view diferente da ação executada;
14 class Index{
15     public function main(){
```

```
16          $this->view->pick("index/index");
17      }
18  }
19
20
21  //Carrega um cabeçalho compartilhado
22  <div class="header"><?php $this->partial("shared/header"); ?></div>
```

As *views* seguem uma hierarquia ao serem carregadas. Após a execução do controlador, é executada uma chamada a pasta de arquivos de visualização, caso exista um arquivo "*index.phtml*" dentro da pasta ele é carregado, em seguida uma nova chamada é feita, agora para a pasta padrão *layout* e um arquivo com o nome do último controlador executado é carregado, e por último é carregado o arquivo de *layout* do último método executado, este arquivo deve estar dentro da pasta de *views* padrão e possuir o mesmo nome do método.

```
1  <?php
2
3  class PostController extends \Phalcon\Mvc\Controller{
4
5      public function index(){
6
7      }
8  }
```

```
1  <!-- app/views/index.phtml -->
2  <html>
3      <head>
4          <title>Exemplo</title>
5      </head>
6      <body>
7
8          <h1>Layout Principal</h1>
9
10         <?php echo $this->getContent(); ?>
11
12     </body>
13 </html>
```

```
1  <!-- app/views/layouts/posts.phtml -->
2
3  <h2>Layout do Controlador Post</h2>
4
5  <?php echo $this->getContent(); ?>
```

```
1  <!-- app/views/posts/show.phtml -->
2
3  <h3>Esta é a view index</h3>
```

A ordem de saída ao efetuar uma chamada a *action index* do controlador *Post* pode ser vista na Figura 3.3 seguindo os arquivos acima seria:

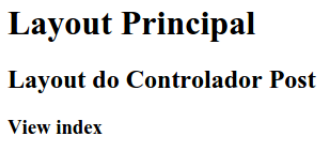


Figura 3.3: Exemplo de renderização de view

Utilizando as classes e métodos listados neste capítulo é possível desenvolver um projeto utilizando as ferramentas básicas fornecidas por *Phalcon* de maneira completa e organizada.

Capítulo 4

Sistema de gerenciamento de conteúdo (SGC)

Inicialmente os dados existentes em páginas *web* eram limitados a *intranet* de empresas, onde existiam paginas estáticas gerenciadas por um *webmaster*, o qual realizava alterações manuais sempre que necessário (Pereira and Bax, 2010).

Porém, com o aumento do acesso a *internet*, milhões de páginas *webs*, empresas com *sites* de *e-commerce*, divulgação de produtos, notícias entre outros diversos tipos de conteúdos, essa forma de administrar *sites* não conseguia mais atender a demanda de informações, para resolver com essa situação a gerência de conteúdo precisou evoluir a um novo patamar.

A idéia de um SGC surgiu na década de 90 com a finalidade de melhorar o gerenciamento de conteúdos em *websites* de organizações. Desde então diversos SGCs foram criados e muitos deles com código livre e gratuito para a utilização. Entre os mais utilizados atualmente estão o *WordPress* e o *Joomla* [1].

Um SGC deve basicamente auxiliar na divulgação e gerenciamento de dados e informações, dispensando a necessidade de conhecimento técnico em linguagens de programação para gerar, armazenar e disponibilizar qualquer tipo de dados na *internet*.

Existem atualmente diversos SGC para as mais diversas funcionalidades, *e-commerce*, *blogs*, gerenciadores de fóruns, sistemas de educação a distância e alguns que somam todas essas opções. Outros possuem uma funcionalidade bastante comum atualmente, a inserção de *plugins* (Raven, 2010).

Segundo Pereira and Bax (2010) um SGC pode ser dividido em três áreas, que são: criação de conteúdos, entrega do conteúdo para o usuário e recuperação da informação.

A criação de conteúdo como o nome diz, se refere a produção do conteúdo que será disponibilizado no *site* à ser gerenciado pelo SGC. A entrega do conteúdo para o usuário é a maneira de como o conteúdo criado chegará aos clientes do *site*, e a recuperação da informação diz respeito ao armazenamento das informações publicadas, ou seja ao banco de dados.

[1] http://w3techs.com

Chagas et al. (2008) diz que um SGC deve possuir as seguintes funcionalidades básicas:

- esquema de segurança baseadas em papéis: Modelo onde cada usuário possui permissões de acordo com a atividade que desempenha no sistema;
- sindicalização do conteúdo: possibilitar de alguma maneira o compartilhamento de informações entre diversos *websites*;
- indexação: anexar ou referenciar conteúdos externos ao *website* onde o conteúdo está disponibilizado;
- busca: Possibilidade de buscar através de palavras chaves, ou outros métodos, conteúdos existentes na página;
- *workflow*: Fluxos de trabalho para gerenciar a publicação de conteúdos, utilizando regras para edição, aprovação, publicação e exclusão.

Capítulo 5

Pluton

O aplicativo *pluton* foi desenvolvido para que um usuário administre um *blog* com o máximo de abstração possível, de forma que não seja necessário conhecimento avançado de desenvolvimento e manutenção de páginas *web* e ainda assim possuir um grande controle sobre a aplicação.

Para o desenvolvimento foram utilizadas as linguagens PHP e *Javascript*, além da linguagem de marcação HTML, o *framework phalcon* para a programação em PHP e a biblioteca *JQuery*[1] para auxiliar no desenvolvimento com *javascript*. O Banco de dados utilizado pela aplicação é o *MySQL*[2] e como servidor *web* o *Apache*[3]. A Aplicação foi desenvolvida para trabalhar com o sistema operacional *ubuntu* 14.04 LTS. Para auxiliar no desenvolvimento foram utilizados os seguintes *plugins*/*APIs*/Ferramentas:

- *Dbeaver*: Ferramenta para administração de banco de dados gratuita e de código fonte livre sob a licença GPL2 [4].
  - Disponível em `http://dbeaver.jkiss.org/`
- *Sublime Text* 3: Editor de texto para diversas linguagens de programação, entre elas *PHP* e *javascript* de alto desempenho, além de possuir diversos *plugins* que auxiliam na criação de códigos fonte.
  - Disponível em `http://www.sublimetext.com/3`.
- *Xdebug*: Extensão para *PHP* que permite a depuração da aplicação em tempo de execução. Auxilia no desenvolvimento exibindo a execução do projeto linha a linha e exibindo erros e/ou *warnings*.
  - Diponível em `http://xdebug.org/`.

[1] biblioteca para *javascript* focada na simplicidade e facilidade no desenvolvimento de *scripts client-side*.
[2] Sistema de Gerenciamento de Banco de Dados (SGBD) que utiliza a linguagem *Structured Query Language* (SQL) como interface.
[3] Servidor *web* de código-fonte livre e compatível com protocolo *Hypertext Transfer Protocol* (HTTP).
[4] Disponível em `http://www.gnu.org/licenses/old-licenses/gpl-2.0.html`

- gliffy Diagrams: Ferramenta online para criação de diagramas.
  - Disponível em `https://www.gliffy.com/`.
- *Google APIs Client Library for PHP*: Biblioteca disponibilizada pela *Google* para utilizar suas ferramentas com a liguagem PHP. Atualmente em versão *beta*.
  - Disponível em `https://developers.google.com/api-client-library/php/?hl=pt-BR`
- *Bootstrap*: *Framework frontend* que auxilia no desenvolvimento de páginas *webs* responsivas. Também possui diversos plugins em *jQuery* que auxilia na criação de diversos componentes.
  - Disponível em: `http://getbootstrap.com/`
- *SweetAlert*: Plugin para *javascript* utilizando *bootstrap* para exibir mensagens de alerta.
  - Disponível em: `http://t4t5.github.io/sweetalert/`
- *chartJS*: Plugin para *javascript* baseado em *HTML5* para exibição de dados em gráfico de forma simplificada.
  - Disponível em: `http://www.chartjs.org/`
- *TinyMCE*: Editor *WYSIWYG*[5] utilizado para edição e publicação de conteúdos *online*.
  - Disponível em: `http://www.tinymce.com/`
- *Raptor*: Editor *WYSIWYG* projetado para edição *in-line*[6] baseado em *HTML5*.
  - Disponível em: `https://www.raptor-editor.com`
- *phpdox*: Ferramenta para geração de documentação automática para *PHP*.
  - Disponível em: `http://www.phpdox.de`
- *Swiftmailer*: Ferramenta que integra em qualquer aplicativo *PHP* fornecendo funcionalidade de enviar *emails*.
  - Disponível em: `http://swiftmailer.org/`

[5]Editor que permite ao usuário visualizar algo similar ao resultado final enquanto cria/edita o documento.
[6]Edição imediata, permite editar o conteúdo diretamente na página *web*.

5.1 Descrição dos Requisitos

5.1.1 Requisitos Funcionais

- RF1: O sistema deve ser configurado automaticamente no primeiro acesso.
- RF2: O sistema deve permitir o cadastro de novas postagens.
- RF3: O sistema deve permitir o cadastro de novos usuários.
- RF4: O sistema deve permitir a desativação ou ativação de usuários.
- RF5: O sistema deve permitir a alteração de informação de usuários já cadastrados.
- RF6: O sistema deve permitir a alteração de postagens já cadastradas.
- RF7: O sistema deve permitir integração com Ferramentas *Google*.
- RF8: O sistema deve permitir integração com Redes Sociais (*Facebook* e *Twitter*).
- RF9: O sistema deve permitir a edição de *layout* do *blog*.
- RF10: O sistema deve permitir a adição de novas funcionalidades (*plugins*).

5.1.2 Requisitos Não Funcionais

- RNF1: O sistema deve ser divido em dois módulos *backend* (Área administrativa) e *frontend* (Blog).
- RNF2: O banco de dados deve ser criado e configurado durante a configuração inicial do sistema.
- RNF3: Os dados de usuários devem ser validados antes de efetuar qualquer alteração de informação dos mesmos.
- RNF4: É preciso validar as informações da postagens antes de cria-las.
- RNF5: Usuários com publicações criadas e postagens não podem ser excluídos do banco de dados, apenas desativados(usuários) ou não exibidos no *blog*(postagens).
- RNF6: O *blog* deve exibir somente postagens com o *status* Publicado e em ordem de data de publicação.
- RNF7: Os comentários do blog devem ser efetuados via *Facebook*.
- RNF8: As permissões de acesso de usuários devem seguir as seguintes regras:

- Super-Administrador: Só pode existir um usuário com permissões de super administrador, ele é o criador do *blog*. O super administrador tem acesso a todos os módulos do sistema, desde a modificação de *layout*, até a publicação de conteúdos. Somente o super administrador pode executar alterações que impliquem visualmente no sistema, e somente ele tem permissão para adicionar novos administradores. Também é permitido somente ao super administrador a instalação de plugins.
- Administrador: O usuário administrador possui permissão para gerenciar a postagem de conteúdo e adicionar e excluir usuários (exceto administradores).
- Editor: Um editor pode visualizar e reger as suas postagens e a dos demais usuários. Possui permissão para aprovar editar ou excluir qualquer conteúdo, desde que essas não tenham sido aprovadas por outros editores ou administradores.
- Autor: Um usuário com permissões de autor pode visualizar, gerenciar e publicar apenas as suas próprias postagens, não tendo acesso a qualquer conteúdo publicado por outros usuários.
- Colaborador: O usuário do tipo colaborador pode apenas escrever e gerenciar suas postagens não tendo permissão para publicá-las. A publicação pode ser feita somente por um editor ou um administrador.

5.2 Utilizando o sistema

5.2.1 Primeiro Acesso

Em um primeiro acesso é solicitado ao usuário que efetue a configuração da aplicação, então ao acessar pelo navegador o endereço *localhost/pluton/admin* é redirecionado para a página inicial com informações sobre a instalação (Fig. 5.1). Aqui é dada uma breve explicação sobre o que é necessário para continuar com a instalação do sistema. O processo de configuração e instalação do *pluton* são efetuados pela classe *SetupController*(Fig. 5.2).

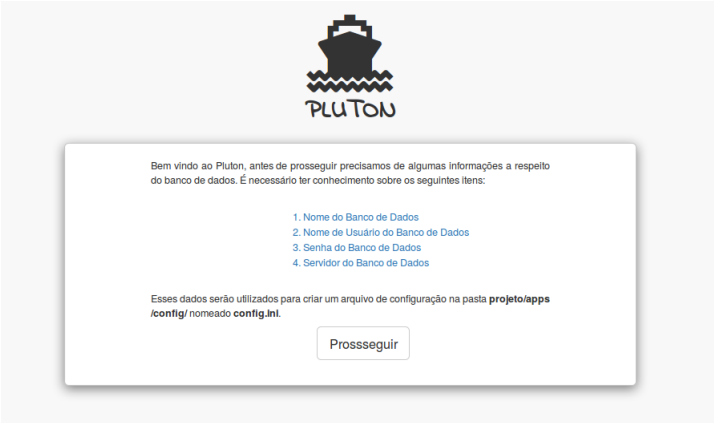


Figura 5.1: Página Inicial da Instalação do Aplicativo.

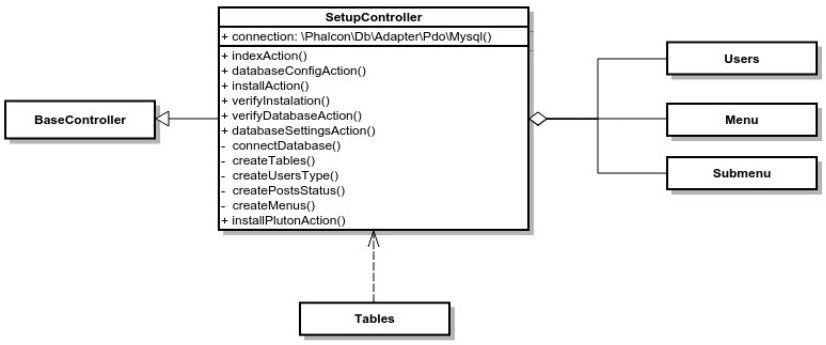


Figura 5.2: Diagrama Classe SetupController

Seguindo para o próximo passo, um formulário (fig 5.3) solicita as informações do banco de dados, nome de usuário e senha de acesso ao banco, além do endereço do servidor. Inserindo os dados e clicando em prosseguir as informações são enviadas para o método *dataBaseSettingsAction*() onde são configuradas as informações para conexão com o banco, efetuada a conexão e então criada as tabelas necessárias para o funcionamento do sistema.

Figura 5.3: Formulário de cadastro do banco de dados.

Com o banco de dados configurado, o próximo passo é inserir as informações sobre o *blog*. Neste formulário (Fig. 5.4) devem ser inseridos o nome do *blog* e os dados do usuário super-administrador, clicando novamente em prosseguir o método *installPlutonAction*() é chamado e efetua a inserção das informações necessárias para o sistema no banco de dados, como status de postagens, tipos de usuários e menus do sistema, além das informações do *blog* e de usuário informadas no formulário. Com o banco de dados configurado e o sistema instalado já é possível acessar o ambiente de administração.

