Frameworks PHP

Nas últimas semanas fiz uma pesquisa para encontrar um framework PHP que eu pudesse usar em meus aplicativos/sites. Encontrei vários frameworks interessantes como

- Symphony
- Prado
- Code Igniter
- Zend Framework

Depois de ver vários deles uma pergunta me ocorreu: "realmente preciso de um framework?". Realmente utilizar um destes frameworks auxilia bastante o desenvolvimento, evitando que você refaça coisas que já existem. Mas muitas vezes estes frameworks possuem alguns problemas:

- são muito maiores do que você precisa;
- a documentação é confusa;
- poucos estão suficientemente maduros;
- é preciso aprender uma nova sintaxe ou maneira de se desenvolver;
- você fica "engessado", é difícil fazer coisas mais avançadas ou que saem do padrão CRUD;

Então pensei que o que eu preciso é somente uma maneira de separar a lógica da apresentação, uma maneira de facilitar o acesso a bancos de dados e algo que me ajude a trabalhar com AJAX. Assim, peguei algumas idéias que vi nas documentações que li e algumas coisas que já utilizava e uni tudo para suprir minhas necessidades.

Modelo de aplicacao

MVC Architecture?

Segundo Rasmus Lerdorf, criador da linguagem PHP, "MVC é palavra do momento em arquiteturas de aplicação web. Ela vem do design de aplicações para desktop orientadas a eventos e não se aplica muito bem no design de aplicações web. Mas felizmente ninguém sabe exatamente o que MVC significa, então você pode chamar seu mecanismo de separação de camada de apresentação de MVC e seguir em frente."

Então, o que estou usando aqui é somente um mecanismo de separar o layout (html+css) do código PHP. Ao invés de usar as três camadas do MVC eu resumi para duas, Visão e Controle.

A parte da visão é representada por templates. Seguindo outra tendência que percebi em outros frameworks a parte de visão é representada por simples scripts PHP ao invés de utilizar uma das ferramentas de Templates como Fast Template, Smarty, etc. É mais simples de trabalhar e não é necessário aprender uma nova sintaxe.

Para a parte do controle eu desenvolvi uma classe chamada app. Nesta classe eu controlo as ações do usuário. Ela age como o controlador das aplicações testando qual ação o usuário escolheu e invocando o método correto (que deverá ser reescrito nas suas subclasses, as novas aplicações). A idéia é que cada aplicação seja uma subclasse da classe app.

Este é o código da classe app:

```
1   <?php
2   error_reporting(1);
```

```
3   /**
4     * Classe que define uma aplicacao
5     * Elton Luis Minetto <eminetto at gmail dot com>
6     * Licenca: GPL
7     */
8   class app {
9           /**
10          * string de conexao com a base de dados
11          * @var string
12          */
13          static $db_string = "";
14
15          /**
16          * Construtor da classe
17          * @return void
18          */
19          function __construct($string) {
20
21                  app::$db_string = $string;
22                  /**
23                  * O construtor eh resposavel por identificar se alguma funcao foi escolhida
24                  * pelo usuario e chamar o metodo especifico
25                  * A variavel 'op' possui o nome da funcao escolhida pelo usuario
26                  */
27                  if(!$_REQUEST[op])
28                          $metodo = 'index'; //funcao padrao. deve estar definida na subclasse
29                  else
30                          $metodo = $_REQUEST[op];
31                  $classe =  get_class($this); //retorna o nome da classe atual. mesmo se for uma
32  subclasse
33                  $ar = array($classe,$metodo);
34                  call_user_func($ar); //executa o metodo chamado
35
36          }
37          /**
38          * funcao que mostra a camada de visao
39          * @param string $view O nome do arquivo.php que eh a visao
40          * @param string[] $dados Os dados a serem trocados na visao
41          * @return void
42          */
43          function showView($view, $dados="") {
44                  if($dados)
45                          extract($dados); //transforma cada um dos indices do vetor de dados em
46  variaveis
47                  include($view);
48          }
49
```

```
50  }
51
```