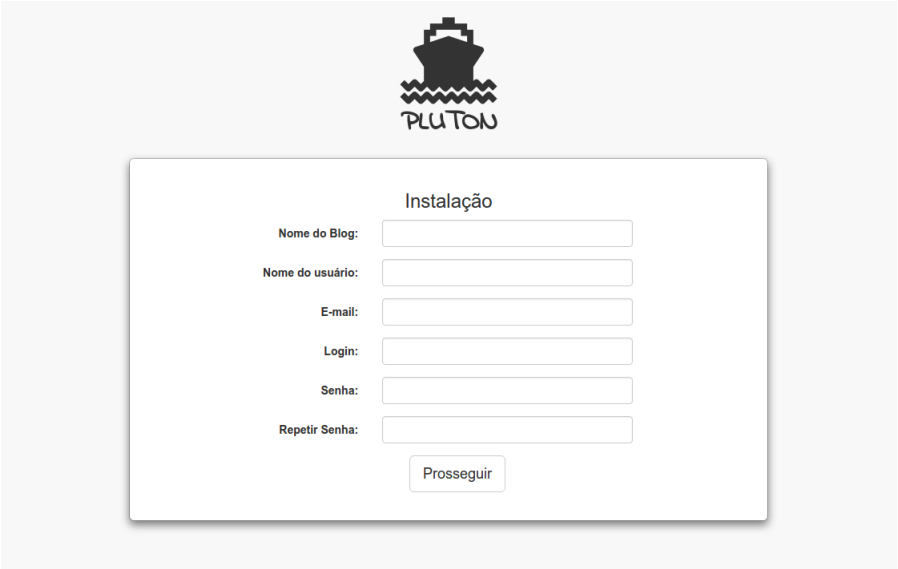


Figura 5.4: Tela Final de Instalação da aplicação.

Área Administrativa

Após finalizar a configuração do *Pluton* o acesso ao sistema fica disponível. Acessando novamente pelo navegador *localhost/pluton/admin* se tudo estiver correto o usuário é redirecionado para a tela de *login* (Fig. 5.5), inserindo usuário/*e-mail* e senha e clicando em "Entrar", o método *loginAction*() da classe *LoginController* (fig 5.6) é executado, ele valida os dados informados e caso estejam corretos, redireciona o usuário para a *dashboard* do sistema (Fig. 5.7).

Figura 5.5: Tela de Login.

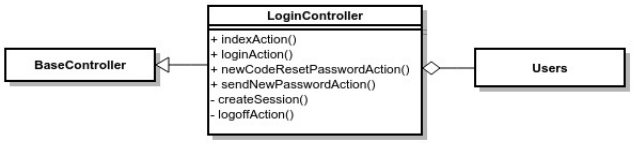


Figura 5.6: Diagrama classe *LoginController*.

A tela principal vista na Figura 5.7 possui um menu lateral com opções para o usuário, um gráfico central com os dados de acesso e postagens (mensais) do blog criado, além de cartões informando total de acessos únicos, quantidade de postagens, quantidades de curtidas na rede social *facebook* e quantidade de seguidores na rede social *twitter*. Todas essas informações são carregadas pelo método *indexAction*() da classe *DashboardController* (Fig. 5.8). Por padrão somente os dados de postagens são exibidos, para exibição de acessos ou dados de redes sociais são necessárias algumas configurações.

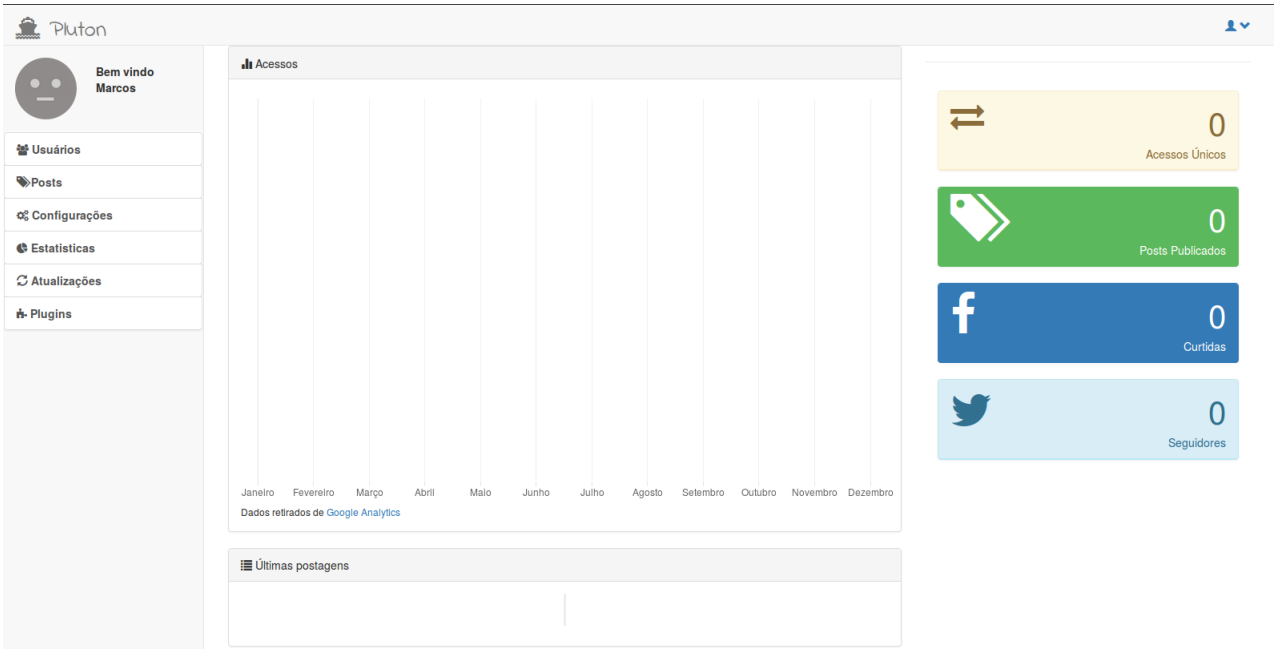


Figura 5.7: *Dashboard*

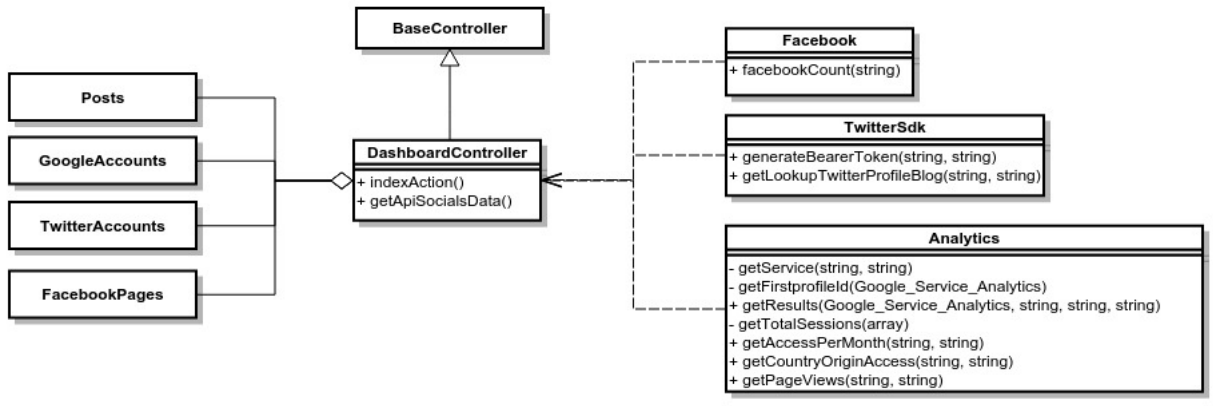


Figura 5.8: Diagrama Classe *DashboardController*

Configurações

A classe *SettingsController*(fig 5.9) gerencia as configurações do sistema. Através dela é possível definir o nome do *blog*, *url*, envio de *e-mails*, além de configurar as ferramentas do *Google* e redes sociais. A seguir é exibido cada uma dessas funcionalidades.

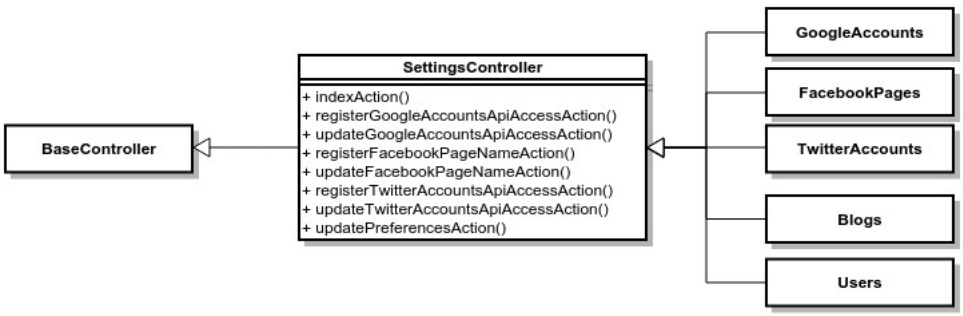


Figura 5.9: Diagrama Classe *SettingsController*

Ao clicar no menu configurações é exibida uma tela com várias abas para configurar algumas funcionalidades de utilização do *pluton*. A primeira aba é a "Preferências" (Fig. 5.10). Aqui é possível alterar o nome do *blog*, informar a *url* do mesmo, além de inserir uma breve descrição que é exibida na tela "Sobre" do *blog*. Descrição esta que pode conter conteúdos HTML.

Ainda nesta tela há a opção de configurar um *email* para uso administrativo, como por exemplo envio de novas senhas, e receber mensagens de leitores do *blog*. Aqui o sistema só aceita endereços de *email* do *gmail* pois o servidor *STP*[7] já é pré-configurado para tal.

No canto inferior direito há um botão "Editar Aparência do Blog", clicando nele é possível fazer algumas edições visuais no blog, uma aba do navegador é aberta com a tela do blog, e ao passar o mouse por cima de alguns campos é exibida a opção editar, clicando nesta opção o conteúdo se torna editável e uma barra de ferramentas é exibida na parte superior da tela, conforme Figura 5.11.

[7]Simple Mail Transfer Protocol (Protocolo de Transferência de Correio Simples)

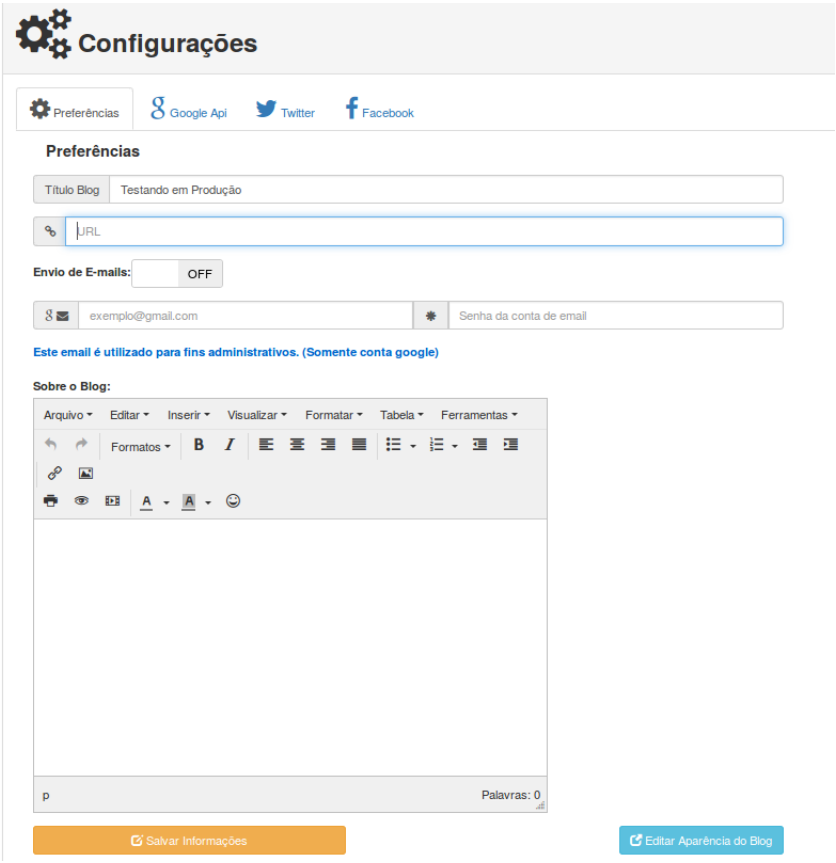


Figura 5.10: Configurações - Aba Preferências

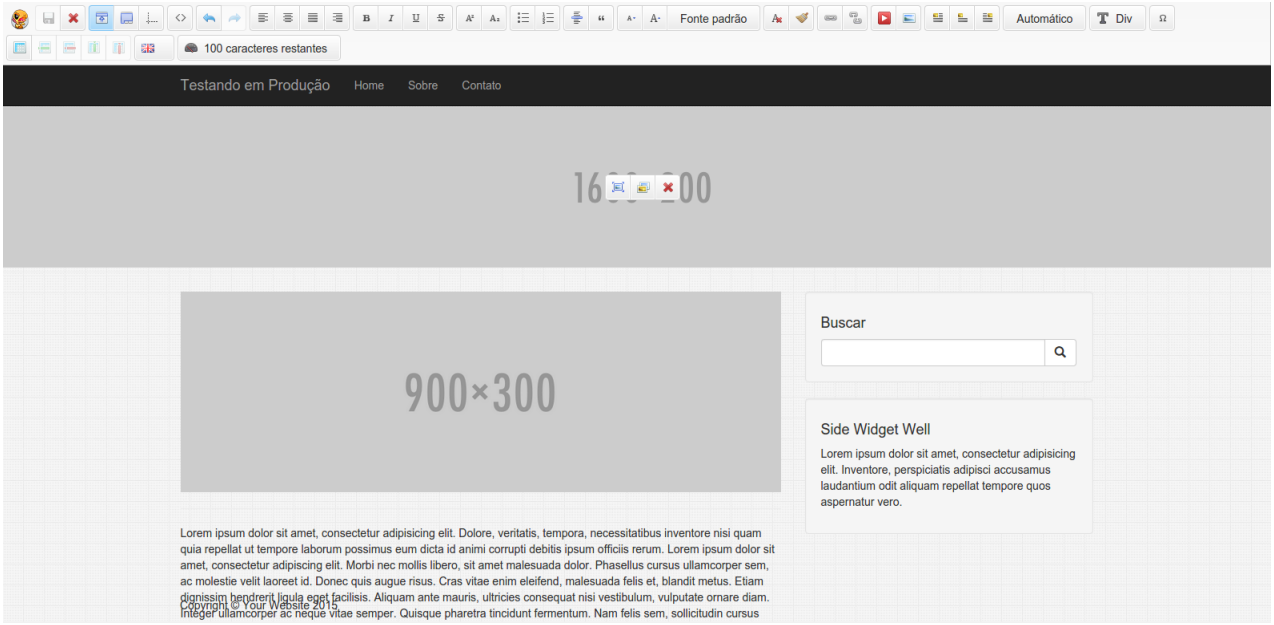


Figura 5.11: Editar aparência do blog

A barra de ferramentas permite entre outras opções, editar o estilo do texto, editar como *HTML* além de inserir vídeos e imagens. Após efetuar as edições que deseja, basta clicar no ícone de disquete para salvar. Automaticamente todas as suas edições são exibidas ao acessar a página.

Entre os campos editáveis estão o nome dos menus superiores, a imagem abaixo dos mesmos, podendo ser trocada por qualquer outro conteúdo desejado pelo usuário, a barra de busca e o menu lateral abaixo da mesma.

A próxima aba é a *Google Api* (5.12), aqui é possível configurar a API para uso de ferramentas *Google* e ativar o uso das mesmas. Para tal é necessário inserir o *email* de desenvolvedor *Google* e a chave de autenticação informadas no *Google Developer Console* [8]. Para a utilização do *Google Analytics* ainda é necessário inserir o *script* de acompanhamento fornecido ao criar a aplicação no site do mesmo. Após inserir os dados e ativar as opções, o gráfico inicial é preenchido com informações de acesso ao blog, o *card* de informação sobre Acessos na tela inicial passará a exibir a quantidade de acessos além de outros dados visíveis na opção Estatísticas.

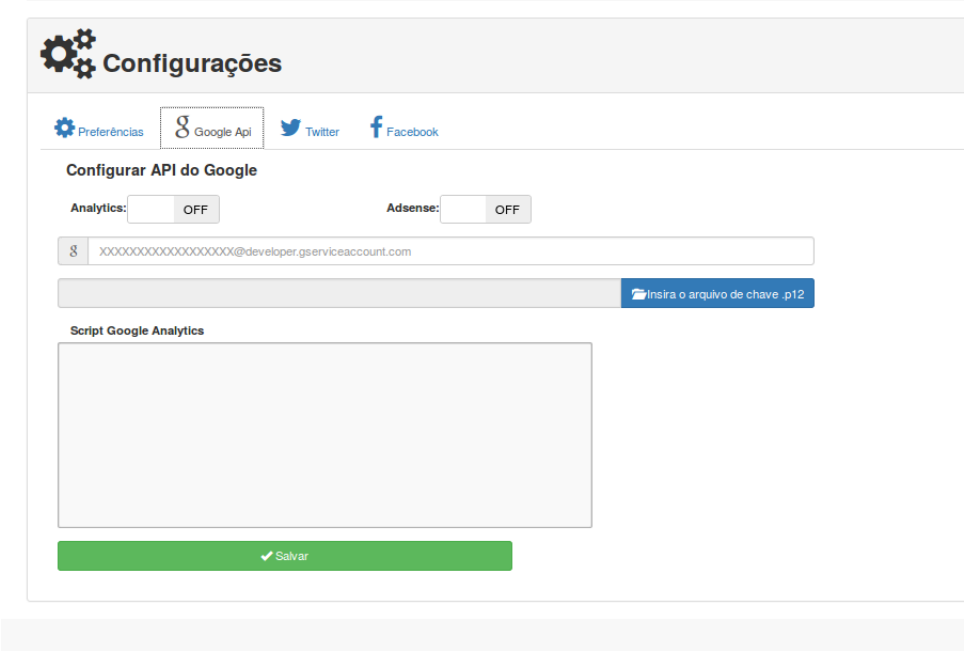


Figura 5.12: Configurações - Aba *Google Api*

Continuando, há a aba *Twitter Api*(Fig. 5.13). Caso queira monitorar a quantidade de seguidores de um determinado perfil diretamente no *pluton*, devem ser informados o *app id* e *app secret* fornecidos pelo mesmo ao instanciar uma nova aplicação no site `http://apps.twitter.com`, além do perfil que deseja monitorar.

A última aba remete ao *facebook* (Fig. 5.14), para coletar dados sobre páginas do

[8]Ferramenta do desenvolvedor do *Google* para gerenciar e visualizar os dados de tráfego, autenticação e informações de faturamento para as *APIs* do *Google* utilizadas em seus projetos.

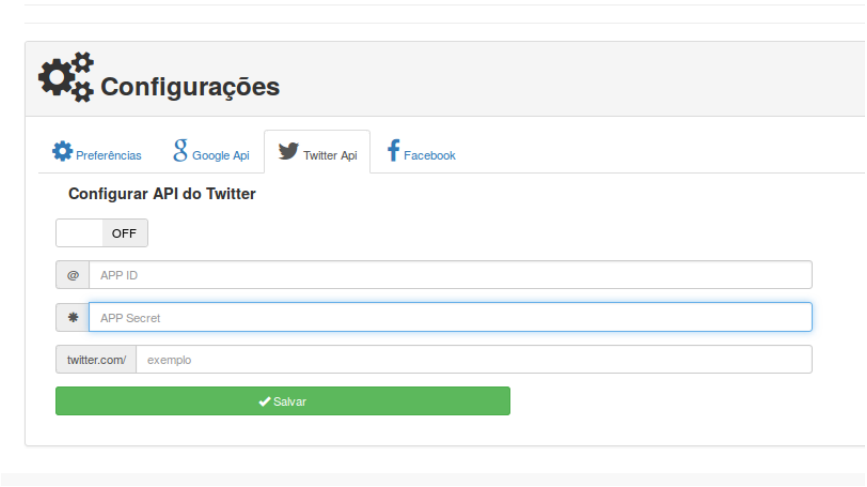


Figura 5.13: Configurações - Aba *Twitter Api*

facebook não é necessário criar uma aplicação do mesmo, pois tais dados estão disponíveis para consulta a partir de qualquer outro site sem necessidade de autenticação, então basta informar o nome da página no devido campo e salvar os dados.

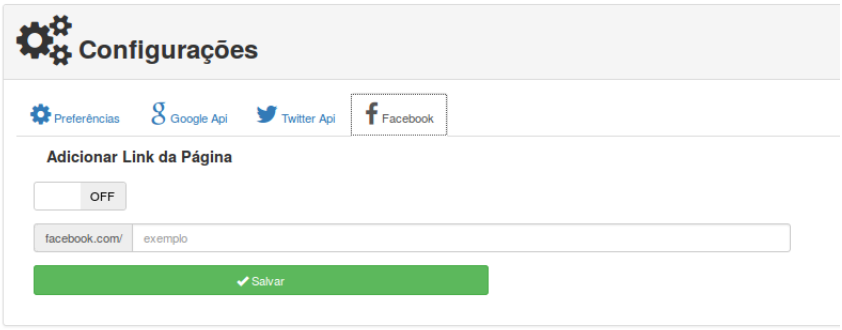


Figura 5.14: Configurações - Aba *Facebook*

As informações das *Apis* vistas acima são todas coletadas utilizando a biblioteca *libcURL* com *PHP*, no caso das *Apis* do *Google* e do *Twitter*, além da coleta de informações também é efetuada uma autenticação de usuário utilizando os dados de acesso de cada uma, pois tais ferramentas limitam a coleta de dados a usuários cadastrados, ao contrário do *facebook*, que disponibiliza os dados sem necessidde de uma conta.

Usuários

Após terminar as configurações do sistema, é possível usufruir de todas as suas funcionalidades por completo. A primeira é o gerenciamento de usuários, ao clicar no primeiro menu da barra lateral mais duas opções são exibidas "Novo" e "Editar".

Ao clicar em "Novo" o método *newUserAction*() da classe *UsersController* (Fig. 5.15)

carrega o formulário de cadastro de novos usuários na tela(Fig. 5.16). Os dados necessários para um novo usuário são: nome, *e-mail*, login, nível de permissão e senha além da imagem de perfil.

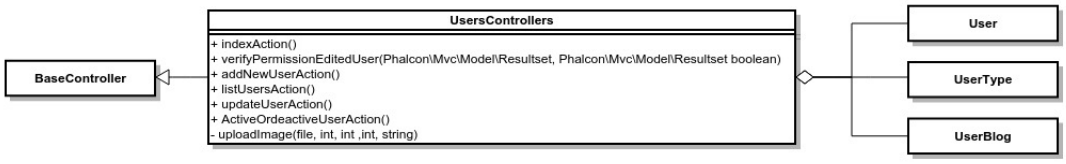


Figura 5.15: Diagrama Classe *UsersController*

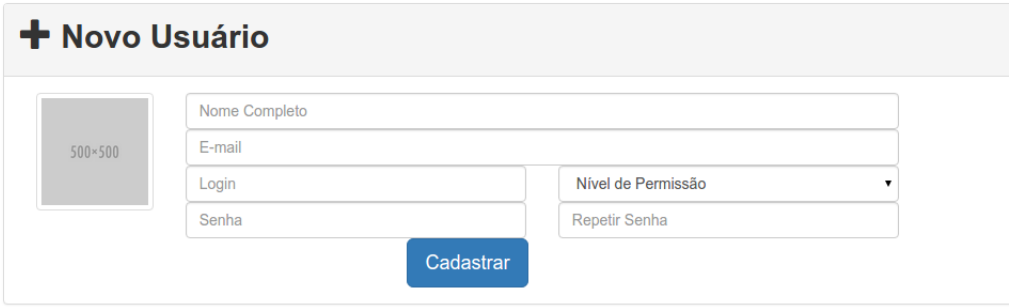


Figura 5.16: Formulário - Cadastro Novo Usuário

Clicando em editar, é exibido uma tabela com todos os usuários cadastrados no sistema (Fig. 5.17), para editar é necessário clicar na opção editar referente ao usuário que se deseja efetuar modificações. É carregado o mesmo formulário para cadastro de usuários, porém com os dados preenchidos.

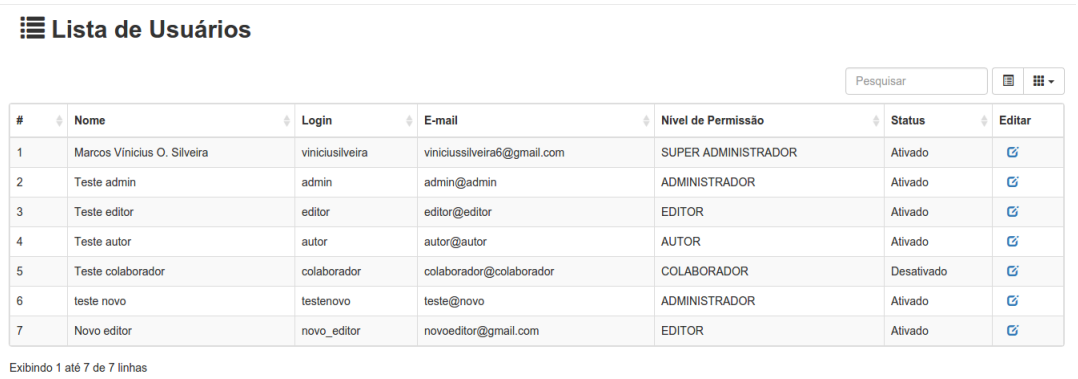


Figura 5.17: *Lista de usuários cadastrados*

Usuários cadastrados que possuem postagens vinculadas aos seus perfis podem ser apenas desativados, pois as postagens de sua autoria não podem ser deletadas. Usuários que não criaram ou publicaram nenhum tipo de conteúdo podem ser deletados sem problemas. Caso seja necessário bloquear o acesso de um usuário e não for possível deletar a conta do mesmo, pode ser feita uma desativação da conta, clicando na opção desativar. A Figura 5.18 exemplifica um formulário de edição de usuários.

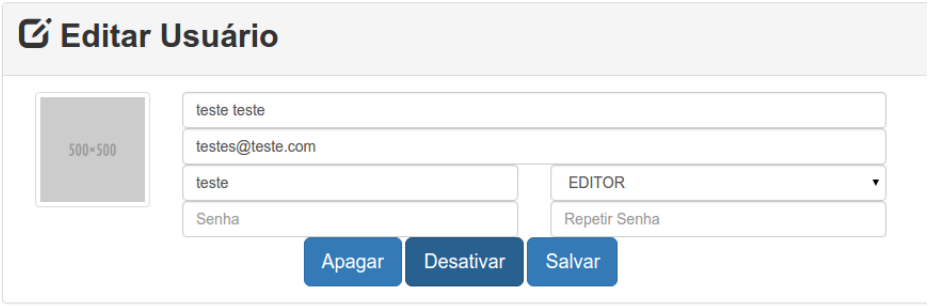


Figura 5.18: Formulário de edição de usuário

Posts

Parte essencial do sistema, onde as informações exibidas para os usuários são criadas, editadas e publicadas. A classe *PostControllers*(Fig. 5.19) gerencia todas as atividades relacionadas as postagens do blog.

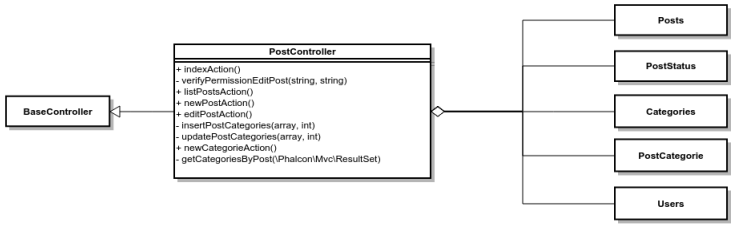


Figura 5.19: Diagrama classe *PostController*

Os dados necessário para criar uma postagem, conforme visto no formulário da Figura 5.20 são:

- Título: Título da postagem a ser exibida no blog.
- Conteúdo da postagem: Texto, imagem, ou qualquer outro conteúdo que deseja disponibilizar para os leitores.
- Autor: Criador do conteúdo disponibilizado.
- Status: Pode ser quatro:
  - Publicado: O *post* é criado e disponibilizado a partir da data informada na criação do mesmo.
  - Pendente: Concluído, porém aguardando aprovação de um editor.
  - Rascunho: *Post* ainda em criação.
  - Lixo: Postagem excluída ou não aprovada para publicação.
- Data: Data de publicação da postagem. O conteúdo será disponibilizado a partir da data informada, se possuir o *status* publicado. Os *posts* são ordenados por data.
- Categorias: *Tags* das postagens, palavras chaves que ajudam a localizar *posts* com conteúdos parecidos.

Uma postagem pode ser criada ou editada, mas nunca excluída, por exemplo: um autor criou uma postagem e colocou com o *status* Pendente. Em seguida um editor visualizou, encontrou um erro ortográfico, corrigiu e alterou o *status* para publicado. Após as alterações efetuadas pelo editor a postagem teve seu conteúdo alterado e também foi publicada. Porém

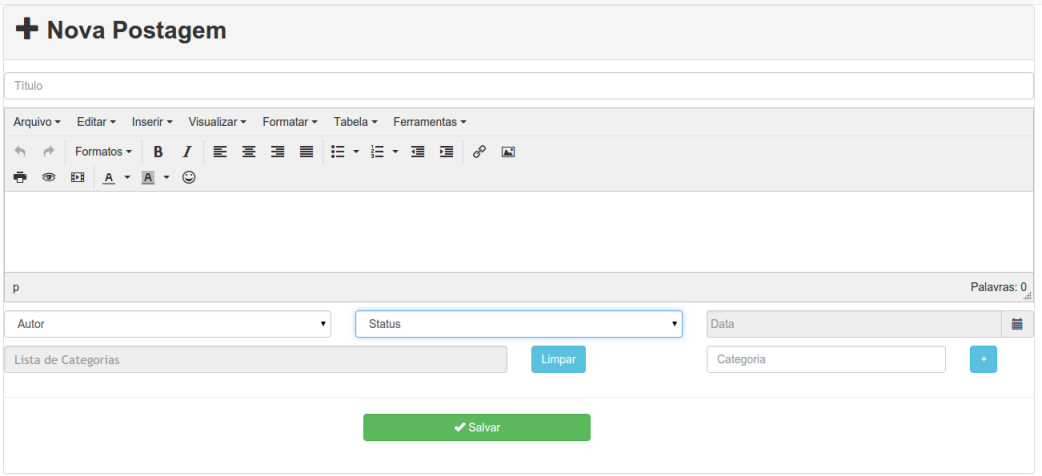


Figura 5.20: Formulário Nova Postagem

caso o editor deseje excluir a postagem deve-se alterar o *status* da mesma para Lixo, assim a postagem é descartada e não publicada.