Outra classe desenvolvida é a classe tabela. Esta classe é responsável por abstrair e facilitar a manipulação de tabelas de bancos de dados. Seu código fonte é descrito abaixo.

```
1   <?
2   /**
3     * Classe generica para trabalhar com tabelas
4     * Elton Luis Minetto <eminetto at gmail dot com>
5     * Licenca: GPL
6     */
7
8   include("adodb/adodb.inc.php"); //a classe depende do adodb
9   include("adodb/adodb-exceptions.inc.php");
10
11  class tabela {
12          /**
13          * nome da tabela
14          * @var string
15          */
16          protected $tabela;
17
18          /**
19          * conexao com a base de dados
20          * @var string
21          */
22          protected $db;
23
24
25          /**
26          * array com os dados usados para resultado
27          * @var string[]
28          */
29          public $dados_result;
30
31          /**
32          * array com os dados usados para insert e update
33          * @var string[]
34          */
35          public $dados_dml;
36
37          /**
38          * array usado pelo adodb para pegar os resultados das consultas
39          * @var string
40          */
```

```
41          public $result;
42
43          /**
44          * Construtor da classe
45          * @param string $tabela O nome da tabela
46          * @return void
47          */
48          public function __construct($tabela) {
49                  $this->tabela = $tabela;
50                  try {
51                          $this->db = NewADOConnection(app::$db_string);
52                  } catch (Exception $e) {
53                          echo "Erro na conexao:".$e->getMessage();
54                  }
55                  $this->dados_result = array();
56                  $this->dados_dml = array();
57          }
58          /**
59          * Funcao que altera o valor da propriedade tabela
60          * @param string $tabela Nome da tabela
61          * @return void
62          */
63          public function setTabela($tabela) {
64                  $this->tabela = $tabela;
65          }
66
67          /**
68          * Funcao que monta a consulta sql para a busca dos dados
69          * @param string[] $campos Array com o nome dos campos a serem buscados
70          * @param string $where Parametros SQL para a pesquisa
71          * @return void
72          */
73          public function get($campos,$where=null) {
74                  //monta o sql
75                  $sql = "select ";
76                  $sql .= implode(",",$campos);
77                  $sql .= " from ".$this->tabela;
78                  if($where) {
79                          $sql .= " where ".$where;
80                  }
81                  try {
82                          $this->result = $this->db->Execute($sql);
83                  } catch (Exception $e) {
84                          echo "Erro na pesquisa:<br>Erro:".$e-
    >getMessage().'<br>SQL:'.$sql.'<br><a href="javascript:history.go(-1)">Voltar</a>';
85                  }
86          }
87
```

```
88          /**
89          * Funcao que retorna um valor booleano indicando se ainda existem resultados
90          * @return bool
91          */
92          public function result() {
93                  try {
94                          if($this->dados_result = @array_change_key_case($this->result-
    >FetchRow(), CASE_LOWER)){ //recebe o array resultante e converte as chaves para minusculo
95                                  return true;
96                          }
97                          else {
98                                  return false;
99                          }
100                 } catch (Exception $e){
101                         echo $e->getMessage();
102                 }
103         }
104
105         /**
106         * Funcao que faz o insert dos dados na tabela
107         * @return void
108         */
109         public function insert() {
110                 $this->db->BeginTrans( );
111                 $sql = "insert into ".$this->tabela."(";
112                 $sql .= implode(",",array_keys($this->dados_dml));
113                 $sql .= ") values (";
114                 $sql .= implode(",",$this->dados_dml);
115                 $sql .= ')';
116                 try {
117                         $this->result = $this->db->Execute($sql);
118                         $this->dados_dml = array();
119                 } catch (Exception $e) {
120                         echo 'Erro na insercao:'.$e->getMessage().'<br><a
    href="javascript:history.go(-1)">Voltar</a>';
121                         exit;
122                 }
123
124         }
125         /**
126         * Funcao que faz o update dos dados na tabela
127         * @param string $where Parametros SQL para a alteracao
128         * @return void
129         */
130         public function update($where) {
131                 $this->db->BeginTrans( );
132                 $sql = "update ".$this->tabela." set ";
133                 foreach($this->dados_dml  as $campo => $valor) {
134
```

```
135                     $sql .= "$campo = $valor,";
136             }
137             $sql = substr($sql,0,strlen($sql)-1);//remove a ultima virgula
138             $where = stripslashes($where);
139             $sql .= " where $where";
140             try {
141                     $this->result = $this->db->Execute($sql);
142                     unset($this->dados_dml);
143             } catch (Exception $e) {
144                     echo "Erro na atualiza  o:<br>SQL:".$sql.'<br>Erro:'.$e-
>getMessage().'<br><a href="javascript:history.go(-1)">Voltar</a>';
145                     exit;
146             }
147     }
148
149     /**
150     * Funcao que faz a exclusao dos dados na tabela
151     * @param string $where Parametros SQL para a exclusao
152     * @return void
153     */
154     public function delete($where=null) {
155             $this->db->BeginTrans( );
156             $sql = "delete from ".$this->tabela;
157             if($where)
158                     $sql .= " where ".stripslashes($where);
159             //echo $sql;
160             try {
161                     $this->result = $this->db->Execute($sql);
162             } catch (Exception $e) {
163                     echo "Erro na exclusão:".$e->getMessage().'<br><a
href="javascript:history.go(-1)">Voltar</a>';
164                     exit;
165             }
166     }
167
168     /**
169      * Interceptador __set. Quando um valor eh alterado ele eh colocano no array de dados
170      * para ser usado em instrucoes DML (insert, update)
171      */
172     function __set($name,$value) {
173             $this->dados_dml[$name] = "'".$value."'";
174     }
175
176     /**
177      * Inserceptador __get. Quando um valor eh solicitado eh entregue o valor
178      * do array de resultados das consultas
179      */
180     function __get($name) {
181
```

```
182                 $name = strtolower($name);
183                 if($name != "dados_result")
184                         return $this->dados_result[$name];
185                 else
186                         return $this->dados_result;
187         }
188
189         /**
190         * Funcao que faz a confirmacao das operacoes
191         * @return void
192         */
193         public function save() {
194                 $this->db->CommitTrans( );
195         }
196
197         /**
198         * Destrutor da classe
199         * @return void
200         */
201         public function __destruct() {
202                 $this->db->close();
203         }
204 }?>
```