As postagens são armazenadas no banco de dados em formato texto. Imagens, vídeos ou qualquer outro tipo de arquivos devem ser armazenados utilizando ferramentas externas, diminuindo as exigências de armazenamento e *internet* do servidor. Também se dá ao fato de existirem ferramentas onde é possivel armazenar estes conteúdos de forma gratuita e segura como *imgur* [9], *youtube*[10], *dropbox*[11] entre outros.

Estatísticas

A exibição das estatísticas conforme visto na Figura 5.21 depende da configuração das *APIS* do *Google*, pois tais informações são coletadas utilizando a ferramenta *Analytics* da empresa.

Nesta tela é exibido o total de acessos, total de *pageviews*, acessos mensais e por país em gráfico além do total de ganhos no mês corrente com propagandas com o *Google Adsense*. Os dados exibidos são básicos, para informações detalhadas o usuário deve acessar os painéis de cada ferramenta citada.

Tanto a tela quanto as informações são carregadas pela função *indexAction* do controlador *StatisticsController* a qual utiliza funções da biblioteca *Analytics* para carregar os dados, conforme pode ser visto na Figura 5.22.

[9] Serviço de *upload de imagens* (`www.imgur.com`).
[10] Serviço de armazenamento e reprodução de vídeos (`www.youtube.com`).
[11] Serviço de armazenamento de arquivos em nuvem (`www.dropbox.com`).

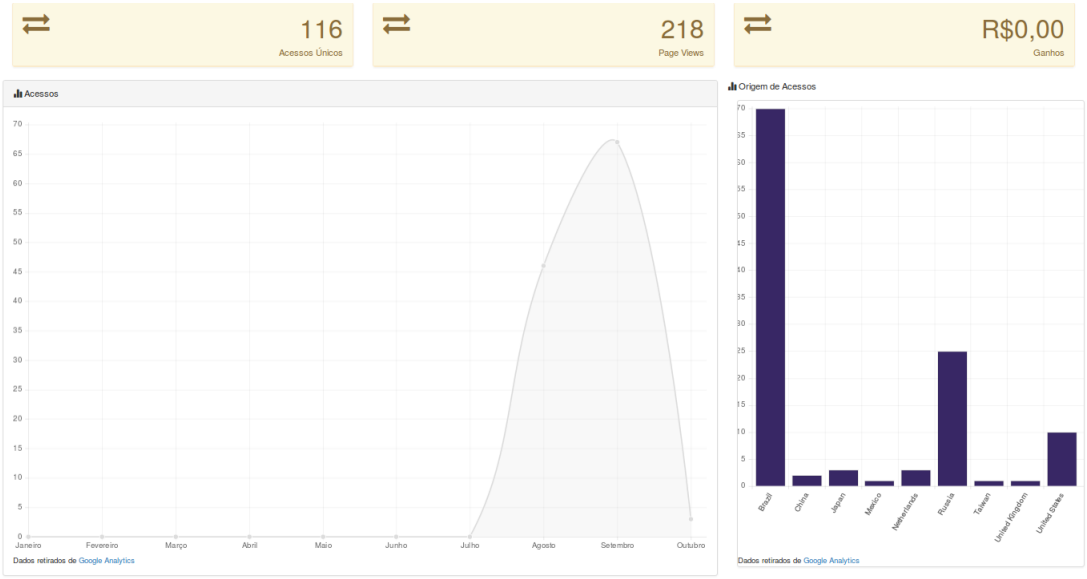


Figura 5.21: Tela Estatísticas

Figura 5.22: Diagrama Classe StatisticsController

Atualizações

O*github* permite a criação de *releases* do projeto hospedado em seus servidores, utilizando esta funcionalidade é possível coletar informações sobre as versões de um determinado projeto para ser informado sobre possíveis atualizações.

A classe *UpdateController*(Fig. 5.23) utiliza *cURL* para coletar informações sobre os *releases* existentes no projeto hospedado no *github* e caso haja uma atualização disponibiliza o link para o usuário efetuar o *download*, conforme pode ser visto nas figuras 5.24 e 5.25.

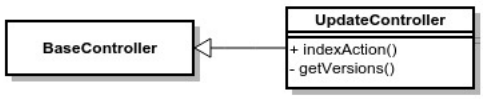


Figura 5.23: Diagrama Classe UpdateController

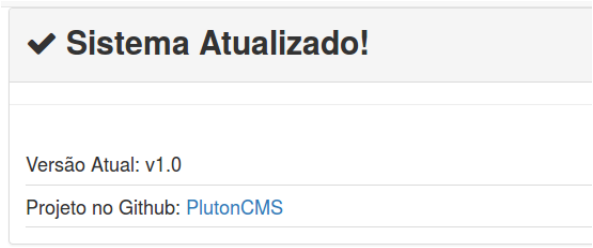


Figura 5.24: Tela Atualizações - Sistema Atualizado

Figura 5.25: Tela Atualizações - Nova versão

Plugins

Plugins são funcionalidades que podem ser adicionadas a um sistema. Por exemplo, um usuário deseja utilizar o sistema *Pluton* porém quer uma funcionalidade que o sistema ainda não possui, o mesmo pode desenvolver esta funcionalidade e, seguindo alguns passos inserir a mesma no sistema e utilizar o *pluton* com a funcionalidade que precisa.

Para desenvolver um *plugin* é necessário seguir algumas regras. Primeiramente o *plugin* deve estar em uma classe com o sufixo *controller* e herdar o controlador principal *BaseController*, além de carregar alguns dados conforme visto no código fonte abaixo:

```
1
2  <?php
3
4  namespace Multiple\Backend\Controllers;
5
6  class NovoController extends BaseController {
7
8      public function indexAction() {
9
10         $this->session->start();
11         //verifica se há usuário ativo
12         if ($this->session->get("user_id") != NULL) {
13             //busca informações do usuário ativo
14             $vars = $this->getUserLoggedInformation();
15             //busca os dados do menu lateral do sistema
16             $vars['menus'] = $this->getSideBarMenus();
17
18             //Seu código fonte aqui...
19
20             //seta os valores retornados para exibição na view
21             $this->view->setVars($vars);
22         }
23     }
24 }
```

Os nomes da classe e do método podem variar desde que possuam os sufixos *Controller* e *Action* respectivamente. Também é possível o usuário criar quantos métodos forem necessários, desde que os que precisem ser acessados através das *views* estejam definidos como públicos.

Após finalizar o desenvolvimento o arquivo de código fonte deve ser salvo com o mesmo nome da classe criada na pasta *apps/Backend/Controllers* do projeto. Por exemplo, se a classe se chama *SalesController* o arquivo deve se chamar *SalesController.php*.

O último passo é informar para o sistema que existe uma nova funcionalidade para ele. Para tal, o menu *Plugin* foi criado.

A Figura 5.26 mostra o formulário de cadastro de novos plugins, a seguir uma pequena explicação de cada campo solicitado.

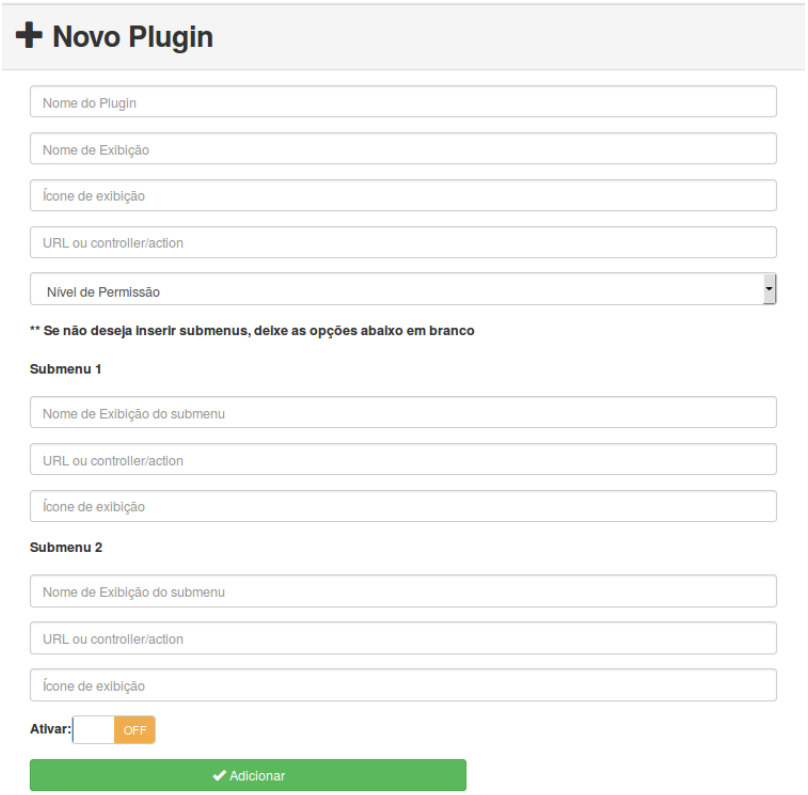


Figura 5.26: Tela Formulário *Plugins*

- Nome do *Plugin*: Nome do *plugin* para armazenamento no banco de dados.
- Nome de Exibição: Nome exibido no menu do sistema.
- Ícone de exibição: Mais especificadamente uma classe das ferramentas *glyphicons* ou *font awesome* que disponibilizam pacotes de ícones para personalização de sistemas *web*.

- URL ou *controller/action*: Aqui é informado o que o menu deve fazer quando o usuário clica no mesmo. Suponhamos que quando o usuário clica no *menu* "MeuPlugin" o sistema deve exibir os submenus do *menu* "MeuPlugin" então neste campo deve ser inserido o id do *submenu*, por padrão é preferível que coloque "#sub-nomeplugin". Caso não haja a necessidade de *submenus* informa-se aqui o controlador e a função que devem ser chamadas, por exemplo meuplugin/index (não há a necessidade de informar os sufixos *Controller* e *Action*).

- Nível de permissão: Quais tipos de usuários terão acessos ao menu.

- Nome de exibição do submenu: Nome de exibição do *submenu*, caso exista.

As demais opções dos submenus seguem a mesma descrição de outras opções de mesmo nome.

Caso não haja a necessidade de *submenus* os campos dos mesmos devem ser deixados em branco.

Após adicionar um *plugin* o menu pode ser visualizado na barra lateral esquerda, ao final das opções. Se tudo estiver correto a nova funcionalidade está instalada e funcionando.

Plugins também podem ser editados, clicando em *plugin*, Editar, uma lista com todos os plugins é exibida (fig 5.27), basta escolher qual deseja efetuar modificações e clicar na opção editar da linha do mesmo. O formulário é carregado para edição do *plugin* e após terminar basta clicar novamente em salvar para que as alterações tenham efeito.

Figura 5.27: Lista *Plugins*

5.3 *Blog*

Um *blog* (contração do termo inglês *web blog*, é uma página *web* que permite uma atualização rápida de conteúdo através de artigos ou posts. Os *blogs* surgiram em 1999 com a utilização do *sofware Blogger* criado pelo norte-americano *Evan Willians*. Foi criado para auxiliar na publicação de conteúdos *online* dispensando conhecimento especializado(Komesu (2004)).

O *blog* criado pelo *pluton* é o *frontend* do sistema, é o resultado da utilização da aplicação para os usuários finais (leitores), onde todo o conteúdo criado é disponibilizado. Ao contrário do *Backend* é mais simples e possui apenas um controlador responsável pelas funcionalidades do mesmo que pode ser visto na Figura 5.28.

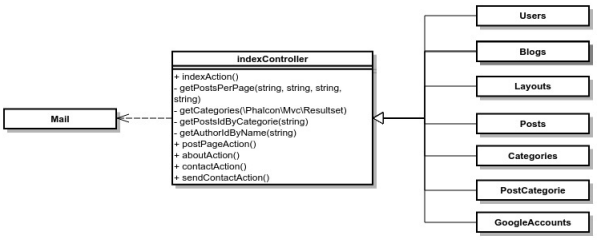


Figura 5.28: Backend - IndexController

Visualmente possui uma *timeline* onde são exibidas as postagens por data, cada *post* contém um título, um autor, e diversas *tags* utilizadas para filtrar conteúdos com os mesmos temas. Uma barra lateral com um campo para busca de postagens e abaixo campos que o usuário pode inserir o que desejar como propagandas ou links de redes sociais. Um menu superior com as opções *Home*, Sobre e Contato. A opção *Home* carrega a página principal do *blog*, a segunda opção, Sobre, carrega uma página com um informações sobre o *site* (Fig. 5.29) e por último a opção contato que carrega um pequeno formulário para envio de mensagens para o *site* (Fig. 5.30), podendo ser utilizada para envio de crítica ou sugestões.

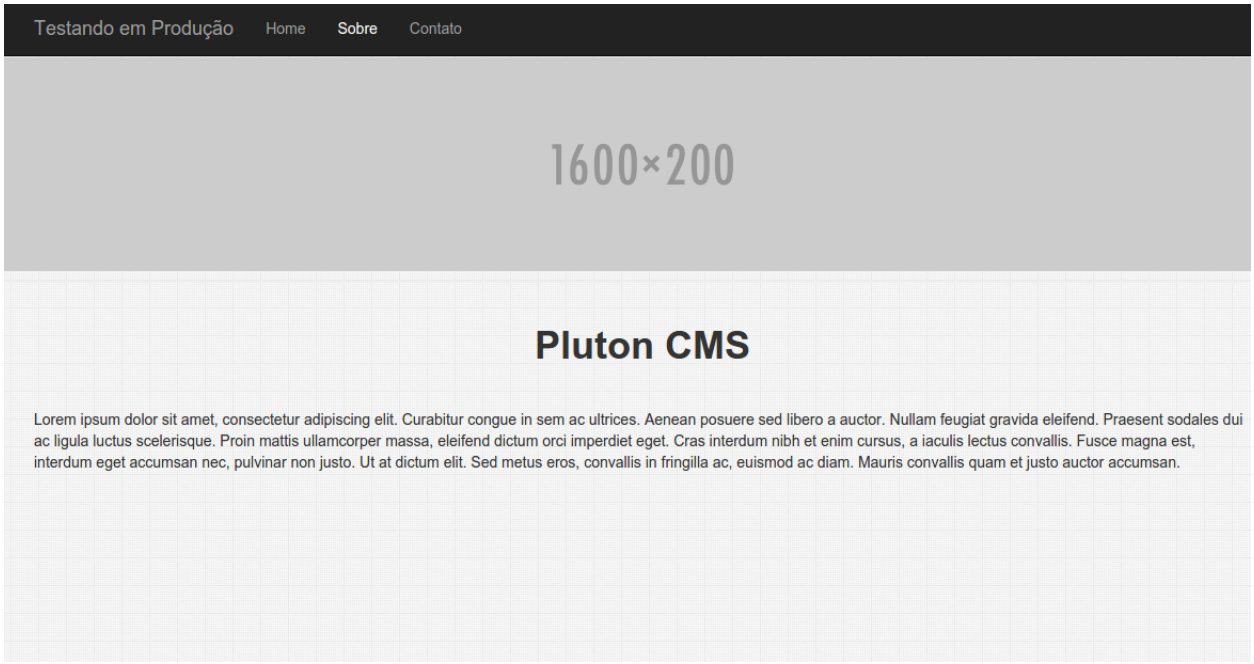


Figura 5.29: *Blog* - Página de informações sobre o blog

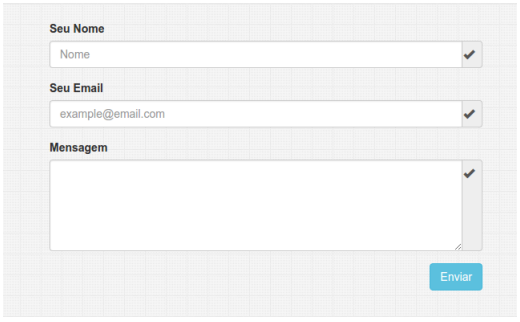


Figura 5.30: *Blog* - Página de Contato

5.3.1 Postagens

As postagens são exibidas de forma prévia na *timeline* do *blog*(Fig. 5.31), para visualizar o conteúdo de uma determinada postagem é necessário abrir a mesma, e então todo o conteúdo é exibido, conforme Figura 5.32.

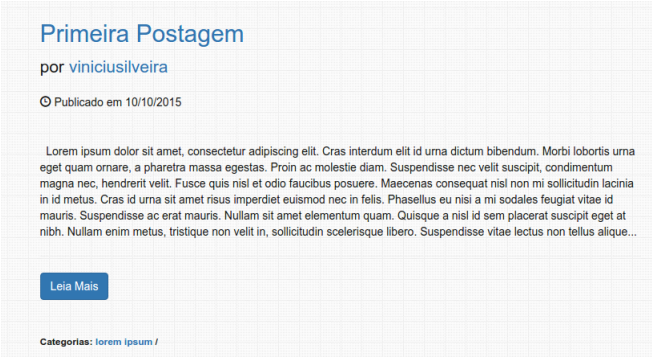


Figura 5.31: Prévia da postagem

Os comentários das postagens são feitos via *facebook*, para tal o usuário deve estar logado com seu perfil da rede social, pois além de ser uma maneira menos complicada de comentar, como exigir o cadastro no site, também facilita a identificação de quem é o publico que está comentando e aumenta a interação entre autores e leitores.

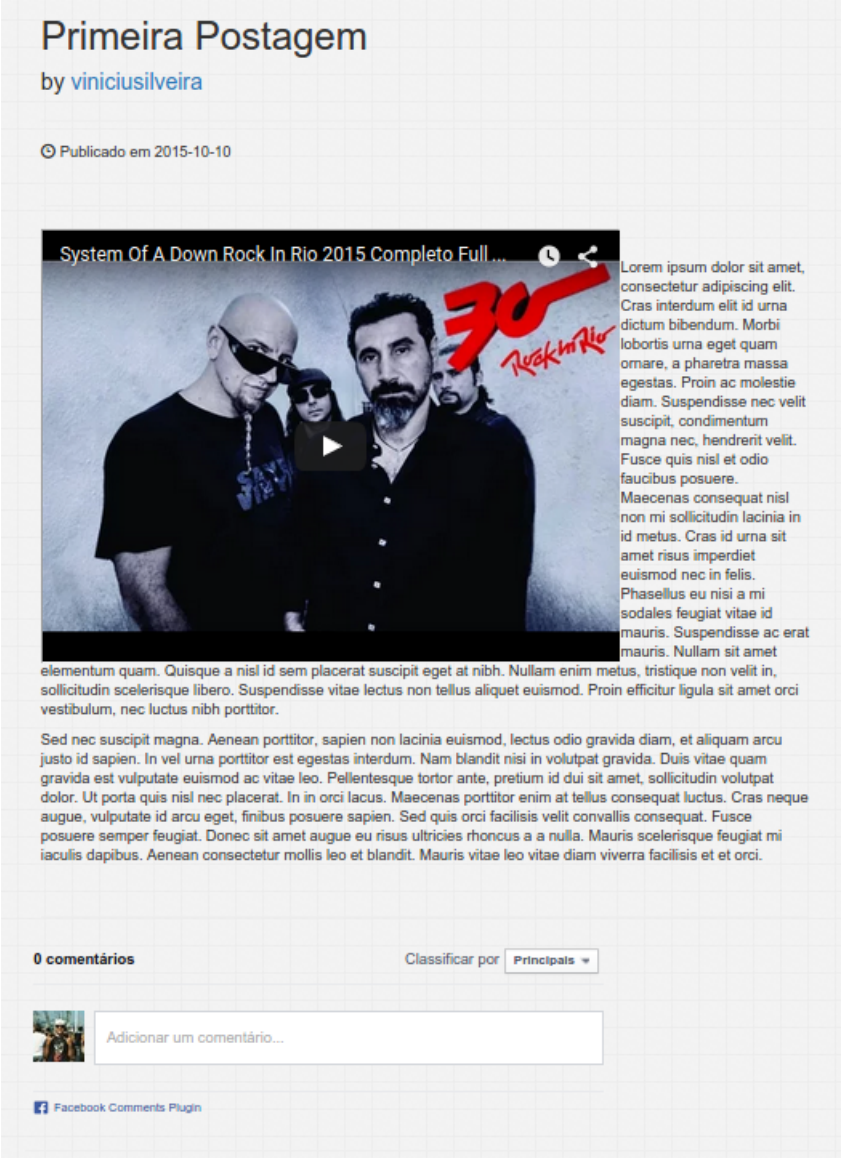


Figura 5.32: Postagem Completa

Capítulo 6

Conclusão

Um *SGC* apesar de ser uma ferramenta demasiadamente simples, possui suas peculiaridades além de diversas funcionalidades. Sendo assim, o desenvolvimento torna-se algo mais aberto, com necessidade de ferramentas com diversas funcionalidades. *Phalcon* se portou da melhor maneira possível neste quesito, disponibilizando métodos para desde a criação de tabelas até a geração de *views*, atendendo de uma maneira excepcional todas as necessidades do sistema.

Sendo um *framework* completo, utiliza-lo inicialmente pode ser trabalhoso se comparado a programação com a linguagem *PHP* pura, devido aos padrões exigidos pelo mesmo. Porém o resultado final é um *software* com código fonte organizado e legível e uma maior facilidade não só na manutenção como desenvolvimento de novas funcionalidades.

Uma outra complicação foi a não existência de uma funcionalidade para envio de e-mails no referido *framework*, necessitando assim a utilização de uma outra ferramenta especifica para tal função chamada *SwifitMailer*. A utilização de tal ferramenta com *phalcon* é documentada em um exemplo de projeto chamado *Vokuro*[1] criado por membros da equipe do *framework* exemplificando a utilização da mesma.

Apesar de ser uma ferramenta nova ainda em ascensão, *Phalcon* é completo e atende as necessidades dos programadores, possui um fórum próprio para solucionamento de dúvidas e uma documentação simples e bem explicativa. Sua utilização simplificou o desenvolvimento de diversas funcionalidades, diminuindo a complexidade dos mesmos e como resultante o tempo de desenvolvimento necessário.

Para projetos futuros, a ferramenta pode ser melhorada com a adição de novas funcionalidades, como permitir postagens de leitores, onde os mesmos possam enviar conteúdos e editores aprovem diretamente no sistema, sem a necessidade de utilização de ferramentas de comunicação externa. Melhor integração com redes sociais, permitindo compartilhamento de conteúdos de maneira automática em páginas de *sites* como *facebook* e *twitter*. Também o desenvolvimento de melhorias e funcionalidades especificas para adaptação do sistema para utiliza-lo para criar páginas acadêmicas, onde professores, coordenadores e secretários

[1] `https://docs.phalconphp.com/pt/latest/reference/tutorial-vokuro.html`

possam disponibilizar informativos para a comunidade.

Referências Bibliográficas

Alexandre, A. and SANTOS, F. (2003). Programação para web utilizando php. *SI.: s. n.*

Anthes, G. (2012). Html5 leads a web revolution. *Communications of the ACM*, 55(7).

Brizeno, M. (2012). Mão na massa: Model viel controller. `http://brizeno.wordpress.com/2012/03/12/mao-na-massa-model-view-controller/`, acesso em agosto de 2014.

Campos, A. (2006). O que é software livre. *BR-Linux. Florianópolis, março de*.

Campos, L. S. and de Souza Ribeiro, M. W. (2007). Realidade virtual aplicada a e-commerce: Proposta de plataforma baseada em vrml e php.

Chagas, F., Carvalho, C., and Silva, J. (2008). Um estudo sobre os sistemas de gerenciamento de conteúdo de código aberto. *Relatório Técnico*.

Dall'Oglio, P. (2009). *PHP Programando com Orientação a Objetos-2ª Edição: Inclui Design Patterns*. Novatec Editora.

Doederlein, O. P. (2012). Artigo java magazine 76 - baixo acoplamento. `http://www.devmedia.com.br/artigo-java-magazine-76-baixo-acoplamento/15853#ixzz3AtkDGn7B`, acesso em agosto de 2014.

Eis, D. and Ferreira, E. (2012). *HTML5 e CSS3 com farinha e pimenta*. Lulu. com.

Farinelli, F. (2011). Conceitos básicos de programação orientada a objetos. 2007.

Gabardo, A. C. (2012). Php e mvc com codeigniter. *São Paulo. Editora Novatec*.

Galante, A. C., Moreira, E. L. R., and Brandão, F. C. (2007). Banco de Dados Orientado a Objetos: Uma Realidade. `http://www.fsma.edu.br/si/edicao3/banco_de_dados_orientado_a_objetos.pdf`, acesso em agosto de 2014.

Grillo, F. D. N. and FORTES, R. P. D. M. (2008). Aprendendo javascript.

Henrajani, A. (2007). Desenvolvimento ágil em java com spring, hibernate e eclipse.

Junior, F. C. (2006). Programando para web com php/mysql.

Komesu, F. C. (2004). Blogs e as práticas de escrita sobre si na internet. *Hipertexto e gêneros digitais: novas formas de construção do sentido. Rio de Janeiro: Lucerna*.

Lockhart, J. (2015). Php do jeito certo. `http://www.douglaspasqua.com/2015/04/20/injecao-de-dependencia-com-php-di/`, acesso em agosto de 2015.

Mealling, M. and Denenberg, R. (2002). Report from the joint w3c/ietf uri planning interest group: Uniform resource identifiers (uris), urls, and uniform resource names (urns): Clarifications and recommendations.

Melo, A. A. d. and NASCIMENTO, M. G. (2007). Php profissional: Aprenda a desenvolver sistemas profissionais orientados a objetos com padrões de projeto. *Editora Novatec. São Paulo-SP, Brazil*.

Miller, S. A. (2014). *Getting Started with Phalcon*. Packt Publishing Ltd.

Minetto, E. L. (2007). Frameworks para desenvolvimento em php. *São Paulo: Novatec*.

Pereira, J. C. and Bax, M. P. (2010). Introdução à gestão de conteúdos. *Revista Gestão & Tecnologia*, 1(1).

Phalcon, T. (2014a). Hello world benchmark. `http://docs.phalconphp.com/en/latest/reference/benchmark/hello-world.html`, acesso em agosto de 2014.

Phalcon, T. (2014b). Phalcon php framework documentation - release 1.3.0. `http://media.readthedocs.org/pdf/phalcon-php-framework-documentation/latest/phalcon-php-framework-documentation.pdf`, acesso em agosto de 2014.

PHP (2014a). Download. `http://php.net/downloads.php`, acesso em outubro de 2014.

PHP (2014b). Usage stats for january 2013. `http://php.net/usage.php`, acesso em outubro de 2014.

Raven, M. (2010). Cms–uma introdução aos sistemas gestores de conteúdo web.

Silva, M. S. (2011). Html5: a linguagem de marcação que revolucionou a web. *São Paulo: Novatec*.

Stenberg, D. (2012). curl-transfer a url.

Apêndice A

Instalação da Aplicação

O material a seguir refere-se à instalação, configuração e execução da aplicação no ubuntu 14.04 LTS, cuja versão pode ser obtida acessando o link abaixo:

`http://releases.ubuntu.com/14.04/`

Primeiramente é necessário instalar as ferramentas básicas para utilização do *Pluton*, a linguagem *PHP*, o banco de dados *Mysql* e o servidor *Apache*. Para tal execute os comandos abaixo na sequência que segue:

```
1 $ sudo apt-get-update
2 $ sudo apt-get install mysql-server apache2 libapache2-mod-php5 php5 php5-
    mysql phpmyadmin
```

Durante a instalação são solicitadas algumas informações para configuração das ferramentas.

Primeiramente é solicitado a criação de uma senha para o *MySQL*, insira uma senha e a repita conforme as imagens A.1 e A.2.

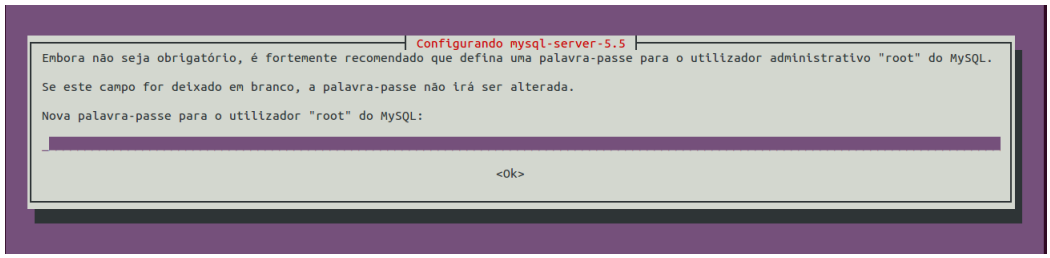


Figura A.1: Configurando senha *MySQL*

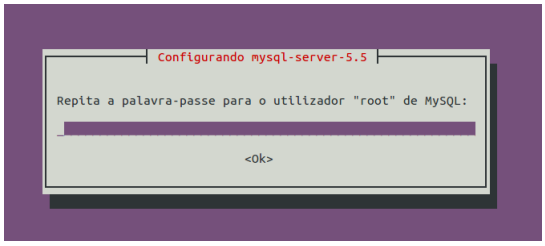


Figura A.2: Repetir Senha *MySQL*

Após configurar o banco de dados, são solicitadas algumas informações para configuração do *phpmyadmin*:

A primeira opção que será exibida é a da escolha do servidor para configuração do *phpmyadmin* (fig. A.3) Escolha o *apache* e tecle *enter*.

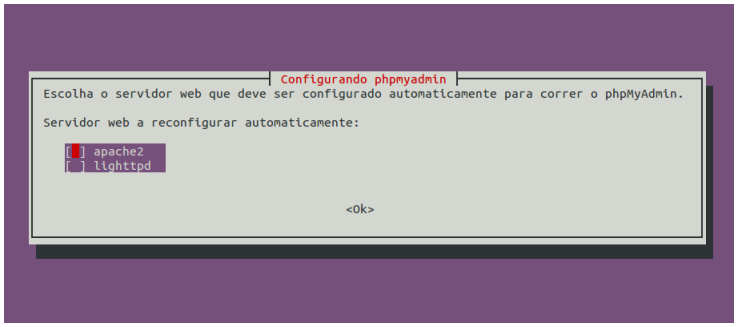


Figura A.3: Escolha do servidor phpmyadmin

Nesta opção (fig. A.4) é perguntado se deseja que o *phpmyadmin* configure uma base de dados padrão para utilização da ferramenta, escolha *Yes* e novamente tecle *enter*.