Exemplo de aplicação

Para ilustrar o funcionamento eu criei uma pequena aplicação com as classes. A aplicação é um sistema de blog, com posts e comentários.

A estrutura de diretórios ficou assim:

```
classes/ - diretório com as classes
        classes/app.php – classe app
        classes/tabela.php – classe tabela para tratamento de tabelas no banco de dados
        classes/adodb -  classes adodb para abstração de bancos de dados. necessário para a classe
tabela.
        classes/JSON.php – para utilizar JSON, usado por algumas páginas que usam AJAX
blog/ – diretório da aplicação
        blog/index.php – subclasse da classe app
        blog/view/ - diretório com as visões
                blog/view/index_view.php – visão inicial
                blog/view/login_view.php – visão da página de login
                blog/view/comentario_view.php – visão dos comentários
                blog/view/admin_view.php – visão da página de administração
                blog/view/estilo.css – arquivo com as definições de CSS para as visões
        blog.sql – arquivo sql com os comandos para criar as tabelas da aplicação
```

A primeira tarefa é criar a base de dados e as tabelas que serão utilizadas no exemplo. Para isso foram executados os seguintes comandos sql (gravados no arquivo blog.sql):

```
1  create database blog;
2  use blog;
3  create table post (id_post int primary key auto_increment, tit_post varchar(255), ds_post text,
   dt_post date);
4
5  create table comentario(id_com int primary key auto_increment,ds_com text, email_com
   varchar(100), id_post int);
```

A base de dados usada neste exemplo é o MySQL.

O código do arquivo index.php do diretório blog deve ser uma subclasse da classe app. O código inicial ficou desta forma:

```
1  <?
2  session_start();
3  include("../classes/app.php"); //faz a inclusão das classes
4  include("../classes/tabela.php");
5  class blog extends app { //cria uma subclasse da classe blog
6
7  }
```

O primeiro método a ser escrito é o método index(). Este método é o método inicial da aplicação. O construtor da classe app sempre vai invocar este método caso não tenha sido escolhida outra opção. Complementando o código:

```
1  <?
2  session_start();
3  include("../classes/app.php"); //faz a inclusão das classes
4  include("../classes/tabela.php");
5
6  class blog extends app { //cria uma subclasse da classe blog
7          * Mostra a pagina inicial*/
8          function index(){
9                  $post = new tabela("post"); //cria uma nova instância da classe tabela
10                 $com = new tabela("comentario"); //conexão com a tabela comentario
11                 $post->get(array("*")); //busca todos (*) os campos da tabela post
12                 $i=0;
13                 while($post->result()) { //enquanto possui resultados
14                         //alimenta o vetor dados que será enviado para a visão
15                         $dados[tit_post][$i] = $post->tit_post;
16                         $dados[dt_post][$i] = $post->dt_post;
17                         $dados[ds_post][$i] = nl2br($post->ds_post);
18                         $dados[id_post][$i] = $post->id_post;
19                         //busca o total de comentários do post
```

```
20                      $com->get(array("count(*) com"),"id_post=$post->id_post");
21                      $com->result();
22                      $dados[num_com][$i] = $com->com;
23                      $i++;
24              }
25              //invoca o método que constroi a visão
26              app::showView("view/index_view.php",$dados); //chama o template
27      }
28  }
```

O código do arquivo view/index_view.php é o seguinte:

```
1   <html>
2   <head>
3   <title>Blog</title>
4   <link href="view/estilo.css" rel="stylesheet" type="text/css"></head>
5   <body>
6   <h2>Blog</h2>
7   <a href="index.php?op=mostraLogin">Admin</a>
8   <br><br>
9   <?
10  for($i=0;$i<count($tit_post);$i++){
11  ?>
12  <div class="titulo"><?=$tit_post[$i]?></div>
13  <div class="data">Postado em <?=$dt_post[$i]?></div><br>
14  <div class="corpo"><?=$ds_post[$i]?></div><br>
15  <div class="comentario"><?=$num_com[$i]?> comentario(s)
16  <a href="index.php?op=comentarios&id_post=<?=$id_post[$i]?>">Adicionar comentário</a></div><br>
17  <?
18  }
19  ?>
20  </body>
21  </html>
```

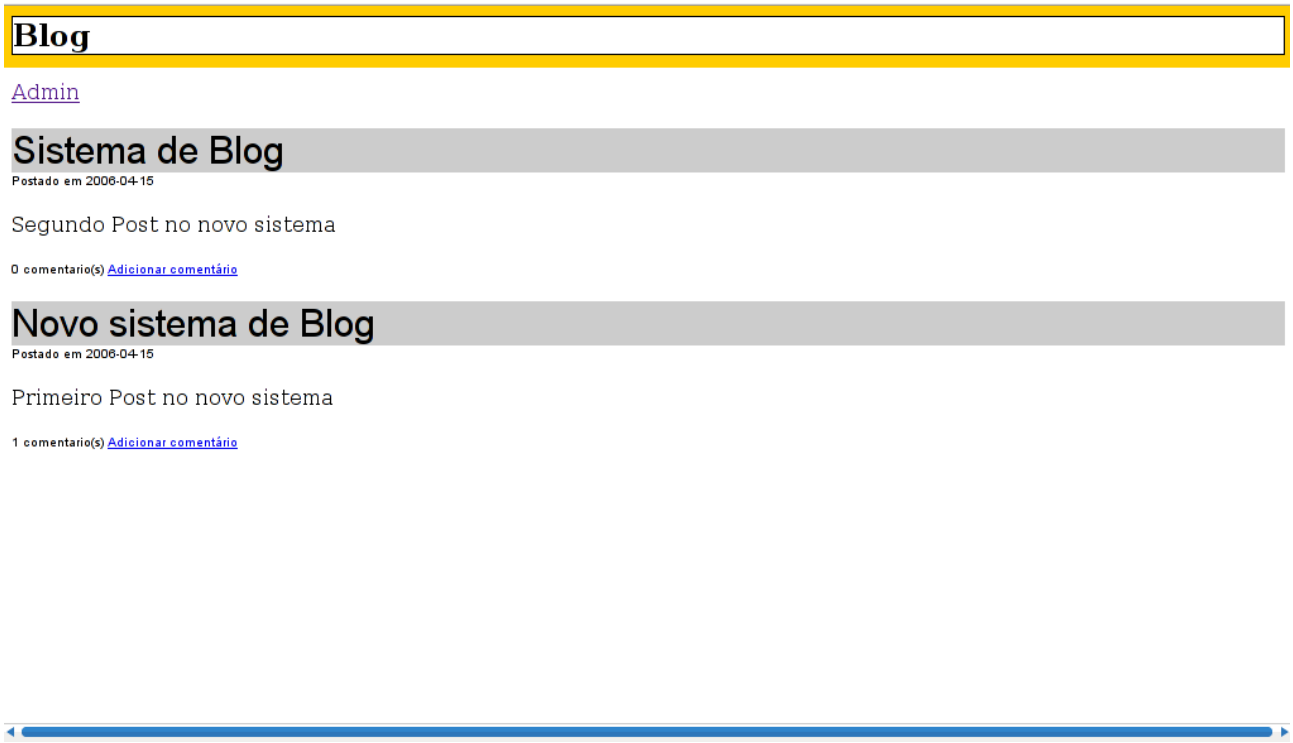
O método showView da classe app vai transformar cada índice do vetor $dados em uma variável ou em um novo vetor. Então o script index_view.php vai simplesmente imprimir seus valores.

Uma nova linha deve ser adicionada no final do arquivo index.php (após a linha 28):

```
$blog = new blog("mysql://root:@localhost/blog");
```

Nesta linha é instanciado um novo objeto da classe blog criada. Como parâmetro para o construtor da classe é enviado a string de conexão com a base dados. Esta string é no formato usado pelo ADODB e a sintaxe para diversos bancos de dados podem ser encontradas no site da ferramenta.

Executando-se a aplicação deve-se obter o seguinte resultado:



Todas as definições de cores, fontes e estilos foram adicionadas no arquivo estilo.css utilizando-se as técnicas de CSS. Desta forma, os arquivos de visão não possuem formatações de estilo e sim somente informações dos dados que devem ser gerados. A parte de formatação fica em separado, o que facilitaria caso fosse necessário alterar as definições de layout da aplicação. O código do arquivo estilo.css é o seguinte:

```
1   h2 {
2           border-style:solid;
3           border-color:#000000;
4           border-width:1px;
5           outline-width:10px;
6           outline-style:solid;
7           outline-color:#FFCC00;
8   }
9   .input {
10          border-style:solid;
11          border-color:#FFCC00;
12          border-width:1px;
13  }
14
15  .titulo {
16          background-color: #CCCCCC     ;
17          font-family:Arial;
18          font-size:30px;
19  }
```

```
20  .data {
21          font-family:Arial;
22          font-size:10px;
23  }
24  .categoria {
25          font-family:Arial;
26          font-size:10px;
27  }
28
29  .comentario {
30          font-family:Arial;
31          font-size:10px;
32  }
33  #login {
34          position:absolute;
35          left:           400px;
36          top:            100px;
37          width:          500px;
38          height:         80px;
39          padding:        10px;
40          background-color: #CCCCCC;
41  }
```