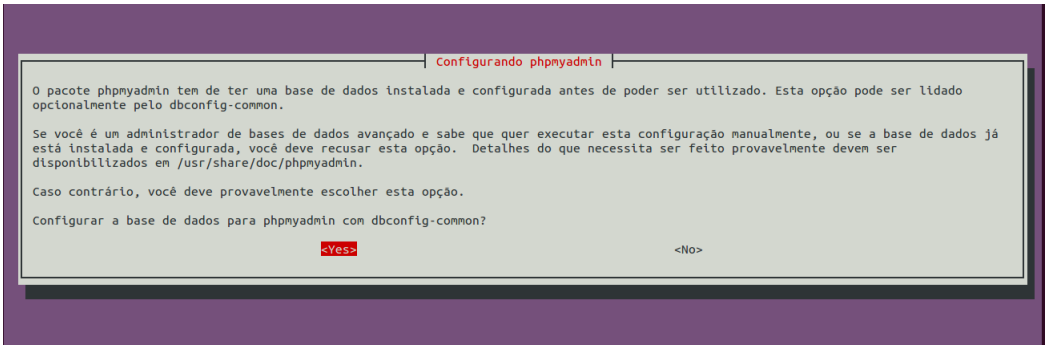


Figura A.4: Base de Dados padrão *phpmyadmin*

A seguir é solicitada a senha do banco de dados, informe a senha que foi utilizada durante a instalação do *MySQL*.

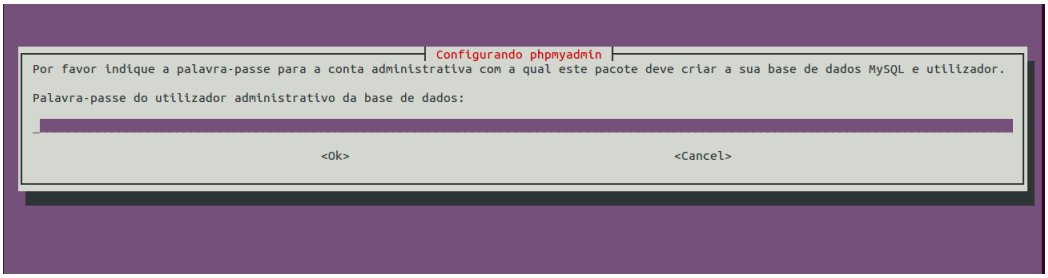


Figura A.5: 3

Por último informe uma senha para o *phpmyadmin* de preferência uma senha diferente do banco de dados. Insira e confirme a senha. Se tudo correu bem, ao acessar pelo navegador o endereço *http://localhost* irá ser exibida a tela parecida com a da figura A.6 sendo esta a tela principal do *apache*. E para saber se o *phpmyadmin* foi instalado corretamente acesse *localhost \phpmyadmin* será exibida a tela de login do mesmo (fig. A.7);

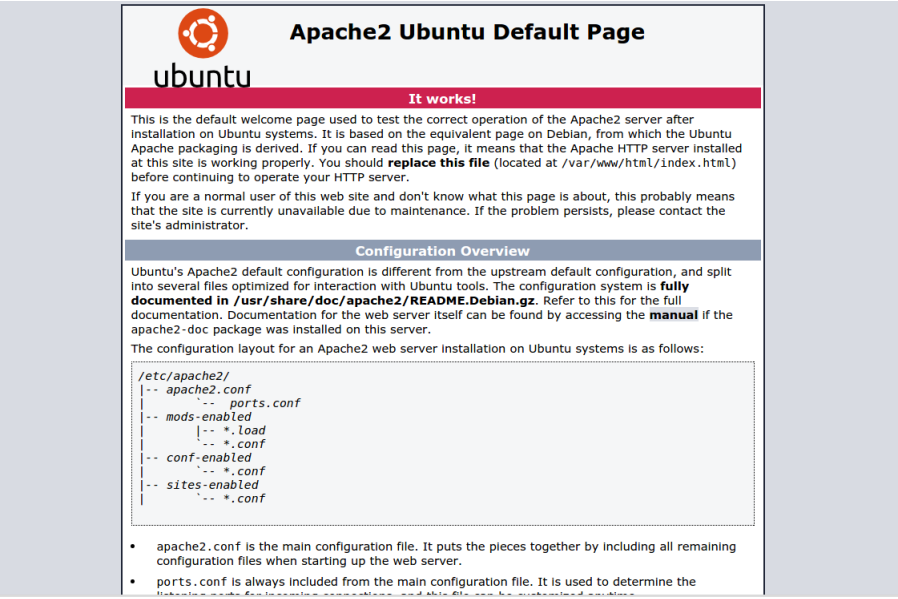


Figura A.6: Página *Apache*

Figura A.7: Tela de *login - phpmyadmin*

Após concluir a instalação, é preciso ativar o modo de reescrita [1] do *apache*.

Primeiramente execute o comando abaixo:

```
$ sudo a2enmod rewrite
```

Em seguida adicione ao arquivo */etc/apache2/sites-available/000-default.conf* o conteúdo abaixo após a linha *DocumentRoot /var/www/html*.

```
<Directory "/var/www/html">
    AllowOverride All
</Directory>
```

O próximo passo é instalar o *framework phalcon*:

```
$ sudo apt-add-repository ppa:phalcon/stable
$ sudo apt-get update
$ sudo apt-get install php5-phalcon
```

Para verificar se a instalação ocorreu sem problemas crie um arquivo na pasta do servidor *web* chamado *info.php*, com o seguinte conteúdo:

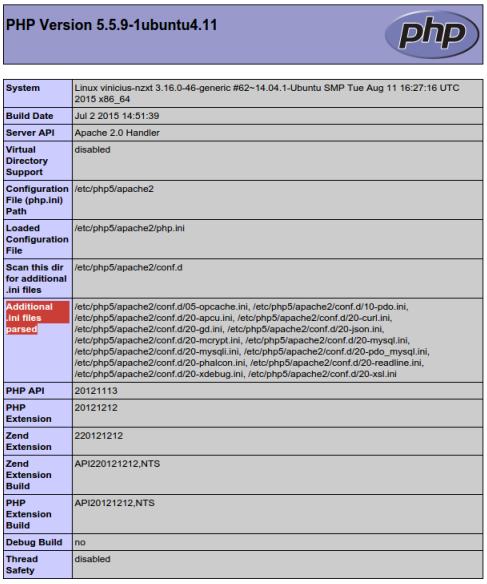
```
<?php
    phpinfo();
```

O resultado é uma página com os dados do *PHP* instalado no servidor, vide imagem A.8. Verifique se existe a opção *mod_rewrite* em *Loaded Modules* e *20-phalcon.ini* em *Additional .ini files parsed*. Se tudo estiver certo, o *framework* e o módulo de reescrita foram configurados corretamente.

O próximo passo é criar um banco de dados com o *MySQL* com o nome de sua escolha porém o mesmo deve possuir a codificação *UTF8*.

Após estes passos o servidor está configurado para executar uma aplicação com *Phalcon*. Copie o código fonte da aplicação para a pasta do servidor *apache*, */var/www/html/*, o nome da pasta também pode ser de sua escolha.

[1] *mod_rewrite*, é um módulo escrito para o servidor Apache, responsável pela reescrita de URLs em páginas Web. Que fornece uma regra simples sem níveis de seção na URL.



PHP Version 5.5.9-1ubuntu4.11

System	Linux vinicius-nzxt 3.16.0-46-generic #62~14.04.1-Ubuntu SMP Tue Aug 11 16:27:16 UTC 2015 x86_64
Build Date	Jul 2 2015 14:51:39
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
Additional .ini files parsed	/etc/php5/apache2/conf.d/05-opcache.ini, /etc/php5/apache2/conf.d/10-pdo.ini, /etc/php5/apache2/conf.d/20-apcu.ini, /etc/php5/apache2/conf.d/20-curl.ini, /etc/php5/apache2/conf.d/20-gd.ini, /etc/php5/apache2/conf.d/20-json.ini, /etc/php5/apache2/conf.d/20-mcrypt.ini, /etc/php5/apache2/conf.d/20-mysql.ini, /etc/php5/apache2/conf.d/20-mysqli.ini, /etc/php5/apache2/conf.d/20-pdo_mysql.ini, /etc/php5/apache2/conf.d/20-phalcon.ini, /etc/php5/apache2/conf.d/20-readline.ini, /etc/php5/apache2/conf.d/20-xdebug.ini, /etc/php5/apache2/conf.d/20-xsl.ini
PHP API	20121113
PHP Extension	20121212
Zend Extension	220121212
Zend Extension Build	API220121212,NTS
PHP Extension Build	API20121212,NTS
Debug Build	no
Thread Safety	disabled

Figura A.8: Informações sobre a instalação do *PHP*

```
1 $ sudo mv pluton /var/www/html/
```

E por último é necessário criar algumas pastas e alterar as permissões de escrita, dentro da pasta da aplicação execute os comandos abaixo:

```
1 $ sudo mkdir apps/config
2 $ sudo chmod –R 777 apps/config/
3 $ sudo mkdir keys
4 $ sudo chmod –R 777 keys
```

Se tudo ocorreu sem problemas ao acessar pelo navegador o endereço *localhost/pluton/admin* irá aparecer a tela vista na figura A.9:

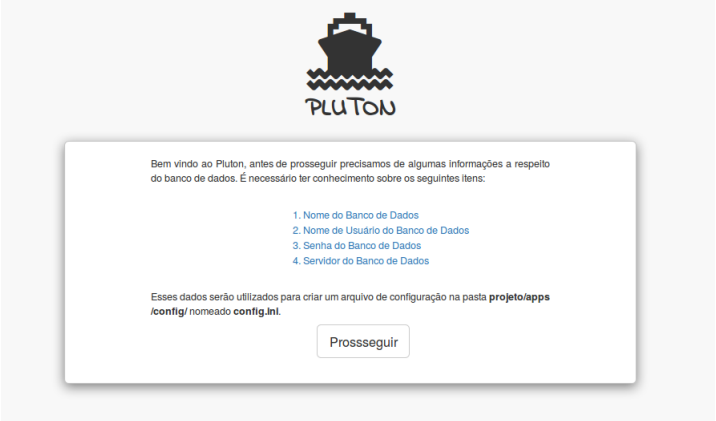


Figura A.9: Página Inicial da Instalação do Aplicativo.

Apêndice B

Configurando a *Google API*

Para utilizar as ferramentas disponibilizadas pelo *Google*, algumas configurações são necessárias. Primeiramente tenha uma conta *google* para o seu site. Caso não tenha crie uma em `https://accounts.google.com/SignUp?hl=pt-BR`.

B.1 *Google Console*

Após criar a conta google é necessário criar uma aplicação no *google console* e ativar as funcionalidades do *Analytics* e do *Adsense*. Acesse `https://console.developers.google.com` e efetue *login* com a conta criada no passo anterior, será redirecionada para a tela vista na figura B.1:

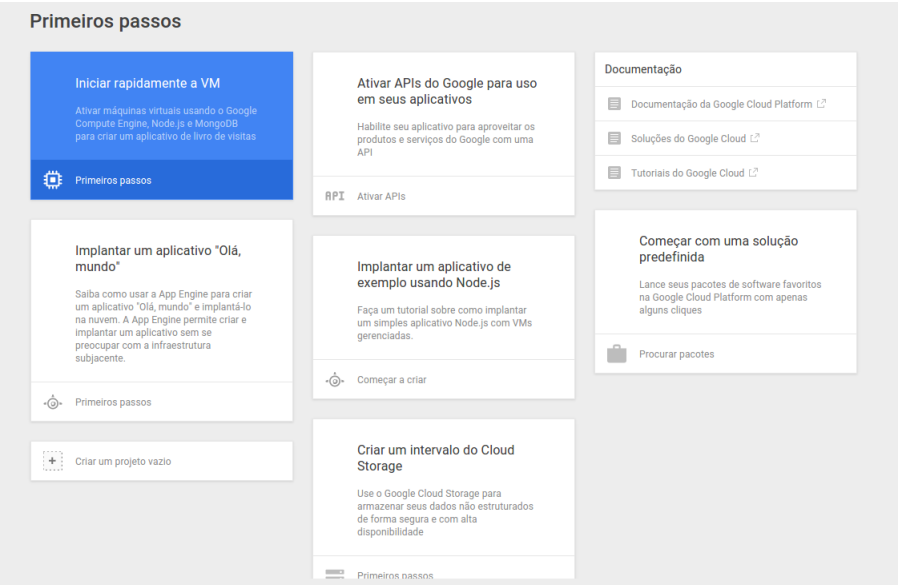


Figura B.1: Página Inicial *Google Console*.

Clique na opção Ativar *APIs* do *Google* para uso em seus aplicativos, será solicitado a criação de um novo projeto, insira um nome para o mesmo e clique em criar. Será exibida a Tela abaixo (fig. B.2):

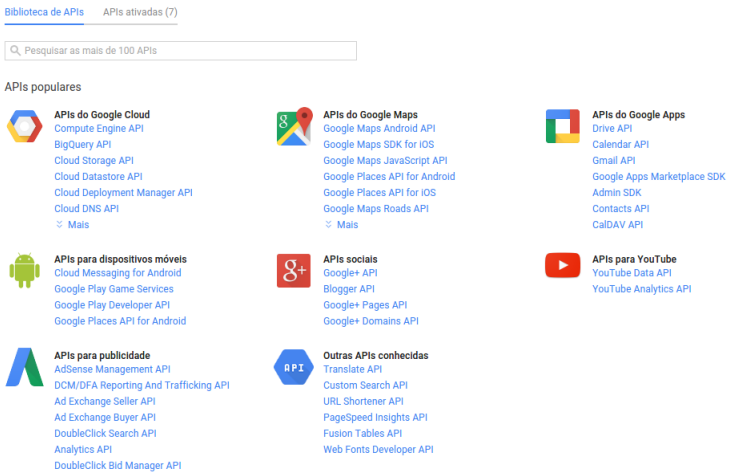


Figura B.2: *Google Console* - Lista de *APIs*

É possível ver na imagem acima na seção *APIs* para publicidade algumas opções, entre elas a *AdSense Management API* e *Analytics API*. Para ativá-las, clique em cima de cada uma e na nova página que abrir clique em ativar *API* e aguarde o processo de ativação terminar.

Após ativar as *APIs* é necessário configurar o acesso as mesmas, no menu lateral (fig. B.3) clique na opção *APIs* e autenticação e em seguida credenciais, será aberta uma página para adicionar uma credencial, antes de adicionar clique na parte superior onde está escrito Tela de consentimento *OAuth*. E informe os dados solicitados no formulário que irá aparecer. Após preencher os dados do formuláro e clicar em salvar é liberado para criar uma chave *OAuth*.

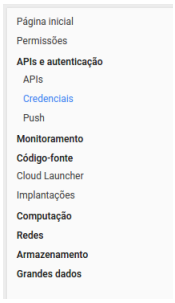


Figura B.3: Menu *Google Console*

Na tela seguinte clique em Adicionar Credenciais e depois em Conta de serviço, conforme a imagem B.4.

Figura B.4: Gerar Nova chave *OAuth 2.0*

Na página que abrir escolha a opção P12 e clique em criar, o site irá criar e fazer o *download* de um arquivo com a extensão .P12 além de exibir na tela o *ID* de desenvolvedor *Google* criado. O arquivo e o *ID* devem ser informados no sistema *Pluton* conforme visto anteriormente em 5.2.1.

B.2 *Google Analytics*

Utilizando a conta *google* criada no passo anterior, acesse o link `https://www.google.com/analytics/web/` e ative uma conta do *Goole Analytics*. Para tal, clique no botão Criar uma conta conforme visto na figura B.5, e em seguida faça *login*.

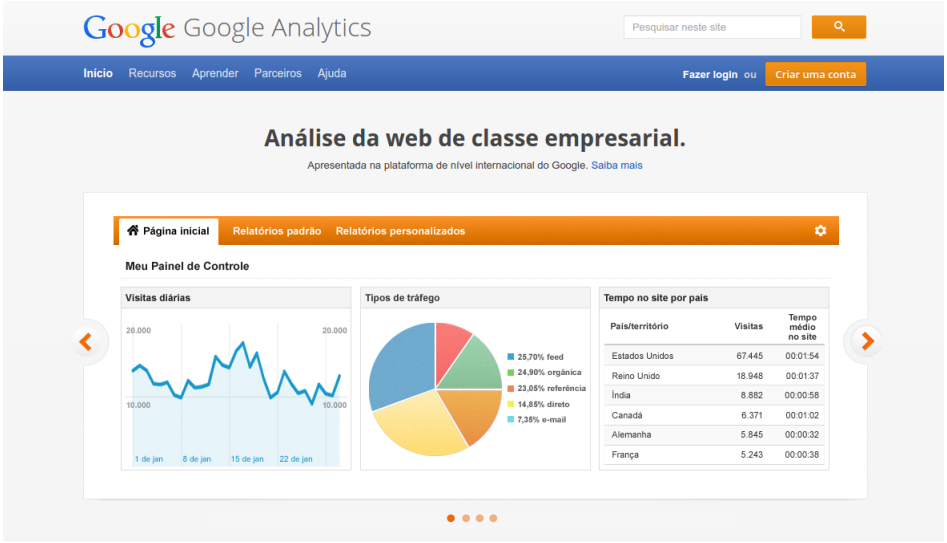


Figura B.5: Página Inicial *Google Analytics*.

Após efetuar o *login* é exibida a página vista na figura B.6, clique em inscreva-se para ir para o próximo passo.

Figura B.6: Nova conta - *Google Analytics*.

Em seguida deve ser preenchido um formulário com os dados da página a ser monitorada (fig. B.7), preencha o formulário e deixe todas as opções abaixo dele marcadas conforme visto na figura B.8 e por fim clique em Obter *ID* de acompanhamento.

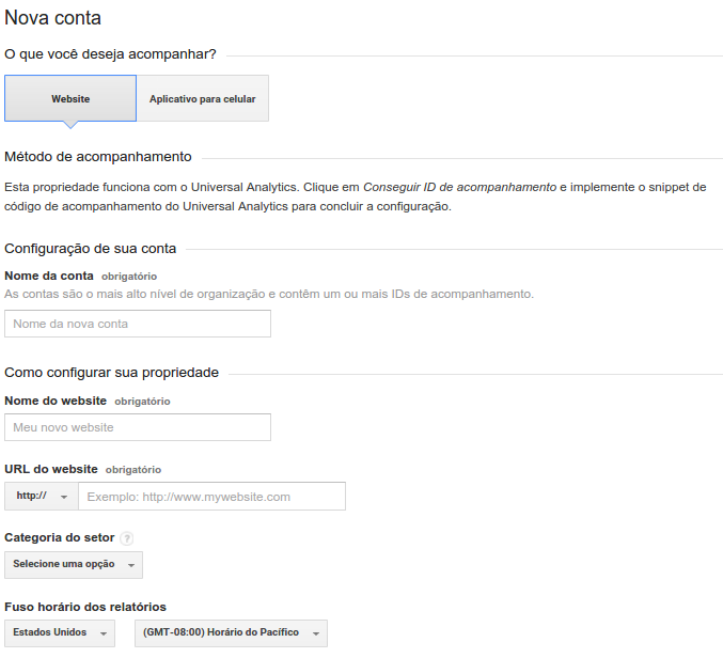


Figura B.7: Formulário Nova conta - *Google Analytics*.

Figura B.8: Opções Nova conta *Google Analytics*.

Será exibida uma página com os dados da sua conta e um código em *javascript* para inserir no *seu* site, este *script* deve ser inserido no *Pluton* conforme visto na seção 5.2.1.

B.3 Google Adsense

Para utilizar o *Google Adsense* seu site deve estar *online* e possuir um domínio que seja aceito pela ferramenta. Os dominios não podem possuir subdomínios como por exemplo exemplo.teste.br nem caminhos após o endereço como exemplo.com/exemplo.

A página do *Adsense* é `http://www.google.com.br/adsense`, acesse e clique na opção começar agora mesmo (fig. B.9).

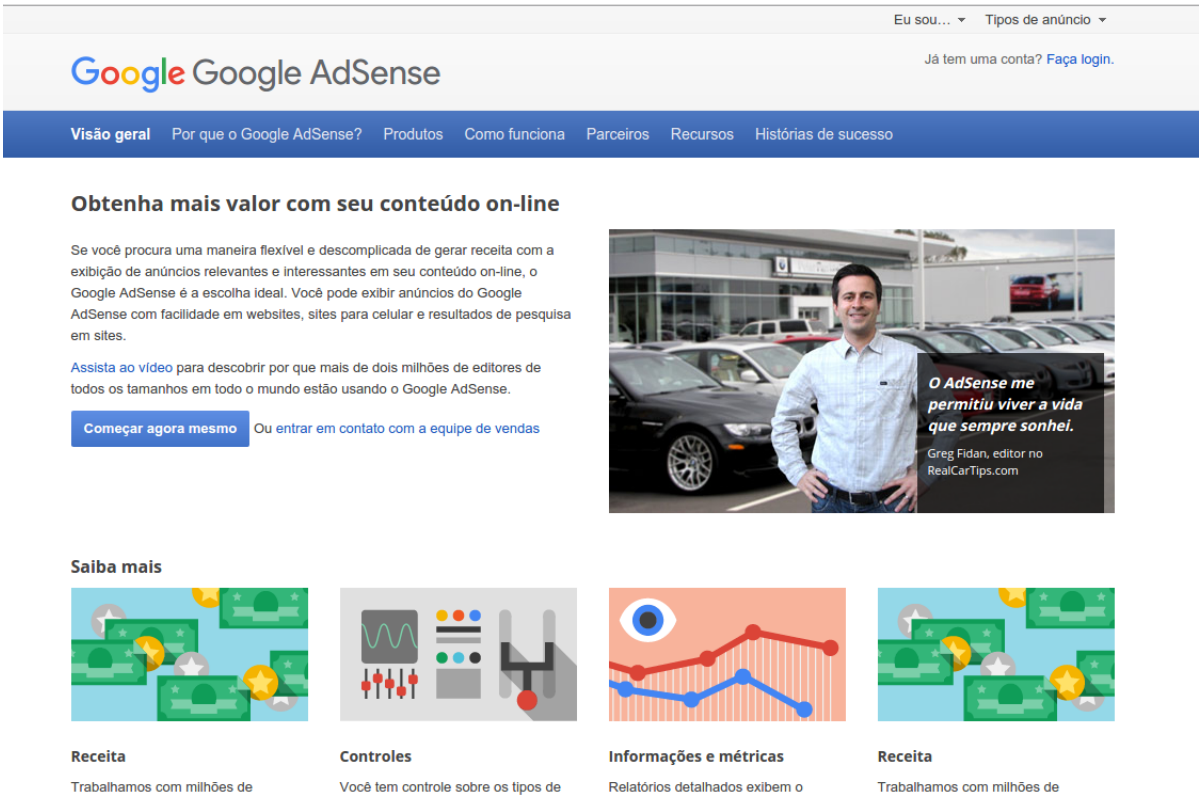


Figura B.9: *Google Adsense*

Após efetuar o login será solicitado o endereço do site e um idioma para o conteúdo a ser exibido (fig. B.10). Inserindo estes dados um novo formulário solicitando algumas informações sobre o usuário é exibido(fig. B.11). Preencha-o e clique em enviar minha inscrição.

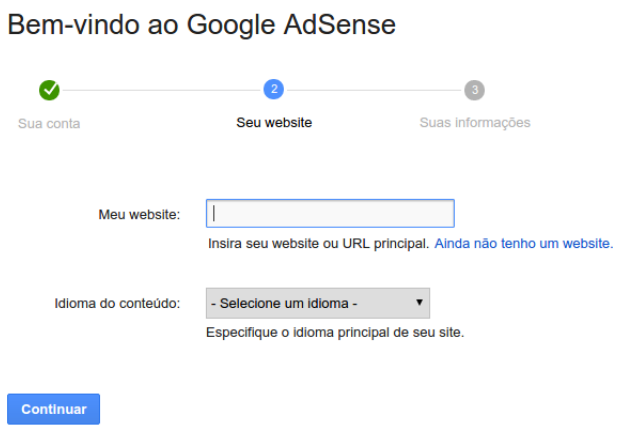


Figura B.10: Formulário *Google Adsense*

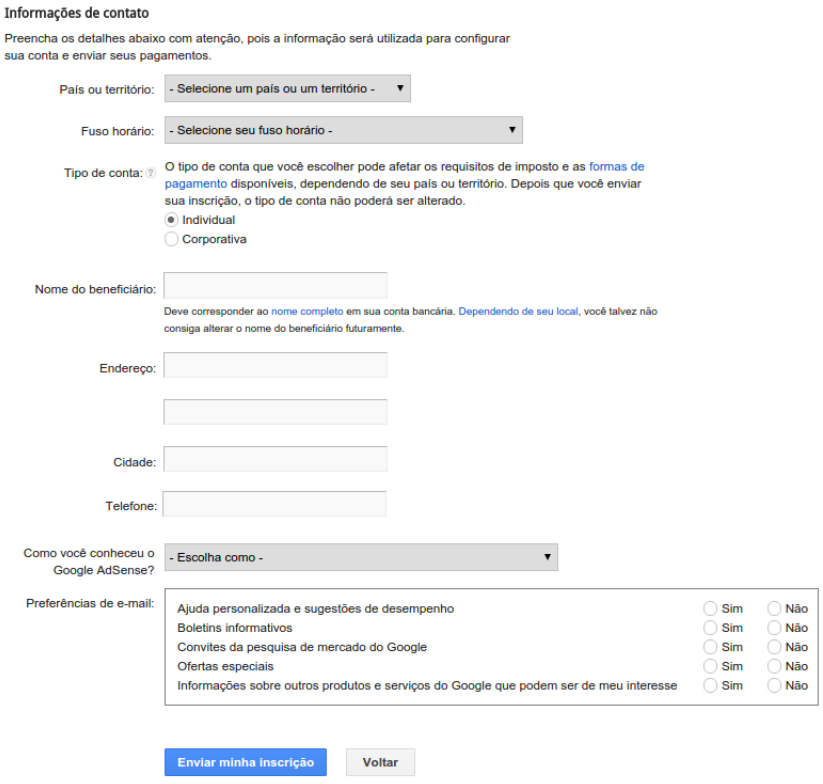


Figura B.11: Formulário dados de usuário - *Google Adsense*

Após preencher e enviar os formulários, aceite o Termo de serviços da ferramenta, o *site* será redirecionado para página inicial, sendo necessário um ultimo passo: inserir as propagandas no *site*. Clique em começar agora e a página será redirecionada pra criação de um novo bloco de anúncios (fig. B.12).

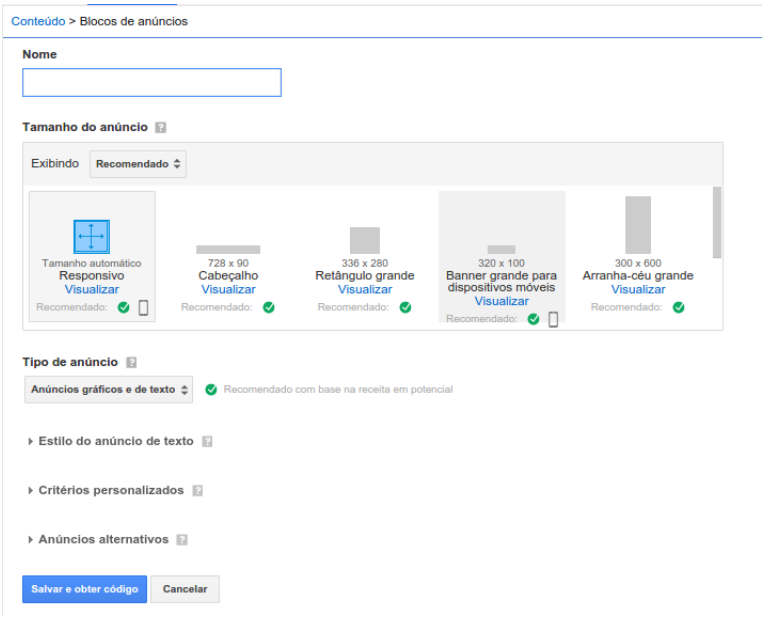


Figura B.12: Novo bloco de anúncios

Escolha as opções conforme desejar e clique em salvar e obter código. Será gerado um *script javascript* conforme a imagem B.13 para inserção na página.

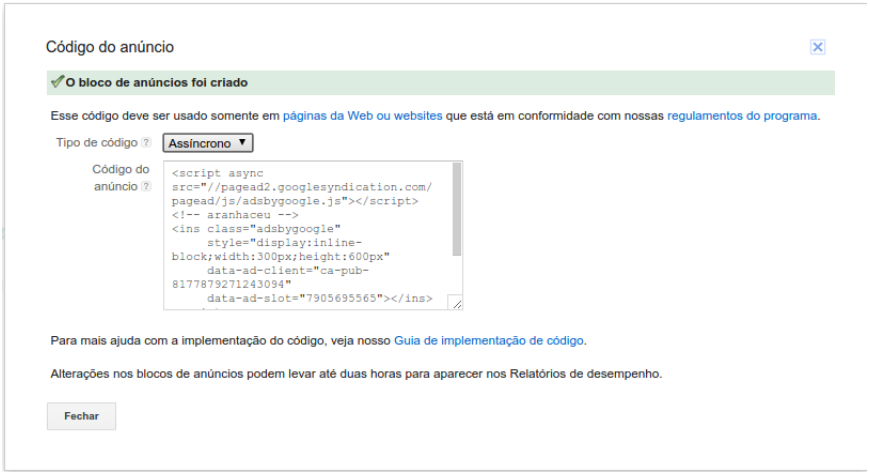


Figura B.13: Código de Propagandas - *Google Adsense*

Para inserir o código o usuário pode utilizar a edição de aparência do blog e colar o código fonte na barra abaixo da caixa de busca, por exemplo, não havendo a necessidade de editar arquivos de código fonte do projeto. Após inserir aguarde o *Google* aprovar seu cadastro e para começar a ter ganhos.