O próximo passo é criar as outras ações da aplicação. Por exemplo, quando o usuário clicar no link comentários ele será direcionado para:

http://localhost/blog/index.php?op=comentarios&id_post=15

Como a variável **op** controla a ação que o usuário escolheu precisamos definir um novo método na classe blog para atender a esta requisição. Então o seguinte código deve ser adicionado aos métodos da classe blog (arquivo index.php, antes da linha 28):

```
1   /* comentarios e formulario para novo comentario*/
2   function comentarios($id_post) {
3           if(!$id_post) //se não existe valor na variavel usa a enviada por GET
4                   $id_post = $_GET[id_post];
5           $post = new tabela("post");//conexao com a tabela post
6           $post->get(array("*"),"id_post=$id_post"); //busca os dados do post solicitado
7           $post->result();
8           $dados[tit_post] = $post->tit_post; //alimenta o vetor de dados para a visão
9           $dados[ds_post] = $post->ds_post;
10          $dados[dt_post] = $post->dt_post;
11          $dados[id_post] = $post->id_post;
12          $comentario = new tabela("comentario"); //conexao com a tabela comentario
13          $comentario->get(array("*"),"id_post=$id_post");
14          $i=0;
15          while($comentario->result()) { //mostra todos os comentarios do post
16                  $dados[ds_com][$i] = $comentario->ds_com;
```

```
17                  $dados[email_com][$i] = $comentario->email_com;
18                  $i++;
19          }
20          app::showView("view/comentario_view.php",$dados);
21  }
```

O método comentarios() faz uso da visão comentario_view.php. Seu código é:

```
1   <html>
2   <head>
3   <title>Blog</title>
4   <link href="view/estilo.css" rel="stylesheet" type="text/css"></head>
5   <body>
6   <h2>Blog</h2>
7
8   <div class="titulo"><?=$tit_post?></div><br/>
9   <div class="data">Postado em <?=$dt_post?></div><br />
10  <div class="corpo"><?=$ds_post?></div><br />
11
12  <h3>Comentários</h3>
13  <?
14  for($i=0;$i<count($ds_com);$i++) {
15  ?>
16  <?=$ds_com[$i]?>
17  <div class="data">Comentado por <?=$email_com[$i]?></div><br />
18  <?
19
20  }
21  ?>
22
23  <h3>Adicionar Comentário</h3>
24
25  <form method="post" action="index.php" name="comentario">
26          <input type="hidden" value="addComentario" name="op">
27          <input type="hidden" value="<?=$id_post?>" name="id_post">
28          <label>E-Mail</label>
29          <input type="text" name="email_com"><br>
30          <label>Comentario</label>
31          <textarea rows="10" cols="40" name="ds_com"></textarea>
32          <br>
33          <input type="submit" value="Enviar"><br>
34  </form>
35  <br>
36  <br>
37  <a href="index.php">Voltar</a><br>
38  </body>
```

```
39  </html>
```

Assim, quando o usuário clicar no link “Adicionar comentário” na página inicial serão apresentados os comentários existentes e um formulário para adição de um novo comentário, conforme a imagem abaixo ilustra:

Quando o usuário submeter os dados do novo comentário a ação “addComentario” será executada, como indicado pelo input hidden chamado op na linha 26 do comentario_view.php. Para que esta ação seja executada o seguinte código deve ser adicionado no arquivo index.php (novamente após a linha 27).

```
1   /* faz a inclusao do comentario na base*/
2   function addComentario(){
3           $com = new tabela("comentario"); //conecta com a tabela comentario
4           $com->id_post = $_POST[id_post]; //altera o valor de acordo com os enviados pelo form
5           $com->ds_com = $_POST[ds_com];
6           $com->email_com = $_POST[email_com];
7           $com->insert(); //insere
8           $com->save(); //faz o commit
9           blog::comentarios($_POST[id_post]); //redireciona para o metodo comentarios
10  }
```

Quando o usuário clica no link Admin da página inicial ele é redirecionado para a visão que mostra

o formulário de login. O método da classe blog que realiza esta ação é:

```
1   /* mostra o formulario de login*/
2   function mostraLogin() {
3           session_start(); //inicia a sessão
4           if(!$_SESSION[logado]) { //se ainda não foi feito a validação
5                   app::showView("view/login_view.php"); //mostra a visão
6           }
7           else {
8                   blog::login(); //senão chama o metodo login()
9           }
10  }
```

E o código fonte do arquivo login_view.php pode ser visualizado abaixo:

```
1   <html>
2   <head><title>Blog Admin</title></head>
3   <link href="view/estilo.css" rel="stylesheet" type="text/css" />
4   <body>
5   <div id="login">
6   <form name="login" action="index.php" method="post">
7           <input type="hidden" name="op" value="login">
8           Login:<input class="input" type="text" name="username"><br>
9           Senha:<input type="password" name="senha"><br>
10          <input type="submit" value="Entrar">
11  </form>
12  </div>
13  </body>
14  </html>
```

O método login da classe blog é responsável pela validação do usuário e por mostrar a visão de administração. Neste exemplo não é feito nenhuma validação específica, isso é deixado a cargo do leitor, podendo implementar algum método de autenticação que seja pertinente.

```
1   /* faz as validacoes de login e mostra os posts cadastrados*/
2   function login() {
3           session_start();
4           if(!$_SESSION[logado]) {
5                   //fazer as validações aqui
6                   $_SESSION[username] = $_POST[username];
7                   $_SESSION[logado] = 1;
8           }
9
10          $post = new tabela("post");
```

```
11          $post->get(array("*"));//busca todos os posts da tabela para mostrar na administração
12          $i=0;
13          while($post->result()) {
14                  $dados[tit_post][$i] = $post->tit_post;
15                  $dados[id_post][$i] = $post->id_post;
16                  $i++;
17          }
18          app::showView("view/admin_view.php",$dados);
19  }
```

A visão de administração é a mais complexa de todas. Além de mostrar os posts já cadastrados na tabela ela fornece opções de exclusão e de alteração dos mesmos. Para melhorar a interação com o usuário é usado técnicas de AJAX para buscar os dados do post antes do usuário realizar a alteração. O código do arquivo admin_view.php é mostrado abaixo, com seus comentários.

```
1   <html>
2   <head><title>Blog Admin</title></head>
3   <link href="view/estilo.css" rel="stylesheet" type="text/css" />
4   <script>
5           function createRequestObject() {
6             var ro;
7             var browser = navigator.appName;
8             if(browser == "Microsoft Internet Explorer"){
9                ro = new ActiveXObject("Microsoft.XMLHTTP");
10            }else{
11               ro = new XMLHttpRequest();
12            }
13            return ro;
14          }
15          var http = createRequestObject(); //cria um objeto XMLHttpRequest
16
17          function busca(id_post){ //recebe o id do post a buscar
18                  //abre uma conexão assincrona com o servidor solicitando os dados do post
19                  http.open('get', 'index.php?op=buscaPost&id_post='+id_post);
20                  //nome da função JavaScript que trata o resultado da conexão
21                  http.onreadystatechange = handleResponse;
22                  http.send(null);
23          }
24          //quando os dados são recebidos esta função é invocada
25          function handleResponse() {
26                  if(http.readyState == 4){ //situação 4 é quando a conexão finalizou
27                                          mostraForm();
28                                          var response = http.responseText; //resultado formatado
29                                          eval("var arr = "+response); //eval avalia uma expressao e
    executa. vai criar um objeto chamado arr com o resultado
30
31                                          document.getElementById('tit_post').value = arr.tit_post;
    //cada indice do vetor do php vira uma propriedade do objeto
32                                          document.getElementById('ds_post').value = arr.ds_post;
```