Apêndice C

Twitter API

Para coletar dados de perfis do *twitter* é necessário possuir uma conta no mesmo e criar um *APP* no site `https://apps.twitter.com/`. Acesse o site, efetue *login* com uma conta do *twitter* ativa e em seguida clique em *Create New App*. Um formulário solicitando informações da aplicação é carregado na tela (fig. C.1), informe os dados ao mesmo, aceite a licença de uso e clique em *Create your Twitter Application*.

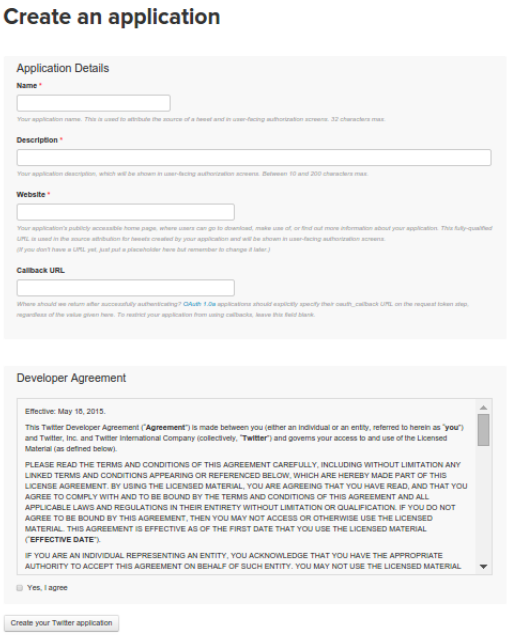


Figura C.1: Formulário *Twitter App*

Após criar o *App* é exibido uma página com os detalhes do mesmo, para ter acesso ao *App key* e *App Secret* clique na opção *manage keys and access tokens*. Ambos serão exibidos conforme na figura C.2.

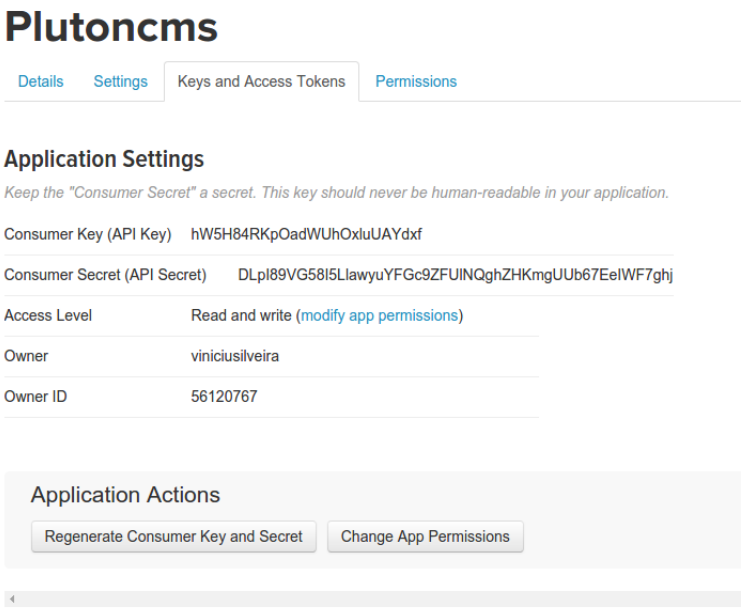


Figura C.2: *Twitter App Key e App Secret*

Apêndice D

Diagramas

D.1 Casos de Uso

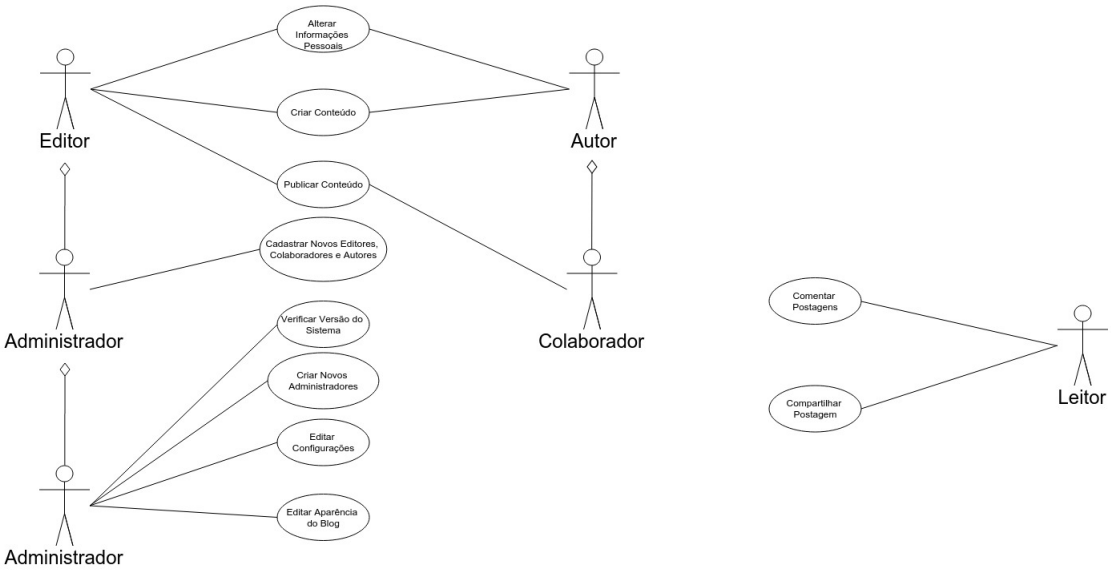


Figura D.1: Diagrama de Casos de Uso

D.2 Diagrama de Classe

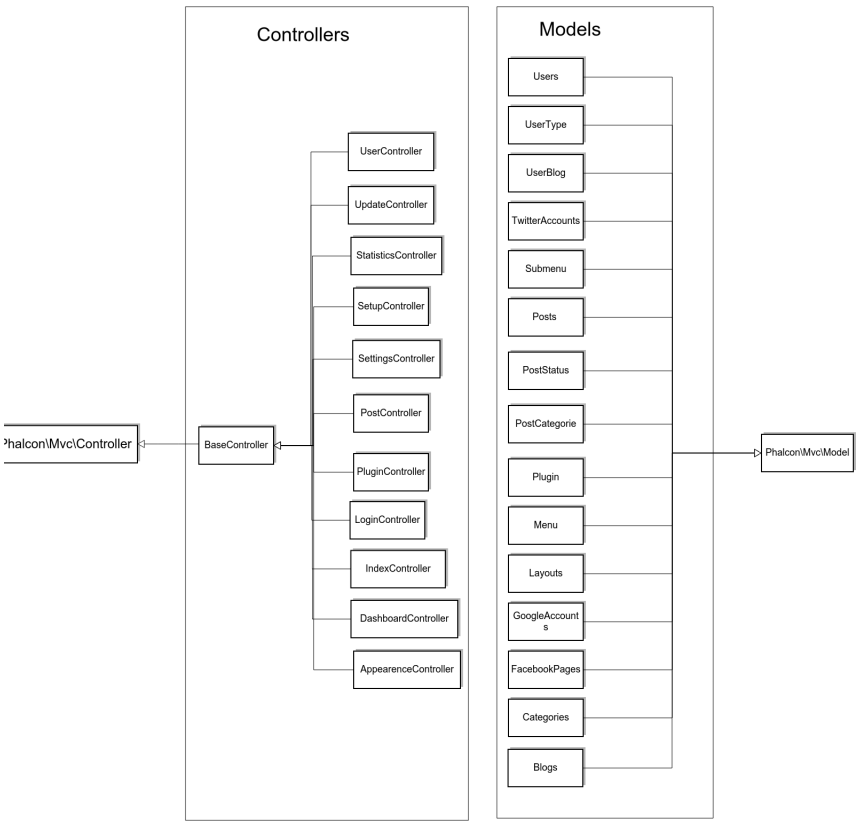


Figura D.2: Diagrama de Classes - *Backend*

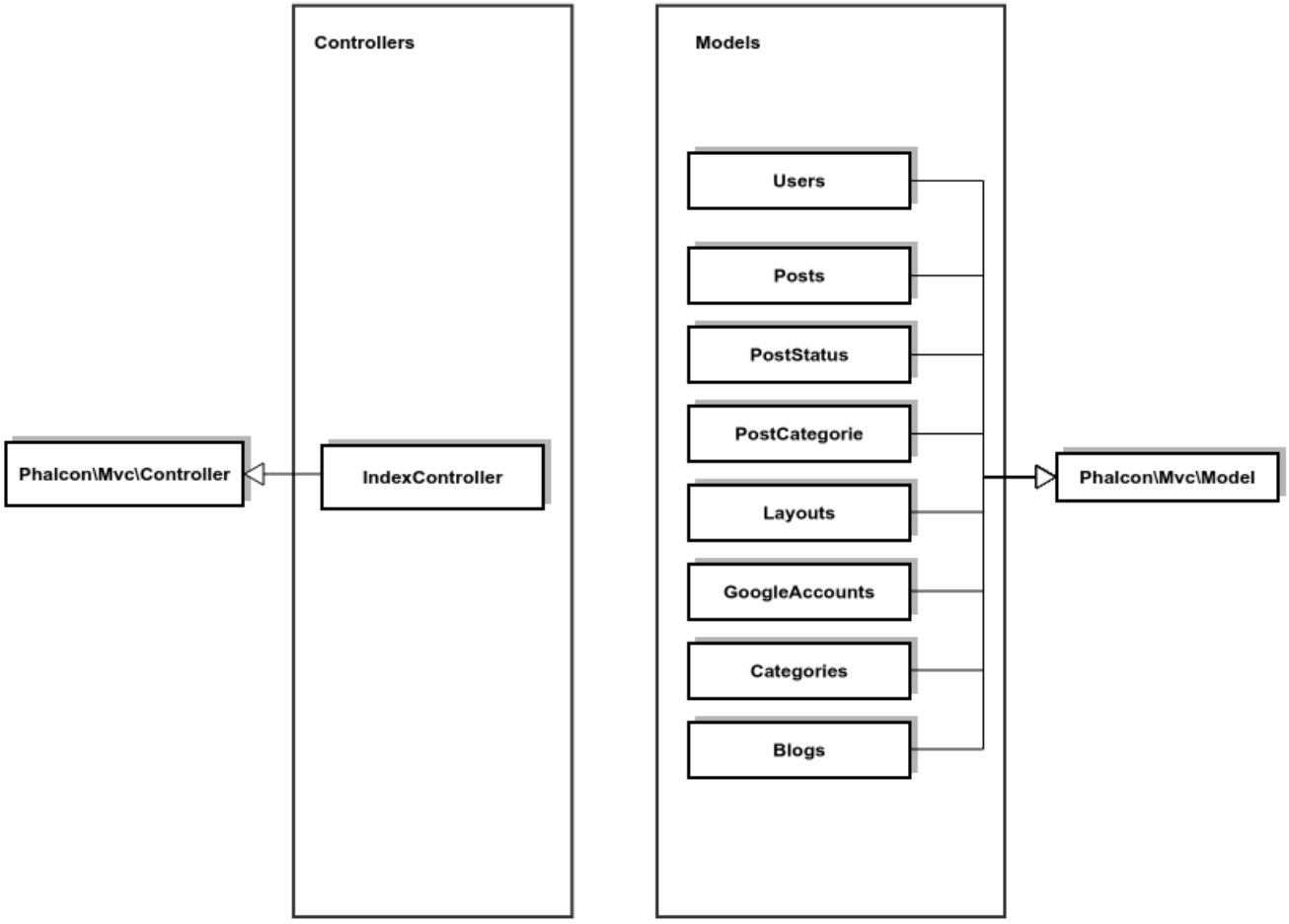


Figura D.3: Diagrama de Classes - *Frontend*

D.3 Diagrama de banco de dados

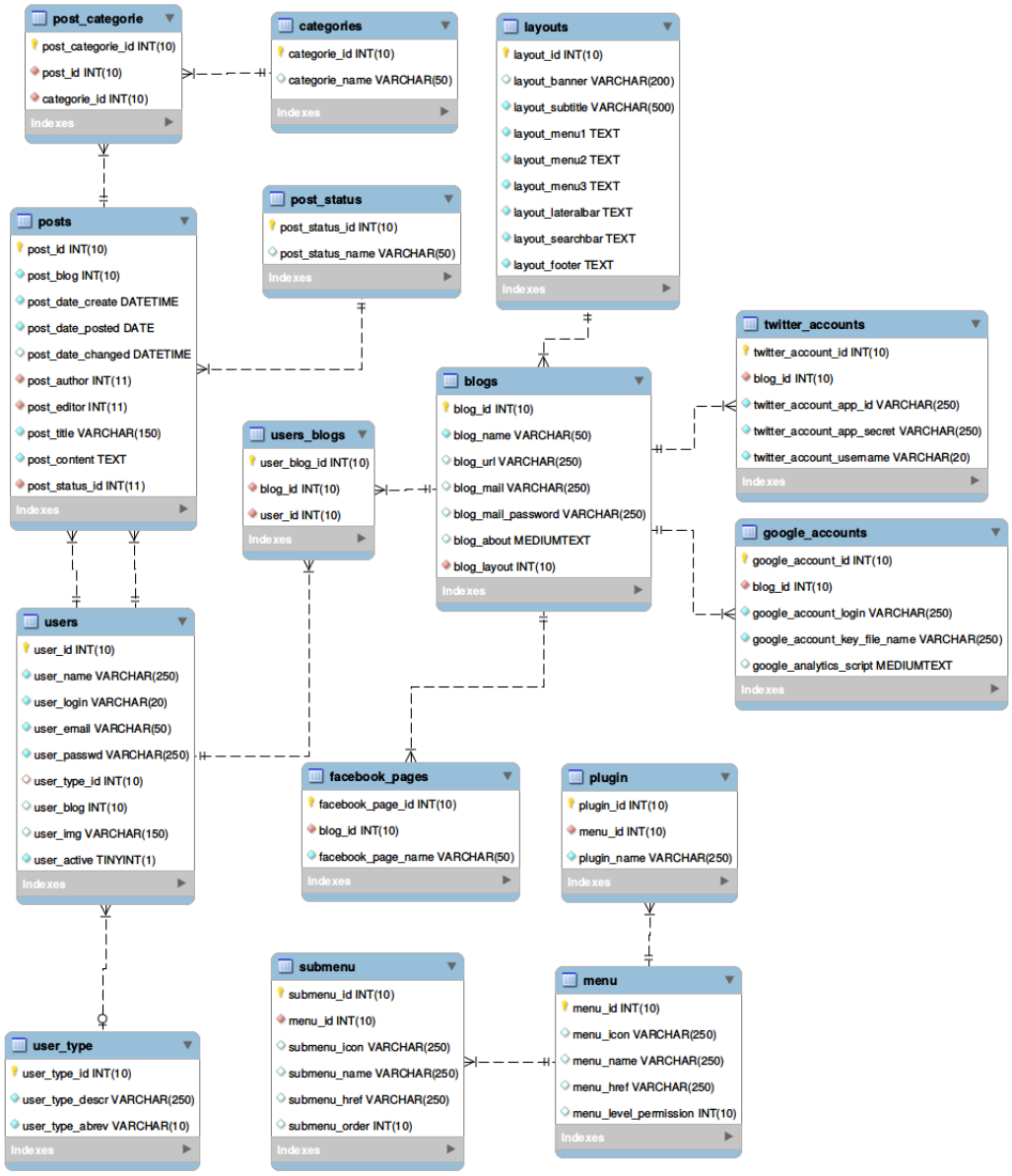


Figura D.4: Diagrama de banco de dados