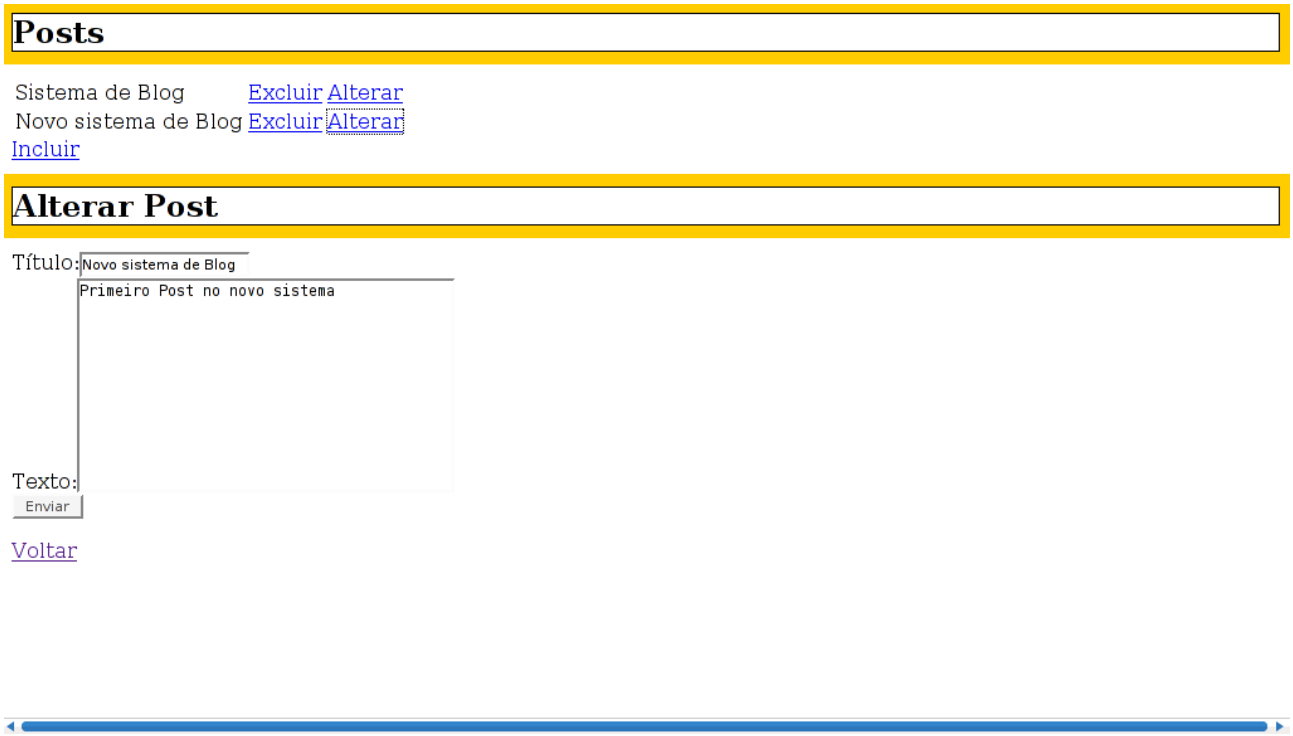
```
                                        document.getElementById('id_post').value = arr.id_post;
                                        document.getElementById('op').value = "altPost";
                                        document.getElementById('titulo').innerHTML = "Alterar
Post";
                                }
                }
                //função que mostra o formulário oculto via CSS
                function mostraForm() {
                        document.getElementById('form').style.visibility = "visible";
                        document.getElementById('form').style.position = "relative";
                        document.getElementById('op').value = "addPost";
                        document.getElementById('titulo').innerHTML = "Adicionar Post";
                }

</script>


<body>
<h2>Posts</h2>
<table>
<?
for($i=0;$i<count($tit_post);$i++) {
?>
        <tr>
                <td><?=$tit_post[$i]?></td>
                <td><a href="index.php?op=del&id_post=<?=$id_post[$i]?>">Excluir</a></td>
                <td><a href="javascript:busca(<?=$id_post[$i]?>)">Alterar</a></td>
        </tr>
<?
}
?>
</table>
<a href="javascript:mostraForm()">Incluir</a><br>
<div id="form">
<h2><div id="titulo">Adicionar Post</div></h2>
<form name="post" action="index.php" method="post">
        <input type="hidden" name="op" value="addPost" id="op">
        <input type="hidden" name="id_post" value="" id="id_post">
        Título:<input type="text" name="tit_post" id="tit_post"><br>
        Texto:<textarea name="ds_post" cols="40" rows="10" id="ds_post"></textarea><br>
        <input type="submit" value="Enviar">
</form>
</div>
<a href="index.php">Voltar</a>
</body>
</html>
```

Quando o usuário clica no link "Alterar" uma conexão assíncrona é aberta com o servidor via AJAX solicitando os dados do post. Os dados são codificados no formato JSON. JSON é "um formato leve para troca de informações. É facil para humanos lerem e escreverem. E é fácil para as máquinas processar e gerar." Uma espécie de XML ligth. É baseado na notação de objetos do JavaScript, o que cai como uma luva para usar com o XMLHttpRequest. A classe blog utiliza o include JSON.php para gerar os dados neste formato. Este script pode ser encontrado no repositório PEAR, no endereço http://pear.php.net/pepr/pepr-proposal-show.php?id=198 e mais informações sobre o JSON podem ser encontrados no http://www.json.org.

Um exemplo da visão de administração é mostrado na figura abaixo:



Abaixo é mostrado o código final da classe blog com todos os métodos comentados acima e os métodos restantes, addPost(), del(), altPost() e buscaPost().

```
1   <?
2   session_start();
3   include("../classes/app.php"); //faz a inclusão das classes
4   include("../classes/tabela.php");
5
6   class blog extends app { //cria uma subclasse da classe blog
7           * Mostra a pagina inicial*/
8           function index(){
9                   $post = new tabela("post"); //cria uma nova instância da classe tabela
10                  $com = new tabela("comentario"); //conexão com a tabela comentario
11                  $post->get(array("*")); //busca todos (*) os campos da tabela post
12                  $i=0;
13                  while($post->result()) { //enquanto possui resultados
```

```
14                      //alimenta o vetor dados que será enviado para a visão
15                      $dados[tit_post][$i] = $post->tit_post;
16                      $dados[dt_post][$i] = $post->dt_post;
17                      $dados[ds_post][$i] = nl2br($post->ds_post);
18                      $dados[id_post][$i] = $post->id_post;
19                      //busca o total de comentários do post
20                      $com->get(array("count(*) com"),"id_post=$post->id_post");
21                      $com->result();
22                      $dados[num_com][$i] = $com->com;
23                      $i++;
24              }
25              //invoca o método que constroi a visão
26              app::showView("view/index_view.php",$dados); //chama o template
27      }
28      /* comentarios e formulario para novo comentario*/
29      function comentarios($id_post) {
30              if(!$id_post)
31                      $id_post = $_GET[id_post];
32              $post = new tabela("post");
33              $post->get(array("*"),"id_post=$id_post");
34              $post->result();
35              $dados[tit_post] = $post->tit_post;
36              $dados[ds_post] = $post->ds_post;
37              $dados[dt_post] = $post->dt_post;
38              $dados[id_post] = $post->id_post;
39              $comentario = new tabela("comentario");
40              $comentario->get(array("*"),"id_post=$id_post");
41              $i=0;
42              while($comentario->result()) {
43                      $dados[ds_com][$i] = $comentario->ds_com;
44                      $dados[email_com][$i] = $comentario->email_com;
45                      $i++;
46              }
47              app::showView("view/comentario_view.php",$dados);
48      }
49
50      /* faz a inclusao do comentario na base*/
51      function addComentario(){
52              $com = new tabela("comentario");
53              $com->id_post = $_POST[id_post];
54              $com->ds_com = $_POST[ds_com];
55              $com->email_com = $_POST[email_com];
56              $com->insert();
57              $com->save();
58              blog::comentarios($_POST[id_post]);
59      }
60
```

```
61          /* mostra o formulario de login*/
62          function mostraLogin() {
63                  session_start();
64                  if(!$_SESSION[logado]) {
65                          app::showView("view/login_view.php");
66                  }
67                  else {
68                          blog::login();
69                  }
70          }
71
72          /* faz as validacoes de login e mostra os posts cadastrados*/
73          function login() {
74                  session_start();
75                  if(!$_SESSION[logado]) {
76                          //fazer as validaÃ§Ãµes
77                          $_SESSION[username] = $_POST[username];
78                          $_SESSION[logado] = 1;
79                  }
80
81                  $post = new tabela("post");
82                  $post->get(array("*"));
83                  $i=0;
84                  while($post->result()) {
85                          $dados[tit_post][$i] = $post->tit_post;
86                          $dados[id_post][$i] = $post->id_post;
87                          $i++;
88                  }
89                  app::showView("view/admin_view.php",$dados);
90          }
91
92          /* faz a inclusao do post*/
93          function addPost() {
94                  $post = new tabela("post");
95                  $post->tit_post = $_POST[tit_post];
96                  $post->ds_post = $_POST[ds_post];
97                  $post->dt_post = date('Y-m-d H:i:s');
98                  $post->insert();
99                  $post->save();
100                 blog::login();
101         }
102
103         /* faz a exclusao do post*/
104         function del() {
105                 $post = new tabela("post");
106                 $post->delete("id_post=$_GET[id_post]");
107                 $post->save();
108                 blog::login();
```

```
109         }
110 
111         /* faz a alteracao do post*/
112         function altPost() {
113                 $post = new tabela("post");
114                 $post->tit_post = $_POST[tit_post];
115                 $post->ds_post = $_POST[ds_post];
116                 $post->update("id_post=$_POST[id_post]");
117                 $post->save();
118                 blog::login();
119         }
120         /* Esta função é invocada pelo AJAX*/
121         function buscaPost() {
122                 include("../classes/JSON.php");
123                 $post = new tabela("post");
124                 $post->get(array("*"),"id_post=$_GET[id_post]");
125                 $post->result();
126                 //todos os dados do post são armazenados em um array
127                 $arr[id_post] = $post->id_post;
128                 $arr[ds_post] = $post->ds_post;
129                 $arr[tit_post] = $post->tit_post;
130                 //um novo objeto do tipo Services_JSON é criado
131                 $json = new Services_JSON();
132                 //os dados são retornados para o AJAX codificados no formato JSON
133                 echo $json->encode($arr);
134         }
135 }
136 
137 $blog = new blog("mysql://root:@localhost/blog");
    ?>
```